REASONING ALGEBRAÏCALLY WITH DESCRIPTION LOGICS

Jocelyne Faddoul

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

Concordia University

Montréal, Québec, Canada

September 2011

CONCORDIA UNIVERSITY
School of Graduate Studies

This is to certify that the thesis prepared

By: **Jocelyne Faddoul**
Entitled: **Reasoning Algebraïcally with Description Logics**

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 commitee:

_____ Chair
Dr. Mingyuan Chen

_____ External Examiner
Dr. Grant Weddell

_____ Examiner
Dr. Hovhannes Harutyunyan

_____ Examiner
Dr. Leila Kosseim

_____ Examiner
Dr. Otmane Ait Mohamed

_____ Supervisor
Dr. Volker Haarslev

Approved _____
                Chair of Department or Graduate Program Director

_____ 20 _____ _____
                                Dr. Robin A.L. Drew, Dean
                                Faculty of Engineering and Computer Science

Abstract

Reasoning Algebraïcally with Description Logics

Jocelyne Faddoul, Ph.D.

Concordia University, 2011

Semantic Web applications based on the Web Ontology Language (OWL) often require the use of numbers in class descriptions for expressing cardinality restrictions on properties or even classes. Some of these cardinalities are specified explicitly, but quite a few are entailed and need to be discovered by reasoning procedures. Due to the Description Logic (DL) foundation of OWL, those reasoning services are offered by DL reasoners. Existing DL reasoners employ reasoning procedures that are arithmetically uninformed and substitute arithmetic reasoning by "don't know" non-determinism in order to cover all possible cases. This lack of information about arithmetic problems dramatically degrades the performance of DL reasoners in many cases, especially with ontologies relying on the use of Nominals and Qualified Cardinality Restrictions.

The contribution of this thesis is twofold: on the theoretical level, it presents algebraïc reasoning with DL (ReAl DL) using a sound, complete, and terminating reasoning procedure for the DL $\mathcal{SHOQ}$. ReAl DL combines tableau reasoning procedures with algebraïc methods, namely Integer Programming, to ensure arithmetically

better informed reasoning. $\mathcal{SHOQ}$ extends the standard DL $\mathcal{ALC}$ with *transitive roles*, *role hierarchies*, *qualified cardinality restrictions* (QCRs), and *nominals*, and forms an expressive subset of OWL. Although the proposed algebraïc tableau is double exponential in the worst case, it deals with cardinalities with an additional level of information and properties that make the calculus amenable and well suited for optimizations. In order for ReAl DL to have a practical merit, suited optimizations are proposed towards achieving an efficient reasoning approach that addresses the sources of complexity related to *nominals* and QCRs.

On the practical level, a running prototype reasoner (HARD) is implemented based on the proposed calculus and optimizations. HARD is used to evaluate the practical merit of ReAl DL, as well as the effectiveness of the proposed optimizations. Experimental results based on real world and synthetic ontologies show that ReAl DL outperforms existing reasoning approaches in handling the interactions between nominals and QCRs. ReAl DL also comes with some interesting features such as the ability to handle ontologies with cyclic descriptions without adopting special blocking strategies. ReAl DL can form a basis to provide more efficient reasoning support for ontologies using *nominals* or QCRs.

Acknowledgments

While completing this thesis, I often felt like I was being transported in a train, unsure when the next stop is going to be, nor when the destination is going to be reached. I am fortunate and blessed to have had many companions who made my ride fun, exciting, and nurturing.

I would like to thank my supervisor, Dr. Volker Haarslev, for his guidance, insightful discussions, encouragement, and constructive feedback. Dr. Haarslev's words since day one, "There is always something that we can do," provided me with a boost every time I felt discouraged or challenged. His confidence in my work allowed me to overcome several obstacles, including completing this thesis from a distance. Without his continuous support, the research presented in this thesis would not have been possible.

I am grateful to many people who have pushed me to pursue my PhD: my undergraduate instructor Dr. Bilal Barake, my work colleague Boutros Chalouhy, my spiritual mentor Fr. Jean Saab, and my brother Miled Faddoul.

It was a pleasure to share doctoral studies and lab space with people like Dr. Hsueh-leng Pai, Dr. Xi Deng, Dr. Yu Ding, Dr. Ahmed Alasoud, Dr. Malek Barho, Hiba Tabbara, Jinane El Hashem, Zeina Nasr, Nasim Farsiniamarj, Francise Gasse, Jiewen Wu, and Dr. Tarek el Salty, who became my dearest friend and listened to my grumbling many, many times.

I thank Cresco management and staff for providing me with office space and

technical support while writing my thesis after moving to Halifax, Nova Scotia.

I wish to thank my family and extended family for providing a loving environment for me, especially my mother Odette, my brother Moussa, my sister Jihane, and my mother-in-law Hanne, who were particularly supportive. My heartfelt gratitude and thanks go out to my brother Miled; the support he gave me goes beyond any tangible and intangible support which anyone can give.

I would like to remember my father Youssef on the 10th anniversary of his passing. He knew my education would bring me to Canada long before I did; this was his premonition 10 years ago while fighting his battle with cancer. It is sad and unfortunate he lost his battle and could not be with me at this time.

My deepest appreciation goes to my husband Joseph, for without his unconditional love and support, this thesis would not have been completed. This work is as much his as it is mine.

My final thanks go to my son Jonas, who's birth has provided me with a gentle diversion from the rigours of PhD life. This thesis is dedicated to him.

"Being an intellectual creates a lot of questions and no answers."

Janis Joplin

"I love fishing.

You put that line in the water and you don't know what's on the other end.

Your imagination is under there."

Robert Altman

"The true sign of intelligence is not knowledge but imagination."

Albert Einstein

Contents

List of Figures

xviii

List of Tables

List of Algorithms

Chapter 1

Introduction

One of the areas of Artificial Intelligence studies the simulation of reasoning to achieve machine intelligence. Simulating a human reasoning process requires that an extensive knowledge about the world be represented and stored in a knowledge base. Among the things that need to be represented are: objects, properties, and relations between objects. A complete representation of "what exists" in a given domain forms an ontology. Ontologies have now been adopted by many disciplines, such as biology and e-science [WBH+05], as part of their integration into the Semantic Web [BLHL01], which is defined as a "web of data" allowing machines to understand the meaning of and process the information on the World Wide Web.

Description logic (DL) [FB07a] is a family of first-order logic formalisms allowing the representation of knowledge in the form of "concepts" (class, unary predicate), "roles" (object property, binary predicate), and "individuals" (class instance). The ability of DL languages to define concepts and relationships between concepts in a systematic and formal manner, makes them ideal to capture the complex relationships and semantics that are often part of many domains (e.g., medical domain). DL is becoming very popular in Knowledge Representation and modelling as it provides the logical foundation for the Web Ontology Language (OWL) [MPSCG08], defined

by the World Wide Web Consortium (W3C) as a standard for representing semantic links and knowledge on the Semantic Web [BHS03].

Nominals (enumerated classes) [Sch94] play an important role in DL as they allow one to express the notion of uniqueness and identity; *nominals* must be interpreted as singleton sets. Many ontologies, for instance in the geo-spatial domain, use *nominals* as names for persons, countries, colours, etc. In fact, the WINE ontology[1] which was designed to describe the features of OWL-DL relies heavily on *nominals* to represent wine colour, body, and flavour. An example for the use of *nominals* would be the representation of a European Union member state concept as an enumeration of the 27 distinct member states of the European Union (EU) as follows: EU_MemberState $\equiv$ {Austria, . . . , UK} where Austria, etc. are all distinct *nominals*.

Qualified Cardinality Restrictions (QCRs) allow one to specify a lower or upper bound on the number of elements related via a certain role with additionally specifying qualities on the related elements. For example, the following concept representation of (Future_EU $\sqsubseteq$ $\geq$ 30MemberOf.EU_MemberSate) states a necessary condition that an instance of the Future_EU concept must have at least 30 member states (using the operator $\geq$, the role MemberOf, and the qualifying concept EU_MemberState). Qualified Cardinality Restrictions have a long history in DLs. Their recently availability in OWL was advocated in [RS05] for modelling ontologies [HB91] in medical domains such as human anatomy [GZB06] and bio-ontology [WBH$^+$05].

It is known that DLs offering *nominals* and QCRs enjoy additional expressive power. There exist no other way, using the DL $\mathcal{SHOQ}$, to close a concept or domain with a finite number of elements except using *nominals*. This means that *nominals* can also emulate concept cardinalities [BBH96] (as was shown in [Tob00]). For example, informally speaking, it turns out that the representation of the EU_MemberState

---

[1]http://www.w3.org/TR/owl-guide/wine.rdf

concept and the disjointness of all 27 *nominals* express an implicit cardinality restriction; the concept EU_MemberState has exactly 27 individuals. However, the representation of Future_EU states that an instance of Future_EU needs to be related to at least 30 different instances of EU_MemberState. The unsatisfiability of Future_EU might be surprising until one remembers that the cardinality of EU_MemberState is implicitly restricted to the 27 member states listed in its definition, so, there cannot exist at least 30 distinct EU member states. Now, it is trivial for humans to realize that the concept Future_EU must be unsatisfiable. However, for existing DL reasoning algorithms, the unsatisfiability of Future_EU is not so trivial.

1.1 Problem Statement

Most modern DL reasoners implement tableau-based decision procedures which usually need to be equipped with a set of optimization techniques [PCS06] because their naïve implementations fail to be practical. Such decision procedures typically check the consistency of an ontology by constructing a so-called pre-model for the ontology. Despite many optimization techniques [FB07b] studied and implemented so far, they do not provide a generic practical DL reasoner. It is easy to find ontologies where one reasoner performs very well while the other is hopelessly inefficient. This is not only due to the high computational complexity of tableau calculi and inference services, but also to the fact that these algorithms create pre-models in an often blind way. Major inefficiency sources can be due to:

- (i) The high degree of non-determinism introduced by (a) the use of General Concept Inclusion axioms (GCIs) or (b) when merging domain elements is necessary,

- (ii) The construction of large models, and

- (iii) The interaction between language constructors.

For instance, in the case of deciding the satisfiability of Future_EU, a standard tableau algorithm creates 30 distinct but anonymous instances of EU_MemberState and then non-deterministically tries to merge them with the 27 *nominals* enumerating the EU member states, until all possibilities are exhausted and the unsatisfiability of Future_EU is returned. The overall fact that 27 *nominals* can never be distributed over 30 distinct instances is lost. This blindness to numbers can result in severe performance degradation not only by affecting the size of the completion models but also by introducing a high degree of non-determinism. The problem gets even worse when a large number of *nominals* is used or when large numbers are used in descriptions such as in the following concept description:

$$\text{Person} \sqsubseteq \; \geq 230 \, \text{hasJoint.}(\text{Movable} \sqcup \text{Semi\_Movable})$$

An ontology containing the representation of EU_MemberState, Future_EU, and the disjointness declaration for the 27 *nominals* was modelled and tested with the OWL ontology editor Protégé 4.0[2], and none of the highly optimized DL reasoners FaCT++,[3] Pellet,[4] or HermiT[5] could decide the satisfiability of EU_MemberState, in this small OWL ontology, within 2 hours of CPU time (using a PC with an AMD 64*2 Dual Core Processor 5200, 2.70 GHZ and 3GB of RAM).

Existing DL reasoning approaches are known to be very weak in handling QCRs especially with large numbers. The numerical restrictions implied by *nominals* and their interaction with QCRs aggravate the complexity. Little progress has been made to handle the interaction between *nominals* and QCRs. In fact, the only efficient

---

[2]http://www.co-ode.org/downloads/protege-x/
[3]http://owl.man.ac.uk/factplusplus/, version 1.2
[4]http://pellet.owldl.org/, version 2.0
[5]http://www.hermit-reasoner.com/, version 1.1

way to handle QCRs is through algebraïc reasoning first reported for the DL $\mathcal{SHQ}$ in [HM01a, HTM01] and more recently in [FH10c]. Decision procedures for expressive DLs enabling both *nominals* and QCRs were published in [HS07] with very weak optimizations if any (no DL reasoner was able to classify the WINE[6] ontology until recent efforts [PCS06]). The optimization techniques for *nominals* proposed in [PCS06] do not address the interaction between *nominals* and QCRs. Recent efforts in [MH08] address inefficient reasoning due to the creation of large tableau models and the presence of *nominals*. Resolution-based reasoning procedures were proposed in [KM06] and were proven to be weak in dealing with large numbers in QCRs. Hypertableaux [MSH07] were recently studied to minimize non-determinism in DL reasoning with no special treatment for QCRs. To the best of our knowledge no arithmetically informed approaches have been reported for ontologies that rely on the use and interaction of both *nominals* and QCRs.

1.1.1   Thesis Objectives

The research presented in this thesis is focused on designing an alternative reasoning approach for DLs handling *nominals* and QCRs rather than optimizing existing ones, which are found to be arithmetically uninformed and inefficient. Before designing and implementing a DL system, one should be clear about the expected goals to be achieved against which the performance/practicality will be measured. The adopted reasoning approach is of a hybrid nature, it consists of a standard tableau-based reasoning algorithm for DL combined with algebraïc reasoning. The algebraïc methods used are inspired and based on the arithmetic reasoning about sets for formal languages first introduced in [OK99]. The main objectives of adopting such a reasoning approach can be summarized as follows:

---

[6]http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf

1. High expressivity of the DL language supported: Available language constructors and properties of their combination or use define the *language expressivity* (see Chapter 2). The work presented in this thesis focuses on providing reasoning support for DL languages that allows all elements in an application domain to be expressed using the available logical language constructs. Ideally, one would be interested in the full expressivity of OWL ($\mathcal{SROIQ}$ [HKS06]) as it is the expressivity supported by Semantic Web applications. This thesis considers fragments of this language, in particular, those that support *nominals* ($\mathcal{O}$) and *number restrictions* ($\mathcal{N}$ or $\mathcal{Q}$) reaching the expressivity of $\mathcal{SHOQ}$.

2. Strong arithmetic reasoning: The reasoning approach must ensure a reasoning support that is strong on the logical (handling boolean operators on concept descriptions) side as well as on the arithmetic side (handling numerical features implied by concept descriptions). Unlike in standard tableau algorithms, the cardinality restrictions implied by *nominals* and those expressed with QCRs are encoded into a system of linear in-equations. The solvability of such a system can be decided using standard Integer Linear Programming algorithms (such as Simplex [CLRS01]). In the case of Future_EU a corresponding system of linear in-equations would immediately be recognized as unsolvable.

3. Correctness: The reasoning procedure must ensure soundness and completeness. Soundness in the sense that every "yes" answer for an inference test is a valid answer. Completeness in the sense that every "no" answer for an inference test is a valid answer.

4. Termination and Efficiency: Implementing sound and complete decision procedures for very expressive DL languages is the ultimate goal of every DL system. However, if such decision procedures do not terminate or do not respond in

reasonable time, then the whole system is not useful. The DL system needs to be equipped with a suite of optimization techniques to ensure efficiency without breaking *correctness* or *termination.*

5. Usability: The reasoning approach need not only provide practical DL reasoning with realistic applications, but also allow existing DL systems to adopt a similar algebraïc reasoning component for better handling of numerical features of concepts. The usability of the approach in realistic application will be assessed by an empirical analysis.

1.2   Thesis Contributions

The work presented in this thesis is of interest to the DL community and should be of value to designers and implementors of DL systems, who will be able to incorporate or use the reasoning approach together with their implemented procedures. The main contributions can be identified while meeting the previously mentioned thesis objectives:

- A first contribution is the design of a decidable calculus for the DL $\mathcal{ALCQ}$ based on the hybrid algebraïc approach. This calculus is published in [FFHM08a] and forms the basis for the algebraïc calculus for the DL $\mathcal{SHQ}$ [Far08], and for handling the full expressivity of $\mathcal{SHOQ}$ [FH10a] which will be covered in Chapter 4. The main activities consists of:

  1. Defining a tableau for $\mathcal{ALCQ}$.

  2. Designing a hybrid reasoning procedure which relies on algebraïc reasoning to decide the satisfiability of QCRs.

  3. Devising proofs of soundness, completeness, and termination of the reasoning procedure.

- A second contribution is the design of a decidable calculus for the DL $\mathcal{SHOQ}$ extending the first contribution to handle a more expressive DL with general TBoxes. The calculus is presented in Chapter 4, it has been published in [FH10a, FHM09] and makes up the theoretical contribution of this thesis due to the following:

  1. Extending a tableau reasoning algorithm with an algebraïc component, for expressive DLs, while maintaining soundness, completeness, and termination.

     - Extending the tableau for $\mathcal{ALCQ}$ into a tableau for $\mathcal{SHOQ}$.
     - Designing a hybrid reasoning procedure which relies on algebraïc reasoning to decide the satisfiability of QCRs and *nominals*.
     - Devising proofs of soundness, completeness, and termination of the reasoning procedure.

  2. The encoding of *nominals* semantics into in-equations handled using algebraïc reasoning. This handling of the numerical restrictions imposed by the *nominals* constructor is novel.

  3. The handling of cycles within concept descriptions or due to GCIs and transitive roles is ensured through the re-use of individuals. This handling of cycles is novel.

- A third contribution relies in analyzing sources of inefficiency in the reasoning algorithm proposed, and investigating possible optimizations. Part of those optimizations are published in [FH10b], and all optimizations considered are discussed in Chapter 5.

- A fourth contribution relies in the design and implementation of a running prototype reasoner to show the practical merit of the hybrid approach equipped

with the proposed optimizations. The prototype reasoner's architecture is described in Chapter 6.

- A last contribution relies in the evaluation of the performance of the reasoning approach, as well as the effectiveness of the proposed optimizations. The empirical evaluation is described in Chapter 7, the preliminary results were published in [FH10b] and a publication of the full evaluation is in preparation [FH11].

1.3   Thesis Outline

The remainder of this thesis can be outlined as follows:

- Chapter 2 introduces the formal syntax and semantics of DL as well as the inference services that are tackled in this thesis.

- Chapter 3 provides a review of existing approaches in dealing with QCRs and *nominals* in DLs.

- Chapter 4 presents a formal description of the proposed calculus along with proofs of soundness, completeness, and termination.

- Chapter 5 discusses optimization techniques aiming at enhancing performance of an implementation of the proposed calculus.

- Chapter 6 presents the architecture of the prototype reasoner, HARD, implementing the optimizations discussed in Chapter 5.

- Chapter 7 presents an empirical evaluation comparing the performance of HARD against existing state-of-the-art reasoners and shows the effectiveness of the adopted optimizations.

- Finally, Chapter 8 concludes this thesis with a summary, open problems and future directions.

Chapter 2

Preliminaries

This chapter introduces background information relevant to the work presented in this thesis. Section 2.1 introduces the syntax, semantics and inference problems of Description Logics with a main focus on the DL $\mathcal{SHOQ}$, which extends the basic DL $\mathcal{ALC}$ with *transitive roles*, *role hierarchies*, *nominals*, and *qualified cardinality restrictions* (QCRs), and which is the main focus of this thesis. Section 2.2 presents tableau algorithms as the most widely used reasoning procedures adopted by most state-of-the-art DL reasoners. The complexity of DL reasoning and the need for optimizations in discussed in Section 2.2.3. Section 2.4 concludes this chapter.

2.1 Description Logics

Description Logic (DL) [FB07a] is a family of knowledge representation languages used to represent and reason about an application's domain elements. DLs stem from Semantic Networks [Qui67] and Frames [Min81]. They are distinguished by their terminological orientation, their well defined logic-based semantics, and their inference capabilities. A typical Knowledge Representation system based on DL provides facilities to set up Knowledge Bases (KBs) and to reason about their content.

A DL KB comprises two components, the TBox and the ABox. The TBox introduces the terminology (i.e., the vocabulary of the represented domain), while the ABox contains assertions about named domain elements in terms of this vocabulary. To define the terminology of a certain KB, DL relies on the notions of "concepts", "roles", and "individuals" combined using a set of operators (DL constructors) into structured and formally well defined descriptions.

Definition 2.1.1 (Concept)   A concept is used to denote a set of domain elements with common characteristics. For example, the concept Man can be used to refer to all persons that are male, and the concept Child can be used to refer to all offsprings of a person. Concepts with no common elements are referred as disjoint. For example, the concepts Man and Woman can be declared as disjoint.

Definition 2.1.2 (Role)   A role is used to denote a binary relationship between domain elements. For example, the role hasChild can be used to define a relationship between a Man and a Child.

Definition 2.1.3 (Individual)   An individual is used to name elements within the represented domain. An individual is usually referred to as an instance of a certain concept (e.g., the individual Joseph is an instance of the concept Man, Joseph : Man), or as related to another individual via a certain role (e.g., the individual Joseph is related to the individual Jonas via the hasChild role, ⟨Joseph, Jonas⟩ : hasChild).

Concepts, roles, and individuals represented using a DL language can be inter-related in such a way that implicit knowledge can be derived from explicitly represented knowledge. This can be done because DL systems not only store terminologies and assertions, but also offer services that reason about them. Typical reasoning

tasks for a terminology are to determine whether a description is satisfiable (i.e., non-contradictory), or whether one description is more general than another, that is, whether the first subsumes the second. Inference problems for an ABox are to find out whether its set of assertions is consistent (i.e., whether it has a model), and whether the assertions in the ABox entail that a certain individual is an instance of a given concept description. Satisfiability checks of descriptions and consistency checks of sets of assertions are useful to determine whether a KB is meaningful at all.

The language for building descriptions is a characteristic of each DL system, and different DL systems are distinguished by their description languages. The description language has model-theoretic semantics. Most common used DLs are considered as decidable fragments of first order predicate logic (FOL); individuals can be seen as constants, concepts as unary predicates, and roles are binary predicates. Therefore, for some DL languages, statements in the TBox and in the ABox can be identified with formulae in first-order logic ($\mathsf{Man} \longrightarrow \mathsf{Person} \wedge \mathsf{Male}$, $\mathsf{Man}(\mathsf{Joseph})$). The DL $\mathcal{ALC}$ is the basic DL language containing the smallest set of DL constructors ($\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$) closed under all boolean operations (conjunction, disjunction, and negation). The following section introduces the formal syntax, semantics and inference services of the DL $\mathcal{ALC}$ which is extended with *transitive roles* (leading to the DL $\mathcal{S}$), *role hierarchies* ($\mathcal{H}$), *nominals* ($\mathcal{O}$), and *qualified cardinality restrictions* ($\mathcal{Q}$) leading to the definition of the DL $\mathcal{SHOQ}$ which is the main family of DLs referenced throughout this thesis.

2.1.1 Basic DL $\mathcal{ALC}$ Syntax and Semantics

In [Sch91], it was shown that $\mathcal{ALC}$ is a syntactic variant of the modal logic K [BL06], where all roles are atomic and complex concepts can be built using boolean operators

($\sqcap$, $\sqcup$, $\neg$), universal restriction ($\forall$), and existential ($\exists$) value restriction on atomic concepts. Let N_C, N_R, and N_I be non-empty and pair-wise disjoint sets of concept names, role names, and individual names respectively. A is used to denote an atomic concept ($A \in N_C$), R is used to denote an atomic role ($R \in N_R$). $\mathcal{ALC}$-concept expressions are defined inductively using the syntax rule in (1), where C, D are $\mathcal{ALC}$-concepts, and $\top$ and $\bot$ are used to abbreviate $(C \sqcup \neg C)$ and $(C \sqcap \neg C)$, respectively.

$$\mathcal{ALC}\text{-concept} \longrightarrow \top \mid \bot \mid A \mid \neg A \mid (C \sqcup D) \mid (C \sqcap D) \mid (\exists R.C) \mid (\forall R.C) \qquad (1)$$

DLs differ from their predecessors in that they are equipped with a formal logic-based semantics. An interpretation $\mathcal{I}$ is defined to give formal semantics. An interpretation is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ where $\Delta^{\mathcal{I}}$ is a non-empty set, called the domain of the interpretation, and $\cdot^{\mathcal{I}}$ is the interpretation function. The interpretation function maps each atomic concept $A \in N_C$ to a subset of $\Delta^{\mathcal{I}}$, each atomic role $R \in N_R$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual $a \in N_I$ to an element of $\Delta^{\mathcal{I}}$. The interpretation function is extended to satisfy $\mathcal{ALC}$-concept expressions as follows:

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

$\mathcal{ALC}$-concepts can be used to describe classes of objects; for example, the concept description (Person $\sqcap$ Male) is used to describe "persons that are male". Using TBox axioms, one can make statements about relations between concepts and roles.

14

Definition 2.1.4 (Concept Inclusion Axiom) A Concept Inclusion Axiom is an expression of the form:

- $C \sqsubseteq D$ referred to as a *concept subsumption axiom*, or

- $C \equiv D$ referred to as a *concept definition axiom* which is an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$. A concept definition axiom is said to be a *primitive definition* if C is a concept name.

In the case where C is not a concept name, a concept inclusion axiom is referred to as a General Concept Inclusion axiom (GCI). Given an interpretation $\mathcal{I}$, the subsumption relation $(C \sqsubseteq D)$ between two concepts C, D holds if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds.

Definition 2.1.5 (TBox) A TBox $\mathcal{T}$ is a finite set of concept inclusion axioms; an example TBox is shown in Figure 1. $\mathcal{T}$ is said to be

- Unfoldable if all concept inclusion axioms consist of concept subsumption axioms of the form $A \sqsubseteq C$ and/or primitive definitions of the form $A \equiv C$, such that all axioms are unique and acyclic. For example, given the unfoldable TBox shown in Figure 1, the definition of Father can be unfolded by replacing Man and Child with their corresponding definitions into:

$$\text{Father} \sqsubseteq \text{Person} \sqcap \neg \text{Female} \sqcap \exists \text{hasChild}.\top \sqcap$$

$$\forall \text{hasChild}.(\text{Person} \sqcap (\neg \text{Female} \sqcup \text{Female}) \sqcap \text{Offspring})$$

- Acyclic if no concept inclusion axiom includes cyclic descriptions.

- General if there exists no restriction on the type of concept inclusion axioms that can be used.

A TBox $\mathcal{T}$ is said to be consistent if there exists an interpretation $\mathcal{I}$ satisfying $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for each $C \sqsubseteq D \in \mathcal{T}$. $\mathcal{I}$ is called a model of $\mathcal{T}$.

TBox axioms	
Man	$\sqsubseteq$ Person $\sqcap$ Male
Child	$\sqsubseteq$ Person $\sqcap$ (Male $\sqcup$ Female) $\sqcap$ Offspring
Male	$\sqsubseteq$ ¬Female
Father	$\sqsubseteq$ Man $\sqcap$ ∃hasChild.⊤ $\sqcap$ ∀hasChild.Child
ABox assertions	
$\langle$Joseph, Jonas$\rangle$: hasChild	
Joseph: Man	

Figure 1: Basic DL knowledge base consisting of a TBox and an ABox.

In the ABox, one describes a specific state of domain elements in terms of concepts and roles. Domain elements are referred to using individuals, by giving them names, and one asserts properties of these individuals using ABox assertions.

Definition 2.1.6 (ABox) Let a, b be two individual names in N_{I}, an ABox $\mathcal{A}$ is defined as a finite set (possibly empty) of

- Concept Membership Assertions of the form $a : C$ (e.g., Joseph : Man), and/or

- Role Membership Assertions of the form $\langle a, b \rangle : R$ (e.g., $\langle$Joseph, Jonas$\rangle$: hasChild).

An ABox $\mathcal{A}$ is said to be consistent w.r.t. $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $a^{\mathcal{I}} \in C^{\mathcal{I}}$ is satisfied for each $a : C$ in $\mathcal{A}$ and $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$ is also satisfied for each $(a, b) : R$ in $\mathcal{A}$. $\mathcal{I}$ is a model of $\mathcal{A}$.

Definition 2.1.7 ($\mathcal{ALC}$ Knowledge Base) A typical DL knowledge base (KB) represented using $\mathcal{ALC}$ consists of two components: a TBox (Terminological Box), and an ABox (Assertional Box) as shown in Figure 1. An $\mathcal{ALC}$ KB is defined as a tuple $\mathcal{K}=(\mathcal{T}, \mathcal{A})$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ is said to be a model of $\mathcal{K}$ iff $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{I}$ is a model of $\mathcal{A}$.

2.1.2 More Expressive DLs

In order to meet the expressivity needs of certain application domains, various DL constructors have been investigated in terms of expressivity and decidability of their corresponding inference services. The corresponding languages are identified by a string of the form $\mathcal{ALC}_{[||\mathcal{C}||]}[\mathcal{S}][\mathcal{R}][\mathcal{H}][\mathcal{O}][\mathcal{I}][\mathcal{Q}][(\mathbf{D})]$ where $\mathcal{ALC}$ stands for basic DL and every letter stands for a certain constructor. One of the most expressive DL languages is now $\mathcal{SROIQ}$ [HKS06] which extends $\mathcal{ALC}$ with *transitive roles*, role composition ($\mathcal{R}$), *nominals* ($\mathcal{O}$), *inverse roles* ($\mathcal{I}$), and *qualified cardinality restrictions* ($\mathcal{Q}$) and is now the DL underlying OWL 2 [MPSCG08]. On the other hand, a less expressive fragment of $\mathcal{ALC}$, the DL $\mathcal{EL}$ [BBL05][1], has recently drawn considerable attention for modelling large scale biomedical KBs. This thesis only considers extensions of $\mathcal{ALC}$ enabling $[\mathcal{O}]$ and $[\mathcal{Q}]$, namely $\mathcal{SHOQ}$. The DL constructors can be grouped into concept constructors and role constructors.

---

[1]$\mathcal{EL}$ underlies the designated OWL2-EL profile of OWL 2 where the only available constructors are $\top$, conjunction ($\sqcap$), and existential restriction ($\exists$).

Concept Constructor	Syntax	Interpretation $\cdot^{\mathcal{I}}$
$\mathcal{ALC}$		
Top	$\top$	$\Delta^{\mathcal{I}}$
Bottom	$\bot$	$\emptyset$
Negation	$(\neg C)$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
Conjunction	$(C \sqcap D)$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
Disjunction	$(C \sqcup D)$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
Value Restriction	$(\forall R.C)$	$\{s \in \Delta^{\mathcal{I}} \mid \forall t \in \Delta^{\mathcal{I}} : \langle s,t \rangle \in R^{\mathcal{I}} \Rightarrow t \in C^{\mathcal{I}}\}$
Existential Restriction	$(\exists R.C)$	$\{s \in \Delta^{\mathcal{I}} \mid \exists t \in \Delta^{\mathcal{I}} : \langle s,t \rangle \in R^{\mathcal{I}} \wedge t \in C^{\mathcal{I}}\}$
Nominals		
$\mathcal{O}$	$\{o\}$	$\{s \in \Delta^{\mathcal{I}} \mid \#\{o\}^{\mathcal{I}} = 1\}$
hasValue	$\exists R.\{o\}$	$\{s \in \Delta^{\mathcal{I}} \mid \exists t \in \Delta^{\mathcal{I}} : \langle s,t \rangle \in R^{\mathcal{I}} \wedge t \in \{o\}^{\mathcal{I}}\}$
OneOf	$\{o_1, \ldots, o_n\}$	$\{o_1\}^{\mathcal{I}} \cup \cdots \cup \{o_n\}^{\mathcal{I}}, n \geq 1$
Number Restriction		
$\mathcal{N}$	$\leq nR.\top$	$\{s \in \Delta^{\mathcal{I}} \mid \#(FIL(R,s)) \leq n\}, n \in \mathbb{N}$
	$\geq nR.\top$	$\{s \in \Delta^{\mathcal{I}} \mid \#(FIL(R,s)) \geq n\}, n \in \mathbb{N}$
Qualified Cardinality Restriction		
$\mathcal{Q}$	$\leq nR.C$	$\{s \in \Delta^{\mathcal{I}} \mid \#(FIL(R,s) \cap C^{\mathcal{I}}) \leq n\}$
	$\geq nR.C$	$\{s \in \Delta^{\mathcal{I}} \mid \#(FIL(R,s) \cap C^{\mathcal{I}}) \geq n\}$
Role Hierarchies		
$\mathcal{H}$	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Transitive Roles		
$\mathcal{S}$	$\mathsf{Trans}(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}, (R^{\mathcal{I}} \text{ transitive})$

Figure 2: Syntax and semantics of the DL $\mathcal{SHOQ}$.

2.1.2.1 Concept Constructors

Expressive DL concept constructors are operators used to extend the syntax and semantics of $\mathcal{ALC}$ in order to capture the descriptions of more elements within a certain domain.

Concept Cardinalities $(\|\mathcal{C}\|)$ Concept cardinalities were proposed in [BBH96] to express a restriction (at-least and at-most) on the number of instances that a certain concept can have within the represented domain. For example, the concept cardinality restriction $(\leq 195\,\mathsf{Country})$ restricts the number of instances of the concept $\mathsf{Country}$ to 195. *Concept cardinalities* extend the $\mathcal{ALC}$ syntax rule in (1) as follows:

$$\mathcal{ALC}_{\|C\|}\text{-concept} \longrightarrow \mathcal{ALC}\text{-concept}\,|\,(\geq nC)\,|\,(\leq nC) \tag{2}$$

The set of *concept cardinalities* used within a KB $\mathcal{K}$ are grouped into a CBox $\mathcal{C}$. An interpretation $\mathcal{I}$ is a model of a CBox $\mathcal{C}$ iff $\mathcal{I}$ is a model of $\mathcal{K}$, and $\mathcal{I}$ it satisfies all *concept cardinalities* in $\mathcal{C}$. Due to their high computational complexity, *concept cardinalities* are not adopted by standard DL languages neither are they part of OWL. They are introduced due to their correspondence with the *nominals* constructor defined below.

Nominals $(\mathcal{O})$ Nominals, known as named individuals, are studied in the areas of hybrid logics [BM01] as well as DLs [Sch94]. In DL, *nominals* allow the naming of domain elements (ABox individuals) to be used within concept descriptions in the TBox. Without the *nominals* constructor, the TBox axioms and the ABox assertions are separated. *Nominals* are used as concept names that must be interpreted as singleton sets and play an important role in DL because they allow one to express the notion of uniqueness and identity. There exist many natural concepts that need

to be modelled using *nominals* such as "Sun", "God", "Concordia University", etc ... Extending $\mathcal{ALC}$ with *nominals* is obtained by additionally defining a set $N_{\text{o}} \subseteq N_{\text{I}}$ of *nominals* each treated as a concept name. When *nominals* are allowed, the $\mathcal{ALC}$ syntax rule in (1) is extended as follows:

$$\mathcal{ALCO}\text{-concept} \longrightarrow \mathcal{ALC}\text{-concept} \,|\, \{o_1, \ldots, o_n\} \tag{3}$$

with $n \geq 1$ and $o_1, \ldots, o_n$ elements in N_{o}. In the literature, sometimes a distinction is made between the *hasValue* constructor and the *oneOf* constructor.

- The *hasValue* constructor uses *nominals* as part of an existential restriction as in (EuropeanCountry $\sqsubseteq$ $\exists$locatedIn.{Europe}) where Europe is a *nominal* (referring to a continent) used to define the concept of a European country as "a country located in the European continent".

- The *oneOf* constructor enumerates *nominals* to define a concept such as

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

where Asia, ..., Australia are all *nominals*. To distinguish a *nominal* name from a concept name within a concept description a *nominal* name is usually surrounded by "{}". Note that throughout this thesis sometimes the "{}" are omitted if the distinction is obvious. On the semantic side, an interpretation $\mathcal{I}$ is required to map every $o \in N_{\text{o}}$ to a singleton set as shown in Figure 2 where # denotes the cardinality of a set. This means that *nominals* can emulate *concept cardinalities*, for example the concept Continent, as defined using the *oneOf* constructor, can have exactly 7 instances, the enumerated *nominals*. Note that, due to their semantic equivalence, sometimes an enumeration of *nominals* is replaced with a disjunction between the enumerated *nominals*. For example, the concept Continent can also be represented using the following syntax: Continent $\equiv$ {Asia} $\sqcup$ {Africa} $\sqcup \cdots \sqcup$ {Australia}.

20

Number Restrictions ($\mathcal{N}$)   Using Number Restrictions, it is possible to specify a lower (at-least restriction) and an upper (at-most restriction) bound on the number of individuals related via a certain role ($R \in N_{\mathrm{R}}$). For example, the *number restriction* ($\geq 279\,\mathsf{hasMember}$) can be used to specify that the $\mathsf{CanadianParliament}$ "must be constituted of at-least 279 seats". *Number restrictions* were studied in [BS99], they extend the $\mathcal{ALC}$ syntax rule in (1) as follows:

$$\mathcal{ALCN}\text{-concept} \longrightarrow \mathcal{ALC}\text{-Concept} \,|\, (\geq nR) \,|\, (\leq nR) \tag{4}$$

On the semantic side, an interpretation $\mathcal{I}$ makes sure that R-fillers satisfy each *number restriction* by extending the interpretation function $.^{\mathcal{I}}$ as shown in Figure 2.

Definition 2.1.8 (R-filler)   A domain element $t \in \Delta^{\mathcal{I}}$ is said to be an R-filler if there exists a domain element $s \in \Delta^{\mathcal{I}}$ such that $\langle s, t \rangle \in R^{\mathcal{I}}$ for a given role $R \in N_{\mathrm{R}}$, t is said to be an R-filler of s. The set of R-fillers of a given domain element $s \in \Delta^{\mathcal{I}}$ is defined as $FIL(R,s) = \{t \in \Delta^{\mathcal{I}} \,|\, \langle s, t \rangle \in R^{\mathcal{I}}\}$ and the set of all R-fillers for a given role R is defined as: $FIL(R) = \bigcup_{s \in \Delta^{\mathcal{I}}} FIL(R,s)$.

Qualified Cardinality Restrictions ($\mathcal{Q}$)   Qualified Cardinality Restrictions (QCRs) act like *number restriction* with additionally specifying qualities on the related individuals. For example, the QCR ($\geq 279\,\mathsf{hasMember.CanadianCitizen}$) specifies that the $\mathsf{CanadianParliament}$ "must be constituted of at-least 279 seats while additionally specifying that members of the Canadian Parliament must be Canadian citizens". When QCRs [HB91] are allowed, the $\mathcal{ALC}$ syntax rule in (1) is extended as follows.

$$\mathcal{ALCQ}\text{-Concept} \longrightarrow \mathcal{ALC}\text{-Concept} \,|\, (\geq nR.C) \,|\, (\leq nR.C) \tag{5}$$

On the semantic side, an interpretation $\mathcal{I}$ is required to satisfy each QCRs by extending the interpretation function $.^{\mathcal{I}}$ as shown in Figure 2.

2.1.2.2 Role Constructors

Different role constructors can be used to express complex role relations using atomic roles. Interesting role constructors have been investigated in the literature (e.g., *transitive roles*, *role hierarchies*, boolean operators on roles and role composition [HKS06]). However, only *role hierarchies*, *transitive roles*, and *inverse roles* will be discussed in this thesis.

Role Hierarchies ($\mathcal{H}$)   Role hierarchies are used to define sub-role and super-role relationships between the roles used in the TBox, they are expressed using a set of Role Inclusion Axioms.

Definition 2.1.9 (Role Inclusion Axiom (RIA))   A RIA is an expression of the form $R \sqsubseteq S$, where $R, S \in N_{\mathrm{R}}$. For example the RIA (hasBrother $\sqsubseteq$ hasSibling) specifies that "every brother is also a sibling"; every individual b related to an individual a via the hasBrother role is also related to a via the hasSibling role. Their semantics are preserved by extending the interpretation function $.^{\mathcal{I}}$ as in Figure 2. A set of RIA is referred to as an RBox $\mathcal{R}$ and is part of the knowledge base $\mathcal{K}$. The transitive reflexive closure of $\sqsubseteq$ on $\mathcal{R}$ is referred to as $\sqsubseteq_*$.

Definition 2.1.10 (Sub-role/super-role)   A role S is said to be a sub-role of a role R in $\mathcal{T}$ if there exists a RIA $\in \mathcal{R}$ such that $(S \sqsubseteq_* R)$. R is said to be a super-role of S in $\mathcal{T}$. For example, hasBrother is a sub-role of hasSibling which is a super-role of hasBrother.

Transitive Roles Transitive roles can be used to represent transitive relations between concepts. For example, $\mathsf{Trans}(\mathsf{isPartOf})$ defines the role $\mathsf{isPartOf}$ as a *transitive role*. The set $N_{R^+} \subseteq N_R$ denotes the set of *transitive roles*.

Definition 2.1.11 (Simple Role) A role R is said to be simple if it is neither transitive nor has a transitive sub-role.

Inverse Roles ($\mathcal{I}$) Inverse Roles are used to express converse relations between individuals using the $^-$ operator. For example, $(\mathsf{hasMember} = \mathsf{isMemberOf}^-)$ can be used to express that $\mathsf{hasMember}$ is the converse relation of $\mathsf{isMemberOf}$. Their semantics are preserved by extending the interpretation function $.^{\mathcal{I}}$ such that given the role S defined as the inverse role of R ($S \equiv R^-$) the following holds:

$$\langle s, t \rangle \in S^{\mathcal{I}} \Longleftrightarrow \langle t, s \rangle \in R^{\mathcal{I}}$$

Concrete Datatypes (D) Concrete Datatypes are used to represent literal values such as numbers and strings. They can be used to describe concepts such as "a toddler is a child whose age is between 1 and 3" ($\mathsf{Child} \sqcap \exists \mathsf{hasAge}.(\mathsf{max3}) \sqcap \mathsf{hasAge}.(\mathsf{min1})$) where $(\mathsf{max3})$ and $(\mathsf{min1})$ are datatypes derived by adding minimum and maximum value constraints on an integer datatype.[2] Extending $\mathcal{ALC}$ with *concrete datatypes* is obtained by additionally defining a set $\mathbf{D}$ of datatypes, and a set N_T of role names such that N_R and N_T are disjoint and $T \in N_T$ is referred to as a concrete role. The $\mathcal{ALC}$ syntax rule in (1) is extended as follows where C is an $\mathcal{ALC}(\mathbf{D})$-Concept, T is a concrete role, and $d \in \mathbf{D}$.

$$\mathcal{ALC}_{(\mathbf{D})}\text{-concept} \longrightarrow \mathcal{ALC}\text{-Concept} \,|\, (\exists T.d) \,|\, (\forall T.d) \tag{6}$$

---

[2]Integers and strings are referred to as primitive datatypes. More complex datatypes can be derived using boolean combinations of primitive datatypes and *number restrictions* qualified with datatypes [HS01].

On the semantic level, the domain of all datatypes is represented by $\Delta_{\mathbf{D}}$ such that:

- each $d \in \mathbf{D}$ is associated with a set $d^{\mathbf{D}} \subseteq \Delta_{\mathbf{D}}$

- $\Delta_{\mathbf{D}}$ is disjoint with $\Delta^{\mathcal{I}}$

- a negated concrete datatype $\neg d$ is interpreted as $\Delta_{\mathbf{D}} \setminus d^{\mathbf{D}}$

- the interpretation function $.^{\mathcal{I}}$ is extended to preserve the semantics of concepts using concrete roles as shown below:

$$
\begin{aligned}
(\exists R.d)^{\mathcal{I}} &= \{s \in \Delta^{\mathcal{I}} \mid \#\{t \mid \langle s,t \rangle \in R^{\mathcal{I}} \wedge t \in d^{\mathbf{D}}\} = 1\} \\
(\forall R.d)^{\mathcal{I}} &= \{s \in \Delta^{\mathcal{I}} \mid \forall t : \langle s,t \rangle \in R^{\mathcal{I}} \Rightarrow t \in d^{\mathbf{D}}\}
\end{aligned}
$$

2.1.3 DL Inference Services

What makes DLs interesting is that they allow one to represent domain knowledge not only for the sake of representing it, but also to reason about it and make implicit knowledge explicit through the use of some inference services. This section briefly introduces standard DL reasoning services: TBox reasoning and ABox reasoning. A more technical description of these services can be found in [BH91b, BLS06, FB07b].

TBox Reasoning: TBox inference services work on concept descriptions in the TBox without any reference to the ABox. These services include:

- **Satisfiability Checking:** A concept C is said to be satisfiable w.r.t. a TBox $\mathcal{T}$ iff there exists a model $\mathcal{I}$ of $\mathcal{T}$ with $C^{\mathcal{I}} \neq \emptyset$, i.e., there exists an individual $s \in C^{\mathcal{I}}$ as an instance of C. $\mathcal{I}$ is called a model of C w.r.t. $\mathcal{T}$.

- **Subsumption Checking:** Given two concept expressions C and D, the concept subsumption service tries to infer subsumption relationships between the

two concepts ($C \sqsubseteq D$ and/or $D \sqsubseteq C$) w.r.t. $\mathcal{T}$. A concept C is subsumed by a concept D ($C \sqsubseteq D$) iff every model $\mathcal{I}$ of $\mathcal{T}$ also satisfies $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A subsumption test can be polynominally reduced to a satisfiability test; $C \sqsubseteq D$ iff the concept expression $C \sqcap \neg D$ is not satisfiable.

- **Classification:** Using subsumption inferences, concept names that appear in the TBox can be arranged in a hierarchical way. This process inference, called classification, is one of the main inference services offered by many DL systems.

ABox Reasoning: ABox inference services work on both components of a DL KB $\mathcal{K}$: the TBox and the ABox. The basic inference services are:

- **KB Consistency:** A knowledge base $\mathcal{K}(\mathcal{T}, \mathcal{A})$ is consistent if it admits a common model for $\mathcal{T}$ and $\mathcal{A}$. Having a consistent knowledge base is crucial for the usefulness of any other inference service; any inference is meaningless if the KB at hand is not consistent.

- **Instance Checking:** An instance checking service consists of checking whether an ABox individual s is an instance of a concept C w.r.t. $\mathcal{T}$ and $\mathcal{A}$. The test is positive if $s^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models, $\mathcal{I}$, of $\mathcal{A}$ and $\mathcal{T}$.

Using *nominals*, concept satisfiability and ABox consistency can be reduced to TBox consistency; a concept C is satisfiable w.r.t. a TBox $\mathcal{T}$ iff ($\mathcal{T} \cup \{\{o\} \sqsubseteq C\}$) is consistent and $o \in N_{\mathrm{o}}$ new in $\mathcal{T}$, an ABox $\mathcal{A}$ is consistent w.r.t. $\mathcal{T}$ iff ($\mathcal{T} \cup \bigcup_{(a:C) \in \mathcal{A}} \{\{a\} \sqsubseteq C\} \cup \bigcup_{((a,b):R) \in \mathcal{A}} \{\{a\} \sqsubseteq \exists R.\{b\}\}$) is consistent. Also, DLs that enable both *transitive roles* and *role hierarchies* can reduce reasoning w.r.t a TBox into reasoning without TBoxes using a technique called *internalization* [HS07, HST99],

which encapsulates TBox axioms into a single concept $C_{\mathcal{T}}$ such that:

$$C_{\mathcal{T}} = \bigsqcap_{(C_i \sqsubseteq D_i) \in \mathcal{T}} (\neg C_i \sqcup D_i) \tag{7}$$

Satisfiability of a concept D w.r.t $\mathcal{T}$ can be reduced to testing the satisfiability of $D \sqcap C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}$ where U is a Transitive Universal Role, super-role of all roles in $\mathcal{T}$.

2.2   DL Reasoning

Many reasoning methods were investigated to handle Dl inference services such as *structural subsumption* (early 90s), *tableau-based* (1991), *automata-based* (2003) [BHLW03], *semantic binary tree* (2005), and *resolution-based* (2006). However, the most widely used one remains *tableau-based* which is introduced in this section.

2.2.1   Tableau Algorithms

The first DL tableau algorithm was designed for the DL $\mathcal{ALC}$ in 1991 [SSS91] and later extended for more expressive logics [BS01, HM04, HKS06, HS01]. The first systems to implement tableau algorithms were KRIS (1991) [BH91a] and CRACK (1995) which behaved reasonably well in practice even though the supported DL inference problems were at-least NP- or even PSPACE-Hard.

In general, tableau algorithms are considered as goal-directed decision procedures. They try to decide the satisfiability of a concept expression by constructing a corresponding model. The idea is that a concept C is satisfiable if a model exists that corresponds to an interpretation of C such that $C^{\mathcal{I}} \neq \emptyset$. Tableau algorithms work on concepts in *Negation Normal Form* [FB07b], they are characterized by an underlying data structure, a set of expansion rules, a number of so-called clash-triggers, and sometimes a set of blocking strategies.

Definition 2.2.1 (Negation Normal Form (NNF)) A concept expression is said to be in NNF if the negation ($\neg$) appears only in front of concept names. NNF can be obtained by pushing negations inwards [HS01] using DeMorgan's law (1-2) and the following equivalences:

(1) $\neg(C \sqcup D) \iff \neg C \sqcap \neg D$ (4) $\neg(\forall R.C) \iff \exists R.\neg C$

(2) $\neg(C \sqcap D) \iff \neg C \sqcup \neg D$ (5) $\neg(\geq nR.C) \iff\ \leq (n-1)R.C$

(3) $\neg(\neg C) \iff C$ (6) $\neg(\leq nR.C) \iff\ \geq (n+1)R.C$

In the following, $nnf(C)$ denotes the NNF of C and $\dot{\neg}C$ denotes the NNF of $\neg C$.

The Data Structure The data structure used to describe the model of a given concept C is usually a directed graph $G(V, E)$ referred to as a "completion graph". V is a set of vertices representing individuals in the domain, and E is a set of edges representing relations between individuals. Every node $x \in V$ is labeled by a set of concept expressions $\mathcal{L}(x)$ that is satisfied by the represented individual. Every edge between two nodes, x and y, is labeled by a set, $\mathcal{L}\langle x, y \rangle$, of role names satisfying the dependencies (Role successor-ship) between the two nodes. A symmetric binary relation $\neq$ is used to keep track of inequalities between two nodes. For most DLs, the construction of a model starts by initializing a root node (x_0) in G such that x_0 must satisfy the concept expression C. This is ensured by setting the label of x_0 to C $(\mathcal{L}(x_o) = \{C\})$.

Definition 2.2.2 (Role-Successor) A node y is said to be a *Role*-Successor of a node x if there exists an edge $\langle x, y \rangle$ with R in its label $(R \in \mathcal{L}\langle x, y \rangle)$ for some $R \in N_{\mathrm{R}}$.

The node y is said to be an R-successor of x which is then said to be ancestor of y.

⊓-Rule **If** $C \sqcap D \in \mathcal{L}(x)$, x is not blocked, and $\{C, D\} \not\subseteq \mathcal{L}(x)$

Then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C, D\}$

⊔-Rule **If** $C \sqcup D \in \mathcal{L}(x)$, x is not blocked, and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$

Then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{E\}$ with $E \in \{C, D\}$

∃-Rule **If** $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and there exists no y such that:

y is a R-successor of x with $C \in \mathcal{L}(y)$

Then create a new node y and set $\mathcal{L}(\langle x, y \rangle) = \mathcal{L}(\langle x, y \rangle) \cup \{R\}$, and

$\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

∀-Rule **If** $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and there exists y such that:

y is an R-successor of x, and $C \notin \mathcal{L}(y)$

Then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

Figure 3: Tableau expansion rules for $\mathcal{ALC}$.

The Expansion Rules The graph G is gradually expanded according to some expansion rules designed to preserve and construct the logical dependencies encoded in C. These expansion rules (also known as tableau completion rules), correspond to constructors in the logic, they expand the initial graph by describing sub-graphs of the completion graph before and after rule application. In many cases, they only operate on a node and its direct neighbours. Figure 3 shows the tableau expansion rules corresponding to the DL $\mathcal{ALC}$.

28
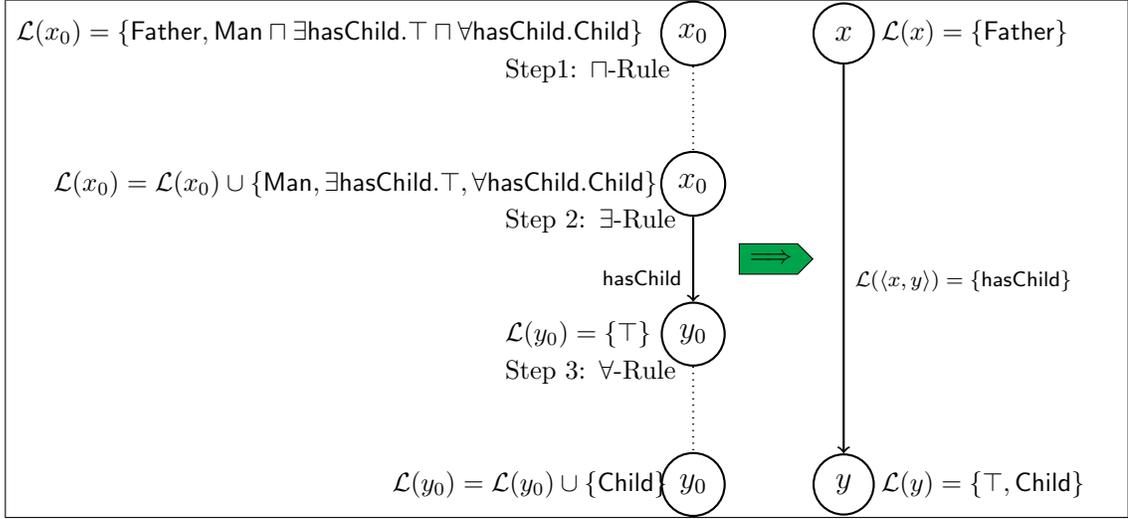
Figure 4: Tableau expansion during a satisfiability test of the concept Father as defined in Figure 1. The expansion on the left shows the step by step application of the expansion rules. The expansion on the right, shows the final completion graph representing a model for the concept Father.

Some rules add new nodes (e.g., $\exists$-Rule), and others yield more than one possible outcome (e.g., $\sqcup$-Rule). The latter ones are known as non-deterministic rules. In order to ensure termination in the case when cyclic descriptions are encountered, the rules employ the notion of blocking, which will be introduced in the next Section (see Definition 2.2.4). In practice, non-determinism means search and it is dealt with by exploring the various possible models. For an un-satisfiable concept, all possible expansions will lead to the discovery of an obvious contradiction known as a clash (see Definition 2.2.3), however, if at-least one expansion leads to a complete and clash-free completion graph, then the concept is satisfiable. When no more rules are applicable, it means that all implicit knowledge has been made explicit and the completion graph is said to be complete. In the case of a satisfiable concept C, a complete and clash free completion graph is found and is said to be a completion model of C. Figure 4 shows the step by step expansion of the completion model for the satisfiability of the concept Father as defined in Figure 1.

29

$$\mathcal{L}(x_0) = \{C, \exists R.C\} \quad x_0$$
Step1: ∃-Rule

$$\mathcal{L}(y_1) = \{C, \exists R.C\} \quad y_1$$
Step 2: ∃-Rule

$$R$$

$$\mathcal{L}(y_2) = \{C, \exists R.C\} \quad y_2$$
Step n: ∃-Rule

$$\mathcal{L}(y_n) = \{C, \exists R.C\} \quad y_n$$

(a) Without blocking

$$\mathcal{L}(x_0) = \{C, \exists R.C\} \quad x_0$$
Step1: ∃-Rule

$$\mathcal{L}(y_1) = \{C, \exists R.C\} \quad y_1$$
Step 2: ∃-Rule

$$R$$

(b) With blocking
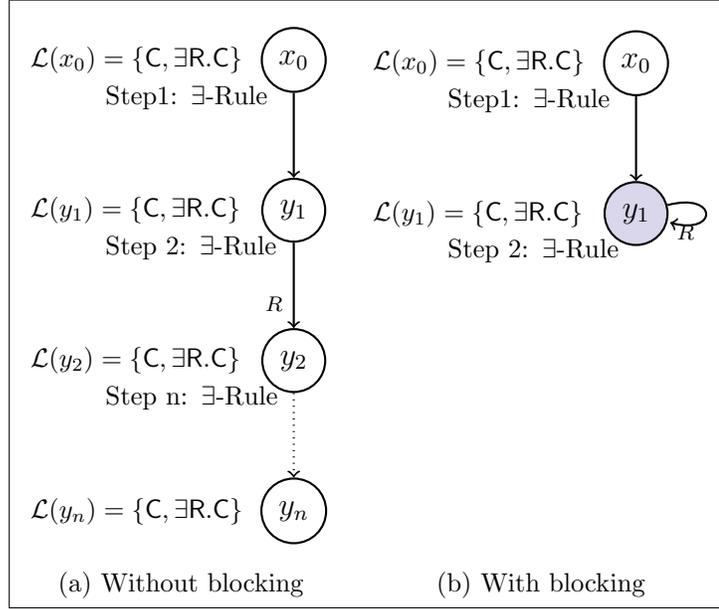
Figure 5: Tableau model for the satisfiability of $C \sqsubseteq \exists R.C$.

Clash Triggers The tableau expansion algorithm stops either when no more rules are applicable (i.e., the completion graph is complete) or when a clash is detected.

Definition 2.2.3 (Clash) A node x is said to contain a clash when a logical dependency is violated such as having x satisfy C and $\neg C$ ($\{C, \neg C\} \subseteq \mathcal{L}(x)$).

Blocking In some cases, expanding the completion graph does not lead to a complete graph. This can happen if the TBox axioms include cycles. For example, if a TBox $\mathcal{T}$ contains the axiom: $C \sqsubseteq \exists R.C$, then the satisfiability of C w.r.t. $\mathcal{T}$ will never stop because the tableau algorithm can go on creating new individuals with repeating structure as shown in the Figure 5a. These situations are handled using *blocking* [HST99, BS01]; the idea is to block a node from applying rules if it needs to satisfy a concept expression that is satisfied by one of its ancestors. Such is the case with the node y_1, in Figure 5b, which is blocked by the node x_0. The main idea is that the blocked node y_1 can use R-successors of x_0 instead of generating new ones.

Definition 2.2.4 (Blocked Node) A node x is said to be directly blocked by a node y if it has an ancestor node y such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$. The node x is said to be blocked if it is directly blocked or one of its ancestors is blocked.

2.2.2 Tableau Algorithms for Expressive DLs

When dealing with more expressive DLs, tableau algorithms need to be extended in many ways in order to preserve the semantics of the handled constructors.

2.2.2.1 Extending the Datastructure

The data structure might need some changes to reflect any additional properties. For example, when *nominals* are enabled, a distinction is made between *nominal nodes* and the so-called *blockable nodes*. A *nominal node* is a node whose label contains a nominal whereas a *blockable node* is a node that is not a *nominal node*. *Nominal nodes* can be arbitrarily interconnected, they are found in arbitrarily complex graphs whereas *blockabe nodes* are only found in tree-like graphs rooted in *nominal* nodes [HKS06, HS05]. This distinction helps preserves the semantics of *nominals* when applying expansion rules, clash triggers or blocking strategies. For example, when merging two nodes is necessary (e.g., case with the $\leq$-Rule) a *blockable node* is always merged with a *nominal node* in order to preserve the *nominals* semantics. Initially, a *nominal* node is added to the completion graph for every *nominal* in the KB. The graph therefore, becomes a forest of tree-like graphs rooted in these *nominal* nodes.

2.2.2.2 Extending Tableau Rules

Tableau rules need to be extended to support additional available constructors as follows:

$\forall_+$-Rule	**If** $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and there exists y such that:
	y is an S-successor of x, $S \in N_{R^+}$ with $S \sqsubseteq_* R$, and $\forall S.C \notin \mathcal{L}(y)$
	Then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall S.C\}$
choose-Rule	**If** $\leq nR.C \in \mathcal{L}(x)$, x is not blocked, and there exists y such that:
	y is an R-successor of x with $\mathcal{L}(y) \cap \{C, \dot{\neg} C\} = \emptyset$
	Then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{E\}$ with $E \in \{C, \dot{\neg} C\}$
$\geq$-Rule	**If** $\geq nR.C \in \mathcal{L}(x)$, x is not blocked, and there are no $y_1, \ldots, y_n$
	R-successors of x with $C \in \mathcal{L}(y_i)$, and $y_i \neq y_j$ for $1 \leq i \leq j \leq n$
	Then create n new nodes $y_1, \ldots, y_n$ as R-successors of x such that
	for $1 \leq i \leq j \leq n$ set $y_i \neq y_j$, and $\mathcal{L}(y_i) = \{C\}$
$\leq$-Rule	**If** $\leq nR.C \in \mathcal{L}(x)$, x is not blocked, and there are $y_1, \ldots, y_m$
	R-successors of x with $C \in \mathcal{L}(y_i)$, and $m \geq n + 1$
	Then select y_j and y_i such that y_j such that not $y_j \neq y_i$, and
	- if y_j is a nominal node, then Merge (y_j, y_i), and remove y_i
	- else if y_i is a nominal node or an ancestor of y_j, then Merge (y_j, y_i)
	- else Merge (y_i, y_j) and remove y_i
O-Rule	**If** $\{o\} \in \mathcal{L}(x) \cap \mathcal{L}(y)$, and not $x \neq y$
	Then Merge (y, x) and remove y

Figure 6: Tableau expansion rules handling the semantics of the added constructors extending $\mathcal{ALC}$ to $\mathcal{SHOQ}$.

- With *number restrictions* ($\mathcal{N}$), the rules in Figure 6 must be added to the ones in

Figure 3 to ensure that individuals satisfy the at-least and at-most restrictions expressed using this constructor. The main idea is that at-least restrictions are treated by generating the required role-successors as new distinct nodes. At most restrictions are treated by non-deterministically merging role-successors whenever the number of role-successors exceeds the number allowed by the restrictions. A create and merge cycle is avoided using the inequality relations between nodes created to satisfy an at-least restriction such that two nodes x and y cannot be merged if $x \neq y$.

- Treating QCRs ($\mathcal{Q}$) is similar to treating *number restrictions* with additionally asserting that the newly created role-successors are also members of the qualifying concept C. Also, in order to detect unsatisfiability of concepts such as $(\geq nR \sqcap \leq mR.C \sqcap \leq mR.\neg C)$ with $n > m$, the non-deterministic *choose*-rule is used such that all R-successors are non-deterministically distributed over C or $\neg C$.

- The *nominals* ($\mathcal{O}$) semantics are handled by extending tableau rules with the *o*-Rule [HS01]. The *O*-Rule works by merging two *nominal* nodes that contain the same nominal in their label. This merging is needed in order to ensure that *nominals* are interpreted as singletons. Also the $\leq$-Rule makes sure that a *nominal node* is never merged with a *blockable node* leading to the loss of the *nominal node*.

2.2.2.3 Extending Clash Triggers

Clash triggers are also extended to include all possible types of clashes.

- With *number restrictions* ($\mathcal{N}$) a new type of clash needs to be detected: when a node x that must satisfy an at-most restriction ($\leq nR$) on its R-successors, and it already has m distinct R-successors with $m > n$.

- With *nominals* ($\mathcal{O}$), clash triggers need also detect the type of clash causing a violation of the *nominals* semantics such as having two distinct nodes, $x \neq y$, with the same *nominal* in their label i.e. having $\{o\} \subseteq (\mathcal{L}(x) \cap \mathcal{L}(y))$.

2.2.2.4 Extending Blocking Strategies

With more expressive logics, sometimes more sophisticated *blocking* strategies need to be used in order to ensure termination.

- In the presence of *nominals*, blocking strategies need to make sure that none of the nodes between a blocking node and the blocked one is a *nominal node*. Otherwise by repeating the cycle, the *nominal* would also be repeated and the semantics violated.

- Transitive roles and GCIs also introduce cycles. The concept description $C \sqcap \exists R.C \sqcap \forall R.(\exists R.C)$ where R is a Transitive Role, the combination of the $\exists R.C$ and $\forall R.(\exists R.C)$ concepts would cause a new node y to be added to the tree with an identical label to x. The expansion process could then be repeated indefinitely. This problem can be dealt with by blocking: halting the expansion process when a cycle is detected.

2.2.2.5 Merging

Some expansion rules require the merging of two nodes in order to satisfy an at-most restriction. When a node x is merged with another node y (`Merge` (x,y)), y inherits all of x's properties including its label, inequalities, ancestors (incoming edges) and successors (outgoing edges). Therefore, the label of x needs to be added to the label of y ($\mathcal{L}(y) = \mathcal{L}(y) \cup \mathcal{L}(x)$), all edges that lead to x are updated so that they lead to y and those leading from x to *nominal* nodes so that they lead from y to the same

nominal nodes. The completion graph is then pruned by removing x and, recursively, all *blockable* role-successors of x.

2.2.2.6   Strategy of Rule Application

The implementations of tableau algorithms have shown that expansion rules often need to be applied according to a certain strategy in order to ensure termination of the procedure. The general idea of the strategy is to apply shrinking rules before any other rule, and to apply these rules to lower depth nodes before applying them to higher depth nodes.

2.2.3   Complexity of DL Reasoning

Analyzing the complexity of DL reasoning is part of studying the inherent difficulty of its reasoning services. A distinction is made between analyzing the computational complexity of an inference service, and analyzing the complexity of the underlying reasoning algorithms to solve an inference service.

The computational complexity of DL inference services is usually determined based on worst-case analysis of the size of a completion model, of a given KB, and the time needed to construct such model. Clearly DLs that enable *nominals*, QCRs, and GCIs enjoy additional expressive power (there is no other way to close a concept or domain with a finite number of elements using the DL $\mathcal{SHOQ}$ except using *nominals*) with high computational complexity. The complexity of reasoning with *nominals* was studied in [AL02, HS01, Sch94, Tob00, Tob01] where *nominals* were found to interact with other DL constructors such as $\mathcal{I}$ and $\mathcal{N}$, and affect the complexity. For example, while checking the satisfiability of an $\mathcal{ALCI}$ ($\mathcal{ALC}$ with *inverse roles*) concept is PSPACE-complete, adding a single *nominal* yields an ExpTime-complete concept Satisfiability problem [AL02]. The use of GCIs in TBoxes yields

35

an ExpTime-complete satisfiability problem for most expressive DLs extending $\mathcal{ALC}$. Table 1 shows the complexity of different DLs as can be found at the DL Complexity Navigator.[3]

DL Language	Satisfiability Checking	KB Consistency
$\mathcal{ALC}$, $\mathcal{ALCQ}$, $\mathcal{ALCOQ}$	PSpace-complete	PSpace-complete
$\mathcal{ALCHOQ}$	N/A	N/A
$\mathcal{SHOQ}$	ExpTime-complete	ExpTime-complete
$\mathcal{ALCOIQ}$	NExpTime-complete	NExpTime-complete
$\mathcal{SROIQ}$	NExpTime-hard	NExpTime-hard

Table 1: Computation complexity of DL inference services using an empty TBox.

The high worst-case complexity initially led to the conjecture that expressive DLs might be of limited practical applicability [BDS93]. Analyzing the efficiency of a DL reasoning algorithm consists of analyzing its soundness, completeness, and termination. While soundness is evaluated by making sure that the algorithm will always find the correct answer, completeness means that the algorithm will explore every possible cases before returning an answer, and termination means that the algorithm will always terminate. Termination is usually of great importance when studying the practical implication of a reasoning algorithm. This is because a correct (sound and complete) reasoning algorithm is of limited use if termination is not guaranteed. While worst-case complexity analysis serve as a theoretical estimate for the termination of a reasoning algorithm, the practical estimate is usually done through a performance analysis on average cases.

---

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

2.3 Practical DL Reasoners

There often remains a considerable gap between the theoretical presentation of a reasoning algorithm and a practical implementation. When analyzing the practical implication of a reasoning algorithm, one needs to distinguish between the theoretical efficiency of a reasoning algorithm (i.e., the theoretical worst case complexity compared to the worst case complexity of the corresponding inference problem), and the practical efficiency (i.e., practical typical case performance), of the reasoning algorithm. Early experiments with DL systems indicated that in practice performance, if not equipped with suited optimizations, is a serious problem even when considering systems handling less expressive extension of $\mathcal{ALC}$ such as $\mathcal{ALCN}$ [HKNP94]. The goal of designing optimization techniques is to achieve practical efficiency.

Most modern DL reasoners implement tableau-based algorithms together with a set of sophisticated optimization techniques. The state-of-the-art (SOTA) DL reasoners are Fact++, RacerPro,[4] Pellet,[5] and Hermit[6] which is based on a hyper-tableau algorithm.

- Fact++ [TH06] is a highly optimized tableau-based DL reasoner supporting OWL DL[7] and partially OWL 2.[8] Fact++ implements the tableau-based reasoning presented in [HS05], and is based on the FaCT system [Hor97] implemented to evaluate the practical efficiency of an optimized tableau-based algorithm for subsumption inferences in the presence of *transitive roles* and GCIs. The optimization techniques implemented by FaCT, such as *semantic branching*, *dependency directed backtracking*, and *absorption* are key techniques that

---

[4]http://www.racer-systems.com/
[5]http://clarkparsia.com/pellet/
[6]http://hermit-reasoner.com/
[7]OWL DL is a sublanguage of OWL which places a number of constraints on the use of the OWL language constructs. See http://www.w3.org/TR/owl-ref/ for more details.
[8]http://www.w3.org/TR/owl2-overview/

have become crucial for every practical DL reasoner implementing tableau-based reasoning. Those techniques are reviewed in Chapter 3.

- Pellet [SPG$^+$07] is a highly optimized tableau-based DL reasoner supporting OWL 2. Pellet was the first DL reasoner to handle *nominals* by implementing suited optimizations for reasoning with *nominals* [PCS06], such as *nominal absorption*, lazy forest generation, also adopted by Fact++.

- RacerPro [HM01b] is a highly optimized tableau-based DL reasoner supporting the DL $\mathcal{SHIQ}$. RacerPro is the only available reasoner providing algebraïc reasoning for efficient handling of QCRs based on the algorithm presented in [HTM01]. RacerPro also implements the signature calculus [HM01a] for dealing with QCRs.

These systems behave well in practice for fragments of DL logics that are optimized for realistic cases. There exist DL reasoners that adopt non-tableau reasoning algorithms for DL. The ones that can support the expressivity of *nominals* are Hoolet,[9] KAON2,[10] and Hermit.

- Hoolet uses a first order theorem prover, Vampire [RV02], to reason with the $\mathcal{SHOIN}$ DL language. With Hoolet, the TBox is translated into a collection of first order logic axioms, based on the logical language semantics, and sent to a theorem prover for consistency checking [TRBH04]. Hoolet works more as a proof of concept rather than as a practical DL reasoner; it is known to be sound but incomplete. It could be useful for testing and illustrating small examples with easy experimentation for expressive DL languages [Lie06].

- KAON2 is an infrastructure that can manage DL KBs. Its reasoning part consists of reducing a $\mathcal{SHIQ}$ KB to a disjunctive datalog program solved by

---

[9]http://owl.man.ac.uk/hoolet/
[10]http://kaon2.semanticweb.org/

well-known deductive database technology such as the magic set transformation [CFGL04]. Its reasoning with the $\mathcal{SHIQ}$ DL is proven sound and complete, yet there is little evidence for its usefulness as a practical DL reasoner compared to tableau-based reasoners. Recent work has been done to extend KAON2 decision procedure to handle *nominals* [KM06]. However, deciding concept Satisfiability in a $\mathcal{SHOIQ}$ KB runs in triple exponential time due to the interaction between *inverse roles*, *nominals* and *number restrictions*.

- Hermit is a recent hyper-tableau based DL reasoner supporting OWL 2. Hermit implements the hyper-tableau based reasoning presented in [MSH09], equipped with the optimizations discussed in [MSH07, GHM10, MH08] for efficient reasoning with GCIs.

2.4 Conclusion

This chapter introduced a formal definition of DL languages in terms of their syntax, semantics and inference services. *Nominals*, and QCRs not only affect the expressive power of a DL language, but also affect the complexity of its inference services as they require complex reasoning algorithms. The most widely used tableau-based reasoning algorithms for DL inferences were introduced. These algorithms are easy to implement however naïve implementations are not successful since their search-like behaviour means a high degree of non-determinism. A careful choice of reasoning algorithm equipped with suited optimization techniques is needed for practical DL reasoning. The following chapter discusses in more details the problem of DL reasoning with *nominals* and QCRs, and report on related research activities as well.

Chapter 3

DL Reasoning With Nominals and QCRs

Most Description Logics reasoners supporting *nominals* and QCRs implement tableau-based decision procedures which usually need to be equipped with a set of optimization techniques because their naïve implementations fail to be practical. Some optimizations have been proposed to enhance the handling of *nominals*, others to enhance the handling of QCRs. However, no optimizations were found to improve the handling of both *nominals* and QCRs. This chapter discusses the challenge of DL reasoning with *nominals* and QCRs; Section 3.1 discusses the sources of inefficient reasoning; Section 3.1.1 shows that a strong correlation exists between *nominals* and numbers, in such a way that *nominals* interact with QCRs; Section 3.2 reports on related work. Since the hybrid approach presented in this thesis is mostly inspired by algebraïc reasoning for DLs, Section 3.3 introduces the atomic decomposition technique which is a fundamental technique for such reasoning.

3.1 From Theory to Practice

There is little experience with DL reasoners supporting *nominals* and QCRs. Decision procedures for expressive DLs enabling both *nominals* and QCRs were published in [HS07] with very weak implementations if any (no DL reasoner was able to classify the WINE[1] ontology until the first efforts in [PCS06]). No existing DL reasoner is able to decide the satisfiability of the Future_EU concept (described in Chapter 1).

The challenge of DL reasoning with *nominals* arise from their syntax and semantics. Recall that the syntax of the nominal constructor allows ABox individuals to be referenced in the TBox. This syntax breaks the TBox-ABox separation, which is usually desired in order to separate TBox reasoning and ABox reasoning by developing separate reasoning procedures. The semantics of *nominals* is challenging because each nominal must be interpreted as exactly one individual, whereas a concept is interpreted as a set of individuals (the formal description of the syntax and semantics of *nominals* was shown in Figure 2). Extending a tableau-based reasoning algorithm with *nominals* was shown in Section 2.2.2; it requires a clear distinction between a *nominal node* and an *individual node* (so-called *blockable node*) in order to preserve the nominal semantics. For instance, in the case when two *individual nodes* need to be merged and one node is a *nominal node*; the *nominal node* must survive (see Figure 11 for an example). In the case when blocking is applicable due to a cycle; there cannot be a *nominal node* between an *individual node* and a blocking *individual node* otherwise by repeating the cycle, the nominal would also be repeated and the semantics is violated. The interaction between *nominals* and numerical restrictions imposed by QCRs leads to the loss of the tree model property, which is usually advantageous for tableau algorithms by allowing them to look for tree-like models [BS01]. Therefore, in the presence of *nominals*, existing DL reasoning algorithms look for forest-like

---

[1]http://www.w3.org/TR/2004/REC-owl-guide-20040210/wine.rdf

models, which consist of trees rooted with arbitrarily interconnected *nominal nodes* (as shown in Figure 11).

In practice, the poor performance of tableau algorithms is due to non-determinism in the expansion rules, which results in search of different possible expansions of the completion graph. It was shown in Section 2.2.2 that a tableau algorithm handling *nominals* and QCRs is extended with at-least two non-deterministic rules: the *choose*-Rule and the $\leq$-Rule. This section illustrates the effect of non-determinism introduced by QCRs and *nominals*.

3.1.1 The Semantics of Nominals, QCRs, and Numbers

QCRs carry explicit numerical restrictions; they set an upper (lower) bound on the number of individuals related via a certain role. On the other hand, *nominals* carry implicit numerical restrictions; they not only name individuals but also count them. For example, the concept definition $\mathsf{BloodType} \equiv \{\mathsf{o^+, A^+, B^+, AB^+, o^-, A^-, B^-, AB^-}\}$, where $\{\mathsf{o^+, A^+, B^+, AB^+, o^-, A^-, B^-, AB^-}\}$ are all *nominals*, means that instances of $\mathsf{BloodType}$ can only be one of the 8 enumerated blood types. This additional information carried with *nominals* interacts with QCRs in a way that can limit the number of instances of a certain concept or fillers for a certain role. Given an individual s, instance of a concept E ($s \in E^{\mathcal{I}}$); $C \in N_{\mathrm{C}}$; $R \in N_{\mathrm{R}}$; $o_1, \ldots, o_n \in N_{\mathrm{o}}$; and n, m non-negative integers, one can distinguish between *local* and *global* numerical restrictions.

3.1.1.1 Local Restrictions

When E is of the form $(\geq nR)$, or $(\leq mR)$, E holds a numerical restriction on the cardinality of the set of R-fillers of s. For example, $s \in (\geq 2R)^{\mathcal{I}}$ imposes that at least 2 individuals, s_1 and s_2, must be R-fillers of s, and therefore the cardinality of the

set of R-fillers of s satisfies $\#FIL(R, s) \geq 2$. When E is of the form $(\forall R.C)$, E holds a numerical restriction; an upper bound on the number of R-fillers of s due to the following:

$$s \in (\forall R.C)^{\mathcal{I}} \qquad \Leftrightarrow \quad s \in (\leq 0R.\neg C)^{\mathcal{I}}$$

$$s \in (\forall R.\{o_1, \ldots, o_n\})^{\mathcal{I}} \quad \Rightarrow \quad s \in (\leq nR)^{\mathcal{I}}$$

These restrictions are local since they only affect the set of individuals that are R-fillers of s, $FIL(R, s)$. For example, having $\mathsf{Joseph} \in (\mathsf{Person})^{\mathcal{I}}$ with Person defined in axiom (8) imposes that at least 1 individual, s_1, must be a $\mathsf{hasBloodType}$-filler of Joseph, and therefore $\#FIL(\mathsf{hasBloodType}, \mathsf{Joseph}) \geq 1$. Also, due to the restriction implied by the definition of $\mathsf{BloodType}$, $\#FIL(\mathsf{hasBloodType}, \mathsf{Joseph}) \leq 8$.

$$\mathsf{Person} \sqsubseteq \; \geq 1\mathsf{hasBloodType}.\mathsf{BloodType} \tag{8}$$

3.1.1.2   Global Restrictions

When E is of the form $E \sqsubseteq \{o_1, \ldots, o_n\}$ ($\{o_1, \ldots, o_n\} \sqsubseteq E$), the *nominals*, $o_1, \ldots, o_n$, enforce a numerical restriction on the cardinality of the set of instances of E; there can be at-most (at-least, assuming $o_1, \ldots, o_n$ are all disjoint) n instances of E corresponding to the interpretation of $o_1^{\mathcal{I}} \cup \cdots \cup o_n^{\mathcal{I}}$. Such at-most (at-least) restrictions carried with *nominals* are global since they can affect the set of all individuals in the domain of interpretation ($\Delta^{\mathcal{I}}$). Nominals can specify *concept cardinalities* (See section 2.1.2.1 for a definition of *concept cardinalities*) as was shown in [Tob00]. For example, the concept $\mathsf{BloodType}$ has exactly 8 instances. These implied *concept cardinalities* can interact with local restrictions; having additionally $s \in (\forall R.E)^{\mathcal{I}}$ means that the set of R-fillers of s is bounded by n, $FIL(R, s) \leq n$ ($FIL(R, s) \geq n$).

3.1.2 Non-Determinisim with QCRs, Nominals, and their Interaction

This section illustrates the non-determinism introduced by tableau expansion rules while satisfying the semantics of QCRs and *nominals.*

3.1.2.1 Non-Determinism with QCRs

QCRs introduce non-determinism in choosing a distribution (*choose*-Rule) for each role-successor created to satisfy at-least restrictions as well as an at-most restriction, and when merging those role-successors is necessary to satisfy at-most restrictions ($\leq$-Rule). Such non-determinism can be aggravated when the number of nodes created in a completion graph increases (either due to a large number of at-least restrictions $\geq nR.C$, or due to large numbers (n) used in these restrictions), possibly the number of rules applied to these nodes increases and some of these rules might also be non-deterministic. Moreover, in cases where an at-most restriction is violated, the non-determinism introduced when merging two nodes can cause a blow up of the search space especially when the qualification used with the at-most restriction also contains a disjunction ($\leq nR.(C \sqcup D)$). The following examples illustrates the effect of non-determinism with QCRs.

Example 3.1.1 (Non-Determinism Due to The *choose*-Rule) When testing the satisfiability of ($\geq nR.D \sqcap \leq mR.C$), a standard tableau algorithm, as described in Section 2.2.2, starts with a node x_0 such that $\mathcal{L}(x_0) = \{(\geq nR.D \sqcap \leq mR.C)\}$. After applying the $\sqcap$-Rule to x_0, the label is extended to $\mathcal{L}(x_0) = \{(\geq nR.D \sqcap \leq mR.C), \geq nR.D, \leq mR.C\}$. Applying the $\geq$-Rule, creates n distinct R-successors, $y_1, \ldots, y_n$, of x_o such that $\mathcal{L}(y_i) = \{D\}$ for each $1 \leq i \leq n$. The *choose*-Rule non-deterministically assigns C, or $\neg C$ to the label of each R-successor of x_0.

$$\mathcal{L}(x_0) = \{(\geq nR.D \sqcap \leq mR.C), \geq nR.D, \leq mR.C\}$$
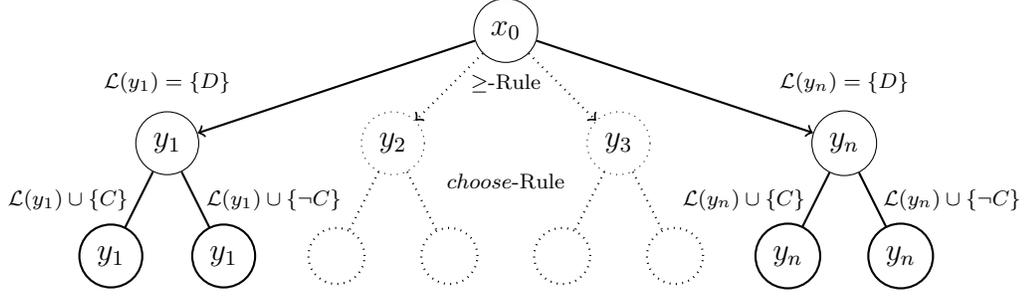
≥-Rule: create n R-successors of x_0 such that $\mathcal{L}(\langle x_0, y_i\rangle) = \{R\}$ for $1 \leq i \leq n$.

choose-Rule: extend the label of each R-successor of x_0 such that $\mathcal{L}(y_i) = \mathcal{L}(y_i) \cup E \in \{C, \neg C\}$.

Figure 7: Completion graph showing one source of non-determinism: the *choose*-Rule.
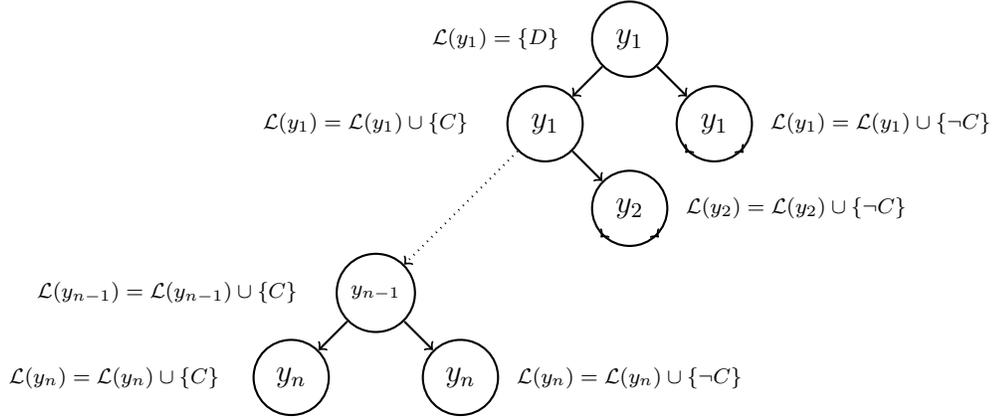
Figure 8: Completion graph expansion tree due to the *choose*-Rule.

The completion graph can therefore be expanded in 2^n different ways, as shown in Figure 8, based on the distribution of these R-successors. Having more than one at-most restriction, also affects the expansion such that if there are q at most restrictions, then the total number of branches becomes equal to $2^{n \times q}$.

Example 3.1.2 (Non-determinism due to the *choose*-Rule and the ≤-Rule)
This example shows the effect of non-determinism due to the ≤-Rule when the number

of R-successors, created to satisfy at-least restrictions ($\geq nR.D$, $\geq pR.E$), exceeds the number allowed by an at-most restriction ($\leq mR.C$), $(n + p) > m$. As shown in Figure 9, the $\geq$-Rule creates $(n + p)$ R-successors of x_0 such that $y_1, \ldots y_n$ are mutually disjoint and $z_1, \ldots, z_p$ are mutually disjoint. This means y-nodes cannot be merged together, and z-nodes cannot be merged together either (to make sure the $\geq$ is always satisfied and avoid a create and merge cycle known as a "yo-yo" effect). The *choose*-Rule non-deterministically expands the completion graph by creating two branching points in the search space for each R-successor of x_0. Having $(n + p)$ R-successors, means that the search space is expanded with 2^{n+p} branches. Since $(n + p) > m$, the $\leq$-Rule non-deterministically merges a y_i node such that $C \in \mathcal{L}(y_i)$ with a z_j node such that $C \in \mathcal{L}(z_i)$ until $(n + p) \leq m$. Assuming that the exceeding number of R-successors is given as $k = \mid m - (n + p) \mid$, this means that k y_i nodes need to be merged with k z_j nodes. The number of ways that the n R-successors $(y_1, \ldots, y_n)$ can be grouped into k R-successor is defined as $K_y = \frac{(n!)}{(n-k)!}$ and the number of ways that the p R-successors, $z_1, \ldots, z_p$, can be grouped into k elements is defined as $K_z = \frac{(p!)}{(p-k)!}$. This means that the number of ways to merge k y_i nodes with k z_j nodes is given as $K_{yz} = \frac{(n!)}{(n-k)!)} \times \frac{(p!)}{(p-k)!}$. The nodes that can be merged are highlighted. Due to non-deterministic rules, the completion graph is expanded in 2^{n+p} ways (*choose*-Rule), and K_{yz} ways ($\leq$-Rule). Note also that having more at-least restrictions, not only affects the expansion of the completion graphs by introducing more expansions due to the *choose*-Rule, but also introduces more possible ways to merge excess R-successors.

$$\mathcal{L}(x_0) = \{\geq nR.D, \geq pR.E, \leq mR.C\}$$
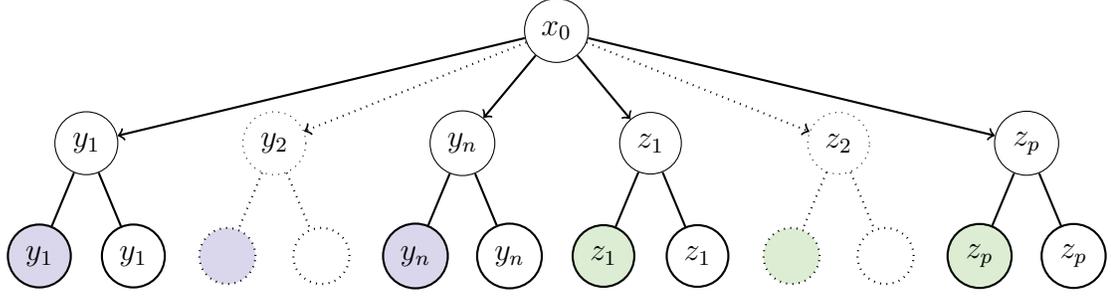
≥-Rule: create n R-successors of x_0 such that $\mathcal{L}(\langle x_0, y_i \rangle) = \{R\}$ for $1 \leq i \leq n$.
≥-Rule: create p R-successors of x_0 such that $\mathcal{L}(\langle x_0, z_j \rangle) = \{R\}$ for $1 \leq j \leq p$.
choose-Rule: extend the label of each R-successor of x_0 with $E \in \{C, \neg C\}$.
≤-Rule: non-deterministically merge y_i with z_j until $\leq mR.C$ is satisfied.

Figure 9: Completion graph expansion showing two sources of non-determinism: the *choose*-Rule and the ≤-Rule.

Example 3.1.3 (Non-determinism due to the *choose*-Rule, ⊔-Rule, and ≤-Rule) This example shows a source of non-determinism when handling QCRs using disjunctive descriptions to qualify R-successors such as in the following concept description: $(\geq nR.D \sqcap \, \geq pR.E \sqcap \, \leq mR.(A \sqcup B))$. As illustrated in Figure 10, in addition to the sources of non-determinism illustrated in Examples 3.1.1 and 3.1.2, the ⊔-Rule introduces non-deterministic expansions of the labels of R-successors of x_o and doubles the number of nodes to be considered by the ≤-Rule.

$$\mathcal{L}(x_0) = \{\geq nR.D, \geq pR.E, \leq mR.(A \sqcup B)\}$$
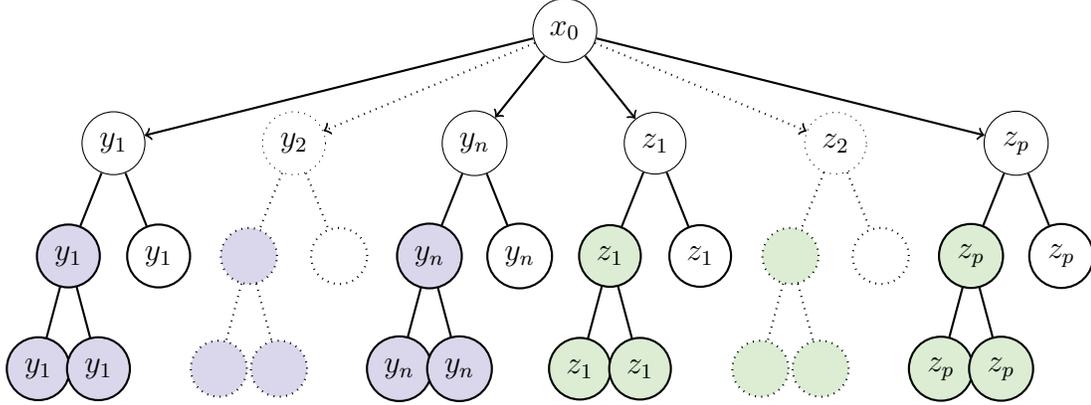
choose-Rule: extend the label of each R-successor of x_0 with $E \in \{(A \sqcup B), (\neg A \sqcap \neg B)\}$
$\sqcup$-Rule: extend the label of each R-successor of x_0, having $A \sqcup B$ in its label, with $E \in \{A, B\}$
$\leq$-Rule: non-deterministically merge y_i with z_j until $\leq mR.(A \sqcup B)$ is satisfied

Figure 10: Completion graph expansion showing three sources of non-determinism: the *choose*-Rule, the $\sqcup$-Rule, and the $\leq$-Rule.

3.1.2.2 Non-Determinism with Nominals

Due to the equivalence between the definition of BloodType in axiom (9), using an enumeration of *nominals*, and that in axiom (10), using a disjunction of *nominals*, *nominals* introduce non-determinism when used with the *oneOf* constructor.

$$\mathsf{BloodType} \equiv \{\mathsf{o}^+, \mathsf{A}^+, \mathsf{B}^+, \mathsf{AB}^+, \mathsf{o}^-, \mathsf{A}^-, \mathsf{B}^-, \mathsf{AB}^-\} \quad (9)$$

$$\mathsf{BloodType} \equiv \{\mathsf{o}^+\} \sqcup \{\mathsf{A}^+\} \sqcup \{\mathsf{B}^+\} \sqcup \{\mathsf{AB}^+\} \sqcup \{\mathsf{o}^-\} \sqcup \{\mathsf{A}^-\} \sqcup \{\mathsf{B}^-\} \sqcup \{\mathsf{AB}^-\} \quad (10)$$

$\mathcal{L}(x_1) = \{o^+\}$ $\{A^+\}$ $\{B^+\}$ $\{AB^+\}$ $\{o^-\}$ $\{A^-\}$ $\{B^-\}$ $\mathcal{L}(x_8) = \{AB^-\}$

$\mathcal{L}(x_0) = \{\geq 1 hasBloodType.BloodType\}$

$\geq$-Rule

$\sqcup$-Rule

$\mathcal{L}(y_1) = \{o^+\}$ $\mathcal{L}(y_8) = \{AB^-\}$

$\geq$-Rule: create hasBloodType-successor, y, of x_0 that $\mathcal{L}(y) = \{\{o^+\} \sqcup \{A^+\} \cdots \sqcup \{AB^-\}\}$
$\sqcup$-Rule: extends the label of each hasBloodType-successor of x_0 using $\{o^+\} \sqcup \{A^+\} \cdots \sqcup \{AB^-\}$
O-Rule: merge two nodes having the same nominal in their label.
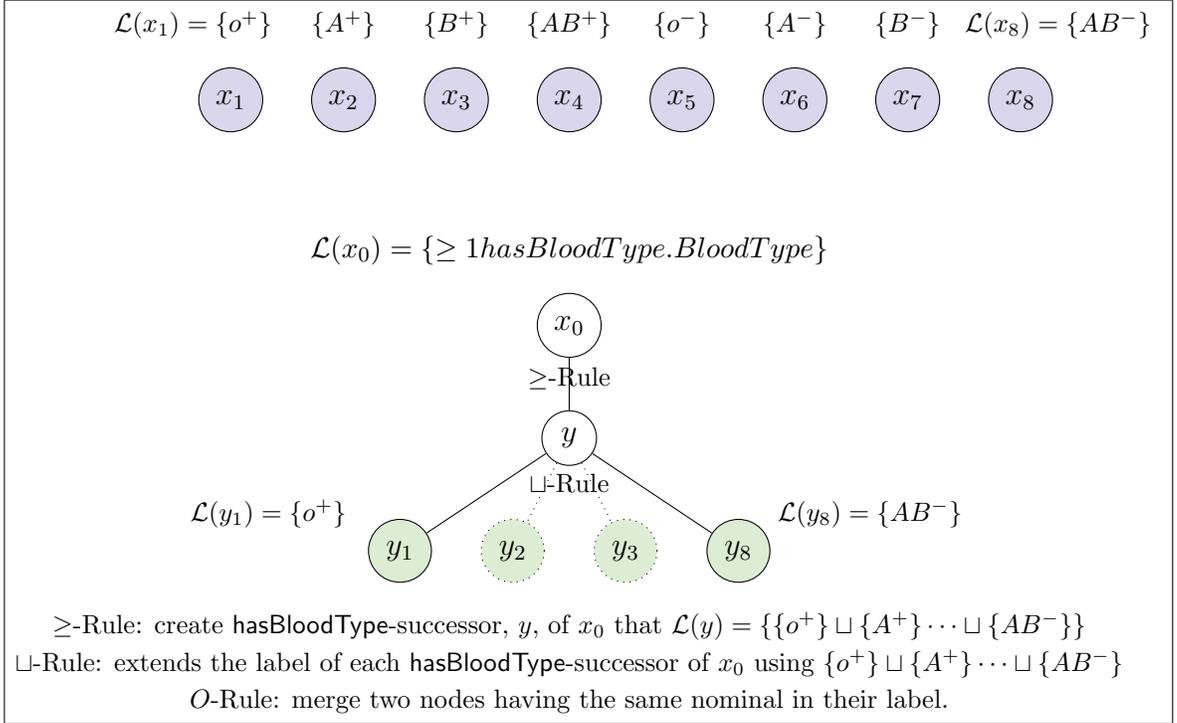
(a) Completion graph expansion showing non-determinism due to *nominals* in QCRs.

$\mathcal{L}(x_1) = \{BloodType, o^+\}$ $\{A^+\}$ $\{B^+\}$ $\{AB^+\}$ $\{o^-\}$ $\{A^-\}$ $\{B^-\}$ $\mathcal{L}(x_8) = \{AB^-\}$

$\{hasBloodType\}$

$\mathcal{L}(x_0) = \{\geq 1 hasBloodType.BloodType\}$
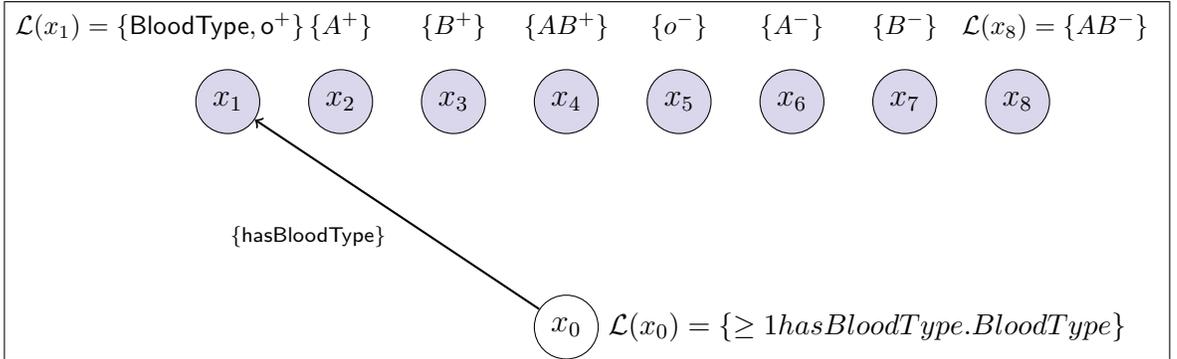
(b) Completion model showing the interaction between *nominals* and QCRs.

Figure 11: Non deterministic interaction between *nominals* and QCRs.

This non-determinism can interact with the non-determinism introduced with QCRs in cases where *nominals* are used to qualify role-successors. For example, the description $\leq 1hasBloodType.BloodType$ specifies that one can have at-most one BloodType such that the hasBloodType-filler must be identified with one of the *nominals* enumerating blood types, see Figure 11.

49

3.2 Optimized Reasoning with Nominals and QCRs

There do not exist many optimization techniques that address *nominals* or QCRs. First performance improvements for tableau-based DL systems handling QCRs have been reported in [HM01a, HTM01] and more recently in [FFHM08b]. Optimizations for tableau-based reasoning with *nominals* were first studied in [PCS06] as simple extension/modification of existing optimizations [FB07b] like *absorption*, *pseudo-model merging* and *caching*. Resolution-based reasoning procedures were proposed in [KM06] and were proven to be weak in dealing with large numbers in QCRs. Hyper-tableau [MSH07] which combines tableau and resolution-based [KM06] reasoning were recently studied to minimize the size of the models created and their degree of non-determinism in DL reasoning with no special treatment for QCRs or *nominals*. This section provides a review of the tableau-based optimization techniques investigated to handle *nominals*, and QCRs. Those technique have been aimed at the non-determinism due to the handling of *nominals* and disjunctions of concepts during a preprocessing phase, a satisfiability test phase, or a subsumption test phase. In Section 3.3, the non-tableau methods used to enhance reasoning with QCRs [HTM01] are reviewed.

3.2.0.3 Preprocessing Optimizations

Preprocessing techniques are performed directly on the syntax of the input to render it more amenable to reasoning and processing. These techniques examine the syntactic structure of input concept expressions and exploit relations (tautology, clash) which are obvious, and can significantly speed up subsequent reasoning. Some of the widely

used preprocessing techniques are *lazy unfolding*,[2] *internalization*,[3] and *absorption* [HT00].

Absorption aims at reducing (if not eliminating) GCIs occurring in a TBox due to the high degree of non-determinism that they introduce. Every GCI of the form $C \sqsubseteq D$ is converted to a disjunction $\neg C \sqcup D$ that is added to every node in the completion graph during an inference service. The search space grows exponentially with the size of GCIs available in a TBox. A standard absorption technique is a rewriting technique aiming at absorbing CGIs into definition axioms of the form $A \sqsubseteq C$, with A a concept name, and C a concept expression, using the following equivalences:

$$C \sqcap D \sqsubseteq E \;\equiv\; C \sqsubseteq \neg D \sqcup E \tag{11}$$

$$C \sqsubseteq D \sqcap E \;\equiv\; ((C \sqsubseteq D) \sqcap (C \sqsubseteq E)) \tag{12}$$

Absorption reduces the effect of disjunctions and makes sure they are only applicable to individuals that are already known to be instances of the appropriate concept. It is worth noting that some axioms in a TBox $\mathcal{T}$ cannot be absorbed, and they are treated via *internalization* or by adding a disjunction to every node in the completion graph. Although absorption can dramatically improve the performance of a DL reasoner, it is known to be a non-deterministic technique since there might be more than one way to absorb a GCI. Finding the best way to absorb a GCI is an open problem and is subject to many research activities [Zuo06, HW06, Wu08].

---

[2]Unfolding is a preprocessing technique which aims at reducing concept inclusion axioms by replacing each occurrence of an atomic concept with its corresponding definition. An example was shown in Definition 2.1.5.

[3]Internalization is a preprocessing technique which encapsulates all relevant information encoded in a TBox $\mathcal{T}$ into a single concept $C_{\mathcal{T}}$ (as shown in Section 2.1.3) which is added to every newly added node in the completion graph.

Nominal Absorption When *nominals* appear in GCIs, standard absorption techniques can no longer be used. Two types of absorption techniques have been considered to absorb GCIs with *nominals* depending on whether the *oneOf* or the *hasValue* constructor (introduced in Section 2.1.2.1) is used. The *OneOf* and the *HasValue* absorption techniques, the first being applicable to GCIs using the *oneOf* constructor, the second being applicable to GCIs using the *hasValue* constructor, try to absorb *nominal*-based disjunctions in order to minimize their undesirable effects.

GCI with oneOf:	GCI with oneOf :
WineColor ≡ {red, rose, white}	WineColor ≡ {red, rose, white}
Standard Absorption:	OneOf Absorption:
¬WineColor ⊔ {red} ⊔ {rose} ⊔ {white} WineColor ⊔ ¬({red} ⊔ {rose} ⊔ {white})	WineColor ⊑ {red} ⊔ {rose} ⊔ {white} in $\mathcal{T}$ (red : WineColor) in $\mathcal{A}$ (rose : WineColor) in $\mathcal{A}$ (white : WineColor) in $\mathcal{A}$

(a) Applying standard absorption.　　　　(b) Applying the OneOf absorption.

Figure 12: Absorption with GCI using the *oneOf* constructor.

The OneOf Absorption The axiom (WineColor ≡ {red, rose, white}) is equivalent to (WineColor ⊑ {red, rose, white}) and ({red, rose, white} ⊑ WineColor). Standard absorption techniques cannot capture this axiom, which results in adding the expressions shown in Figure 12a to every node in the completion graph expansion yielding a great number of backtracking points being added to the search space. The *OneOf Absorption*, was introduced in [Sir06] to absorb this type of GCIs into primitive definitions

as shown in Table 12b using the following equivalence:[4]

$$C \equiv \{a_1, a_2, \ldots, a_n\} \quad \Leftrightarrow \quad C \sqsubseteq \{a_1, a_2, \ldots, a_n\}, \text{and}, (a_1 : C), \ldots, (a_n : C) \quad (13)$$

The HasValue Absorption   Applying standard absorption to GCIs using the *has-Value* constructor can result in a big number of backtracking points being added to the search space. For example, the number of choice points can grow significantly if there is a big number of instances of the Wine concept, given the standard absorption (using equivalence (11)) of Riesling definition axiom as shown in Figure 13a. Due to the equivalence in (11), we also have $\exists$madeFrom.{RieslingGrape} $\sqsubseteq \neg$Wine$\sqcup \geq$ 2madeFrom. The *HasValue Absorption*, introduced in [Sir06], absorbs the definition axiom of Riesling into the ABox assertions in Figure 13b. This can be done using the semantics of *nominals* and the following equivalence[5]:

$$\exists R.\{o\} \sqsubseteq C \quad \Leftrightarrow \quad \{o\} \sqsubseteq \forall R^-.C \quad (14)$$

Notice that the same number of disjuncts is introduced. However with the *Has-Value* absorption, the effect of the disjuncts is localized to instances of Riesling concept instead of instances of the Wine concept . This can work well if the number of instances of Riesling is considerably less than the ones of Wine. The effect of this technique is problem dependent, even if disjunction is localized to a different concept, this does not necessarily mean that the disjunction will be applied to a small number of individuals. Also the use of *inverse roles* might render this optimization useless - if the logical language at hand does not enable *inverse roles* - or less efficient by rendering the reasoning task more complex if no optimization techniques were adopted to deal with *inverse roles*.

---

[4]See [Sir06] for a proof.

[5]This equivalence is a special variant of $\exists R.C \sqsubseteq D \Leftrightarrow C \sqsubseteq \forall R^-.D$ which is proved in [Din08].

GCI with hasValue:
Riesling $\equiv$ Wine $\sqcap$ $\leq$ 1madeFrom $\sqcap$ $\exists$madeFrom.{RieslingGrape}
Standard Absorption:
Wine $\sqsubseteq$ Riesling $\sqcup$ $\forall$madeFrom.$\neg${RieslingGrape}$\sqcup$ $\geq$ 2madeFrom

(a) Applying standard absorption.

GCI with hasValue:
Riesling $\equiv$ Wine $\sqcap$ $\leq$ 1madeFrom $\sqcap$ $\exists$madeFrom.{RieslingGrape}
HasValue Absorption:
({RieslingGrape} : $\forall$madeFrom$^-$.(Riesling $\sqcup$ $\neg$Wine$\sqcup$ $\geq$ 2madeFrom))

(b) Applying *hasValue* absorption.

Figure 13: Absorption with a GCI using the *hasValue* constructor.

3.2.1 Satisfiability Optimizations

A satisfiability test is usually the core test of most inference services. Satisfiability optimizations aim at enhancing the order in which to apply tableau expansion rules, and the order in which to investigate possible expansions of non-deterministic rules. In terms of applying an order of rule application, expansion rules that create new nodes in the completion graph (e.g., $\exists$-Rule, $\geq$-Rule) are usually assigned the lowest priority. In practice, it was shown that such ordering of rule applications can have an effect on performance. In terms of ordering non-deterministic expansions for a concept expression that includes a disjunction $(C_1 \sqcup C_2 \sqcup C_3 \ldots)$, standard tableau algorithms use a technique known as *syntactic branching*: it allows the algorithm to non-deterministically choose an unexpanded disjunction $(C_1 \sqcup C_2 \sqcup C_3 \ldots)$ in the label $(\mathcal{L}(x))$ of a node (x) and add each of the disjuncts in $(C_1 \sqcup C_2 \sqcup C_3 \ldots)$ to $\mathcal{L}(x)$. In some case, the algorithm might need to explore different completion graphs

corresponding to different disjuncts (maybe all) before the test terminates. Additionally, completion graphs corresponding to each of $C_1 \sqcup C_2 \sqcup C_3 \dots$ are not disjoint and non-deterministically exploring them can lead to the recurrence of an unsatisfiable disjunct in more than one graph which renders the whole algorithm inefficient. *Semantic branching*, *boolean constraint propagation*, *dependency directed backtracking*, *heuristic guided search*,[6] and *caching* [TH05] are some of the optimizations designed and implemented to handle this source of inefficiency.

3.2.1.1   Semantic Branching

As shown in Figure 14, instead of choosing an unexpanded disjunction, *semantic branching* chooses a single unexpanded disjunct, C, from $\mathcal{L}(x)$ and explores the two models for C and $\neg C$ (added to $\mathcal{L}(x)$). The two models are disjoint and recurrence is avoided as shown in Example 3.2.1, which also shows that, compared to *syntactic branching*, *semantic branching* can have a dramatic effect in pruning the search space.

Example 3.2.1   Semantic Branching vs Syntactic Branching When testing the satisfiability of $(C \sqcup D_1) \sqcap (C \sqcup D_2)$ with C an unsatisfiable expression, semantic branching allows a more reduced search space than semantic branching as shown in Figure 14.

---

[6]*Heuristic guided search* can be used to guide the choice of the disjuncts in order to minimize the search space.

$$\mathcal{L}(x_0) = \{(C \sqcup D_1), (C \sqcup D_2)\}$$

$$\mathcal{L}(x_0) = \mathcal{L}(x_0) \cup \{D_1\}$$

$$\mathcal{L}(x_0) = \mathcal{L}(x_0) \cup \{D_2\}$$

Clash $\Leftarrow \mathcal{L}(x_0) = \mathcal{L}(x_0) \cup \{C\}$

Clash $\Leftarrow \mathcal{L}(x_0) = \mathcal{L}(x_0) \cup \{C\}$

(a) Syntactic Branching

$$\mathcal{L}(x_0) = \{(C \sqcup D_1), (C \sqcup D_2)\}$$

$$\mathcal{L}(x_0) = \mathcal{L}(x_0) \cup \{C\} \Rightarrow \text{Clash}$$

$$\mathcal{L}(x_0) = \mathcal{L}(x_0) \cup \{\neg C, D_1, D_2\}$$

(b) Semantic Branching
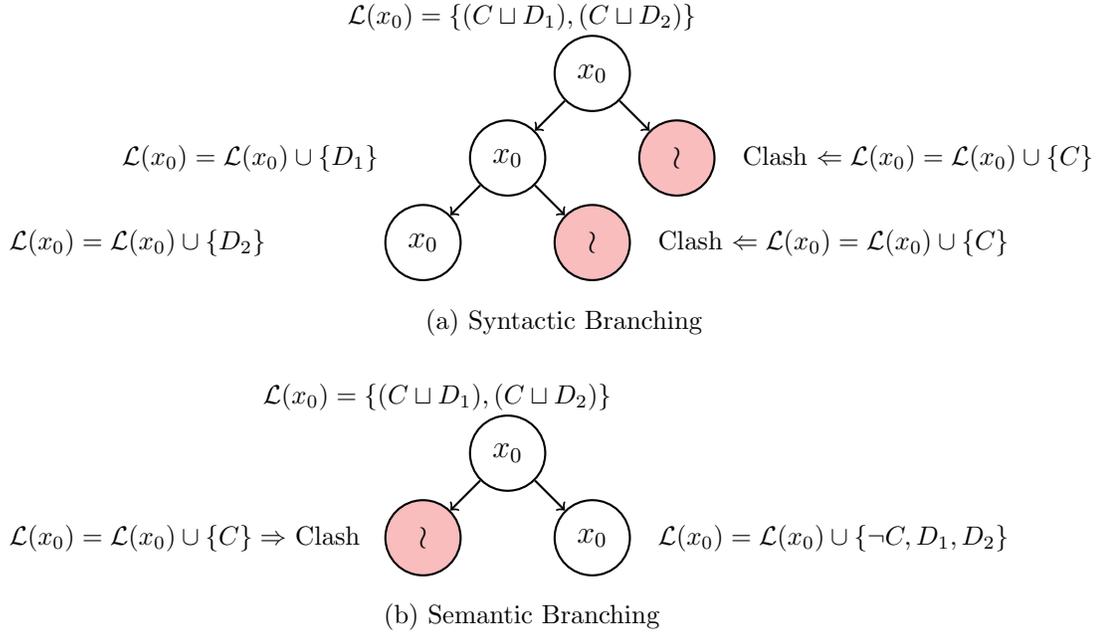
Figure 14: *Syntactic branching* versus *semantic branching* during the satisfiability test of $(C \sqcup D_1) \sqcap (C \sqcup D_2)$.

3.2.1.2   Boolean Constraint Propagation

Boolean Constraint Propagation (BCP), also known as *simplification*, works by examining disjunctions and simplifying them where possible so that it can later deterministically expand single disjunctions in $\mathcal{L}(x)$. This technique can greatly reduce the search space especially when used with *semantic branching* [FB07b]. It can be used with a wide range of DL languages without increasing the size of the search space.

3.2.1.3   Dependency Directed Backtracking

Sometimes large amounts of unproductive backtracking search is caused by inherent unsatisfiability encapsulated in sub-problems, a problem known as *thrashing* (as shown in Figure 15a).

(a) Thrashing

(b) Back-jumping
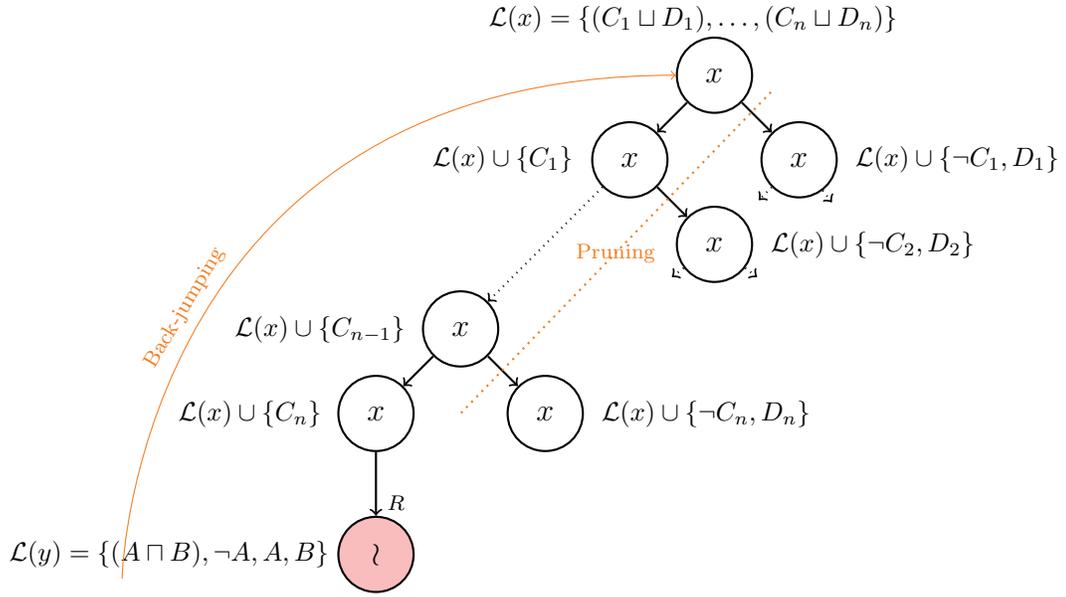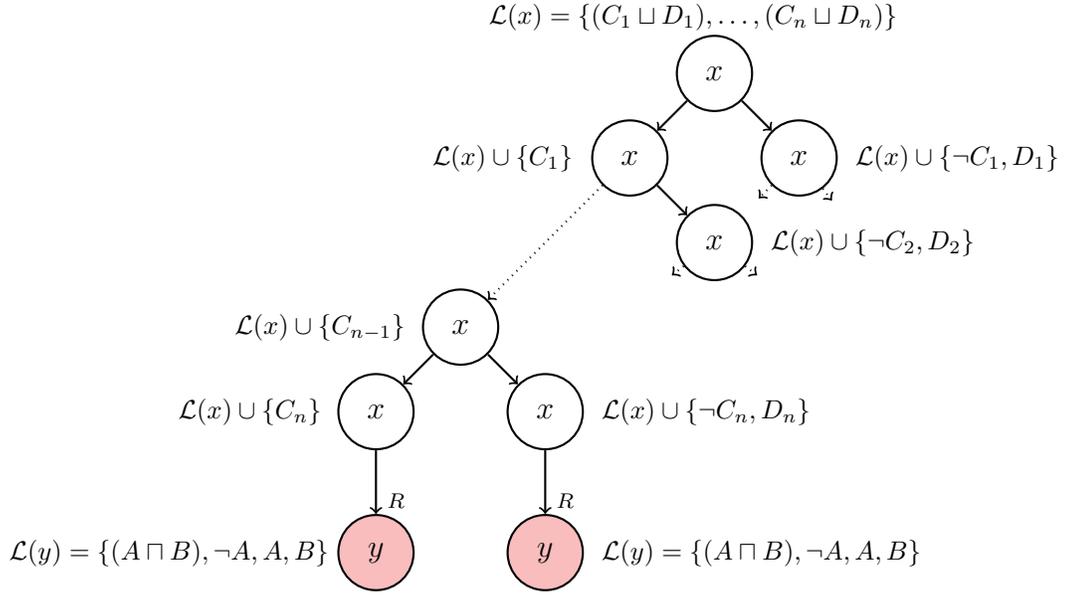
Figure 15: Effect of Dependency Directed Backtracking.

Dependency directed backtracking addresses this problem by labelling concepts with a dependency set indicating the non-deterministic expansion choices on which they depend. When a clash is discovered, the dependency sets of the clashing concept

s can be used to identify the most recent non-deterministic expansion where an alternative choice might alleviate the cause of the clash. The algorithm can then jump back over intervening non-deterministic expansions without exploring any alternative choices (as shown in Example 3.2.2). This technique is also known as *backjumping* and it is used in solving Constraint Satisfiability Problems (CSP) [Bak95] as well as in solving satisfiability problems in DL [Hor97].

Example 3.2.2   Resolving Thrashing Using Backjumping Let $\mathcal{L}(x) = \{(C_1 \sqcup D_1), ..., (Cn \sqcup Dn), \exists R.(A \sqcap B), \forall R \neg A\}$. Figure 15a shows how the problem of thrashing can occur in backtracking, and Figure 15b shows how back-jumping resolves thrashing.

3.2.1.4   Caching

When role-successor nodes are created, many of these nodes have common labels, particularly due to the application of the $\forall$-Rule. The repeated satisfiability checks of these common labels can be avoided using *caching* [FB07b, HT99]. Creating role-successor nodes is delayed until all other expansion possibilities are exhausted and the set of concept expressions that constitute the label of each successor is computed. If two successors are found to have the same label, then they will have the same satisfiability test, which is computed once with the status result saved and applied not just to both nodes, but to any of their successor nodes having the same label. *Caching* can interact with *backjumping* mainly because two nodes (especially those that were not expanded) that have the same concept labels may not necessarily have the same dependency sets. When *caching* is used with an unsatisfiable role-successor x, a weak form of *backjumping* can be used by computing x's dependency set as the

union of the dependency sets of the concept names in $\mathcal{L}(x)$. Caching can be highly effective with problems of repeated structure. However, it causes storage overhead and can sometimes interact with *backjumping* and degrade performance. It is also logic (supported DL constructors) and problem dependent.

3.2.1.5 Forest Caching

In the presence of *nominals*, *caching* the satisfiability status of a node cannot be directly used since ABox assertions can affect concept satisfiability; *nominals* can be referred to across different nodes of the completion graph and new concepts may propagate to a previously cached node. As was shown in Figure 11, a *nominal node* (for every nominal) is added to the initial completion graph when testing the satisfiability of a concept C relying on *nominals*. The forest-like shape of the completion graph can result in a large number of completion rules to be triggered. The *forest caching* technique [PCS06, Sir06] is a *caching* technique used in the presence of *nominals* to save the state of the completion graph after an initial consistency check. The cached state is used as an initial completion graph for subsequent concept satisfiability tests. This technique avoids repeating the process of expanding *nominal nodes* from their initialization state which may involve a large number of expansion rules. However, in order to ensure correctness of the technique, the whole status of the expansion needs to be saved including non deterministic choices that remain to be explored and dependency sets for nodes and edges' labels. Saving the whole status, affects memory consumption.

3.2.1.6 Lazy Forest Generation

Another optimization technique used to reduce the overhead of a large number of expansion rules, triggered by the use of a forest, is to include *nominal nodes* in the

initial completion graph only if the *nominal rule* (*O*-Rule) is applied when checking concept satisfiability. Because unless a *nominal rule* is applicable, the satisfiability of a concept does not depend on the individuals in the ABox. The combination of *lazy forest generation* [PCS06, Sir06] and *caching* may interact *with dependency-directed backtracking* and, in order to ensure the correctness of the technique, the initial set of *nominal nodes* is generated whenever *back-jumping* is applied, even if the *nominal rule* is not applicable.

3.2.1.7   Using The Signature Calculus

When the number of role-successors introduced in a completion graph becomes large, the non-determinism in merging these role-successors in order to preserve the satisfiability of at-least and at-most restrictions (as shown in Examples (3.1.1-3.1.3) can possibly cause a combinatorial explosion. The signature calculus is introduced in [HM01a] to handle such inefficiency for DL handling the expressivity of $\mathcal{ALCQH}$. The signature calculus generates a so-called *proxy role-successor* node for each $\geq nR.C$. The proxy node represents the n R-successors sharing a common restriction, C called signature. On the other hand, a *proxy role-successor* x sometimes need to be split into more than one (case when a new signature extends the one represented by x for some of the role-successors represented by x), or merged with a proxy role successor y (case when x and y violate an at-most restriction) in cases where the restrictions on role-successors are not satisfied.

3.3   The Algebraïc Method

If one knows that a person has two sons and three daughters, one can easily deduce that this person has five children. So far, DL reasoning remains blind with such reasoning about numbers; this is because the reasoning procedures treat numerical

restrictions implied by concept definitions using expansion rules that construct completion models by searching case by case until numerical restrictions are satisfied. Such blindness to numbers results in highly non-deterministic handling of numerical restrictions implied by QCRs and *nominals*, as was shown in Sections 3.1.2.1 and 3.1.2.2. For example, in order for the entailment in (15) to hold, a DL reasoning algorithm would need to know about all restrictions and relationships between the concept definitions of Son, Daughter, Child, and the roles hasSon, hasDaughter, and hasChild.

$$\geq 2\mathsf{hasSon} \sqcap \geq 3\mathsf{hasDaughter} \models \geq 5\mathsf{hasChild} \tag{15}$$

None of the optimizations presented in the previous sections aims directly at handling non-determinism caused by numerical restrictions either through merging excess role-successors ($\leq$-Rule), choosing a distribution of role-successors based on their qualifications (*choose*-Rule), or when merging and distributing role-successors interacts with a non-deterministic distribution of *nominals*. First efforts [HTM01] to efficiently deal with the numerical restrictions implied by QCRs were based on combining tableau-based algorithms with algebraïc reasoning such that concept satisfiability is reduced to pure linear in-equation solving. The approach demonstrated the performance gain using algebraïc reasoning for the DL $\mathcal{SHQ}$. However, the approach in [HTM01] was based on a recursive calculus and no proofs for soundness, completeness, and termination were given.

Algebraïc reasoning for set description languages including DL was first introduced in [OK96] and later in [OK99] where it was investigated how a concept satisfiability check can be reduced to a pure in-equation solving problem. The DL operators discussed handle only the expressiveness of $\mathcal{ALCQ}$ ($\mathcal{ALC}$ extended with QCRs) and neither *nominals* nor GCIs were taken into account, with no formal calculus for the

approach. The basic idea is that numerical features of the concept sets (cardinality information) are turned into arithmetic terms and put into linear in-equations which are easily handled by arithmetic equation solvers, and entailments like the one (15) are handled efficiently. The results from [HTM01] have lately been sharpened in [Far08, FH10c] where a decidable hybrid calculus for $\mathcal{SHQ}$, based on the hybrid decision procedure for the DL $\mathcal{ALCQ}$ [FFHM08a], is proposed along with a practical implementation showing that algebraïc reasoning can dramatically improve performance with QCRs [Far08, FH10c]. In [Din08] the algebraïc reasoning is combined with tableau reasoning to achieve an improved worst-case upper bound for deciding the satisfiability of $\mathcal{ALCFI}$ concepts.

The hybrid approach presented in this thesis is mainly inspired by, and based on the atomic decomposition technique, which is illustrated in this section.

3.4   Atomic Decomposition

The *atomic decomposition* technique allows encoding the numerical restrictions on concepts and role fillers into in-equations. The satisfiability of the numerical restrictions can therefore be decided by solving the encoded in-equations. Given a finite set of sets, which is referred to as $\mathcal{D}$, the atomic decomposition considers all possible ways to decompose $\mathcal{D}$ into mutually disjoint atomic sets. These atomic sets are considered the atoms of the Boolean Algebra consisting of the closure of sets under union, intersection, and complement. In the case of the entailment in (15), the atomic decomposition works on the sets of hasSon-fillers, hasDaughter-fillers, and hasChild-fillers represented using arbitrarily overlapping sets respectively as sons, daughters and children shown in Figure 16. As it can be seen, different overlaps result in different mutually disjoint areas representing subsets of $\mathcal{D} = \{\mathsf{Children}, \mathsf{Sons}, \mathsf{Daughters}\}$. Each subset represents the set of role fillers satisfying a concept expression which can

be derived based on set conjunctions and complement operations as follows:

$$C = \text{Children} \cap \neg\text{Sons} \cap \neg\text{Daughters} \qquad CS = \text{Children} \cap \text{Sons} \cap \neg\text{Daughters}$$

$$S = \neg\text{Children} \cap \text{Sons} \cap \neg\text{Daughters} \qquad SD = \neg\text{Children} \cap \text{Sons} \cap \text{Daughters}$$

$$D = \neg\text{Children} \cap \neg\text{Sons} \cap \text{Daughters} \qquad CD = \text{Children} \cap \neg\text{Sons} \cap \text{Daughters}$$

$$CSD = \text{Children} \cap \text{Sons} \cap \text{Daughters}$$
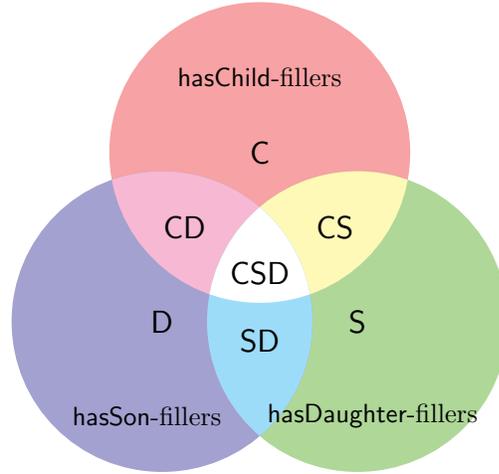
Figure 16: Atomic Decomposition on $\mathcal{D} = \{\text{Children}, \text{Sons}, \text{Daughters}\}$ representing hasChild-fillers, hasSon-fillers, and hasDaughter-fillers.

The corresponding sets can be derived from their decomposed subsets using union operations:

$$\text{Children} = C \cup CS \cup CD \cup CSD$$

$$\text{Sons} = S \cup CS \cup SD \cup CSD$$

$$\text{Daughters} = D \cup CD \cup SD \cup CSD$$

Since the decomposed subsets are all disjoint, and the cardinality function of disjoint sets is additive, one can encode the number restriction expressions in into

arithmetic terms:

$$\geq 2\mathsf{hasSon} \implies \mathsf{S} + \mathsf{CS} + \mathsf{SD} + \mathsf{CSD} \geq 2$$

$$\geq 3\mathsf{hasDaughter} \implies \mathsf{D} + \mathsf{CD} + \mathsf{SD} + \mathsf{CSD} \geq 3$$

$$\leq 4\mathsf{hasChild} \implies \mathsf{C} + \mathsf{CS} + \mathsf{CD} + \mathsf{CSD} \leq 4$$

For ease of presentation, the cardinality function over a set (#) is dropped, and each set name is used to represent the cardinality of the corresponding concept set.

Given a decomposition set $\mathcal{D}$ of size n, the atomic decomposition considers 2^n possibilities of overlaps between the elements of $\mathcal{D}$. However, relations between concepts, such as disjointness and subsumption, can be further exploited and encoded into arithmetic terms allowing a reduced number of atomic subsets.

Concept relation	DL notation	Arithmetic encoding
Sons and Daughters are disjoint	Sons $\sqsubseteq \neg$Daughters	SD $=0$, CSD $=0$
Daughters are Children	Daughters $\sqsubseteq$ Children	D$=0$
Sons are Children	Sons $\sqsubseteq$ Children	S$=0$
Children are either sons or daughters	Children $\sqsubseteq$ Sons $\sqcup$ Daughters	C$=0$

Table 2: Encoding relations between concepts into arithmetic terms.

And the original entailment $\geq 2\mathsf{hasSon} \sqcap \geq 3\mathsf{hasDaughter} \models \geq 5\mathsf{hasChild}$ which is decided by testing the unsatisfiability of $\geq 2\mathsf{hasSon} \sqcap \geq 3\mathsf{hasDaughter} \sqcap \leq 4\mathsf{hasChild}$ can be reduced to solving the following system of linear in-equations:

$$\mathsf{CS} \geq 2 \tag{16}$$

$$\mathsf{CD} \geq 3 \tag{17}$$

$$\mathsf{CS} + \mathsf{CD} \leq 4 \tag{18}$$

64

It is trivial for an in-equation solver to find out that there is no solution and the original problem $\geq 2\mathsf{hasSon} \sqcap \geq 3\mathsf{hasDaughter} \sqcap \leq 4\mathsf{hasChild}$ is unsatisfiable. Thus, the entailment $(\geq 2\mathsf{hasSon} \sqcap \geq 3\mathsf{hasDaughter}) \models \geq 5\mathsf{hasChild}$ holds.

3.5 Discussion and Conclusion

This chapter illustrates the challenge of DL reasoning in the presence of *nominals* and QCRs, as well as the semantic interaction between the two constructors. A review of related optimization techniques is provided, however a complete list and evaluation on the effectiveness of these techniques is beyond the scope of this thesis and can be found in the articles where these techniques were described. The purpose of introducing these technique is to show that they do not aim at a better handling of the interaction between the two constructors, but at the non-determinism caused by this interaction. This is because the reasoning is based on an algorithm blind to the numerical features, and interactions between *nominals* and QCRs. Although better informed calculi have been investigated to handle QCRs [HTM01, HM01a, OK99], no formal calculus with proofs of soundness and completeness is devised, and the extension to handle more expressive constructors including *nominals* is not clear. Also, the optimizations designed to handle the nominal constructor are designed more on the syntactic level, such as the nominal absorption techniques, without taking into consideration their implied numerical semantics.

Nominals are powerful; once they are available, a significant gain in expressivity is added to the language at hand. A lot of existing ontologies rely heavily on the use of *nominals* (Wine ontology for example). The absence of efficient reasoning with nominals, means that a lot of existing ontologies will not have a practical reasoner and not much can be inferred from these ontologies. On the other hand, one

might argue that there is no need to handle large numbers (with QCRs) in ontologies. However, this seems to be a chicken and egg problem; there does not exists a lot of ontologies using large numbers because no available reasoner handles large numbers efficiently. There exists a lot of cases where one needs to use large numbers in QCRs such as specifying that a person has 230 movable and semi movable joints (Person $\sqsubseteq \geq$ 230hasJoint.(MovableJoint $\sqcup$ SemiMovableJoint)) as part of the human skeletal system [MHWZ06] representation. Also due to the implied numerical restrictions imposed by *nominals*, these numbers do not necessarily need to be explicitly represented by QCRs. Thus in order to be a practical DL reasoning component, the full expressivity of OWL2 must be handled including *nominals*. However, no performance improvements have been reported for KBs that rely on the use and interaction of both *nominals* and QCRs.

The next chapter proposes a more informed tableau-based reasoning algorithm for a better handling of *nominals*, QCRs and their interaction. The algorithm is based on a hybrid approach combining tableau-based reasoning (as introduced in Chapter 2), with algebraïc reasoning (as introduced in Section 3.3) and handles the expressivity of the DL $\mathcal{SHOQ}$.

Chapter 4

Hybrid Algebraïc Reasoning Procedure for $\mathcal{SHOQ}$

This chapter demonstrates how a standard tableau reasoning algorithm for $\mathcal{SHOQ}$ can be extended with an algebraïc component while maintaining soundness, completeness and termination. Recall that $\mathcal{SHOQ}$ is the basic DL $\mathcal{ALC}$ extended with *transitive roles* ($\mathcal{S}$), *role hierarchies* ($\mathcal{H}$), *nominals* ($\mathcal{O}$) and QCRs ($\mathcal{Q}$). Since *nominals* carry numerical restrictions, algebraïc reasoning is used to ensure their semantics while still handling their interaction with QCRs. The result is a hybrid reasoning algorithm which is more informed about arithmetic constraints imposed by concept descriptions. In particular, a better handling of numerical restrictions implied by *nominals*, QCRs and their interactions is ensured. It turns out the proposed algebraïc reasoning comes with novel characteristics, which are discussed in Section 4.7, and can be used to address the major sources of inefficient reasoning for $\mathcal{SHOQ}$.

4.1 General Overview

The algebraïc reasoning for Description Logics, proposed in this chapter, consists of combining tableau-based DL reasoning with the algebraïc method in an effort to overcome the challenges of reasoning with *nominals* and QCRs while still handling their interaction (See Section 3.1 for a detailed review of the challenging interaction between the two constructors). The tableau-based reasoning is based on a standard tableau for $\mathcal{ALC}$ [BS01], as introduced in Section 2.2.1, modified and extended to work with an algebraïc reasoning component. Algebraïc reasoning is based on the assumption that domain elements consist of a set of individuals divided into subsets depending on their role filler membership and/or concept membership. *Nominals* and QCRs represent cardinality restrictions on their corresponding sets: *nominals* are singleton sets, QCRs represent at-least and at-most restrictions on the cardinalities of the corresponding sets of role fillers. As was discussed in Section 3.1.1 the numerical restrictions imposed by *nominals* are global, therefore, a global form of the atomic decomposition technique, as introduced in Section 3.4, is considered to allow the following:

- The computation of all possible intersections between domain elements by applying it on the sets of role fillers and *nominals* in contrast to the approaches presented in [OK97, HM01a, FFHM08a, FFHM08b] where the atomic decomposition is applied on sets of role fillers of a given individual.

- The handling of possible interactions between *nominals* and role fillers.

- The encoding of the *nominals* semantics into in-equations.

- Reducing the satisfiability of *nominals* semantics and QCRs into in-equation solving.

An integer linear programming (ILP) algorithm (such as Simplex) with the objective of minimizing the sum of all cardinalities can be used to solve the encoded in-equations. If no solution for the in-equations is possible, this means that the domain elements cannot be distributed between sets without violating the cardinality restrictions. When a solution is returned, the domain elements are distributed among sets without violating any at-least or at-most restrictions, or any nominal semantics.

Tableau expansion rules are used to generate a completion graph model based on the distribution of the domain elements while also maintaining the satisfiability of concept descriptions that use propositional operators ($\sqcap, \sqcup, \neg$) and $\forall$, $\forall_{\setminus}$ operators, or invoking the algebraïc component if additional numerical restrictions need to be satisfied. When creating role-successor nodes, only one proxy node is used as a representative for an atomic set.

Before describing the calculus, a preprocessing of concept descriptions is defined in Section 4.1.1 to allow a distinction between the numerical restrictions and the qualifications expressed by QCRs by rewriting concept descriptions in $\mathcal{SHOQ}$ DL into concept description in $\mathcal{SHON}_{\mathcal{R}\setminus}$. The principles underlying the non-tableau reasoning methods used are discussed in Section 4.2. A tableau for the DL $\mathcal{SHON}$ is defined in Section 22 and the algebraïc method for $\mathcal{SHON}$ is described in Section 4.4. The hybrid reasoning algorithm is detailed in Section 4.5. Proofs of soundness completeness and termination are devised in Section 4.6. A discussion of the algorithm is shown in Section 4.7. The chapter is concluded in Section 4.8. For convenience of the reader and ease of reference, a list of all notations introduced in this chapter is compiled into Section A.1.

4.1.1 Preprocessing

Recall from Section 2.1.3 that subsumption and satisfiability inferences can be reduced to each other. Satisfiability of concepts w.r.t. a knowledge base can also be reduced to knowledge base consistency: a concept C is satisfiable w.r.t. $KB(\mathcal{T}, \mathcal{R})$ if and only if $(\mathcal{T} \cup \{\{o\} \sqsubseteq C\}, \mathcal{R})$ is consistent, for o a *nominal* that does not occur in C or $\mathcal{T}$. As a consequence, in the remainder of this chapter and without loss of generality, we will restrict our attention to knowledge base consistency. Also, recall that when checking a $KB(\mathcal{T}, \mathcal{R})$ consistency, the concept axioms in $\mathcal{T}$ can be internalized into a single axiom $\top \sqsubseteq C_{\mathcal{T}}$ such that $C_{\mathcal{T}}$ abbreviates $\prod_{(C \sqsubseteq D) \in \mathcal{T}} nnf(\neg C \sqcup D)$, where C, D refer to general concept descriptions, as introduced in Section 2.1.3 (before NNF was introduced).

A KB consistency test can be performed by checking the consistency of $\{o\} \sqsubseteq C_{\mathcal{T}}$ with $o \in N_o$ new in $\mathcal{T}$, which means that at least $\{o\}^{\mathcal{I}} \subseteq C_{\mathcal{T}}^{\mathcal{I}}$ and $C_{\mathcal{T}}^{\mathcal{I}} \neq \emptyset$. Moreover, since $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$ then every domain element must also satisfy $C_{\mathcal{T}}$ (every domain element is a member of $C_{\mathcal{T}}$).

Definition 4.1.1 (Qualifying Concept) A qualifying concept D is a concept used to impose a qualification, D, on the set of R-fillers for some role $R \in N_R$. Let $Q_C(R) = \{D \mid \forall S.D$ occurs in $C_{\mathcal{T}}$ with $R \sqsubseteq_* S \in \mathcal{R}\}$ define the set of *qualifying concepts* for $R \in N_R$, and let $\mathcal{Q}_C = \bigcup_{R \in N_R} Q_C(R)$ define the set of *qualifying concepts* in $C_{\mathcal{T}}$. The set $\mathcal{Q}_C^{\neg} = \{\neg D \mid D \in \mathcal{Q}_C\}$ defines the set of negated *qualifying concepts* in their NNF. A mapping is maintained between a *qualifying concept* and the NNF of its complement using a bijection $\dot{\neg}_Q : Q_C \longrightarrow Q_C^{\neg}$ such that given a qualifying concept $D \in \mathcal{Q}_C$, $\dot{\neg}_Q(D) = \dot{\neg}D$, $\dot{\neg}D \in Q_C^{\neg}$, and $\mathcal{Q}_C \cap \mathcal{Q}_C^{\neg} = \emptyset$.

In order to allow the applicability of the algebraïc method, a separation between

numerical restrictions and their qualifications is needed. This is done using Algorithm 4.1.1, which rewrites concept expressions occurring in $C_{\mathcal{T}}$, similarly to [OK99], allowing a separation between numerical restrictions and their qualifications. Since qualifications on role fillers can be encapsulated by the use of GCIs and *transitive roles* (as will be elaborated in Section 4.2.3), Algorithm 4.1.1 also allows a bookkeeping of negated *qualifying concepts* in their preprocessed NNF into the set $Q_C^{\neg}$.

Definition 4.1.2 (Role-Set Difference Operator) Given $C_{\mathcal{T}}$, a set N_R of roles, $\mathcal{R}$ a set of RIAs, and Q_C a set of *qualifying concepts*, we define a new concept operator $\forall_{\backslash}$, the *role-set difference* operator, used for descriptions like $\forall(R \backslash S).D$ such that R, S in N_R and D a $\mathcal{SHOQ}$ concept. The $\forall_{\backslash}$ operator is based on set semantics such that given an interpretation $\mathcal{I}$, then $(\forall(R \backslash S).D)^{\mathcal{I}} = \{s \in \Delta^{\mathcal{I}} \mid \langle s, t \rangle \in R^{\mathcal{I}} \wedge \langle s, t \rangle \notin S^{\mathcal{I}} \Rightarrow t \in D^{\mathcal{I}}\}$ is satisfied.

Algorithm 4.1.1 *rw*: Given the $\mathcal{SHOQ}$ concepts $A \in N_C$, C, D; $R \in N_R$; $\mathcal{R}$, the set of RIAs; and $Q_C^{\neg}$, the set of negated qualifying concepts, the following rewriting holds:

1: $rw(A, N_R, \mathcal{R}, Q_C^{\neg}) \longrightarrow A$

2: $rw(\neg A, N_R, \mathcal{R}, Q_C^{\neg}) \longrightarrow \neg A$

3: $rw((C \sqcap D), N_R, \mathcal{R}, Q_C^{\neg}) \longrightarrow (rw(C, N_R, \mathcal{R}, Q_C^{\neg}) \sqcap rw(D, N_R, \mathcal{R}, Q_C^{\neg}))$

4: $rw((C \sqcup D), N_R, \mathcal{R}, Q_C^{\neg}) \longrightarrow (rw(C, N_R, \mathcal{R}, Q_C^{\neg}) \sqcup rw(D, N_R, \mathcal{R}, Q_C^{\neg}))$

5: $rw(\neg C, N_R, \mathcal{R}, Q_C^{\neg}) \longrightarrow rw(\dot{\neg} C, N_R, \mathcal{R}, Q_C^{\neg})$

6: $rw(\forall R.C, N_R, \mathcal{R}, Q_C^{\neg}) \longrightarrow \forall R.rw(C, N_R, \mathcal{R}, Q_C^{\neg} \cup rw(\dot{\neg} C, N_R, \mathcal{R}, Q_C^{\neg}))$

7: $rw(\geq nR.C, N_R, \mathcal{R}, Q_C^{\neg}) \longrightarrow (\geq nR' \sqcap \forall R'.rw(C, N_R \cup \{R'\}, \mathcal{R} \cup \{R' \sqsubseteq R\}, Q_C^{\neg}))$

//same with $\geq nR.\top$

8: $rw(\leq nR.C, N_R, \mathcal{R}, Q_C^{\neg}) \longrightarrow$

$(\leq nR' \sqcap \forall R'.rw(C, N_R \cup \{R'\}, \mathcal{R}, Q_C^{\neg}) \sqcap \forall (R \backslash R').rw(\dot{\neg} C, N_R \cup \{R'\}, \mathcal{R} \cup \{R' \sqsubseteq R\}, Q_C^{\neg}))$

//same with $\leq nR.\top$

71

Let $\mathcal{SHON}_{\mathcal{R}\backslash}$ denote the DL $\mathcal{SHO}$ extended with unqualified cardinality restrictions ($\mathcal{N}$) and the role-set difference operator ($_{\mathcal{R}\backslash}$), Algorithm 4.1.1 is applied to $C_{\mathcal{T}}$ such that $rw(C_{\mathcal{T}}, N_{\mathrm{R}}, \mathcal{R}, Q_{\mathrm{C}}^{\neg})$ returns an equi-satisfiable concept expression ($C_{\mathcal{T}}'$) in the DL $\mathcal{SHON}_{\mathcal{R}\backslash}$ as shown in Lemma 4.1.3. Note that $\mathcal{SHON}_{\mathcal{R}\backslash}$ is not closed under negation; for example, with $\forall R.(\forall S.C)$, the qualifying concept for R corresponds to $\forall S.C$, and the negation of the qualifying concept for R, $\dot{\neg}(\forall S.C)$ is equal to $\geq 1S.\neg C$, which is not in $\mathcal{SHON}_{\mathcal{R}\backslash}$. Therefore, a bookkeeping of the preprocessed form of $\dot{\neg}C$, for every *qualifying concept* C, into $Q_{\mathrm{C}}^{\neg}$ is required. $Q_{\mathrm{C}}^{\neg}$ is initially empty; it is extended with the preprocessed form of $\dot{\neg}C$ every time rw is applied to a concept of the form $\forall R.C$. This means that when rw is applied to $\forall R.(\forall S.C)$, the following expression $\geq 1S_1 \sqcap \forall S_1.\neg C$, corresponding to the preprocessed form of $\dot{\neg}(\forall S.C)$, is added to $Q_{\mathrm{C}}^{\neg}$ such that $\dot{\neg}_Q(\forall S.C) = \geq 1S_1 \sqcap \forall S_1.\neg C$ is in $\mathcal{SHON}_{\mathcal{R}\backslash}$. Also, N_{R} and $\mathcal{R}$ are extended with a fresh role R' new in $\mathcal{T}$ every time the conditions in lines 7 and 8 are applicable. It can be shown that Lemma 4.1.3 holds.

Lemma 4.1.3 (Preserving Satisfiability) Rewriting $C_{\mathcal{T}}$ according to Algorithm 4.1.1 preserves satisfiability. Satisfying $C_{\mathcal{T}}$ w.r.t. $\mathcal{R}$ consists of satisfying $C_{\mathcal{T}}'$ w.r.t. $\mathcal{R}$.

Proof. It is easy to see that satisfiability is preserved for atomic concepts, negated concepts, conjunctions and disjunctions of concepts. Let C, D be $\mathcal{SHOQ}$ concepts; n, m non-negative integer numbers; and R a role name in N_{R}, with $\mathcal{R}$ a set of role implications. One can show that $\geq nR.C$ is satisfiable iff $rw(\geq nR.C, N_{\mathrm{R}}, \mathcal{R}, Q_{\mathrm{C}}^{\neg})$ is satisfiable, and $\leq mR.D$ is satisfiable iff $rw(\leq mR.D, N_{\mathrm{R}}, \mathcal{R}, Q_{\mathrm{C}}^{\neg})$ is satisfiable. Since *transitive roles* are not used in at-least restrictions $\geq nR.C$ with $n > 1$, nor in at most restrictions with $n > 0$, preserving the satisfiability of $rw(\geq 1R.C, N_{\mathrm{R}}, \mathcal{R}, Q_{\mathrm{C}}^{\neg})$ is an easy consequence of the following proofs.

72

1. If $\geq nR.C$ is satisfiable then $\geq nR' \sqcap \forall R'.C$ is satisfiable w.r.t. $\mathcal{R}$.

   Proof. Assume that $\geq nR.C$ is satisfiable, this means that there exists a non-empty interpretation $\mathcal{I}$ with:

   (a) an individual $s \in \Delta^{\mathcal{I}}$ such that $s \in (\geq nR.C)^{\mathcal{I}}$, and

   (b) n distinct individuals $t_1 \dots t_n \in \Delta^{\mathcal{I}}$ such that $t_i \in (FIL(R, s) \cap C^{\mathcal{I}})$ for $1 \leq i \leq n$.

   One can construct the interpretation, $\mathcal{I}'$, of $\geq nR' \sqcap \forall R'.C$ from $\mathcal{I}$. Let $\mathcal{I}' = \mathcal{I}$, R' a new role name in N_{R} such that $FIL(R', s) = FIL(R, s) \cap C^{\mathcal{I}}$. For $s \in \Delta^{\mathcal{I}'}$ the following holds:

   (a) $s \in (\geq nR')^{\mathcal{I}'}$ since $FIL(R', s) \subseteq FIL(R, s)$ and there exists $t_1 \dots t_n \in FIL(R', s)$,

   (b) One can add $R' \sqsubseteq R \in \mathcal{R}$, and $\mathcal{I}'$ satisfies $\mathcal{R}$ because by definition of R' all the R'-fillers are also R-fillers,

   (c) $s \in (\forall R'.C)^{\mathcal{I}'}$ since $FIL(R', s) \subseteq C^{\mathcal{I}'}$, and

   (d) $s \in (\geq nR.C)^{\mathcal{I}}$ is not violated.

   Hence, if $\geq nR.C$ is satisfiable then $\geq nR' \sqcap \forall R'.C$ is also satisfiable w.r.t. $\mathcal{R}$.

2. If $\geq nR' \sqcap \forall R'.C$ is satisfiable w.r.t. $\mathcal{R}$ with $R' \sqsubseteq R \in \mathcal{R}$ then $\geq nR.C$ is satisfiable.

   Proof. Assume that $\geq nR' \sqcap \forall R'.C$ w.r.t. $\mathcal{R}$ is satisfiable, this means that there exists a non-empty interpretation $\mathcal{I}'$ with:

   (a) An individual $s \in \Delta^{\mathcal{I}'}$ such that $s \in (\geq nR' \sqcap \forall R'.C)^{\mathcal{I}'}$, and

(b) n distinct individuals $t_1 \ldots t_n \in \Delta^{\mathcal{I}'}$ such that $t_i \in FIL(R', s)$ and $t_i \in C^{\mathcal{I}'}$

for $1 \leq i \leq n$.

It is easy to construct the interpretation $\mathcal{I}$ of $\geq nR.C$ from $\mathcal{I}'$; setting $\mathcal{I} = \mathcal{I}'$ gives $s \in (\geq nR.C)^{\mathcal{I}}$ since there already exists n distinct individuals $t_1 \ldots t_n \in \Delta^{\mathcal{I}}$ satisfying $t_i \in FIL(R, s) \cap C^{\mathcal{I}}$ for $1 \leq i \leq n$. Hence, if $\geq nR' \sqcap \forall R'.C$ is satisfiable w.r.t. $\mathcal{R}$ then $\geq nR.C$ is also satisfiable.

3. If $\leq mR.D$ is satisfiable then $\leq mR' \sqcap \forall R'.D \sqcap \forall(R \backslash R').\neg D$ is satisfiable w.r.t. $\mathcal{R}$.

Proof. Assume that $\leq mR.D$ is satisfiable, this means that there exists a non-empty interpretation $\mathcal{I}$ with:

(a) An individual $s \in \Delta^{\mathcal{I}}$ such that $s \in (\leq mR.D)^{\mathcal{I}}$, and

(b) At most m individuals $t_1 \ldots t_m \in \Delta^{\mathcal{I}}$ such that $t_i \in FIL(R, s)$ and $t_i \in D^{\mathcal{I}}$ for $1 \leq i \leq m$.

One can construct the interpretation, $\mathcal{I}'$, of $\leq mR' \sqcap \forall R'.D \sqcap \forall(R \backslash R').\neg D$ from $\mathcal{I}$. Let $\mathcal{I}' = \mathcal{I}$ and one can create a new role name R' in N_R such that $FIL(R', s) = FIL(R, s) \cap D^{\mathcal{I}}$. For $s \in \Delta^{\mathcal{I}'}$ the following holds:

(a) $s \in (\leq mR')^{\mathcal{I}'}$ since $FIL(R', s) \subseteq FIL(R, s)$ and there exists $t_1 \ldots t_m \in FIL(R', s)$,

(b) One can add $R' \sqsubseteq R \in \mathcal{R}$ and $\mathcal{I}'$ satisfies $\mathcal{R}$ because by definition of R' all the R'-fillers are also R-fillers,

(c) $s \in (\forall R'.D)^{\mathcal{I}'}$ since $FIL(R', s) \subseteq D^{\mathcal{I}'}$,

(d) $s \in (\forall R \backslash R'.\neg D)^{\mathcal{I}'}$. Since there can be at most m individuals in $FIL(R, s)$ $\cap D^{\mathcal{I}}$, this means that all intersections with $FIL(R, s)$ that do not also intersect with $FIL(R', s)$ cannot intersect with $D^{\mathcal{I}'}$; $FIL(R, s) \setminus FIL(R', s) \subseteq$ $\neg D^{\mathcal{I}'}$. Therefore, one can safely assign $t_1, \ldots, t_m$ to $D^{\mathcal{I}'}$ and all individuals in $(FIL(R,\ s) \backslash (FIL(\text{R',s})))$ can be assigned to $(\neg D^{\mathcal{I}'})$, and

(e) $s \in (\leq mR.D)^{\mathcal{I}}$ is not violated.

Hence if $\leq mR.D$ then $\leq mR' \sqcap \forall R'.D \sqcap \forall (R \backslash R').\neg D$ is satisfiable w.r.t. $\mathcal{R}$.

4. If $\leq mR' \sqcap \forall R'.A \sqcap \forall (R \backslash R').\neg A$ is satisfiable w.r.t. $\mathcal{R}$ then $\leq mR.A$ is satisfiable

Proof. Assume that $\leq mR' \sqcap \forall R'.A \sqcap \forall (R \backslash R').\neg A$ is satisfiable w.r.t. $\mathcal{R}$, this means that there exists a non-empty interpretation $\mathcal{I}'$ with:

(a) An individual $s \in \Delta^{\mathcal{I}'}$ such that $s \in (\leq mR' \sqcap \forall R'.D \sqcap \forall (R \backslash R').\neg D)^{\mathcal{I}'}$,

(b) At most m distinct individuals $t_1 \ldots t_m \in \Delta^{\mathcal{I}'}$ such that $t_i \in FIL(R, s)$ and $t_i \in D^{\mathcal{I}'}$ for $1 \leq i \leq m$, and

(c) $FIL(R, s) \setminus FIL(R', s) \subseteq \neg D^{\mathcal{I}'}$.

It is easy to construct the interpretation $\mathcal{I}$ of $\leq mR.D$ from $\mathcal{I}'$; setting $\mathcal{I} = \mathcal{I}'$ gives $s \in (\leq mR.D)^{\mathcal{I}}$ since there already exist at most m distinct individuals $t_1 \ldots t_m \in \Delta^{\mathcal{I}}$ satisfying $t_i \in FIL(R, s) \cap D^{\mathcal{I}}$ for $1 \leq i \leq m$. Hence, if $\leq mR' \sqcap \forall R'.A \sqcap \forall (R \backslash R').\neg A$ is satisfiable w.r.t. $\mathcal{R}$ then $\leq mR.D$ is also satisfiable.

∎

Preprocessing Examples This section illustrates the process of applying the preprocessing algorithm with two examples. Example 4.1.4 illustrates the preprocessing of a concept expression, and Example 4.1.5 shows the preprocessing of a TBox.

Example 4.1.4 Applying rw to the concept expression in (19) where $N_R = \{R\}$, $N_o = \{o\}$, $\mathcal{R} = \emptyset$, and $Q_C^- = \emptyset$ gives the concept expression in (20) with $\mathcal{R} = \{R_1 \sqsubseteq R, R_2 \sqsubseteq R\}$, $Q_C^- = \emptyset$ and $N_R = \{R, R_1, R_2\}$.

$$\geq 1\, R.(\{o\} \sqcap\, \leq 1\, R.\{o\}) \tag{19}$$

$$\geq 1R_1 \sqcap \forall R_1.(\{o\} \sqcap\, \leq 1R_2 \sqcap \forall R_2.\{o\} \sqcap \forall R \backslash R_2.\neg\{o\}) \tag{20}$$

Example 4.1.5 Let the TBox $\mathcal{T}$ in Figure 17 represent the EU member states example, as introduced in Chapter 1.

EU_MemberState	$\equiv \{\mathsf{Austria}\} \sqcup \ldots \sqcup \{\mathsf{UK}\}$
Future_EU	$\sqsubseteq\, \geq 30\mathsf{MemberOf.EU\_MemberState}$

Figure 17: TBox axioms representing the EU_MemberState example.

The TBox is internalized into $C_{\mathcal{T}}$ as shown in Figure 18a. Initially, $N_R = \{\mathsf{MemberOf}\}$, $\mathcal{R} = \emptyset$, $Q_C^- = \emptyset$ and $rw(C_{\mathcal{T}}, N_R, \mathcal{R}, Q_C^-)$ extends N_R to $N_R = \{\mathsf{M'}, \mathsf{MemberOf}\}$, $\mathcal{R}$ to $\mathcal{R} = \{\mathsf{M'} \sqsubseteq \mathsf{MemberOf}\}$, and $C_{\mathcal{T}}$ to $C_{\mathcal{T}}'$ shown in Figure 18b.

$$
\begin{aligned}
C_{\mathcal{T}} = \;& (\neg \mathsf{EU\_MemberState} \sqcup \{\mathsf{Austria}\} \sqcup \ldots \sqcup \{\mathsf{UK}\}) \sqcap \\
& ((\neg\{\mathsf{Austria}\} \sqcap \ldots \sqcap \neg\{\mathsf{UK}\}) \sqcup \mathsf{EU\_MemberState}) \sqcap \\
& (\neg \mathsf{Future\_EU} \sqcup \; \geq 30\,\mathsf{MemberOf.EU\_MemberState})
\end{aligned}
$$

(a) TBox internalization into $C_{\mathcal{T}}$.

$$
\begin{aligned}
C'_{\mathcal{T}} = \;& (\neg \mathsf{EU\_MemberState} \sqcup \{\mathsf{Austria}\} \sqcup \ldots \sqcup \{\mathsf{UK}\}) \sqcap \\
& ((\neg\{\mathsf{Austria}\} \sqcap \ldots \sqcap \neg\{\mathsf{UK}\}) \sqcup \mathsf{EU\_MemberState}) \sqcap \\
& (\neg \mathsf{Future\_EU} \sqcup \; \geq 30\, \mathsf{M'} \sqcap \forall \mathsf{M'.EU\_MemberState})
\end{aligned}
$$

(b) Applying rw to $C_{\mathcal{T}}$ gives $C'_{\mathcal{T}}$.

Figure 18: TBox internalization into $C'_{\mathcal{T}}$ in $\mathcal{SHON}_{\mathcal{R}\backslash}$.

4.2 Algebraïc Reasoning and $\mathcal{SHOQ}$

When considering algebraïc reasoning for the DL $\mathcal{ALCQ}$, algebraïc reasoning need only capture the numerical restrictions implied by the QCRs constructor. However, in order to use the algebraïc method with the DL $\mathcal{SHOQ}$, one must consider the following implied restrictions due to the expressivity of the language constructors: global numerical restrictions imposed by *nominals* ($\mathcal{O}$), cycles introduced by the use of *transitive roles* and GCIs ($\mathcal{S}$), qualifications on roles imposed by the use of *role hierarchies* and GCIs ($\mathcal{H}$).

4.2.1 Global Numerical Restrictions

Recall from Section 3.1.1 that the numerical restrictions implied by *nominals* are global restrictions that affect domain elements as a whole. These restrictions could

interact with the numerical restrictions imposed by QCRs as it is the case with the definition of Future_EU in (22), which implies a numerical restriction that is local to MemberOf-fillers, and the definition of EU_MemberState in (21), which implies a numerical restriction that is global and affects all elements in the domain. This means that applying the algebraïc method locally to each individual as is the case with algebraïc reasoning for $\mathcal{ALCQ}$ [FFHM08a, FFHM08b, FH10c] can no longer ensure soundness; the algebraïc reasoner may satisfy local numerical restrictions imposed by QCRs without necessarily satisfying the global ones imposed by *nominals*. A global form of applying the atomic decomposition technique on a decomposition set capturing the semantics of *nominals*, QCRs, and the interaction between the two constructors is needed.

$$\text{EU\_MemberState} \equiv \{\text{Austria}\} \sqcup \ldots \sqcup \{\text{UK}\} \tag{21}$$

$$\text{Future\_EU} \sqsubseteq \geq 30\text{MemberOf}.\text{EU\_MemberState} \tag{22}$$

4.2.2 Cyclic Descriptions

When *transitive roles* and/or GCIs are allowed, as is the case with the DL $\mathcal{SHOQ}$, one must keep in mind that the process of expanding a node's label based on concept description, may no longer terminate because cyclic descriptions can repeat concept labels through nodes as is the case with the tableau algorithm described in Section 2.2.1. For example, having the concept description (23) in the label of a node x, such that R is a transitive role, the completion rules introduce a node y as an R-successor of x such that after applying $\forall$-Rule and the $\forall_+$-Rule, the node y has the same label as x. Traditional tableau algorithms [HS01] for DLs with *transitive roles* and GCIs implement blocking strategies to detect and handle cycles. The algorithm presented in this chapter shows how global partitioning of domain elements allows a

re-use strategy which can also handle cycles.

$$A \sqcap \geq 1R \sqcap \forall R.A \sqcap \forall R.(\geq 1R \sqcap \forall R.A) \tag{23}$$

4.2.3 Encapsulated Qualifications on Role-Fillers

The preprocessing algorithm described in Algorithm 4.1.1 serves as a separation between numerical restrictions and their qualifications. Since a newly introduced role is used for each QCR, the separation is more syntactic, because semantically, every $\forall R'.C$ is directly linked to its R'-fillers and no other concept expression uses R'. On the other hand, the qualifications on role fillers need not always be explicitly used with QCRs, but encapsulated through the use of GCIs and *role hierarchies* . The following example illustrates such a case.

Example 4.2.1 Assume checking the satisfiability of the concept $(\geq 1S_1.(A \sqcap B_1) \sqcap \geq 1S_2.(A \sqcap B_2))$ w.r.t. the TBox $\mathcal{T}$ shown in Figure 19.

$$
\begin{array}{rcl}
A & \sqsubseteq & \geq 1R.B \\
B_1 & \sqsubseteq & \forall R.C \\
B_2 & \sqsubseteq & \forall R.\neg C
\end{array}
$$

Figure 19: TBox example.

The numerical restriction $(\geq 1R.B)$ encapsulated in A is common to S_1-fillers and S_2-fillers which both require an R-filler being a member of B. On the other hand, S_1-fillers and S_2-fillers have different *qualifying concepts* for their R-fillers due to the axioms for concepts B_1, B_2. S_1-fillers which are members of B_1 must have R-fillers being members of C, and S_2-fillers which are members of B_2 must have R-fillers being members of $\neg C$. The satisfiability test can be done by adding the axiom (24) to $\mathcal{T}$

with $o \in N_o$ a nominal new in $\mathcal{T}$.

$$\{o\} \sqsubseteq \ \geq 1S_1.(A \sqcap B_1) \sqcap \ \geq 1S_2.(A \sqcap B_2) \tag{24}$$

In principle, when preprocessing a KB by applying the rewriting algorithm, one has two choices: Case (1) or Case (2). When the TBox is unfoldable one can opt for case (1) and otherwise one has to consider case (2).

- Case (1): Unfolding $\mathcal{T}$ by replacing A with $(\geq 1R.B)$, B_1 with $\forall R.C$, and B_2 with $\forall R.\neg C$ would make all $(\geq nR.C)$ restrictions explicit. The TBox can be internalized to $C_{\mathcal{T}}$ shown in Figure 20a and rewriting QCRs results in $C'_{\mathcal{T}}$ as shown in Figure 20b. A distinction can be made between R-fillers of S_1-fillers and those of S_2-fillers because rw uses a different role for each occurrence of $\geq 1R.B$.

$$C_{\mathcal{T}} = \ \neg\{o\} \sqcup (\geq 1S_1.(\geq 1R.B \sqcap \forall R.C) \sqcap \ \geq 1S_2.(\geq 1R.B \sqcap \forall R.\neg C))$$

(a) TBox internalization into $C_{\mathcal{T}}$ after unfolding $\mathcal{T}$.

$$C'_{\mathcal{T}} = \ \neg\{o\} \sqcup (\geq 1S_{11} \sqcap \forall S_{11}.(\geq 1R_1 \sqcap \forall R_1.B \sqcap \forall R.C) \sqcap$$
$$\geq 1S_{21} \sqcap \forall S_{21}.(\geq 1R_2 \sqcap \forall R_2.B \sqcap \forall R.\neg C))$$

(b) Applying Algorithm 4.1.1 to $C_{\mathcal{T}}$.

Figure 20: Association between roles and their qualifications after rewriting, when $\mathcal{T}$ is unfoldable.

In this case, the algebraïc method will automatically consider the cases when R-fillers have different qualifications due to R_1 and R_2 which are sub-roles of R.

- Case (2): When the TBox is not unfolded or cannot be unfolded then $C_{\mathcal{T}}$ is of the form shown in Figure 21a and $C'_{\mathcal{T}}$ is shown in Figure 21b.

80

$$
\begin{aligned}
C_{\mathcal{T}} = \quad &\neg A \sqcup \geq 1R.A \sqcap \\
&\neg B_1 \sqcup \forall R.C \sqcap \\
&\neg B_2 \sqcup \forall R.\neg C \sqcap \\
&\neg\{o\} \sqcup \geq 1S_1.(A \sqcap B_1) \sqcap \geq 1S_2.(A \sqcap B_2))
\end{aligned}
$$

(a) TBox internalization into $C_{\mathcal{T}}$.

$$
\begin{aligned}
C'_{\mathcal{T}} = \quad &\neg A \sqcup \geq 1R_1 \sqcap \forall R_1.A \sqcap \\
&\neg B_1 \sqcup \forall R.C \sqcap \\
&\neg B_2 \sqcup \forall R.\neg C \sqcap \\
&\neg\{o\} \sqcup \geq 1S_{11} \sqcap \forall S_{11}.(A \sqcap B_1) \sqcap \geq 1S_{21} \sqcap \forall S_{21}.(A \sqcap B_2))
\end{aligned}
$$

(b) Rewriting $C_{\mathcal{T}}$ into $C'_{\mathcal{T}}$.

Figure 21: Association between roles and their qualifications after rewriting when $\mathcal{T}$ is not unfolded.

In this case, the qualifications differentiating R-fillers are still encapsulated in B_1 and B_2; S_{11}-fillers and S_{21}-fillers have different *qualifying concepts* for their R_1-fillers. R_1-fillers of S_{11}-fillers must also be members of C and this is encapsulated in B_1, and R_1-fillers of S_{21}-fillers must be members of $\neg C$ and this is encapsulated in B_2.

This example shows the problem of encapsulated qualifications on role fillers due to the use of GCIs. These qualifications could also be inherent due to a role hierarchy or role transitivity, and if not taken into consideration make the algebraïc method incomplete. After a tableau for the DL $\mathcal{SHON}_{\mathcal{R}\backslash}$ is presented in the following section, Section 4.4, shows how the algebraïc method handles the challenge of encapsulated qualifications using a proper global decomposition set.

4.3 A Tableau for the DL $\mathcal{SHON}_{\mathcal{R}\backslash}$

This section defines a tableau for the DL $\mathcal{SHON}_{\mathcal{R}\backslash}$ based on the standard tableau for the DL $\mathcal{SHOQ}$, which was introduced in [HS01].[1] It is important to note that $\mathcal{SHON}_{\mathcal{R}\backslash}$ is not closed under negation due to the preprocessing step described in Algorithm 4.1.1, but this does not cause a problem because the proposed calculus never negates a preprocessed concept.[2]

Given concept description C (in the DL $\mathcal{SHON}_{\mathcal{R}\backslash}$), let *clos(C)* define the smallest set of concepts such that:

- (a) $C \in clos(C)$,

- (b) if $A \in N_R$ and $A \in clos(C)$ then $\neg A \in clos(C)$,

- (c) if $(E \sqcap D)$ or $(E \sqcup D) \in clos(C)$ then $E, D \in clos(C)$,

- (d) if $\forall R.D$ or $\forall R\backslash S.D \in clos(C)$ then $D \in clos(C)$.

The size of *clos(C)* is bounded by the size of C. The set of relevant sub-concepts of a TBox $\mathcal{T}$ is then defined as $clos(\mathcal{T}) = clos(C'_{\mathcal{T}})$.

Definition 4.3.1 ($\mathcal{SHON}_{\mathcal{R}\backslash}$ Tableau) Given a $\mathcal{SHOQ}$ KB$(\mathcal{T}, \mathcal{R})$ which has been preprocessed into a $\mathcal{SHON}_{\mathcal{R}\backslash}$ KB$(\mathcal{T}, \mathcal{R})$, T $= (\mathbf{S}, \mathcal{L}, \mathcal{E})$ defines a tableau for $(\mathcal{T}, \mathcal{R})$ as an abstraction of a model for $\mathcal{T}$. $\mathbf{S}$ is a non-empty set of individuals, $\mathcal{L} : \mathbf{S} \longrightarrow 2^{clos(\mathcal{T})}$ is a mapping between each individual and a set of concepts, and $\mathcal{E} : N_R \longrightarrow 2^{\mathbf{S} \times \mathbf{S}}$ is a mapping between each role and a set of pairs of individuals in $\mathbf{S}$. For all $s, t \in \mathbf{S}$, $A \in N_C$, $C, D \in clos(\mathcal{T})$, $o \in N_o$, $R, S \in N_R$, and given the definition $R^T(s) = \{t \in \mathbf{S} \,|\, \langle s, t \rangle \in \mathcal{E}(R)\}$, properties (1) - (11) must always hold:

---

[1] For convenience, the definition of a standard tableau for $\mathcal{SHOQ}$ is illustrated in Section A.2.

[2] The negations of *qualifying concepts* are computed using $\dot{\neg}_Q$ which returns $\mathcal{SHON}_{\mathcal{R}\backslash}$ concepts (See Section 4.1.1).

1. $C'_{\mathcal{T}} \in \mathcal{L}(s)$

2. If $A \in \mathcal{L}(s)$ then $\neg A \notin \mathcal{L}(s)$.

3. If $C \sqcap D \in \mathcal{L}(s)$ then $C \in \mathcal{L}(s)$ and $D \in \mathcal{L}(s)$.

4. If $C \sqcup D \in \mathcal{L}(s)$ then $C \in \mathcal{L}(s)$ or $D \in \mathcal{L}(s)$.

5. If $\forall S.C \in \mathcal{L}(s)$ and $\langle s, t \rangle \in \mathcal{E}(S)$ then $C \in \mathcal{L}(t)$.

6. If $\forall S.C \in \mathcal{L}(s)$ and $\langle s, t \rangle \in \mathcal{E}(R)$ with $R \sqsubseteq_* S$ and $R \in N_{R+}$ then $\forall R.C \in \mathcal{L}(t)$.

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

8. If $(\geq nR) \in \mathcal{L}(s)$ then $\# R^T(s) \geq n$.

9. If $(\leq mR) \in \mathcal{L}(s)$ then $\# R^T(s) \leq m$.

10. If $\langle s, t \rangle \in \mathcal{E}(R)$ and $R \sqsubseteq_* S \in \mathcal{R}$, then $\langle s, t \rangle \in \mathcal{E}(S)$.

11. For each $o \in N_o$, $\#\{s \in \mathbf{S} \,|\, o \in \mathcal{L}(s)\} = 1$.

Lemma 4.3.2 A $\mathcal{SHOQ}$ knowledge base $\text{KB}(\mathcal{T}, \mathcal{R})$ is consistent iff there exists a tableau T for $(\mathcal{T}, \mathcal{R})$.

Proof. The proof is similar to the one found in [HS07]. Property (6) ensures that the qualification restrictions due to role transitivity are enforced while taking into consideration *role hierarchies*. Property 7 of this tableau ensures that the semantics of the $\forall (R \backslash S).C$ operator is preserved. Property 11 ensures that the semantics of *nominals* are preserved. $\blacksquare$

4.4 The Algebraïc Method for $\mathcal{SHON}_{\mathcal{R}\backslash}$

In [FHM08, FHM09] the algebraïc method is combined with tableau reasoning and is applied globally to reduce the satisfiability of concept descriptions using QCRs and/or *nominals* into in-equation solving. A key technique to enable the algebraïc reasoning is the atomic decomposition (introduced in Section 3.4) which allows the decomposition of a set of elements into mutually disjoints subsets. We illustrate how this technique can enable the algebraïc method for the DL $\mathcal{SHON}_{\mathcal{R}\backslash}$ with GCIs by using the appropriate *decomposition set*. Unlike in the other approaches, the *decomposition set* includes roles, *qualifying concepts* and *nominals*.

4.4.1 Atomic Decomposition and $\mathcal{SHON}_{\mathcal{R}\backslash}$

In order to allow the atomic decomposition technique to capture the semantics of *nominals*, *qualifying concepts*, and roles, one has to define the proper decomposition set.

Capturing role fillers Let $N_{R'}$ denote the set of role names newly introduced by Algorithm 4.1.1. Given a role $R \in N_R$, let $H(R)$ denote the set of all newly introduced sub-roles of R: $H(R) = \{R' \mid R' \sqsubseteq_* R, R' \neq R, R' \in N_{R'}\}$. There is no need to add S such that $S \sqsubseteq R$, and $S \notin N_{R'}$, to $H(R)$ since S does not occur in QCRs anymore after preprocessing. For every role $R' \in H(R)$, the set of R'-*fillers* forms a subset of the set of R-*fillers* ($FIL(R') \subseteq FIL(R)$). Let $\overline{R'}$ be the complement of R' relative to $H(R)$, the set of $\overline{R'}$-*fillers* is then defined as $FIL(\overline{R'}) = (FIL(R) \setminus FIL(R'))$.

Capturing qualifying concepts In order to distinguish the cases when role fillers have different qualifications, as was discussed in Example 4.2.1, the atomic decomposition must also consider when $FIL(R)$ intersects with the interpretation of a qualifying concept. For this purpose, one can use a set of *qualifying concepts* of R, $Q_C(R)$ as defined in Definition 4.1.1. Since $D \in Q_C(R)$ could be a complex expression or a nominal, one can refer to a qualification using a qualification name q for each $D \in Q_C(R)$. Let N_Q be the set of all qualification names assigned. For clarity purposes and ease of presentation no distinction is made between a qualification name and its corresponding *qualifying concept*; we take the liberty to use the *qualifying concept D* when referring to its name $q \in N_Q$, and vice versa.[3] Let $Q_N(R)$ denote the set of qualification names for a role ($R \in N_R$) then $Q_N(R) = Q_C(R)$.[4]

Definition 4.4.1 (Decomposition Set) Let $\mathcal{D}_R = (H(R) \cup Q_N(R))$ define the decomposition set for R-fillers. $\mathcal{D}_R$ is a decomposition set since each subset P of $\mathcal{D}_R$ ($P \subseteq \mathcal{D}_R$) defines a unique set of roles and/or qualification names that admits an interpretation $P^\mathcal{I}$ corresponding to the unique intersection of role fillers and interpretation of *qualifying concepts* for the roles and qualification names in P: $P^\mathcal{I} = \bigcap_{R' \in (P \cap H(R))} FIL(R') \cap \bigcap_{R'' \in (H(R) \setminus P)} FIL(\overline{R''}) \cap \bigcap_{D \in (P \cap N_Q)} D^\mathcal{I} \cap \bigcap_{D \in (N_Q \setminus P)} (\dot{\neg}_Q(D))^\mathcal{I}$.

$P^\mathcal{I}$ cannot overlap with role fillers for roles that do not appear in P since it is assumed to overlap with their complement. Similarly, in the case when $Q_N(R) \neq \emptyset$, $P^\mathcal{I}$ cannot overlap with the interpretation of a qualifying concept whose corresponding qualification name is not in P because it overlaps with the interpretation of its complement. This makes all $P^\mathcal{I}$ disjoint as in [OK99] and the set of all $P \subseteq \mathcal{D}_R$

---

[3]In [FH10a], a mapping between qualification names and their corresponding concept expressions is maintained using a bijection $\theta : N_Q \longrightarrow \mathcal{Q}_C$; in case a nominal $o \in N_o$ has been used as a qualifying concept expression then o is also used as the qualification name and $\theta(o) = o$.

[4]If the mapping θ is used as in [FH10c] then $Q_N(R) = \{q \in N_Q \mid q \in Q_C(R)\}$

defines a partitioning of D_R.
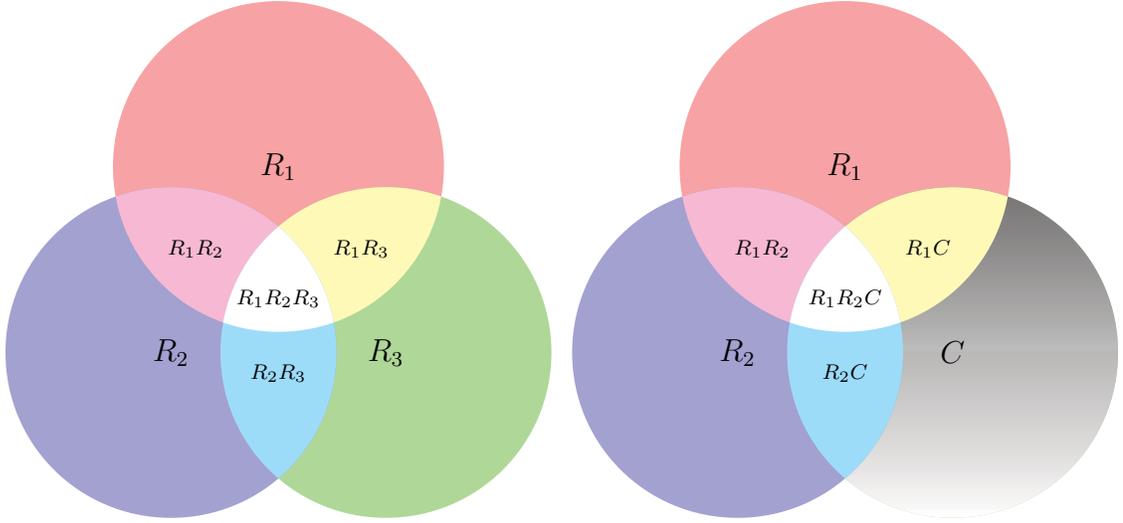
(a) Atomic decomposition of $\mathcal{D}_R = \{R_1, R_2, R_3\}$. (b) Atomic decomposition of $\mathcal{D}_R = \{R_1, R_2, C\}$.

Figure 22: Atomic decomposition of $\mathcal{D}_R$.

Example 4.4.2 Assuming a decomposition set $\mathcal{D}_R = \{R_1, R_2, R_3\}$ with $H(R) = \{R_1, R_2, R_3\}$ and $Q_N(R) = \emptyset$ and the decomposition as shown in Figure 22a, then if $P_1 = \{R_1, R_2\}$ and $P_2 = \{R_2, R_3\}$ this means that P_1 is the partition name for $FIL(R_1) \cap FIL(R_2) \cap FIL(\overline{R_3})$ which is equal to $P_1^{\mathcal{I}}$ and P_2 is the partition name for $FIL(R_2) \cap FIL(R_3) \cap FIL(\overline{R_1})$, and therefore, although $P_1 \cap P_2 = \{R_2\}$ we have $P_1^{\mathcal{I}} \cap P_2^{\mathcal{I}} = \emptyset$.

Since $\mathcal{SHON}_{\mathcal{R}\backslash}$ does not allow $\geq nR$ or $\leq nR$ concept expressions using role complements, no role complement will be explicitly used. For ease of presentation, the role complements are not listed in a partition name. Also, since a qualification is not applicable unless there exists a corresponding role filler, there is no need to consider a partition P, $P \subseteq Q_N(R)$, if P includes a qualification name without including a role. For example, Figure 22b shows the decomposition of $\mathcal{D}_R = \{R_1, R_2, C\}$ and the

part which corresponds to the partition $P = \{C\}$ does not need to be considered.

Capturing nominals   For each nominal $o \in N_o$, $\{o\}^{\mathcal{I}}$ can interact with R-*fillers* for some R in N_R such that $(\{o\}^{\mathcal{I}} \subseteq FIL(R))$. Also the same nominal o can interact with R-*fillers* and S-*fillers* for $R, S \in N_R$ such that R, S do not necessarily share sub-roles or super-roles in $\mathcal{R}$. This means that R-*fillers* and S-*fillers* could interact with each other due to their common interaction with the same nominal o. These interactions lead to the following definition of a *Global Decomposition Set* (GDS).

Definition 4.4.3 (Global Decomposition Set)   A global decomposition set is defined as the set of all role names, *qualifying concepts*, and *nominals* occurring in $C'_{\mathcal{T}}$ as: $\mathcal{DS} = \bigcup_{R \in N_R} \mathcal{D}_R \cup N_o$. When C and $\neg C$ are both used as *qualifying concepts*, only C is included in $\mathcal{DS}$. Applying the decomposition technique on $\mathcal{DS}$ defines a *global partitioning* of domain elements.

Definition 4.4.4 (Global Partitioning)   A global partitioning on domain elements is defined as follows: Let $\mathcal{P}$ be the set of the disjoint partition names defined for the decomposition of $\mathcal{DS}$: $\mathcal{P} = \{P \mid P \subseteq \mathcal{DS}\}$. Then $\mathcal{P}^{\mathcal{I}} = \Delta^{\mathcal{I}}$ because it includes all possible domain elements which correspond to a nominal and/or a role filler $\mathcal{P}^{\mathcal{I}} = \bigcup_{P \subseteq \mathcal{DS}} P^{\mathcal{I}}$.

4.4.2   Partitions and Signatures

A given model $\mathcal{I}$ of a KB $(\mathcal{T}, \mathcal{R})$ consists of domain elements grouped into mutually disjoint partitions. Each partition represents a signature of concept descriptions that is common to all elements in the partition. The signature F of a partition p is given

based on the elements represented by p such that:

$$F = \prod_{o \in (N_o \cap p)} \{o\} \sqcap \prod_{R \in (p \cap N_{R'}), \forall R.C \in T'} C \sqcap \prod_{D \in N_Q} D$$

F represents a $\mathcal{SHON}_{\mathcal{R}\backslash}$ concept expression, and a model $\mathcal{I}$ of $\mathcal{T}$ satisfies a signature F iff $F^{\mathcal{I}} \neq \emptyset$ such that:

$$F^{\mathcal{I}} = \left\{ \begin{array}{l} \bigcap_{o \in (N_o \cap p)} \{o\}^{\mathcal{I}} \cap \bigcap_{o' \in (N_o \backslash p)} \{\neg o'\}^{\mathcal{I}} \cap \\ \bigcap_{R \in (p \cap N_{R'}), \forall R.C \in T'} C^{\mathcal{I}} \cap \\ \bigcap_{D \in (N_Q \cap p)} D^{\mathcal{I}} \cap \bigcap_{D' \in (N_Q \backslash D)} (\dot{\neg}_Q D')^{\mathcal{I}} \end{array} \right\}$$

Lemma 4.4.5 Given a model $\mathcal{I}$ of $\mathcal{T}$, for each non-empty partition $p^{\mathcal{I}} \subseteq \mathcal{P}^{\mathcal{I}}$, and two domain elements $i, j \in p^{\mathcal{I}}$, if $i \in F^{\mathcal{I}}$ (F is the signature of p) then: (1) $j \in F^{\mathcal{I}}$ and, (2) there exists no other domain element $i' \in \Delta^{\mathcal{I}}$ such that $i' \in p^{\mathcal{I}} \cap F^{\mathcal{I}} \cap p'^{\mathcal{I}}$ for some partition $p'^{\mathcal{I}} \subseteq \mathcal{P}^{\mathcal{I}}$ different from $p^{\mathcal{I}}$.

Proof. It is easy to prove (2) since all partitions are disjoint by definition. For (1), given $R_1, \ldots R_n \in N_R$, $o_1, \ldots, o_n \in N_o$, $q_1 \ldots q_n \in N_Q$, and $i, j \in \Delta^{\mathcal{I}}$ we consider Cases 1-5.

- Case 1 - *Nominals* partition: $p^{\mathcal{I}}$ is a *nominals* partition, then it corresponds to some partition name $p \in \mathcal{P}$ of the form $p = \{o_1, \ldots, o_n\}$ and individuals in $p^{\mathcal{I}}$ satisfy the signature F such that $F^{\mathcal{I}} = p^{\mathcal{I}} = (\{o_1\}^{\mathcal{I}} \cap \ldots \cap \{o_n\}^{\mathcal{I}})$, assuming N_R and N_Q are empty. Given the *nominals* semantics, $i \in F^{\mathcal{I}}$ and if there exists $j \in p^{\mathcal{I}}$ then $j \in F^{\mathcal{I}}$ since $i = j$; there can only be one element in $p^{\mathcal{I}}$.

- Case 2 - Role fillers partition: $p^{\mathcal{I}}$ is a role fillers partition, then it corresponds to some partition name $p \in \mathcal{P}$ of the form $p = \{R_1, \ldots, R_n\}$ and individuals in $p^{\mathcal{I}}$ satisfy $p^{\mathcal{I}} = (FIL(R_1) \cap \ldots \cap FIL(R_n))$. If $i, j \in p^{\mathcal{I}}$ then $i, j \in (FIL(R_1) \cap$

88

$\dots \cap FIL(R_n))$; assume $i \notin (FIL(R_1) \cap \dots \cap FIL(R_n))$ then i is a nominal or an R_x-filler for some $x > n$. However i cannot be a nominal or a member of a *qualifying concept* since $p \cap N_o = \emptyset$, and $p \cap N_Q = \emptyset$. Without loss of generality, assuming $i \in FIL(R_1)$ but $i \notin (FIL(R_2) \cap \dots \cap FIL(R_n))$ this means that i belongs to a partition $p'^{\mathcal{I}}$ corresponding to some partition name $p' \in \mathcal{P}$ such that $R_1 \in p'$ and $\{R_2\}, \dots, \{R_n\} \not\subseteq p'$. Now we have p' different from p with $i \in (p \cap p')$, this is a contradiction since partitions are disjoint. Therefore, $i \in (FIL(R_1) \cap \dots \cap FIL(R_n))$, and by analogy we prove that $j \in (FIL(R_1) \cap \dots \cap FIL(R_n))$. Therefore both i and j must satisfy the signature F such that $F^{\mathcal{I}} = \cap_{\forall R_1.C_1 \in \mathcal{T}} C_1^{\mathcal{I}} \dots \cap_{\forall R_n.C_n \in \mathcal{T}} C_n^{\mathcal{I}}$, assuming N_o and N_Q are empty.

- Case 3 - Role fillers with qualifications partition: $p^{\mathcal{I}}$ is a role fillers partition with qualifications, then it corresponds to some partition name $p \in \mathcal{P}$ of the form $p = \{q_k, R_l\}$ for some k, l, $1 \leq k, l \leq n$, and individuals in $p^{\mathcal{I}}$ satisfy $p^{\mathcal{I}} = (\bigcap_{1 \leq k \leq n} q_k^{\mathcal{I}} \cap \bigcap_{q \in (N_Q \setminus \{q_1, \dots, q_k\})} \dot{\neg}_Q q^{\mathcal{I}} \cap \bigcap_{1 \leq l \leq n} FIL(R_l))$. If $i, j \in p^{\mathcal{I}}$ then $i, j \in (FIL(R_1) \cap \dots \cap FIL(R_n)) \cap \bigcap_{1 \leq k \leq n} q_k^{\mathcal{I}} \cap \bigcap_{q \in (N_Q \setminus \{q_1, \dots, q_k\})} \dot{\neg}_Q q^{\mathcal{I}}$. Similar to Case 2, we can prove that $i, j \in (FIL(R_1) \cap \dots \cap FIL(R_n))$ and i cannot be a nominal since $p \cap N_o = \emptyset$. Now we need to prove that $i, j \in \bigcap_{1 \leq k \leq n} q_k^{\mathcal{I}} \cap \bigcap_{q \in (N_Q \setminus \{q_1, \dots, q_k\})} \dot{\neg}_Q q^{\mathcal{I}}$.

Let us assume that $i \notin (\bigcap_{1 \leq k \leq n} q_k^{\mathcal{I}} \cap \bigcap_{q \in (N_Q \setminus \{q_1, \dots, q_k\})} \dot{\neg}_Q q^{\mathcal{I}})$ and without loss of generality, let $i \in q_1^{\mathcal{I}}$ but $i \notin (q_2^{\mathcal{I}} \cap \dots \cap q_n^{\mathcal{I}})$ this means that i belongs to a partition $p'^{\mathcal{I}}$ corresponding to some partition name $p' \in \mathcal{P}$ such that $q_1 \in p'$ and $\{q_2\}, \dots, \{q_n\} \not\subseteq p'$. Now we have p' different from p with $i \in (p \cap p')$, this is a contradiction since partitions are disjoint. Therefore, $i \in (\bigcap_{1 \leq k \leq n} q_k^{\mathcal{I}} \cap \bigcap_{q \in (N_Q \setminus \{q_1, \dots, q_k\})} \dot{\neg}_Q q^{\mathcal{I}})$, and by analogy we prove that : $j \in (\bigcap_{1 \leq k \leq n} q_k^{\mathcal{I}} \cap \bigcap_{q \in (N_Q \setminus \{q_1, \dots, q_k\})} \dot{\neg}_Q q^{\mathcal{I}})$. Hence, both i and j must satisfy the signature F such that $F^{\mathcal{I}} = \cap_{\forall R_1.C_1 \in \mathcal{T}} C_1^{\mathcal{I}} \dots \cap_{\forall R_n.C_n \in \mathcal{T}} C_n^{\mathcal{I}} \cap (\bigcap_{1 \leq k \leq n} q_k^{\mathcal{I}} \cap \bigcap_{q \in (N_Q \setminus \{q_1, \dots, q_k\})} \dot{\neg}_Q q^{\mathcal{I}})$.

- Case 4 - *Nominals* and role fillers partition: $p^{\mathcal{I}}$ is a role filler partition of *nominals*, then it corresponds to some partition name $p \in \mathcal{P}$ of the form $p = \{o_k, R_l\}$ for some k, l, $1 \leq k, l \leq n$, and individuals in $p^{\mathcal{I}}$ satisfy $p^{\mathcal{I}} = (\bigcap_{1 \leq k \leq n} \{o_k\}^{\mathcal{I}} \cap \bigcap_{1 \leq l \leq n} FIL(R_l))$. Given the *nominals* semantics and similarly to case 1 if there exists $i, j \in p^{\mathcal{I}}$ then $i = j$. The signature F for $p^{\mathcal{I}}$ is such that it satisfies $F^{\mathcal{I}} = \bigcap_{1 \leq k \leq n} \{o_k\}^{\mathcal{I}} \cap \bigcap_{\forall R_1.C \in \mathcal{T}} C^{\mathcal{I}}$.

- Case 5 - *Nominals* and role fillers partition with qualifications: this case can be reduced to case 4 where additionally *nominals* satisfy the qualifications.

There is no need to consider the cases when a partition is for individuals with qualifications without being role fillers since these cases do not occur. A qualification is only applicable on a role filler as defined by the semantics of the language. ∎

4.5   The Algebraïc Tableau Algorithm for $\mathcal{SHON}_{\mathcal{R}\backslash}$

This section describes an algebraïc tableau algorithm which decides the existence of a tableau for a $\mathcal{SHON}_{\mathcal{R}\backslash}$ TBox $\mathcal{T}$. The algorithm is hybrid because it relies on tableau completion rules working together with an in-equation solver to construct a tableau as an abstraction of a model of $\mathcal{T}$. Tableau completion rules work in such a way to (1) decide the satisfiability of concept descriptions that use propositional operators ($\sqcap, \sqcup, \neg$) and $\forall$, $\forall_\backslash$ operators, (2) encode numerical restrictions on *nominals*, role fillers, and their qualifications into a set of in-equations processed by an in-equation solver, and (3) make sure that a numerical solution satisfies logical restrictions by constructing a pre-model of the solution. The pre-model is represented using a *compressed completion graph* (CCG).

Definition 4.5.1 (Compressed Completion Graph) The *compressed comple-tion graph* (CCG) is different from the "so-called" completion graph (introduced in Section 2.2.1) used in standard tableau algorithms for $\mathcal{SHOQ}$ [HS01] and is defined as follows.

- A (CCG) is a directed graph $G = (V, E, \mathcal{L}, \mathcal{L}_\mathrm{E}, \mathcal{L}_\mathrm{P})$. Where nodes represent domain elements and the edges between the nodes represent role relations. Each node $x \in V$ is labeled with three labels: $\mathcal{L}(x)$, $\mathcal{L}_E(x)$ and $\mathcal{L}_P(x)$, and each edge $\langle x, y \rangle \in E$ is labeled with a set, $\mathcal{L}(\langle x, y \rangle) \subseteq N_\mathrm{R}$, of role names.

  - $\mathcal{L}(x)$ denotes a set of concept expressions, $\mathcal{L}(x) \subseteq clos(\mathcal{T})$, that the domain element, i_x, represented by x must satisfy.

  - $\mathcal{L}_P(x)$ denotes a partition name and is used as a tag for x based on the partition that i_x belongs to. A partition name can include role names, nominals or qualification names $\mathcal{L}_P(x) \subseteq \mathcal{DS}$.

    * When a role $R \in N_\mathrm{R}$ appears in $\mathcal{L}_P(x)$ this means that i_x belongs to the partition representing the set of *R-fillers*. The node x can therefore be used as an R-successor. When an R-successor is needed for a node y, x is checked as a candidate (see *e*-Rule).

    * When a nominal $o \in N_\mathrm{o}$ appears in $\mathcal{L}_P(x)$ this means that $i_x \in o^\mathcal{I}$, and $\{o\}$ is added to $\mathcal{L}(x)$ when x is created. On the other hand if a nominal $o \in N_\mathrm{o}$ does not appear in $\mathcal{L}_P(x)$ this means that i_x satisfies the complement of $\{o\}$, $i_x \in (\neg\{i\})^\mathcal{I}$ and $(\neg\{o\})$ is added to $\mathcal{L}(x)$ when x is created (see *fil*-Rule).

    * When a qualification name $q \in N_\mathrm{Q}$ appears in $\mathcal{L}_P(x)$ this means that i_x satisfies the qualifying concept mapped to q, $i_x \in q^\mathcal{I}$ and q is added to $\mathcal{L}(x)$ when x is created. As with the *nominals* case, if a qualification

name $p \in N_Q$ does not appear in $\mathcal{L}_P(x)$ this means that i_x satisfies the complement of the qualifying concept mapped to p, $i_x \in \dot{\neg}_Q(p)^{\mathcal{I}}$ and $\dot{\neg}_Q p$ is added to $\mathcal{L}(x)$ when x is created (see *fil*-Rule).

– $\mathcal{L}_E(x)$ denotes a set ξ_x of in-equations that must have a non-negative integer solution. The set ξ_x is the encoding of number restrictions, qualifications and *nominals* (as defined in Section 4.5.1) that must be satisfied for x. In order to make sure that local numerical restrictions for a node x are satisfied while the global restrictions carried with *nominals* are not violated, the in-equation solver collects all in-equations and variable assignment in $\mathcal{L}_E$ before returning a distribution. This makes sure that an initial distribution of *nominals* and/or role fillers is globally preserved while still satisfying the numerical restrictions (a distribution of role fillers) for each node in the completion graph. $\mathcal{L}_E(x)$ also contains a set of in-equations with one variable such that $\mathcal{L}_E(x)$ can be extended with $v \geq 1$ and $v \leq 0$. This form of in-equations is used to denote a range of values for variables, (i.e., a variable can have the value zero or at-least 1) as is done by the *ch*-Rule.

• There is no distinction between nodes having a nominal in their label and other nodes.

• The CCG relies on the use of proxy nodes (see Definition 4.5.2) as representatives for domain elements distributed over the same partition. The use of proxy nodes was first introduced in [HM01a].

• Using $\mathcal{L}_P(x)$ as a tagging allows for the re-use of existing nodes instead of creating new ones. For example if the roles R, S appear in $\mathcal{L}_P(x)$ then x can be used as an R-successor and then re-used as an S-successor or vice versa.

- No blocking strategies are implemented and no merging of existing nodes is possible. Termination is a natural consequence of the re-use of nodes.

- An in-equation solver collects and checks the satisfiability of numerical restrictions.

Definition 4.5.2 (Proxy Node) A proxy node is a representative for the elements of each partition. Proxy nodes can be used due to Lemma 4.5.4 since partitions are disjoint and all elements within a partition P satisfy P's signature.

4.5.1 Satisfying Numerical Restrictions Using Algebraïc Reasoning

Given a partitioning $\mathcal{P}$ for the decomposition set $\mathcal{DS} = (N_{R'} \cup N_o \cup N_Q)$ for $\mathcal{T}$, one can reduce a conjunction of $(\geq nR)$ and $(\leq mR)$ in $\mathcal{L}(x)$ to a set of in-equations and check their satisfiability using an in-equation solver based on the following principles.

P0: Mapping Cardinalities to Variables. We assign a variable name v for each partition name P such that v can be mapped to a non-negative integer value n using $\sigma : \mathcal{V} \longrightarrow \mathbb{N}$ such that $\sigma(v)$ denotes the cardinality of $P^{\mathcal{I}}$. Let $\mathcal{V}$ be the set of all variable names and $\alpha \colon \mathcal{V} \longrightarrow \mathcal{P}$ be a one-to-one mapping between each partition name $P \in \mathcal{P}$ and a variable $v \in \mathcal{V}$ such that $\alpha(v) = P$, and if a non-negative integer n is assigned to v using σ then $\sigma(v) = n = \#P^{\mathcal{I}}$. Given $L \subseteq \mathcal{DS}$, let V_L denote the set of variable names each mapped to a partition p such that $L \subseteq p$, V_L is defined as

$$
V_L = \left(
\begin{array}{l}
\{v \in \mathcal{V} \,|\, p \in \alpha(v) \text{ for each } p \in (L \cap N_{\mathrm{R}})\} \cap \\[4pt]
\{v \in \mathcal{V} \,|\, oq \in \alpha(v) \text{ for each } oq \in (L \cap (N_o \cup N_{\mathrm{Q}}))\} \cap \\[4pt]
\{v \in \mathcal{V} \,|\, oq \notin \alpha(v) \text{ for each } \neg oq \in (L \cap (N_o \cup N_{\mathrm{Q}}))\}
\end{array}
\right)
$$

P1: Encoding Number Restrictions, Qualifications and Nominals Into In-equations. Since the partitions in $\mathcal{P}$ are mutually disjoint and the cardinality function, of disjoint partitions, is additive one can encode a cardinality restriction on partition's elements using ξ such that $\xi(L, \geq, n) = v_1 + \cdots + v_k \geq n$, and $\xi(L, \leq, m) = v_1 + \cdots + v_k \leq m$ where $\{v_1, \ldots, v_k\} \subseteq V_L$ and $L \subseteq \mathcal{DS}$. Hence, a lower (upper) bound on the cardinality of the set of domain elements distributed over the partitions in $\mathcal{P}$ can be encoded into in-equations as follows:

- (i) Bounds on role fillers: concepts of the form $(\geq nR)$ and $(\leq mR)$ in the label of a node x express lower and upper bounds n and m, respectively, on the cardinality of the set $FIL(R, i_x)$ for some $R \in N_{\mathrm{R}}$. These bounds can be reduced into in-equations using $\xi(L, \geq, n)$ and $\xi(L, \leq, m)$ for $L = \{R\}$ or $L = \{R, q\}$, if additionally we have $\forall S.C$ such that $(R \sqsubseteq_* S)$ with $C \in DS$ and $q = C$. Consider a node x in a CCG G, such that x is labelled with the concept expression from Example 4.1.4, $\mathcal{L}(x) = \{\geq 1R_1, \forall R_1.(\{o\} \sqcap \leq 1R_2 \sqcap \forall R_2.\{o\} \sqcap \forall R \backslash R_2.\neg\{o\})\}$, then the bounds on $FIL(R_1, i_x)$ are encoded into in-equation (25). Assuming another node, y, in G such that $\mathcal{L}(y) = \{\{o\}, \leq 1R_2, \forall R_2.\{o\}, \forall R \backslash R_2.\neg\{o\}\}$, then the bounds on $FIL(R_2)$ are encoded into in-equation (26). In both in-equations, every variable v, mapped to a partition p, is indexed with the names of the elements forming p.

- (ii) Bounds imposed by *nominals*: *Nominals* carry cardinality restrictions; they not only name individuals but also allow for counting individuals. Therefore, the cardinality of a partition with a nominal o can only be equal to 1 based on the *nominals* semantics; $\#\{o\}^{\mathcal{I}} = 1$. This bound on the cardinality of the *nominals* partitions can be encoded into in-equations using $\xi(\{o\}, \geq, 1)$ and $\xi(\{o\}, \leq, 1)$ for each nominal $o \in N_{\mathrm{o}}$. In the case of Example (4.1.4) then the *nominals* semantics is encoded into in-equations (27) and (28).

$$v_{R_1} + v_{R_1o} + v_{R_1R_2} + v_{R_1R_2o} \geq 1 \tag{25}$$

$$v_{R_2o} + v_{R_1R_2o} \leq 1 \tag{26}$$

$$v_o + v_{R_1o} + v_{R_2o} + v_{R_1R_2o} \geq 1 \tag{27}$$

$$v_o + v_{R_1o} + v_{R_2o} + v_{R_1R_2o} \leq 1 \tag{28}$$

When the *nominals* semantics is encoded into in-equations together with the bounds on role fillers, the interaction between *nominals* and role fillers is handled while preserving that there is one individual for each $o \in N_o$: $\#\{o\}^{\mathcal{I}} = 1$.

P2: Getting a Solution. Given a set ξ_x of in-equations in $\mathcal{L}_E(x)$, an integer solution defines the mapping σ for each variable v occurring in ξ_x to a non-negative integer n denoting the cardinality of the corresponding partition. For example, assuming $\sigma(v_{\{R_1,R_2\}}) = 1$ and $\alpha(v_{\{R_1,R_2\}}) = \{R_1, R_2\}$, this means that the corresponding partition $(\alpha(v_{\{R_1,R_2\}}))^{\mathcal{I}}$ must have 1 element; $\#(FIL(R_1) \cap FIL(R_2)) = 1$. Additionally, it is desirable sometimes to minimize the sum of all variables in order to ensure a minimum number of role fillers at each level. A solution defining σ, then defines a distribution of individuals that is consistent with the numerical restrictions encoded in ξ_x and the hierarchy expressed in $\mathcal{R}$. Getting a solution for ξ_x can be considered as an Integer Linear Programming problem [Dan63], which is the problem of maximizing or minimizing a linear function over a convex polyhedron specified by linear and non-negativity constraints, and ξ_x represents an IP model.

Definition 4.5.3 (IP Model) An IP model consists of an objective function that needs to be optimized subject to a set of linear constraints on that function, and is considered a special type of Linear Programming (LP) problems with additionally

constraining the values of all variables to integer values. An example of a an LP problem is shown below:

Minimize

$$Z = v_1 - v_2 + a_1 v_3 - a_2 v_4$$

subject to the constraints:

$$v_1 + v_2 + b_1 v_3 + b_3 v_4 = c_1$$
$$b_4 v_2 + v_3 + b_5 v_4 \leq c_2$$

Where Z is the objective function whose value needs to be optimized, v_i are the variables whose optimal values need to found w.r.t the set of constraints consisting of linear in-equations, and a_i, b_i, c_i are constants derived from the specification of the LP problem. When encoding number restrictions with QCRs and *nominals* into an ILP problem, a_i and b_i can take only the values of $(0, 1)$. For example, if v_i has been assigned $v_i \geq 1$ by the *ch*-Rule then b_i is set to 1, otherwise b_i is set to 0. All a_i are set to 1 because the objective function is to minimize the variables occurring in the linear constraints. Also, c_i are derived from the numbers used in cardinality restrictions. For example, in the case of encoding a nominal's semantics into a linear constraint, c_i is always equal to 1 (otherwise it is equal to a non-negative integer number).

Integer Programming (IP) problems can be solved using the widely known Simplex [CLRS01] method for LP, extended with the branch and bound technique to solve the integer constraints. Branch and bound (also known as branch and cut) complements the Simplex method and works in a divide-and-conquer strategy to find an integer solution. For instance, if a non-integer solution is returned by the Simplex method such that a variable is assigned the value $v = 0.66$, then branch and bound would try to find a solution by trying $v = 0$ and $v = 1$. A minimal solution is desired, in a sense where less variables are assigned the values of ≥ 1, in order to keep the

completion model of smaller size thus allowing less expansion rules to become applicable. However, a less optimal solution does not affect the correctness of algebraïc DL reasoning, which relies on IP mainly to decide the satisfiability or unsatisfiability of the numerical restrictions imposed by *nominals* and QCRs. In the scope of algebraïc reasoning for DL, a solution does not necessarily need to be optimal, and variants of the Branch and bound technique, which do not always consider the optimal solution for sake of finding a solution quicker, can be considered.

Lemma 4.5.4 (Using a Proxy Individual) *Given a graph G as a representation of a model $\mathcal{I}$ for a TBox $\mathcal{T}$, P a non-empty partition in $\mathcal{P}^{\mathcal{I}}$, and n a non-negative integer assigned by the in-equation solver such that $n = \#P$. It is sufficient to create one proxy node in G as a representative of the n individuals in P.*

Proof. Lemma 4.5.4 is an easy consequence of Lemma 4.4.5. Creating a proxy node x for P in G allows to test the satisfiability of P's signature (see Section 4.4.2). If x satisfies the signature, then m elements can also satisfy it and m is decided by the in-equation solver. x cannot violate cardinality bounds on role fillers and *nominals* since these bounds are numerically satisfied by the in-equation solver. However, if x does not satisfy the signature of P due to a clash, this means that P must be empty because its signature is unsatisfiable. ∎

4.5.2 Deciding KB Consistency

<div style="border:1px solid">

⊓-Rule   **If** $C \sqcap D \in \mathcal{L}(x)$, and $\{C, D\} \nsubseteq \mathcal{L}(x)$

           Then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C, D\}$

⊔-Rule   **If** $C \sqcup D \in \mathcal{L}(x)$, and $\{C, D\} \cap \mathcal{L}(x) = \emptyset$

           Then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{E\}$ with $E \in \{C, D\}$

∀-Rule   **If** $\forall R.C \in \mathcal{L}(x)$ and there exists y such that $\mathcal{L}(\langle x, y \rangle) \cap (H(R) \cup \{R\}) \neq \emptyset$,

           and $C \notin \mathcal{L}(y)$

           Then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

∀₊-Rule   **If** $\forall R.C \in \mathcal{L}(x)$ and there exists y such that $\mathcal{L}(\langle x, y \rangle) \cap (H(S) \cup \{S\}) \neq \emptyset$,

           $S \in N_{R^+}$ with $S \sqsubseteq_* R$, and $\forall S.C \notin \mathcal{L}(y)$

           Then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{\forall S.C\}$

</div>

Figure 23: Completion rules for $\mathcal{SHON}_{\mathcal{R}\backslash}$ - Part I.

Recall that one can decide the consistency of $\mathrm{KB}(\mathcal{T}, \mathcal{R})$ by checking the consistency of $C'_{\mathcal{T}}$ using $i \in N_o$ new in $\mathcal{T}$ such that $i^{\mathcal{I}} \in C'^{\mathcal{I}}_{\mathcal{T}}$ and every new individual satisfies $C'_{\mathcal{T}}$. The algorithm starts with the CCG $G = (\{r_0\}, \emptyset, \emptyset, \mathcal{L}_{\mathrm{E}}, \emptyset)$. With $\mathcal{L}_{\mathrm{E}}(r_o) = \bigcup_{o \in N_o} \{\xi(\{o\}, \leq, 1), \xi(\{o\}, \geq, 1)\}$ which is an encoding of the *nominals* semantics.

The node r_0 is artificial and is not considered as part of the pre-model, it is only used to process the numerical restrictions on *nominals* using the in-equation solver which returns a distribution for them. The distribution of *nominals* (solution) is processed by the *fil*-Rule (see Figure 24) which, based on a non-deterministic distribution of *nominals*, initializes the proxy nodes for *nominals*.

$\forall_{\backslash}$-Rule   **If** $\forall(R\backslash S).C \in \mathcal{L}(x)$, and there exists y such that:

$\mathcal{L}(\langle x,y\rangle) \cap (H(R) \cup \{R\}) \neq \emptyset$, $\mathcal{L}(\langle x,y\rangle) \cap (H(S) \cup \{S\}) = \emptyset$, and $C \notin \mathcal{L}(y)$

Then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

$\bowtie$-Rule   **If** $(\bowtie nR) \in \mathcal{L}(x)$ for $\bowtie \in \{\leq, \geq\}$,

Then If $\forall S.C \in \mathcal{L}(x)$ with $R \sqsubseteq_* S$ and $\xi(\{R,C\}, \bowtie, n) \notin \mathcal{L}_{\mathrm{E}}(x)$

Then set $\mathcal{L}_{\mathrm{E}}(x) = \mathcal{L}_{\mathrm{E}}(x) \cup \{\xi(\{R,C\}, \bowtie, n)\}$

Else If $\xi(\{R\}, \bowtie, n) \notin \mathcal{L}_{\mathrm{E}}(x)$ **Then** set $\mathcal{L}_{\mathrm{E}}(x) = \mathcal{L}_{\mathrm{E}}(x) \cup \{\xi(\{R\}, \bowtie, n)\}$

ch-Rule   **If** there exists v occurring in $\mathcal{L}_{\mathrm{E}}(x)$ such that $\{v \geq 1, v \leq 0\} \cap \mathcal{L}_{\mathrm{E}}(x) = \emptyset$

Then set $\mathcal{L}_{\mathrm{E}}(x) = \mathcal{L}_{\mathrm{E}}(x) \cup \{V\}$, $V \in \{v \geq 1, v \leq 0\}$, and

set $\mathcal{L}_{\mathrm{E}}(y) = \mathcal{L}_{\mathrm{E}}(y) \cup \{V\}$, for all nodes y in G such that v occurs in $\mathcal{L}_E(y)$

fil-Rule   **If** there exists v occurring in $\mathcal{L}_{\mathrm{E}}(x)$ with $\sigma(v) = m$ and $m > 0$, and

there exists no y with $\mathcal{L}_P(y) = \alpha(v)$

Then 1. create a new node y, 2. set $\mathcal{L}_P(y) = \alpha(v)$, 3. set $\mathcal{L}_{\mathrm{E}}(y) = \mathcal{L}_{\mathrm{E}}(x)$,

4. set $\mathcal{L}(y) = \left(\begin{array}{l} \{C'_{\mathcal{T}}\} \cup \bigcup_{o \in (\alpha(v) \cap N_o)} o \cup \bigcup_{i \in (N_o \backslash \alpha(v))} \neg i \ \cup \\ \bigcup_{q \in (\alpha(v) \cap N_{\mathrm{Q}})} q \cup \bigcup_{p \in (N_{\mathrm{Q}} \backslash \alpha(v))} \dot{\neg}_Q p \end{array} \right)$

e-Rule   **If** $(\bowtie nR) \in \mathcal{L}(x)$, and there exists y such that $R \in \mathcal{L}_P(y)$ and $R \notin \mathcal{L}(\langle x,y\rangle)$

Then If $\forall S.C \in \mathcal{L}(x)$ with $R \sqsubseteq_* S$ and $C \in \mathcal{L}_P(y)$, or

$\forall S.C \notin \mathcal{L}(x)$ with $R \sqsubseteq_* S$

Then set $\mathcal{L}(\langle x,y\rangle) = \mathcal{L}(\langle x,y\rangle) \cup \{R\}$, and

If $\mathcal{L}_{\mathrm{E}}(x) \nsubseteq \mathcal{L}_{\mathrm{E}}(y)$ **Then** set $\mathcal{L}_{\mathrm{E}}(y) = \mathcal{L}_{\mathrm{E}}(y) \cup \mathcal{L}_{\mathrm{E}}(x)$

Figure 24: Completion rules for $\mathcal{SHON}_{\mathcal{R}\backslash}$ - Part II.

After at least one nominal is created, G is expanded by applying the completion

rules given in Figures 23 and 24 until no more rules are applicable or when a clash occurs (see Section 4.5.4 for an explanation of the rules). No clash triggers or rules other than the *fil*-Rule apply to r_o. When no rules are applicable or there is a clash, a CCG is said to be *complete*.

Definition 4.5.5 (Clash)   A node x in $(V \setminus \{r_0\})$ is said to contain a *clash* if:

- (i) $\{C, \neg C\} \subseteq \mathcal{L}(x)$, or

- (ii) a subset of in-equations $\xi_x \subseteq \mathcal{L}_E(x)$ does not admit a non-negative integer solution.

When G is complete and there is no clash, this means that the numerical as well as the logical restrictions are satisfied ($C_{\mathcal{T}}'^{\mathcal{I}} \neq \emptyset$) and there exists a pre-model for $\mathcal{T}$: the algorithm returns that $\mathcal{T}$ is consistent. Otherwise the algorithm returns that $\mathcal{T}$ is inconsistent.

4.5.3   Strategy of Rule Application

Given a node x in the CCG, the completion rules in Figures 23 and 24 are applicable based on the following priorities:

- **Priority 1:** $\sqcap$-*Rule*, $\sqcup$-*Rule*, $\forall$-*Rule*, $\forall_+$-*Rule*, *ch-Rule*, $\bowtie$-*Rule*, *e-Rule*.

- **Priority 2:** *fil-Rule*.

- **Priority 3:** $\forall_\setminus$-*Rule*.

The rules with Priority 1 can be fired in arbitrary order. The *fil-Rule* has Priority 2 to ensure that all at-least and at-most restrictions for a node x are encoded and

satisfied by the in-equation solver before creating any new nodes. This justifies why role fillers or *nominals* are never merged nor removed from G; a distribution of role fillers and *nominals* either survives into a complete model or fails due to a clash. Also, assigning the *fil-Rule* Priority 2 helps in early clash detection in the case when the in-equation solver detects a numerical clash even before new nodes are created. The $\forall_\backslash$-*Rule* has Priority 3 to ensure that the semantics of the $\forall_\backslash$ operator are not violated. We enforce the creation of all possible edges between a node and its successors before applying the $\forall_\backslash$ operator semantics. This rule priority is needed to ensure the completeness (see Lemma 4.6.3) of the algorithm.

4.5.4 Explaining the Rules

The $\sqcap$-Rule, $\sqcup$-Rule, $\forall$-Rule and the $\forall_+$-Rule in Figure 23 are similar to the ones introduced in Section 2.2.1.

$\forall_\backslash$-**Rule**. This rule is used to enforce the semantics of the role set difference operator $\forall_\backslash$ introduced at preprocessing. Given a node x, this rule makes sure that all R-successors of x that are not also S-successors of x are labelled. Together with the *ch*-Rule (see explanation below), this rule has the same effect as the *choose*-rule, introduced in Section 2.2.2 and discussed in Section 3.1.2.1.

$\bowtie$-**Rule.** This rule encodes the numerical restrictions in the label $\mathcal{L}$ of a node x, for some role $R \in N_R$, into a set of in-equations maintained in $\mathcal{L}_E(x)$ (see P1 in Section 4.5.1). An in-equation solver is always active and responsible for finding a non-negative integer solution σ (see P2 in Section 4.5.1) or triggering a clash if no solution is possible. If the in-equations added by this rule do not trigger a clash, then the encoded at-least/at-most restriction can be satisfied by a possible distribution of role fillers. We distinguish two cases:

- Case (i): R-fillers of x must also satisfy a qualified restriction C due to a

101

$\forall S.C$ restriction on a role S such that $R \sqsubseteq_* S$ and C is either a nominal or a qualification name in $\mathcal{DS}$. Then the numerical restriction is encoded on partitions $P \in \mathcal{P}$ with $P^{\mathcal{I}} \subseteq (C^{\mathcal{I}} \cap FIL(R))$ which means $\{R, C\} \subseteq P$.

- Case (ii): There exist no qualified restrictions on R-fillers of x due to a $\forall$ restriction on a role S such that $R \sqsubseteq_* S$. In this case the numerical restriction is encoded on partitions $P \in \mathcal{P}$ with $P^{\mathcal{I}} \subseteq FIL(R)$ which means $\{R\} \subseteq P$

Unlike in [FH10a, FHM09, FFHM08b], a distinction needs to be made between case (i) and case (ii) in order to preserve completeness of the algorithm. Otherwise given two nodes x and y in G such that $\{\geq 1R, \forall S.C\} \subseteq \mathcal{L}(x)$, and $\{\geq 1R, \forall S.\dot{\neg}_Q C\} \subseteq \mathcal{L}(y)$ with $R \sqsubseteq_* S \in R$, then the encoded in-equations in $\mathcal{L}_E(x)$ and $\mathcal{L}_E(y)$ rely on variables for partitions $P \in \mathcal{P}$ such that $P^{\mathcal{I}} \subseteq FIL(R)$ and the qualifications imposed by $\forall S.C$ and $\forall S.\dot{\neg}C$ are lost because then one would have $FIL(R, x) \equiv FIL(R, y)$ whereas $FIL(R, x) \subseteq C^{\mathcal{I}}$ and $FIL(R, y) \subseteq (\dot{\neg}_Q C)^{\mathcal{I}}$. See Section 4.2.3 for an illustration of encapsulated qualifications which motivate the use of *qualifying concepts*.

ch-**Rule.** This rule checks for empty partitions while ensuring completeness of the algorithm. Given a set of in-equations in the label $(\mathcal{L}_E)$ of a node x and a variable v such that $\alpha(v) = P$ and $P \in \mathcal{P}$ we distinguish between two cases:

- (i) The case when $P^{\mathcal{I}}$ must be empty $(v \leq 0)$; this happens when restrictions on elements of this partition trigger a clash because the signature of P cannot be satisfied. For instance, if $\{\forall R_1.A, \forall R_2.\neg A\} \subseteq \mathcal{L}(x)$, $v_{R_1 R_2} \geq 1 \in \mathcal{L}_E(x)$ and there exists a node y with $\mathcal{L}_P(y) = \{R_1, R_2\}$ and $\{R_1, R_2\} \subseteq \mathcal{L}(\langle x, y \rangle)$ the qualifications on R_1 and R_2-fillers trigger a clash $\{A, \neg A\} \subseteq \mathcal{L}(y)$ and $v_{R_1 R_2} \leq 0$ is enforced.

- (ii) The case when $P^{\mathcal{I}}$ must have at least one element $(1 \leq m \leq \sigma(v))$; if $P^{\mathcal{I}}$

102

can have at least one element without causing any clash, this means that the signature of P is satisfiable and we can have m elements also in $P^{\mathcal{I}}$ without a clash.

Since the in-equation solver is unaware of partition signatures imposing restrictions on role fillers, an explicit distinction between cases (i) and (ii) is needed. This distinction is done by non-deterministically assigning ≤ 0 or ≥ 1 for each variable v occurring in $\mathcal{L}_{\mathrm{E}}(x)$. The *ch*-Rule needs only fire once for each variable v. However, v can also occur within the label $\mathcal{L}_E$ of a node y. In order to avoid the applicability of the *ch*-Rule to a node y with v, after the *ch*-rule is applicable on the node x for v, the variable choices are propagated to all nodes y such that $v \in \mathcal{L}_E(y)$.

fil-**Rule**. This rule is used to generate proxy nodes depending on the distribution (σ) returned by the in-equation solver. The rule is fired for every non-empty partition P based on $\sigma(v)$. It generates one proxy node y as the representative for the m elements assigned to $P^{\mathcal{I}}$ by the in-equation solver. The node y is tagged with its partition name using $\alpha(v)$ in $\mathcal{L}_P(y)$. The set of in-equations is accumulated in $\mathcal{L}_E(y)$. *Nominals* and qualifications satisfied by the partition elements are extracted from the partition name and added to $\mathcal{L}(y)$. $C'_{\mathcal{T}}$ is added to $\mathcal{L}(y)$ to ensure that every node created by the *fil*-Rule also satisfies $C'_{\mathcal{T}}$.

e-**Rule.** This rule creates the edges between the proxy nodes created by the *fil*-Rule. If $\geq nR \in \mathcal{L}(x)$, for some R, this means that x must be connected to a number r of R-fillers such that $n \leq r$. If $\leq mR \in \mathcal{L}(x)$ then x could be connected to a maximum number r' of R-fillers such that $r' \leq m$. If there exists a node y such that $R \in \mathcal{L}_P(y)$, this means that a distribution of R-fillers has been assigned by the in-equation solver such that the numbers n and m are satisfied and y is a representative for a number p of R-fillers such that $r \leq p \leq r'$. We distinguish between two cases:

- Case (i): R-fillers of x must also satisfy a qualified restriction C due to a $\forall S.C$

103

restriction on a role S such that $R \sqsubseteq_* S$. In this case, if C is also in $\mathcal{L}_P(y)$ then the partition represented by y intersects with $C^\mathcal{I}$ and y is a member of C.

- Case (ii): There exists no qualified restrictions on R-fillers of x due to a $\forall S.C$ restriction on a role S such that $R \sqsubseteq_* S$. In this case there is no restriction on the partitions intersecting with R-fillers.

In both cases, an edge can safely be created between x and y such that $R \in \mathcal{L}(\langle x, y \rangle)$ and this edge is also a representative for the number p of edges between x and the p elements represented by y. If S is also in $\mathcal{L}_P(y)$ this means that the p R-fillers represented by y are also S-fillers and y is a representative for a partition $p \in \mathcal{P}$ such that $p^\mathcal{I} \subseteq FIL(R) \cap FIL(S)$. Therefore y can be re-used to connect x or another node y having $\geq n'S$ or $\leq m'S$, $n' \leq n$ and $m' \geq m$, in their label. In the case where $n = 0$ or $m = 0$ the CCG will not have any nodes representing the corresponding role fillers, because the in-equation solver will not assign a distribution of fillers, and the e-Rule will not fire. One might argue that the e-Rule does not need to fire for $\leq mR \in \mathcal{L}(x)$. However, if we have a node x with $\{\geq 1R_1, \forall R_1.C, \leq 1R_2, \forall R_2.C, \forall R \setminus R_2.\neg C\} \subseteq \mathcal{L}(x)$ with $R_1 \sqsubseteq R$, and $R_2 \sqsubseteq R$ and a node y such that $\mathcal{L}_P(y) = \{R_1, R_2\}$, then if the e-Rule only fires for $\geq 1R_1$ then the edge created between x and y will satisfy only $R_1 \in \mathcal{L}(\langle x, y \rangle)$ and the $\forall_\setminus$-Rule propagates $\neg C$ to y leading to a clash making the algorithm incomplete because y has also been assigned as an R_2-filler.

4.5.5 Example

To better illustrate the calculus, we demonstrate it by checking the consistency of the TBox $\mathcal{T}$ from Example 4.2.1 which we adapt to include cycles as shown in Figure 25. $\mathcal{T}$ contains cyclic descriptions ($A \sqsubseteq \geq 1R.A$), *nominals* ($\{o\}$) and numerical restrictions ($\geq 1R.A$) with *qualifying concepts* ($\forall R.C$), and can be used to highlight some

of the strong features (see Section 4.7) of the algebraïc tableau algorithm presented in this chapter.

A	$\sqsubseteq$	$\geq 1R.A$
B_1	$\sqsubseteq$	$\forall R.C$
B_2	$\sqsubseteq$	$\forall R.\neg C$
$\{o\}$	$\sqsubseteq$	$\geq 1S_1.(A \sqcap B_1) \sqcap \geq 1S_2.(A \sqcap B_2))$

(a) TBox axioms in $\mathcal{T}$.

(b) Initial role hierarchy in $\mathcal{R}$.
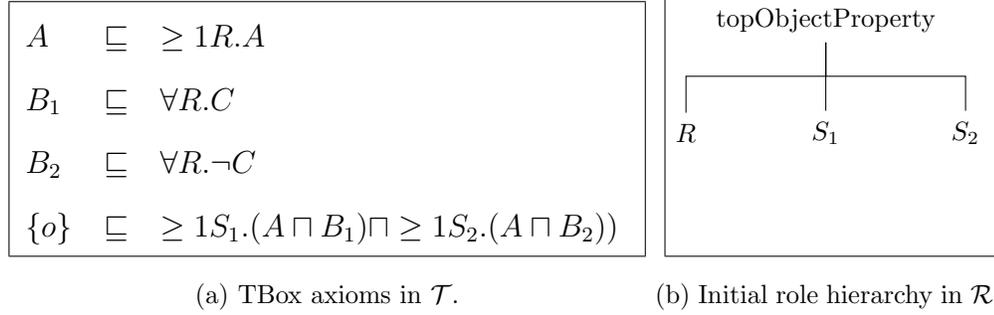
Figure 25: Example TBox with cycles, *nominals*, and *qualifying concepts*.

The set of *nominals* referenced in concept descriptions consists of

$$N_\text{o} = \{o\} \tag{29}$$

the set of *qualifying concepts* consists of

$$N_\text{Q} = \{C\} \tag{30}$$

and the set of role names used in number restrictïons consists of

$$N_\text{R} = \{R, S_1, S_2\} \tag{31}$$

such that the hierarchy between the roles is as shown in Figure 25b.

Algorithm 4.1.1 rewrites $C_\mathcal{T}$, as was illustrated for Example 4.2.1, into $C'_\mathcal{T}$ as shown in Figure 26a, and extends the role hierarchy in Figure 25b with the newly introduced roles as shown in Figure 26b. The consistency of $(\mathcal{T}, \mathcal{R})$, is reduced to checking the consistency of $(i \sqsubseteq C'_\mathcal{T}$ with $i \in N_\text{o}$ new in $\mathcal{T})$. In order to apply the

atomic decomposition, the different sets which are used to build the decomposition set $\mathcal{DS}$ are identified in Table 3.

$$C_{\mathcal{T}}' = \quad \neg A \sqcup (\geq 1R_1 \sqcap \forall R_1.A) \sqcap$$
$$\neg B_1 \sqcup \forall R.C \sqcap$$
$$\neg B_2 \sqcup \forall R.\neg C \sqcap$$
$$\neg\{o\} \sqcup (\geq 1S_{11} \sqcap \forall S_{11}.(A \sqcap B_1) \sqcap \geq 1S_{21} \sqcap \forall S_{21}.(A \sqcap B_2))$$

(a) Internalization of $\mathcal{T}$ into $C_{\mathcal{T}}'$ after preprocessing $C_{\mathcal{T}}$.

(b) Extended hierarchy.
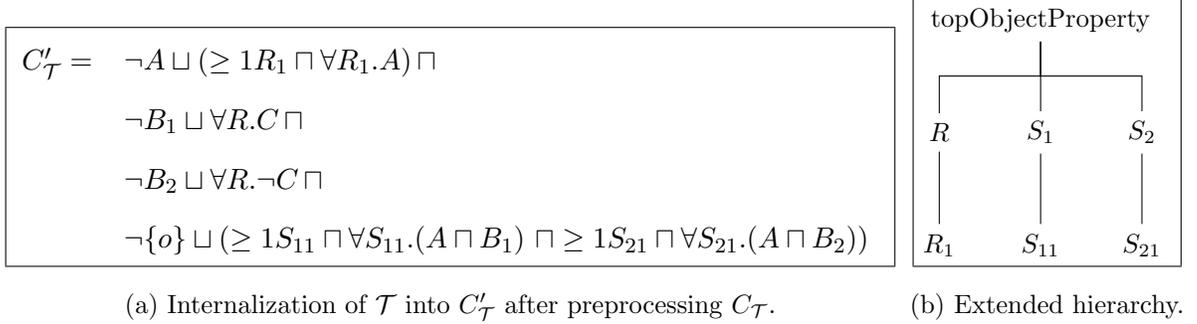
Figure 26: TBox internalization into $C_{\mathcal{T}}'$ in $\mathcal{SHON}_{\mathcal{R}\backslash}$.

Roles	Nominals	Qualifying Concepts
$N_{R'} = \{R_1, S_{11}, S_{21}\}$	$N_o = \{o, i\}$	$N_Q = \{C\}$
$H(R) = \{R_1\}$		$Q_C(R) = \{C\}, Q_C^{\neg}(R) = \{\neg C\}$
$H(S_1) = \{S_{11}\}$		$Q_C(S_1) = \emptyset, Q_C^{\neg}(S_1) = \emptyset$
$H(S_2) = \{S_{21}\}$		$Q_C(S_2) = \emptyset, Q_C^{\neg}(S_2) = \emptyset$
$D_R = H(R) \cup Q_C(R) = \{R_1, C\}$		
$D_{S_1} = H(S_1) \cup Q_C(S_1) = \{S_{11}\}$		
$D_{S_2} = H(S_2) \cup Q_C(S_2) = \{S_{21}\}$		

Table 3: Identifying decomposition set elements for $\mathcal{T}$.

The global decomposition set $\mathcal{DS} = \bigcup_{R \in N_R} \mathcal{D}_R \cup N_o$, as defined in Definition 4.4.3, consists of $\mathcal{DS} = \{R_1, C, S_{11}, S_{21}\} \cup \{o, i\}$.[5] The atomic decomposition of $\mathcal{DS}$, as shown in Figure 27,[6] defines the partitioning $\mathcal{P} = \{\{R_1\}, \{S_{11}\}, \{S_{21}\}, \ldots, \{R_1, S_{11}, S_{21}, o, i, C\}\}$ of domain elements. Let $\mathcal{V}$ define the set of variables associated with each partition in $\mathcal{P}$: $\mathcal{V} = \{v_{R_1}, v_{S_{11}}, v_{S_{21}}, \ldots, v_{R_1 S_{11} S_{21} o i C}\}$. The calculus

---

[5]We only include C to $\mathcal{Q}_C$ if C and $\neg C$ are used as *qualifying concepts*.

[6]Some partitions are left unnamed in the figure for better clarity.

starts with the CCG $G = (\{r_0\}, \emptyset, \mathcal{L}, \mathcal{L}_E, \emptyset)$ with $\mathcal{L}_E(r_0) = \{\xi(\{o\}, \geq, 1), \xi(\{o\}, \leq, 1),$ $\xi(\{i\}, \geq, 1), \xi(\{i\}, \leq, 1)\}$ as shown in Figure 28. A distribution of *nominals* is decided by the *ch*-Rule branching on *nominals*' variables. Notice that when the *ch*-Rule assigns all *nominals*' variables ≤ 0, a clash is detected because there is no solution for ξ_{r_0}; the *nominals* semantics is violated.
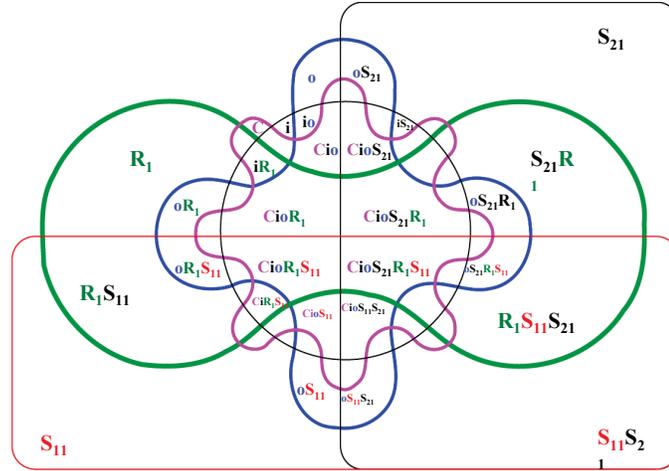
Figure 27: Atomic Decomposition of $\mathcal{DS} = \{R_1, S_{11}, S_{21}, o, i, C\}$.

Consider the following cases of *nominals* distributions:

- Case (a): The *ch*-Rule rule assigns *nominals* variables such that $v_{\mathrm{oi}} \geq 1$ and all other variables ≤ 0. The expansion of a CCG for this case starts as shown in Figure 28 and continues as shown in Figure 29. The final CCG is illustrated in Figure 30.

- Case (b): The *ch*-Rule rule assigns *nominals* variables such that $v_{\mathrm{i}} \geq 1$, $v_{\mathrm{o}} \geq 1$ and all other variables ≤ 0. The CCG for this case is illustrated in Figure 31.

Considering case (a), the expansion of the CCG, by application of the completion rules described in Figures 23 and 24, is illustrated in Figure 28. Once the *ch*-Rule is not applicable anymore, the in-equation solver returns a solution σ such that $\sigma(v_{\mathrm{oi}}) = 1$ and all other variables are set to zero.
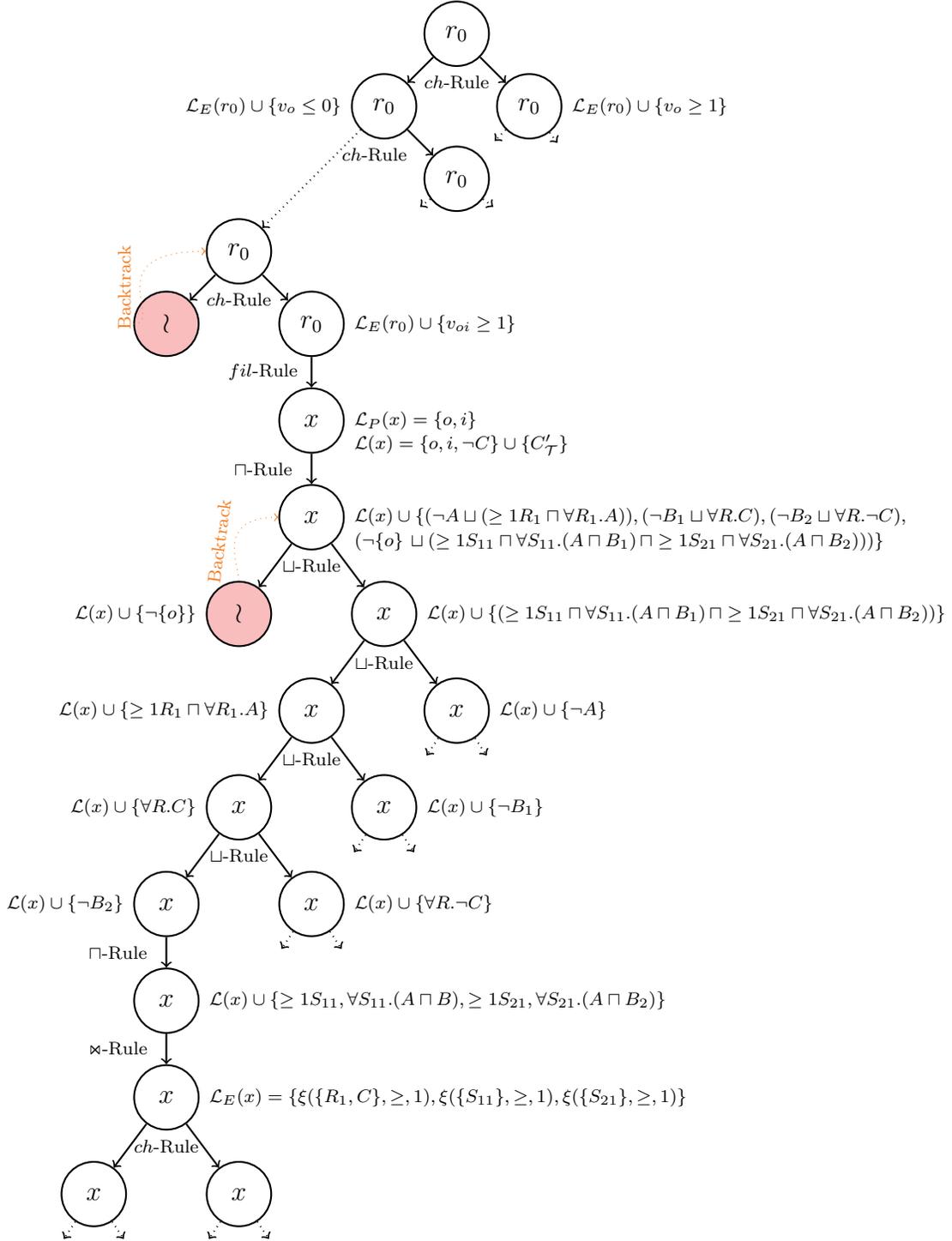
Figure 28: Expansion tree considering a distribution of *nominals* such that $v_{oi} \geq 1$. The expansion continues in Figure 29.

The *fil*-Rule becomes applicable to r_0 and one new node x is created such that $\mathcal{L}_P(x) = \alpha(v_{\mathrm{oi}}) = \{o, i\}$, $\mathcal{L}_{\mathrm{E}}(x) = \mathcal{L}_{\mathrm{E}}(r_0)$, and

$$
\mathcal{L}(x) = \left\{
\begin{array}{l}
\bigcup_{o \in (\alpha(v) \cap N_o)} o \cup \bigcup_{i \in (N_o \setminus \alpha(v))} \neg i \\
\cup \bigcup_{q \in (N_Q \cap \alpha(v))} q \cup \bigcup_{p \in (N_Q \setminus (N_Q \cap \alpha(v)))} \dot{\neg}_Q p \\
\cup \{C'_{\mathcal{T}}\}
\end{array}
\right\} = \{o, i\} \cup \{\neg C\} \cup \{C'_{\mathcal{T}}\}
$$

The $\sqcap$-Rule, $\sqcup$-Rule, and $\bowtie$-Rule become applicable to x such that $\mathcal{L}(x)$ and $\mathcal{L}_E(x)$ are extended to $\mathcal{L}(x) \cup \{o, i, \neg C, \geq 1R_1, \forall R_1.A, \forall R.C, \neg B_2, \geq 1S_{11}, \forall S_{11}.(A \sqcap B_1), \geq 1S_{21}, \forall S_{21}.(A \sqcap B_2)\}$,[7] and $\mathcal{L}_E(x) \cup \{\xi(\{R_1, C\}, \geq, 1), \xi(\{S_{11}\}, \geq, 1), \xi(\{S_{21}\}, \geq, 1)\}$ respectively. The *ch*-Rule becomes applicable to x in order to branch on role fillers partitions' variables. Consider the branching points where $v_{R_1 C}$, $v_{S_{11}}$ and $v_{S_{21}}$ are all ≥ 1 in $\mathcal{L}_E(x)$, and all other applicable variables are set to ≤ 0. The expansion tree is illustrated in Figure 29 which complements the one shown in Figure 28. A solution to the in-equations in $\mathcal{L}_E(x)$ preserving the initial variable assignment in $\mathcal{L}_E(r_0)$ assigns $v_{S_{11}}$, $v_{S_{21}}$, and $v_{R_1 C}$ the value 1. The *fil*-Rule creates the nodes y_1, y_2, and y_3 such that: $\mathcal{L}_P(y_1) = \alpha(v_{S_{11}}) = \{S_{11}\}$, $\mathcal{L}_P(y_2) = \alpha(v_{S_{21}}) = \{S_{21}\}$, $\mathcal{L}_P(y_3) = \alpha(v_{R_1 C}) = \{R_1, C\}$, $\mathcal{L}_E(y_1) = \mathcal{L}_E(y_2) = \mathcal{L}_E(y_3) = \mathcal{L}_E(x)$, $\mathcal{L}(y_1) = \{\neg o, \neg i, \neg C\} \cup \{C'_{\mathcal{T}}\}$, $\mathcal{L}(y_2) = \{\neg o, \neg i, \neg C\} \cup \{C'_{\mathcal{T}}\}$, and $\mathcal{L}(y_3) = \{\neg o, \neg i, C\} \cup \{C'_{\mathcal{T}}\}$.

The *e*-Rule becomes applicable to x three times and the edges $\langle x, y_1 \rangle$, $\langle x, y_2 \rangle$, and $\langle x, y_3 \rangle$ are created such that: $\mathcal{L}(\langle x, y_1 \rangle) = \{S_{11}\}$, $\mathcal{L}(\langle x, y_2 \rangle) = \{S_{21}\}$, and $\mathcal{L}(\langle x, y_3 \rangle) = \{R_1\}$. The $\forall$-Rule enforces the qualifications on role successors such that A is added to $\mathcal{L}(y_3)$, $(A \sqcap B_1)$ is added to $\mathcal{L}(y_1)$, and $(A \sqcap B_2)$ is added to $\mathcal{L}(y_2)$.

---

[7]Considering the left branch of each $\sqcup$-Rule branching points as illustrated in the first part of the CCG shown in Figure 28.
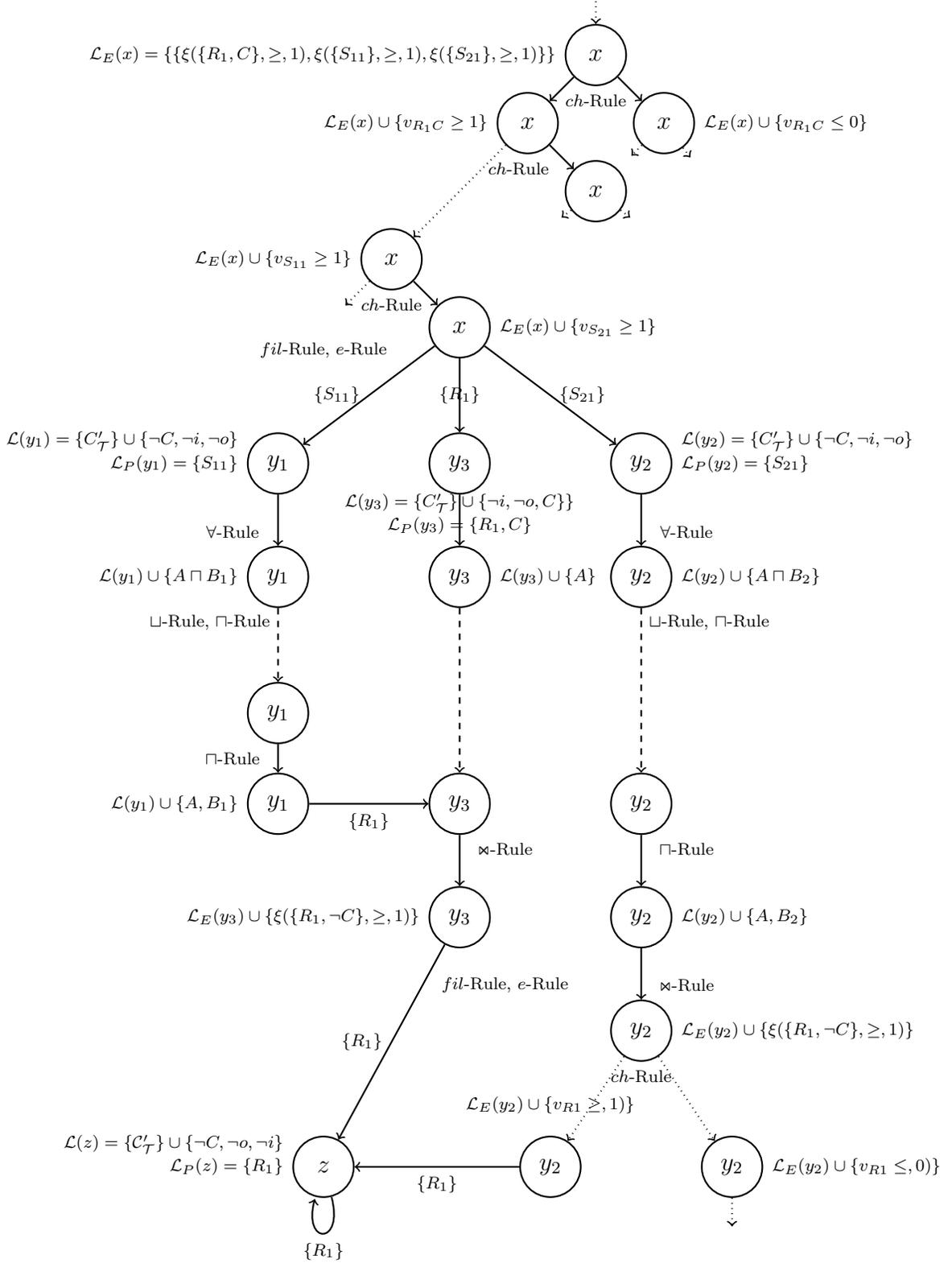
Figure 29: Expansion tree considering a distribution of role fillers such that $v_{R_1C} \geq 1$, $v_{S_{11}} \geq 1$, and $v_{S_{21}} \geq 1$.

Since $C'_{\mathcal{T}}$ is in the label of each of y_1, y_2, y_3, the $\sqcap$-Rule, and $\sqcup$-Rule become applicable to y_1, y_2, and y_3. Consider that after applying these rules, the labels of y_1, y_2, y_3 are as follows:

$$\mathcal{L}(y_1) = \{\neg o, \neg i, \neg C, A, B_1, \geq 1R_1, \forall R_1.A, \neg B_2, \forall R.C\}$$
$$\mathcal{L}(y_2) = \{\neg o, \neg i, \neg C, A, B_2, \geq 1R_1, \forall R_1.A, \neg B_1, \forall R.\neg C\}$$
$$\mathcal{L}(y_3) = \{\neg o, \neg i, C, A, \geq 1R_1, \forall R_1.A, \neg B_1, \forall R.\neg C\}$$

Since $\geq 1R_1 \in \mathcal{L}(y_1)$ and $R_1 \in \mathcal{L}_P(y_3)$, the e-Rule becomes applicable to y_1 and an edge is created between y_1 and y_3 with $\mathcal{L}(\langle y_1, y_3 \rangle) = \{R_1\}$. Notice how y_3 is re-used because it satisfies the conditions for the e-Rule and no other node does. The $\bowtie$-Rule is applicable to y_2 and y_3 such that $\xi(\{R_1, \neg C\}, \geq, 1)$ is added to $\mathcal{L}_E(y_2)$ and $\mathcal{L}_E(y_3)$. Consider the case when the ch-Rule assigns $v_{R_1} \geq 1$ and all other applicable variables (to the newly added in-equation) to ≤ 0. The in-equation solver collects all in-equations (and previous solutions) and assigns v_{R_1} the value 1. The fil-Rule becomes applicable to y_3 and y_2 and one new node z is created such that: $\mathcal{L}_P(z) = \alpha(v_{R_1}) = \{R_1\}, \mathcal{L}_E(z) = \mathcal{L}_E(y_2), \mathcal{L}(z) = \{\neg o, \neg i, \neg C\} \cup \{C'_{\mathcal{T}}\}$.

The e-Rule becomes applicable to y_2 and an edge is created between y_2 and z such that $\mathcal{L}(\langle y_2, z \rangle) = \{R_1\}$. The node z is re-used by the e-Rule on y_3 to create an edge between y_3 and z such that $\mathcal{L}(\langle y_3, z_1 \rangle) = \{R_1\}$. Due to $C'_{\mathcal{T}}$ in $\mathcal{L}(z)$, the $\sqcap$-Rule, $\sqcup$-Rule, and $\bowtie$-Rule apply to z such that $\mathcal{L}(z) = \{\neg o, \neg i, \neg C, A, \geq 1R_1, \forall R_1.A, \neg B_1, \forall R.\neg C\}$, and $\mathcal{L}_E(z) = \mathcal{L}_E(z) \cup \xi(\{R_1, \neg C\}, \geq, 1)$. However, $\xi(\{R_1, \neg C\}, \geq, 1)$ has already been satisfied by the in-equation solver which means that the e-Rule is now applicable to z re-using z to create an edge such that $\mathcal{L}(\langle z, z \rangle) = \{R_1\}$. No rules are applicable anymore and no clash has been detected: there is a complete and clash-free CCG as shown in Figure 30 consisting of the nodes x, y_1, y_2, y_3, z[8] and the TBox is consistent.

---

[8]The node r_0 is artificial and will be ignored since it is not part of the pre-model.
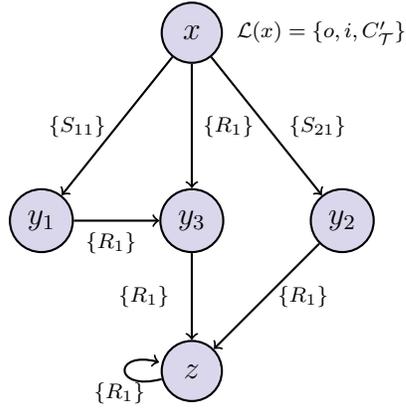
Figure 30: CCG representing a model in case when $\sigma(v_{oi}) = 1$, and $\mathcal{L}_P(x) = \{o, i\}$.
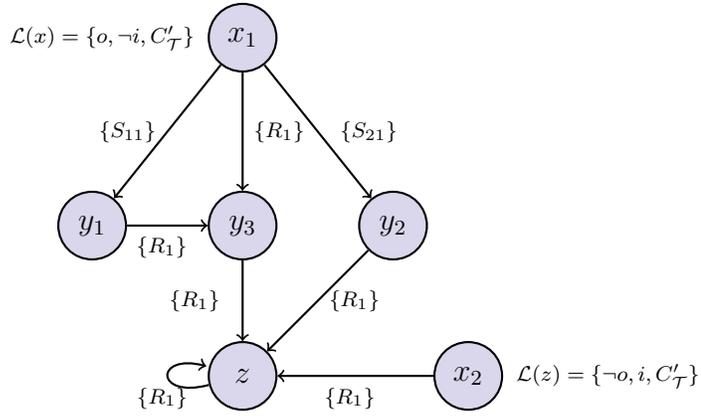
Figure 31: CCG representing a model in case when $\sigma(v_0) = 1$, $\sigma(v_i) = 1$, $\mathcal{L}_P(x) = \{o, C\}$, $\mathcal{L}_P(y) = \{i\}$.

4.6 Proofs

The soundness, completeness and termination of the algorithm presented in this chapter are consequences of Lemmas 4.3.2, Lemma 4.6.1, 4.6.2, and 4.6.3.

Lemma 4.6.1 (Termination)   When started with a $\mathcal{SHOQ}$ TBox $\mathcal{T}$, the proposed algorithm terminates and is worst-case double exponential.

Proof.   Let $l = \#clos(\mathcal{T})$, n_r denote the size of $N_{R'}$, n_o denote the size of N_o, and n_q denote the size of N_Q, termination of the algebraïc tableau algorithm is guaranteed due to the following.

- The rewriting in Algorithm 4.1.1 can be done in linear time and does not affect termination.

- Computing a partitioning $\mathcal{P}$ for $\mathcal{T}$: in the worst case $\#\mathcal{DS} = \#\{N_{R'} \cup N_Q \cup N_o\}$, and the size of $\mathcal{P}$ is bounded by $2^{r+o+q} - 1$ since we do not consider the empty partition. Although this computation is exponential, it is done at-most once.

- Getting a distribution of individuals (solution for the in-equations) will not affect termination of the completion rules. The Simplex method is considered to be one of the most significant algorithms for solving IP problems; its worst case complexity is exponential in the number of variables. However, it is very efficient in practice and converges in polynomial time for many input problems, in particular those with a fixed number of variables [HHLS86], as is the case with ξ in the label of a CCG nodes, where the number of variables is fixed to $(2^{r+o+q} - 1)$.

- The algorithm constructs a graph consisting of a set of arbitrarily interconnected nodes by applying completion rules which do not remove nodes from the graph,

nor remove concepts from node labels or edge labels. For each node x:

- the number of times the *fil*-Rule can be applied is bounded by the size of $\mathcal{P}$. In the worst case, one node is created for each partition. It is not possible to have more nodes than the size of $\mathcal{P}$, in the graph, since each node is either a nominal or a role filler and in both cases it must be in some partition in $\mathcal{P}$.

- the number of times the *e*-Rule is applied for each $\bowtie nR$ restriction is bounded by n (the largest number used in a QCR restriction). In the worst case individuals satisfying $\bowtie nR$ are distributed into n partitions. The total number that this rule can be applied is bounded by $l * n$.

- the *ch*-Rule non-deterministically assigns each variable to ≥ 1 or ≤ 0. Each variable is assigned once per completion graph which means that in the worst case when all possible completion graphs are explored, the *ch*-Rule is applied $2^{(2^{r+o+q}-1+1)} - 1$ times.

- all other rules are applied at most l times.

- Traditional termination problems due to cyclic TBoxes and the so-called "yo-yo" effect are not encountered:

  - cyclic definitions do not cause a termination problem since nodes having the same label (case when blocking is needed with other algorithms) will eventually be mapped to the same partition and only one proxy node is created. This justifies why we do not need any blocking strategies, the re-use of individuals acts like a natural block.

  - The "yo-yo" effect [Lut02] of infinitely creating and merging nodes cannot occur since in a given CCG, nodes are neither removed nor merged.

■

Lemma 4.6.2 (Soundness)   If the completion rules can be applied to $\mathcal{T}$ such that they yield a complete and clash-free CCG, then $\mathcal{T}$ has a tableau.

Proof.   A tableau T $= (\mathbf{S}, \mathcal{L}', \mathcal{E})$ can be obtained from a clash-free CCG G $= (V, E, \mathcal{L}, \mathcal{L}_{\mathrm{E}}, \mathcal{L}_P)$ by mapping nodes in G to individuals in T which can be defined from G as T such that: $\mathbf{S} = V \setminus \{r_0\}$, $\mathcal{L}'(x) = \mathcal{L}(x)$, and $\mathcal{E}(R) = \{\langle x, y \rangle \in E \mid (H(R) \cup \{R\}) \cap \mathcal{L}(\langle x, y \rangle) \neq \emptyset\}$. We show that T is either a tableau or can be easily extended to a tableau for $\mathcal{T}$ since properties (1) - (11) of a tableau (see Def. 4.3.1) are either satisfied or can be easily satisfied.

- Property (1): Assume there exists an individual x in $\mathbf{S}$ such that $C_{\mathcal{T}} \notin \mathcal{L}'(x)$, this means that the corresponding node x in G also satisfies $C_{\mathcal{T}} \notin \mathcal{L}(x)$. This case is not possible first because x cannot be r_0 and second because $C_{\mathcal{T}}$ is added to $\mathcal{L}(x)$ for every node x created in G by the *fil*-Rule. Hence $C_{\mathcal{T}} \in \mathcal{L}'(x)$ for every $x \in \mathbf{S}$ and Property (1) is satisfied.

- Property (2): Assume there exists an individual x in $\mathbf{S}$ such that $A \in \mathcal{L}'(x)$ and $\neg A \in \mathcal{L}'(x)$ this means that there exists a corresponding node x in G such that $A \in \mathcal{L}(x)$ and $\neg A \in \mathcal{L}(x)$. This case is not possible since G is clash-free. Hence, Property (2) is satisfied.

- Property (3): Assume there exists an individual x in $\mathbf{S}$ such that $C \sqcap D \in \mathcal{L}'(x)$, $C \in \mathcal{L}'(x)$, and $D \notin \mathcal{L}'(x)$ this means that there exists a corresponding node x in G such that $C \sqcap D \in \mathcal{L}(x)$, $C \in \mathcal{L}(x)$, and $D \notin \mathcal{L}(x)$. Having $C \sqcap D \in \mathcal{L}(x)$, $C \in \mathcal{L}(x)$, and $D \notin \mathcal{L}(x)$ makes the $\sqcap$-Rule applicable to x in G however this

115

case is not possible since G is complete. Hence Property (3) is satisfied and we can similarly prove that Property (4) is also satisfied.

- Property (5): Assume $\forall S.C \in \mathcal{L}'(x)$ and $\langle x, y \rangle \in \mathcal{E}(S)$ then we must have $C \in \mathcal{L}'(y)$. Having $\langle x, y \rangle \in \mathcal{E}(S)$ means that $\mathcal{L}(\langle x, y \rangle) \cap (H(S) \cup \{S\}) \neq \emptyset$. Since G is complete and clash free then C must be in $\mathcal{L}(y)$ otherwise the $\forall$-Rule conditions are met and the rule is applicable to G. Since $C \in \mathcal{L}(y)$ this means that $C \in \mathcal{L}'(y)$ and Property (5) is satisfied.

- Property (6): Assume there exists an individual x in $\mathbf{S}$ such that $\forall S.C \in \mathcal{L}'(x)$ and there exists an individual $y \in \mathbf{S}$ such that $\langle x, y \rangle \in \mathcal{E}(R)$ and R is a transitive role such that $R \sqsubseteq_* S \in \mathcal{R}$ then we must have $\forall R.C \in \mathcal{L}'(y)$. Since we have $\langle x, y \rangle \in \mathcal{E}(R)$ this means that $\mathcal{L}(\langle x, y \rangle) \cap (H(R) \cup \{R\}) \neq \emptyset$, and having $R \in N_{R+}$ with $R \sqsubseteq_* S \in \mathcal{R}$ then we have $\forall R.C \in \mathcal{L}(y)$ otherwise the $\forall_+$ would be applicable. Therefore, since $\forall R.C \in \mathcal{L}(y)$ then $\forall R.C \in \mathcal{L}'(y)$ and Property (6) is satisfied.

- Property (7): Assume $\forall R \backslash S.C \in \mathcal{L}'(x)$ and $\langle x, y \rangle \in \mathcal{E}(R)$ but not in $\mathcal{E}(S)$ then we must have $C \in \mathcal{L}'(y)$. Since we have $\langle x, y \rangle \in \mathcal{E}(R)$ and $\langle x, y \rangle \notin \mathcal{E}(S)$, this means that $\mathcal{L}(\langle x, y \rangle) \cap (H(R) \cup \{R\}) \neq \emptyset$ and $\mathcal{L}(\langle x, y \rangle) \cap (H(S) \cup \{S\}) = \emptyset$ respectively. C must be in $\mathcal{L}(y)$ otherwise the $\forall_\backslash$-Rule would be applicable to G. Since $C \in \mathcal{L}(y)$ this means that $C \in \mathcal{L}'(y)$ and Property (7) is satisfied

- Property (8): Assume $(\geq nS) \in \mathcal{L}'(x)$ then completeness of G implies that there exist j proxy individuals $y_1 \ldots y_j$ each representing a partition of m_i individuals such that $\sum_{i=1}^{j} m_i = n$ and $S \in \mathcal{L}(\langle x, y_i \rangle)$ $(1 \leq i \leq j)$. Due to Lemma 4.5.4, we can replicate each y_i, $m_i - 1$ times and set $\mathbf{S} = \mathbf{S} \cup \{y_{i_k}\}$ and $\mathcal{L}(\langle x, y_{i_k} \rangle) = S$ with $1 \leq k \leq m_i - 1$, then we have $\#S^{\mathcal{T}}(x) \geq n$ and property (8) is satisfied.

One might think that replicating individuals could result in violating the *nom-inals* semantics (Property 11) for example by replicating a nominal individual. However, this case can never happen since *nominals* are represented by proxy individuals y_i belonging to a partition with only one individual, $m_i = 1$ always holds for *nominals* partitions and is encoded by the in-equations (see Property (11) below). Similarly, Property (9) cannot be violated due to replication of individuals; partition sizes (m_i) are assigned such that all at-least and at-most restrictions are satisfied (see Property (9) below).

- Property (9): Assume $(\leq mS) \in \mathcal{L}'(x)$ and $\#S^{\mathcal{T}}(x) \leq m$ is violated. This means that we have j proxy individuals $y_1 \ldots y_j$ each representing a partition of m_i individuals such that $\sum_{i=1}^{j} m_i > m$. This case cannot happen for two reasons: (1) Having the lowest priority for the *fil-Rule*, nodes are created only after making sure that all at-least and at-most restrictions for a node x are satisfied by a distribution of role fillers (a non-negative integer solution for the in-equations in $\mathcal{L}_E(x)$). This means that no nodes will be created that violate an at-most restriction. (2) G is clash free which means that for each $(\leq mS) \in \mathcal{L}(x)$ we have $\xi(\{S\}, \leq, m)$ in $\mathcal{L}_E(x)$ and there is no $\xi(\{S\}, \geq, n)$ in $\mathcal{L}_E(x)$ and $n > m$.

- Property (10): If the distribution is not consistent with $\mathcal{R}$, then for some $(R' \sqsubseteq_* R)$, there exists an R'-*filler* y assigned to a partition P with $R' \in P$ and $P^{\mathcal{I}} \subseteq (FIL(R') \setminus FIL(R))$. This case is not possible due to the definition of H(R) which assumes that R is implied in P whenever $R' \in P$ and $R' \in H(R)$. Hence, this property is always satisfied.

- Property (11): G cannot contain two nodes x and y such that for some nominal $o \in N_o$ we have $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$. Since each node in G is a representative for a

117

partition P then having two nodes x and y with $o \in \mathcal{L}(x) \cap \mathcal{L}(y)$ means that there are two partitions P_1 and P_2 such that $o \in P_1 \cap P_2$. However since partitions are disjoint (Lemma 4.4.5) and due to the *nominals* semantics encoded into $\xi(\{o\}, \leq, 1)$ and $\xi(\{o\}, \geq, 1)$ in $\mathcal{L}_E(r_0)$ the in-equation solver will assign the nominal o to only one partition P_1 or P_2 and all other partitions will have $\neg o$ in the label of their proxies. In addition, no nodes that are created can be removed or merged, and no *nominals* individual can be replicated to satisfy Property 8. Therefore, the set of nodes with a nominal o in their label always satisfies property 11.

∎

Lemma 4.6.3 (Completeness) If $\mathcal{T}$ has a tableau, then the completion rules can be applied to $\mathcal{T}$ such that they yield a complete and clash-free CCG.

Proof. Let $\mathrm{T} = (\mathbf{S}, \mathcal{L}', \mathcal{E})$ be a tableau for $\mathcal{T}$, T can be used to guide the application of the completion rules. We define the mapping function π from nodes in the graph $G = (V, E, \mathcal{L}, \mathcal{L}_{\mathrm{E}}, \mathcal{L}_{\mathrm{P}})$ to individuals in $\mathbf{S}$, inductively with the creation of new nodes, such that for each $x, y \in V$, roles $R, S \in N_{\mathrm{R}}$ and a partition name $p \in \mathcal{P}$ we have:

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

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

3. $\xi(\{R\}, \geq, n) \in \mathcal{L}_{\mathrm{E}}(x)$ implies $\#R^T(\pi(x)) \geq n$

4. $\xi(\{R\}, \leq, n) \in \mathcal{L}_{\mathrm{E}}(x)$ implies $\#R^T(\pi(x)) \leq n$

5. $\xi(\{R, q\}, \geq, n) \in \mathcal{L}_{\mathrm{E}}(x)$ implies $\#(R^T(\pi(x)) \cap q^{\mathcal{I}}) \geq n$

6. $\xi(\{R, q\}, \leq, n) \in \mathcal{L}_E(x)$ implies $\#(R^T(\pi(x)) \cap q^{\mathcal{I}}) \leq n$

The claim is that having a CCG G that satisfies the properties of π we can apply the completion rules defined in Figure 23 and 24, when applicable, to G without violating the properties of π. Initially G consists of the artificial node r_0 such that $\bigcup_{o \in N_o}\{\xi(\{o\}, \geq, 1), \xi(\{o\}, \leq, 1)\} \subseteq \mathcal{L}_E(r_0)$ and at least one node x_0 with some $o \in \mathcal{L}(x_0)$ is created. Given a tableau T for G, we can set $s_0 = \pi(x_0)$ for some $s_0 \in \mathbf{S}$.

We show that whenever we can apply a completion rule to G, the properties of π are not violated: applying the $\sqcap$-Rule, $\sqcup$-Rule, or the $\forall$-Rule strictly extends the label of a node x and this does not violate properties of π due to properties (1)-(5) of a tableau. Let us consider applying the other rules to a given node x:

- The $\forall_+$-**Rule**: Having a node x in G such that $\forall R.C \in \mathcal{L}(x)$ and there exists a node y with $\mathcal{L}(\langle x, y \rangle) \cap (H(S) \cup \{S\}) \neq \emptyset$ and S is a transitive role such that $S \sqsubseteq_* R$, this means that there exists $\pi(x) \in \mathbf{S}$ such that $\forall R.C \in \mathcal{L}'(\pi(x))$ and there exists $\pi(y) \in \mathbf{S}$ such that $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}(S)$. Applying the $\forall_+$-Rule adds $\forall S.C$ to $\mathcal{L}(y)$ thus preserving Property (6) of a tableau ($\forall S.C \in \mathcal{L}'(\pi(y))$) without violating π.

- The $\forall_\backslash$-**Rule**: Having $\forall R \backslash S.C \in \mathcal{L}(x)$ with $\mathcal{L}(\langle x, y \rangle) \cap (H(R) \cup \{R\}) \neq \emptyset$ and $\mathcal{L}(\langle x, y \rangle) \cap (H(S) \cup \{S\}) = \emptyset$ this means that $\forall R \backslash S.C \in \mathcal{L}'(\pi(x))$ with $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}(R)$ and $\langle \pi(x), \pi(y) \rangle \notin \mathcal{E}(S)$. Applying the $\forall_\backslash$-Rule adds C to $\mathcal{L}(y)$ which means that C is now in $\mathcal{L}'(\pi(y))$ and Property 7 of a tableau is satisfied. This property along with properties of π cannot be violated later for example by having $\langle \pi(x), \pi(y) \rangle \in \mathcal{E}(S)$ due to the strategy of rule application which forces the $\forall_\backslash$-Rule to be applicable to a node only when no other rules are applicable. In particular, the e-Rule cannot be applied to x such that $\mathcal{L}(\langle x, y \rangle) \cap (H(S) \cup \{S\}) \neq \emptyset$, which would add $\langle \pi(x), \pi(y) \rangle$ to $\mathcal{E}(S)$, after the $\forall_\backslash$-Rule had been applied. For example, consider the following scenario:

119

- Initially let $\{\geq nR, \geq mS, \forall R.A, \forall R \backslash S.\neg A\} \subseteq \mathcal{L}(x)$ and y be a proxy node with $\mathcal{L}_P(y) = \{R, S\}$

- after applying the *e-Rule* for some $\geq nR \in \mathcal{L}(x)$ and the $\forall$-*Rule* for $(\forall R.A) \in \mathcal{L}(x)$, y is an *R-filler* of x with $\{A\} \subseteq \mathcal{L}(y)$

- after applying the $\forall_{\backslash}$-*Rule* for $(\forall R \backslash S.A) \in \mathcal{L}(x)$ we have $\{A, \neg A\} \subseteq \mathcal{L}(y)$ with y an *R-filler* of x.

This case cannot happen. Due to the strategy of rule applications in Section 4.5.3, the $\forall_{\backslash}$-*Rule* cannot be applied if the *e*-Rule is also applicable. The rule priorities make sure that the $\forall(R \backslash S)$ semantics are enforced only when no more nodes can be *S-fillers* of x and Properties (5) and (7) of the definition of a tableau are preserved.

- The $\bowtie$-**Rule**: If $(\geq nR)$ or $(\leq mR) \in \mathcal{L}(x)$, then $(\geq nR), (\leq mR) \in \mathcal{L}'(\pi(x))$, this implies that $\#R^T(\pi(x)) \geq n$, $\#R^T(\pi(x)) \leq m$, (properties 8 and 9 of a tableau). Applying the $\bowtie$-Rule, extends $\mathcal{L}_E(x)$ wither with $\xi(\{R\}, \geq, n)$ or $\xi(\{R\}, \leq, m)$ if no qualifications on a super-role of R apply or with $\xi(\{R, C\}, \geq, n)$ or $\xi(\{R, C\}, \leq, m)$ if a *qualifying concept* C also applies on R-fillers of x. In both cases the properties of π and those of a tableau are not violated.

- The *fil*-**Rule**: Since the *fil*-Rule has priority 2 then every $(\geq nR), (\leq mR) \in \mathcal{L}(x)$ has already been encoded into in-equation in $L_E(x)$ and due to the clash-freeness of T this means that there exists a distribution of role fillers satisfying every $(\geq nR), (\leq mR) \in \mathcal{L}(x)$. The distribution of fillers is encoded in the solution σ for $\mathcal{L}_E(x)$ and applying the *fil*-Rule creates a proxy individual y as a representative for each corresponding partition based on σ returned by the in-equation solver. Every node created is tagged with the proper partition name using $\mathcal{L}_P$ and the set of in-equations is propagated using $\mathcal{L}_E(x)$ to y. $\mathcal{L}_P$ is

later used by the *e*-Rule to create the proper edges between the nodes. Since the creating of nodes is guided by the solution, σ, returned by the in-equation solver and due to the rule priority, the number of nodes created the *fil*-Rule cannot violate properties of a tableau nor π.

- The ***e-Rule***: For each $(\geq nR) \in \mathcal{L}(x)$ we have $(\geq nR) \in \mathcal{L}'(\pi(x))$ which means that $\#R^T(\pi(x)) \geq n$ must be satisfied. The *e*-Rule is applied to connect x to its *R-fillers* such that with each ith $(1 \leq i \leq n)$ application of this rule an edge is created between x and some proxy individual y_i such that $R \in \mathcal{L}_P(y_i)$ and y_i represents m_i (the number of elements assigned to a partition by the in-equation solver) individuals of a partition p.

  After all edges have been created we have j proxy *R-fillers* each representing m_i individuals such as $\sum_{i=1}^{j} m_i \geq n$. Due to Lemma 4.5.4 we can replicate each y_i, $m_i - 1$ times and by setting $\mathcal{L}(\langle x, y_{i_k} \rangle) = \{R\}$ with $1 \leq i \leq j$ and $1 \leq k \leq m_i - 1$ and by setting $\pi = \pi[y_{1_1} \to t_{1_1}, \ldots, y_{i_k} \to t_{i_k}]$ with $t_{1_1} \ldots t_{i_k}$ tableau elements in T satisfying $\#R^T(\pi(x)) \geq n$. We can see that $\#R^T(\pi(x)) \geq n$ is satisfied without violating π. By analogy, we can prove that applying the *e*-Rule for each $(\leq nR) \in \mathcal{L}(x)$ does not violate π.

The resulting graph G is clash free due to the following:

1. G cannot contain a node x such that $\{A, \neg A\} \subseteq \mathcal{L}(x)$ since $\mathcal{L}(x) \subseteq \mathcal{L}'(\pi(x))$ and Property 2 of the definition of a tableau would be violated.

2. G cannot contain a node x such that $\mathcal{L}_E(x)$ is unsolvable. If $\mathcal{L}_E(x)$ is unsolvable, this means that for some role $R \in N_R$ we have:

   - $\{\xi(\{R\}, \geq, n)\} \subseteq \mathcal{L}_E(x)$, and there is no possible distribution of *R-fillers* satisfying $\geq nR \subseteq \mathcal{L}(x)$, hence, property 8 of a tableau would be violated due to the equivalence properties between $\xi(\{R\}, \geq, n) \in \mathcal{L}_E(x)$ and

121

$\#R^T(\pi(x)) \geq n$ respectively, or

- $\{\xi(\{R\}, \leq, n)\} \subseteq \mathcal{L}_{\mathrm{E}}(x)$, and there is no possible distribution of *R-fillers* satisfying $\leq mR$, hence, property 9 of a tableau would be violated due to the equivalence properties between $\xi(\{R\}, \leq, m) \in \mathcal{L}_{\mathrm{E}}(x)$ and $\#R^T(\pi(x)) \leq m$.

4.7 Discussion

This section highlights some of the novel characteristics of the hybrid algebraïc reasoning algorithm presented in this chapter.

4.7.1 Completion Graph Characteristics

A compressed completion graph G for a KB $(\mathcal{T}, \mathcal{R})$ consists of the artificial root node r_0, which is not part of the model for KB, and arbitrarily interconnected proxy nodes. Unlike standard tableau algorithms for $\mathcal{SHOQ}$ [HS01], a tree-like or forest-like shape restriction is not enforced on the shape of the completion graph. This feature is desirable since not all models are necessarily tree-shaped [MH08]. Such freedom in the completion graph shape is considered novel because a tree-shaped/forest-shaped feature is usually crucial for termination of standard tableau algorithm. It also allows a better handling of KBs with complex structures for example, a KB for the human anatomy does not necessarily have a tree-shaped model (or a tree-shaped completion graph). Restricting a model to a tree-like one would unnecessarily complicate constructing G.

4.7.2 Using an In-equation Solver

Using an in-equation solver to find a solution for the in-equations encoding *nominals* and QCRs, allows the algorithm to scale better when the size of the numbers used with QCRs increases.

$$
\begin{array}{rcl}
A & \sqsubseteq & \geq nR.A \\
B_1 & \sqsubseteq & \forall R.C \\
B_2 & \sqsubseteq & \forall R.\neg C \\
\{o\} & \sqsubseteq & \geq nS_1.(A \sqcap B_1) \sqcap \geq nS_2.(A \sqcap B_2))
\end{array}
$$

Figure 32: Example TBox $\mathcal{T}$.

For example, applying the algebraïc algorithm to test the consistency of the TBox $\mathcal{T}$ shown in Figure 32 for large values of n ($n = 100$) will not affect the behaviour of the algorithm as was reported in [FH10c] for the DL $\mathcal{SHQ}$. This makes its extension to more expressive logics more promising.[9] Additionally, the in-equation solver facilitates early clash (Definition 4.5.5 (ii)) detection, and ensures that a minimum number of role fillers is considered by setting the objective function to minimize the sum of variables considered. See Sections 7.2.1, and 7.2.2.6 for an empirical evaluation of this feature.

4.7.3 Termination

As illustrated with the example in Section 4.5.5, termination of the completion graph expansion is naturally inherent. Unlike traditional DL reasoning algorithms for $\mathcal{SHOQ}$, a tree-like model property with cycle detection techniques/blocking strategy is not crucial for termination. Nodes created are neither merged nor pruned

---

[9]Large values of n are known to be problematic for most DL reasoners supporting $\mathcal{SHQ}$.

which means that there is no need to handle the so-called "yoyo" effect, of infinitely creating and merging nodes, or to manage incoming and outgoing edges between nodes.

4.7.4   Proxy Nodes and Their Re-use

The completion graph used in this calculus is called "compressed completion graph" and this is due to the use and re-use of proxy nodes as representatives for nodes having common restrictions. Using proxy nodes helps minimize the number of nodes to be created and the number of completion rules to be triggered, which means that in some cases non-determinism can be minimized because some of the completion rules are non-deterministic. When creating a representation for a distribution of domain elements, let p_a denote the number of partitions used, $p_a = \#\mathcal{P}$, p_o denote the number of *nominals*, and $p_\bowtie$ denote the number of at-least and at-most restrictions, we consider the following cases:

- Case 1: All domain elements fall in the same partition and only one proxy is created. The KB is under-constrained and an over-constrained representation of it is created. In this case only one node is created other than r_0.

- Case 2: All elements satisfying an at-least or an at-most restriction are in the same partition and only one proxy is created for each at-least or at-most restriction. In this case $p_a = \max\{p_\bowtie, p_o\}$ if *nominals* interact with role fillers, or $p_a = (p_\bowtie + p_o)$ if *nominals* do not interact with role fillers. The total number of nodes created equals p_a.

- Case 3: Elements satisfying each at-least and at-most restriction of the form $\bowtie nR$ are in n different partitions and n proxy nodes are created for each $\bowtie nR$ restriction. In this case $p_a = \max\{(n \times p_\bowtie), p_o\}$ if *nominals* interact with role

fillers, or $p_\mathrm{a} = (n \times p_\bowtie + p_o)$ if *nominals* do not interact with role fillers. The total number of nodes created equals p_a.

On the other hand, nodes that are created can later be re-used. The re-use of individuals has also been proposed in [MH08] recently. However, the re-use implemented by the algorithm presented in this chapter is more informed. Once a node is created, it is tagged, using $\mathcal{L}_P$, based on the partition it belongs to. Which means that each node is tagged by the signature its representative elements can satisfy without violating a number restriction. For example when an element is assigned to a partition labeled $\{R_1, R_2\}$ this means that this element is a potential R_1-filler and a potential R_2-filler. The *e*-Rule uses and re-uses this individual whenever an R_1-filler or an R_2-filler is needed. In a sense, once a distribution of domain elements is assigned by the in-equation solver, the proxy node re-use is deterministic, there is no guessing of which individuals can be re-used. This form of re-use still ensures termination while preserving soundness and completeness. One could say that the re-use acts like blocking in the case of cyclic descriptions. However, there is no use of any cycle detection mechanism, and the re-use is not intended for termination because it is not only used in the case of cycles. The use of a proxy nodes together with the re-use of nodes could work as a double optimization to reduce non-determinism and model sizes especially since KBs are often naturally under-constrained, which facilitates individual re-use.

4.7.5   Caching

The *ch*-Rule described in Figure 24 performs a semantic split for groups of elements (a single partition) and not necessarily for each element as is the case with tableau algorithms using the *choose*-rule (described in Section 6), which non-deterministically chooses a distribution for each role filler. It is interesting to note that the splitting of the *ch*-Rule allows some form of global caching. Partitions represent signatures

(Lemma 4.4.5) and variables are used to represent the cardinalities of these partitions. Then, if a variable must be zero, this means that the signature for the corresponding partitions is unsatisfiable. This result is carried throughout the search by setting the corresponding variable to zero and no individuals are assigned to that partition. However, if a variable v_P is ≥ 1 this means that the signature of $\alpha(v) = P$ is satisfiable and at least one individual x is a member of this signature. Whenever a new individual is needed satisfying the signature of P, x is re-used.

4.7.6   The EU Example

Consider testing the consistency of the the TBox $\mathcal{T}$ representing the EU member states as defined in Example 4.1.5. For ease of presentation, the concept names, role names, and *nominals* are replaced with a one letter symbol such that $N_{\mathrm{R}} = \{\mathsf{M}', \mathsf{M}\}$, $\mathcal{R} = \{\mathsf{M}' \sqsubseteq \mathsf{M}\}$, $N_{\mathrm{o}} = \{\mathsf{i}, \mathsf{o}_1, \ldots, \mathsf{o}_{27}\}$. The global decomposition set consists of $\mathcal{DS} = \{\mathsf{M}', \mathsf{i}, \mathsf{o}_1, \ldots, \mathsf{o}_{27}\}$ and the global partitioning of $\mathcal{DS}$ results in $(2^{29} - 1)$ partitions. However, all instances of $\mathsf{EU\_MemberStates}$ are disjoint and one can safely ignore partitions having more then one nominal. Figure 33 shows the corresponding partitioning; in total one only need to consider $(2 \times 28 + 1)$ partitions.

Considering an initial distribution of *nominals* and after applying the completion rules in Figure 23 and Figure 24 then the CCG contains a node (x) such that $\mathcal{L}_P(x) = \{\mathsf{i}, \neg\mathsf{o}_1, \ldots, \neg\mathsf{o}_{27}\}$ and $\mathcal{L}(x) = \{\mathsf{i}, \neg\mathsf{E}, \neg\mathsf{o}_1, \ldots, \neg\mathsf{o}_{27}, \geq 30\mathsf{M}', \forall\mathsf{M}'.\mathsf{E}\}$. The $\bowtie$-Rule would add $\xi(\{\mathsf{M}'\}, \geq, 30)$ to the set, $\mathcal{L}_E(x)$, of in-equations encoding the *nominals* semantics.

$$\mathcal{L}_E(x) = \left\{ \begin{array}{l} \xi(\{\mathsf{i}\}, \geq, 1), \xi(\{\mathsf{i}\}, \leq, 1), \xi(\{\mathsf{o}_1\}, \geq, 1), \xi(\{\mathsf{o}_1\}, \leq, 1), \ldots, \\ \xi(\{\mathsf{o}_{27}\}, \geq, 1), \xi(\{\mathsf{o}_{27}\}, \leq, 1), \xi(\{\mathsf{M}'\}, \geq, 30) \end{array} \right\}$$

The unsatisfiability of $\mathcal{L}_E(x)$ can be immediately detected by the in-equation solver considering that $v_{\mathsf{M}'} \leq 0$ and $v_{\mathsf{M}'\mathsf{o}_1}, \ldots, v_{\mathsf{M}'\mathsf{o}_{27}}$ are all ≥ 1 as the initial distribution of *nominals*.

126
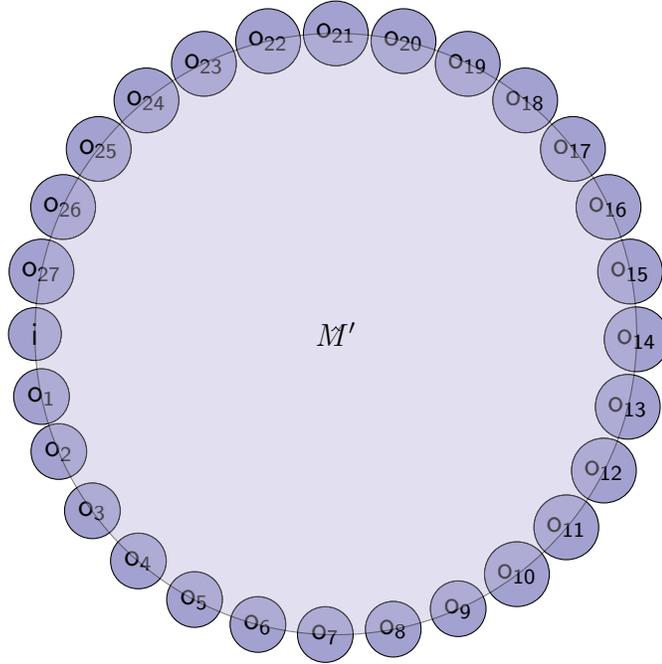
Figure 33: Partitioning of $\mathcal{DS} = \{\mathsf{M}', \mathsf{i}, \mathsf{o}_1, \ldots, \mathsf{o}_{27}\}$

In comparison with standard tableau algorithms for $\mathcal{SHOQ}$ when checking the satisfiability of $\geq 30\mathsf{M}'.\mathsf{EU}$, 30 anonymous individuals are created and then non-deterministically identified with the 27 *nominals*. Considering that we have 30 individuals that need to be distributed over 27 there are $\frac{30!}{3!} = 4.420 \times 10^{31}$ cases to be considered. In the case of the algebraïc method, one would have to consider, in the worst case, $2^{28 \times 2 + 1} - 1 = 1.441 \times 10^{17} - 1$ cases for the *ch*-Rule until $v_{\mathsf{M}'} \leq 0$ and $v_{\mathsf{M}'\mathsf{o}_1}$, ..., $v_{\mathsf{M}'\mathsf{o}_{27}}$ are all ≥ 1.

4.8 Conclusion

This chapter presents an algebraïc tableau reasoning algorithm for $\mathcal{SHOQ}$. Unlike available reasoning algorithms for $\mathcal{SHOQ}$, the algebraïc tableau method allows a calculus that is explicitely informed about the numerical restrictions on domain elements. The algebraïc reasoning is based on the *atomic decomposition technique* which

127

is applied on the proper global decomposition set allowing the calculus to handle the various interactions between *nominals*, role fillers and their qualifications.

When creating an abstraction of a model, only one representative element is created for each partition and tagged by the partition signature. Using a representative element not only helps in reducing the size of the pre-model generated but also allows for re-using elements. Due to the re-use, the calculus naturally handles cyclic descriptions without the need for any blocking strategies to ensure termination.

The upper bound on the size of the search space is double exponential to the size of the input problem, mainly due to non-determinic expansions. For soundness and completeness proofs, non-deterministic expansions are not given much consideration because one only needs to prove that the search will always find a solution if one exists and it will always terminate. From a theoretical point of view, it is enough to devise/analyze the upper bound on the worst case complexity. However, when considering a practical implementation, it is crucial to give careful consideration to non-deterministic expansions; in particular how to reduce the size of the search space and how to explore it in an efficient manner. This will be discussed in the following chapter.

Chapter 5

Towards Practical Algebraïc Reasoning With DL

5.1 Introduction

Reasoning Algebraïcally with Description Logics (ReAl DL) was presented in Chapter 4 using a sound and complete hybrid algebraïc tableau-based reasoning algorithm for the DL $\mathcal{SHOQ}$. If not amenable to optimizations, the practical usefulness of ReAl DL is questionable and the contribution of this thesis remains purely theoretical. Such usefulness debate has always accompanied the design of decision procedures for expressive DLs. This is because expressivity usually carries out an inevitable high worst case complexity. The satisfiability problem for $\mathcal{SHOQ}$ is ExpTime-complete; available DL systems handling $\mathcal{SHOQ}$ implement a wide range of optimization techniques without which, these systems fail to efficiently handle satisfiability problems of growing size. As discussed in the previous chapter, the hybrid algebraïc reasoning algorithm for ReAl DL which runs in double exponential time in the worst case will not have any practical merit if not amenable to optimizations. The theoretical efficiency (w.r.t. the worst case complexity) might be even questioned due to the gap between

the complexity of the satisfiability problem, which is ExpTime-complete, and that of the reasoning algorithm which is 2ExpTime.

Due to its hybrid nature, the algebraïc tableau algorithm can be seen as a mixed algorithm implementing both search and constraint satisfaction problems (CSP). A wide range of optimizations have been investigated to enhance the performance of search based satisfiability algorithms [FB07b, THPS07]. Also, a wide range of optimizations have been studied to enhance the performance of CSPs [DF02]. Some of the optimizations for DL search based satisfiability have already been adapted from earlier versions of optimizations for CSPs [Bak95], namely *dependency directed backtracking* (see Section 3.2.1.3 for a review). In this chapter, two major sources of inefficiency, that cannot be addressed through existing optimizations, are identified with *non-determinism* and *global partitioning*. Sections 5.3 through 5.6 discuss how non-determinism can be handled more efficiently at different phases using *preprocessing optimizations*, *look-back optimizations*, *look-ahead optimizations* and a combination of both. Section 5.7 discusses how partitioning can be optimized using lazy partitioning. A discussion of the different suggested techniques is presented in Section 5.9 before the chapter is concluded in Section 5.10.

5.2   From Theory to Practice

When moving from theory to practice, there has always been a significant gap between the design of a DL reasoning algorithm and its practical implementation. Despite the high worst case complexity of DL inference services, experiments with early DL systems such as KRIS, and lately with SOTA DL systems have shown that applying suited (even simple) optimization techniques could lead to a significant improvement in the empirical evaluation of a DL system. These optimizations render the adopted reasoning algorithm useful for realistic applications, even in cases

where non-optimized implementations of the algorithm are hopelessly intractable. The worst case complexity of the hybrid algebraïc tableau-based satisfiability algorithm is demonstrated in Section 4.6 as being double exponential. The theoretical complexity result is not surprising because the satisfiability problem of expressive DLs is usually inevitably at-least exponential. However, the algorithm might be considered theoretically inefficient because the complexity of the satisfiability algorithm (double exponential) came out greater than the complexity of the satisfiability problem itself (single exponential). Such a gap between the complexity of an algorithm and that of the problem might be due to the fact that the algorithm was designed in such a way to facilitate proofs of its soundness and completeness without much consideration to its worst case complexity or practical implementation.

It might seem a little bit discouraging to consider the practical implication of an algorithm with a questionable theoretical worst case complexity. However, a high worst-case complexity is not uncommon with practical DL systems. For example, the hyper-tableau satisfiability algorithms [MSH07, MSH09] designed to handle *general concept inclusion axioms* (GCIs) more efficiently with the DLs $\mathcal{SHIQ}$ and $\mathcal{SHOIQ}$ share a double exponential worst case complexity, whereas satisfiability with $\mathcal{SHIQ}$ is ExpTime-complete and that with $\mathcal{SHOIQ}$ is NExptime-complete. Also, the algebraïc tableau reasoning algorithm [FH10c] designed for the DL $\mathcal{SHQ}$ runs in double exponential time whereas satisfiability with $\mathcal{SHQ}$ is ExpTime-complete. On the other hand, systems based on optimized implementations of these algorithms demonstrate significant performance improvement showing their practical impact in solving specialized problems. So far, no better way has been reported in solving QCRs other than through algebraïc reasoning. Also, Hermit is the first reasoner able to classify ontologies which had previously been proven too complex for any available reasoner

131

to handle[1].

When considering a practical implementation for the hybrid algebraïc algorithm one can quickly identify the two major sources of inefficiency: *non-determinism* in expansion rules, and *global partitioning* which aggravates the non-deterministic *ch*-Rule. Specialized techniques to handle those inefficiency sources are discussed in the following sections.

5.2.1 Towards Practical Non-Determinism

Poor performance of tableau-based DL systems has been usually associated with non-determinism in tableau expansion rules. The algorithm for $\mathcal{SHON}_{\mathcal{R}\backslash}$, as described in Section 4.5, implements two non-deterministic rules: the *ch*-Rule and the $\sqcup$-Rule. Each time the $\sqcup$-Rule is applied to a node x in a *compressed completion graph G* with $(C_1 \sqcup C_2 \sqcup \cdots \sqcup C_n) \in \mathcal{L}(x)$, a list of child graphs is returned where the label $\mathcal{L}(x)$ is extended with C_i in each child graph. As shown in Figure 34, the size of the list of child graphs is bounded by n. Each time the *ch*-Rule is applied to a node x a list of child graphs is returned where the label $\mathcal{L}_E(x)$ is extended with one operand in each child graph. As shown in Figure 35, the size of the list of child graphs is bounded by 2. Each child graph represents a choice point in the search tree. The algorithm explores choice points until no more rules are applicable or a clash occurs.

The search space is explored in a depth-first manner such that once a clash is encountered either logically or arithmetically, the nearest alternative choice point is explored. This is called backtracking and it usually involves two phases:

- *Forward phase.* The forward phase usually begins when a non-deterministic rule is applicable. In case of the *ch*-Rule, this phase consists of selecting a range value for the variable v such that the label of the node $\mathcal{L}_E(x)$ is extended with

---

[1]Hermit is also the first reasoner based on hyper-tableaux reasoning [MSH09].

either $v \geq 1$ or $v \leq 0$.

$$\mathcal{L}(x) = \{(C_1 \sqcup C_2 \sqcup \cdots \sqcup C_n)\}$$

$$\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1\}$$
$$\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_2\}$$
$$\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_n\}$$
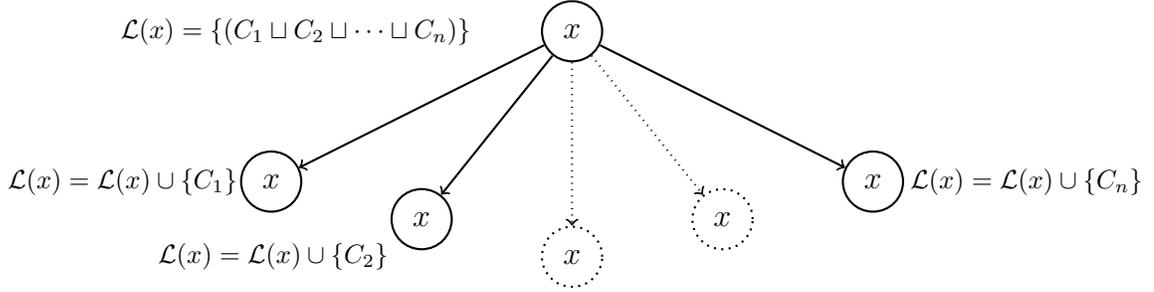
Figure 34: Expansion tree due to the $\sqcup$-Rule.

$$\mathcal{L}_E(x) = \{v_1 + v_2 + \cdots + v_n \bowtie m\}$$
$$\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v_1 \geq 1\}$$
$$\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v_1 \leq 0\}$$
$$\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v_2 \leq 0\}$$
$$\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v_{n-1} \geq 1\}$$
$$\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v_n \geq 1\}$$
$$\mathcal{L}_E(x) = \mathcal{L}_E(x) \cup \{v_n \leq 0\}$$
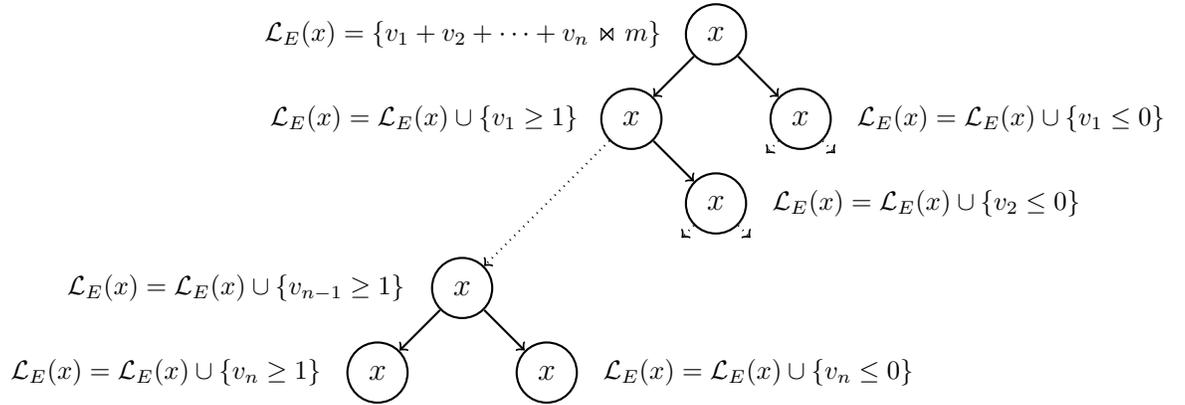
Figure 35: Expansion tree due to the *ch*-Rule.

In case of the $\sqcup$-rule, this phase consists of selecting a disjunct C_i with $1 \leq i \leq n$ from $(C_1 \sqcup C_2 \ldots \sqcup C_n)$ such that the label of the node is extended with C_i.

- *Backward phase* This phase usually begins when the current assigned variable range or the selected disjunct causes a clash. In this case backtracking returns to the previous choice point and looks for a new extension of the node's label.

Due to the high degree of non-determinism, the algorithm can easily fail to be scalable or even practical because the search space is double exponential to the size of the problem. Exploring the whole search space would inevitably lead to intractability for all but the smallest problems. Also, naïve backtracking suffers from thrashing

133

(an example of thrashing was given in Section 3.2.1.3 Figure 15) which refers to rediscovering the same inconsistencies and the same partial successes during search. Therefore, the practical usability of the algebraïc algorithm can be considered by designing optimization techniques aiming at reducing the size of the search space through improved backtracking and finding more efficient ways to explore it through heuristics.

There are typically two types of procedures that have been designed to improve backtracking with search based algorithm and CSPs. Those employed before the search algorithm is started to set a bound on the size of the search space, and those employed dynamically during the execution of the algorithm to decide which branching points can be safely pruned from the search space. In the context of $\mathcal{SHON}_{\mathcal{R}\backslash}$, preprocessing optimizations aiming at reducing the number of variables used can be considered, as well as dynamically improving the pruning power of backtracking through *look back* and *look ahead optimizations. Look back optimizations* aim at guiding backtracking as soon as the algorithm encounters a clash and is ready to backtrack during a *backward phase. Look ahead optimizations* aim at discarding choice points during a *forward phase* as soon as a non-deterministic rule is applied.

5.2.2 Towards Practical Partitioning

Another source of inefficiency with the algebraïc tableau algorithm is the global partitioning, which comes with an exponential blow up of variables, that if naïvely treated by the *ch*-Rule, gives a double exponential worst case algorithm. For example, given a global decomposition set $\mathcal{DS}$ of size d, the atomic decomposition considers 2^d partitions while computing a global partitioning on $\mathcal{DS}$. A naïve implementation of such an algorithm initializes 2^d variables, and the *ch*-Rule considers 2^{2^d} branching points

in the worst-case. A naïve implementation is doomed because it runs in double expo-
nential time and space. Even with optimized expansions of the *ch*-Rule, the algorithm
suffers from an exponential overhead; that of computing the global partitions. On the
other hand, not all partitions need to be considered and the semantic split (nature of
the algorithm) allows a great deal of optimizations. *Lazy partitioning* and *lazy nomi-
nal generation* can be used to minimize or delay the effect of the global partitioning,
these techniques are discussed in Section 5.7, and 5.8 respectively.

5.3 Preprocessing Optimizations

This section discusses optimizations that are used before applying tableau expansion
rules, aiming at bounding the size of the search space or determining a fixed order in
which a distribution of *nominals* is considered.

Given a $\mathcal{SHON}_{\mathcal{R}\backslash}$ TBox $\mathcal{T}$, let N_R denote the set of role names used in concept
descriptions occurring in $\mathcal{T}$, $\mathcal{P}$ denotes a partitioning on the global decomposition
set $\mathcal{DS} = N_\mathrm{o} \cup N_{\mathrm{R}'} \cup N_\mathrm{Q}$ (where N_o is the set of *nominals*, $N_{\mathrm{R}'}$ is the set of newly
introduced roles after applying Algorithm 4.1.1, and N_Q is the set of qualifying con-
cepts), and G denotes a *compressed completion graph* (CCG). The size of $\mathcal{P}$ greatly
affects the size of the search space needed to complete a KB consistency test. Due
to the applicability of the *ch*-Rule for variables mapped to each partition $p \in \mathcal{P}$, the
size of the search space grows exponentially to the size of $\mathcal{P}$. Recall from Section
4.6 that the size of $\mathcal{P}$ is exponential to the size of the global decomposition set $\mathcal{DS}$,
$\#\mathcal{P} = 2^{n_{r'}+n_o+n_q} - 1$. Therefore, a smaller decomposition set $\mathcal{DS}$ means a smaller
number of partitions hence variables. However the size of $\mathcal{DS}$ is fixed by the num-
ber of *nominals* (n_o), the number of newly introduced roles $(n_{r'})$, and the number of
qualifying concepts (n_q) occurring in $\mathcal{T}$. On the other hand one can exploit TBox and
RBox axioms to detect and discard *quasi-noGood* (see Definition 5.3.3) and *noGood*

(see Definition 5.3.1) partitions either initially when the global partitioning is being computed, or during tableau expansion due to an unavoidable clash. This section presents techniques that can be used to initially discard *quasi-noGood* and *noGood* partitions and we leave it to Section 5.5.2 to discuss techniques that can be used to dynamically discard such partitions. Recall from Section 4.4.2 that for each partition $p \in P$ corresponds a signature F such that a partition p is empty if $F^{\mathcal{I}}$ is empty.

Definition 5.3.1 (noGood partition)   A partition $p \in \mathcal{P}$ is said to be noGood w.r.t $\mathcal{T}'$ if the signature, F, of p cannot be satisfied ($F^{\mathcal{I}} = \emptyset$). In other words, no domain element can be assigned to this partition without causing a clash, and therefore p must be empty. This can happen in the following cases given C_1, C_2 disjoint *nominals* or disjoint qualifying concepts in $\mathcal{T}$ ($C_1, C_2 \in N_o \cup N_Q$) and R_1, R_2 newly introduced role names in $N_{R'}$ such that:

- $\{C_1, C_2\} \subseteq p$,

- $\{R_1, R_2\} \subseteq p$ with $\forall R_1.C_1, \forall R_2.C_2 \in \mathcal{T}'$,

- $\{C_1, R_2\} \subseteq p$ with $\forall R_2.C_2 \in \mathcal{T}'$.

In all cases, the signature F of p is such that $\{C_1, C_2\} \subseteq F$ and $F^{\mathcal{I}} = \emptyset$ w.r.t $\mathcal{T}'$ because $C_1 \sqsubseteq \neg C_2 \in \mathcal{T}'$. For example, the partition $\mathsf{p_c} = \{\mathsf{hC_2}, \mathsf{hC_3}\}$ is a *noGood* partition w.r.t $\mathcal{T}'$ in Figure 36 because an element j assigned to $\mathsf{p_c}$ is an $\mathsf{hC_2}$-filler and an $\mathsf{hC_3}$-filler for some element i ($j \in FIL(\mathsf{hC_2}, i) \cap FIL(\mathsf{hC_3}, i)$) such that ($i$: BusyParent) and $\langle i, j \rangle : \mathsf{hC_1}$, $\langle i, j \rangle : \mathsf{hC_2}$. The element j must satisfy $F^{\mathcal{I}}$ such that (j : Female) because of $\forall \mathsf{hC_2}.\mathsf{Female}$ and (j : Male) because of $\forall \mathsf{hC_3}.\mathsf{Male}$. However, this is not possible because Male and Female are disjoint in $\mathcal{T}$ which makes $F^{\mathcal{I}} = \emptyset$.

A *noGood* partition p cannot be extended with elements from $\mathcal{DS}$ to form a new non-empty partition p'. This is because the unsatisfiability of F is inherent in the

signature, F', of p'. For example, the partitions $\mathsf{p_d} = \{\mathsf{hC_2}, \mathsf{hC_3}, \mathsf{Canada}\}$ is also a *noGood* partition.

Parent	$\sqsubseteq$ Person $\sqcap \geq 1$hasChild.Child
BusyParent	$\sqsubseteq$ Parent ≥ 3hasChild.Female $\sqcap \geq 2$hasChild.Male
Male	$\sqsubseteq \neg$Female
ParentOfACanadian	$\sqsubseteq$ Person $\sqcap \geq 1$hasChild.Canadian
Country	$\equiv \{$Afghanistan, ..., Canada, ..., Zimbabwe$\}$
Canadian	$\sqsubseteq$ Person $\sqcap \geq 1$bornIn.$\{$Canada$\}$
CanadianResident	$\sqsubseteq$ Person $\sqcap \geq 1$livesIn.$\{$Canada$\}$
Person	$\sqsubseteq$ (Male $\sqcup$ Female) $\sqcap \leq 1$bornIn.Country

(a) TBox axioms occurring in the $\mathcal{SHOQ}$ TBox $\mathcal{T}$.

Parent	$\sqsubseteq$ Person $\sqcap \geq 1$hC$_1 \sqcap \forall$hC$_1$.Child
BusyParent	$\sqsubseteq$ Parent ≥ 3hC$_2 \sqcap \forall$hC$_2$.Female $\sqcap \geq 2$hC$_3 \sqcap \forall$hC$_3$.Male
Male	$\sqsubseteq \neg$Female
ParentOfACanadian	$\sqsubseteq$ Person $\sqcap \geq 1$hC$_4 \sqcap \forall$hC$_4$.Canadian
Country	$\equiv \{$Afghanistan, ..., Canada, ..., Zimbabwe$\}$
Canadian	$\sqsubseteq$ Person $\sqcap \geq 1$bI$_2 \sqcap \forall$bI$_2$.$\{$Canada$\}$
CanadianResident	$\sqsubseteq$ Person $\sqcap \geq 1$lI$_1 \sqcap \forall$lI$_1$.$\{$Canada$\}$
Person	$\sqsubseteq$ (Male $\sqcup$ Female) $\sqcap \leq 1$bI$_1 \sqcap \forall$bI$_1$.Country $\sqcap \forall$(bornIn\bI$_1$).$\neg$Country

(b) TBox axioms occurring in the $\mathcal{SHON_{R\backslash}}$ TBox $\mathcal{T}'$ after preprocessing the TBox $\mathcal{T}$.

(c) Extended role hierarchy in $\mathcal{R}$ after preprocessing.

Figure 36: TBox axioms in the $\mathcal{SHON_{R\backslash}}$ TBox $\mathcal{T}'$ after preprocessing the TBox $\mathcal{T}$.

Definition 5.3.2 (noGood variable) A variable $v \in \mathcal{V}$ is said to be a noGood variable if v is mapped to a noGood partition.

Definition 5.3.3 (quasi-noGood partition) A partition $p \in \mathcal{P}$ is said to be quasi-noGood w.r.t $\mathcal{T}$ if p is an empty partition and the unsatisfiability of its signature, F, is caused by a $\forall R \setminus R'.C$ qualification:

- $R_1 \in p$ and $R_2 \notin p$ with $\forall R_1.C_1$ and $\forall R \setminus R_2.C_2$ such that $R_1 \sqsubseteq R, R_2 \subseteq R \in \mathcal{R}$ and $C_1 \sqsubseteq \neg C_2 \in \mathcal{T}$.

For example, The partition $\mathsf{p_d} = \{\mathsf{bl_2}\}$ is a quasi-noGood partition w.r.t $\mathcal{T}'$ in Figure 36 with $\{\forall \mathsf{bornIn} \setminus \mathsf{bl_1}.\neg\mathsf{Country}\}$. An element j assigned to $\mathsf{p_d}$ as a bornIn-filler and $\mathsf{bl_2}$-filler for $i : \mathsf{Canadian}$ $(j \in FIL(\mathsf{bornIn}, i) \cap FIL(\mathsf{bl_2}, i))$ must satisfy $F^{\mathcal{I}}$ such that $j : \{\mathsf{Canada}\}$ because of $\forall \mathsf{bl_2}.\{\mathsf{Canada}\} \in \mathcal{T}$ and $j : \neg\mathsf{Country}$ because of $\forall \mathsf{bornIn} \setminus \mathsf{bl_1}.\neg\mathsf{Country} \in \mathcal{T}$. However, this is not possible because now j must satisfy $j : \{\neg\mathsf{Canada}, \mathsf{Canada}\}$ because $\neg\mathsf{Country} \equiv \neg\mathsf{Afghanistan} \sqcap \cdots \neg\mathsf{Canada} \sqcap \cdots \neg\mathsf{Zimbabwe}$.

Unlike a noGood partition, a quasi-noGood partition's name p can be extended with elements from $\mathcal{DS}$ to represent a new partition p' which is not necessarily empty. For example, the partition $\mathsf{p_d}$ can be extended with $\{\mathsf{bl_1}\}$ to form the partition $\mathsf{p_e} = \{\mathsf{bl_1}, \mathsf{bl_2}\}$ and $\mathsf{p_e}$ is not a quasi-noGood partition nor it is a noGood partition. This is because the signature, F', of $\mathsf{p_e}$ no longer includes $\neg\mathsf{Country}$, enforced by the role complement of $\mathsf{bl_1}$, and elements in $\mathsf{p_e}$ do not need to satisfy the qualification imposed by $(\forall \mathsf{bornIn} \setminus \mathsf{bl_1}.\neg\mathsf{Country})$.

5.3.1 Initially Bounding the Size of the Search Space

Given a TBox $\mathcal{T}$, a global partitioning ($\mathcal{P}$) on the global decomposition set ($\mathcal{DS}$), a partition $\mathsf{p} \in \mathcal{P}$ can be safely discarded based on the following techniques.

Using role hierarchy relations Role hierarchy relations allow one to discard partitions either by enforcing super-role relations or by exploiting the role hierarchy relations as follows:

- *Enforcing super-role relations*: a partition p can be safely discarded if p represents the set of R-fillers (for some Role $R \in N_R$) not intersecting with the set of S-fillers (for some role $S \in N_R$) and S is a super-role of R ($R \sqsubseteq S$) in $\mathcal{R}$. This can be done because when the set of R-fillers does not intersect with the set of S-fillers, it is assumed to intersect with the complement[2] of S-fillers. Therefore p must be empty because it represents a set of role-fillers violating the semantics of super-role relations which enforce that every R-filler is also an S-filler whenever $R \sqsubseteq S$. For example, assuming the role hierarchy shown in Figure 36c is extended with the following RIAs: $\mathsf{hasDaughter} \sqsubseteq \mathsf{hasChild}$ and $\mathsf{hasSon} \sqsubseteq \mathsf{hasChild}$, then the role $\mathsf{hasChild}$ is a super-role for $\mathsf{hasDaughter}$ and hasSon. Rewriting the concept expression ($\geq 3\mathsf{hasDaughter.Child} \sqcap \geq 2\mathsf{hasSon.Child}$) w.r.t to the TBox in Figure 36 extends the role hierarchy for $\mathsf{hasChild}$ in Figure 36c with the one shown in Figure 37 where $\mathsf{hD_1}$ and $\mathsf{hS_1}$ are the newly introduced roles for $\mathsf{hasDaughter}$ and hasSon respectively. The partition $\mathsf{p_a} = \{\mathsf{hD_1}\}$ represents $\mathsf{hasDaughter}$-fillers that are neither hasSon-fillers ($\{\mathsf{hS_1}\} \cap \mathsf{p} = \emptyset$) nor $\mathsf{hasChild}$-fillers ($\{\mathsf{hC_1}, \mathsf{hC_2}, \mathsf{hC_3}, \mathsf{hC_4}\} \cap \mathsf{p_a} = \emptyset$), $\mathsf{p_a}$ can be safely discarded because every $\mathsf{hasDaughter}$-filler must also be a $\mathsf{hasChild}$-filler.

---

[2]Recall that the a role complement relation is not expressed in the DL $\mathcal{SHOQ}$ and the name for a role complement is not used as part of a partition's name.
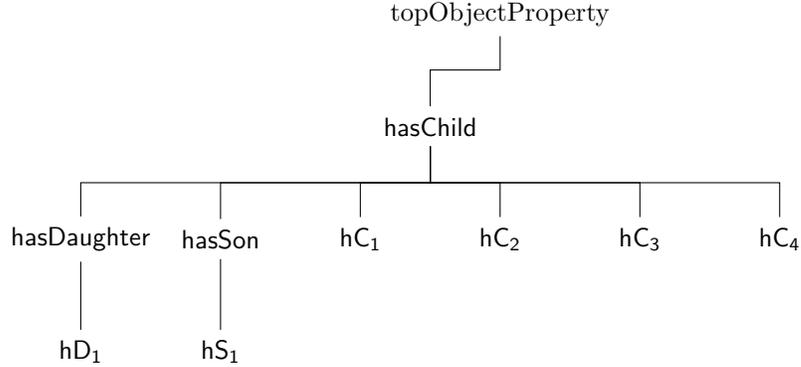
Figure 37: Role hierarchy for hasChild extended with hasDaughter ⊑ hasChild and hasSon ⊑ hasChild.

- *Exploiting role hierarchy relations*: a partition p can be safely discarded if p represents the set of R-fillers intersecting with the set of S-fillers such that the roles R and S do not share any super-role or sub-role unless p also represents a nominal o. This can be done because role-fillers from different parts of a hierarchy do not necessarily interact, and are sometimes assumed disjoint. For example, given that hasChild-fillers are members of the concept Child and bornIn-fillers are members of the concept Country, there is no need to consider the set of hasChild-fillers intersecting with the set of bornIn-fillers. The only case when two roles from a different hierarchy would share a role filler is when this role filler is a nominal. For example, in the case of $p_b = \{bl_2, ll_1, \{Canada\}\}$, the set of bornIn-fillers interacts with the set of livesIn-fillers for domain elements (member of Canadian and CanadianResident as defined in Figure 36) that live in the same country ($\{Canada\}$) where they were born even though the roles bornIn and livesIn do not share any super-role or sub-role. A less restricted version of this optimization is discussed in [OK99] and is applicable to the DL $\mathcal{SHN}_{\mathcal{R}\backslash}$ such that if two roles do not share a sub-role or super-role one can omit the partitions where their role-fillers intersect.

Using disjointness relations   A partition $p \in \mathcal{P}$ can safely be discarded if p can be identified as a noGood partition due to the disjointness relations between two concept descriptions C and D ($C \equiv \neg D \in \mathcal{T}$) such that given the signature F of p, the following holds:

- $C, D \subseteq F$. For example, all partitions $\mathsf{p_a}$ such that $\{\mathsf{Canada}, \mathsf{USA}\} \cap \mathsf{p_a} \neq \emptyset$ can be safely discarded from $\mathcal{P}$ for the TBox shown in Figure 36 because $\{\mathsf{Canada}\}$ and $\{\mathsf{USA}\}$ are disjoint[3] *nominals* that cannot refer to the same country.

- $R, S \subseteq F$ and $\forall R.C, \forall R.D$. For example, all partitions $\mathsf{p_a}$ such that $\{\mathsf{hS_1}, \mathsf{hD_1}\} \cap \mathsf{p_a} \neq \emptyset$ can be safely discarded from $\mathcal{P}$ for the TBox shown in Figure 36 because sons and daughters are disjoint due to $\mathsf{Male} \sqsubseteq \neg\mathsf{Female}$.

- $C, S \subseteq F$ and $\forall S.D$. For example, all partitions $\mathsf{p_a}$ such that $\{\mathsf{USA}, \mathsf{bl_2}\} \cap \mathsf{p_a} \neq \emptyset$ can be safely discarded from $\mathcal{P}$ for the TBox shown in Figure 36 because $\mathsf{bl_2}$-fillers must be identified with the nominal $\{\mathsf{Canada}\}$ due to $\forall \mathsf{bl_2}.\{\mathsf{Canada}\}$ and cannot interact with $\{\mathsf{USA}\}$ due to the disjointness relation between the two *nominals*.

In all cases, p is a noGood partition because no element can be assigned to p without causing a clash. Also, all partitions p' such that $p \subseteq p'$ can be safely discarded.

Using told nominal interactions with Roles   Some obvious interactions between *nominals* and role fillers due to the *hasValue* constructor or the $\forall$-constructor can be exploited to discard partitions.

Definition 5.3.4 (Told Nominal Filler)   A nominal $o \in N_o$ is said to be a Told nominal Filler if o is a qualifying concept ($\forall R.\{o\}$), or o is subsumed by a qualifying

---

[3]The disjointness relation between every two *nominals* is assumed implicitly in this TBox.

concept ($\forall R.C$ and $C \equiv \{o, a, b, c \ldots\}$), for a role R. In this case, R is said to be a told nominal role. For example, each one of the *nominals* enumerated in the definition of Country, in Figure 36, is a Told nominal Filler for the role bl_1, and bl_2 is a told nominal role for $\{Canada\}$.

A partition $p \in \mathcal{P}$ can safely be discarded if one of the following interactions between a nominal and a role holds.

- $\{R\} \subseteq p$, R is a told nominal role for $\{o\}$, and $\{o\} \not\subseteq p$. In this case, p is an empty partition because having $\{o\} \not\subseteq p$ implies that p represents the set of R-fillers intersecting with the complement of $\{o\}$, thus violating the semantics of told *nominals* with their told nominal roles; R-fillers must intersect with $\{o\}$ because R is a told nominal role for $\{o\}$. For example, the partition $p_a = \{bl_2, bl_1\}$ can be safely discarded from the partitioning for the TBox in Figure 36 because bl_2 is a told nominal role for $\{Canada\}$ and bl_2-fillers must be identified with $\{Canada\}$, but $\{Canada\} \not\subseteq p_a$. In such cases, p must be empty because it is a noGood partition and any partition p' such that $p \subseteq p'$ can also be discarded.

- $\{R'', o\} \subseteq p$, R' is a told nominal role for $\{o\}$, $\{R'\} \not\subseteq p$, and $R' \sqsubseteq R \in \mathcal{R}$, $R'' \sqsubseteq R \in \mathcal{R}$, and $\forall R \setminus R'.\neg\{o\} \in \mathcal{T}$. In this case, p is an empty partition because only R'-fillers can intersect with $\{o\}$; all other R-fillers intersect with the complement of $\{o\}$. For example, the partition $p_a = \{\{Canada\}, bl_1\}$ can be safely discarded from the partitioning for the TBox in Figure 36 because all bornIn-fillers not intersecting with bl_2-fillers cannot be identified with $\{Canada\}$. In such cases p must be empty because it is a quasi-noGood partition.

Discarding partitions using told nominal interactions with roles does not affect the algorithm's completeness. This is because a fresh role R' is introduced by the rewriting algorithm for each $\bowtie nR.C$. This means that no R'-fillers will be assigned

other than those intersecting with the interpretation of C. Therefore, eliminating partitioning for R'-fillers not intersecting with the nominal implied by C is safe.

Reducing the size of $\mathcal{DS}$ One could reduce a decomposed qualification over the same role into a single qualification due to the following : $\forall R.C \sqcap \forall R.D \iff \forall R.(C \sqcap D)$. Instead of having two qualifying concepts C, and D for R and the size of N_Q is increased by 2, one would have one qualifying concept E such that $(E \equiv C \sqcap D)$ and the size of N_Q is increased by 1. Hence, the size of $\mathcal{DS}$ is reduced.

5.3.2 Heuristic Guided Nominal Distribution

During the expansion of a *compressed completion graph*, the distribution of *nominals* over partitions, among the global partitioning $\mathcal{P}$, is somehow decided by the *ch*-Rule. In fact, a nominal o can be assigned to a partition $p \in \mathcal{P}$ only if the variable v mapped to p satisfies $(v \geq 1) \in \mathcal{L}_E(r_0)$ and $o \in \alpha(v)$. In practice, the *ch*-Rule is applicable to variables mapped to *nominals* partitions before any other rule, and the *ch*-Rule always branches on $v \leq 0$ before branching on $v \geq 1$.

The *heuristic guided nominal distribution* technique aims at minimizing the size of the search space by guiding the applicability of the *ch*-Rule in such a way that the search tree is explored by first considering choice points where each nominal is distributed over a more promising partition. It does so by exploiting told *nominals* and their told nominal roles to guess a distribution for *nominals* over partitions representing told nominal role fillers intersecting with the corresponding told *nominals*. Given a told nominal o and its corresponding told nominal role R, a partition p with $\{o, R\} \subseteq p$ is considered more promising for o than a partition p' such that $\{R\} \not\subseteq p'$. For example, when considering the partitioning for the TBox in Figure 36, the partition $p_a = \{bl_2, bl_1, Canada\}$ is considered more promising for the nominal Canada than

the partition $p_b = \{bl_1, \mathsf{Canada}\}$. Such a distribution is considered more promising because it takes into consideration the interaction between told *nominals* (Canada) and their told nominal roles (bl_2) in advance.

In order to make sure that *nominals* are distributed over their more promising partitions, an ordering of variables is imposed during the application of the *ch*-Rule. Since the *ch*-Rule branches on $v \leq 0$ before branching on $v \geq 1$, the variables corresponding to more promising partitions are ordered last. By doing this, and considering a depth-first search, all variables corresponding to less promising partitions for a nominal o are assigned $v \leq 0$ until the most promising variable v' is reached. Once the *ch*-Rule is applied on v', and in order to ensure that the semantics of o is preserved (see also the *ch-Rule look ahead* technique in Section 5.4.1), only one branch needs to be considered: $v' \geq 1$. This optimization only affects the order in which the *ch*-Rule branches on variables and therefore does not affect completeness of the algorithm.

5.4 Look Ahead Optimizations For Backtracking

Look ahead optimizations aim at reducing the size of the search space by discarding choice points as soon as a non-deterministic rule is applied. The following look-ahead techniques can be applied during the expansion of a compressed completion graph.

5.4.1 *ch*-Rule Look Ahead

The *ch*-Rule look ahead technique aims at discarding choice points when the *ch*-Rule is applied to a node x, with v occurring in $\mathcal{L}_E(x)$ and $\alpha(v) = p$, as follows:

- Discard branching on $v \leq 0$. The child graph in which $\mathcal{L}_E(x)$ is extended with $v \leq 0$ can be safely discarded if one of the following holds:

– $\{o\} \subseteq p$ and all the variables mapped to partitions representing the nominal o have been assigned to ≤ 0 in $\mathcal{L}_E(x)$. In this case, the child graph in which $\mathcal{L}_E(x)$ is extended with $v \leq 0$ will have an arithmetic clash because the encoded in-equation for the nominal o $(\xi(\{o\}, =, 1) \in \mathcal{L}_E(x))^4$ cannot be satisfied. Such is the case when the *heuristic guided nominal distribution* technique is applied and the *ch*-Rule is applied to the most promising variable for a given nominal.

– $\{R\} \subseteq p$ and assigning v the value zero will render the in-equation $(\xi(\{R\}, \bowtie, n) \in \mathcal{L}_E(x))$ infeasible. For example, the *ch*-Rule can safely discard branching on $v_a \leq 0$ if $hC_1 \in \alpha(v_a)$ and the in-equation $\xi(\{hC_1\}, \geq, 1)$ becomes infeasible because all other variables for hC_1 have been identified as *noGood* variables.

• Discard branching on $v \geq 1$. The child graph in which $\mathcal{L}_E(x)$ is extended with $v \geq 1$ can be safely discarded if assigning v the value i such that $i \geq 1$ results in an infeasible in-equation for some role R.

Discarding these choice points can be done in one of two ways: either by detecting obvious clashes or by detecting arithmetic clashes using the Constraint Solver. Note that when the Constraint Solver is invoked to check the feasibility of the system of in-equations during a *look ahead* phase, there is no need to consider integer solutions only and the solutions do not need to be kept. Looking for real solutions can be sometimes faster than looking for integer solutions, and during the *look ahead* phase, the branching decisions are affected in the cases of infeasibility; if no real solution exists, for sure no integer one exists either.

---

[4]For ease of presentation $\xi(\{o\}, =, 1)$ is used as an abbreviation for $\xi(\{o\}, \geq, 1)$ and $\xi(\{o\}, \leq, 1)$.

5.4.2 ⊔-Rule Look Ahead

The ⊔-Rule look ahead technique aims at discarding choice points when the ⊔-Rule is applied to a node x with $(C_1 \sqcup C_2 \sqcup \cdots \sqcup C_n) \in \mathcal{L}(x)$ and $n \geq 2$. The child graph G_i in which $\mathcal{L}(x)$ is extended with C_i can be safely discarded if one of the following holds:

- Infeasible in-equation: If C_i is an at-least restriction $(\geq nR)$ and all variables mapped to partitions representing R-fillers are *noGood* variables. The child graph G_i will have an arithmetic clash because the in-equation $\xi(\{R\}, \bowtie, n) \in \mathcal{L}_E(x)$ cannot be satisfied and the set of in-equations $(\xi(x))$ becomes infeasible.

- Clashing concept: if C_i is a concept description (or a nominal) and we have $\neg C_i \in \mathcal{L}(x)$. The child graph G_i will have a logical clash due to $\{C, \neg C\} \in \mathcal{L}(x)$. For example, given a node x such that $\{(C \sqcup (D \sqcap E)), \neg C\} \subseteq \mathcal{L}(x)$, the ⊔-Rule look ahead discards the branch adding C to $\mathcal{L}(x)$ because $\neg C \in \mathcal{L}(x)$. Expanding $(C \sqcup (D \sqcap E))$ becomes deterministic and adds $(D \sqcap E)$ to $\mathcal{L}(x)$.

The ⊔-Rule look ahead technique is similar to the boolean constraint propagation (BCP) (introduced in Section 3.2.1.2) with the advantage of avoiding disjuncts leading to arithmetic clashes, or leading to a violation of the *nominals* semantics. Recall that when the fil-Rule is applied and a node x is created such that $o \notin \mathcal{L}_p(x)$, $\neg o$ is added to $\mathcal{L}(x)$. Hence, the ⊔-Rule look ahead can avoid expansions which result in having $\{o, \neg o\} \in \mathcal{L}(x)$.

5.4.3 Active Roles Heuristic

The atomic decomposition technique considers all possible intersections between the sets of role fillers, *nominals*, and qualifying concepts, when computing the global partitioning. Although this global partitioning is required to ensure the completeness

$\mathcal{L}(\mathsf{a}) = \{\mathsf{Parent}, \mathsf{Person}, \geq 1\mathsf{hC}_1, \forall\mathsf{hC}_1.\mathsf{Child}, (\mathsf{Male} \sqcup \mathsf{Female}), \leq 1\mathsf{bl}_1, \forall\mathsf{bl}_1.\mathsf{Country}, \forall(\mathsf{bornIn}\backslash\mathsf{bl}_1).\neg\mathsf{Country}\}$

$\mathcal{L}(\mathsf{b}) = \{\mathsf{Child}\}$ (b) (c) $\mathcal{L}(\mathsf{c}) = \{\mathsf{Country}, \{\mathsf{Afghanistan}, \mathsf{USA}, \mathsf{Canada}, \ldots, \mathsf{Zimbabwe}\}\}$
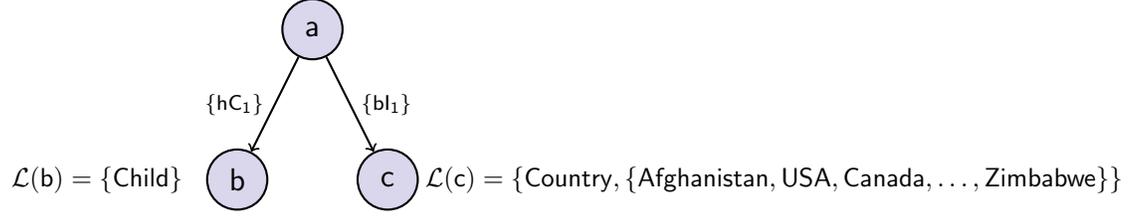
Figure 38: Clash free compressed completion graph for Parent.

of the algorithm, not all partitions are used, to distribute domain elements, within each satisfiability test. Consider for example the clash free CCG in Figure 38 for the satisfiability test of the concept Parent as defined in Figure 36. The partition $\mathsf{p_a} = \{\mathsf{hC}_1, \mathsf{hC}_2\}$ is part of the global partitioning $\mathcal{P}$ for $\mathcal{T}'$, however $\mathsf{p_a}$ is never used because no hC_2-fillers are needed to satisfy an at-least restriction ($\geq \mathsf{nhC}_2 \in \mathcal{L}(\mathsf{a})$) in order to decide on the satisfiability of the concept Parent.

The *active roles heuristic* aims at identifying those partitions that are potentially used, within a satisfiability test, in order to reduce the size of the search space by avoiding unnecessary partitions. It does so by labelling partitions that are potentially used as *active partitions* for *active roles* and restricting the applicability of the *ch*-Rule to *active variables* only.

Definition 5.4.1 (Active Variable)   a variable v mapped to a partition p is said to be active if the corresponding partition p is an *active partition*.

Definition 5.4.2 (Active Partition)   A partition $p \in \mathcal{P}$ is said to be active w.r.t $\mathcal{T}$ if every $R \in (p \cap N_{R'})$ is an *active role*. Also, all the qualifying concepts in p are qualifying concepts for *active roles*. For example, given the TBox in Figure 36 such that the role names in $N_{R'} \cup N_R$ follow the hierarchy shown in Figure 36c, and the CCG G as shown in Figure 38. The partition $\mathsf{p_a} = \{\mathsf{hC}_1, \mathsf{bl}_l\}$ is an *active partition*

because $\leq 1bl_1 \in \mathcal{L}(a)$ and $\geq 1hC_1 \in \mathcal{L}(a)$. However, the partition $p_b = \{hC_1, hC_2\}$ is not an *active partition* because there does not exists a node $x \in G$ such that $(\bowtie \, nhC_2) \in \mathcal{L}(x)$.

Definition 5.4.3 (Active Role)   A role $R \in N_{R'}$ is said to be active if there exists a node x in G such that $\bowtie \, nR \in \mathcal{L}(x)$, $\bowtie \in \{\leq, \geq\}$. For example, given the CCG shown in Figure 38 the role hC_1 is an active role because $(\geq 1hC_1 \in \mathcal{L}(a))$.

The set of *active roles* is updated every time the $\bowtie$-Rule is fired to a node x and is propagated to a node y whenever an edge is created between the node x and y. Also when encoding number restrictions into in-equations only *active variables* are considered. By delaying the applicability of the *ch*-Rule until a variable is being activated, this heuristics helps avoid unnecessarily branching on variables until it becomes necessary. In other words, this heuristics delays the use of all global partitions, and it is maximally effective in cases of having disjunctive or nested QCRs, otherwise all partitions are activated.

This optimization can interact with the *heuristic guided nominal distribution* technique because a told nominal role may not yet be activated initially (when the *ch*-rule is applied for nominal variables on r_o the set of *active roles* is empty). However the *active roles heuristic* can be set to automatically activate told nominal roles.

5.5   Look Back Optimizations for Backtracking

Back-jumping or *conflict-directed backtracking* are improved backtracking methods adapted to DL reasoning as dependency directed backtracking (DDB) [HT99] (introduced in Section 3.2.1.3). While adopting existing DDB techniques for DL reasoning helps prune the search space due to the $\sqcup$ rule, this technique does not prune the

148

choice points due to the *ch*-Rule, which is the rule responsible for the double exponential blow up of the search space. This section shows how backtracking can be optimized during the following two phases:

- Phase 1 - *Back-jumping* - During this phase an adapted form of DDB analyzes the source of a clash and decides how far to backtrack.

- Phase 2 - *Learning* - After consulting the sources of a clash, the algorithm can learn that a certain partition is a *noGood* partition or a *quasi-noGood* partition. This learned information is recorded as new constraints in form of *quasi-noGood* and *noGood variables.*

5.5.1 Backjumping

Recall from Section 3.2.1.3 that DDB aims at safely bypassing choice points; it works by replacing choice points that caused a clash with ones that are more promising to succeed. Once a clash is encountered, the clash sources are analyzed in such a way to allow the algorithm to back-jump to a choice point in the search space where the same clash is avoided. In the case of reasoning with $\mathcal{SHON}_{\mathcal{R}\backslash}$, three types of clash handlers are considered for analyzing the sources of a clash and setting the next choice point to visit in such a way that the clash is avoided.

Logical clash handler The logical clash handler is invoked whenever a logical clash is encountered in the label of a node x with $\{C, \neg C\} \subseteq \mathcal{L}(x)$ and C is a concept expression. Once this clash handler is fired, the dependencies of C and those of $\neg C$ are consulted for alternative choice points where either the source of C, or the source of $\neg C$, is not enforced on x. The algorithm is then set to back-jump to the nearest choice point. If no alternative choice point is found this means that C and $\neg C$ are always enforced on x and no clash-free CCG G with $x \in G$ exists. In this

case the node x is a representative for a *noGood* partition p because no element can be assigned to p without causing a clash. The variable v_x, representing the partition for x, is therefore a *noGood* variable and the algorithm can safely back-jump to the choice point where $v_x \leq 0$ thus bypassing all the choice points for $v_x \geq 1$.

$\sqcup$-**Rule clash handler**   Sometimes applying the $\sqcup$-Rule to a node x with $(C_1 \sqcup C_2 \sqcup \cdots \sqcup C_n) \in \mathcal{L}(x)$ returns an empty list of child graphs. This can happen when the $\sqcup$-*Rule look ahead* optimization (introduced in Section 5.4.2) is enabled and all child graphs G_i $(1 \leq i \leq n)$, each extending $\mathcal{L}(x)$ with C_i, must be discarded because they lead to obvious clashes. In this case, the $\sqcup$-*Rule clash handler* is fired and the dependencies of $(C_1, C_2, \ldots, C_n)$ are consulted for alternative choice points where the source of one of $(C_1, C_2, \ldots, C_n)$ is not enforced on x. The algorithm is then set to back-jump to the nearest choice point if one exists. Otherwise, similar to the case with the *logical clash handler*, v_x is set as *noGood* and the algorithm can safely back-jump to the choice point where $v_x \leq 0$.

Arithmetic Clash Handler   The arithmetic clash handler is used to detect and handle obvious arithmetic clashes. An obvious arithmetic clash is detected when an in-equation cannot be satisfied due to the assignment of the occurring variables; adding such an in-equation to $\xi(x)$ in $\mathcal{L}_E(x)$ will render the set of in-equations infeasible because it cannot be satisfied. By detecting these arithmetic clashes the clash handler can avoid unnecessary runs of the Simplex procedure, and allow a smart back-jumping to a graph where the in-equation is not necessarily infeasible. The arithmetic clash handler is invoked every time an unsatisfiable in-equation is encountered by considering the following cases:

- The in-equation is an at-least restriction and all the (*active*) variables are assigned values such that the cardinality of the in-equation is not satisfied.

– All the variables are *noGoods*. This clash can be detected as early as when the $\bowtie$-rule is being applied to a node x for an expression $\bowtie nR$, the variable assignment for R is checked and the node is set to have an arithmetic clash.

– The sum of the variable values assigned by Simplex does not satisfy the cardinality. For example, if all the variables have been assigned to ≤ 0 by the *ch*-Rule. This clash can be detected as early as when the $\bowtie$-rule is being applied to a node x having $\geq nR \in \mathcal{L}(x)$. In case the in-equation is not encoded, the clash handler randomly chooses a variable v which is not a *noGood* and back-jumps to the branching point where v is assigned to ≥ 1 by the *ch*-Rule. If no other variable assignment is possible then the in-equation cannot be satisfied for the node, and the clash handler checks for alternative choice points for x where the in-equation is not added to the label. If no such alternative exists, the node x is set to *noGood*.

- The in-equation is an at-most restriction and all variable assignments (due to previous runs of Simplex) do not satisfy the cardinality (their sum is greater then the cardinality). For example, if all the non-zero variables are nominal variables, then these variables can only be assigned the value 1. If there are more nominal variables than the cardinality, then the in-equation is infeasible, and the handler chooses a non-nominal variable v_R mapped to the partition representing R-fillers for the corresponding role R, and sets the algorithm to back-jump to the choice point where $v_R \leq 0$.

- *ch*-Rule: if the *ch*-Rule is being applied to a node x on a variable v_a for a nominal o or a role R_a and the list of child graphs is empty because v_a cannot be ≤ 0 nor ≥ 1 while satisfying the in-equation for R_a ($\xi(R_a, \bowtie, n)$) or the semantics of the nominal o ($\xi(o, \bowtie, 1)$). This means that a different distribution of the variables for o is required. When this clash handler is fired, it randomly

picks an active variable v_b which is set to $v_b \leq 0$ in the current branching graph and sets the algorithm to back-jump to the branching graph where $v_b \geq 1$.

- Nodes are always checked for an arithmetic clash with an at-least restriction where all the active variables become $noGood$s.

This optimization can interact with the *active roles heuristic* optimization. For example, in the case when the clash handler chooses a variable v_p and sets the algorithm to back-jump to the CCG G' with $v_p \geq 1$, but the partition p mapped to v_p is such that $R \in p$ for a role R that is not yet activated in G'. Expanding G' with $v_p \geq 1$ is not possible because the *active roles* conditions for the applicability of the *ch*-Rule cannot be met. In order to ensure completeness, the set of active roles is back-propagated as follows: once a clash occurs at a given node x such that the algorithm is set to back-jump to G', the set of *active roles* for x is copied either to x in G', or to r_o if x is a role-successor that is not yet created in G'.

5.5.2 Learning

A partition p which survives the partition elimination techniques during preprocessing is not necessarily non-empty. This is because some clashes (see Definition 4.5.5) cannot be easily detected by examining TBox and RBox axioms. The algorithm might assign elements to a partition p and discover later that assigning elements to p leads to unavoidable clashes and therefore p must be empty. Since the algorithm discovers those empty partitions due to clashes, clash handlers are adapted to detect and set *quasi-noGood* or *noGood* partitions by studying the sources of a clash (as described in the previous section). However, the same *noGood* may be rediscovered over and over again as the algorithm explores different branches in the search space. To avoid running into the same clash, the back-jumping algorithm can be augmented

with a learning phase which is called *back-jump learning*, and which is responsible for capturing the characteristics of a clashing partition as well as learning which elements are responsible for the clash in order to detect other non-empty partitions. The learned information about empty partitions is recorded as global *noGood* variables.

Back-jump Learning   Whenever a clash handler encounters a clash with no alternative choice point for the clashing descriptions, the search algorithm is said to encounter a dead-end. The back-jump learning technique can be implemented as soon as a dead-end is encountered and just before the algorithm is ready to back-jump. Back-jump learning works as follow:

- Once a dead end is encountered identify the clashing node's partition, p.

- Identify the partition elements responsible for the clash. For example, if $p = \{S_1, R_1, R_2, o_1\}$ and the clash encountered was due to $\{C, \neg C\} \in \mathcal{L}(x)$ then the dependencies for C and $\neg C$ are checked. If the dependency for C is $\forall R_1.C$ and the dependency for $\neg C$ is $\forall R_2.\neg C$, then this means that R_1 and R_2 are the elements of p responsible for the clash. One might argue that such partitions might already have been eliminated due to the *disjoint qualification* technique. However, these cases might not be as obvious; consider for example $(\geq 1S_1 \sqcap \forall S_1.(\geq 1R_1 \sqcap \forall T.C) \sqcap \geq 1R_2 \sqcap \forall R_2.\neg C)$ with $R_1 \sqsubseteq T$, $R_2 \sqsubseteq T$.

- Learn the clashing combination of role fillers and generalize. For example, if having $\{R_1, R_2\} \subseteq p$ inevitably leads to a clash, then one can learn that every partition p' having $\{R_1, R_2\} \subseteq p'$ is inevitably empty. For every learned empty partition p', the *noGood* constraint is recorded as an index to the single variable mapped to p instead of storing p and its signature.

This learning technique enables some form of caching (see section 3.2.1.4) the unsatisfiability of the signature of a noGood partition. Unlike caching, and due to

its learning phase, this technique does not work with problems of repeated structure only. It is also estimated not to degrade performance like known learning algorithms for graph-based search where the recorded dependencies and learned information cause performance degradation in some cases. The mapping between variables and partitions, allows *noGood* constraints to be stored as indexes for the corresponding variables (see Chapter 6 for more details on how the indexing of variables is done). Even in the case of assigning multiple *noGood*s, the process of storing and retrieving *noGood*s is not expected to degrade performance.

5.6   Look Ahead with Back-jumping

The look ahead techniques presented in the previous sections aim at discarding choice points of non-deterministic rules. In some cases of applying these non-deterministic rules, the look-ahead techniques discard all possible choice points resulting in an empty list of child graph. When this happens, no further expansion of the CCG is possible and naïve backtracking is triggered. In the case of the *ch*-Rule, the look ahead technique is adapted with smarter back-jumping than naïve backtracking using the *ch-Rule back-jumping heuristics*.

ch-**Rule backjumping heuristic**   The *ch-Rule back-jumping heuristics* chooses a variable v and sets the algorithm to back-jump to the branching point where the choice for v is more likely to succeed. Here is how it works: if the *ch*-Rule is applied to a node x with $v \in \mathcal{L}_E(x)$, and

- the choice point for $v \leq 0$ is discarded because the in-equation in $\mathcal{L}_E(x)$ encoding an at-least restriction becomes infeasible, and

- the choice point for $v \geq 1$ is discarded because v is a *noGood* variable.

Then choose a variable v_b within the set of variable occurring in $\mathcal{L}_E(x)$ and set the algorithm to back-jump to the choice point where $v_b \geq 1$. The intuition is that the completion graph G' with $v_b \geq 1 \in \mathcal{L}(x)$ is more likely to be clash free. In order to preserve completeness, this heuristic guesses the variable with the closest branching point.

5.7   Lazy Partitioning

The optimization techniques presented in the previous sections affect either the number of choice points a non-deterministic rule can have, or the number of times a rule can fire. For example, the *active roles heuristic* affects the number of times the *ch*-Rule can fire by restricting its applicability to *active* variables. On the other hand, these techniques do not avoid the overhead of computing exponentially many partitions. This is because a partition is computed before it is flagged as *quasi-noGood*, *noGood*, or *active* and there are exponentially many partitions[5]. This means that the algorithm is best-case exponential because computing all partitions initially requires exponentially many steps and is likely to cause an exponential number of (*quasi-noGood*, *noGood*, or *active*) checks to be triggered. On the other hand, such exponential computational overhead is sometimes unnecessary. Consider the CCG G shown in Figure 39 where applying the $\sqcup$-Rule to the node x returns a list of child graphs $G_1, \ldots, G_i$ where $\mathcal{L}(x)$ is extended with $(\geq n_j R_j \sqcap \forall R_j.D)$ in each G_j, $(1 \leq j \leq i)$. In each graph G_j the satisfiability of $(\geq n_j R_j \sqcap \forall R_j.D)$ is checked independently. In a sense, R_j-fillers do not interact with R_{j-1} or R_{j+1}-fillers and therefore computing the partitions where these R-fillers intersect is unnecessary.

---

[5]The total number of partitions is exponential to the size of $\mathcal{DS}$ as defined in Chapter 4 Lemma 4.6.1 as $\#\mathcal{DS} = \#\{N_{R'} \cup N_Q \cup N_O\}$.

$$\mathcal{L}(x) = \{\geq mS_1, \forall S_1.E, ((\geq n_1 R_1 \sqcap \forall R_1.D) \sqcup \cdots \sqcup (\geq n_i R_i \sqcap \forall R_i.D))\}$$

$\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\geq n_1 R_1 \sqcap \forall R_1.D)\}$ — x in G_1 · · · · · · · x in G_i — $\mathcal{L}(x) = \mathcal{L}(x) \cup \{(\geq n_i R_i \sqcap \forall R_i.D)\}$
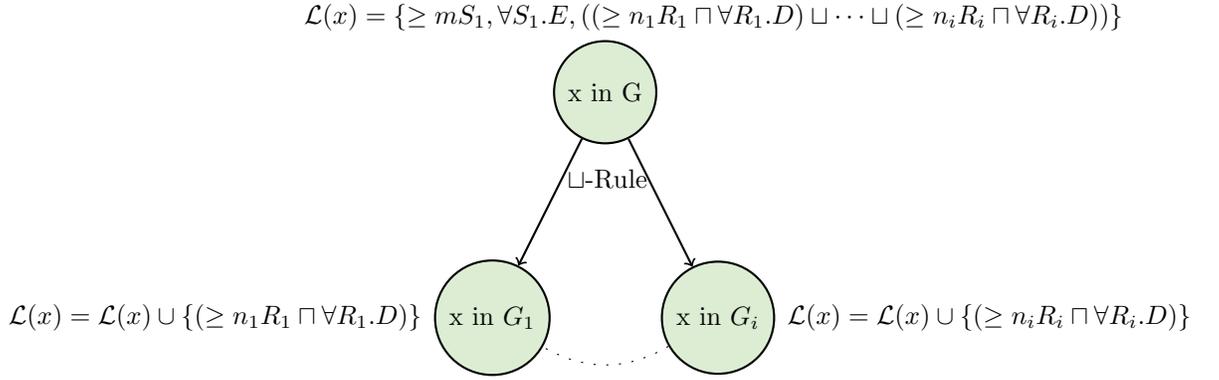
Figure 39: Expansion tree of a clash free CCG which shows that initially computing a global partitioning is not necessary.

Lazy partitioning aims at delaying the process of computing partitions for a certain role R until necessary. It does so by applying partitioning incrementally at each node using an incremental decomposition set. The incremental decomposition set is composed of *nominals* and *active roles* only. This is because until a role R is *active* there is no need to consider the partitions for R-fillers. Here is how it works: the set of *active roles* for a CCG G is defined as $\mathcal{AR} = \{R|\ R \text{ is active}\}$ (See definition 5.4.3 for a formal definition of *active roles*). A role R is activated and added to $\mathcal{AR}$ every time the $\bowtie$-Rule is applied to a node x. An incremental decomposition set for G is defined as $\mathcal{IDS} = \bigcup_{R \in \mathcal{AR}} \mathcal{D}_R \cup N_{\text{o}} \cup N_{\text{Q}}$[6] and an incremental partitioning set $\mathcal{IP}$ is defined as an incremental partitioning on $\mathcal{IDS}$. Every time the $\bowtie$-Rule is applied to a node x for $\bowtie nR_a \in \mathcal{L}(x)$, R_a is added to $\mathcal{IDS}$ and the incremental partitioning set is expanded such that $\mathcal{IP} = \mathcal{IP} \cup \bigcup_{p \in \mathcal{IP}} (p \cup \{R_a\}) \cup \bigcup_{o \in N_o} \{R_a, o\} \cup \{R_a\}$[7]. In the case of the example in Figure 39, the $\mathcal{IDS}$ for G_1 consists of $\{S_1, R_1\}$ which is

---

[6]Note also that N_{Q} denotes the set of the qualifying concepts for *active roles* only.

[7]With the assumption that *nominals* are disjoint, only one nominal is included in a partition. Otherwise intersections between *nominals* need also to be considered.

greatly smaller than the global decomposition set $\mathcal{DS} = \{S_1, R_1, \ldots, R_i\}$ and only 2^2 partitions need to be computed instead of 2^{i+1} partitions.

This technique can greatly reduce the size of the search space. However, similar to the *active roles heuristics*, and if not carefully implemented, this technique might interact with other optimizations such as *backjumping* and violate the completeness of the algorithm. Therefore, every time back-jumping is triggered the set of *active roles* is back-propagated and partitions are considered even if these roles are not activated in the graph where the algorithm is set to back-jump. Also, every time the *ch-Rule back-jumping heuristics* needs to guess a variable, it must choose an active one.

5.8   Lazy Nominal Generation

As a first step, the hybrid algebraïc algorithm always guesses an initial distribution of all *nominals* occurring in the TBox. Considering all *nominals* distributions is necessary to ensure the completeness of the algorithm. However, in some cases, similar to what was discussed in the previous section not all *nominals* come into play in each satisfiability test.

For example, there exists cases where *nominals* do not interact with Roles. Computing intersections of these *nominals* with every Role in $\mathcal{DS}$ ($\mathcal{IDS}$) becomes an unnecessary overhead. The *lazy nominal generation* technique aims at delaying the generation of nominal nodes until it becomes necessary. It does so by exploiting Told nominal relations with their told nominal roles, such as as soon as a told nominal role is activated, the corresponding Told nominal can be activated as well. Once a nominal is activated, it is considered within the decomposition set where its interaction with Roles is taken into consideration. Otherwise, and in order to preserve the *nominals* semantics, the nominal is distributed over a partition p representing the nominal itself without intersecting with any element in $\mathcal{DS}$ ($\mathcal{IDS}$).

5.9 Discussion

The optimizations presented in this chapter focus on optimizing consistency reasoning through optimizing the satisfiability testing. This does not mean that the algebraïc reasoning algorithm is not amenable to other optimizations developed to simplify and preprocess the ontology such as *lazy unfolding* and *absorption* (introduced in Chapter 3) which have been widely used to reduce the size of the search space due to a large number of axioms in the TBox. In fact, this chapter focuses more on optimizations aiming at improving the satisfiability test where the complexity is due to the hybrid nature of the algorithm.

The preprocessing optimizations discussed to initially bound the size of the search space not only minimize the size of $\mathcal{P}$ by avoiding unnecessary computations of *noGood* partitions, but also minimize the number of choice points of the non-deterministic *ch*-Rule which becomes deterministic with *noGood* variables. If the *ch*-Rule is applied to a node x with a *noGood* variable v, then the number of child graphs is reduced to 1 because the only possible choice point for v is $v \leq 0$. In this case, the label $\mathcal{L}_E(x)$ can be extended without creating a child graph as is done with deterministic rules. On the other hand, a possible drawback of using *role hierarchy relations* to enforce the RBox hierarchy relations is that the re-use of individuals becomes restricted to those identified with *nominals* and role fillers.

The $\sqcup$-*Rule look ahead* optimization can be seen as some form of *boolean constraint propagation* (BCP) on the disjuncts allowed in a node's label. Therefore its effectiveness could be relatively limited and problem dependant. The use of *noGoods* during *back-jump learning* can be seen as a form of caching the unsatisfiability of the signature of the *noGood* partition.

Dependency directed backtracking (DDB) is adopted by most tableau-based reasoners due to its efficiency. The advantage of the technique presented in this chapter

over the well known DDB, for DL reasoning, is that the dependencies for a concept description, within the label of a node, can be tracked down to the application of one non-deterministic rule. Whereas with tableau algorithms implementing the non-deterministic *choose*-Rule (as introduced in Section 2.2.2), the dependencies of a concept description within the label of a node need to consider the dependencies due to the application of two non-deterministic rules: the *choose*-Rule and the $\sqcup$-*Rule*. The interaction between the two dependencies has been linked to the performance degradation of tableau reasoners with QCRs. This interaction has also been identified as an open problem rather than an implementation detail [Hor02].

The *active role heuristic* and the *lazy partitioning* optimizations both simulate some form of local partitioning, as in [Far08], by using the partitions that are locally applicable to a node. This can greatly enhance performance with $\mathcal{SHQ}$ TBoxes where a global partitioning is not necessarily required. The advantage of using these optimizations is best reached in case of disjunctive ($\geq nR \sqcup \geq nS$) or nested ($\geq nR \sqcap \forall R.(\geq nS)$) QCRs. In case of disjunctions and when branching on $\geq nR$ there is no need to consider intersections with S and vice versa. The challenge with these optimizations is that once enabled the algorithm loses some kind of a look-ahead especially with the *heuristic guided nominal distribution* optimization. If a told nominal role is not yet activated one cannot favour a nominal partition intersecting with this role. However, one can still adapt *lazy partitioning* and *active role heuristic* so that when *nominals* partitions are initially computed, the atomic decomposition still considers the partitions where *nominals* intersect with their told nominal role fillers even if the corresponding roles are not yet activated. Unlike the *active partitions* heuristic, *Lazy partitioning* does not compute partitions until all corresponding Roles have been activated. The *active role heuristic* does not avoid computing a global partitioning and therefore is likely to be less efficient than *lazy*

partitioning.

Lazy nominal generation is very similar to the *lazy forest generation* [PCS06] technique (introduced in Section 3.2.1.6), used to delay the generation of *nominals* until necessary. However, once a clash occurs *lazy forest generation* does not avoid computing the initial forest, whereas *lazy nominal generation* propagates information through back-jumping.

Finally, it would be interesting to investigate if the form of caching enabled by the *back-jump learning* technique could be exploited to yield a single exponential algorithm as in [MM00, Din08].

5.10 Conclusion

This chapter discussed a range of optimization techniques that can be used to improve non-determinism in the algebraïc reasoning algorithm. Some of these techniques are based on existing well known optimizations for search based tableau algorithms in general, and DL tableau algorithms in particular such as DDB. However, they are designed and adapted to work with the algebraïc reasoning approach. The primary goal of discussing the optimizations is to prove the utility of the algebraïc algorithm in realistic applications. This will be assessed in Chapter 7 through an empirical analysis.

Chapter 6

HARD - A Hybrid Algebraïc Reasoner for DL

This chapter presents a prototype Hybrid Algebraïc Reasoner for DL (HARD). HARD is based on the algebraïc tableau reasoning algorithm presented in Chapter 4, and implements the optimization techniques discussed in Chapter 5. HARD will be used as a test bed for ReAl DL (Reasoning Algebraïcally with DL). The main goal of HARD is to show the practical merits of combining algebraïc reasoning with standard tableau reasoning for DL with nominals and QCRs, as well as the impact of the optimization techniques proposed. Given an ontology file, HARD decides whether the underlying ontology is consistent or not. This chapter presents the general architecture of HARD which is implemented using JAVA (JRE 1.6) and OWL-API (2.2)[1] and consists of the following main components:

- Ontology Loader - The Ontology Loader is responsible for loading an ontology selected by the user. The Ontology Loader is described in Section 6.1.

---

[1] `http://owlapi.sourceforge.net/`

- Configuration Controller - The Configuration Controller is responsible for checking and storing different user preferences such as which optimizations to enable or disable. The Configuration Controller is described in Section 6.2.

- Reasoner Manager - The Reasoner Manager is responsible for managing the tasks between the different components of HARD. It is described in Section 6.3.

- Preprocessor - The Preprocessor is responsible for making sure that the input ontology is of the format accepted by the reasoner by applying the preprocessing algorithm required for ReAl DL. It is described in Section 6.4.

- Tableau Reasoner - The Tableau Reasoner is responsible for applying the tableau expansion rules in the proper order. It is described in Section 6.5

- Constraint Solver - The Constraint Solver is responsible for solving the set of in-equations generated by the Tableau Reasoner using the Simplex [CLRS01] procedure. It is described in Section 6.6.

- Clash Handler - The Clash Handler is responsible for detecting and handling clashes. Different clash handlers are used to detect and handle different types of clashes. The Clash Handler is described in Section 6.7.

6.1   Ontology Loader

The Ontology Loader allows HARD to accept a test case ontology in the form of .owl file designed using an ontology editor such as Protégé[2] and saved in the RDF/XML format.[3] One can load an ontology by selecting an .owl file residing on the computer directory. The ontology file is loaded into an `OWLOntology` object that is manipulated by the OWL-API.

---

[2]http://protege.stanford.edu/
[3]http://www.w3.org/TR/rdf-syntax-grammar/

6.2 Configuration Controller

The Configuration Controller allows HARD to fetch user preferences before running a KB consistency test. These preferences are stored as `static` global variables members of a `Preferences` class which is also global. Based on the different components of HARD, there are different types of preferences that can be enabled. They can be grouped as follows:

Reasoner Preferences To assess the performance of ReAl DL, one needs to compare it against existing reasoning algorithms implemented by SOTA (state-of-the-art) reasoners. Therefore, JAVA APIs for each of those reasoners have been integrated into HARD's application implementation. These preferences allow the user to select the reasoner to perform the KB consistency check. A user interface with radio buttons allows the user to select one of:

- Fact++(version 1.4.1):[4] a highly optimized tableau-based DL reasoner implemented in C++ and supporting OWL-DL[5] and partially OWL 2.[6] A system description of Fact++ can be found in [TH06].

- Hermit (version 1.2.3):[7] a recent hypertableau-based DL reasoner implemented in JAVA and supporting OWL 2.

- Pellet (version 2.2.0):[8] a highly optimized tableau-based DL reasoner implemented in JAVA and supporting OWL 2. Pellet was the first DL reasoner to handle nominals [PCS06]. A system description of Pellet can be found in [SPG+07].

---

[4]`http://code.google.com/p/factplusplus/`
[5]OWL-DL is a sublanguage of OWL which places a number of constraints on the use of the OWL language constructs. See `http://www.w3.org/TR/owl-ref/` for more details.
[6]`http://www.w3.org/TR/owl2-overview/`.
[7]`http://hermit-reasoner.com/`
[8]`http://clarkparsia.com/pellet/`

- RacerPro (version 2.0):[9] a highly optimized tableau-based DL reasoner implemented in LISP and supporting the DL $\mathcal{SHIQ}$. RacerPro implements algebraïc reasoning for dealing with QCRs based on the algorithm presented in [HTM01]. RacerPro tests are invoked using JRacer 2.0.[10] A system description of Racer can be found in [HM01b]. A system description of Hermit can be found in [SMH].

- HARD: a prototype reasoner based on ReAl for the DL $\mathcal{SHOQ}$ implemented in JAVA and equipped with the optimization techniques discussed in Chapter 5. Due to time constraints, HARD does not handle qualifying concepts (see Section 6.8 for more details) and is described in detail in the following sections.

Optimizations Preferences   In order to assess the efficiency of the optimization techniques proposed in the previous chapter, one can select which optimizations to enable during the reasoning procedure. A user can turn an optimization ON or OFF. Among the optimizations discussed in Chapter 5, note that the *ch-Rule look ahead optimization* described in Section 5.4.1 is implemented in such a way to detect obvious arithmetic clashes. Invoking the Constraint Solver during the look ahead phase of this optimization was not implemented due to time constraints.

Constraint Solver Preferences   The generated system of in-equations can be solved using Simplex methods. The user can choose between using an integrated implementation of the Simplex procedure as in [Far08], and using the LPSolver[11] which is a JAVA-based API designed to solve linear programming problems using the Simplex method. Both implementations rely on branch and bound techniques in finding the integer solution.

---

[9]http://www.racer-systems.com/
[10]JRacer is network-based client JAVA API for accessing RacerPro.
[11]http://code.google.com/p/lpsolver/

If the LPSolver is selected, the user can select more preferences such as setting the objective function to minimize one of the following sets of variables: those that have been assigned by a previous solution, those occurring in at-most restrictions only, or all variables.[12]

Output Preferences

- Generate a log file. When this option is enabled, and the HARD reasoner is selected, a file name `SHOQ-‘‘ontologyfilename’’-log.txt` is generated. This file contains a trace of each function called along with time stamps and error descriptions. This log helps debugging the system and monitoring the flow between the different reasoner components. The log file helps the process of validating the execution of the algorithm and allows a better tracking of errors. In case of an exception or an error, it is easy to report the last execution before the error and trace it back.

- Generate a statistics file. When this option is enabled during execution of the reasoning procedure, information regarding the execution of the consistency test is collected. When the HARD reasoner is selected, this information includes: preprocessing time, partitioning time, run-time, number of arithmetic clashes, number of logical clashes, etc. Otherwise, the file contains the times taken to initialize the selected reasoner, load the ontology file, preprocess the ontology, and perform a satisfiability test along with the total time since initialization. At the end of the TBox consistency test an excel file is generated `Runtime-‘‘ontologyfilename’’.xls` containing the collected information. This file allows one to find out where most of the time is spent and which types of clashes are more frequent etc.

---

[12]Setting the objective function to maximize was much slower and therefore it was disabled.

- Generate a *compressed completion graph* file. When this option is enabled and in case the input ontology is consistent, the CCG information is printed into a separate text file named `CCGraph.txt`. This file helps validate the completion model generated by HARD.

- Generate an ontology information file. When this option is enabled, a file named `SHOQ-onto.txt` is generated containing information about the preprocessed ontology in $\mathcal{SHON}_{\mathcal{R}\backslash}$ including information collected when preprocessing the ontology (e.g. the total number of nominals, QCRs, etc).

When the performance is evaluated only the "generate statistics file" is enabled, as generating the other files results in unnecessary overhead.

6.3   Reasoner Manager

As shown in Figure 40, the Reasoner Manager is responsible for the overall program flow. It coordinates and manages different tasks between different components. The Reasoner Manager first passes the preferences selected by the user to the Configuration Controller, and then passes the ontology file to the Ontology Loader, which in turn loads the ontology into an `OWLOntology` object that can be manipulated using the OWL API. The corresponding Preprocessor is then invoked (based on the selected reasoner) to apply the rewriting algorithm on the ontology object. Once preprocessing is completed, the ontology is passed to the Tableau Reasoner which performs a KB consistency test. In the case of the HARD reasoner being selected, and during execution of the tableau expansion rules, the Reasoner Manager takes care of calling the Constraint Solver with the encoded in-equations and returning the solution to the Tableau Reasoner. Whenever a clash occurs, the Reasoner Manager makes sure that the appropriate Clash Handler is triggered. Upon completion, the Reasoner Manager

166

generates the output files based on the preferences selected by the user.
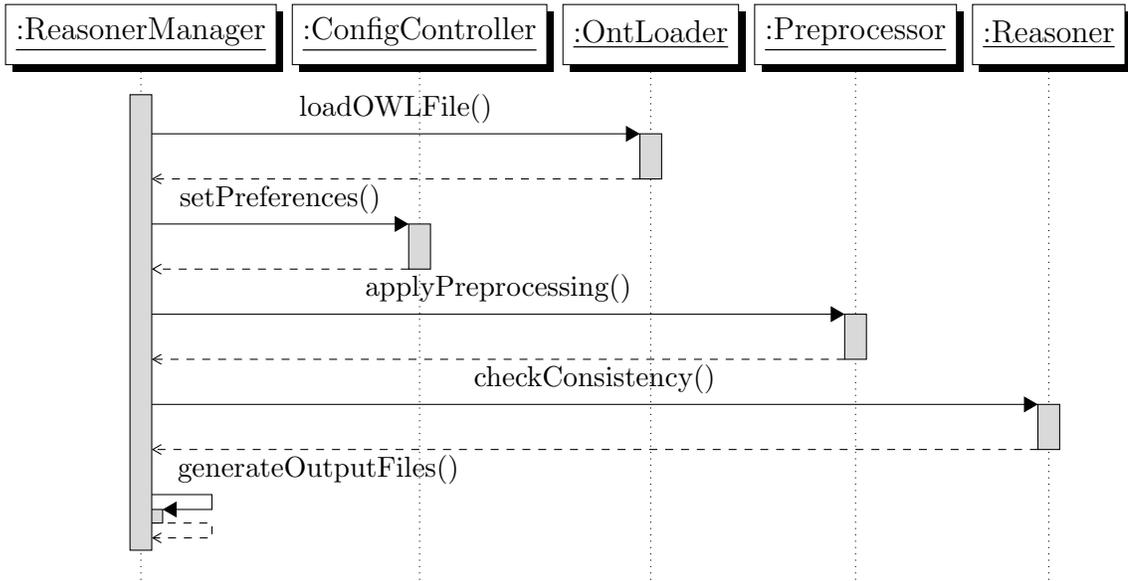
Figure 40: General sequence diagram showing the control flow of the Reasoner Manager.

6.4 Preprocessor

The Preprocessor applies the necessary processing to the `OWLOntology` object loaded by the Ontology Loader and returning the new processed `OWLOntology` object to the Reasoner Manager. Preprocessing is performed directly on the `OWLOntology` object without affecting the original ontology file.

Since the OWL API is used to manipulate the ontology object (`OWLOntology`), we show the correspondence between DL syntax and OWL syntax in Table 4 and use the OWL syntax when referring to implemented procedures throughout this chapter. Preprocessing is performed in case the HARD reasoner was selected. Otherwise, each reasoner's API is responsible for applying any necessary preprocessing required. Preprocessing an `OWLOntology` object before passing it to the Tableau Reasoner involves the following procedures:

167

$\mathcal{SHOQ}$ DL notation	OWL notation
$C \sqsubseteq D$	SubClassOf($C\,D$)
$C \equiv D$	EquivalentClasses($C\,D$)
$\top$	Thing
$\bot$	Nothing
$\neg C$	ObjectComplementOf(C)
$C \sqcap D$	ObjectIntersectionOf($C\,D$)
$C \sqcup D$	ObjectUnionOf($C\,D$)
$\leq nR.C$	ObjectMaxCardinality($n\,R\,C$)
$\geq nR.C$	ObjectMinCardinality($n\,R\,C$)
$\geq nR.C \sqcap \leq nR.C$	ObjectExactCardinality($n\,R\,C$)
$\forall R.C$	ObjectAllValuesFrom($R\,C$)
$R \sqsubseteq S$	SubObjectProperty($R\,S$)
Concept name A	OWLClass
C, A, $\{o\}$, R	OWLObject
$\{o\}$	OWLIndividual
R	OWLObjectProperty

Table 4: Correspondence between DL syntax and OWL syntax.

Enforcing *negation normal form* Since all concept descriptions (referred as `OWLDescription`) are assumed to be in their *negation normal form* (NNF), as introduced in Definition 2.2.1 of Chapter 2, this procedure replaces all `OWLDescription`s occurring in `OWLOntology` with their NNF (i.e., negation occurs only in front of concept names or nominals).

Converting $\mathcal{SHOQ}$ descriptions to $\mathcal{SHON}_{\mathcal{R}\backslash}$ descriptions Since the reasoning algorithm handles expressions conforming with the syntax and semantics of the DL $\mathcal{SHON}_{\mathcal{R}\backslash}$ as introduced in Section 4.1.1, `OWLDescription`s are replaced with equisatisfiable ones in the $\mathcal{SHON}_{\mathcal{R}\backslash}$ NNF format. This is done by implementing Algorithm 4.1.1, which is the rewriting algorithm responsible for rewriting QCRs into unqualified cardinality restrictions and extending the role hierarchy with the newly introduced roles, as introduced in Section 4.1.1.

Collecting the global decomposition set elements When `OWLDescription`s are being processed, the set of role names (N_R) occurring in `OWLOntology` is formed as a set of `OWLObjectProperty` objects. This set is extended with the set of $N_{\mathrm{R}'}$ every time a newly introduced role is created. Similarly the set of nominals (N_o) is formed as a set of `OWLIndividual` objects and is extended every time a nominal is encountered within an `OWLDescription`. The set of *qualifying concepts* is not maintained because *qualifying concepts* are not handled by the prototype reasoner.

Bookkeeping told nominals and their roles As `OWLDescription`s are processed, they are also analyzed (depending on which optimization is turned on) in such a way that told nominals are identified and stored.

Note that the bookkeeping of negated *qualifying concepts* as required for the handling of *qualifying concepts* is not implemented because the prototype reasoner does not handle *qualifying concepts* (See Section 4.1.1 for a review on the handling of *qualifying concepts*). Hence, even though the resulting DL $\mathcal{SHON}_{\backslash\mathcal{R}}$ is not closed under negation, the reasoner will no longer negate an `OWLDescription` after preprocessing is complete.

6.5 Tableau Reasoner - Inference Engine

The Tableau Reasoner implements the tableau calculus presented in Section 4.5 and is responsible for deciding on a KB consistency check. It does so by constructing a compressed completion graph (`CCGraph`) using the tableau expansion rules while ensuring that these rules are applied in the proper priority by enforcing a rule application strategy. This reasoner works on a `CCGraph` object, as illustrated in Figure 41, which consists of a set of proxy nodes (`pNodes`) and a set of edges (`pEdges`) between these nodes. Each proxy node object (`ProxyNode`) contains an `OWLIndividual` instance (`owlInd`) representing a domain element or a nominal, and a cardinality value (`cardinality`) denoting the number of elements represented by this proxy node. If the proxy node represents a nominal, then the cardinality value is set to 1; otherwise, it is set to a non-negative integer value.
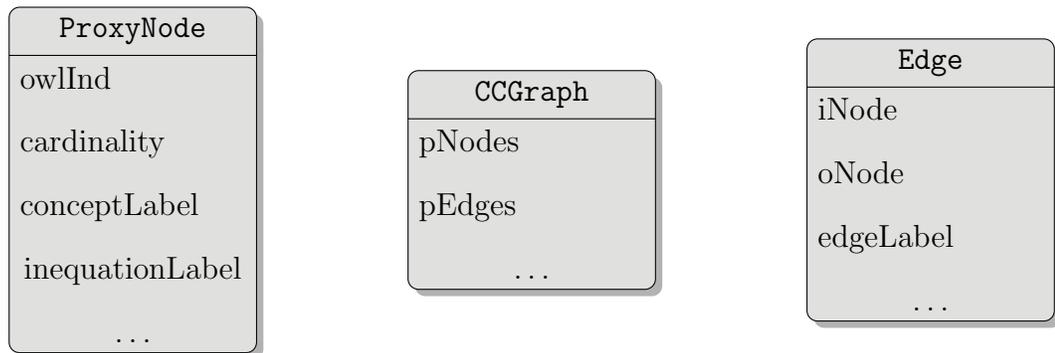
Figure 41: Representation of the CCGraph object class.

Each `Edge` object consists of an incoming node (`iNode`), an outgoing node (`oNode`), and a label (`edgeLabel`) representing the role relations between the individuals represented by the two nodes. The Tableau Reasoner builds a `CCGraph` by applying expansion rules on each node in `pNodes` until a clash is detected or no more rules are applicable.

170
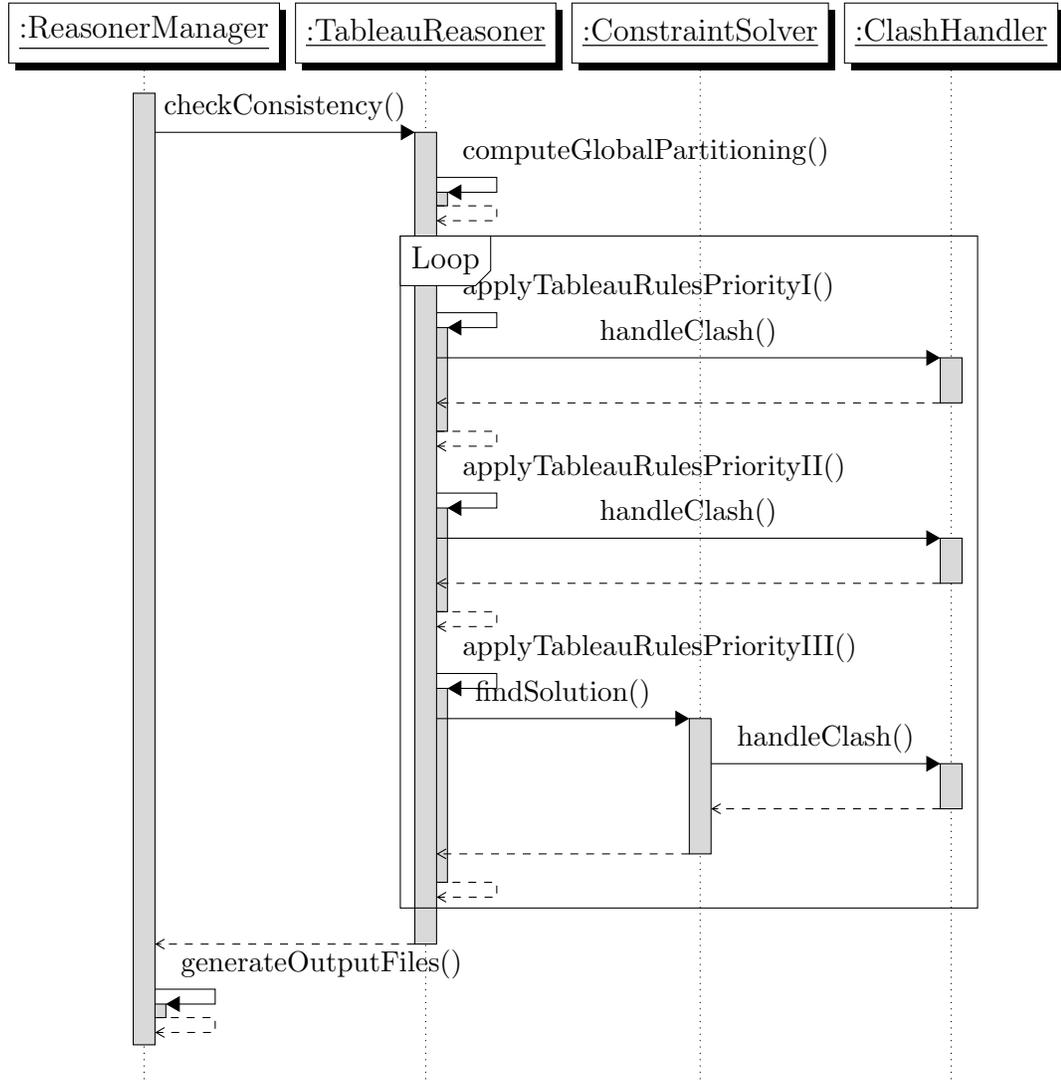
Figure 42: General sequence diagram for the HARD Tableau Reasoner performing a consistency test.

Figure 42 shows the main steps handled by the Tableau Reasoner during a KB consistency check. At first, a `CCGraph` G is initialized with only one node (`rNode`) in `pNodes`. `rNode` represents the node r_0; the `owlInd` is set to `null` and the cardinality value is set to zero because r_0 does not represent any domain element. In order to initialize the node's in-equation label with the encoding of the nominals semantics, the global partitioning is computed by calling the `computeGlobalPartitioning()` procedure.

EuropeanCountry	$\sqsubseteq\ \geq 1$locatedIn.Continent $\sqcap \forall$locatedIn.$\{$Europe$\}$
Continent	$\equiv \{$Asia, Europe$\}$

Figure 43: TBox axioms representing a European country concept description.

`computeGlobalPartitioning()` This procedure is responsible for computing global partitioning. It starts by combining the elements of the sets of $N_{R'}$ and N_o collected at preprocessing into $\mathcal{DS}$ represented as an array (`DecSetArray` consisting of `OWLObject`s). For example, consider the decomposition set corresponding to the TBox shown in Figure 43, $\mathcal{DS} = \{$locatedIn, Asia, Europe$\}$[13] such that we have the following sets of nominals $N_o = \{$Europe, Asia$\}$, and role names $N_R = \{$locatedIn$\}$, the array representing $\mathcal{DS}$ is as follows: `DecSetArray = [locatedIn, Asia, Europe]` which means that:

$$\text{DecSetArray[0]} = \text{locatedIn}$$
$$\text{DecSetArray[1]} = \text{Asia}$$
$$\text{DecSetArray[2]} = \text{Europe}$$

A hashMap, `DecSetIndexMapping`, keeps a mapping between `OWLObject`s and their corresponding array index in `DecSetArray`. Every entry $<$(`OWLObject, Integer`)$>$ in `DecSetIndexMapping` corresponds to an elements of $\mathcal{DS}$ represented by `OWLObject`, and its corresponding array index represented by an `Integer`. In the case of $\mathcal{DS} = \{$locatedIn, Asia, Europe$\}$, the mapping is represented as follows:

$$\text{DecSetIndexMapping} = \{(\text{locatedIn}, 0), (\text{Asia}, 1), (\text{Europe}, 2)\}$$

The use of array indexes and index mapping allows a direct access to an `OWLObject`. It is also used to compute the partitioning based on a binary representation of the

---

[13]For clarity locatedIn is used to refer to R' such that $R' \sqsubseteq$ locatedIn and R' is the role name introduced after preprocessing ≥ 1locatedIn.Continent into $\geq 1R' \sqcap \forall R'$.Continent.

array elements. Each binary number refers to a certain partition name p such that the zero digits represent the nominals or role names not included in p, whereas the 1 digits represent the nominals or role names included in p. For example, the binary number `a = 001` consists of 3 digits each representing an array index in `DecSetArray`, the first digit from right to left is ''1'' and it corresponds to array index 0, the second digit is ''0'' and it corresponds to array index 1, the third digit is ''0'' and it corresponds to array index 2. `a` refers to the partition name $p_a = \{\mathsf{locatedIn}\}$ because the "1" digit represents `DecSetArray[0]= locatedIn`. Similarly, the binary number `b = 011` refers to the partition name $p_b = \{\mathsf{locatedIn}, \mathsf{Asia}\}$.

Variable
partitionIndex
value
$\ldots$
getPartitionObjects()

The integer values for these binary representations are used as indexing for the variables used to represent each partition name. For example, the variable index for p_a is equal to $(0 \times 2^2 + 0 \times 2^1 + 1 \times 2^o) = 1$ and that for p_b is equal to $(0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0) = 2 + 1 = 3$.

For every element d in $\mathcal{DS}$ the set of all possible partitions including d is computed. In the case of $\mathcal{DS} = \{\mathsf{locatedIn}, \mathsf{Asia}, \mathsf{Europe}\}$ the partitions names computed for each `OWLObject` are shown in Table 5. A `hasMap` (`PartitionsIndexMapping`) consisting of <`OWLObject`, `Set`<`Integer`>> is used to keep a mapping between each `OWLObject` and the set of variable indexes referring to the corresponding partitions names including this object. The partition mapping for $\mathsf{locatedIn}$, Asia, and Europe is as shown below:

`PartitionsIndexMapping` $=\{(\mathsf{locatedIn}, \{1, 3, 5, 7\}), (\mathsf{Asia}, \{2, 3, 6, 7\}), (\mathsf{Europe}, \{4, 5, 6, 7\})\}$

OWLObject	Binary representation of partition names	Variable indexes
locatedIn	001, 011, 101, 111	1, 3, 5, 7
Asia	010, 011, 110, 111	2, 3, 6, 7
Europe	100, 101, 110, 111	4, 5, 6, 7

Table 5: Binary representation and corresponding variable indexes for the decomposition of $\mathcal{DS} = \{\text{locatedIn}, \text{Asia}, \text{Europe}\}$ as represented in `DecSetArray = [locatedIn, Asia, Europe]`.

It is at this stage that the partition elimination techniques are invoked. A set of *noGood* variables and *illegal* variables is maintained. For example, variables with index 7 and 6 are *noGood* variables because they correspond to partition names including the names for two nominals that are disjoint (Asia and Europe). When the *ch*-Rule is applicable to `rNode`, the set of variables (indexes) for a nominal are easily fetched from the `partionIndexMapping`, the set difference is computed between this set and the set of *noGood/illegal* variables. The branching is only done on variables that are neither *noGood* nor *illegal* (e.g. v_2 and v_3 in the case of Asia).

The *ch*-Rule is applied to `rNode` such that for each nominal, every variable index is assigned a choice point (e.g. $v_2 \geq 1$, $v_3 \leq 0$) as illustrated in Figure 44.
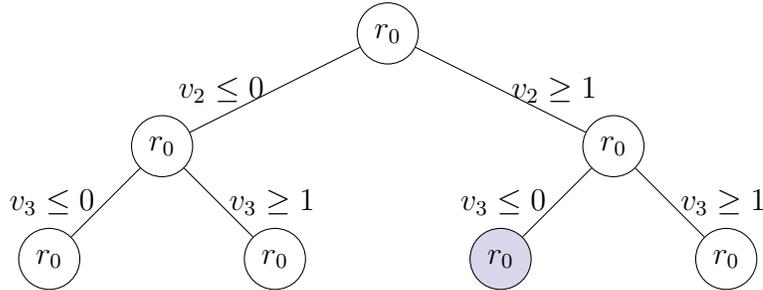
Figure 44: Expansion of the search space due to the applicability of the *ch*-Rule on the variables for the nominal Asia. The left branch corresponds to the choice point for $v_i \leq 0$, and the right branch corresponds to the choice point for $v_i \geq 1$ with $i = 2$, and 3. The *ch*-Rule is not applied to v_6 and v_7 because those are *noGood* variables.

The *ch*-Rule selection, together with the encoding of the nominals semantics form a set of in-equations. The encoding of the nominals semantics into in-equations, given the highlighted branching point in Figure 44, is formed as follows:

Nominal	Encoded in-equation
Asia	$v_2 + v_3 = 1$
	$v_2 \geq 1$
Europe	$v_4 + v_5 = 1$

If a solution is returned, then the value of each corresponding variable is set to the one assigned by the Constraint Solver. Apply the *fil*-Rule and initialize nominal nodes based on the solution returned by Simplex.

applyTableauRulesPriorityI() This procedure is responsible for checking if a rule with Priority 1 is applicable to any node in the CCGraph. These rules correspond to the implementations of the following rules: $\sqcap$-Rule, $\forall$-Rule, $\forall_+$-Rule, $\bowtie$-Rule, *e*-Rule, $\sqcup$-Rule, and *ch*-Rule, and they are consulted in the order listed. Except for the *ch*-rule, these rules are applicable to all nodes but `rNode`.

applyTableauRulesPriorityII() This procedure is responsible for checking if a rule with Priority 2 is applicable to any node in the CCGraph. The *fil*-Rule is implemented in such a way that it is only applicable to an `rNode` object, where a collection of `inequationLabel` objects is maintained. Once applicable, the rule collects all inequation labels from the nodes in G not previously added to its own label, and transforms the `inequationLabel` into constraints passed to the proper Constraint Solver for a solution. If no solution is found, then the graph G is marked as clashed and the arithmetic clash handler is fired. Otherwise, the variable values are updated with the returned solution and the corresponding nodes are created and added to `pNodes`.

applyTableauRulesPriorityIII() This procedure is responsible for checking if a rule with Priority 3 is applicable to any node in the CCGraph. Only one rule has Priority 3 and it corresponds to the implementation of the $\forall_\backslash$-Rule. The following section classifies the expansion rules based on how they extend the `CCGraph` objects, and discusses the rule application strategy.

6.5.1 Rule Application Strategy

HARD implements a rule application strategy which makes sure that rules are applied based on their priorities. In practice, the order in which rules within the same priority are applied affects the performance of the reasoner. Before discussing the strategy adopted by HARD in applying an order of which rules within the same priority are fired, a distinction is made between *deterministic*, *non-deterministic*, *generating* and *non-generating* rules.

Deterministic Rules A deterministic rule extends either the labels ($\mathcal{L}_E(x)$ referred as `inequationLabel` or $\mathcal{L}(x)$ referred as `nodeConceptLabel`) of a certain node (`iNode` element of `pNodes`) or the label (`edgeLabel`) of a certain edge (`iEdge` element of `pEdges`), or the set of nodes (`pNodes`) by introducing a new node. For example, among the tableau rules described in Figures 23 and 24 of Section 4.5.2, we identify the $\sqcap$-Rule, $\forall$-Rule, $\forall_+$-Rule, $\forall_\backslash$-Rule, $\bowtie$-Rule, e-Rule, and the fil-Rule as deterministic rules. The implementation of these rules is straightforward and can be easily translated from their descriptions.

Non-Deterministic Rules A non-deterministic rule extends the label ($\mathcal{L}_E(x)$ referred as `inequationLabel` or $\mathcal{L}(x)$ referred as `nodeConceptLabel`) of a node (`iNode` element of `pNodes`) while also expanding the search space (`CCGraphTree`), which consists of a tree of `CCGraph` objects. For example, among the tableau rules described in

Section 4.5.2, the ⊔-Rule and the *ch*-Rule are non-deterministic rules. For a demonstration on how these rules expand the label of a node while also expanding the search tree see Figures 34, 35 and 39.

Generating Rules   A generating rule introduces new nodes to the CCG. The only generating rule is the *fil*-Rule which extends the set of nodes in `CCGraph` with new nodes based on the solutions returned by the Constraint Solver. Every time a node is created, the list of nominals is added to the concept label description, as well as the negation of the nominals not included in the list.

Non-Generating Rules   A non-generating rule does not extend the set of nodes within the completion graph. This means that *non-generating* rules are rules which are not *generating*. *Deterministic* and *Non-deterministic* rules can also be considered non-generating.

Both *deterministic* and *non-deterministic* rules extend the labels of a certain node. *Deterministic* rules can be applied without expanding the search space unlike *non-deterministic* rules which additionally expand the search space with choice points for every extension of a node's label. This means that a bookkeeping of choice points is required along with a bookkeeping of dependencies.

HARD implements a rule application strategy which facilitates early clash detection with a minimal search space. This is done by enforcing that *deterministic* rules are applied before *non-deterministic* ones in order for clashes to be detected before a further expansion of the search space. Also, *generating* rules are applicable before *non-generating* rules, which enforces a breadth-first order of application of rules within a completion graph.

6.6 Constraint Solver

The Constraint Solver is responsible for solving the system of linear in-equations, accumulated due to the applicability of the $\bowtie$-Rule, using *Integer Programming* (IP) 22. Recall from Definition 4.5.3 that an IP model consists of an objective function that needs to be optimized subject to a set of linear constraints on that function, and is considered a special type of Linear Programming (LP) problems with additionally constraining the values of all variables to integer values. Integer Programming (IP) problems can be solved using the widely known Simplex [CLRS01] method for LP, extended with the branch and bound technique to solve the integer constraints.

HARD relies on two different implementations of the Simplex method; one is hard coded and accessed directly in the implementation of HARD through a Simplex module, and one is accessed through an external Constraint Solver; the non-commercial LPSolver.

Simplex module The Simplex module is based on the implementation in [CLRS01] which is also used in [Far08] and is responsible for finding a non-negative integer solution using the branch-and-bound method. Unlike the implementation in [Far08], our Simplex module does not implement the *ch*-Rule directly and does not implement any of the optimizations discussed to enforce a certain ordering on variables. Also the Simplex module returns either only one solution or no solution.

LPSolver The LPSolver is invoked through its own API. It is invoked by the *fil*-Rule which is also responsible for expanding the completion graph based on the solution returned by the LPSolver. The IP model is passed using a file containing an lp-model in lp-format. The lp-format is the LPSolver native format to read and write IP models; its input syntax consists of a set of algebraïc expressions and integer declarations in the following order:

$$<objective\ function>$$

$$<constraint> *$$

$$<declaration> *$$

The $<objective\ function>$ is a linear combination of optional variables ending with a semicolon, preceded by "min':' to indicate that the objective function is to be minimized. The objective function is required, but can be empty. As discussed in Section 6.2, and depending on the Constraint Solver preferences selected by the user, the set of variables used in this function could refer to the variables that have already been assigned a value, those occurring in an at-most restriction only, or all variables used.

The $<constraint> *$ is an optional constraint name followed by a colon, plus a linear combination of variables and constants followed by a relational operator, followed again by a linear combination of variables and constants, ending with a semicolon. The relational operator can be any of the following: "<=" "=" or ">=". Two types of constraints are modelled, $<Nominals\ constraints>$ and $<QCRs\ constraints>$. The $<Nominals\ constraints>$ represent the encoding of the nominals semantics into linear constraints. The $<QCRs\ constraints>$ represent the encoding of the QCRs into linear constraints. Another set of constraints is also passed, the $<assigned\ variables\ constraints>$, which consists of passing previous solutions for variables as constraints.

The $<declaration> *$ is used to define integer variables. Using the following syntax: "int" var [","] var [","] var ... ";"

179

```
/* Objective function */
min: +v₂ + v₃ + v₄ + v₅;
/* Nominals Constraints */
Asia : v₂ + v₃ = 1;
Europe : v₄ + v₅ = 1;
/* Inequations Constraints */
locatedIn : v₁ + v₃ + v₅ ≥ 1;
/* Assigned Variables */
v₂ = 1;
v₅ = 1;
/* Variable bounds */
int v₂, v₃, v₄, v₅;
```

Figure 45: Example of an IP model in lp-format representing the encoding of nominals and QCRs constraints represented in the definition of a EuropeanCountry.

Figure 45 shows the lp-format representing the encoding of nominals and QCRs constraints represented in the definition of a EuropeanCountry as shown in Figure 43. The assigned variables correspond to an initial nominal distribution over the following partitions $p_a = \{\text{Asia}\}$, and is represented by $v_2 = 1$, and $p_b = \{\text{Europe}, \text{locatedIn}\}$, and is represented by $v_5 = 1$

The result of solving the IP model is returned back to the Reasoner Manager which either calls the Clash Handler, in case no solution was found, or returns the solution back to the Tableau Reasoner for modelling.

6.7 Clash Handler

There are three types of clashes that can occur during a satisfiability test: (1) the logical clash, (2) the arithmetic clash, and the (3) OR clash. Once a clash is detected, the Clash Handler makes sure that the appropriate handler is fired. The following three clash handlers are implemented:

Algorithm 6.7.1 Pseudo-code for the Logical Clash Handler.

Algorithm 6.7.1: LOGICAL CLASH HANDLER(C)

$k_1 \leftarrow$ GETALTERNATIVECHOICEPOINT(C)

$k_2 \leftarrow$ GETALTERNATIVECHOICEPOINT($ObjectComplementOf(C)$)

if ($k_1 = 0$ **and** $k_2 = 0$)

then $\begin{cases} cIndex \leftarrow \text{GETCLASHEDNODEINDEX}() \\ noGoods.\text{ADD}(cIndex) \\ nextDDBCraph \leftarrow \text{GETGRAPHINDEX}(cIndex \geq 1) \end{cases}$

else $\begin{cases} \textbf{if } (k_1 \geq k_2) \\ \quad \textbf{then } nextDDBGraph \leftarrow k_2 \\ \quad \textbf{else } nextDDBGraph \leftarrow k_1 \end{cases}$

Logical clash handler. The logical clash handler detects clashes due to the occurrence of `C` and `ObjectComplementOf (C)` within the label of a node. When such a clash is detected, Algorithm 6.7.1 is fired to set the corresponding backtracking point if one exists. The pseudo-code for the `GetAlternativeChoicePoint` is presented in Algorithm 6.7.2.

Algorithm 6.7.2 Pseudo-code for the `GetAlternativeChoicePoint` procedure.

Algorithm 6.7.2: GETALTERNATIVECHOICEPOINT(C)

if (C instanceOf OWLObjectUnionOf)

then $\begin{cases} \textbf{comment: Get the graph where the } \sqcup\text{-Rule was applied on } C \\[4pt] pGraph \leftarrow \text{GETPARENTGRAPH}(C) \\[4pt] \textbf{comment: Get the list of branching graphs for } C \\[4pt] orGraphBranches \leftarrow \text{GETORBRANCHES}(pGraph, C) \\[4pt] \textbf{comment: Iterate through the branches to find an alternative} \\[4pt] \textbf{for each } childGraph \in orGraphBranches \\[4pt] \quad \textbf{do if } (childGraph.\text{ISCLASHFREE}()) \\[4pt] \textbf{return } (childGraph.Index) \end{cases}$

else if (C instanceOf OWLObjectALLRestriction)

then $\begin{cases} cFiller \leftarrow C.\text{GETFILLER}() \\[4pt] \textbf{return } (\text{GETALTERNATIVECHOICEPOINT}(cFiller)) \end{cases}$

Algorithm 6.7.3 Pseudo-code for the OR Clash Handler.

Algorithm 6.7.3: OR CLASH HANDLER($D = (C_1 \sqcup C_2 \ldots \sqcup C_n)$)

$k \leftarrow \text{GETALTERNATIVECHOICEPOINT}(D)$

if ($k = 0$)

then $\begin{cases} cIndex \leftarrow \text{GETCLASHEDNODEINDEX}() \\[4pt] noGoods.\text{ADD}(cIndex) \\[4pt] nextDDBCraph \leftarrow \text{GETGRAPHINDEX}(cIndex \geq 1) \end{cases}$

else $nextDDBGraph \leftarrow k$

OR-Rule clash handler. The OR-Rule clash handler detects and handles the clashes due to the applicability of the $\sqcup$-Rule to a node x where all the disjuncts in $(C_1 \sqcup C_2 \cdots \sqcup C_n)$ are skipped by the *look ahead optimizations* because they will lead to clashes. When this clash is detected, the node is set to clashed and Algorithm 6.7.3 is fired to set the corresponding backtracking point if one exists.

Algorithm 6.7.4 Pseudo-code for the Arithmetic Clash Handler.

Algorithm 6.7.4: INFEASIBLEQCRHANDLER(QCR)

$NextDDBGraph \leftarrow$ TREATINFEASIBLEQCR(QCR)

if $(NextDDBGraph == 0)$

then $\begin{cases} cSource \leftarrow \text{GETSROUCE}(QCR) \\ NextDDBGraph \leftarrow \text{GETALTERNATIVECHOICEPOINT}(cSource) \\ \textbf{if } (NextDDBGraph == 0) \\ \quad \textbf{then} \begin{cases} cIndex \leftarrow \text{GETCLASHEDNODEINDEX}() \\ noGoods.\text{ADD}(cIndex) \\ nextDDBCraph \leftarrow \text{GETGRAPHINDEX}(cIndex \geq 1) \end{cases} \end{cases}$

Arithmetic clash handler. The arithmetic clash handler detects and handles obvious arithmetic clashes as well as arithmetic clashes detected because the Constraint Solver returned no solution for the IP model. If the QCRs responsible for the clash are identified, then Algorithm 6.7.4 is fired to set the corresponding backtracking point if one exists. Otherwise, the node is set as clashed and standard backtracking takes care of exploring alternative choice points.

A statistics module keeps track of the different types of clashes encountered. A file named `STAT-"OWLFileName".txt` is generated at the end of a consistency check test containing all the information gathered by the different clash handlers including

clashes counts and types.

Algorithm 6.7.5 Pseudo-code for the `TreatInfeasibleQCR` procedure.

Algorithm 6.7.5: TREATINFEASIBLEQCR($QCR, rVariables$)

if (QCR instanceOf atLeastRestriction)

then $\begin{cases} \textbf{comment:} \text{find an active variable set to zero} \\[2mm] vIndex \leftarrow \text{GETBACKUMPINGVARIABLE}(rVariables, 0) \\[2mm] \textbf{comment:} \text{backjump to where this variable is not zero} \\[2mm] nextDDBCraph \leftarrow \text{GETGRAPHINDEX}(vIndex \geq 1) \end{cases}$

else if ALLNOMINALVARIABLES($rVariables$)

then $\begin{cases} \textbf{comment:} \text{choose a variable to eliminate} \\[2mm] iIndex \leftarrow \text{GETBACKUMPINGVARIABLE}(rVariables, 1) \\[2mm] \textbf{comment:} \text{identify the nominal involved} \\[2mm] iNominal \leftarrow iIndex.\text{GETNOMINALFROMPARTITION}(iIndex) \\[2mm] iRole \leftarrow QCR.\text{GETOWLOBJECTPROPERTYEXPRESSION}() \\[2mm] \textbf{comment:} \text{choose a nominal variable not intersection with iRole} \\[2mm] vIndex \leftarrow \text{GETNOMINALVARWITHOUTROLE}(iRole, iNominal) \\[2mm] \textbf{comment:} \text{backjump to where this variable is not zero} \\[2mm] nextDDBCraph \leftarrow \text{GETGRAPHINDEX}(vIndex \geq 1) \end{cases}$

6.8 Concluding Remarks

In this chapter, the main components that make up HARD, the prototype reasoner implementing ReAl DL, were presented. During the implementation of HARD the

extreme programming[14] technique was adopted; this means that the implementation was started from scratch with a basic sound and complete implementation handling at least the basic DL with nominals and QCRs ($\mathcal{ALCOQ}$). The implementation was repetitively extended along with regression testing until when this thesis was written and it handles $\mathcal{SHOQ}$ except for the handling of *qualifying concepts*. Note that *qualifying concepts* do not affect the complexity of the Tableau Reasoner implemented by HARD, they only need to be enabled in the presence of GCIs where concept descriptions contain *qualifying concepts*. Once enabled, the handling of *qualifying concepts* affects the bookkeeping phase during preprocessing in order to compute N_{Q}. The size of N_{Q} extends the size of $\mathcal{DS}$ as well as the size of the global partitioning $\mathcal{P}$. Also, the conditions of applicability of the $\bowtie$-Rule and the fil-Rule must be adapted. Therefore, they do not affect the purpose and evaluation of the algebraïc method because the effect of an increasing decomposition set can be evaluated using an increasing set of nominals or QCRs. The handling of *qualifying concepts* is missing strictly due to time constraints; the validation of possible interactions with implemented optimizations could not be estimated to be completed within the time limit.

The algebraïc approach in dealing with QCRs discussed in [HTM01] is part of RacerPro's reasoning algorithm. When a partitioning is computed for a certain role, RacerPro's satisfiability test is fired for each partition to check if that partition is satisfiable or not. This means that with an exponential number of partitions, an exponential number of recursive satisfiability checks is invoked which makes the implementation best case exponential.

The hybrid algebraïc reasoning algorithm for $\mathcal{SHQ}$ presented in [Far08] extends the one in [FFHM08b] with role hierarchies and transitive roles. It was evaluated using an optimized prototype implementation which is based on a logical module

---

[14]http://www.extremeprogramming.org/

working together with an arithmetic module implementing Simplex. As the name suggests, the logical module is responsible for applying tableau rules while the arithmetic module is responsible for solving the constraints generated by the QCRs using Simplex methods added with branch-and-bound. A major divergence to the calculus is the implementation of the *ch*-Rule directly into the arithmetic module which returns not only one solution but all possible solutions due to the different choices assigned by the *ch*-Rule. In a sense, the logical module does not have any information about the variables during execution of the tableau rules. This limits the effectiveness of implemented backtracking methods because the dependencies for variables cannot be recorded. When a reference is made to the Simplex module implemented within HARD, it refers to the version from [Far08] without the implementation of *ch*-rule and the different heuristics for variables like the don't care variable assignment.

Unlike the implementation presented in [Far08] and handling the DL $\mathcal{ALCHQ}$, where a local repetitive decomposition of role fillers is adopted and solutions returned by the Constraint Solver are not kept from one node to another, the implementation of HARD is based on a global decomposition set, and a global assignment of the solutions to the in-equations carried from one node to another. On the other hand, the use of indexing of variables to refer to their corresponding partitions is based on the idea from [Far08] with the use of a different data structure.

6.8.1 Limitations

The following problems were encountered during the implementation of HARD:

- The prototype reasoner uses a parser that can only handle ontologies in the RDF/XML format.

- The implementation of the Simplex procedure from [Far08] contains some bugs. There are test cases where the Constraint Solver implemented could not find a

solution whereas if the LPSolver was selected a solution is found. Some of these test cases are listed in Section B.2.

- A `java.lang.NumberFormatException` is encountered if the size of $\mathcal{DS}$ is ≥ 63. This is a JAVA limitation on the size of an array which can be at most equal to `Integer.MAX_VALUE` which is equal to $2^{63} = 2147483647$. This puts a limitation on the size of $\mathcal{DS}$ that can be handled to 63. This problem can be overcome using a different data structure such that the limitation is BigInteger.MAX_VALUE. However, the incremental decomposition and the use of the lazy decomposition techniques avoid running into this problem in most test cases used.

- Java randomly resets `static` counters used to count the number of `CCGraph` created when applying tableau rules.

- Java randomly frees up some memory and some `CCGraph` are disposed from the hashtree, which causes a wrong result in case when the procedure requires back-jumping to a `CCGraph` which has been already disposed.

These cases were very rare and were encountered during the process of debugging the implementation but not encountered during the performance evaluation phase. Some of the errors encountered were the motivation behind integrating the LPSolver as an API into HARD. The additional benefit of such an integration is that HARD can be easily modified to use an off-the-shelf highly optimized Constraint Solver. Finally, integrating available reasoner's APIs allows a more consistent way to evaluate the reasoning performance. An evaluation of the performance of a naïve implementation, as well as the optimizations required in order to have a practical reasoner are discussed in the next chapter.

Chapter 7

Performance Evaluation

This chapter aims at evaluating the practical aspect of ReAl DL (Reasoning Algebraïcally for Description Logics). The evaluation of the approach is twofold: first, the practical performance of HARD, the prototype reasoner described in Chapter 6, and which implements the hybrid algebraïc tableau reasoning algorithm proposed in this thesis, is compared against existing SOTA (state of the art) reasoners for which available native libraries are integrated into HARD's application. Second, the effectiveness of the optimization techniques proposed in Chapter 5 is evaluated by comparing runs of test cases where one or more optimization(s) are enabled by the user preferences.

This chapter is structured as follows: Section 7.1 describes the methodology adopted to evaluate HARD's performance. Section 7.2 describes the tests suites and cases used and the reported run-times. Section 7.3 highlights key optimizations as well as their effect on the performance of HARD. Section 7.4 gives a general analysis of HARD's performance, limitations and problems. Section 7.5 concludes the chapter.

7.1 Evaluation Methodology

The implementation of ReAl DL into the prototype reasoner (HARD) has two main objectives:

1. To show that the optimized ReAl DL is efficient. In a sense, show that HARD can solve problems with *nominals* and QCRs better than existing reasoners which do not implement algebraïc reasoning.

2. To show that the optimization techniques proposed in Chapter 5 are effective. These optimizations bring a dramatic speed-up to (typical case) satisfiability tests. Without these optimizations HARD cannot be considered practical.

These objectives can be met by evaluating HARD through TBox consistency tests using real world ontologies which at least include a pattern of the problem being tackled. This means that the ontologies used must be based on a DL expressivity that is at least the basic DL $\mathcal{ALC}$ extended with *nominals* ($\mathcal{O}$) and QCRs ($\mathcal{Q}$). This is because the algebraïc reasoning algorithm was proposed to address inefficient reasoning with QCRs and *nominals*. Also, since the algebraïc method has been adopted previously in [HTM01, FH10c] to handle QCRs, HARD is also tested against test cases relying on the use of QCRs without necessarily using *nominals*. By doing this, one can verify if previous results in handling QCRs algebraïcally still hold, and if the complexity of handling *nominals* using a global decomposition affects the reasoning even when the test cases do not include *nominals*.

The scalability of HARD is tested by using ontologies of different complexity. Note however that ontologies of generally large size (hundreds of concepts) are not considered, this is because although the size greatly affects the complexity of reasoning, the prototype reasoner is not intended to tackle such a complexity. Hence, the test cases are limited to small ontologies including complex patterns that challenge reasoning

about nominals and QCRs by existing tableau-based reasoning methods.

7.1.1 Choosing Benchmarks

The benchmarks used include existing prototype ontologies, such as the ones used in [FH10c] and [HSV11], to evaluate reasoning with QCRs, newly designed prototype ontologies (hand crafted) and an adaptation or subtraction of existing real world ontologies to evaluate reasoning with QCRs and *nominals*. In fact, most existing real-world ontologies, if not adapted, cannot serve as benchmarks for HARD because of the following reasons:

- Most real world ontologies are based on a DL expressivity not including QCRs; ontology designers have been avoiding the use of this constructor even if comes very natural in many domains for the following reasons:

  1. QCRs were recently added to the new version of OWL, OWL 2 [MPSCG08],

  2. QCRs are not very well handled with most existing reasoning approaches especially when the numbers used in QCRs are high. It has been reported many times that existing reasoners cannot handle QCRs without the use of algebraïc methods, e.g., this is the case with tableau-based DL reasoning [Hor02], resolution-based DL reasoning [KM06] and hypertableau-based DL reasoning [MSH09].

Some ontologies compensate the use of QCRs with the use of *concrete datatypes* (e.g., the Semantic Web for Earth and Environmental Terminology known as the SWEET[1] ontology), which are introduced in Section 2.1.2.2. Also most bioinformatic ontologies (SNOMED, GALEN, etc ...) rely on the expressivity of

---

[1] http://sweet.jpl.nasa.gov/

the DL $\mathcal{EL}$ which allows conjunction ($\sqcap$), existential restrictions on concepts ($\exists$) and neither QCRs nor *nominals*.

- Most real world ontologies are based on a DL expressivity which allows the use of constructors not supported by HARD such as *inverse roles* ($\mathcal{I}$) and *concrete datatypes* (see Section 2.1.2.2 for an introduction of these constructors).

- Most real world ontologies contain a large number of GCIs or domain elements. HARD does not implement optimizations (*absorptions*, etc...) required for efficient reasoning with large ontologies.

Adapting a real world ontology to serve as a benchmark for HARD therefore involves extracting a subset ontology, eliminating the use of operators that are not supported by HARD, and enforcing the use of QCRs where suitable.[2]

The scalability of HARD is tested using ontologies of various characteristics such that these characteristics affect the complexity of DL reasoning in general or the complexity of ReAl DL implemented by HARD. In general, it is the *expressivity* and the *size* of the ontology that affect reasoning with that ontology; concept satisfiability for the DL $\mathcal{ALCHOQ}$ is PSpace-complete but becomes ExpTime-complete if transitive roles are used or if GCIs are enabled in the TBox.[3] The size of an ontology is measured using the TBox size, the ABox size, the number of GCIs used, or the size of the generated model. These are characteristics which affect DL reasoning in general without reference to the implemented reasoning procedure. On the other hand, the complexity analysis in Chapter 4 for ReAL DL has shown that the size of the *global decomposition set* ($\mathcal{DS}$) highly affects the complexity of such a reasoning procedure. The size of $\mathcal{DS}$ is measured using the *number of nominals* used (size of N_o), *number of* QCRs (size of $N_{R'}$), and the *number of qualifying concepts* (size of Q_N). The size

---

[2]For example, the use of datatypes could often be replaced by the use of QCRs.

[3]See the description logics complexity navigator at `http://www.cs.man.ac.uk/~ezolin/dl/` for more details on how each DL constructor affects the complexity of reasoning.

of N_o affects the size of the TBox and the size of the ABox, the size of $N_{R'}$ affects the size of the TBox and the size of the generated model, and the size of Q_N affects the size of the TBox. Therefore, when considering ontologies of different TBox sizes one could focus on the size of $N_o + N_{R'}$. There is no need to consider the number of *qualifying concepts* because these are not handled by HARD. Also, when considering the size of an ABox one could focus on the size of N_o, and to consider the size of the generated model one could focus on either the number of QCRs (size of $N_{R'}$) or the size of the numbers used in those QCRs. As mentioned earlier, ontologies with large TBoxes or with a large number of GCIs are not interesting for HARD because HARD is not equipped with the necessary optimizations (absorption, unfolding, etc ...) to process such ontologies.

7.1.2 Comparing With SOTA Reasoners

To asses the performance of HARD, one needs to compare it against existing reasoning algorithms implemented by SOTA (state-of-the-art) reasoners. As described in the previous chapter, JAVA APIs for each of those reasoners have been integrated into HARD's application implementation and user preferences allow one to select one of Fact++, Hermit, Pellet, RacerPro, or HARD to perform the KB consistency test.

These systems implement different reasoning algorithms for DL. For example, Fact++ implements tableau-based DL reasoning whereas Hermit implements hyper-tableau reasoning. Each reasoner comes equipped with numerous optimizations some of which are specific to handle a certain complexity. For example, Hermit implements core blocking when dealing with ontologies with cyclic TBoxes of a large sizes and therefore, Hermit can classify ontologies that no other reasoner can classify. Also, a certain system might be slow for a specific test case due to various effects that can sometimes hardly be tracked down to the reasoning algorithm adopted. Therefore, it

might be hard to associate a reasoning performance degradation or speedup to the reasoning algorithm adopted.

7.1.3   Evaluating The Implemented Optimizations

To evaluate the effectiveness of the various optimizations proposed in Chapter 5 and implemented in HARD, test cases are evaluated where optimizations are turned ON or OFF. The speed-up factor is measured for key optimizations.

7.1.4   Evaluation Platform

The set of benchmarks used for evaluation consists of .owl files representing OWL ontologies, in the RDF/XML format, modelled using Protégé 4.1.[4] The tests are performed on a PC running Windows XP Media Centre Edition (version 2002 with Service Pack 3) with 2.93 GB of RAM and 2.40 GHZ AMD Athlon(tm) 64 Processor. The run-time of each test is reported in milliseconds as the average run-time of three to five separate executions of the test using the same reasoner. Each run-time represents the time needed for a selected reasoner to perform a KB consistency test.

7.2   Test Cases

This section describes the test cases that were used, and reports the run-times needed for HARD to decide a KB consistency compared to other reasoners. Since the reasoners are invoked for a KB consistency test on the loaded ontology, the satisfiability check of a certain concept C is tested by adding the following axiom $\top \sqsubseteq \neg\{a\} \sqcup C$ to the ontology, where $\{a\}$ is a freshly introduced nominal. The reported time is in milliseconds, and the TimeOut is set to 10 minutes (600000 ms). The first set of

---

[4]http://protege.stanford.edu/

benchmarks consists of test cases using the basic DL $\mathcal{ALC}$ extended with QCRs and *role hierarchies*. The second set of benchmarks consists of test cases using basic DL $\mathcal{ALC}$ extended with *nominals*, QCRs, *role hierarchies* and GCIs.

7.2.1 Test Cases for QCRs and Role Hierarchies

As mentioned earlier, we find it imperative to report on HARD's performance against the test cases that were used in [Far08] to evaluate the performance of algebraïc reasoning with the DL $\mathcal{ALCHQ}$. We first adapt these test cases to the DL $\mathcal{ALCQ}$ as done in [HSV11] and then we test them using the expressivity of $\mathcal{ALCHQ}$ as was done in [FH10c]. We choose to use the same test cases for two main reasons: first, we would like to show how the global partitioning needed in the presence of *nominals* affects performance. Second, we would like to show that even with global partitioning, the calculus is amenable to optimizations that allow a similar performance as with a local partitioning. In order to make this thesis self contained we repeat the descriptions of those test cases as we report on the performance. In some cases we consider more complex variants of the concept descriptions used. Each test case consists of a TBox consistency test where the TBox $\mathcal{T}$ includes the description of a concept C, for which we want to check the satisfiability, and the TBox axiom $\top \sqsubseteq \neg\{\mathsf{a}\} \sqcup \mathsf{C}$ where a is a freshly introduced *nominal*. If the test description uses roles such that the hierarchy between these roles is flat, we refer to the test case as $\mathsf{C}_{\mathcal{ALCQ}}$. Otherwise, if the hierarchy is not flat, we refer to the test case as $\mathsf{C}_{\mathcal{ALCHQ}}$. In the following we describe the test cases designed to test the algebraïc reasoning approach with qualified cardinality restrictions against the following parameters:

1. *The size of the numbers used in qualified cardinality restrictions,*

2. *The number of qualified cardinality restrictions used,*

194

3. *The ratio of the number of at-least restrictions to the number of at-most restrictions*, and

4. *The satisfiability versus the unsatisfiability of the given concept expression.*

7.2.1.1 Testing the size of the numbers used in QCRs

The effect of increased numbers used in QCRs is tested using the concept C defined using the DL $\mathcal{ALCQ}$ as follows:

$$C_{\mathcal{ALCQ}} \quad \sqsubseteq \quad \geq 2ir.(A \sqcup B) \sqcap \leq ir.A \sqcap \leq ir.B \sqcap (\leq (i-1)r.\neg A \sqcup \leq jr.\neg B)$$

Since we have $\top \sqsubseteq \neg\{a\} \sqcup C_{\mathcal{ALCQ}}$ this means that a is a member of $C_{\mathcal{ALCQ}}$, $(a : C_{\mathcal{ALCQ}})$, and $C_{\mathcal{ALCQ}}$ is satisfiable if a satisfies the following:

1. $a : (\geq 2ir.(A \sqcup B)) \Longrightarrow a$ must have at least $2i$ r-fillers such that each r-filler satisfies $(A \sqcup B)$, we refer to these r-fillers satisfying $(A \sqcup B)$ as r_1-fillers.

2. $a : (\leq ir.A) \Longrightarrow$ at most i r-fillers of a can be members of A. We refer to these r-fillers that are members of A as r_2-fillers and enforce that all other r-fillers $(r \setminus r_2$-fillers$)$ of a become members of $\neg A$.

3. $a : (\leq ir.B) \Longrightarrow$ at most i r-fillers of a can be members of B. We refer to those r-fillers that are members of B as r_3-fillers and enforce that all other r-fillers $(r \setminus r_3$-fillers$)$ become members of $\neg B$.

4. $a : (\leq (i-1)r.\neg A) \sqcup (\leq jr.\neg B)) \Longrightarrow$ there can be at most $(i-1)$ r-fillers of a that are members of $\neg A$, OR there can be at most j r-fillers of a that are members of $\neg B$. We refer to r-fillers that are members of $\neg A$ as r_4-fillers and those that are members of $\neg B$ as r_5-fillers.

$\mathcal{L}(\mathsf{a}) = \{\mathsf{C}_{\mathcal{ALCQ}}, \geq 2\mathsf{ir}_1, \forall \mathsf{r}_1.(\mathsf{A} \sqcup \mathsf{B}), \leq \mathsf{ir}_2, \forall \mathsf{r}_2.\mathsf{A}.\forall \mathsf{r} \setminus \mathsf{r}_2.\neg\mathsf{A}, \leq \mathsf{ir}_3, \forall \mathsf{r}_3.\mathsf{B}, \forall \mathsf{r} \setminus \mathsf{r}_3.\neg\mathsf{B},$

$((\leq (\mathsf{i}-1)\mathsf{r}_4 \sqcap \forall \mathsf{r}_4.\neg\mathsf{A} \sqcap \forall \mathsf{r} \setminus \mathsf{r}_4.\mathsf{A}) \sqcup (\leq \mathsf{jr}_5 \sqcap \forall \mathsf{r}_5.\neg\mathsf{B} \sqcap \forall \mathsf{r} \setminus \mathsf{r}_5.\mathsf{B}))\}$

a:1

⊔-Rule

$\mathcal{L}(\mathsf{a}) \cup \{(\leq \mathsf{jr}_5 \sqcap \forall \mathsf{r}_5.\neg\mathsf{B} \sqcap \forall \mathsf{r} \setminus \mathsf{r}_5.\mathsf{B})\}$ **a:1**   **a:1** $\mathcal{L}(\mathsf{a}) \cup \{(\leq (\mathsf{i}-1)\mathsf{r}_4 \sqcap \forall \mathsf{r}_4.\neg\mathsf{A}, \sqcap \forall \mathsf{r} \setminus \mathsf{r}_4.\mathsf{A})\}$

⊓-Rule   ⊓-Rule

$\mathcal{L}(\mathsf{a}) \cup \{(\leq \mathsf{jr}_5, \forall \mathsf{r}_5.\neg\mathsf{B}, \forall \mathsf{r} \setminus \mathsf{r}_5.\mathsf{B})\}$ **a:1**   **a:1** $\mathcal{L}(\mathsf{a}) \cup \{(\leq (\mathsf{i}-1)\mathsf{r}_4, \forall \mathsf{r}_4.\neg\mathsf{A}, \forall \mathsf{r} \setminus \mathsf{r}_4.\mathsf{A})\}$

⋈-Rule   ⋈-Rule

$\mathcal{L}_\mathsf{E}(\mathsf{a}) \cup \{\xi(\mathsf{r}_5, \leq, \mathsf{j})\}$ **a:1**   **a:1** $\mathcal{L}_\mathsf{E}(\mathsf{a}) \cup \{\xi(\mathsf{r}_4, \leq, (\mathsf{i}-1))\}$

ch-Rule   *ch*-Rule
fil-Rule   *fil*-Rule

Solution Found! **a:1**   **a:1** Arithmetic Clash!

$\{\mathsf{r}_1, \mathsf{r}_2, \mathsf{r}_5\}$ / *e*-Rule \ $\{\mathsf{r}_1, \mathsf{r}_3\}$

$\mathcal{L}_\mathsf{P}(\mathsf{b}) = \{\mathsf{r}1, \mathsf{r}2, \mathsf{r}5\}$ **b:i**   **c:i** $\mathcal{L}_\mathsf{P}(\mathsf{c}) = \{\mathsf{r}_1, \mathsf{r}_3\}$

∀-Rule   ∀-Rule

$\mathcal{L}(\mathsf{b}) = \{(\mathsf{A} \sqcup \mathsf{B}), \mathsf{A}, \neg\mathsf{B}\}$ **b:i**   **c:i** $\mathcal{L}(\mathsf{c}) = \{(\mathsf{A} \sqcup \mathsf{B}), \mathsf{B}\}$

$\forall_{\setminus}$-Rule

c:i $\mathcal{L}(\mathsf{c}) \cup \{\neg\mathsf{A}\}$
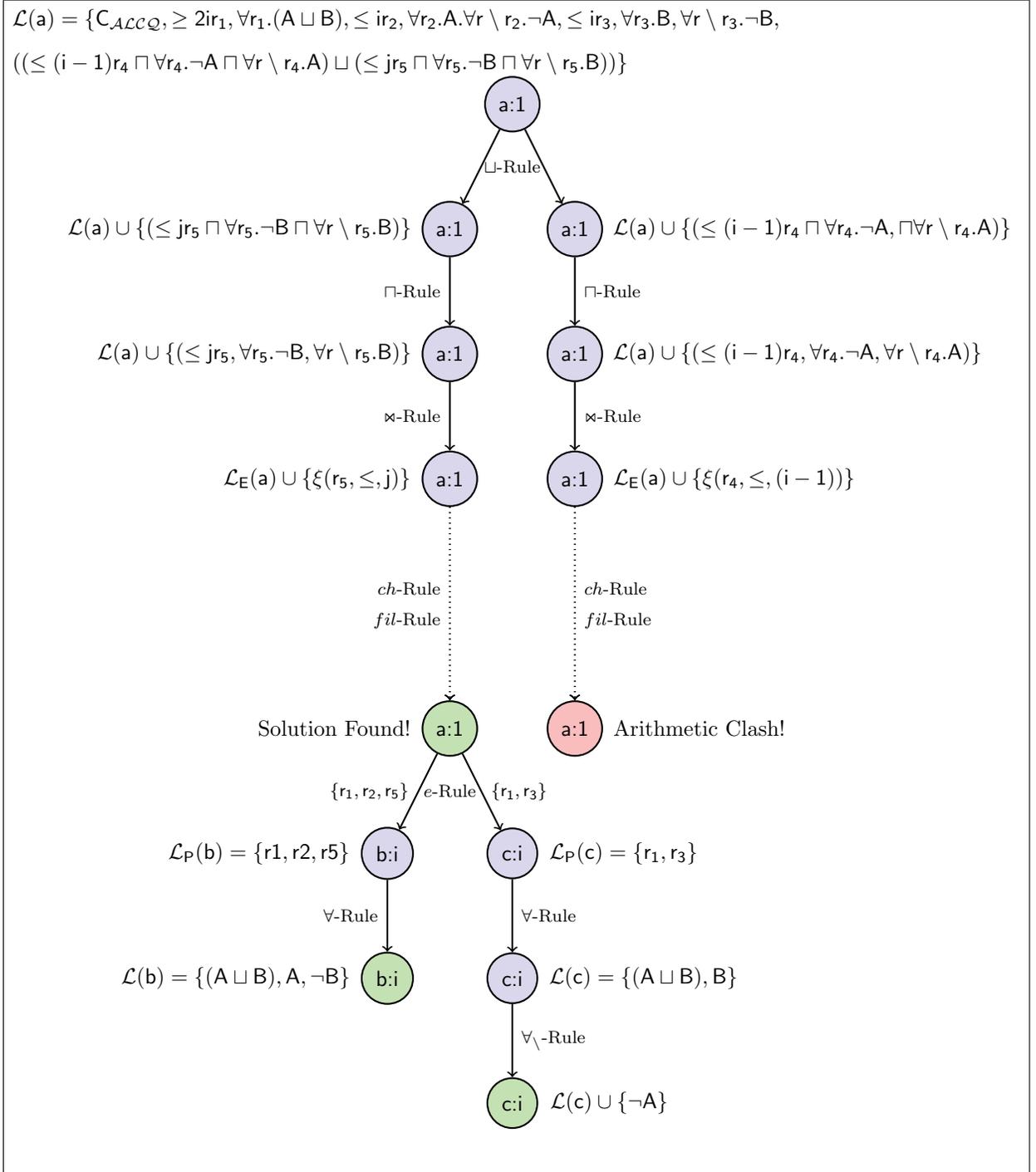
Figure 46: Expansion tree showing an expansion of the ⊔-Rule leading to a clash free CCG for $\mathsf{C}_{\mathcal{ALCQ}}$ with $\mathsf{j} = \mathsf{i}$, after applying the tableau rules described in Figures 23 and 24. Note that the QCRs in $\mathcal{L}(\mathsf{a})$ have already been preprocessed according to Algorithm 4.1.1, described in Section 4.1.1, such that $\geq 2\mathsf{ir}.(\mathsf{A} \sqcup \mathsf{B})$ is rewritten into $\geq 2\mathsf{ir}_1 \sqcap \forall \mathsf{r}_1.(\mathsf{A} \sqcup \mathsf{B})$. The nodes highlighted in green represent a completion model for $\mathsf{C}_{\mathcal{ALCQ}}$.

196

The domain element a satisfies (1), (2), and (3) for all values of $i > 0$, however, in order to satisfy (4) a must also satisfy $a : (\leq jr.\neg B)$ because $a : (\leq (i-1)r.\neg A)$ cannot be satisfied. Therefore, the satisfiability of C depends on the value of j; if $j \geq i$ then C becomes satisfiable because $a : (\leq jr.\neg B)$ is satisfiable. Otherwise if $j \leq (i-1)$ then C becomes unsatisfiable because $a : (\leq (i-1)r.\neg B)$ cannot be satisfied.

$C_{\mathsf{SAT}-\mathcal{ALCQ}}$ is used to refer to a satisfiable case of $C_{\mathcal{ALCQ}}$ where $j = i$, and $C_{\mathsf{UnSAT}-\mathcal{ALCQ}}$ to refer to an unsatisfiable case of $C_{\mathcal{ALCQ}}$ where $j = (i-1)$. In a first set of tests, $C_{*-\mathsf{lin}-\mathcal{ALCQ}}$, the numbers are increased linearly using i such that the values of i range between 1 and 10. In a second set of tests, $C_{*-\mathsf{exp}-\mathcal{ALCQ}}$, the numbers are increased exponentially using $i = 10^k$ and the values of k range between 1 and 6.

In the case of $C_{\mathsf{SAT}-\mathsf{lin}-\mathcal{ALCQ}}$, the KB is consistent because a model exists that can be represented by a clash-free *compressed completion graph* G. Figure 46 illustrates the expansion of the CCG G, where a domain element represented by the proxy node $(a : 1)$ is a member of the concept $C_{\mathcal{ALCQ}}$[5]. The node $(b : i)$ is a proxy node representing i domain elements that are r-fillers of a intersecting with r_1-fillers of a, r_2-fillers of a, r_5-fillers of a. Due to the qualifications imposed by the $\forall$-Rule, these role fillers are members of $(A \sqcap \neg B)$. The node $(c : i)$ is a proxy node representing i domain elements that are r-fillers of a intersecting with r_1-fillers of a and r_3-fillers of a. Due to the qualifications imposed by the $\forall$-Rule, and the $\forall_{\setminus}$, these role fillers are members of $(B \sqcap \neg A)$. The domain elements represented by b satisfy the restrictions imposed by (1), (2), (3) and (4) and the domain elements represented by c satisfy the restrictions imposed by (1), (2) and (3). Whereas in the case of $C_{\mathsf{UnSAT}-\mathsf{lin}-\mathcal{ALCQ}}$, $\mathcal{T}$ is inconsistent because $C_{\mathcal{ALCQ}}$ is unsatisfiable for $j = i - 1$ and there exists no model for $\mathcal{T}$. The effect of increasing the numbers used in QCRs on reasoning performance is shown respectively in Figures 47, 48 for satisfiable and unsatisfiable cases of $C_{\mathcal{ALCQ}}$.

---

[5]For clarity and ease of presentation, only the concept expressions relevant to a rule application are shown in the label of a node.
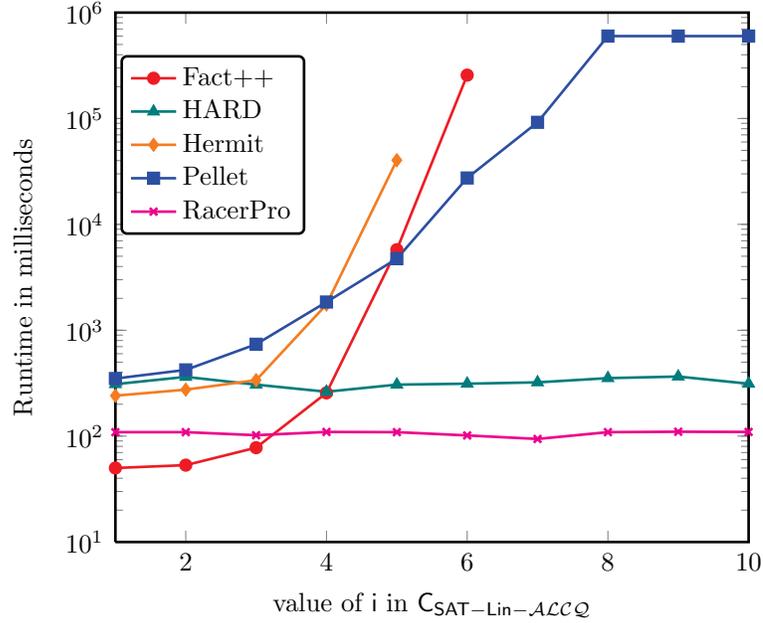
(a) Increasing i *linearly* in a *satisfiable* concept expression
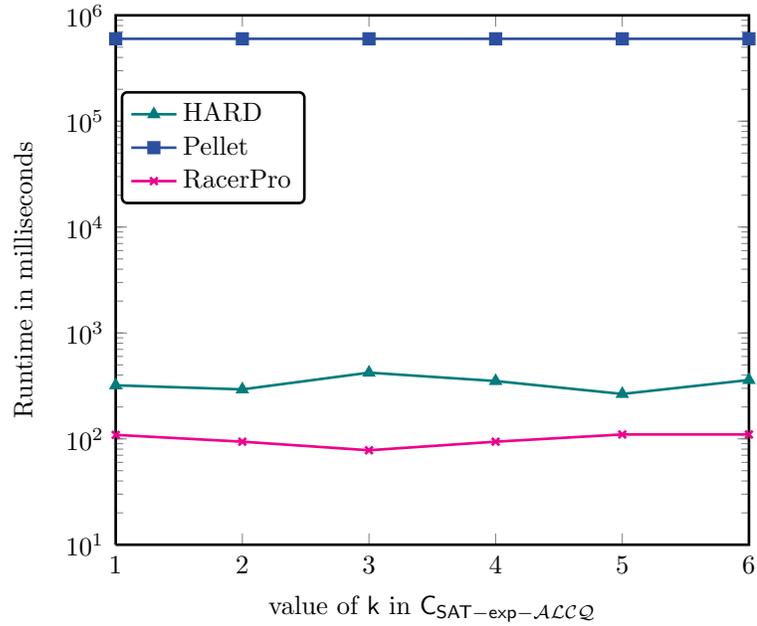
(b) increasing i *exponentially* in a *satisfiable* concept expression;

$i = 10^k, 1 \leq k \leq 6$

Figure 47: Effect of increasing the size of the numbers used in QCRs in satisfiable cases of $C_{\mathcal{ALCQ}}$ with $i = j$, and $i = 10^k$ in case of increasing the numbers exponentially. No results can be reported for Hermit and Fact++, which crash with these test cases.

Each figure consists of two subfigures, subfigure (a) considers increasing the numbers linearly, and subfigure (b) considers increasing the numbers exponentially. Whenever a test case result is not shown in the figure, this is because the corresponding reasoner either returned an error or timed out. It is easy to see that the performance of HARD and RacerPro is not affected by the size of the numbers used in a QCRs, even when the numbers are very large and i is increased exponentially. On the other hand, the other reasoners are dramatically affected by the increase in the size of the numbers, even when the increase is only linear and with very small values of i. For example, when testing $C_{SAT-lin-\mathcal{ALCQ}}$ case for $i = 6$ Fact++ and Hermit cannot handle the test case due to some error. Also, Pellet times out whenever $i \geq 8$. In the case when the numbers are increased exponentially, only HARD and RacerPro can decide the TBox consistency even in less than 0.5 seconds, none of other reasoners can handle any case either due to some error (Fact++) or because the reasoner cannot decide the test within the time limit (Pellet, Hermit).

The stability of HARD and RacerPro in solving these test cases shows the advantage of solving QCRs using algebraïc reasoning over adopting another reasoning approach. Notice that the due to the use of *proxy nodes* (see Definition 4.5.2), the same CCG is valid for cases where the numbers are increased linearly and for those when the numbers are increased exponentially. In the first case, the proxy nodes b and c represent i elements. In the latter case, they represent 10^k $(1 \leq k \leq 6)$ elements. On the other hand, HARD requires slightly more time to solve unsatisfiable cases and this is because most unsatisfiable cases presented in this thesis are usually harder to solve than the satisfiable ones.[6] In particular, all possible attempts to solve the encoded in-equations are exhausted before deciding unsatisfiability.

Consulting the statistics file generated by HARD shows that HARD spends more

---

[6]For a comparison of performance between satisfiable and unsatisfiable cases see Section C.1.

time trying to find a solution for the generated in-equations than it does in satisfiable cases. Such behaviour is expected because in this case, HARD tries all possible expansions of the *ch*-Rule, without any specific optimizations to avoid repetitively solving the same in-equations. In the cases with Fact++, Hermit and Pellet, the performance dramatically degrades with unsatisfiable cases and this is because if no particular optimization is used to detect numerical clashes, these reasoners need to try all possible attempts to merge the $2i$ fillers created to satisfy the at-least restriction and exceeding the numbers $(i-1)$ and i allowed in the at-most restrictions before deciding unsatisfiability.

A variant of $C_{\mathcal{ALCQ}}$ is considered by adding *role hierarchies* and is defined using the DL $\mathcal{ALCHQ}$ as follows:

$$C_{\mathcal{ALCHQ}} \sqsubseteq\ \geq 2irs.(A \sqcup B) \sqcap\ \leq is.A \sqcap\ \leq ir.B \sqcap (\leq (i-1)t.\neg A \sqcup\ \leq it.\neg B)$$

such that the roles, rs, s, r and t follow the hierarchy defined in the RBox $\mathcal{R}$.

$$\mathcal{R} = \left\{ \begin{array}{l} r \sqsubseteq t, s \sqsubseteq t \\ rs \sqsubseteq r, rs \sqsubseteq s \end{array} \right\}$$

Figure 49 shows the results in case of increasing the numbers linearly in satisfiable and unsatisfiable cases. For most reasoners, running the test cases using the DL $\mathcal{ALCHQ}$ did not seem to have any effect on their performance versus running the test cases without *role hierarchies*. The only affected reasoners were Fact++ and Pellet which implement tableau-based reasoning. When deciding on cases with *role hierarchies* ($C_{\mathcal{ALCHQ}}$), Fact++ is slower in deciding satisfiable cases and Pellet is slower in deciding unsatisfiable ones.
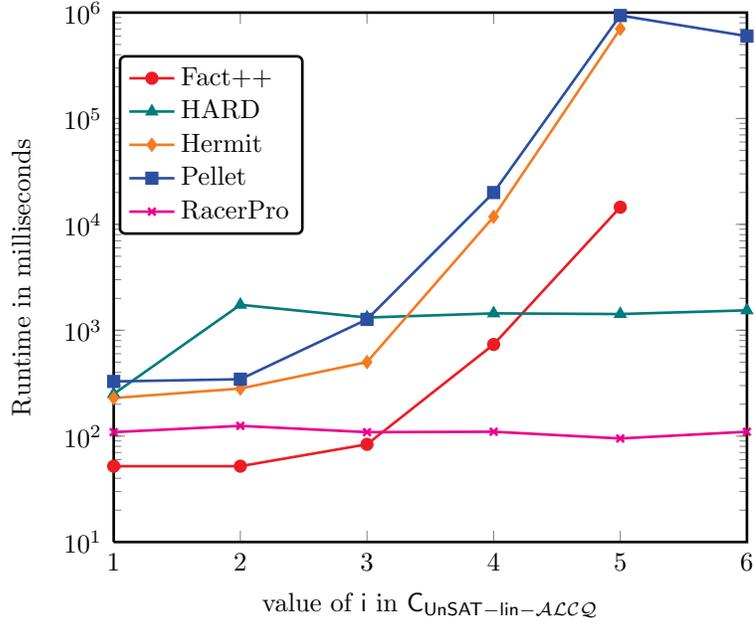
(a) Increasing i *linearly* in an *unsatisfiable* concept expression
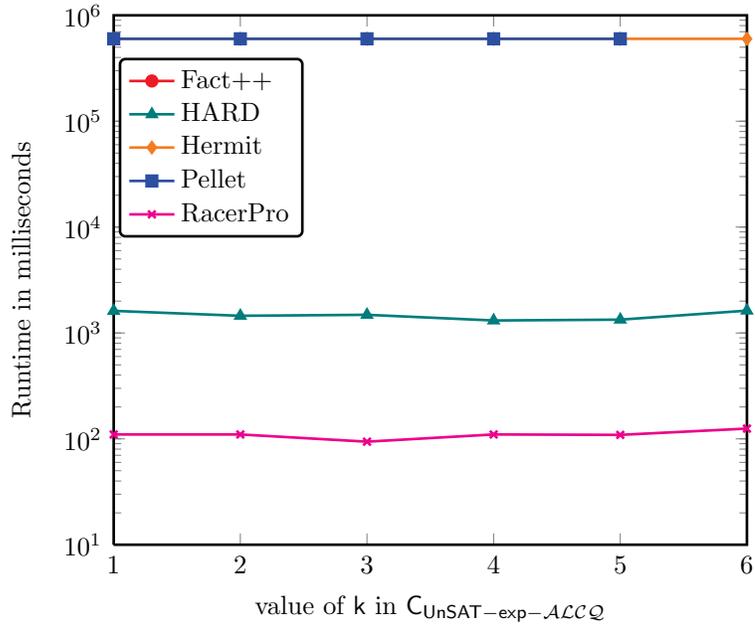
(b) increasing i *exponentially* in an *unsatisfiable* concept expression;

$i = 10^k, 1 \leq k \leq 6$

Figure 48: Effect of increasing the size of the numbers used in QCRs with unsatisfiable cases of $C_{\mathcal{ALCQ}}$ where $j = i - 1$, and $i = 10^k$ in case of increasing the numbers exponentially.
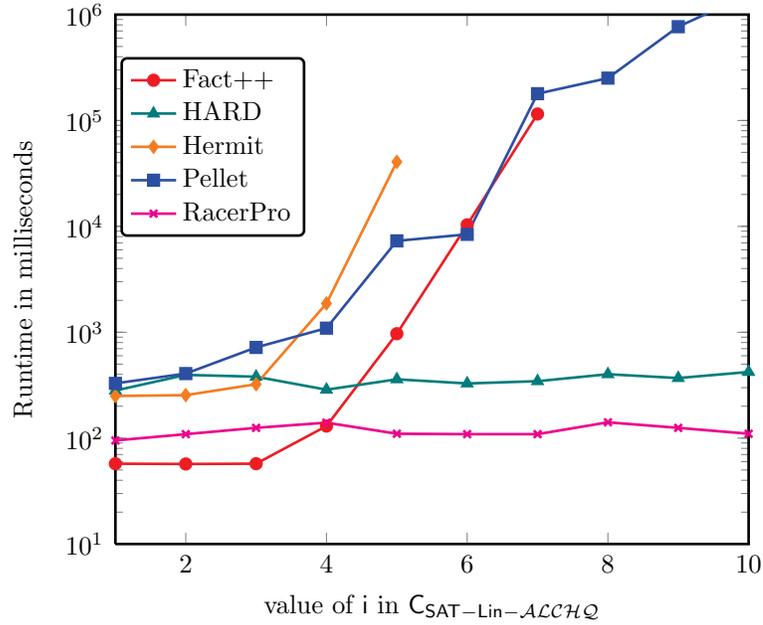
(a) Increasing i *linearly* in a *satisfiable* concept expression.
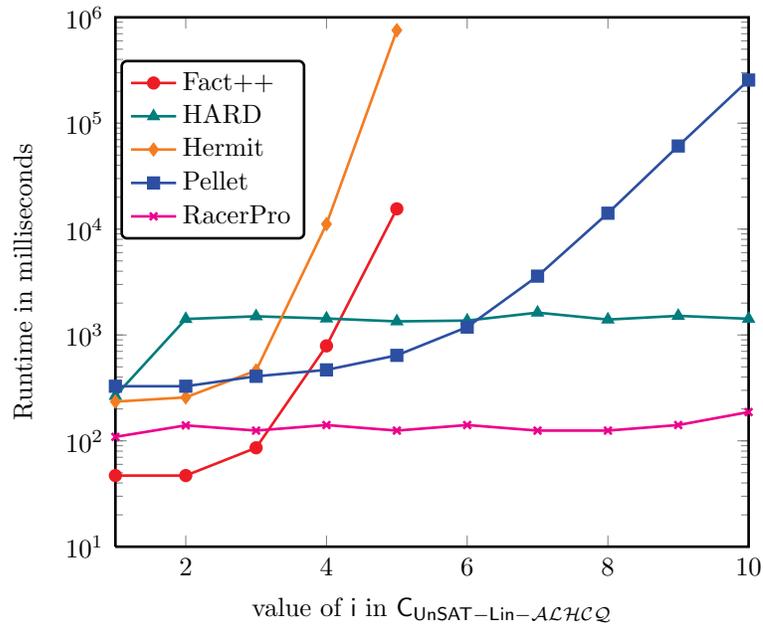
(b) increasing i *linearly* in an *unsatisfiable* concept expression.

Figure 49: Effect of *linearly* increasing the size of the numbers used in QCRs with unsatisfiable cases of $C_{\mathcal{ALCHQ}}$ where $j = i - 1$ and the hierarchy between the roles used is not flat.

7.2.1.2 Testing The Number of Qualified Cardinality Restrictions

It was shown in Section 4.6 that the complexity of the hybrid algorithm implemented by HARD is characterized by a double exponential function of the number of QCRs, q. It is therefore expected that as the number of QCRs increases, the performance of HARD degrades. The effect of increased number of QCRs is tested using the concept $C_{QCR-\mathcal{ALCQ}}$ defined below:

$$
\begin{aligned}
C_{QCR-\mathcal{ALCQ}} \quad &\sqsubseteq \; \geq 4nr.\top \\
&\sqcap \geq 2nr.C_1 \sqcap \ldots \sqcap \geq 2nr.C_i \\
&\sqcap \leq nr.(\neg C_1 \sqcup \neg C_2) \sqcap \ldots \sqcap \leq nr.(\neg C_i \sqcup \neg C_{i+1})
\end{aligned}
$$

As the name suggests, $C_{QCR-\mathcal{ALCQ}}$ is defined using the expressivity of the DL $\mathcal{ALCQ}$.

Using a non flat role hierarchy   Another variant of $C_{QCR-\mathcal{ALCQ}}$ is tested by using a non flat role hierarchy between the roles, rs and r, used in QCRs as defined in the RBox $\mathcal{R} = \{rs \sqsubseteq r\}$. We refer to this variant as $C_{QCR-\mathcal{ALCHQ}}$ defined as follows:

$$
\begin{aligned}
C_{QCR-\mathcal{ALCHQ}} \quad &\sqsubseteq \; \geq 4nrs.\top \\
&\sqcap \geq 2nr.C_1 \sqcap \ldots \sqcap \geq 2nr.C_i \\
&\sqcap \leq nr.(\neg C_1 \sqcup \neg C_2) \sqcap \ldots \sqcap \leq nr.(\neg C_i \sqcup \neg C_{i+1})
\end{aligned}
$$

Using different values for the numbers used in QCRs   Also, one variant of $C_{QCR-\mathcal{ALCQ}}$ is considered by using different values for the numbers used in QCRs as was done in [HSV11].[7] We refer to this variant as $C_{QCR-var-\mathcal{ALCQ}}$ defined as follows:

---

[7]The results in [HSV11] show that using a different number for each QCR degrades the performance of $\mathcal{ALCQ}2SMT$ reasoning algorithm.

$$C_{\mathsf{QCR-var-}\mathcal{ALCQ}} \quad \sqsubseteq\; \geq 4nr.\top$$

$$\sqcap \geq 2nr.C_1 \sqcap \geq 2(n-1)r.C_2 \sqcap \cdots \sqcap \geq 2(n-i+1)r.C_i$$

$$\sqcap \leq nr.(\neg C_1 \sqcup \neg C_2) \sqcap \cdots \sqcap \leq (n-1)r.(\neg C_2 \sqcup \neg C_3)$$

$$\leq (n-i+1)r.(\neg C_i \sqcup \neg C_{i+1})$$

The various definitions of the concept $C_{\mathsf{QCR-*-}\mathcal{ALCQ}}$ consist of one at-least restriction ($\geq 4nr.\top$) gradually extended for each value of i such that i ranges between 1 and 10 and $C_{\mathsf{QCR-*-}\mathcal{ALCQ}}$ remains satisfiable for all values of i. The ratio of the number of at-least restrictions to the number of at-most restrictions is kept fixed by extending ($\geq 4nr.\top$) with i at-least and i at-most restrictions for every instance of the problem. The number of QCRs is a function of i: $q = 2i + 1$, and the numbers used for each restriction range between 1 ($n = 1$), and 40 ($n = 10$). For all values of i ($1 \leq i \leq 10$) and n ($n = 1, 5, 10$) considered, the concept definition of $C_{\mathsf{QCR-*-}\mathcal{ALCQ}}$ is satisfiable.

The effect of increasing the number of QCRs on reasoning performance is shown in Figures 50 and 51. In Figures 50a and 50b the results are shown for $n = 1$ and $i = 1, 2, \ldots, 20$, and $n = 5$ and $i = 1, 2, \ldots, 10$, respectively, for $C_{\mathsf{QCR-}\mathcal{ALCQ}}$. Figures 51a and 51b show the results for $n = 10$, $i = 1, 2, \ldots, 10$ for $C_{\mathsf{QCR-}\mathcal{ALCQ}}$ and $C_{\mathsf{QCR-var-}\mathcal{ALCQ}}$ respectively. Clearly, Fact++ is the only reasoner which maintains a stable runtime for different values of n and i. Hermit does not seem to be affected by the number of QCRs when $n = 1$, however, and surprisingly Hermit could not solve any of the tests for $n = 5, 10$[8]. This might be due to the fact that the individual re-use optimization is only effective with $\exists R.C$ equivalent to $\geq 1R.C$, and therefore as i grows, the effect of this optimization becomes minimal.

---

[8]With these test cases Hermit would quickly timeout or run out of memory for the JAVA heap space.
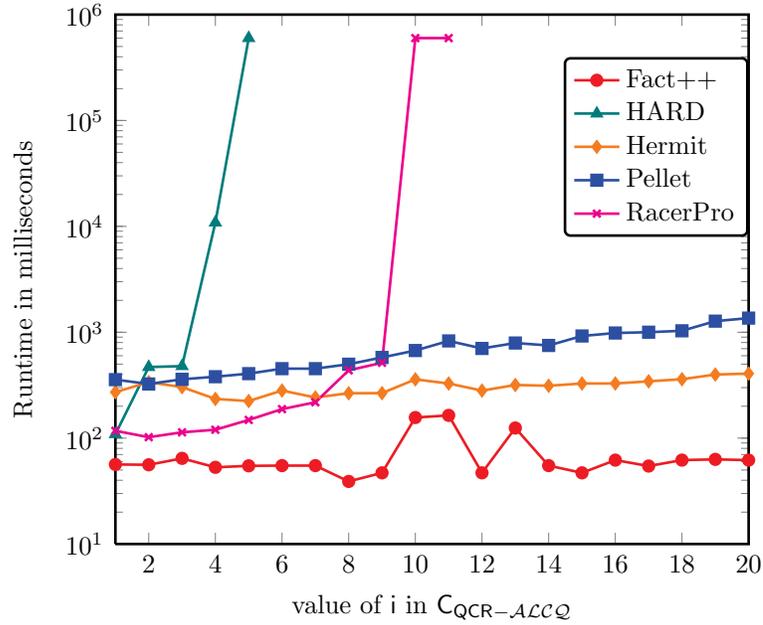
(a) Using the *same* values with added QCRs with n = 1.
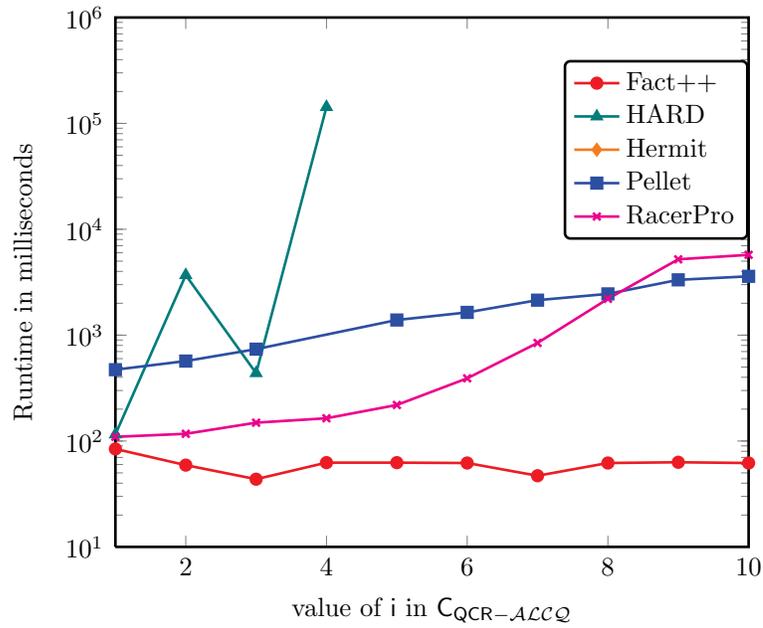
(b) Using the *same* values with added QCRs with n = 5.

Figure 50: Effect of increasing the number of QCRs in a *satisfiable* concept expression.
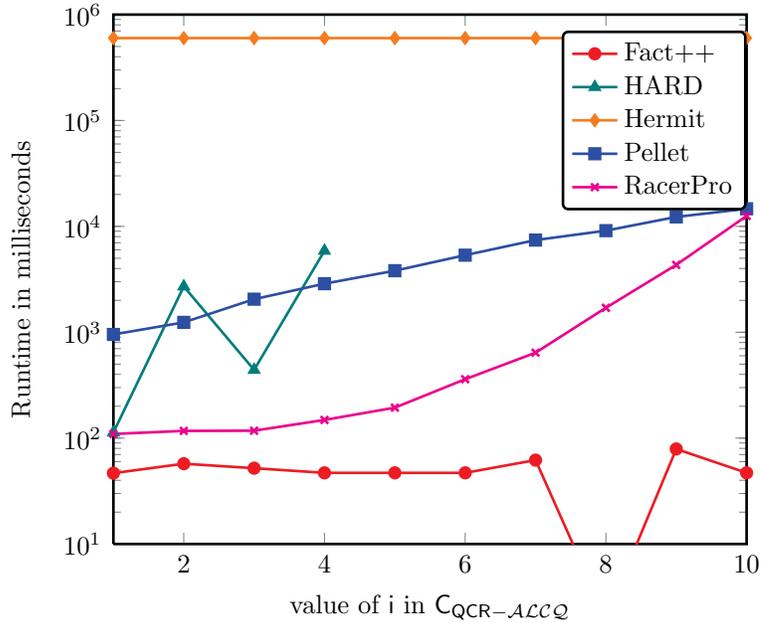
(a) Using the *same* values with added QCRs with $n = 10$.
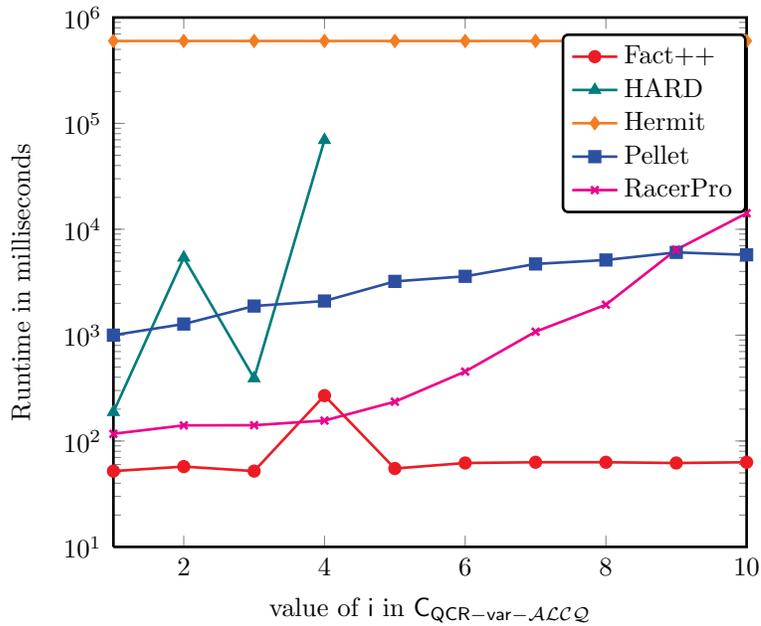
(b) Using *different* values with added QCRs with $n = 10$.

Figure 51: Effect of increasing the number of QCRs in a *satisfiable* concept expression.

Pellet is affected by the number of QCRs and by the values of the numbers used. The overall performance seems to be affected by the total number of individuals that

the reasoner needs to create or merge in order to satisfy the restrictions imposed by the QCRs. To better illustrate this, we take for example the case where $i = 6$ and $n = 10$ and compare the results for $C_{QCR-var-\mathcal{ALCQ}}$ and $C_{QCR-\mathcal{ALCQ}}$. Clearly Pellet can solve $C_{QCR-var-\mathcal{ALCQ}}$ because the total number of individuals needed to satisfy the at-least restrictions is 105 whereas with $C_{QCR-\mathcal{ALCQ}}$ the number of individuals created to satisfy the at-least restrictions grows to 120. RacerPro is dramatically affected by the number of QCRs and cannot solve the tests for $i \geq 10$.

These test cases can be considered very hard for both tableau and algebraïc reasoning procedures. As the number of at-least restrictions increases the number of individuals that need to be created also increases. Also, as the number of at-most restrictions increases the number of ways that the individuals can be distributed also increase. The numbers used in QCRs consistently degrade the performance of tableau reasoners as they grow. For algebraïc reasoners, the number of QCRs means a decomposition set with a larger size and a *ch*-Rule with more cases to consider which consistently degrade the performance of HARD. For $i \geq 6$ HARD runs into a stack overflow error.[9]

When comparing the results between the use of different values for n we noticed a difference in performance with HARD. The fact that HARD is not sensitive to the values used in QCRs, and the fact that HARD's performance was sometimes faster with greater values of n, suggest that the performance speedup/degradation was due to some factor other than the number of cardinality restrictions used or the values of the numbers used with those QCRs. After analyzing the information generated in the statistics file in terms of the applicability of the *ch*-Rule and the different types of clashes returned, we noticed that the performance degradation was associated with a greater number of applicability of the *ch*-Rule and a greater number

---

[9]This error is due to a JAVA limitation with numbers. However, alternative ways of representing the numbers could be investigated such as using `java.lang.BigNum`.

of arithmetic clashes. For example, for three separate runs of the test $C_{\mathsf{QCR}-\mathcal{ALCQ}}$ with $\mathsf{i} = 2$ and $\mathsf{n} = 1, 5, 10$, the number of times the *ch*-Rule was applied was 424, 26, and 1512 respectively. We also compared the number of times the *ch*-Rule is applicable for different runs of $C_{\mathsf{QCR}-\mathcal{ALCQ}}$ for the same values of i and n. It turned out that the order in which the *ch*-Rule selects variables (to branch on) greatly affects performance. Some variable choices affect how quickly a distribution of individuals is found, and other variable choices could result in hitting the worst case because all possible explorations of these variables need to be tried before a solution is found.

We also considered some variant of $C_{\mathsf{QCR}-\mathcal{ALCQ}}$ where we use disjunctive descriptions between at-least/at-most restrictions. Although the size of the global decomposition set remains the same, different decompositions need to be activated/computed and a smaller set of in-equations needs to be considered in each branch. We refer to this test case as $C_{\mathsf{QCR-disjunctive-atLeast}-\mathcal{ALCQ}}$ when the disjunctions are considered between at-least restrictions,

$$
\begin{aligned}
C_{\mathsf{QCR-disjunctive-atLeast}-\mathcal{ALCQ}} \quad &\sqsubseteq (\geq 4\mathsf{nr}.\top \sqcup \geq 2\mathsf{nr}.C_1 \sqcup \cdots \sqcup \geq 2\mathsf{nr}.C_i) \\
&\sqcap \leq \mathsf{nr}.(\neg C_1 \sqcup \neg C_2) \sqcap \cdots \sqcap \leq \mathsf{nr}.(\neg C_i \sqcup \neg C_{i+1})
\end{aligned}
$$

and $C_{\mathsf{disjunctive-atMost}-\mathcal{ALCQ}}$ when the disjunctions are considered between at-most restrictions.

$$
\begin{aligned}
C_{\mathsf{QCR-disjunctive-atMost}-\mathcal{ALCQ}} \quad &\sqsubseteq \geq 4\mathsf{nr}.\top \sqcap \geq 2\mathsf{nr}.C_1 \sqcap \cdots \sqcap \geq 2\mathsf{nr}.C_i \\
&\sqcap (\leq \mathsf{nr}.(\neg C_1 \sqcup \neg C_2) \sqcup \cdots \sqcup \leq \mathsf{nr}.(\neg C_i \sqcup \neg C_{i+1}))
\end{aligned}
$$
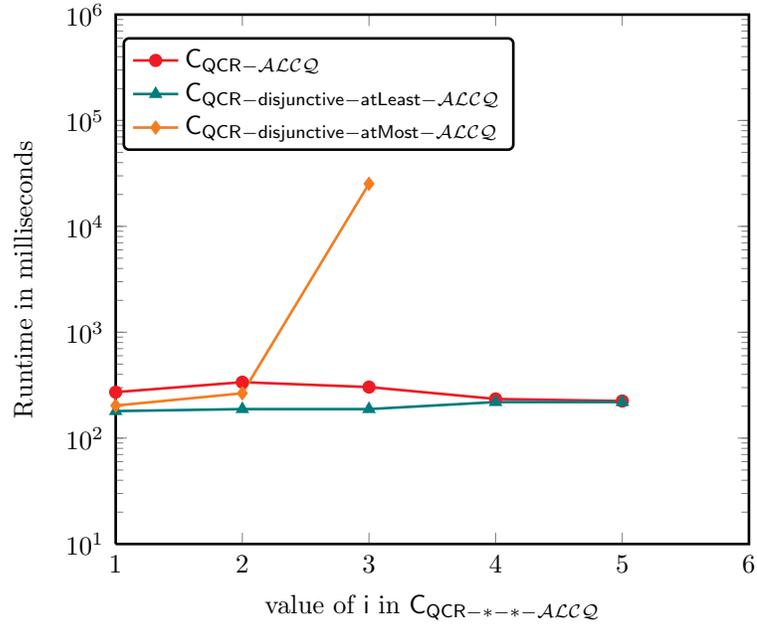
Figure 52: Effect of disjunctions between QCRs on performance of Hermit.
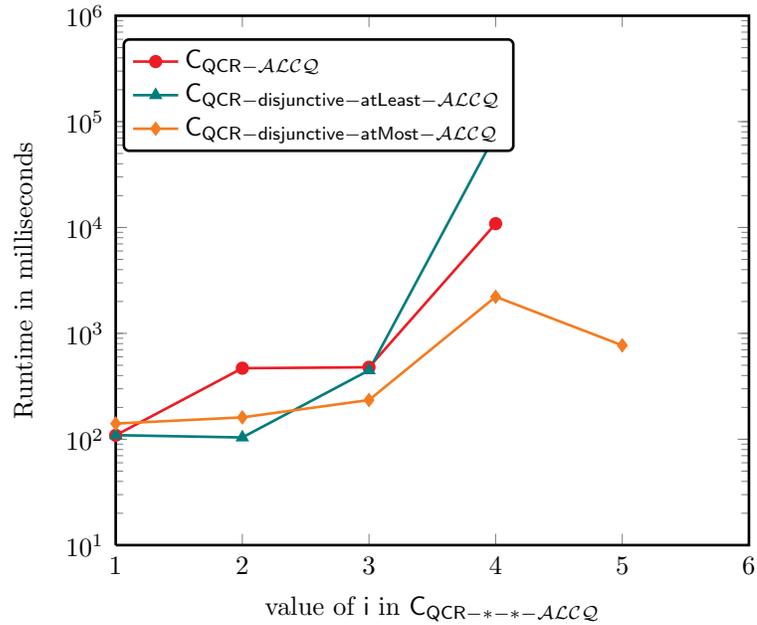
Figure 53: Effect of disjunctions between QCRs on performance of HARD.

The reasoners implementing tableau-based reasoning like Fact++, Pellet, and RacerPro did not have any performance degradation/speedup in solving different

variants of $\mathsf{C}_{\mathsf{QCR}-*-*-\mathcal{ALCQ}}$. However, Hermit and HARD seem to be affected by disjunctions; Hermit is slightly faster with $\mathsf{C}_{\mathsf{QCR}-\mathsf{disjunctive}-\mathsf{atLeast}-\mathcal{ALCQ}}$ cases, but Hermit quickly runs out of memory with $\mathsf{C}_{\mathsf{disjunctive}-\mathsf{atMost}-\mathcal{ALCQ}}$ cases even for small values of i as shown in Figure 52. HARD, on the other hand, performs better with $\mathsf{C}_{\mathsf{QCR}-\mathsf{disjunctive}-\mathsf{atMost}-\mathcal{ALCQ}}$ as shown in Figure 53. The statistics file showed that HARD spends less time solving the in-equations. This might be due to the fact that it is easier to solve a system of in-equations with less at-most restrictions than it is with less at-least restrictions.

7.2.1.3 Ratio of the Number of at-least to the Number of at-most Restrictions

The ratio of the number of at-least to the number of at-most restrictions is defined as $R_{QCRs} = \frac{Q_{at-least}}{Q_{at-most}}$. The previous set of benchmarks focused on increasing the number of QCRs without affecting the ratio between the number of at-least and the number of at-most restrictions used, $R_{QCRs} = \frac{\mathsf{i}+1}{\mathsf{i}}$. In this set of benchmarks, we fix the total number of QCRs used and change the ratio, R_{QCRs}. We use a satisfiable concept expression C whose pattern is similar to the one defined in the previous section. The total number of QCRs is fixed to m such that for each test case, the number of at-most restrictions varies with i and the number of at-least restrictions is equivalent to $(\mathsf{m} - \mathsf{i})$. We test $\mathsf{C}_{\mathcal{ALCQ}}$ with $\mathsf{m} = 10$ and $\mathsf{n} = 1$ because for larger values of n some reasoners like Hermit had performance degradation.[10]

$$
\begin{aligned}
\mathsf{C}_{\mathsf{QRatio}-\mathcal{ALCQ}} \ &\sqsubseteq\ \geq 4\mathsf{nr}.\top \\
&\sqcap\ \geq 2\mathsf{nr}.\mathsf{C}_1 \sqcap \ldots \sqcap\ \geq 2\mathsf{nr}.\mathsf{C}_{\mathsf{m}-\mathsf{i}-1} \\
&\sqcap\ \leq \mathsf{nr}.(\neg\mathsf{C}_1 \sqcup \neg\mathsf{C}_2) \sqcap \ldots\ \leq \mathsf{nr}.(\neg\mathsf{C}_\mathsf{i} \sqcup \neg\mathsf{C}_{\mathsf{i}+1})
\end{aligned}
$$

---

[10]Hermit cannot solve the test cases for larger values of n within the time limit.

The effect of the R_{QCRs} on the performance of reasoning is shown in Figure 54. All existing reasoners need negligible time to decide on the satisfiability of $C_{\mathsf{QRatio}-\mathcal{ALCQ}}$ and maintain a stable response time as R_{QCRs} changes except for HARD. It is hard to associate the performance degradation of HARD to an increasing variable. Notice that all the test cases share a common overhead which is the size of $\mathcal{DS}$. This means that in the worst case and for all the test cases HARD needs to consider the same number of possibilities for the ch-Rule and for partitions. What makes a test case run faster/slower is how quickly a distribution of r-fillers is found. Since the qualifications are not necessarily disjoint, this means that what affects the run-time is the solvability of the encoded in-equations. Notice for example, that when there are no at-least (at-most) restrictions, the set of in-equations consists only of at-most (at-least) restrictions and since the arithmetic solver tries to minimize, the solution is trivial and that's why the test cases for ($i = 0, 10$) have similar performance. This is in contrast to [FH10c], where the results showed that the performance is affected by the number of at-least restrictions. We cannot conclude the same because the implementation of our ch-Rule does not depend on at-least restrictions. As long as some at-least restrictions exists, the more at-most restrictions are there, the harder it becomes to find an optimal solution for the generated set of in-equations. The statistics file showed that even for test cases with similar numbers of choices for the ch-Rule, more time was spent in the constraint solver. Configuring different choices for the objective function did not seem to have any effect on the performance of the constraint solver. This set of test cases suggests that some work can be done to enhance the constraint solver or find a better one than LPSolver. Similar to the result shown with $C_{\mathsf{QCR}-\mathsf{disjunctive}-\mathsf{atMost}-\mathcal{ALCQ}}$, the less at-most restrictions, the faster is the reasoning and less time is spent in the Constraint Solver.

211
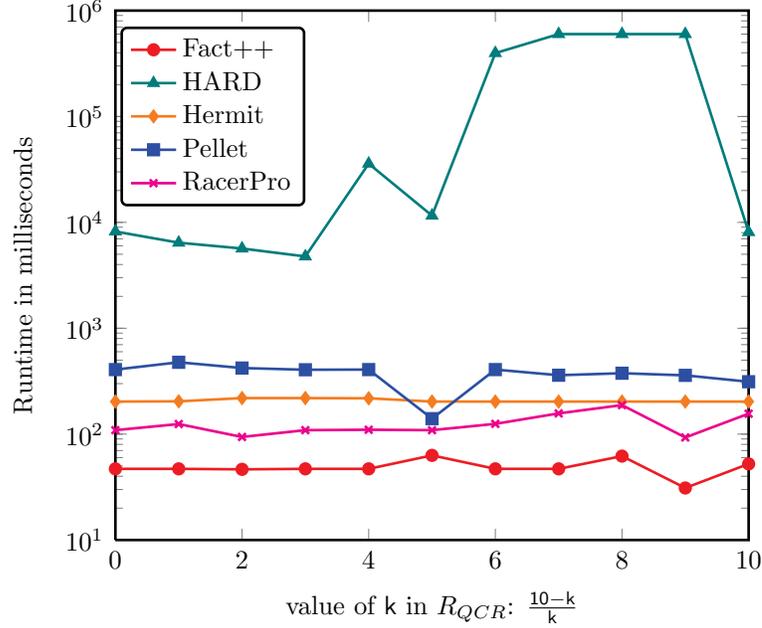
Figure 54: Effect of the ratio R_{QCR} of the number of at-least to the number of at-most restrictions in a concept expression.

7.2.1.4 Satisfiable Versus Unsatisfiable Concepts

The numbers used in QCRs affect the satisfiability of concept expressions. We evaluate reasoning performance for satisfiable and unsatisfiable cases using a set of test cases based on the following concept expression:

$$\begin{aligned}
\mathsf{C_{SAT-UnSAT-\mathcal{ALCQ}}} \quad & \sqsubseteq\; \geq 3\mathsf{nr}.(\mathsf{A} \sqcap \mathsf{B}) \sqcap\; \geq 3\mathsf{nr}.(\neg\mathsf{A} \sqcap \mathsf{B}) \\
& \sqcap\; \geq 3\mathsf{nr}.(\mathsf{A} \sqcap \neg\mathsf{B}) \sqcap\; \geq 3\mathsf{nr}.(\neg\mathsf{A} \sqcap \neg\mathsf{B}) \\
& \sqcap\; \leq \mathsf{inr}.\top
\end{aligned}$$

The number of QCRs is fixed to 5 as well as the ratio between at-least and at-most restrictions ($R_{QCR} = 4$). Each at-least restriction requires $3\mathsf{n}$ r-fillers such that no two at-least restrictions can share their r-fillers because these r-fillers satisfy mutually disjoint qualifications. Therefore, a domain element $\mathsf{a} : \mathsf{C_{SAT-UnSAT}}_{\mathcal{ALCQ}}$ requires at-least $12\mathsf{n}$ distinct r-fillers to render the concept $\mathsf{C_{SAT-UnSAT-\mathcal{ALCQ}}}$ satisfiable. When

212

$i \geq 12n$, $C_{\mathsf{SAT-UnSAT}-\mathcal{ALCQ}}$ is satisfiable, otherwise when $i < 12n$, $C_{\mathsf{SAT-UnSAT}-\mathcal{ALCQ}}$ is not satisfiable. We run our experiments using $n = 1, 10$ and $i = 1, 2, 4, \ldots, 24$. The results are shown in Figure 55 with Figure 55a showing the results for $n = 1$ and Figure 55b showing the results for $n = 10$.

The advantage of adopting algebraïc reasoning is clearly shown by the stability of both HARD and RacerPro in deciding satisfiable and unsatisfiable cases for both cases $n = 1$ and $n = 10$.[11] For most reasoners, and with a small number of individuals, in case when $n = 1$, there is no difference in performance between satisfiable or unsatisfiable cases as illustrated in Figure 55a. With the exception of Hermit whose performance degrades as the value of i increases whether a case is satisfiable or not. In fact, Hermit's performance deteriorates with the cases when $n = 10$. The effect of satisfiable and unsatisfiable cases on tableau-based reasoners is better shown in Figure 55b when a larger numbers of individuals needs to be managed. Fact++ performance degrades up to 2 orders of magnitude with unsatisfiable cases. Pellet on the other hand, cannot decide on satisfiable cases within the time limit, and its performance is 2 orders of magnitude slower than that of HARD when deciding unsatisfiable cases.

---

[11]In [FH10c] the hybrid reasoning algorithm showed unexpected instability because in some cases the arithmetic reasoner needed more time to decide on the unsatisfiablity of the encoded in-equations.
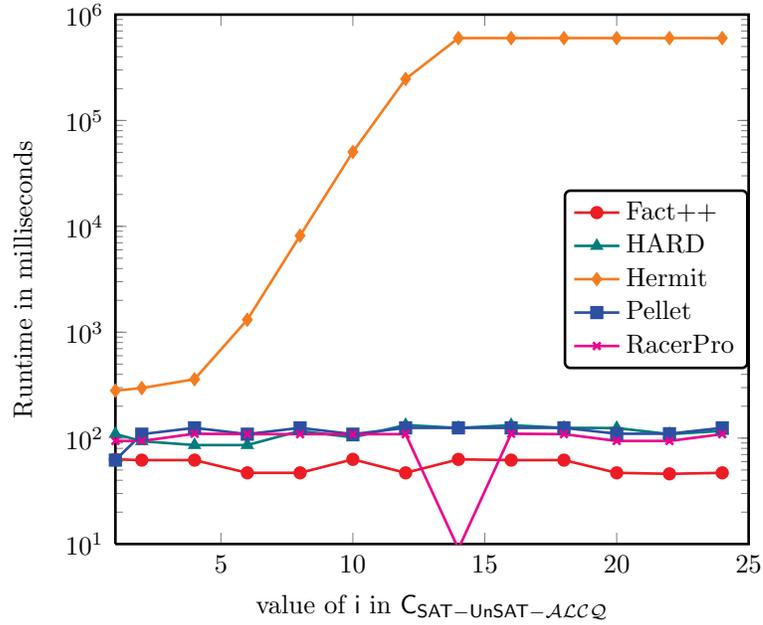
(a) Using *small* numbers in QCRs, $n = 1$
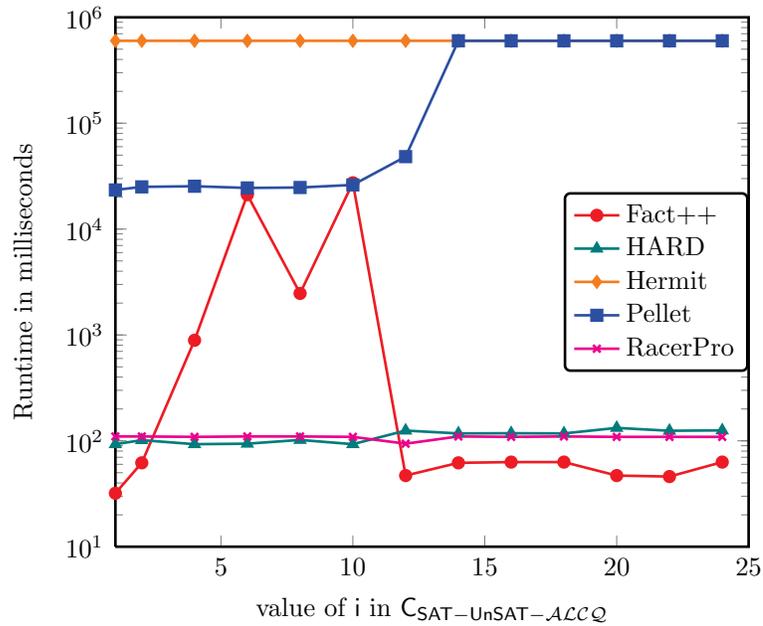
(b) Using *large* numbers in QCRs, $n = 10$

Figure 55: Effect of the numbers in QCRs in *satisfiable* versus *unsatisfiable* expressions.

7.2.1.5 Backtracking

The effect of backtracking is tested using a set of TBox consistency tests where every TBox $\mathcal{T}$ includes the following axioms.

$$C_{\mathsf{Back}-\mathcal{ALCQ}} \quad \sqsubseteq\; \geq 3r.D_1 \sqcap \ldots \sqcap\; \geq 3r.D_i \sqcap\; \leq 3i-1r.\top$$

$$D_q \qquad\qquad \sqsubseteq \neg D_p \text{ for all } q < p$$

$C_{\mathsf{Back}-\mathcal{ALCQ}}$ is satisfiable if a domain element $a : (C_{\mathsf{Back}-\mathcal{ALCQ}})$ has $3i$ r-fillers such that, for a given value of i, 3 r-fillers satisfy each $\geq 3r.D_i$ and the total number of these r-fillers cannot exceed $3i-1$. This means that some r-fillers must satisfy $(D_q \sqcap D_p)$ for some $1 \leq p, q \leq i$, however this is not possible because all D_p and D_q are disjoint. This renders the concept $C_{\mathsf{Back}-\mathcal{ALCQ}}$ unsatisfiable. In the case of a tableau algorithm, backtracking is involved each time an r-filler satisfying D_p is merged with an r-filler satisfying D_q. With algebraïc reasoning, backtracking is involved each time a distribution of individuals puts r_i-fillers and r_j-fillers in the same partition (i.e the *ch*-Rule assigns the corresponding variable ≥ 1).

Figure 56 shows that Hermit implements poor backtracking strategies whereas HARD implements backtracking strategies that are competitive with the ones implemented by SOTA reasoners. Notice however that $C_{\mathsf{Back}-\mathcal{ALCQ}}$ does not rely on any disjunctive descriptions. This means that non-determinism is due to the *ch*-Rule for algebraïc reasoning, and the non-determinism in merging individuals for tableau reasoning. Notice that the numbers used are very small, this means that in the case where individuals are distributed over distinct partitions, the *ch*-Rule might perform as many non-deterministic choices as the $\leq$-Rule for merging individuals.
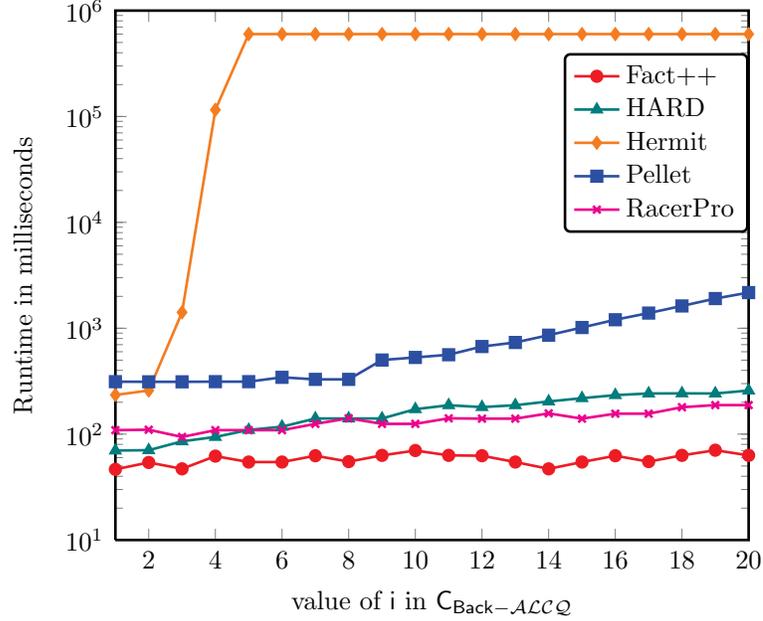
Figure 56: Effect of backtracking with $C_{\mathsf{Back}-\mathcal{ALCQ}}$.

A more complicated test case is considered where disjunctive QCRs are used with disjunctive qualifications.

$$C_{\mathsf{Back-disjunctive}-\mathcal{ALCQ}} \quad \sqsubseteq\ \geq (j+1)r.D_1 \sqcap \ldots \sqcap\ \geq (j+1)r.D_i$$
$$\sqcap\ \leq jr.(D_1 \sqcup D_2) \sqcup\ \leq jr.(D_2 \sqcup D_3) \sqcup \ldots \sqcup\ \leq jr.(D_{i-1} \sqcup D_i)$$
$$D_q \qquad\qquad\qquad \sqsubseteq \neg D_p \text{ for all } q < p$$

Non-determinism in tableau reasoning now has three sources: the *choose*-Rule (or *ch*-Rule for algebraïc reasoning), the $\sqcup$-Rule, and non-determinism in merging individuals exceeding the number in the at-most restriction. Since each one of $\sqcup$-Rule and *choose*-Rule rules is applicable to each created individual, the greater the size of j, the less efficient the reasoning; Figure 57 shows how increasing j with just one number affects performance.

216
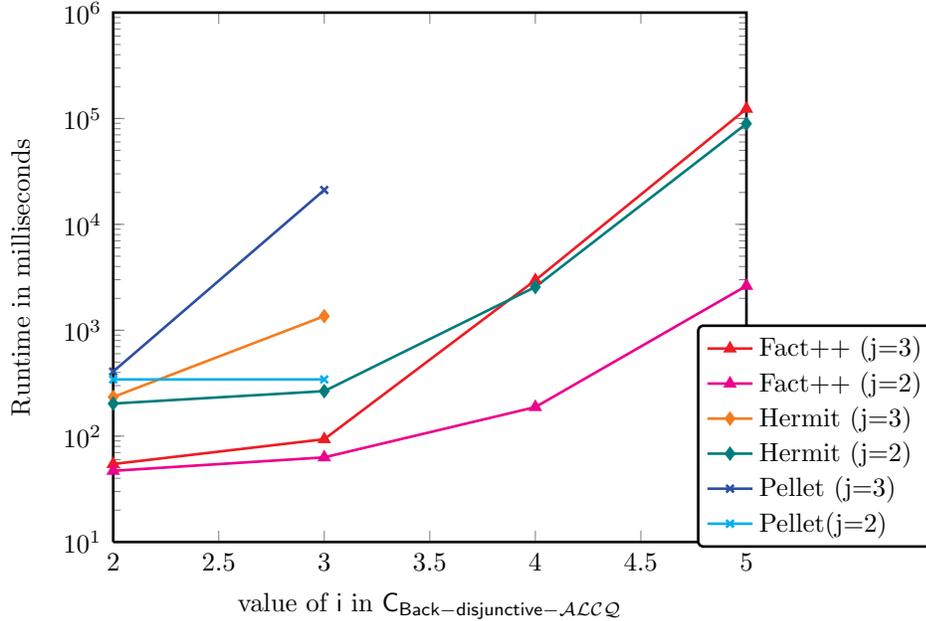
Figure 57: Effect of backtracking with $C_{Back-disjunctive-\mathcal{ALCQ}}$.

We focus on the effect of non-determinism and backtracking and run the test with $j = 3$. The results are shown in Figure 58. Racer and HARD are the best in dealing with disjunctions, whereas, Hermit and Pellet easily fail with only 2 disjunctions. Fact++ scales better, but, it quickly times out for $i \geq 4$. This test case shows how the interaction between non-determinism in the *choose*-Rule and the $\sqcup$-Rule blows up the search space of tableau reasoning. Whereas, the algebraïc approach allows a better scalability because of two main things: (1) the *ch*-Rule is applied for every variable, which could represent n individuals, which means that the semantic split over the qualifications used in at-most restrictions is done over groups of individuals rather than for each individual as is the case with the *choose*-Rule. Non-Determinism in concept descriptions does not interact with non-determinism in distributing individuals among the restrictions used as qualifications in at-most restriction. This allows a more fine grained backtracking.
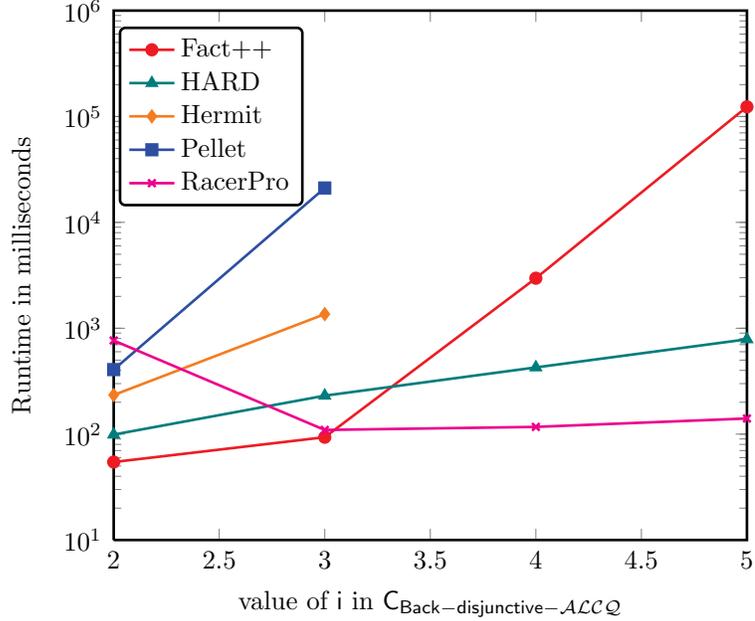
Figure 58: Effect of backtracking with $C_{\mathsf{Back-disjunctive}-\mathcal{ALCQ}}$.

We also run the test where $\leq \mathsf{jr}.(D_1 \sqcup D_2) \sqcup \leq \mathsf{jr}.(D_2 \sqcup D_3) \sqcup \ldots \sqcup \leq \mathsf{jr}.(D_{i-1} \sqcup D_i)$ is replaced with $\leq \mathsf{jr}.(D_1 \sqcup D_2 \sqcup D_3 \sqcup \ldots \sqcup D_i)$ with values of i between 1 and 10. Fact++, Racer and HARD can solve the test very quickly (less than half a second). However, Pellet and Hermit cannot solve the test within the time limit even for small values of i, (e.g., 2).

Although HARD performs better than other non-algebraïc reasoners, we noticed that the order in which the *ch*-Rule is applied has a dramatic effect on performance. Although the reported run-time is still high, there are runs where HARD was able to return results much faster. The difference in run-times was associated with a different order in which the *ch*-Rule was applied because it affects how quickly a *noGood* variable is discovered. In fact, the sooner *noGood* variables are discovered the more efficient is the pruning of the search space.

7.2.2 Test Cases for QCRs and Nominals

This section presents an evaluation of the performance of HARD using test cases, which rely on the use of *nominals* and QCRs, where the two constructors ($\mathcal{O}$ and $\mathcal{Q}$) interact. The used test cases range between some adaptations of real world ontologies, and some synthetic ones. The real world ontologies are selected such that they are small in size, and use *nominals* and QCRs in concept descriptions; such ontologies are the TIME[12], the KOALA, the COUNTRIES, and the WINE ontologies which are part of the Protégé ontology library, and which can be downloaded from `http://protege.stanford.edu/download/ontologies.html`. The synthetic ontologies are designed to test the scalability of the algebraic reasoning approach based on the following parameters: the number of *nominals* used (size of N_o), the depth of the role hierarchy, the use of cycles within concept descriptions, and the occurrence of QCRs within nested descriptions.

7.2.2.1 Tests Cases Based on Real World Ontology Adaptation

The characteristics of the adapted ontologies are summarized in Table 6. Notice that the expressivity of the DL underlying the input file ontology is adapted to the expressivity handled by HARD.

Ontology	Original DL expressivity	Nominals in adapted version	Roles in adapted version	Adapted DL expressivity
WINE	$\mathcal{SHOIN}_{(D)}$	12	5	$\mathcal{ALCHOQ}$
KOALA	$\mathcal{ALCON}_{(D)}$	6	4	$\mathcal{ALCOQ}$
COUNTRIES	$\mathcal{ALCIN}_{(D)}$	13	4	$\mathcal{ALCOQ}$
TIME	$\mathcal{SHOIN}_{(D)}$	14	3	$\mathcal{ALCON}$

Table 6: General characteristics of test cases based on adapted real world ontologies.

---

[12]`http://www.w3org/2006/time`.

7.2.2.2   The WINE ontology

One cannot consider *nominals* in DL without referring to the WINE ontology which relies heavily on the use of *nominals* for representing the different kinds of wines based on their flavour, colour, texture, etc . . .. The WINE ontology is designed using the DL $\mathcal{SHOIN}_{(D)}$ and it contains 206 *nominals*. It was originally developed as a DL example for the CLASSIC system [BMPSR91], and later it was expanded to an ontology tutorial[NM01].[13] Tableau-based reasoners remained unable to classify it until the optimizations for *nominals* were introduced in[PCS06], and now adopted in most reasoners that handle the *nominals* constructors. In order to be suitable for testing with HARD, an extract of the WINE ontology is used where only the expressivity of the DL $\mathcal{ALCHOQ}$ is used. The extracted TBox $\mathcal{T}$ is shown in Figure 59 where *nominals* appear using the *oneOf* constructor to define WineColor, WineBody, etc, or using the *hasValue* constructor to define the concepts representing the different types of wines.

The set of *nominals* referenced in concept descriptions consists of

$$N_o = \{red, rose, white, full, light, medium, delicate, moderate, strong, dry, offdry, sweet\} \quad (32)$$

and the set of role names used in number restrictions consists of

$$N_R = \{hasWineDescriptor, hasBody, hasColor, hasFlavor, hasSugar\} \quad (33)$$

The hierarchy between roles within the RBox $\mathcal{R}$ is depicted in Figure 60, which shows the initial role hierarchy of the WINE ontology in Figure 60a. This hierarchy is maintained by the reasoners considered except for HARD which extends it to the

---

[13]See more details about the WINE ontology and its usage at `http://www.w3.org/TR/2004/REC-owl-guide-20040210/#Usage`.

hierarchy shown in Figure 60b[14], after the preprocessing algorithm is executed.

Wine	$\sqsubseteq = 1hasBody.WineBody \sqcap = 1hasColor.WineColor$
	$\sqcap = 1hasFlavor.WineFlavor \sqcap = 1hasSugar.WineSugar$
DessertWine	$\equiv Wine \sqcap \forall hasSugar.\{offdry, sweet\}$
LateHarvest	$\sqsubseteq Wine \sqcap \forall hasFlavor.\{moderate, strong\} \sqcap \forall hasSugar.\{sweet\}$
IceWine	$\equiv DessertWine \sqcap LateHarvest \sqcap \exists hasColor.\{white\}$
IceWine	$\sqsubseteq \forall hasBody.\{full, medium\} \sqcap \forall hasFlavor.\{moderate, strong\}$
WineDescriptor	$\equiv WineColor \sqcup WineTaste$
WineTaste	$\sqsubseteq WineDescriptor$
WineColor	$\equiv \{red, rose, white\}$
WineBody	$\equiv \{full, light, medium\}$
WineFlavor	$\equiv \{delicate, moderate, strong\}$
WineSugar	$\equiv \{dry, offdry, sweet\}$

Figure 59: TBox axioms in the WINE ontology.

(a) Role hierarchy for the WINE ontology. (b) Role hierarchy for the WINE ontology after applying the preprocessing algorithm.
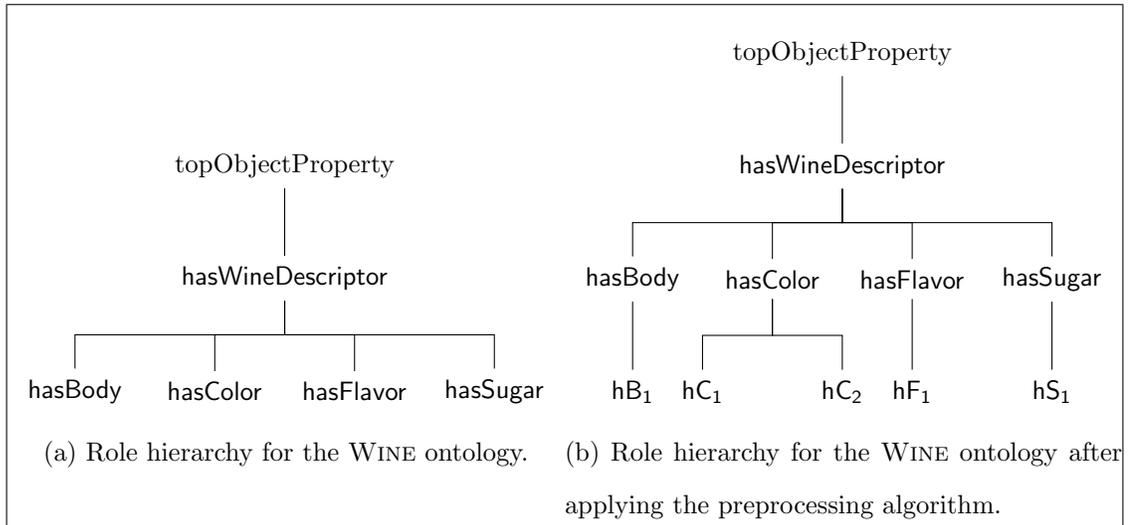
Figure 60: Role hierarchy within the RBox $\mathcal{R}$ for the WINE ontology.

---

[14]The extended hierarchy is maintained during test execution only.

Test case	Fact++	HARD	Hermit	Pellet
WINE-$C_{IceWine}$	63	566	219	336

Table 7: Runtimes in milliseconds with the test cases for the WINE ontology.

The satisfiability of the concept IceWine is tested using KB consistency tests where the TBox $\mathcal{T}$ includes $\top \sqsubseteq \neg\{a\} \sqcup$ IceWine with a a freshly introduced nominal. The concept IceWine is satisfiable and Figure 83, of Appendix C.2.1, shows a CCG for the test case WINE-$C_{IceWine}$. The run-times of the different reasoners for deciding the satisfiability of this concept are shown in Table 7.

Although the satisfiability of IceWine can be considered a simple test case, HARD's performance is not the best, and this is due to the non-determinism in selecting an initial distribution of *nominals* (*ch*-Rule). For example, the *nominals* used to describe the concept WineSugar must be distributed such that the restrictions imposed by the definition of Wine are satisfied. This distribution of *nominals* must also satisfy $\forall$hasSugar.$\{$offdry, sweet$\}$ because IceWine is also a DessertWine, and $\forall$hasSugar.$\{$sweet$\}$ because IceWine is a LateHarvest. In a sense, although dry, offdry, and sweet are *told nominals* for hasSugar, only sweet must be used as a hasSugar-filler. Since HARD does not implement any special processing of $\forall$hasSugar restrictions, an initial distribution of *nominals* does not necessarily consider sweet as the only hasSugar-filler. The performance of HARD is affected by how quickly a distribution for *nominals* is found such that sweet is the only filler for hasSugar. Similarly, although full, light, and medium are all *told nominals* for hasBody, only one of full, medium is allowed for iceWine and there can be only one WineBody for an instance of Wine. Although HARD is not the fastest in deciding the satisfiability of IceWine, HARD's performance can be improved by using some heuristics to process the $\forall$ restrictions and placing some ordering/priorities based on which *nominals* are distributed as role fillers.

7.2.2.3 The Koala ontology

Male	≡	∃hasGender.{male}
Female	≡	∃hasGender.{female}
KoalaWithPhD	⊑	∃hasDegree.{PhD}

Figure 61: TBox axioms using the *hasValue nominals* constructor.

The Koala ontology is a simple ontology about marsupials and humans. It is designed using the DL $\mathcal{ALCON}_{(D)}$, and is part of the Protégé ontology library.[15] *Nominals* are referenced using the *hasValue* constructor as shown in Figure 61.

In order to be suitable for HARD, expressions relying on the use of *concrete datatypes* are removed. For example, axiom (34), which uses the datatype property isHardWorking, is removed.

$$\text{Koala} \sqsubseteq \text{isHardWorking.false} \tag{34}$$

Also, expressions of the form $(\bowtie\ nR.\top \sqcap \forall R.C)$ are replaced with $(\bowtie\ nR.C)$. For example, the TBox axiom (35) is replaced with axiom (36). This is done because $(\bowtie\ nR.\top \sqcap \forall R.C)$ is based on the unqualified *number restrictions* constructor $(\mathcal{N})$, which is not interesting for HARD, because we want to test QCRs $\mathcal{Q}$ even though the two expressions may not admit identical models.

$$\text{Animal} \sqsubseteq\ \geq 1\,\text{hasHabit} \sqcap \forall \text{hasHabitat.Habitat} \tag{35}$$

$$\text{Animal} \sqsubseteq\ \geq 1\,\text{hasHabitat.Habitat} \tag{36}$$

The concepts Gender and Degree are poorly defined using ABox assertions. For example, the individuals male, and female are used in ABox assertions such that these

---

[15] http://protege.stanford.edu/plugins/owl/owl-library.

individuals are instances of the concept Gender; (male: Gender), and (female: Gender). However, male and female are the only allowed instances for Gender, and this should be defined in the TBox. Such poor representation of concepts is replaced by definitions of the concepts Gender and Degree in the TBox using the *oneOf* constructor with the *nominals* male, female, BA, BS, MA, PhD as follows:

$$\text{Gender} \equiv \{\text{male}, \text{female}\} \tag{37}$$

$$\text{Degree} \equiv \{\text{BA}, \text{BS}, \text{MA}, \text{PhD}\} \tag{38}$$

The final KOALA TBox includes the axioms shown in Figure 62 and has the expressivity of the $\mathcal{ALCOQ}$.

KoalaWithPhD	$\equiv$	Koala $\sqcap$ $\exists$hasDegree.{PhD}
Koala	$\sqsubseteq$	Marsupials $\sqcap$ $\exists$hasHabitat.Forest
Marsupials	$\sqsubseteq$	Animal
Animal	$\sqsubseteq$	≥ 1 hasHabitat.Habitat $\sqcap$ $= 1$ hasGender.Gender
MaleStudentWith3Daughters	$\sqsubseteq$	$\forall$hasGender.Male $\sqcap$ $= 3$ hasChildren.Female
MaleStudentWith3Daughters	$\sqsubseteq$	Student
Student	$\sqsubseteq$	Person $\sqcap$ $\exists$ hasHabitat.University
Person	$\sqsubseteq$	Animal
Female	$\sqsubseteq$	$\exists$hasGender.{female}
Male	$\sqsubseteq$	$\exists$hasGender.{male}
Gender	$\equiv$	{male, female}
MaleStudentWithnDaughters	$\sqsubseteq$	hasGender.Male $\sqcap$ $= n$ hasChildren.Female
MaleStudentWithnDaughters	$\sqsubseteq$	Student

Figure 62: TBox axioms in the adapted KOALA ontology. The expression $= nR.C$ abbreviates $\geq nR.C \sqcap \leq nR.C$.

(a) Initial role hierarchy for the KOALA ontology.

(b) Extended role hierarchy for the KOALA ontology after applying the preprocessing algorithm.
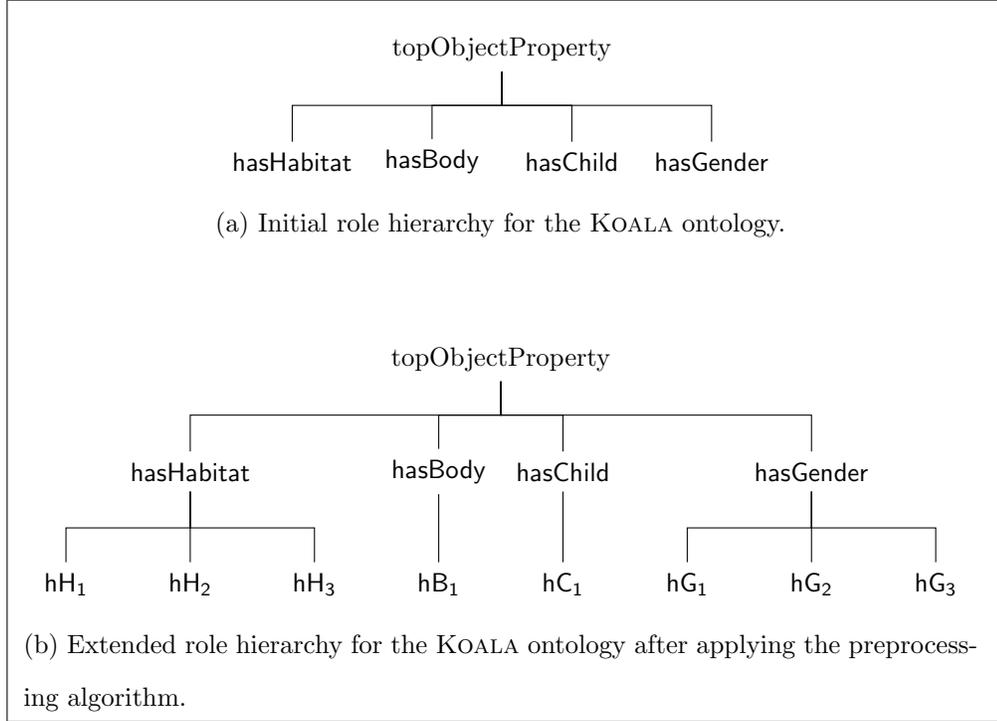
Figure 63: Role hierarchy within the RBox $\mathcal{R}$ for the Koala ontology.

The set of *nominals* referenced in concept descriptions consists of

$$\mathsf{N_o} = \{\mathsf{BA}, \mathsf{BS}, \mathsf{MA}, \mathsf{PhD}, \mathsf{male}, \mathsf{female}\} \tag{39}$$

and the set of role names used in number restrictions consists of

$$\mathsf{N_R} = \{\mathsf{hasChildren}, \mathsf{hasDegree}, \mathsf{hasGender}, \mathsf{hasHabitat}\} \tag{40}$$

such that the hierarchy between the roles is shown in Figure 63a. HARD extends this hierarchy after applying Algorithm 4.1.1 into the one shown in Figure 63b.

The concept MaleStudentWith3Daughters implicitly uses *nominals* through the concepts Male and Female. The satisfiability of KoalaWithPhD, MaleStudentWith3Daughters, and (KoalaWithPhD $\sqcap$ MaleStudentWith3Daughters) is tested using KB consistency tests where the TBox $\mathcal{T}$ includes $\top \sqsubseteq \neg\{\mathsf{a}\} \sqcup \mathsf{C}$.

Test case	Fact++	HARD	Hermit	Pellet
KOALA-C_1	55	168	250	382
KOALA-$C_2(n = 3)$	51	332	238	463
KOALA-$C_2(n = 10)$	50	309	39324	669
KOALA-$C_{1-2}(n = 3)$	59	426	258	410
KOALA-$C_{1-2}(n = 10)$	52	438	39672	535

Table 8: Runtimes in milliseconds with the test cases for the KOALA ontology.

The test case KOALA-C_1 refers to the KB consistency test where C refers to the concept KoalaWithPhD. KOALA-$C_2(n = 3)$, and KOALA-$C_{1-2}(n = 3)$ refer to the KB consistency tests where C refers to the concepts MaleStudentWithnDaughters and (KoalaWithPhD ⊓ MaleStudentWithnDaughters) with $n = 3$ respectively. Similarly KOALA-$C_2(n = 10)$, and KOALA-$C_{1-2}(n = 10)$ refer to the KB consistency tests where C refers to the concepts MaleStudentWithnDaughters and (KoalaWithPhD ⊓ MaleStudentWithnDaughters) with $n = 10$. Figure 85, in Appendix C.2.2, shows a CCG for KOALA-C_1, Figure 87 shows a CCG for KOALA-$C_2(n)$, and Figure 88 shows a CCG for KOALA-$C_{1-2}(n)$. The existence of these complete and clash free CCGs shows that the concepts being tested are all satisfiable. The run-times for the corresponding test cases are shown in Table 8.

Although the tests considered are simple cases where the concepts are satisfiable, the results show that Pellet is the worst performing reasoner in all cases, even though Hermit is terrible in handling cases with large values of n. HARD's performance is, as expected, affected by the number of QCRs to satisfy (HARD is slightly slower with KOALA-C_{1-2}), but insensitive to n. Fact++ can routinely process these test cases.

7.2.2.4 The Countries ontology

The Countries[16] ontology represents the ISO 3166 Code[17] list of countries using the DL $\mathcal{ALCIN}_{(D)}$. The fact that every country has exactly one English name is represented using axiom (41), where nameEnglish is a functional *datatype role* whose range is a `string`.

$$\text{Country} \sqsubseteq = 1\text{nameEnglish} \tag{41}$$

Since nameEnglish is functional, each country can have exactly one English name. However, the fact that a country's English name must be an English word is missing because the range for nameEnglish is any `string`. Also, the fact that a country's English name must be the same name used in every model of an English country is lost. For example, if we have an individual country Canada as an instance of the concept Country, one could have $\langle$Canada,"Kanata"$\rangle$: nameEnglish[18] in one model, and $\langle$Canada,"Canada"$\rangle$: nameEnglish in some other model.

The Countries ontology is adapted so that the fact that every country has exactly one English name is represented using axiom (42) where hasEnglishName is an object property whose range is the concept EnglishCountryName defined in (43) using the *oneOf* constructor as an enumeration of *nominals* representing existing English country names[19].

$$\text{Country} \sqsubseteq = 1\text{hasEnglishName.EnglishCountryName} \tag{42}$$

$$\text{LongEnglishCountryName} \equiv \{\text{Afghanistan, AlandIslands}, \dots, \text{Zimbabwe}\} \tag{43}$$

$$\text{LongEnglishCountryName} \sqsubseteq \text{EnglishCountryName} \tag{44}$$

---

[16]`http://www.bpiresearch.com/BPMO/2004/03/03/cdl/Countries`.

[17]`http://www.iso.org/iso/English_country_names_and_code_elements`.

[18]*Kanata* is the First Nations word where the name of Canada originated.

[19]In total, there exists 248 official short names for countries as listed at `http://www.iso.org/iso/english_country_names_and_code_elements`.

The fact that a country's English name must be the same one used in every model is represented using axioms like Canada $\sqsubseteq \exists$hasEnglishName.{Canada} where {Canada} is a nominal representing the EnglishCountryName used for Canada.

The European Union Countries Example The EU example which was discussed in Section 4.7.6 is integrated into the COUNTRIES ontology. Recall that the EU example represents the European Union (EU) member states as an enumeration of 27 *nominals*, each representing a member state. The resulting TBox is shown in Figure 64, where an interaction between *nominals* and role fillers is required to satisfy the concept EuropeanUnion. In a first variant of this example, the representation of EuropeanUnion does not rely on concepts from the COUNTRIES ontology. This simple representation is referred as COUNTRIES-$\mathsf{C_{simple-EU}}$.

A more complex representation of the EU example is considered where the concept Country is used within the definition of EuropeanUnion, and an enumeration of the English names for the member states is used to define ShortEnglishCountryName. For ease of presentation and better clarity, we use the short english name for a country when referring to an EU member state. The resulting TBox is shown in Figure 66 and this test case is referred as COUNTRIES-$\mathsf{C_{complex-EU}}$. Another variant of the COUNTRIES-$\mathsf{C_{*-EU}}$ ontology is considered where only the six inner member state of the EU are represented. The simple representation of the InnerEuropeanUnion is shown in Figure 65 and a more complicated one is shown in Figure 67.

Unlike the original COUNTRIES ontology, the COUNTRIES-$\mathsf{C_{*-*EU}}$ ontologies use the DL $\mathcal{ALCOQ}$. The satisfiability of (Inner)EuropeanUnion is tested using KB consistency tests where $(\top \sqsubseteq \neg\{\mathsf{a}\} \sqcup (\mathsf{Inner})\mathsf{EuropeanUnion})$ is included in the TBox $\mathcal{T}$. Notice that (Inner)EuropeanUnion is satisfiable in the case where $(\mathsf{n} \leq 6)$ $\mathsf{n} \leq 27$, and unsatisfiable otherwise. The run-times for the corresponding test cases are shown in Table 9.

228

State	$\sqsubseteq$ Country
EUMemberState	$\equiv \{$Austria, Belgium, . . . , UnitedKingdom$\}$
EuropeanUnion	$\sqsubseteq \forall$hasMember.EUMemberState
EuropeanUnion	$\sqsubseteq \geq$ nhasMember.State

Figure 64: TBox axioms in the COUNTRIES-C$_{\mathsf{simple-EU}}$ ontology.

State	$\sqsubseteq$ Country
EUInnerMemberState	$\equiv \{$Italy, Netherlands, Belgium, France, Germany, Luxembourg$\}$
InnerEuropeanUnion	$\sqsubseteq \forall$hasMember.EUInnerMemberState
InnerEuropeanUnion	$\sqsubseteq \geq$ nhasMember.State

habi

Figure 65: TBox axioms in the COUNTRIES-C$_{\mathsf{simple-IEU}}$ ontology.

The complexity of the test cases for COUNTRIES-C$_{\mathsf{*-simple-*EU}}$ is due to the non-determinism in merging the domain elements that are hasMember-fillers of a with the *nominals* enumerated in the definition of InnerEuropeanUnion. A standard tableau algorithm decides the satisfiability of (Inner)EuropeanUnion by creating n distinct anonymous domain elements related to a, the instance of (Inner)EuropeanUnion, via the hasMember role. The $\forall$hasMember.EU(Inner)MemberState forces these elements to become members of the EU(Inner)MemberState concepts. This means that each one of these elements must be identified with one of the *nominals* enumerated in the definition of EU(Inner)MemberState. The (un)satisfiability is discovered after all possibilities of merging the n elements with the (6)27 *nominals* are exhausted. Notice that Pellet's performance is the worst in handling this complexity especially in the case of unsatisfiability. Fact++ and Hermit handle satisfiable cases better, however their performance deteriorates as the number of *nominals* increases with unsatisfiable cases. HARD's performance is affected by the increase in the number of *nominals*

229

with satisfiable cases. This performance degradation is expected since HARD needs to consider more partitions as the number of *nominals* increases.

State	$\sqsubseteq$ Country
Country	$\sqsubseteq=$ 1hasEnglishName.ShortEnglishCountryName
Country	$\sqsubseteq=$ 1locatedIn.Continent
Continent	$\equiv \{Europe\}$
ShortEnglishCountryName	$\equiv \{AT, BE, \ldots, UK\}$
EUMemberState	$\equiv \{AT, BE, \ldots, UK\}$
EuropeanUnion	$\sqsubseteq \forall$hasMember.EUMemberState
EuropeanUnion	$\sqsubseteq \geq$ nhasMember.(Country $\sqcap \exists$locatedIn.$\{Europe\}$)

Figure 66: TBox axioms in the COUNTRIES-C$_{complex-EU}$ ontology.

State	$\sqsubseteq$ Country
Country	$\sqsubseteq=$ 1hasEnglishName.ShortEnglishCountryName
Country	$\sqsubseteq=$ 1locatedIn.Continent
Continent	$\equiv \{Europe\}$
ShortEnglishCountryName	$\equiv \{IT, NL, BE, FR, DE, LU\}$
EUInnerMemberState	$\equiv \{IT, NL, BE, FR, DE, LU\}$
InnerEuropeanUnion	$\sqsubseteq \forall$hasMember.EUInnerMemberState
InnerEuropeanUnion	$\sqsubseteq \geq$ nhasMember.(Country $\sqcap \exists$locatedIn.$\{Europe\}$

Figure 67: TBox axioms in the COUNTRIES-C$_{complex-IEU}$ ontology.

However, the fact that (7)30 elements can never be distributed over (6)27 *nominals* is quickly discovered by HARD which maintains a stable performance with unsatisfiable cases. We study how the number of *nominals* and the numbers used in the QCRs affects the performance of these reasoners in Section 7.2.2.6.

230

Test case	Fact++	HARD	Hermit	Pellet
COUNTRIES-$C_{\mathsf{SAT-simple-IEU}(n=6)}$	47	219	188	625
COUNTRIES-$C_{\mathsf{UnSAT-simple-IEU}(n=7)}$	94	110	703	8234
COUNTRIES-$C_{\mathsf{SAT-simple-EU}(n=27)}$	62	343	312	19641
COUNTRIES-$C_{\mathsf{UnSAT-simple-EU}(n=30)}$	TO	109	TO	TO
COUNTRIES-$C_{\mathsf{SAT-complex-IEU}(n=6)}$	32	146781	234	703
COUNTRIES-$C_{\mathsf{UnSAT-complex-IEU}(n=7)}$	94	142	1078	19952
COUNTRIES-$C_{\mathsf{SAT-complex-EU}(n=27)}$	63	TO	344	29546
COUNTRIES-$C_{\mathsf{UnSAT-complex-EU}(n=30)}$	TO	125	TO	TO

Table 9: Runtimes in milliseconds with the test cases for the COUNTRIES ontology.

The complexity of the test cases for COUNTRIES-$C_{*-\mathsf{complex}-*\mathsf{EU}}$ is due to the fact that the *nominals* interacting with the hasMember-fillers also interact with the hasEnglishName-fillers at different levels of the completion graph. Fact++ does not seem to be affected by this complexity as it maintains a similar performance as with COUNTRIES-$C_{*-\mathsf{simple}-*\mathsf{EU}}$ cases. Pellet's performance and Hermit's performance consistently degrade as the number of *nominals* increases and it is worst with unsatisfiable cases. HARD's performance is directly affected by this complexity, especially if an initial distribution of *nominals* does not take into consideration their interaction with hasEnglishName-fillers (case when the hasEnglishName role is not activated) and assigns these *nominals* to hasMember partitions. As soon as the hasEnglishName role is activated the satisfiability of $= 1$hasEnglishName is not possible, and the distribution of *nominals* is no longer valid. The algorithm considers a different distribution for each nominal, which means that the choice points for the *ch*-Rule with nominal variables are exhausted until a distribution is found which takes into consideration the interaction between the *nominals*, hasEnglishName-fillers and hasMember-fillers.

If the lazy nominal generation is disabled, HARD can solve it in 1546 milliseconds. In the case of Countries-$C_{SAT-complex-EU}$ a blow up in the number of partitions that needs to be considered results in a `java.lang.NumberFormatException` which is an implementation problem and can be enhanced by a smarter way of representing variables and indexes for partitions. On the other hand, the numerical unsatisfiablity remains trivial.

The Canadian Parliament   An interesting extension to the Countries ontologies is considered by representing the members of the Canadian parliament based on their distribution over Canadian provinces.[20] The resulting TBox is shown in Figure 68.

None of the available reasoners can decide the satisfiability of the concept CanadianParliament. In the case of HARD, a `java.lang.NumberFormatException` is thrown due to the blow up in the size of the decomposition set. Different TBox extractions are considered:

- Only members of the Canadian parliament within the provinces across atlantic Canada are considered. The corresponding TBox is shown in Figure 69 and the test case is referred as Countries-$C_{Parliament-atlantic}$.

- Only members within a province having the lowest number of seats are considered. The corresponding TBox is shown in Figure 70, and the test case is referred to as Countries-$C_{Parliament-PE}$.

- Only members within a province having the highest number of seats are considered. The corresponding TBox is shown in Figure 71, and the test case is referred to as Countries-$C_{Parliament-ON}$.

---

[20]As described in the definition of House of Commons of Canada at `http://en.wikipedia.org/wiki/House_of_Commons_of_Canada#Members_and_electoral_districts`.

CanadianParliament	$\sqsubseteq\, \geq$ 279hasMember.(Person $\sqcap\, \geq$ 1livesIn.CanadianProvince)
CanadianParliament	$\sqsubseteq\, =$ 305hasMember.(Person $\sqcap\, \geq$ 1livesIn.CanadianProvince)
CanadianParliament	$\sqsubseteq\, \geq$ 95hasMember.(Person $\sqcap\, \geq$ 1livesIn.{ON})
CanadianParliament	$\sqsubseteq\, =$ 106hasMember.(Person $\sqcap\, \geq$ 1livesIn.{ON})
CanadianParliament	$\sqsubseteq\, =$ 75hasMember.(Person $\sqcap\, \geq$ 1livesIn.{QC})
CanadianParliament	$\sqsubseteq\, \geq$ 28hasMember.(Person $\sqcap\, \geq$ 1livesIn.{BC})
CanadianParliament	$\sqsubseteq\, =$ 36hasMember.(Person $\sqcap\, \geq$ 1livesIn.{BC})
CanadianParliament	$\sqsubseteq\, \geq$ 21hasMember.(Person $\sqcap\, \geq$ 1livesIn.{AB})
CanadianParliament	$\sqsubseteq\, =$ 28hasMember.(Person $\sqcap\, \geq$ 1livesIn.{AB})
CanadianParliament	$\sqsubseteq\, =$ 14hasMember.(Person $\sqcap\, \geq$ 1livesIn.{MT})
CanadianParliament	$\sqsubseteq\, =$ 14hasMember.(Person $\sqcap\, \geq$ 1livesIn.{SK})
CanadianParliament	$\sqsubseteq\, =$ 11hasMember.(Person $\sqcap\, \geq$ 1livesIn.{NS})
CanadianParliament	$\sqsubseteq\, =$ 10hasMember.(Person $\sqcap\, \geq$ 1livesIn.{NB})
CanadianParliament	$\sqsubseteq\, =$ 7hasMember.(Person $\sqcap\, \geq$ 1livesIn.{NL})
CanadianParliament	$\sqsubseteq\, =$ 4hasMember.(Person $\sqcap\, \geq$ 1livesIn.{PE})
CanadianProvince	$\equiv$ {ON, AB, QC, BC, SK, MB, NL, NS, NB, PE}

Figure 68: TBox axioms in the Countries-C$_{\mathsf{Parliament-full}}$ ontology

CanadianParliament	$\sqsubseteq\, =$ 305hasMember.(Person $\sqcap\, \geq$ 1livesIn.CanadianProvince)
CanadianParliament	$\sqsubseteq\, \geq$ 11hasMember.(Person $\sqcap\, \geq$ 1livesIn.{NS})
CanadianParliament	$\sqsubseteq\, \geq$ 10hasMember.(Person $\sqcap\, \geq$ 1livesIn.{NB})
CanadianParliament	$\sqsubseteq\, \geq$ 7hasMember.(Person $\sqcap\, \geq$ 1livesIn.{NL})
CanadianParliament	$\sqsubseteq\, \geq$ 4hasMember.(Person $\sqcap\, \geq$ 1livesIn.{PE})
CanadianProvince	$\equiv$ {ON, AB, QC, BC, SK, MB, NL, NS, NB, PE}

Figure 69: TBox axioms in the Countries-C$_{\mathsf{Parliament-atlantic}}$ ontology.

CanadianParliament	$\sqsubseteq$ = 305hasMember.(Person $\sqcap$ $\geq$ 1livesIn.CanadianProvince)
CanadianParliament	$\sqsubseteq$ $\geq$ 4hasMember.(Person $\sqcap$ $\geq$ 1livesIn.{PE})
CanadianProvince	$\equiv$ {ON, AB, QC, BC, SK, MB, NL, NS, NB, PE}

Figure 70: TBox axioms in the COUNTRIES-$C_{\mathsf{Parliament-PEI}}$ ontology

CanadianParliament	$\sqsubseteq$ = 305hasMember.(Person $\sqcap$ $\geq$ 1livesIn.CanadianProvince)
CanadianParliament	$\sqsubseteq$ $\geq$ 106hasMember.(Person $\sqcap$ $\geq$ 1livesIn.{ON})
CanadianProvince	$\equiv$ {ON, AB, QC, BC, SK, MB, NL, NS, NB, PE}

Figure 71: TBox axioms in the COUNTRIES-$C_{\mathsf{SAT-Parliament-ON}}$ ontology

Test case	Fact++	HARD	Hermit	Pellet
COUNTRIES-$C_{\mathsf{Parliament-full}}$	ERR[20]	ERR[20]	TO	ERR[20]
COUNTRIES-$C_{\mathsf{Parliament-atlantic}}$	260	375	TO	TO
COUNTRIES-$C_{\mathsf{Parliament-ON}}$	203454	360	TO	TO
COUNTRIES-$C_{\mathsf{Parliament-PEI}}$	63	218	TO	TO

Table 10: Runtimes in milliseconds with the test cases for the COUNTRIES ontology including the Canadian Parliament representation.

The run-times for the corresponding test cases are shown in Table 10. Hermit and Pellet cannot solve any of the test cases within the time limit. Fact++'s performance degrades as the number used with $\geq$ nhasMember increases. Note however, that replacing $\geq n$ hasMember with $= n$ hasMember results in none of the tableau reasoners being able to solve even the smallest examples. This might be due to the fact that a $= nR.C$ is equivalent to $\geq nR.C \sqcap \leq nR.C$, and reasoning with more $\leq nR.C$ is less efficient due to the applicability of the non-deterministic *choose*-Rule.

---

[20]Fact++ and Pellet run out of memory for the JAVA heap space, and HARD runs into a JAVA number format exception due to the blow up of variables.

7.2.2.5 The TIME ontology

$$
\begin{aligned}
\mathsf{DateTimeDescription} \quad &\sqsubseteq\,\leq 1\mathsf{dayOfWeek}\sqcap\forall\mathsf{dayOfWeek.DayOfWeek} \\
&\sqcap\,\leq 1\mathsf{timeZone}\sqcap\forall\mathsf{timeZone.TimeZone} \\
&\sqcap\,=1\mathsf{unitType}\sqcap\forall\mathsf{unitType.TemporalUnit} \\
\mathsf{DayOfWeek} \quad &\equiv\{\mathsf{Sunday,Monday,\ldots,Saturday}\} \\
\mathsf{TemporalUnit} \quad &\equiv\{\mathsf{unitDay,unitHour,\ldots,unitYear}\}
\end{aligned}
$$

Figure 72: Some TBox axioms in the TIME ontology

The TIME ontology is part of the ontologies within the Semantic Web for Earth and Environmental Terminology (SWEET) project.[21] The ontologies can be downloaded from `http://sweet.jpl.nasa.gov/sweet`. The Time ontology relies on the DL $\mathcal{SHOIN}_{(D)}$, a detailed description is available at `http://www.w3.org/TR/owl-time/` and the OWL file can be downloaded from `http://www.w3.org/2006/time`. A subset of the ontology has been converted to the DL expressivity of $\mathcal{ALCON}$ such that *concrete datatype* roles were discarded as well as expressions relying on them. Also, roles that are not referenced within concept expressions are discarded. The resulting TBox contains the axioms shown in Figure 72. The set of *nominals* referenced in concept descriptions consists of

$$
\begin{aligned}
\mathsf{N_o}=\quad &\{\mathsf{Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, unitDay} \\
&\mathsf{unitHour, unitMinute, unitMonth, unitSecond, unitWeek, unitYear}\} \quad (45)
\end{aligned}
$$

and the set of role names used in number restrictions consists of

$$
\mathsf{N_R}=\{\mathsf{unitType, dayOfWeek, timeZone}\} \quad (46)
$$

---

[21]The SWEET ontology provides an upper-level ontology for Earth system science. The SWEET ontologies include several thousand terms, spanning a broad extent of Earth system science and related concepts (such as data characteristics) using the OWL language.

Test case	Fact++	HARD	Hermit	Pellet
TIME-C$_{\mathsf{SAT-DTD}}$	59	195	223	328

Table 11: Runtimes in milliseconds with the test cases for the TIME ontology.

used in the definition of DateTimeDescription which consists of number restrictions with role fillers that interact with the *nominals* used in the definitions of DayOfWeek and TemporalUnit. The test case TIME-C$_{\mathsf{SAT-DTD}}$ refers to the TBox consistency tests for the satisfiability of the concept DateTimeDescription which is satisfiable.

7.2.2.6 Synthetic Test Cases

The test cases used in the previous section make use of the *oneOf* constructor and/or the *hasValue* constructor. The test cases focus more on a real world occurrence or use of *nominals* and their effect on reasoning performance when they interact with role fillers. Most of those test cases are simple and do not pose significant challenge to known reasoning algorithms. In this section, some synthetic test cases are designed to test the scalability of ReAl DL based on the following four criteria: number of *nominals* used, the depth of the role hierarchy, the use of cycles within concept descriptions and the nesting in the use of QCRs. The purpose of these test cases is to evaluate the effect of different DL expressivity features used in concept descriptions, where an interaction between *nominals* and QCRs is expected, on the reasoning performance.

- *The number of nominals.* The reasoning performance is evaluated as the number of *nominals* (size of N_{o}) used in the TBox increases. The purpose of these tests is to stress HARD's reasoning.

- *The depth of the role hierarchy.* The reasoning performance is evaluated as the hierarchy between the roles used in QCRs becomes deeper. The purpose of

236

these tests is to evaluate how HARD handles decomposition of roles within a hierarchy of a deeper level.

- *The use of cycles in concept descriptions.* Cyclic descriptions require the use of blocking strategies in order for reasoning to terminate. The purpose of these test cases, is to evaluate how HARD handles cycles through the re-use of proxy nodes compared to other reasoners that need to be equipped with blocking strategies to ensure termination.

- *The use of nested nominals and QCRs within concept descriptions.* The reasoning performance is evaluated as the QCRs used become nested within concept descriptions. The purpose of these test cases is to evaluate how the interaction between *nominals* and/or QCRs at different levels of the completion graph affects performance.

Every test case is evaluated using two variants: a satisfiable case, and an unsatisfiable one. In the following, the cases which are satisfiable are referred to as $C_{SAT-*-*}$, and those that are unsatisfiable are referred to as $C_{UnSAT-*-*}$.

Effect of increasing the number of nominals The effect of increasing the number of *nominals* within a concept description, enforced as a qualification on role fillers, is shown using a set of synthetic ontologies, where the TBox includes the concept $C_{nominals-\mathcal{ALCOQ}}$ defined using the DL $\mathcal{ALCOQ}$ as follows:

$$C_{nominals-\mathcal{ALCOQ}} \sqsubseteq \geq jR.E$$
$$E \equiv \{o_1, o_2, \ldots, o_i\}$$
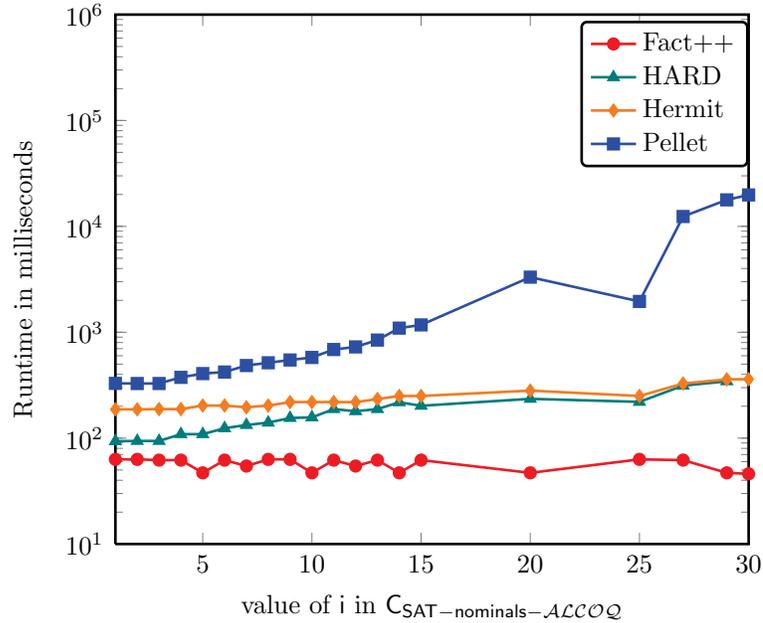
Figure 73: Effect of increasing the number of nominals with the numbers used in QCRs in a *satisfiable* concept expression $C_{SAT-nominals-\mathcal{ALCOQ}}$ with $j = i$.
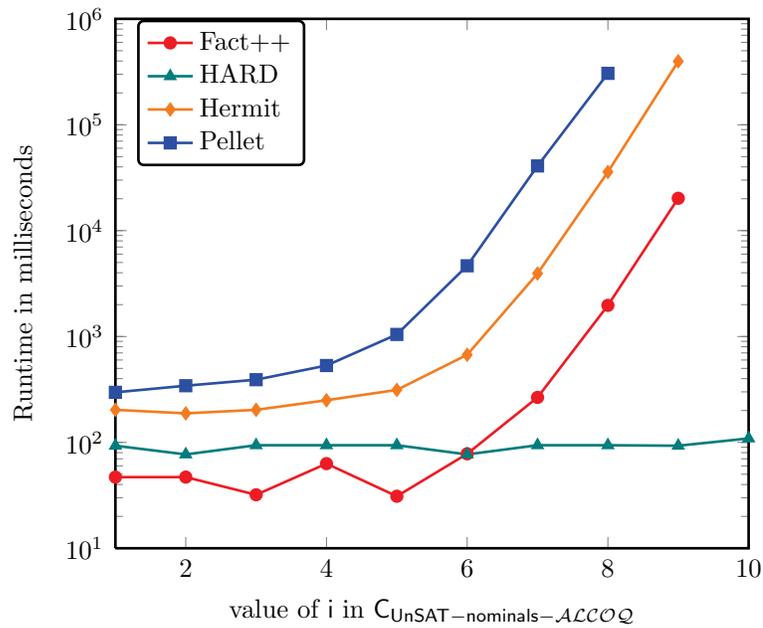
Figure 74: Effect of increasing the number of *nominals* with the numbers used in QCRs in an *unsatisfiable* concept expression $C_{UnSAT-nominals-\mathcal{ALCOQ}}$ with $j = i + 1$.

In the case of $C_{SAT-nominals-\mathcal{ALCOQ}}$, the value of i is such that $i \geq j$ in order for $C_{nominals-\mathcal{ALCOQ}}$ to be satisfiable. In the case of $C_{UnSAT-nominals-ALCOQ}$, the value of i is such that j=i+1. In each test, the value of i is increased and the performance results are shown in Figure 73 for $C_{SAT-nominals-\mathcal{ALCOQ}}$ cases, and Figure 74 for $C_{UnSAT-nominals-\mathcal{ALCOQ}}$ cases. In a second set of test cases, we fix i to 5 and replace j with $10k + 1$. Since j is always greater than i, $C_{nominals-\mathcal{ALCOQ}}$ becomes unsatisfiable and we refer to this set of test cases as $C_{UnSAT-fixed-nominals-\mathcal{ALCOQ}}$.

Running the cases with $C_{nominals-\mathcal{ALCOQ}}$ have shown that the interaction between *nominals* and QCRs has a direct effect on reasoning performance as the number of *nominals* increases in a concept description relying on the *oneOf* constructor. If the role fillers exceed the number of *nominals* allowed, the unsatisfiability of the concept description ($C_{UnSAT-nominals-\mathcal{ALCOQ}}$) cannot be solved without the algebraic approach; HARD is the only reasoner which decides the unsatisfiability, within the time limit, as the number of *nominals* increases. Figure 74 shows that the other reasoners compete in deciding unsatisfiability. Even though the different optimizations implemented make some reasoners faster than others, when it comes to scalability, the performance degradation seems to be a function of the same variable due to a common complexity bound that is reached. The complexity is due to the increase of both the number of *nominals* and the numbers used in QCRs. For instance as shown in Figure 75, some reasoners (Fact++) scale better if the number of *nominals* is fixed while increasing the value of the number used in QCRs. The test results with the cases where the number of *nominals* is fixed but the numbers used in QCRs is increased, case $C_{UnSAT-fixed-nominals-\mathcal{ALCOQ}}$. Therefore, we can conclude that it is the interaction and increase of both *nominals* and QCRs which is weakly handled by tableau reasoners.
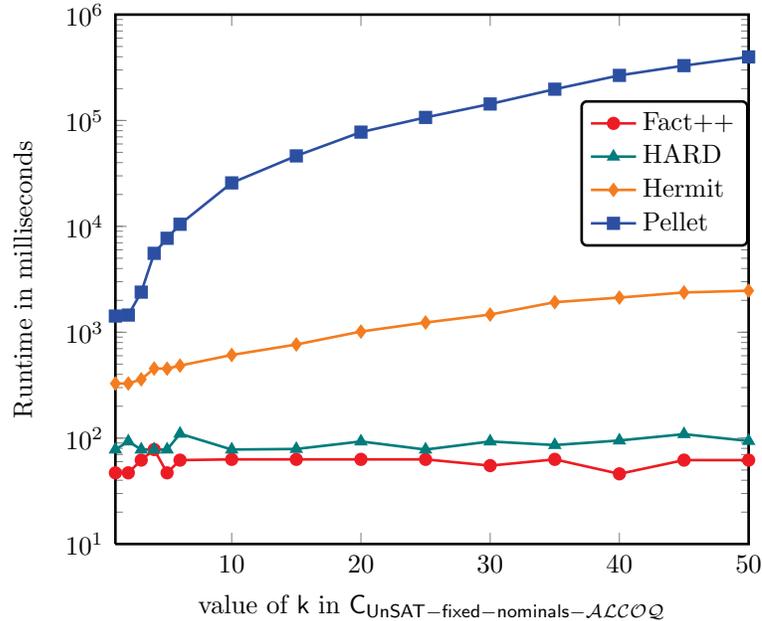
Figure 75: Effect of increasing the size of the numbers used in QCRs with unsatisfiable cases of $C_{\mathsf{UnSAT-fixed-nominals}-\mathcal{ALCOQ}}$ where the number of *nominals* is fixed, $i = 5$ and $j = 10k + 1$.

However, in the case when the number of role fillers does not exceed the number of *nominals* allowed, case with $C_{\mathsf{SAT-nominals}-\mathcal{ALCOQ}}$, most reasoners did not have a problem deciding satisfiability with an increased number of *nominals*, except for Pellet whose performance degrades as the number of *nominals* increases. Fact++ is stable, Hermit and HARD need slightly more time to process the increased number of *nominals*.

Effect of increasing the depth of the role hierarchy   The effect of the depth of the role hierarchy on the performance of reasoning is tested using the role hierarchy shown in Figure 76 where roles from different levels of the hierarchy are used in QCRs in the definition of the concept C for which the test cases are referred to as $C_{*-R-deep-\mathcal{ALCHQ}}$ when there is no nominal interaction and $C_{*-R-deep-\mathcal{ALCHOQ}}$ there is an interaction between *nominals* and role fillers. The interaction with *nominals* is enabled by adding the definition of D to the TBox axioms.

240

$$C_{R-deep-\mathcal{ALCHQ}} \sqsubseteq \geq nR_1.A \sqcap \geq nR_2.B \sqcap \geq nR_{1a}.A \sqcap \geq nR_{2a}.B \sqcap \leq mR.\top \sqcap \forall R.\top$$

$$C_{R-deep-\mathcal{ALCHOQ}} \sqsubseteq \geq nR_1.A \sqcap \geq nR_2.B \sqcap \geq nR_{1a}.A \sqcap \geq nR_{2a}.B \sqcap \leq mR.\top \sqcap \forall R.D$$

$$D \equiv \{o_1, o_2, \ldots, o_n\}$$
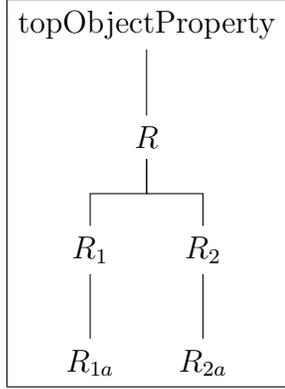
Figure 76: Role hierarchy within the RBox $\mathcal{R}$.

Test case	Fact++	HARD	Hermit	Pellet
$C_{SAT-R-deep-\mathcal{ALCHQ}}$	39	250	635	349
$C_{UnSAT-R-deep-\mathcal{ALCHQ}}$	39	266	7891	7329
$C_{SAT-R-deep-\mathcal{ALCHOQ}}$	47	375	875	344
$C_{UnSAT-R-deep-\mathcal{ALCHOQ}}$	47	126	7792	7214

Table 12: Runtimes in milliseconds with the test cases using a *deep* role hierarchy.

The performance results displayed in Table 12 for the cases with $C_{*-R-deep-*}$ show the effect of the *depth* of the role hierarchy in $\mathcal{R}$: even though the depth used is only minimal, Pellet's performance degrades with unsatisfiable cases and Hermit's performance degrades with satisfiable and unsatisfiable cases. On the other hand, Fact++ and HARD's performance are not affected by the depth of the role hierarchy.

Effect of increasing the nesting level of QCRs   The effect of using QCRs in nested expressions on the performance of reasoning is tested using the satisfiability of $C_{\mathsf{nested}-\mathcal{ALCHQ}}$. In this case the qualification on the role fillers of R_1 includes a QCRs.

$$C_{\mathsf{nested}-\mathcal{ALCHQ}} \quad \sqsubseteq\, \geq nR_1.(A \sqcap\, \geq nR_2.B) \sqcap\, \geq nR_{1a}.(A \sqcap\, \geq nR_{2a}.B) \sqcap\, \leq mR.\top \sqcap \forall R.\top$$

When the nominal interaction is enabled we refer to the test case as $C_{\mathsf{nested}-\mathcal{ALCHOQ}}$.

$$C_{\mathsf{nested}-\mathcal{ALCHOQ}} \quad \sqsubseteq\, \geq nR_1.(A \sqcap\, \geq nR_2.B) \sqcap\, \geq nR_{1a}.(A \sqcap\, \geq nR_{2a}.B) \sqcap\, \leq mR.\top \sqcap \forall R.D$$

$$D \quad \equiv \{o_1, o_2, \dots, o_n\}$$

The performance results displayed in Table 13 for the cases with $C_{*-\mathsf{nested}-*}$ show the effect of the *nesting* within concept descriptions: most reasoners can quickly decide these test cases. A slight performance degradation is noticed with HARD while deciding unsatisfiable cases when nested expression allow an interaction between *nominals* and role fillers at different levels. Such a degradation is somehow expected, especially if due to the interaction between *nominals* and role fillers, an initial distribution of *nominals* (due to the *ch*-Rule) leads to a clash, and backtracking is required until the interaction is taken into consideration when the *nominals* are initially distributed over partitions. Pellet's performance is unexpectedly the worst when deciding satisfiability cases. Appendix C Figure 89 shows a compressed completion graph for $C_{\mathsf{SAT}-\mathsf{nested}-\mathcal{ALCHOQ}}$.

Test case	Fact++	HARD	Hermit	Pellet
$C_{\mathsf{SAT}-\mathsf{nested}-\mathcal{ALCHQ}}$	31	219	258	375
$C_{\mathsf{UnSAT}-\mathsf{nested}-\mathcal{ALCHQ}}$	63	140	266	385
$C_{\mathsf{SAT}-\mathsf{nested}-\mathcal{ALCHOQ}}$	47	313	274	325411
$C_{\mathsf{UnSAT}-\mathsf{nested}-\mathcal{ALCHOQ}}$	47	516	266	381

Table 13: Runtimes in milliseconds with the test cases using *nesting* occurrences of QCRs within concept expressions.

Effect of using cyclic descriptions The effect of using cycles in concept descriptions on the performance of reasoning is tested using the satisfiability of $\mathsf{C}_{\text{cyclic}-\mathcal{ALCHQ}}$ when there is no nominal interaction and $\mathsf{C}_{\text{cyclic}-\mathcal{ALCHOQ}}$ where *nominals* interact through the definition of D.

$$\mathsf{C}_{\text{cyclic}-\mathcal{ALCHQ}} \sqsubseteq \geq n\mathsf{R}_1.\mathsf{C} \sqcap \geq n\mathsf{R}_2.\mathsf{C} \sqcap \geq n\mathsf{R}_{1a}.\mathsf{C} \sqcap \geq n\mathsf{R}_{2a}.\mathsf{C} \sqcap \leq m\mathsf{R}.\top \sqcap \forall\mathsf{R}.\top$$

$$\mathsf{C} \sqsubseteq \mathsf{C}_{\text{cyclic}-\mathcal{ALCHQ}}$$

$$\mathsf{C}_{\text{cyclic}-\mathcal{ALCHOQ}} \sqsubseteq \geq n\mathsf{R}_1.\mathsf{C_o} \sqcap \geq n\mathsf{R}_2.\mathsf{C_o} \sqcap \geq n\mathsf{R}_{1a}.\mathsf{C_o} \sqcap \geq n\mathsf{R}_{2a}.\mathsf{C_o} \sqcap \leq m\mathsf{R}.\top \sqcap \forall\mathsf{R}.\mathsf{D}$$

$$\mathsf{D} \equiv \{\mathsf{o}_1, \mathsf{o}_2, \ldots, \mathsf{o}_n\}$$

$$\mathsf{C_o} \sqsubseteq \mathsf{C}_{\text{cyclic}-\mathcal{ALCHOQ}}$$

The performance results displayed in Table 14 for the cases with $\mathsf{C}_{*-\text{cyclic}-*}$ show the effect of *cycles* within concept description. As expected, HARD does not need to implement any blocking strategies to ensure termination. The termination is guaranteed by the re-use of individuals. Even though Hermit is known to implement a sophisticated blocking strategy notice how it's performance degrades with $\mathsf{C}_{*-\text{cyclic}-*}$ with and without *nominals* interaction for satisfiable and unsatisfiable cases. Pellet on the other hand, needs considerably more time to decide unsatisfiable cases with cycles. Appendix C Figures 91 and 90 show examples of a CCG for $\mathsf{C}_{\text{SAT}-\text{cyclic}-\mathcal{ALCHQ}}$, it is easy to see how cycles are handled by the re-use of proxy nodes.

Test case	Fact++	HARD	Hermit	Pellet
$\mathsf{C}_{\text{SAT}-\text{cyclic}-\mathcal{ALCHQ}}$	47	202	594	359
$\mathsf{C}_{\text{UnSAT}-\text{cyclic}-\mathcal{ALCHQ}}$	63	249	6468	7345
$\mathsf{C}_{\text{SAT}-\text{cyclic}-\mathcal{ALCHOQ}}$	47	281	1297	374
$\mathsf{C}_{\text{UnSAT}-\text{cyclic}-\mathcal{ALCHOQ}}$	63	110	6453	6500

Table 14: Runtimes in milliseconds with the test cases using *cyclic* concept expressions.

7.3 Optimizations Effects

This section reports on run times for the previously discussed test cases where one or more optimizations are disabled. The purpose of running these tests is twofold: first, the effect of the proposed optimizations in Chapter 5 is evaluated, second, key optimizations are highlighted. Optimizations effect are measured by calculating their speed up factor S which is defined as $S(O) = \frac{t_{OFF}}{t_{ON}}$ where t_{OFF} is the run time reported when the optimization O is turned OFF and t_{ON} is the run time reported when O is turned ON. In the following sections, optimizations are grouped based on their dependencies, and the phase of the satisfiability test when they are applied. In a first group, the effect of exploiting *told nominal* relations is evaluated based on using the *told nominals heuristic* optimization, and the *lazy nominal generation* optimization. In a second group, the effect of optimized back-jumping is evaluated using the *look ahead*, *look back*, and the *learning* optimization techniques. In a third group, the effect of reduced partitioning is evaluated using the *incremental decomposition* optimization, the *lazy nominal generation*, and the *active roles heuristics*. Finally, the overall effect of all the optimizations used is evaluated.

7.3.1 Effect of Exploiting Told Nominal Interactions

Recall that *told nominal* interactions with roles can be used to reduce the size of the search space at preprocessing as described in Section 5.3.1. Also, told nominal interactions allow enabling the *heuristic guided nominal distribution* technique, described in Section 5.3.2, and which enforces an ordering on role variables when processed by the *ch*-rule; and the *lazy nominal generation* technique, which allows a nominal partition to be used on demand, as described in Section 5.8.

Test case	THA	T-A	TH-	T- -	- - -
WINE-$C_{IceWine}$	566	265	TO	TO	TO
KOALA-C_1	168	140	859	312	TO
KOALA-$C_2(n=3)$	332	360	546	14078	TO
KOALA-$C_{1-2}(n=3)$	426	1719	267641	TO	TO
COUNTRIES-$C_{SAT-simple-IEU(n=6)}$	219	328	156	172	922
COUNTRIES-$C_{UnSAT-simple-IEU(n=7)}$	109	109	234	250	73532
COUNTRIES-$C_{SAT-simple-EU(n=27)}$	343	328	343	2031	TO
COUNTRIES-$C_{UnSAT-simple-EU(n=30)}$	109	109	234	235	TO
TIME-$C_{SAT-DTD}$	195	202	TO	TO	TO

Table 15: Runtimes in milliseconds with the test cases with real world ontologies where one or more THA optimization (s) is (are) turned OFF.

Test case	S(THA)	S(H)	S(A)	S(HA)
WINE-$C_{IceWine}$	10596	0	1060	1060
KOALA-C_1	3577	1	5	2
KOALA-$C_2(n=3)$	1807	1	2	42
KOALA-$C_{1-2}(n=3)$	1408	4	628	1408
COUNTRIES-$C_{SAT-simple-IEU(n=6)}$	1749	1	1	6
COUNTRIES-$C_{UnSAT-simple-IEU(n=7)}$	5505	1	2	2
COUNTRIES-$C_{SAT-simple-EU(n=27)}$	4	1	1	1
COUNTRIES-$C_{UnSAT-simple-EU(n=30)}$	668	1	2	2
TIME-$C_{SAT-DTD}$	3073	1	3073	3073

Table 16: Speed up factor for the optimizations relying on *told nominal* interactions.

In the remainder of this section T, H, and A are used to refer to the *told nominal interactions with roles*, the *heuristic guided nominal distribution*, and the *lazy nominal generation* techniques respectively. The effect of enabling those techniques is

measured by running the test cases listed in Table 15 where one-of or all-of T, H, and A are disabled; the disabled optimization is marked using "-". The corresponding run times are shown in Table 15, and the speed up factor of each optimization is shown in Table 16.

The results show that the optimizations relying on *told nominal* interactions are crucial; without enabling these optimizations most test cases time out. The *Heuristic guided nominal distribution* does not always speed up reasoning, in case of tests with a minor speedup factor. This is expected because the *heuristic guided nominal distribution* does not affect the size of the number of partitions computed, but it affects how much faster a good distribution of individuals between partitions is found. The highest speed up factor for this optimization is reported with test case KOALA-$C_{1-2}(n = 3)$. The *lazy nominal generation* technique always improves performance, and the highest speed up factor for this optimization is reported with test case TIME-$C_{SAT-DTD}$.

Table 17 shows the different characteristics of the test cases in terms of; the size of the set of activated roles (AD_R), due to the *active roles heuristic* technique; the size of the set of *nominals* (N_o) occurring in the TBox for the test case used; and the size of the set of activated *nominals* (AN_o), due to the *lazy nominal generation* technique, for the test case. The size of AD_R and the size of AN_o determine the size of the global decomposition set $(\mathcal{DS})$, and hence affect the size of the global partitioning $(\mathcal{P})$ that needs to be computed. For example, if the *lazy nominal generation* technique is enabled, and assuming that the *active roles heuristic* is enabled as well, the total number of partitions (size of $\mathcal{P}$) that needs to be computed is equal to $2^{(\text{size of } AD_R + \text{ size of } AN_o)}$. Whereas, if the *lazy nominal generation* is not enabled, the total number of partitions that needs to be computed is equal to $2^{(\text{size of } AD_R + \text{ size of } N_o)}$.

Test case	size of AD_R	size of N_o	size of AN_o
WINE-$C_{IceWine}$	5	12	12
KOALA-C_1	4	6	3
KOALA-C_2(n = 3)	5	6	2
KOALA-C_{1-2}(n = 3)	7	6	3
COUNTRIES-$C_{SAT-simple-IEU(n=6)}$	1	6	6
COUNTRIES-$C_{UnSAT-simple-IEU(n=7)}$	1	6	6
COUNTRIES-$C_{SAT-simple-EU(n=27)}$	1	27	27
COUNTRIES-$C_{UnSAT-simple-EU(n=30)}$	1	27	27
TIME-$C_{SAT-DTD}$	1	14	14

Table 17: Characteristics of the elements of the global decomposition set for the different test cases - Part I.

Test case	size of $\mathcal{P} =$ $2^{\text{size of}(AD_R+N_o)}$	size of reduced $\mathcal{P} =$ $2^{\text{size of}(AD_R+AN_o)}$	resize factor $\frac{\text{size of } \mathcal{P}}{\text{size of reduced } \mathcal{P}}$
WINE-$C_{IceWine}$	131072	131072	1
KOALA-C_1	1024	128	8
KOALA-C_2(n = 3)	2048	128	16
KOALA-C_{1-2}(n = 3)	8192	1024	8
COUNTRIES-$C_{SAT-simple-IEU(n=6)}$	128	128	1
COUNTRIES-$C_{UnSAT-simple-IEU(n=7)}$	128	128	1
COUNTRIES-$C_{SAT-simple-EU(n=27)}$	268435456	268435456	1
COUNTRIES-$C_{UnSAT-simple-EU(n=30)}$	268435456	268435456	1
TIME-$C_{SAT-DTD}$	32768	32768	1

Table 18: Characteristics of the elements of the global decomposition set for the different test cases - Part II.

Table 18 shows the resize factor due to the *lazy nominal generation* technique which allows a reduced size of $\mathcal{P}$. The resize factor is computed based on the following formula:

$$\text{resize factor} = \frac{\text{size of } \mathcal{P}}{\text{size of reduced } \mathcal{P}} = \frac{2^{(\text{size of } N_o)}}{2^{(\text{size of } AN_o)}} \tag{47}$$

The *lazy nominal generation* is mostly effective with the KOALA ontology test cases, because these test cases have the highest resize factor. Figure 63 shows the newly introduced roles and the hierarchy between them after preprocessing the KOALA ontology. This hierarchy is the same for all of the KOALA test cases. However, each test case has a different decomposition set based on the concept satisfiability and the corresponding activated roles and *nominals*. When the *lazy nominal generation* technique is enabled, the number of *nominals* to be activated is equal to the size of AN_o, otherwise all *nominals* appearing in the TBox are generated and $AN_o = N_o$. This optimization avoids unnecessary computations of intersections between *nominals* and role fillers. An increased number of intersections affects performance because of the increased size of the search space due to the *ch*-rule. It is interesting to conclude which characteristics of a TBox allow a maximum benefit from this optimizations.

Those characteristics are shown in Tables 19 and 20 where, D_{Ro} denotes the set of roles (within AD_R) that require a nominal as a role filler when deciding the consistency of a KOALA ontology. In Table 20, D_R - AD_R shows the size of the total number of roles within the TBox (D_R), versus the total number of roles activated for the test case (AD_R). N_o - AN_o shows the size of the total number of *nominals* within the TBox (N_o), versus the total number of the *nominals* activated by the *lazy nominal generation* technique (AN_o). D_{Ro} shows the total number of *told nominal* roles (i.e. roles that require a nominal as a role filler), and N_{Ro} shows the total number of *nominals* used as role fillers.

248

Test case	AD_R	AN_o	D_{Ro}
KOALA-C_1	$\{hD_1, hH_1, hH_2, hG_1\}$	$\{PhD, male, female\}$	$\{hD_1, hG_1\}$
KOALA-$C_2(n=3)$	$\{hG_1, hG_2, hG_3, hC_1, hH_1, hH_2\}$	$\{male, female\}$	$\{hG_1, hG_2, hG_3\}$
KOALA-$C_{1-2}(n=3)$	$\{hD_1, hH_1, hH_2, hG_1, hG_2, hG_3, hC_1\}$	$\{PhD, male, female\}$	$\{hD_1, hG_1, hG_2, hG_3\}$

Table 19: Characteristics of the KOALA test cases - part I

Test case	D_R - AD_R	N_o - AN_o	D_{Ro}	N_{Ro}
KOALA-C_1	7 - 4	6 - 3	2	2
KOALA-$C_2(n=3)$	7 - 6	6 - 2	3	2
KOALA-$C_{1-2}(n=3)$	7 - 7	6 - 3	4	3

Table 20: Characteristics of the KOALA test cases - part II

When the set of *nominals* activated is smaller than the set N_o appearing in the TBox, having more *told nominal* roles for the same nominal helps also decrease the set of *nominals* being activated. If T is the only *optimization* enabled, then the more *told nominal* roles there are, the less partitions need to be considered.

7.3.2 Effect of Enhanced Back-jumping

This section presents an evaluation of the effect of enhanced back-jumping. Recall that back-jumping can be enhanced through *look ahead* and *look back* techniques to reduce the size of the search space, as described in Sections 5.4 and 5.5. In particular, the *ch-Rule look ahead*, and the $\sqcup$-*Rule lookahead* techniques are evaluated as *look ahead* techniques. As for *look back* techniques, the *back-jumping* and the *learning* techniques are considered. The evaluated techniques are referred to as the LAB techniques, L is used to refer to the *Learning* technique, A is used to refer to the *look Ahead* techniques, and B is used to refer to the *look Back* techniques. The effect of enabling those techniques is measured by running the test cases listed in Table 21

where one-of or all-of L, A, and B are disabled; the disabled optimization is marked using "-".

Test case	LAB	LA-	L-B	-AB	- -B	-A-	L- -	- - -
WINE-$C_{IceWine}$	566	1265	2641	2078	5140	563	2733	11578
KOALA-$C_{1-2(n=3)}$	426	344	438	391	374	344	345	1610
COUNTRIES-$C_{UnSAT-simple-IEU(n=7)}$	110	124	9578	1844	8547	1796	155	9671
COUNTRIES-$C_{UnSAT-simple-EU(n=30)}$	117	203	TO	TO	TO	TO	TO	TO
TIME-$C_{SAT-DTD}$	195	187	172	202	187	202	173	187

Table 21: Runtimes in milliseconds with the real world test cases where one or more LAB optimization(s) is(are) turned OFF.

Test case	S(LAB)	S(LA)	S(LB)	S(AB)	S(B)	S(A)	S(L)
WINE-$C_{IceWine}$	20	9	1	5	2	5	4
KOALA-$C_{1-2(n=3)}$	4	1	1	1	1	1	1
TIME-$C_{SAT-DTD}$	1	1	1	1	1	1	1
COUNTRIES-$C_{UnSAT-simple-IEU(n=7)}$	88	78	16	1	1	87	17
COUNTRIES-$C_{UnSAT-simple-EU(n=30)}$	5128	5128	5128	5128	2	5128	5128

Table 22: Speed up factor of the LAB optimizations used for enhanced back-jumping.

Test case	size of $\mathcal{DS}$ = size of AD_R + size of AN_o
WINE-$C_{IceWine}$	17
KOALA-$C_{1-2}(n = 3)$	10
COUNTRIES-$C_{UnSAT-simple-IEU(n=7)}$	7
COUNTRIES-$C_{UnSAT-simple-EU(n=30)}$	28
TIME-$C_{SAT-DTD}$	15

Table 23: Characteristics of the real world test cases used to evaluate the LAB optimizations.

The corresponding run times are shown in Table 21, and the speed up factor of each optimization is shown in Table 22. TO is used to refer to a test case which could not be checked within the time limit (600000 ms) set for the reasoner. To calculate the speed up factor for an optimization where T_{OFF} is a TO, the value of 600000 is used to get an estimate of the minimum speed up factor for that optimization. N/A is used if a certain speed up factor is not applicable, such is the case when TO could not be reached due to some error in the reasoner. The errors encountered are either due to a saturated JAVA heap space, or due to a partition's set whose size exceeds the number allowed by JAVA (see Section 6.8.1 for more details).

Considering the case with the real world ontologies, it is interesting to observe that the effect of the LAB optimizations seems to be maximized when these techniques are enabled together. The optimization with the minimal effect for all test cases is the *look back* technique. This minimal effect is to be expected mainly because the test cases considered do not rely on disjunctive descriptions, which means that not a lot of logical clashes would have alternative choice points. Instead clashes result in identifying *noGood* partitions and therefore, back-jumping works better with *learning*. Since, most of these test cases do not rely on the use of disjunctions, the size of the global partitioning ($\mathcal{P}$) is the main source of a search space blow up due to the non-determinism of the *ch*-Rule. The search space is exhausted if the test case is unsatisfiable. Therefore, although COUNTRIES-$C_{UnSAT-simple-IEU(n=7)}$ has a smaller decomposition set than TIME-$C_{SAT-DTD}$, most optimizations have a much higher speedup factor with Countries-$C_{UnSAT-simple-IEU(n=7)}$, which is unsatisfiable.

Another interesting observation, is that even when two cases are satisfiable, not only the size of $\mathcal{DS}$ affects non-determinism, but also the type of interactions between *nominals* and QCRs. For example, TIME-$C_{SAT-DTD}$ has a larger $\mathcal{DS}$ than the KOALA-$C_{1-2(n=3)}$, but a minimal speedup factor is reported for the LAB optimizations. This is

because the interaction of *nominals* with role fillers is very simple and straightforward in the case of Time-$C_{SAT-DTD}$, but occurs at different levels in the case of Koala-$C_{1-2(n=3)}$. The more complicated the interactions are between *nominals* and QCRs, the harder it becomes to guess/come up with an initial distribution of *nominals* which survives the expansion/test case. This explains why in the case of Wine-$C_{IceWine}$, more backtracking is needed to guess the right distribution of *nominals*, even though the size of $\mathcal{DS}$ is not much higher than with Time-$C_{SAT-DTD}$. Additional results are shown with synthetic ontologies tests in Tables 29, 30, and 28 of Appendix C, where it is also shown that in general, the speedup factor increases with the size of $\mathcal{DS}$.

7.3.3 Effect of Enhanced Partitioning

This section presents an evaluation of the effect of the optimizations used to enhance the computation of partitioning through the use of *lazy partitioning* and *lazy nominal generation* techniques as described in Sections 5.7 and 5.8 respectively. The evaluated techniques are referred to as the PRA techniques: P is used to refer to the *lazy Partitioning* optimization, R refers to the *active Roles heuristic* optimization, and A refers to the *lAzy nominal generation* optimization. The effect of enabling those optimizations is measured by running the test cases where one or all of PRA techniques are disabled. The corresponding run times are shown in Table 24, and the speedup factors are shown in Table 25. Note that the speedup factors are only calculated for P and R individually. This is because these two optimizations do not depend on each other, and they are not expected to enhance reasoning if both enabled. In fact, if they are both enabled, the P optimization takes over because partitions are no longer created and later activated. Instead, once a partition is created, it is automatically activated.

Test case	PRA	-RA	P-A	PR-	- -A	-R-	P- -	- - -
WINE-C_{IceWine}	566	TO	516	TO	TO	ERR	ERR	ERR
KOALA-C_1	168	53343	156	859	52718	ERR	ERR	ERR
KOALA-$C_{2(n=3)}$	332	TO	266	546	444171	ERR	ERR	ERR
KOALA-$C_{1-2(n=3)}$	426	49055	1703	267641	44406	ERR	ERR	ERR
COUNTRIES-$C_{\text{SAT}-\text{simple}-\text{EU}(n=27)}$	325	155	344	343	140	407	375	251
COUNTRIES-$C_{\text{UnSAT}-\text{simple}-\text{EU}(n=30)}$	117	156	110	234	126	249	281	218
TIME-$C_{\text{SAT}-\text{DTD}}$	195	218	188	TO	251	375	218	374

Table 24: Runtimes in milliseconds with the real world test cases where one or more PRA optimization (s) is (are) turned OFF.

Test case	S(P)	S(R)	S(PA)	S(RA)
WINE-C_{IceWine}	1060	0.9	N/A	N/A
KOALA-C_1	318	0.9	N/A	N/A
KOALA-$C_{2(n=3)}$	1807	0.8	N/A	N/A
KOALA-$C_{1-2(n=3)}$	115	4	N/A	N/A
COUNTRIES-$C_{\text{SAT}-\text{simple}-\text{EU}(n=27)}$	0	1	1.2	1.1
COUNTRIES-$C_{\text{UnSAT}-\text{simple}-\text{EU}(n=30)}$	1	0.9	2.1	2.4
TIME-$C_{\text{SAT}-\text{DTD}}$	1	0.9	1.9	1.1

Table 25: Speed up factor of the PRA optimizations used for enhanced partitioning.

The corresponding speedup factor is calculated as follows: $S(P) = \frac{T_{-RA}}{T_{PRA}}$, $S(R) = \frac{T_{P-A}}{T_{PRA}}$, $S(PA) = \frac{T_{-R-}}{T_{PRA}}$, and $S(RA) = \frac{T_{P--}}{T_{PRA}}$. Whenever T_{***} corresponds to a TO entry, the value of 600000 (= 10 minutes) is used instead to give a minimum speedup factor for the corresponding optimization; for example, the speedup factor of P with the WINE-C_{IceWine} is calculated as $S(P) = \frac{T_{-RA}}{T_{PRA}} = \frac{600000}{566} = 1060$. In the case when T_{***} was not calculated because of an error (ERR), N/A is used to denote that the corresponding speedup factor could not be calculated. It is easy to see that the speedup factor of the *lazy partitioning* technique is much higher than that of the

active roles heuristic. This is because the *active roles heuristic* does not avoid the overhead of computing partitions. An interesting observation is the significant effect of the *lazy nominal generation* when combined with either the *active roles heuristic,* the *lazy partitioning,* or with both optimizations. If the *lazy nominal generation* optimization is disabled with at least one of PR- optimizations, the test cases with the highest speed up factor for P and R run out of memory.

7.3.4   Overall Optimizations Effect

In the previous sections, the effect of groups of optimizations were evaluated. Optimizations were grouped based on the level on which they are applied, or based on their interoperability. In this section, we study the overall effect of all the implemented dynamic optimizations, which aim at reducing non-determinism dynamically, together with those aiming at enhancing preprocessing.

Test case	Optimizations ON	Optimizations OFF	Speedup
Wine-$C_{IceWine}$	566	TO	1059.60
Koala-C_1	168	TO	3576.75
Koala-$C_{2(n=3)}$	332	TO	1807.23
Koala-$C_{1-2(n=3)}$	426	TO	1408.45
Countries-$C_{SAT-simple-IEU(n=6)}$	219	453	2.07
Countries-$C_{UnSAT-simple-IEU(n=7)}$	110	TO	2739.73
Countries-$C_{SAT-simple-EU(n=27)}$	343	TO	1749.27
Countries-$C_{UnSAT-simple-EU(n=30)}$	109	TO	5504.59
Time-$C_{SAT-DTD}$	195	TO	3072.98

Table 26: Runtimes in milliseconds showing the overall optimizations effect. The speedup factor is calculated as $\frac{OptimizationOFF}{OptimizationON}$, and the value 600000 milliseconds is used for TO - Part I.

254

Test case	Optimizations ON	optimizations OFF	Speedup
$C_{SAT-lin-\mathcal{ALCQ}(i=10)}$	313	251	0.80
$C_{UnSAT-lin-\mathcal{ALCQ}(i=10)}$	1385	218	0.16
$C_{QCR-\mathcal{ALCQ}(n=4)}$	10867	15532	1.43
$C_{QCR-var-\mathcal{ALCQ}(n=4)}$	69585	15047	0.22
$C_{QCR-disjunctive-atMost-\mathcal{ALCQ}(n=4)}$	2218	3624	1.63
$C_{QCR-disjunctive-Least-\mathcal{ALCQ}(n=4)}$	76182	1675	0.02
$C_{QRatio-\mathcal{ALCQ}(n=5)}$	11578	TO	51.82
$C_{SAT-UnSAT-\mathcal{ALCQ}(n=24)}$	117	547	4.68
$C_{SAT-UnSAT-\mathcal{ALCQ}(n=10)}$	102	452	4.45
$C_{Back-disjunctive-\mathcal{ALCQ}(i=10)}$	72553	TO	8.27
$C_{UnSAT-nested-\mathcal{ALCHOQ}}$	516	TO	1162.79
$C_{SAT-cyclic-\mathcal{ALCHQ}}$	202	TO	2970.30
$C_{UnSAT-cyclic-\mathcal{ALCHQ}}$	249	TO	2409.64
$C_{SAT-cyclic-\mathcal{ALCHOQ}}$	281	TO	2135.23
$C_{UnSAT-cyclic-\mathcal{ALCHOQ}}$	110	TO	5454.55

Table 27: Runtimes in milliseconds showing the overall optimizations effect. The speedup factor is calculated as $\frac{OptimizationOFF}{OptimizationON}$, and the value 600000 milliseconds is used for TO - Part II.

It turns out that without enabling the optimizations for preprocessing (initial partition elimination techniques) most test cases time out, and the speedup factor could not be calculated. Therefore, the partition elimination techniques were kept enabled as part of the partitioning rather than as an optimization. The results, as shown in Figures 26 and 27, show that the optimizations proposed and implemented in HARD significantly improve the performance; without these optimizations, HARD times out in most test cases. In fact, most DL reasoners fail to have any practical merit if naïvely implemented; the first DL reasoner (KRIS) implementing a tableau

algorithm for $\mathcal{ALC}$ was useless. Many attempts to speed up tableau based reasoners, like KRIS, did not succeed until these reasoners were equipped with the optimizations proposed in [Hor97]. Such is the case with almost every proposed (tableau and non-tableau) reasoning procedure for expressive DLs.

It is easy to speculate that HARD would be of no practical merit due to the high complexity of the algorithm implemented. However, the performance analysis presented in this chapter shows that with the design of suited optimizations, one can achieve a speed up of 3 orders of magnitude compared to a naïve implementation. The speed up improvement not only allows a better performance compared to a naïve implementation of algebraïc reasoning, but also compared to implementations based on less informed calculi, where the speedup improvement can be of 2 orders of magnitude, as was shown in Section 7.2.

7.4   Discussion

The overall effect of the optimization adopted by HARD shows that the practical merit of ReAl DL can be easily questioned, if no suitable optimizations were found. This section discusses the practical performance of ReAl DL compared to existing approaches in handling different types of interactions between *nominals*, QCRs and *role hierarchies*, as well as the effect of the optimizations adopted.

7.4.1   Practical Performance

The proposed optimizations allow a reasoner to handle QCRs and *nominals* better than any existing reasoner handling *nominals*. The total number of test cases relying on the use of *nominals* is 85, and HARDs reported performance was better than at least one other reasoner in 73 test cases (86%). Among the test cases with nominals,

56 cases are based on the interaction between an increasing number of *nominals* and QCRs, HARD outperformed at least one other reasoner (by several orders of magnitude) in all these cases (100%). In Section 7.2.1, the performance results show the advantage of using ReAl DL to handle QCRs. Such results are not any news because similar results have already been shown in [Far08] and [HTM01]. The purpose of the evaluation is however, to show that the optimizations discussed in Chapter 5 allow a more efficient reasoning even though the added complexity of handling *nominals* was expected to degrade performance. In particular, the optimizations aiming at enhanced partitioning give the algorithm a pay-as-you-go characteristics by simulating some form of local decomposition, and allow to reproduce the results reported with the algebraïc approach for $\mathcal{SHQ}$ if the input ontology does not rely on *nominals*. Also, previous evaluation criteria were extended to evaluate the performance of HARD in dealing with non-determinism due to disjunctions and the added expressivity of $\mathcal{SHOQ}$ (see Section 7.2.2). The main results of the performance evaluation are grouped into results showing a poor performance of HARD and those showing a strong performance.

Poor performance   The performance of HARD was poor in the following cases:

- The number of QCRs used within the label of a node is large. A performance degradation is associated with an increased number of QCRs within the label of a node as with the test cases $\mathsf{C}_{\mathsf{QCR}-\mathcal{ALCHQ}}$. An implementation limitation was also reported when the increasing number of QCRs exceeds 13 due to a limitation of the JAVA long type. On the other hand, not so many test cases or concept descriptions exists where within the same label of a node more than 13 QCRs need to be solved. This performance degradation is more due to an implementation restriction than it is due to a restriction in the theoretical basis of the algorithm. A smarter implementation would not necessarily lead to such

poor performance.

- The number of at-most restrictions within a concept description is high. When the number of at-most restriction increases, the Constraint Solver needed more time to find a solution. This performance degradation might be enhanced by having the Constraint Solver find any integer solution rather than always search for an optimal one. From a satisfiability point of view, it is enough to test whether the encoded in-equations admit a solution or not. The solution itself is not so interesting and does not affect a completion model; this is because empty partitions are not represented and non-empty ones are represented using a proxy nodes, the cardinality of the partition does not have an effect on the completion model. From a completion graph expansion point of view, a minimal number of proxy nodes is usually desirable because it can reduce the number of expansion rules to be applicable, some of which could be non-deterministic. It might be interesting to investigate the characteristics of a given ontology (or the encoded in-equations) where the algorithm could switch minimality (ON/OFF) for the sake of performance due to a minimum number of nodes (less expansions) versus a faster Constraint Solver.

Strong performance   HARD's performance was as good as existing reasoner or better in the following cases:

- The numbers used in QCRs are large. The stability of HARD and RacerPro in solving the test cases $C_{*-*-\mathcal{ALCQ}}$ shows the advantage of solving QCRs using algebraïc reasoning over adopting another reasoning approach. This was also shown in test cases where a large number used in QCRs interacts with a large number of *nominals*. HARD maintains a stable performance, and performs better than existing reasoners in satisfiable and unsatisfiable cases by several

orders of magnitude in 100% of the cases. The performance stability is not only maintained because HARD relies on LPSolver to decide the satisfiability of QCRs, but also because HARD relies on the use of proxy nodes to model a solution. For example, the same CCG is valid for different values of n in $C_{\mathsf{SAT}-*-*}$. This means that the large numbers used in QCRS do not necessarily affect the size of the completion graph where it is known that more nodes in the completion graph invoke more rules, some of which might be non-deterministic.

- The depth of the hierarchy between roles is greater than 1.

- The unsatisfiability of a concept expression is caused by disjunctive descriptions or due to unsatisfiable numerical restrictions, or a combination of both.

- The concept descriptions include cycles. HARD's re-use of individuals showed a strong handling of cycles within concept descriptions where a natural halt is guaranteed even without the use of special blocking strategies.

Although the use of *concrete datatypes* is usually cheaper for reasoners, not much can be inferred when they are used. Recall from the COUNTRIES ontology, that when a country's English name property is modelled using a *concrete datatype*, the fact that a countrys English name must be the same name used in every model of an English country is lost. A better handling of *nominals* allows real world ontologies to rely more on the *nominals* constructor. Such is the case with the test cases in Section 7.2.2 where the use of *nominals* replaced some uses of *concrete datatypes*.

7.4.2 Effect of Adopted Optimizations

The goal of the optimizations proposed in Chapter 5 is to have an almost optimal algorithm with respect to the worst-case complexity of concept satisfiability with the

DL $\mathcal{SHOQ}$. While some optimizations target a better handling of a source of inefficiency due to the use of a certain constructor, the overall effect of the optimizations is dramatic. In fact, without optimizations, HARD cannot solve some small problems within realistic time (10 minutes to few hours) without running out of memory. Such results are not surprising, this was already expected due to the double exponential complexity of a naïve tableau algorithm.

The optimizations evaluated are the optimizations aiming at enhancing the major sources of inefficiency with ReAl DL, namely global partitioning and non-determinism. With respect to practical partitioning, using *role hierarchy relations* to discard partitions can be further exploited in such a way that the encoding of QCRs can be clustered into sets of independent systems of in-equations such that the satisfiability of each system can be checked independently while still guaranteeing that the satisfiability of the combination of the systems of in-equations still hold.

The effect of the *disjointness relations* is problem dependent and is best reached when the qualifications used with QCRs rely on concepts such that these concepts are disjoint. This optimization may not enhance performance in ontologies where disjointness relations between concepts are minimal. On the other hand, the disjointness relations between nominals are implicitly assumed, however, in most ontologies the disjointness relations between *nominals* are not explicitly declared either because ontology designers overlook the fact that these *nominals* might interact or because they assume that *nominals* never interact. For example, the *nominals* {Canada} and {USA} used to enumerate countries can never interact because Canada and USA are assumed to be disjoint concepts and {Canada} $\sqsubseteq \neg${USA} may not be explicitly declared. However, sometimes different *nominals* are used to refer to the same concept such is the case with the *nominals* enumerating longEnglishName and shortEnglishName and referring to same countries such as {Canada} and {CA}. The disjointness

assumption in this case might result in wrong answers unless the {Canada} and {CA} are explicitly declared as equivalent. Usually, there are more disjoint *nominals* than equivalent ones, and therefore it is more reasonable to assume disjointness rather than assume possible equivalence. In that case, the *disjointness relations* always improves performance with ontologies using *nominals*.

Using *told nominals interactions with roles* works well and the effect is maximized when combined with *heuristic guided nominal distribution* and *lazy nominal generation*. The effect of *heuristic guided nominal distribution* can be further enhanced by studying the restrictions on qualifications (using the ∀ constructor) as discussed in Section 7.2.2.2 with the WINE ontology. It is interesting to note that the type of interactions between *nominals* and their told nominal roles affects back-jumping. In order to facilitate how quickly a good nominal distribution is found, one could investigate ways to apply ordering heuristics.

Lazy nominal generation allows a reduced partitioning especially when combined with *active roles heuristic* or *lazy partitioning*. Those optimizations allow a pay-as-you-go characteristics for the algorithm when we have a large number of *nominals* but only few are needed. The effect of *active roles heuristic* and *lazy partitioning* is maximized with ontologies using disjunctions or nested expressions with QCRs. Unlike *active roles heuristic*, *lazy partitioning* does not compute a partition before activating it, therefore, it has a higher speed up factor because it avoids the computation of unnecessary partitions.

Non-determinism with ReAl DL is caused by the ⊔-Rule and the *ch*-Rule. These rules expand the search space by extending different labels of a certain node. Therefore, the dependencies of their branches do not interact with each other, which allows a more fine grained backtracking unlike the case with the standard tableau algorithm for $\mathcal{SHOQ}$ where non-determinism is due to the ⊔-Rule and the *choose*-Rule both

extending the same label of a node with concept descriptions causing dependencies from the application of both rules to interact, and making DDB less efficient.

The effectiveness of the ⊔-*Rule lookahead* can be considered problem dependent. On the other hand, the *ch-Rule lookahead* can improve performance in most cases. This is because unless all partitions are assigned elements, there is always going to be cases where a *ch*-Rule branch must be discarded due to a clash. In fact, there are always more partitions than QCRs and *nominals*. The *ch*-Rule could be further enhanced by implementing some form of dependencies for variables, allowing an arithmetic clash (non-obvious one) to be tracked down to a variable choice point, allowing enhanced *back-jumping* in case of arithmetic clashes detected by the constraint solver. For now, the only *back-jumping* implemented in case of arithmetic clashes is during the lookahead phase which can detect obvious arithmetic clashes.

7.5 Conclusion

In this chapter, the performance of the prototype reasoner (HARD), implementing ReAl DL, is evaluated using a set of real world and synthetic ontologies. The evaluation of the effect of different $\mathcal{SHOQ}$ constructors on reasoning performance shows the advantage of using algebraïc reasoning to handle QCRs, role hierarchies, *nominals*, and the interaction between these constructors. Even though the handling of *nominals* imposes a global partitioning, the optimizations adopted allow a pay-as-you-go characteristic for the overhead of handling and dealing with a global partitioning. The labelling of nodes based on the partition where they belong to and the re-use of these nodes allows HARD to handle cyclic descriptions even without adopting a special blocking strategy. Most generated CCGs are of compact size due to the use of proxy nodes.

This evaluation shows the practical contribution of the algebraïc reasoning algorithm and the effectiveness of the optimization techniques adopted. The main implementation limitation encountered is due to the use of JAVA array, to store and retrieve the elements of the global decomposition set, and the use of array indexes in binary representation to refer to a partition. Note that some of the test cases were also run on a Mac OS X Version 10.6.7 with 2.66 GHz Intel Core 2 Duo processor and 4GB of memory. No significant difference in the performance was reported compared to running the tests on the Windows OS, and the runtimes reported in this chapter were restricted to the ones running on windows OS. Some future directions for the algebraïc approach and a better handling of the cases where HARD did not perform very well will be discussed in the next chapter.

Chapter 8

Conclusions and Outlook

The popularity of Description Logics in Knowledge Representation and modelling, has made DLs the topic of many research efforts focused on extending their expressivity as well as their reasoning efficiency. It has become essential for a DL not only to handle the expressivity to model all elements within an application's domain, but also to allow efficient reasoning when the expressivity is fully used. It was shown (in Chapter 3) that existing standard tableau-reasoning procedures are arithmetically uninformed and blind, which renders DL reasoners implementing them very inefficient in handling the expressivity of *nominals* and QCRs through inferences.

Nominals capture the notion of uniqueness and identity; they are needed in many real world domains as names for concepts with only one instance (e.g., "God", "Red", "Canada", etc...). The use of QCRs for modelling domain elements was found essential since advocated in [RS05]. The lack of real world ontologies relying on a rich usage of both *nominals* and QCRs is mainly due to the inability of existing DL reasoners to handle these constructors efficiently. The problem of DL reasoning with QCRs and *nominals* is sometimes referred to as a chicken-and-egg problem. Ontology designers often substitute the use of *nominals* or QCRs with *concrete datatypes*, as was shown in some of the ontologies referred to throughout this thesis, for the sake of reasoning

efficiency. However, the use of *concrete datatype* does not enforce the semantics of *nominals* or QCRs, which means that less inferences can be deduced.

The main objective of this thesis as outlined in Section 1.1.1 was to design an efficient reasoning algorithm to handle the expressivity of DLs enabling the QCRs and *nominals* constructors, as most existing reasoners handle such DLs poorly. Instead of optimizing them, a better informed approach is desirable. The rest of this chapter shows which objectives have been met while identifying the main contributions of this thesis.

8.1 Research Methodology and Contributions

The research methodology adopted consists of devising a theoretically sound, complete, and terminating reasoning algorithm, and designing a practical implementation of such an algorithm. The contribution of this thesis is therefore twofold: theoretical and practical.

8.1.1 Theoretical Contributions

- A hybrid algebraïc reasoning algorithm handling $\mathcal{ALCQ}$, which is the basic DL $\mathcal{ALC}$ extended with QCRs, was proposed and published in [FFHM08a]. This calculus is the first sound, complete, and terminating hybrid algebraïc tableau-based reasoning algorithm handling QCRs.

- An extension of the algorithm to handle the DL $\mathcal{SHOQ}$, which is $\mathcal{ALCQ}$ extended with *nominals*, *role hierarchies* and GCIs, is proposed and published in [FH10a]. This calculus was described in full details in Chapter 4 along with proofs for soundness, completeness, and termination. Hence, Objectives (1), (2), and (3), from Section 1.1.1, are met.

- Optimization techniques suitable for the calculus were designed and proposed in Chapter 5. The goal of the optimizations is to achieve an optimal average (typical)-case behaviour of the algorithm w.r.t its worst-case complexity. Some of the optimization techniques were adapted from standard DL systems or CSPs so that they take into account the hybrid nature of the algorithm. Novel optimizations were proposed such as the use of *noGood* variables. These optimizations are crucial to meet Objective (4).

Novel features

1. A novel feature of algebraïc reasoning with DL is the encoding of the *nominals* semantics using IP models. The price of such a feature is a double exponential complexity due to handling of these semantics globally (see Section 4.6).

2. Another novel feature is due to the re-use of role-successors which ensures termination of a completion graph expansion without the need for implementing sophisticated blocking strategies. Such re-use of nodes could lead to unnecessarily over-constraining a role-successor in some situations.

3. Numerical restrictions are satisfied before creating domain elements, which means that nodes, that created in a CCG, are never merged and there is no need for a mechanism of merging or handling the so-called "yoyo" effect (a possibly infinite cycle of creating and merging domain elements which causes a termination problem).

4. Due to the partitioning of the domain element into equivalence classes, elements with the same restrictions fall into the same partition. We use one proxy element as a representative for each partition's elements. This means that there is no need to implement any blocking strategies to ensure termination since no two

elements with the same restrictions will be created. The use of proxy nodes is inspired by [HM01a], it allowed the use of *noGood partitions* which enabled some form of *caching* and *learning*, and it can also be used to address the creation of large models (another source of inefficient DL reasoning).

5. Due to (1), (2), (3), and (4) in contrast to existing tableau algorithms, a tree model property with cycle detection techniques is not crucial to ensure termination of a completion graph expansion.

8.1.2 Practical Contributions

A running prototype reasoner, HARD was developed and evaluated against existing SOTA DL reasoners, as described in Chapters 6 and 7, respectively. To the best of our knowledge, HARD is the first DL reasoner which employs algebraïc reasoning handling the *nominals* constructor.

HARD consists of five main components. The Reasoner Controller gets the input ontologies file, selected from the computer directory, and delegates tasks to Configuration Loader, Parser, Inference Engine, and Constraint Solver in order to decide the consistency of the ontology. The Configuration Loader loads the ontology file into an ontology object and stores relevant information about the ontology. The Inference Engine applies the reasoning procedure to the ontology, and encodes the numerical restrictions imposed by nominals and QCRs into an IP model. Finally, the Constraint Solver solves the set of constraints generated by the Inference Engine by calling an external constraint solver, LPSolver. Note that the main difference between HARD and other standard DL systems is that, HARD relies on both the Inference Engine and the Constraint Solver to complete the reasoning tasks, whereas the standard DL systems need only the Inference Engine since they do not generate IP models during the completion rule application.

The empirical evaluation in Chapter 7 showed that HARD outperforms existing SOTA reasoners in about 85% of the test cases (some of which were based on real world ontologies) with a speedup factor reaching 2-3 orders of magnitude. Hence, Objective (5) is met.

8.2   Open and Future Work

The importance of the research done relies in the scalability of ReAl DL to handle more expressive logics and the usability of ReAl DL by existing or future DL reasoners. Before discussing the scalability, we highlight some of the open work.

Open work   Some of the open work was left due to time constraints, and some was spotted while analyzing the empirical evaluation.

- Handling *qualifying concepts*. Although *qualifying concepts* (see Definition 4.1.1), were part of the theoretical presentation of ReAl DL in Chapter 4, they were not handled by the prototype reasoner due to time constraints (see Section 6.8 for details). Once enabled, and as an optimization for reducing the size of the global decomposition set ($\mathcal{DS}$), one could make use of the equivalence between ($\forall R.C \sqcap \forall R.D$) and $\forall R.(C \sqcap D)$ to group *qualifying concepts* for the same role into one *qualifying concept*, hence reducing the size of N_Q (which becomes bounded by the size of N_R) and $\mathcal{DS}$.

- Ordering heuristics for the applicability of the *ch*-Rule. It was shown in Chapter 7 that the order in which the *ch*-Rule is applied on variable could have an effect on performance. Therefore, a possible optimization would be to devise some ordering heuristics which allows the *ch*-Rule to guess the best variable to branch on.

268

- Applying global decomposition on the fly. The satisfiability of the *nominals* semantics together with that of QCRs is challenging only in cases when at-most restrictions (either imposed by QCRs or by *nominals*) interact with those imposed by at-least restrictions. Therefore, even when *active roles heuristics* or *lazy partitioning* are enabled, one could delay applying a global decomposition until at-most restrictions are encountered, or until a clash occurs. For example, considering the expression $(\geq 3R.C \sqcap \geq 5S.C)$, such that R and S do not share any hierarchy or any *told nominal*, one could assume that initially every restriction is satisfied by exactly one proxy node and apply a decomposition only if a clash occurs instead of once a role is activated.

- Exploiting relations between concepts and roles to group partitions. Recall that in Chapter 5, relations between concepts and roles are exploited to identify *noGood* and *quasi-noGood* partitions. As a complement, and similar to what is done based on *told nominal* relations, one could exploit relations between roles (e.g, R and S) and their qualifications (e.g., C and D respectively) in order to guess the best initial distribution of R-fillers and S-fillers between partitions.

- Grouping QCRs into separate independent sets. An increased number of QCRs directly affects the performance of ReAL DL reasoning because it affects the size of the global decomposition set. One could investigate how to solve an increased number of QCRs using separate, and independent systems of in-equations (using separate decomposition sets) such that the combination of the solutions from all groups still preserves the semantics of the encoded constraints.

Scalability   The scalability of ReAl DL relies in its ability to support more expressive DL languages. The hybrid algebraïc tableau-based reasoning algorithm presented

in [FFHM08a, FFHM08b] forms the basis for the hybrid algebraïc tableau-based reasoning algorithms for the DLs $\mathcal{ALCOQ}$ [FHM08, FHM09], $\mathcal{SHQ}$ [FH10c], and $\mathcal{SHOQ}$ [FH10a].

- It would be interesting to extend ReAl DL to support more expressive DL languages, such as $\mathcal{SROIQ}$. The DL $\mathcal{SROIQ}$ is the logic foundation for OWL2. In addition to the language constructors supported by $\mathcal{SHOQ}$, the DL $\mathcal{SROIQ}$ also allows *inverse role*, and role relations. The major challenge for this extension would be the handling of the problematic interaction between *nominals*, QCRs and *inverse roles* ($\mathcal{I}$). The handling of *inverse roles* is ongoing research [Pou].

- It would be interesting to investigate if the form of *caching* enabled by the use of *noGoods* can be exploited to yield a single exponential algorithm.

- Also, it would be interesting to investigate if handling QCRs using IP models allows a more relaxed restrictions for the numbers allowed with *transitive roles*.

Usability A typical question one is faced to answer after reaching a certain goal would be "'What is going to happen next?". In the case of the ReAl DL discussed in this thesis, one would be interested in investigating the usability of the approach in contexts other than a prototype reasoner.

- A straightforward answer would be the adoption of ReAL DL by existing DL reasoners, which perform badly when handling the interaction between nominals and QCRs is required. For instance, RacerPro, already adopts some form of ReAL DL with QCRs, but does not handle *nominals*; it could therefore be extended or adapted with the hybrid calculus presented in this thesis for a handling of the *nominals* expressivity. On the other hand, existing tableau-based reasoners handling the expressivity of both *nominals* and QCRs could

be amenable to adopting algebraïc reasoning together with their sophisticated optimization techniques due to the tableau basis of ReAl DL.

- Since *nominals* are considered as a more general form of ABox individuals, a possible application of ReAL DL would be with ABox reasoning, or even in query answering in case when database queries are reformulated using ontology terms ("concepts", "roles", and "individuals") as in [NF04].

- An interesting applicability of algebraïc reasoning can be that of ensuring that a minimal model is always generated. Due to the use of proxy nodes, and IP problems with the objective of minimizing constraints, one could investigate if a minimal model generation could be established.

- Finally, the translation of DL constraints into IP models could render the transition of optimization problems, which are heavily used in company management (e.g., planning, production, transportation, technology), into the Semantic Web even more interesting. Typical optimization problems in management would be to maximize profits or minimize costs given certain limited resources. A semantic representation of such problems using DL and IP models is surely an exciting combo for business ontologies.

"I have a dream for the Web [in which computers] become capable of analysing all the data on the Web – the content, links, and transactions between people and computers. A "Semantic Web," which should make this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The "intelligent agents" people have touted for ages will finally materialize."

Berners-Lee & Fischetti in [BLF99]

Bibliography

[AL02]   Carlos Areces and Carsten Lutz. Concrete domains and nominals united. In Carlos Areces, Patrick Blackburn, Maarten Marx, and Ulrike Sattler, editors, *Proceedings of the fourth Workshop on Hybrid Logics HyLo'02*, pages 145–149. Springer-Verlag, 2002.

[Bak95]   Andrew B. Baker. *Intelligent Backtracking On Constraint Satisfaction Problems: Experimental And Theoretical Results*. PhD thesis, Graduate School of the University of Oregon, 1995.

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

[BBL05]   Franz Baader, S. Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, pages 364–369, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[BDS93]   Martin Buchheit, Francesco M. Donini, and Andrea Schaerf. Decidable reasoning in terminological knowledge representation systems. *Artificial Intelligence Research*, 1(RR-93-10):109–138, 1993.

[BH91a]   Franz Baader and Bernhard Hollunder. KRIS: Knowledge representation and inference system. *SIGART Bull.*, 2(3):8–14, 1991.

[BH91b]   Franz Baader and Bernhard Hollunder. A terminological knowledge representation system with complete inference algorithms. In *Proceedings of the First International Workshop on Processing Declarative Knowledge*, volume 572, pages 67–85, Kaiserslautern (Germany), 1991. Springer–Verlag.

[BHLW03]   Franz Baader, Jan Hladik, Carsten Lutz, and Frank Wolter. From tableaux to automata for description logics. volume 57, pages 247–279, Amsterdam, The Netherlands, The Netherlands, 2003. IOS Press.

[BHS03]   Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. In *Mechanizing Mathematical Reasoning: Essays in Honor of Jörg Siekmann on the Occasion of His 60th Birthday*, Lecture Notes in Artificial Intelligence, pages 228–248. Springer, 2003.

[BL06]   Franz Baader and Carsten Lutz. *Handbook of Modal Logic*, volume 3 of *Studies in Logic and Practical Reasoning*, chapter Description Logic. Elsevier Science and Technology Books, 2006.

[BLF99]   Tim Berners-Lee and Mark Fischetti. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*. Orion Business, 1999.

[BLHL01]   Tim Berners-Lee, James Hendler, and Ora Lassila. The semantic web: A new form of web content that is meaningful to computers will unleash a revolution of new possibilities. *Scientific American*, (285):34–43, 2001.

[BLS06]   Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. CEL - A polynomial-time reasoner for life science ontologies. In *IJCAR*, pages 287–291, 2006.

[BM01]   Patrick Blackburn and Maarten Marx. Third int. workshop on hybrid logic HyLo'01. *Logic Journal of the IGPL*, 9(5), 2001.

[BMPSR91]   Ronald J. Brachman, Deborah L. McGuiness, Peter F. Patel-Schneider, and Lori A. Resnick. Living with CLASSIC: when and how to use a KL-ONE-like language. In John Sowa, editor, *Principles of semantic networks: Explorations in the representation of knowledge*, pages 401–546. Morgan Kaufmann, San Mateo, US, 1991.

[BS99]   Franz Baader and Ulrike Sattler. Expressive number restrictions in description logics. *Journal of Logic and Computation*, 9(3):319–350, 1999.

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

[CFGL04]   Chiara Cumbo, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. Enhancing the magic-set method for disjunctive datalog programs. In *Logic Programming)*, Lecture Notes in Computer Science, pages 371–385. Springer, 2004.

[CLRS01]   Thomas H. Cormen, Charles E. Leiserson, Ronald L. Ribest, and Clifford Stein. *Introduction to Algorithms*, chapter The simplex algorithm, pages 790–804. MIT Press, second edition, 2001.

[Dan63]   George Bernard Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.

[DF02]   Rina Dechter and Daniel Frost. Backjump-based backtracking for constraint satisfaction problems. *Artificial Intelligence*, 136(2):147–188, April 2002.

[Din08]   Yu Ding. *Tableau-based Reasoning for Description Logics with Inverse Roles and Number Restrictions*. PhD thesis, Concordia University, 2008.

[Far08]   Nasim Farsinia. Combining integer programming and tableau-based reasoning: A hybrid calculus for the description logic $\mathcal{SHQ}$. Master's thesis, Concordia University, 2008.

[FB07a]   Deborah L. McGuinness Daniele Nardi Peter F. Patel-Schneide Franz Baader, Diego Calvanese. *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter Basic Description Logics. Cambridge University Press, 2007.

[FB07b]   Deborah L. McGuinness Daniele Nardi Peter F. Patel-Schneide Franz Baader, Diego Calvanese. *The Description Logic Handbook: Theory, Implementation, and Applications*, chapter Implementation and optimisation techniques. Cambridge University Press, second edition edition, 2007.

[FFHM08a] Jocelyne Faddoul, Nasim Farsinia, Volker Haarslev, and Ralf Möller. A hybrid tableau algorithm for $\mathcal{ALCQ}$,. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Proceedings of the 21st International Workshop on Description Logics (DL 2008), Dresden, Germany, May 13-16, 2008*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.

[FFHM08b] Jocelyne Faddoul, Nasim Farsinia, Volker Haarslev, and Ralf Möller. A hybrid tableau algorithm for $\mathcal{ALCQ}$. In Malik Ghallab, Constantine D.

Spyropoulos, Nikos Kakotakis, and Nikolaos M. Avouris, editors, *18th European Conference on Artificial Intelligence (ECAI 2008)*, volume 178 of *Frontiers in Artificial Intelligence and Applications*, pages 725–726. IOS Press, 2008.

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

[FH10b] Jocelyne Faddoul and Volker Haarslev. Optimizing algebraic tableau reasoning for $\mathcal{SHOQ}$: First experimental results. In Volker Haarslev, David Toman, and Grant E. Weddell, editors, *Proceedings of the 23rd International Workshop on Description Logics (DL 2010), Waterloo, Ontario, Canada, May 4-7, 2010*, volume 573 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2010.

[FH10c] Nasim Farsiniamarj and Volker Haarslev. Practical reasoning with qualified number restrictions: A hybrid abox calculus for the description logic. *AI Communications*, 23(2-3):205–240, 2010.

[FH11] Jocelyne Faddoul and Volker Haarslev. Optimized algebraic tableau reasoning for the description logic $\mathcal{SHOQ}$. *Journal of Artificial Intelligence Research (JAIR) - In preparation*, 2011.

[FHM08] Jocelyne Faddoul, Volker Haarslev, and Ralf Möller. Hybrid reasoning for description logics with nominals and qualified number restrictions. Technical report, Institute for Software Systems (STS), Hamburg University of Technology, 2008. http://www.sts.tu-harburg.de/tech-reports/papers.html.

[FHM09]   Jocelyne Faddoul, Volker Haarslev, and Ralf Möller. Algebraic tableau reasoning for the description logic $\mathcal{ALCOQ}$. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 22nd International Workshop on Description Logics (DL 2009), Oxford, UK, July 27-30, 2009*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[GHM10]   Birte Glimm, Ian Horrocks, and Boris Motik. Optimized description logic reasoning via core blocking. 2010.

[GZB06]   Christine Golbreich, Songmao Zhang, and Olivier Bodenreider. The foundational model of anatomy in OWL: experience and perspective. *Journal of Web Semantics*, 4(3):181–195, 2006.

[HB91]   Bernhard Hollunder and Franz Baader. Qualifying number restrictions in concept languages. In *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning, KR-91*, pages 335–346, Boston (USA), 1991.

[HHLS86]   Johan Torkel Hastad, Bettina Helfrich, Jeffrey Lagarias, and Claus Peter Schnorr. Polynomial time algorithms for finding integer relations among real numbers. In *Lecture Notes in Computer Science*, volume 1986, pages 105–118. Publisher Springer Berlin / Heidelberg, 1986.

[HKNP94]   Jochen Heinsohn, Daniel Kudenko, Bernhard Nebel, and Hans-Jürgen Profitlich. An empirical analysis of terminological representation systems. *Artificial Intelligence*, 68(2):367–397, 1994.

[HKS06]   Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible $\mathcal{SROIQ}$. In *Proc. of the 10th Int. Conf. on Principles of Knowledge*

Representation and Reasoning (KR 2006), pages 57–67. AAAI Press, 2006.

[HM01a]   Volker Haarslev and Ralf Möller. Optimizing reasoning in description logics with qualified number restrictions. In Carole A. Goble, Deborah L. McGuiness, Ralf Möller, and Peter F. Patel-Schneider, editors, *Description Logics*, volume 49 of *CEUR Workshop Proceedings*, pages 142–151, 2001.

[HM01b]   Volker Haarslev and Ralf Möller. RACER system description. *Lecture Notes in Computer Science*, 2083:701–705, 2001.

[HM04]   Jan Hladik and Jörg Model. Tableau systems for $\mathcal{SHIO}$ and $\mathcal{SHIQ}$. In *Proceedings of the 2004 International Workshop on Description Logics (DL2004)*, 2004.

[Hor97]   Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[Hor02]   Ian Horrocks. Backtracking and Qualified Number Restrictions: Some Preliminary Results. In *In Proceedings of the 2002 Description Logic Workshop*, volume 63, pages 99–106. CEUR, 2002.

[HS01]   Ian Horrocks and Ulrike Sattler. Ontology reasoning in the $\mathcal{SHOQ}$(D) description logic. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI 2001)*, pages 199–204. Morgan Kaufmann, Los Altos, 2001.

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

[HS07]   Ian Horrocks and Ulrike Sattler. A tableau decision procedure for
            $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 39(3):249–276, October
            2007.

[HST99]   Ian Horrocks, Ulrike Sattler, and Stephan Tobies. Practical reason-
            ing for expressive description logics. In *Proceedings of the 6th Interna-
            tional Conference on Logic for Programming and Automated Reasoning
            (LPAR'99)*, pages 161–180. Springer-Verlag, 1999.

[HSV11]   Volker Haarslev, Roberto Sebastiani, and Michele Vescovi. Automated
            reasoning in $\mathcal{ALCQ}$ via SMT. In *23rd International Conference on Au-
            tomated Deduction*, volume 6803, pages 283–298. Lecture Notes in Com-
            puter Science, July 31 - August 5 2011.

[HT99]   Ian Horrocks and Stephan Tobies. Optimisation of terminological rea-
            soning. LTCS-Report LTCS-99-14, LuFG Theoretical Computer Sci-
            ence, RWTH Aachen, 1999.

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

[HTM01]   Volker Haarslev, Martina Timmann, and Ralf Möller. Combining
            tableaux and algebraic methods for reasoning with qualified number
            restrictions. In *Description Logics*, pages 152–161, 2001.

[HW06]   Alexander K. Hudek and Grant E. Weddell. Binary absorption in
            tableaux-based reasoning for description logics. In *Proceedings of the
            18th International Workshop on Description Logics (DL 2006), Lake
            District, UK, May 30-June1*, 2006.

279

[KM06]   Yevgeny Kazakov and Boris Motik. *A Resolution-Based Decision Procedure for $\mathcal{SHOIQ}$*, volume 4130/2006, pages 662–677. Springer Berlin / Heidelberg, 2006.

[Lie06]   Thorsten Liebig. Reasoning with OWL – system support and insights –. Technical Report TR-2006-04, Ulm University, Ulm, Germany, September 2006.

[Lut02]   Carsten Lutz. PSpace Reasoning with the Description Loigc $\mathcal{ALCF}(D)$. *Logic Journal of the IGPL*, 10(5):535–568, 2002.

[MH08]   Boris Motik and Ian Horrocks. Individual reuse in description logic reasoning. In *International Joint Conference on Automated Reasoning (IJCAR 2008)*, pages 242–258, 2008.

[MHWZ06]   Elaine N. Marieb, Katja Hoehn, Patricia Brady Wilhelm, and Nina Zanetti. *Human Anatomy & Physiology: International Edition with Human Anatomy and Physiology Atlas, 7/E*. Pearson Higher Education, 7 edition, 2006.

[Min81]   Marvin Minsky. A framework for representing knowledge. In J. Haugeland, editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 95–128. MIT Press, Cambridge, MA, 1981.

[MM00]   Francesco M.Donini and Fabio Massacci. EXPtime tableaux for $\mathcal{ALC}$. *Artificial Intelligence*, 124(1):87–138, 2000.

[MPSCG08]   Boris Motik, Peter F. Patel-Schneider, and Bernardo Cuenca Grau. OWL 2 web ontology language: Direct semantics. 2008. Latest version available at http://www.w3.org/TR/owl2-semantics/.

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

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

[NF04]   Chokri Ben Necib and Johann-Christoph Freytag. Using ontologies for database query reformulation. In *Proceedings of the 18th Conference on Advances in Databases and Information Systems (ADBIS)*, Budapest, Hungary, 2004.

[NM01]   Natalya Fridman Noy and Deborah L. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical report, Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

[OK96]   Hans Jürgen Ohlbach and Jana Koehler. Reasoning about sets via atomic decomposition. Technical Report TR-96-031, Berkeley, CA, 1996.

[OK97]   Hans Jürgen Ohlbach and Jana Koehler. Role hierarchies and number restrictions. In *Proceedings of the 1997 International Workshop on Description Logics (DL'97)*, 1997.

[OK99]   Hans Jürgen Ohlbach and Jana Koehler. Modal logics description logics and arithmetic reasoning. *Artificial Intelligence*, 109(1-2):1–31, 1999.

[PCS06]   Bijan Parsia, Bernardo Cuenca Grau, and Evren Sirin. From wine to water: Optimizing description logic reasoning for nominals. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR)*, pages 90–99, 2006.

[Pou]   Laleh Roosta Pour. Algebraïc reasoning with the description logic $\mathcal{SHIQ}$. Master's thesis, Concordia University.

[Qui67]   Ross Quillian. *Word concepts: A theory and simulation of some basic semantic capabilities*, volume 12 of *Behavioral Science*, pages 410–430. September 1967.

[RS05]   Alan Rector and Guus Schreiber. Qualified cardinality restrictions QCRs: Constraining the number of values of a particular type for a property. 2005. http://www.w3.org/2001/sw/BestPractices/OEP/QCR.

[RV02]   Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. *AI Commun.*, 15(2):91–110, 2002.

[Sch91]   Klaus Schild. A correspondence theory for terminological logics: preliminary report. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sidney, AU, 1991.

[Sch94]   Andrea Schaerf. Reasoning with individuals in concept languages. *Data and Knowledge Engineering*, 13(2):141–176, 1994.

[Sir06]   Evren Sirin. *Combining Description Logic Reasoning with AI Planning for Composition of Web Services*. PhD thesis, University of Maryland, College Park, June 2006.

[SMH]   Rob Shearer, Boris Motik, and Ian Horrocks. Hermit: A highly-efficient OWL reasoner.

[SPG⁺07]   Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.

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

[TH05]   Dmitry Tsarkov and Ian Horrocks. Ordering heuristics for description logic reasoning. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 609–614, 2005.

[TH06]   Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

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

[Tob00]   Stephan Tobies. The complexity of reasoning with cardinality restrictions and nominals in expressive description logics. *Journal of Artificial Intelligence Research*, 12:199–217, 2000.

[Tob01]   Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, LuFG Theoretical Computer Science, RWTH-Aachen, Germany, 2001.

[TRBH04]   Dmitry Tsarkov, Alexandre Riazanov, Sean Bechhofer, and Ian Horrocks. Using Vampire to reason with OWL. In Sheila A. McIlraith, Dimitris Plexousakis, and Frank van Harmelen, editors, *Proc. of the*

2004 International Semantic Web Conference (ISWC 2004), number 3298 in Lecture Notes in Computer Science, pages 471–485. Springer, 2004.

[WBH⁺05]   Katy Wolstencroft, A. Brass, Ian Horrocks, Phillip Lord, Ulrike Sattler, Robert Stevens, and Daniele Turi. A little semantic web goes a long way in biology. In *4th Int. Semantic Web Conference*, volume 3792, pages 786–800, Ireland, 2005.

[Wu08]   Jiewen Wu. Plan-based Axiom Absorption for Tableaux-based Description Logics Reasoning. Master's thesis, Concordia University, 2008.

[Zuo06]   Ming Zuo. High Performance Absorption Algorithms for Terminological Reasoning in Description Logics. Master's thesis, Concordia University, 2006.

Glossary

$(\mathcal{D})$ Denotes the use of *concrete datatypes* within a DL language.

$\mathcal{A}$ Denotes an ABox.

ABox   Assertional Box.

$\mathcal{ALC}$ The basic Description Logic language.

$\mathcal{ALCQ}$ A DL language extending $\mathcal{ALC}$ with *qualified cardinality restrictions*.

$\mathcal{ALCHOQ}$ A DL language extending $\mathcal{ALC}$ with *role hierarchies*, *nominals*, and *qualified cardinality restrictions*.

$\mathcal{ALCOQ}$ A DL language extending $\mathcal{ALC}$ with *nominals*, and *qualified cardinality restrictions*.

API   Application Programming Interface.

CCG — Compressed Completion Graph.

DDB — Dependency Directed Backtracking.

DL — Description Logic.

$\mathcal{EL}$ — A less expressive DL than $\mathcal{ALC}$ where the only available constructors are $\top$, conjunction, and existential restriction ($\exists$).

FOL — First-Order Logic.

GCI — General Concept Inclusion axiom.

$\mathcal{H}$ — Role hierarchies.

HARD — Hybrid Algebraïc Reasoner for Description Logics.

$\mathcal{I}$ — Inverse roles.

ILP — Integer Linear Programming.

Java SE — Java Standard Edition, also known as J2SE.

KB — A Description Logic Knowledge Base consisting of a TBox and an ABox.

$\mathcal{N}$ Cardinality restrictions.

NNF   Negation Normal Form.

$\mathcal{O}$ Nominals.

OWL   Web Ontology Language.

OWL DL   An OWL specie that has close correspondence with the Description Logic $\mathcal{SROIQ}$.

$\mathcal{Q}$ See QCR.

QCR   Qualified Cardinality Restriction.

$\mathcal{R}$ Denotes an RBox.

ReAl   Reasoning Algebraïcally with Description Logics.

RIA   Role Inclusion Axioms.

$\mathcal{S}$ A DL language extending $\mathcal{ALC}$ with *transitive roles*.

SOTA   State-of-the-art.

$\mathcal{SROIQ}$ A DL language extending $\mathcal{ALC}$ with *transitive role axioms, role hierarchy, nominals, inverse roles,* and *qualified cardinality restrictions.*

$\mathcal{T}$ Denotes a TBox.

TBox   Terminological Box

W3C   World Wide Web Consortium.

XML   Extensible Markup Language.

Appendices

Appendix A

ReAl DL

This appendix complements Chapter 4, which describes a hybrid algebraïc reasoning procedure for $\mathcal{SHOQ}$. Section A.1 gives a reference for the notations defined and used throughout the chapter. Section A.2 shows the definition of a standard tableau for $\mathcal{SHOQ}$, which can be compared to the tableau defined for ReAl DL.

A.1 List of Notations Used

Notation	Explanation
$Q_\mathrm{C}(R)$	defines the set of qualifying concepts for a role $R \in N_\mathrm{R}$. $Q_\mathrm{C}(R)=$ $\{D \mid \forall S.D$ occurs in $C_\mathcal{T}$ with $R \sqsubseteq_* S \in \mathcal{R}\}$ (see also Definition 4.1.1).
$\mathcal{Q}_\mathrm{C}$	denotes the set of all *qualifying concepts* in a KB.
$\mathcal{Q}_\mathrm{C}^{\neg}$	denotes the set of negated *qualifying concepts* in their NNF
$\dot{\neg}_\mathcal{Q}$	denotes a bijection $\dot{\neg}_\mathcal{Q} : Q_\mathrm{C} \longrightarrow Q_\mathrm{C}^{\neg}$ mapping a qualifying concepts with the NNF of its complement
$H(R)$	denotes the set of all sub-roles of R. $H(R) = \{R' \mid R' \sqsubseteq_* R, R' \neq R\}$.

Notation	Explanation
N_Q	denotes the set of qualification names used to refer to *qualifying concepts*. For clarity purposes, qualification names and qualifying concepts are used interchangeably throughout this thesis and $N_Q = Q_C$.
$Q_N(R)$	denotes the set of qualification names in N_Q used to refer to *qualifying concepts* for R. For clarity purposes, qualification names and qualifying concepts are used interchangeably throughout this thesis and $Q_N(R) = Q_C(R)$.
D_R	$D_R = H(R) \cup Q_N(R)$. It denotes the decomposition set for R-fillers (see Definition 4.4.1).
$\mathcal{DS}$	$\mathcal{DS} = \bigcup_{R \in N_R} \mathcal{D}_R \cup N_o$. It denotes a *global decomposition set* (see Definition 4.4.3).
$\mathcal{P}$	$\mathcal{P} = \{P \mid P \subseteq \mathcal{DS}\}$. It denotes a global partitioning on $\mathcal{DS}$ (see Definition 4.4.4).
σ	defines a mapping $\sigma : \mathcal{V} \longrightarrow \mathbb{N}$ between variables and integer numbers. It is used to refer to a solution for ξ_x.
α	defines a mapping $\alpha : \mathcal{V} \longrightarrow \mathcal{P}$ between variables and partition names.
$\mathcal{L}(x)$	denotes a set of concept expressions that must be satisfied by x, $\mathcal{L}(x) \subseteq Clos(\mathcal{T})$.
$\mathcal{L}_E(x)$	denotes a set of in-equations that must have a non-negative integer solution. $\mathcal{L}_E(x) = \xi_x$
$\mathcal{L}_P(x)$	denotes a partition name and is used as a tag for x, $\mathcal{L}_P(x) \subseteq \mathcal{DS}$.
ξ_x	denotes the encoding of *number restrictions*, *qualifications* and *nominals* that must be satisfied for x.
N_R	denotes the set of role names used in a TBox $\mathcal{T}$.
N_Q	denotes the set of *qualifying concepts* occurring in $\mathcal{T}$.
$N_{R'}$	denotes the set of newly introduced role names in a TBox $\mathcal{T}'$.

Notation	Explanation
N_o	denotes the set of nominals occurring in $\mathcal{T}$.
i_x	denotes the domain element $i \in \Delta^{\mathcal{T}}$ represented by the proxy node x.
V_L	denotes the set of variable names each mapped to a partition p such that $L \subseteq p$.
$\xi(L, \geq, n)$	denotes $v_1 + \cdots + v_k \geq n$, where $\{v_1, \ldots, v_k\} \subseteq V_L$ and $L \subseteq \mathcal{DS}$.
$\xi(L, \leq, m)$	denotes $v_1 + \cdots + v_k \leq m$, where $\{v_1, \ldots, v_k\} \subseteq V_L$ and $L \subseteq \mathcal{DS}$.

A.2 Standard Tableau for $\mathcal{SHOQ}$

The following definition describes a standard tableau for the DL $\mathcal{SHOQ}$ as first presented in [HS01]. For the purpose of clarity and ease of comparison with the tableau defined in Section 4.3, the properties handling *concrete datatypes* are ignored.

Definition A.2.1 (Standard $\mathcal{SHOQ}$ Tableau) Given a $\mathcal{SHOQ}$ KB$(\mathcal{T}, \mathcal{R})$, T $= (\mathbf{S}, \mathcal{L}, \mathcal{E})$ defines a tableau for $(\mathcal{T}, \mathcal{R})$ as an abstraction of a model for $\mathcal{T}$. $\mathbf{S}$ is a non-empty set of individuals, $\mathcal{L} : \mathbf{S} \to 2^{clos(\mathcal{T})}$ is a mapping between each individual and a set of concepts, and $\mathcal{E}$: $N_R \to 2^{\mathbf{S} \times \mathbf{S}}$ is a mapping between each role and a set of pairs of individuals in $\mathbf{S}$. For all $s, t \in \mathbf{S}$, $A \in N_C$, $C, D \in clos(\mathcal{T})$, $o \in N_o$, $R, S \in N_R$, and given the definition $R^T(s, C) = \{t \in \mathbf{S} \,|\, \langle s, t \rangle \in \mathcal{E}(R) \text{ and } C \in \mathcal{L}(t)\}$, the following properties must always hold:

1. $C_{\mathcal{T}} \in \mathcal{L}(s)$

2. If $A \in \mathcal{L}(\text{s})$ then $\neg A \notin \mathcal{L}(\text{s})$.

3. If $C \sqcap D \in \mathcal{L}(\text{s})$ then $C \in \mathcal{L}(s)$ and $D \in \mathcal{L}(s)$.

4. If $C \sqcup D \in \mathcal{L}(\text{s})$ then $C \in \mathcal{L}(s)$ or $D \in \mathcal{L}(s)$.

5. If $\langle s, t \rangle \in \mathcal{E}(R)$ and $R \sqsubseteq_* S \in \mathcal{R}$, then $\langle s, t \rangle \in \mathcal{E}(S)$.

6. If $\forall S.C \in \mathcal{L}(s)$ and $\langle s,t \rangle \in \mathcal{E}(S)$ then $C \in \mathcal{L}(t)$.

7. If $\forall S.C \in \mathcal{L}(s)$ and $\langle s,t \rangle \in \mathcal{E}(R)$ with $R \sqsubseteq_* S$ and $R \in N_{R+}$ then $\forall R.C \in \mathcal{L}(t)$.

8. If $(\geq nR.C) \in \mathcal{L}(s)$ then $\#R^T(s,C) \geq n$.

9. If $(\leq mR.C) \in \mathcal{L}(s)$ then $\#R^T(s,C) \leq m$.

10. If $\{\leq nR.C, \geq nR.C\} \cap \mathcal{L}(s) \neq \emptyset$, and $\langle s,t \rangle \in \mathcal{E}(R)$, then $\{C, \neg C\} \cap \mathcal{L}(t) \neq \emptyset$.

11. For each $o \in N_o$, if $\{o\} \in \mathcal{L}(s) \cap \mathcal{L}(t)$, then $s = t$.

Appendix B

HARD

This appendix complements Chapter 6, which describes the implementation of the prototype reasoner (HARD). Section B.1 shows the overall architecture of the tableau reasoner. Section B.2 gives examples of failed test cases.

B.1   The Tableau Reasoner

Figure 77 shows a collaboration diagram between the Tableau Reasoner, the Clash Handler and the Constraint Solver during an expansion of a CCG for deciding a KB consistency.
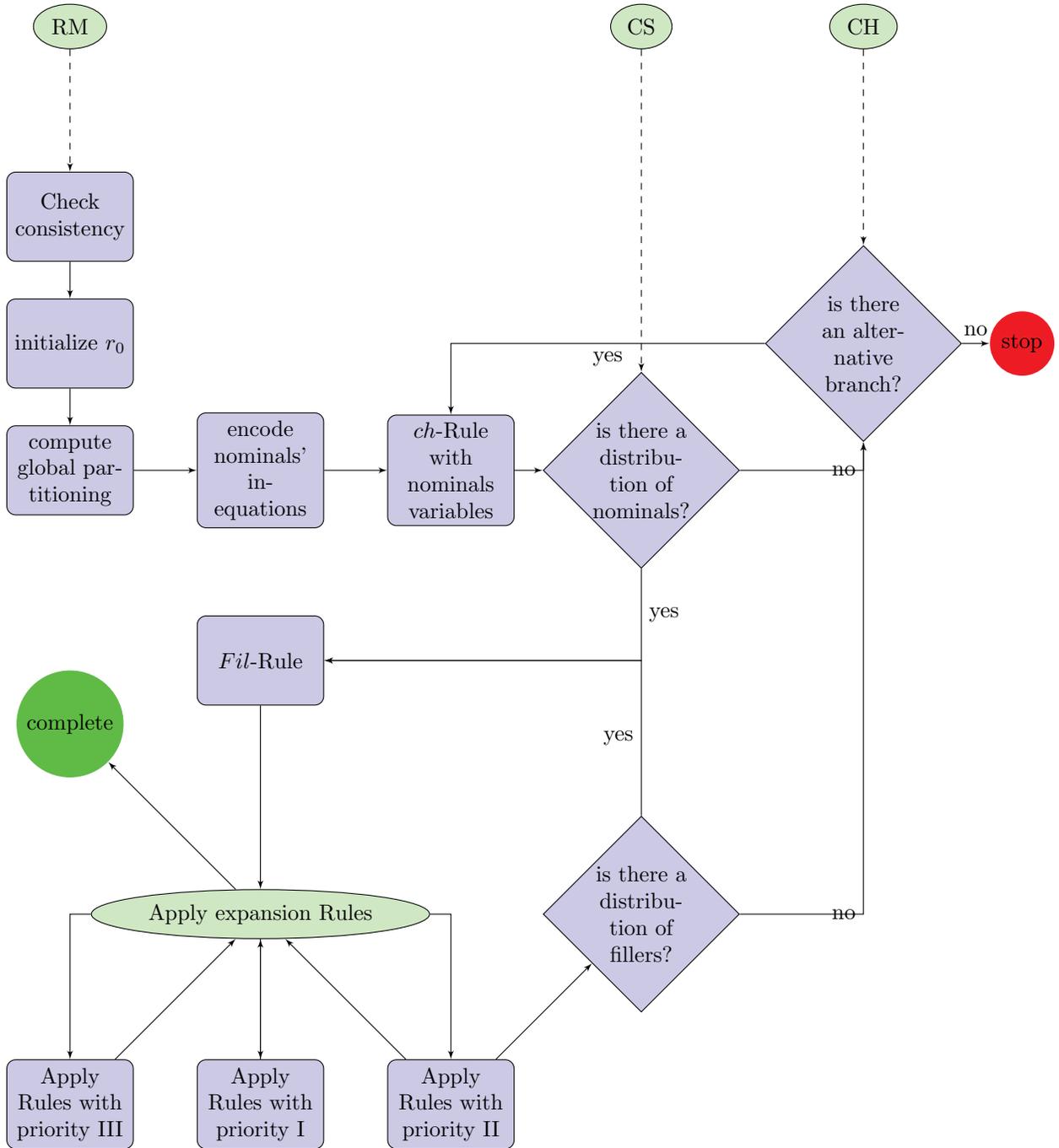
Figure 77: Collaboration diagram during a consistency check highlighting the interaction between the Reasoner Manager (RM), the Constraint Solver (CS), and the Clash Handler (CH) in the cases of arithmetic clashes due to nominals.

B.2 Failing Test Cases

This section is intended to give some sample test cases where the implementation of the Simplex procedure from [Far08] failed to return an answer due to some bugs.

Test case A - wrong results The Simplex module cannot find a solution for a feasible set of in-equation. This bug can be systematically reproduced with the following test case:

$$C_{\text{nominals}-\mathcal{ALCOQ}} \quad \sqsubseteq \geq jR.E$$
$$E \quad \equiv \{o_1, o_2, \ldots, o_i\}$$

where $j = i = 10$. However a solution is found if $j = 9$ and $i = 10$.

Test case B - A `java.lang.NumberFormatException` is encountered when initializing Simplex constraints. This error can be systematically reproduced with the following test case:

$$\leq 10R_1 \sqcap \geq 30R_2 \sqcap \geq 30R_3 \sqcap \geq 30R_4 \sqcap \geq 30R_5 \tag{48}$$

On the other hand, it is not encountered with the same test case using a different numbers

$$\leq 5R_1 \sqcap \geq 10R_2 \sqcap \geq 10R_3 \sqcap \geq 10R_4 \sqcap \geq 10R_5 \tag{49}$$

This problem is mainly due to the representation of the numbers (30) as Simplex constraints and was not encountered if the lpSolver is selected as a constraint solver.

Appendix C

Evaluation

C.1   Test Cases for QCRs

The effect of increased numbers used in QCRs is tested using the concept $C_{\mathcal{ALCQ}}$ defined using the DL $\mathcal{ALCQ}$ as follows:

$$C_{\mathcal{ALCQ}} \sqsubseteq\ \geq 2ir.(A \sqcup B) \sqcap\ \leq ir.A \sqcap\ \leq ir.B \sqcap\ (\leq (i-1)r.\neg A \sqcup\ \leq jr.\neg B)$$

Recall from Section 7.2.1 that the satisfiability of $C_{\mathcal{ALCQ}}$ depends on the value of j; if $j \geq i$ then $C_{\mathcal{ALCQ}}$ becomes satisfiable. Otherwise if $j \leq (i-1)$ then $C_{\mathcal{ALCQ}}$ becomes unsatisfiable. Figures 78-80 show the effect of satisfiable cases versus unsatisfiable cases on the reasoning performance of Fact++, Pellet, and Hermit. $C_{\mathsf{SAT-Lin-}\mathcal{ALCQ}}$ refers to a satisfiable cases, and $C_{\mathsf{UnSAT-Lin-}\mathcal{ALCQ}}$ refers to an unsatisfiable case. In both cases, the values of the numbers used in QCRs are increased linearly.
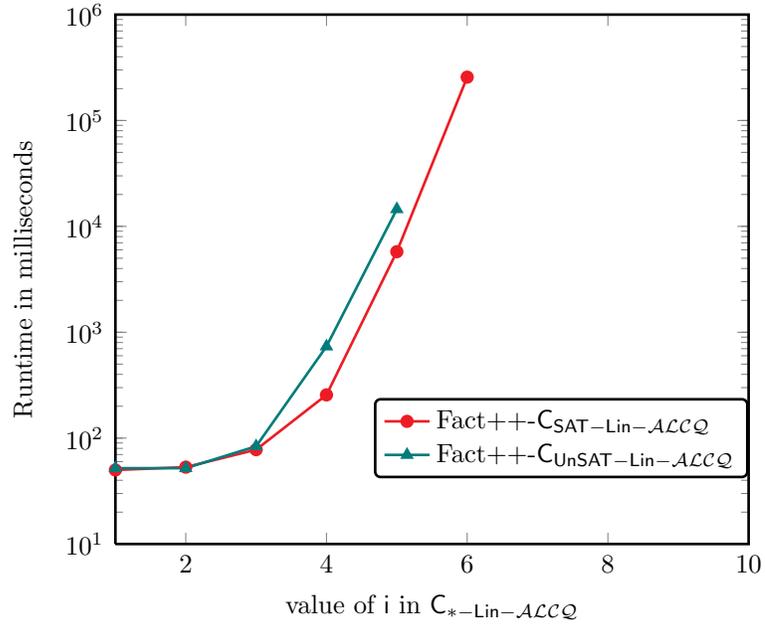
Figure 78: Effect of the satisfiability of $C_{*-Lin-\mathcal{ALCQ}}$ on the runtime performance of Fact++.
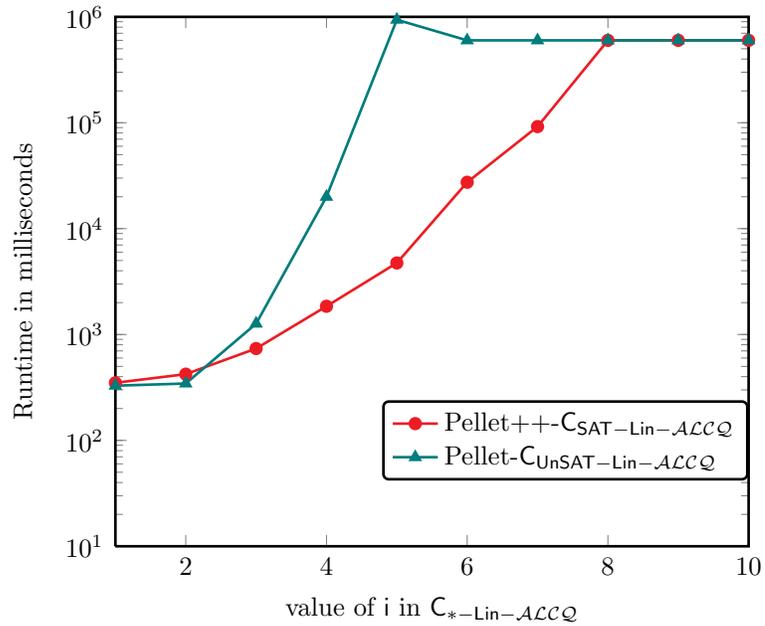

Figure 79: Effect of the satisfiability of $C_{*-Lin-\mathcal{ALCQ}}$ on the runtime performance of Pellet.
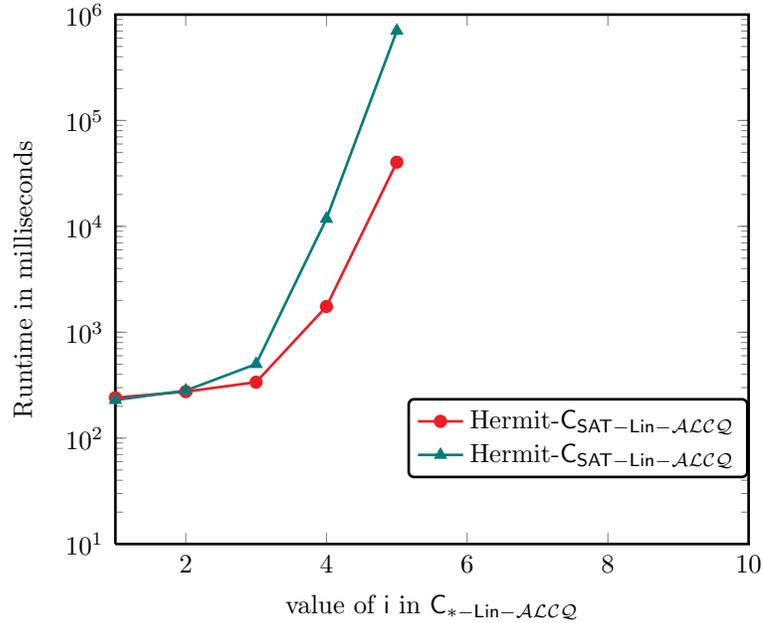
Figure 80: Effect of the satisfiability of $\mathsf{C}_{*-\mathsf{Lin}-\mathcal{ALCQ}}$ on the runtime performance of Hermit.

C.2 Test Cases With Real World Ontologies

This section provides additional information about real world ontology adaptations (from Section 7.2.2) tested with HARD reasoner. For each test case, the TBox used is repeated as well as the role hierarchy. Also, the extended role hierarchy after applying the preprocessing algorithm is shown.

C.2.1 The WINE ontology

The TBox used with the WINE ontology test case in Section 7.2.2.2 is shown in Figure 81, the original role hierarchy as well as the extended one are shown in Figure 82. Figure 83 shows a CCG representing a model for the concept IceWine.

Wine	$\sqsubseteq \; = 1\mathrm{hasBody.WineBody} \sqcap \; = 1\mathrm{hasColor.WineColor}$
	$\sqcap \; = 1\mathrm{hasFlavor.WineFlavor} \sqcap \; = 1\mathrm{hasSugar.WineSugar}$
DessertWine	$\equiv \mathrm{Wine} \sqcap \forall \mathrm{hasSugar.\{offdry, sweet\}}$
LateHarvest	$\sqsubseteq \mathrm{Wine} \sqcap \forall \mathrm{hasFlavor.\{moderate, strong\}} \sqcap \forall \mathrm{hasSugar.\{sweet\}}$
IceWine	$\equiv \mathrm{DessertWine} \sqcap \mathrm{LateHarvest} \sqcap \exists \mathrm{hasColor.\{white\}}$
IceWine	$\sqsubseteq \forall \mathrm{hasBody.\{full, medium\}} \sqcap \forall \mathrm{hasFlavor.\{moderate, strong\}}$
WineDescriptor	$\equiv \mathrm{WineColor} \sqcup \mathrm{WineTaste}$
WineTaste	$\sqsubseteq \mathrm{WineDescriptor}$
WineColor	$\equiv \{\mathrm{red, rose, white}\}$
WineBody	$\equiv \{\mathrm{full, light, medium}\}$
WineFlavor	$\equiv \{\mathrm{delicate, moderate, strong}\}$
WineSugar	$\equiv \{\mathrm{dry, offdry, sweet}\}$

Figure 81: TBox axioms in the WINE ontology.

(a) Role hierarchy for the WINE ontology. (b) Role hierarchy for the WINE ontology after applying the preprocessing algorithm.
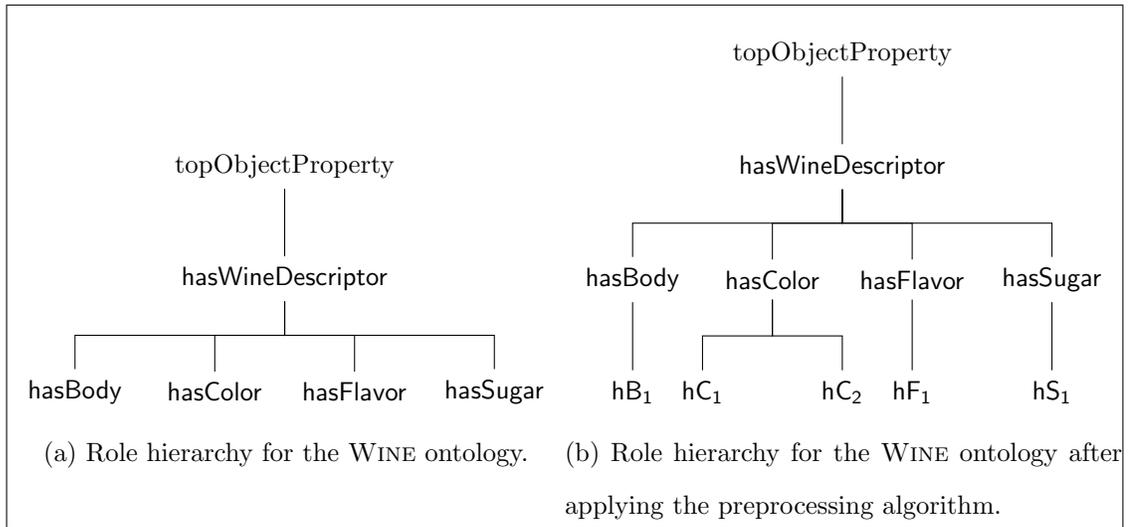
Figure 82: Role hierarchy within the RBox $\mathcal{R}$ for the WINE ontology.

$\mathcal{L}(a) = \{\text{IceWine}, \text{Wine}, = 1hB_1, \forall hB_1.\text{WineBody}, = 1hC_1, \forall hC_1.\text{WineColor}, = 1hF_1,$
$\forall hF_1.\text{WineFlavor}, = 1hS_1, \forall hS_1.\text{WineSugar}, \text{DessertWine}, \forall \text{hasSugar}.\{\text{offDry}, \text{sweet}\}, \geq 1.hC_2,$
$\forall hC_2.\{\text{white}\}, \forall \text{hasBody}.\{\text{full}, \text{medium}\}, \text{LateHarvest}, \forall \text{hasFlavor}.\{\text{moderate}, \text{strong}\},$
$\forall \text{hasSugar}.\{\text{sweet}\}$

$\mathcal{L}(b) = \{\text{WineColor}, \{\text{white}\}\}$

$\mathcal{L}(e) = \{\text{WineBody}, \{\text{full}, \text{medium}\}\}$

$\mathcal{L}(d) = \{\text{WineFlavor}, \{\text{moderate}, \text{strong}\}, \{\text{strong}\}\}$ $\mathcal{L}(c) = \{\text{WineSugar}, \{\text{sweet}\}, \{\text{offDry}, \text{sweet}\}\}$
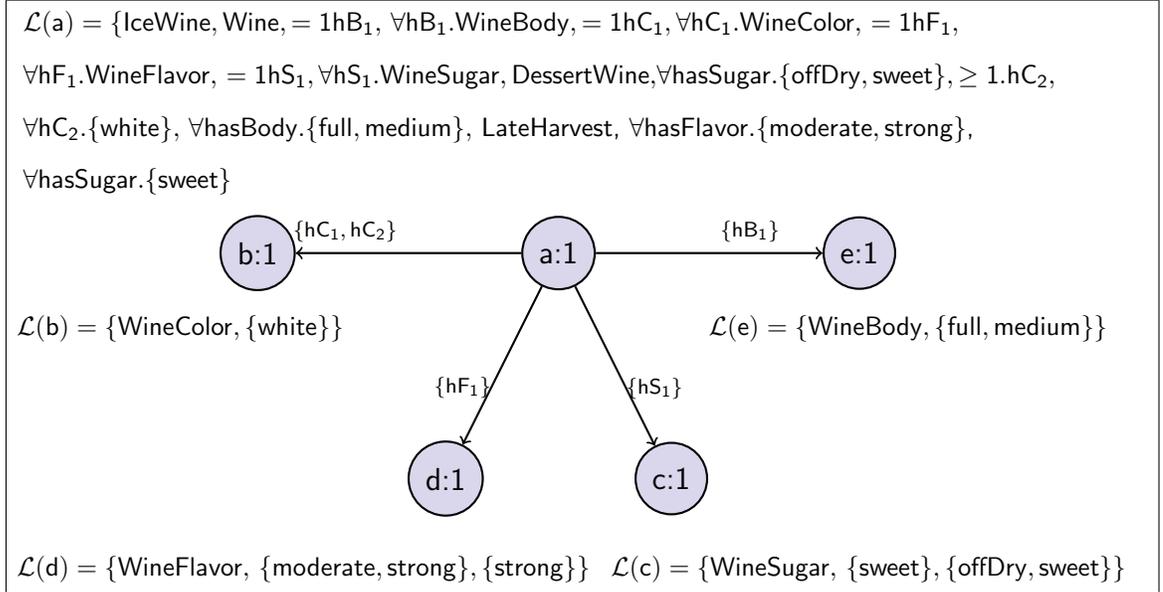
Figure 83: Clash free compressed completion graph for WINE-C_{IceWine} test case.

C.2.2   The KOALA ontology

The TBox axioms for the KOALA ontology test case in Section 7.2.2.3 are shown in Figure 84, the original role hierarchy as well as the extended one are shown in Figure 86. Figure 85 shows a CCG representing a model for the concept KoalaWithPhD. Figure 87 shows a CCG representing a model for the concept MaleStudentWithnDaughters, and Figure 88 shows a *compressed completion graph* representing a model for the concept (KoalaWithPhD ⊓ MaleStudentWithnDaughters).

Notice that the proxy nodes having a nominal in their label (e.g., node c, e) represent a partition with only one individual, the nominal in the label. The completion graph does not need to distinguish between a proxy node representing a nominal and a proxy node representing domain elements. It is the `cardinality` value, which is equal to 1 in case of a nominal node (e.g., (e:1)), of the `ProxyNode` object[1] which represents the nominals semantics.

---

[1]See Figure 41 in Section 6.5 for more details on the `ProxyNode` object.

KoalaWithPhD	$\equiv$	Koala $\sqcap$ $\exists$hasDegree.$\{$PhD$\}$
Koala	$\sqsubseteq$	Marsupials $\sqcap$ $\exists$hasHabitat.Forest
Marsupials	$\sqsubseteq$	Animal
Animal	$\sqsubseteq$	≥ 1 hasHabitat.Habitat $\sqcap$ $= 1$ hasGender.Gender
MaleStudentWith3Daughters	$\sqsubseteq$	$\forall$hasGender.Male $\sqcap$ $= 3$ hasChildren.Female
MaleStudentWith3Daughters	$\sqsubseteq$	Student
Student	$\sqsubseteq$	Person $\sqcap$ $\exists$hasHabitat.University
Person	$\sqsubseteq$	Animal
Female	$\sqsubseteq$	$\exists$hasGender.$\{$female$\}$
Male	$\sqsubseteq$	$\exists$hasGender.$\{$male$\}$
Gender	$\equiv$	$\{$male, female$\}$
MaleStudentWithnDaughters	$\sqsubseteq$	hasGender.Male $\sqcap$ $= n$ hasChildren.Female
MaleStudentWithnDaughters	$\sqsubseteq$	Student

Figure 84: TBox axioms in the KOALA ontology.

$\mathcal{L}(a) = \{$ Koala-C_1, Koala, ≥ 1hD_1, $\forall$hD_1.$\{$PhD$\}$, Marsupials, ≥ 1hH_1, $\forall$hH_1.Forest, Animal, ≥ 1hH_2, $\forall$hH_2.Habitat, $= 1$hG_1, $\forall$hG_1.Gender$\}$
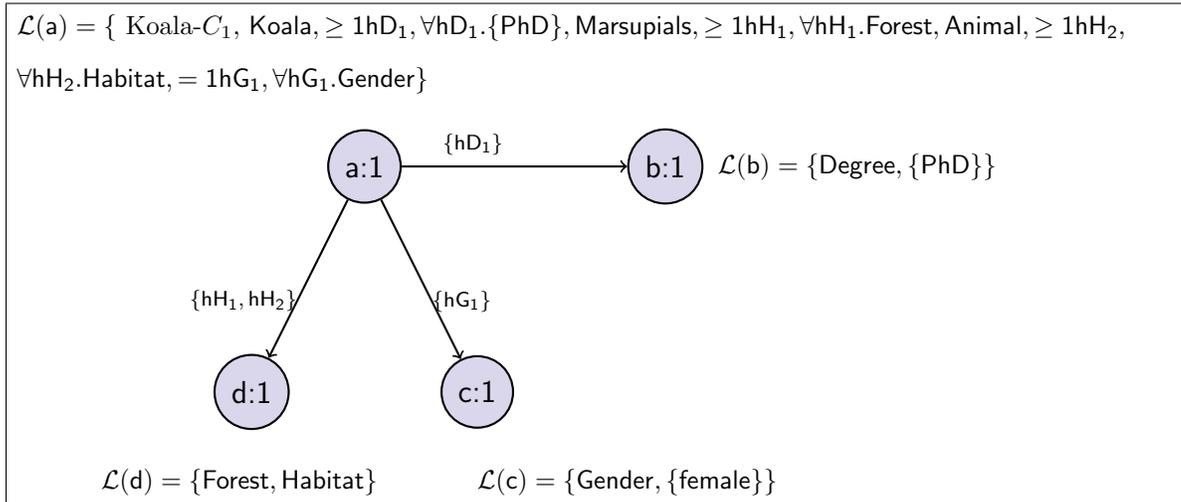

Figure 85: Clash free CCG for KOALA-C_1 representing a pre-model for the concept KoalaWithPhD. Note that all proxy nodes represent partitions with one element.
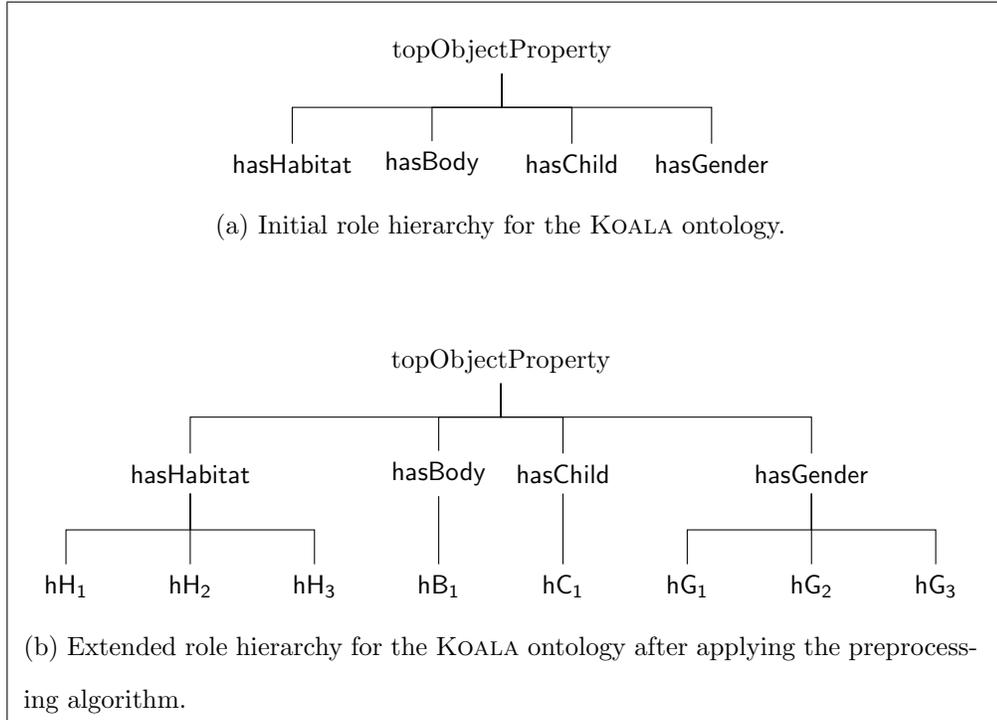
Figure 86: Role hierarchy within the RBox $\mathcal{R}$ for the KOALA ontology.
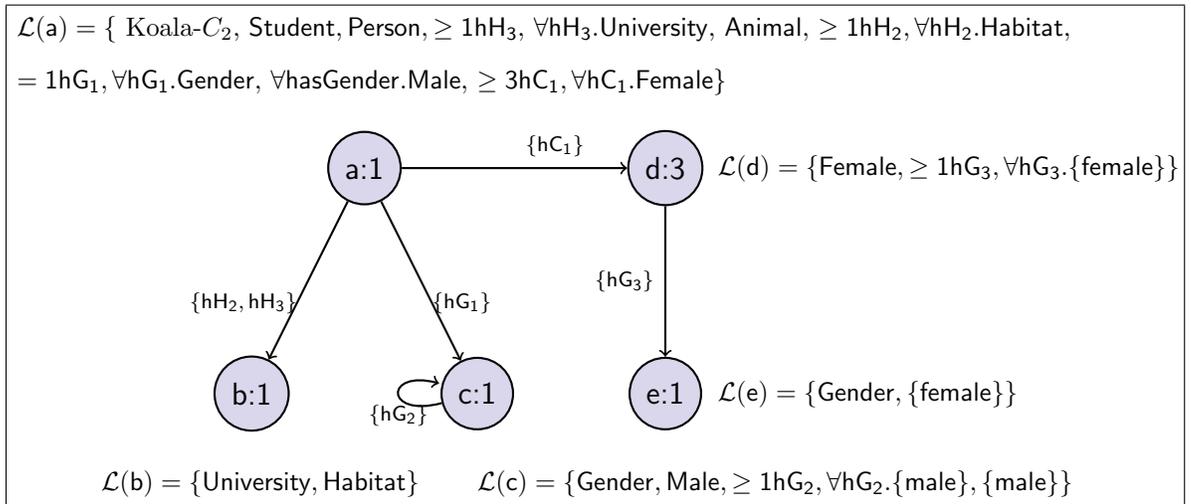
Figure 87: Clash free CCG for KOALA-$C_2(n = 3)$ representing a pre-model for the concept MaleStudentWithnDaughters. Note that the proxy node d represents the partition for hC_1 which consists of 3 domain elements members of the concept Female.

$\mathcal{L}(a) = \{$ Koala-C_{1-2}, Koala, $\geq 1hD_1, \forall hD_1.\{PhD\}$, Marsupials, $\geq 1hH_1, \forall hH_1.$Forest, Animal, $\geq 1hH_2,$
$\forall hH_2.$Habitat, $= 1hG_1, \forall hG_1.$Gender, Student, Person, $\geq 1hH_3, \forall hH_3.$University,$\forall$hasGender.Male,
$\geq 3hC_1, \forall hC_1.$Female$\}$

$\mathcal{L}(f) = \{$Degree, $\{PhD\}\}$

$\mathcal{L}(d) = \{$Female, $\geq 1hG_2, \forall hG_2.\{female\}\}$

$\mathcal{L}(e) = \{$Gender, $\{female\}\}$

$\mathcal{L}(b) = \{$Forest, Habitat, University$\}$ $\mathcal{L}(c) = \{$Male, $\geq 1hG_3, \forall hG_3.\{male\}, \{male\}\}$
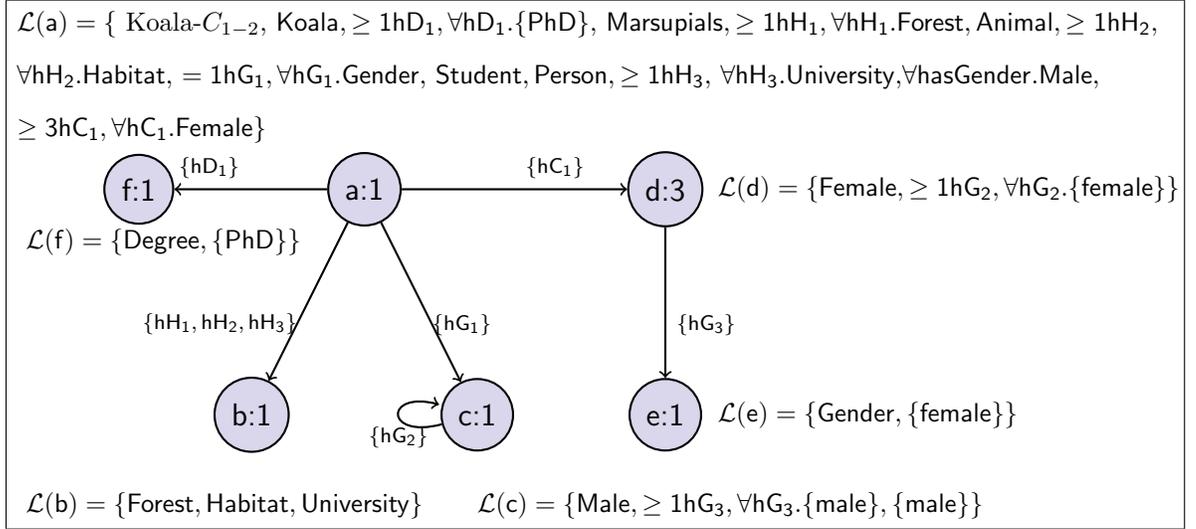
Figure 88: Clash free CCG for KOALA-$C_{1-2}(n=3)$ representing a pre-model for (KoalaWithPhD $\sqcap$ MaleStudentWithnDaughters). The proxy node b represents a partition where all hasHabitat-fillers intersect. This can be done because no disjoint relation exists between Forest and University.

C.3 Test Cases With Synthetic Ontologies

The effect of using QCRs in nested expressions on the performance of reasoning is tested using the satisfiability of $C_{nested-\mathcal{ALCHQ}}$ as introduced in Section 7.2.2.6. In this case the qualification on the role fillers of R_1 includes a QCRs. When the nominal interaction is enabled, the test case is referred to as $C_{nested-\mathcal{ALCHOQ}}$. Figure 89 shows a CCG for $C_{SAT-nested-\mathcal{ALCHOQ}}$.

$$C_{nested-\mathcal{ALCHOQ}} \sqsubseteq \geq nR_1.(A \sqcap \geq nR_2.B) \sqcap \geq nR_{1a}.(A \sqcap \geq nR_{2a}.B) \sqcap \leq mR.\top \sqcap \forall R.D$$
$$D \equiv \{o_1, o_2, \ldots, o_n\}$$
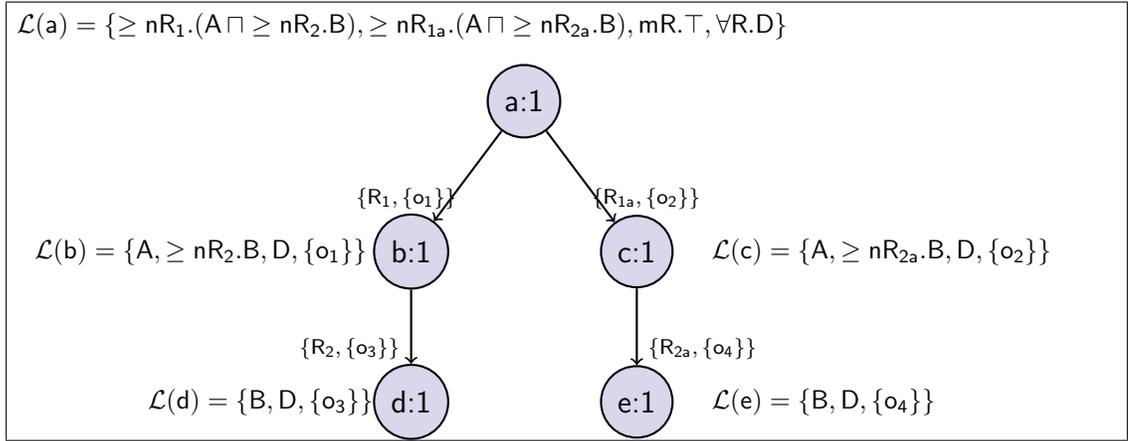
Figure 89: Clash free compressed completion graph for $C_{\mathsf{SAT-nested}-\mathcal{ALCHOQ}}$.

The effect of using cycles in concept descriptions on the performance of reasoning is tested using the satisfiability of $C_{\mathsf{cyclic}-\mathcal{ALCHQ}}$ when there is no *nominals* interaction.

$$C_{\mathsf{cyclic}-\mathcal{ALCHQ}} \sqsubseteq\; \geq nR_1.C \sqcap\; \geq nR_2.C \sqcap\; \geq nR_{1a}.C \sqcap\; \geq nR_{2a}.C \sqcap\; \leq mR.\top \sqcap \forall R.\top$$

$$C \sqsubseteq C_{\mathsf{cyclic}-\mathcal{ALCHQ}}$$

Figures 91 and 90 show examples of a CCG for $C_{\mathsf{SAT-cyclic}-\mathcal{ALCHQ}}$, it is easy to see how cycles are handled by the re-use of proxy nodes.
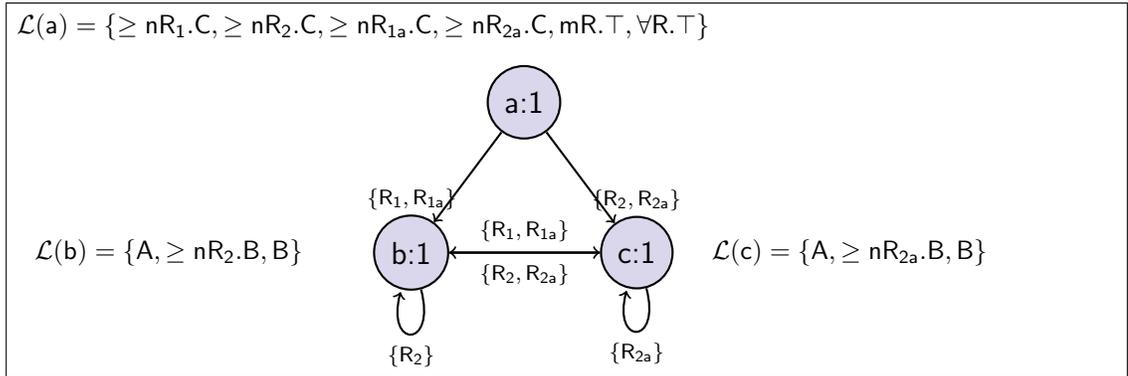


Figure 90: Clash free compressed completion graph for $C_{\mathsf{cyclic}-\mathcal{ALCHQ}}$.

$\mathcal{L}(\mathsf{a}) = \{\geq n\mathsf{R}_1.\mathsf{C}, \geq n\mathsf{R}_2.\mathsf{C}, \geq n\mathsf{R}_{1a}.\mathsf{C}, \geq n\mathsf{R}_{2a}.\mathsf{C}, m\mathsf{R}.\top, \forall\mathsf{R}.\top\}$

a:1

$\{\mathsf{R}_1, \mathsf{R}_{1a}, \mathsf{R}_2, \mathsf{R}_{2a}\}$

c:1 $\mathcal{L}(\mathsf{c}) = \{\mathsf{C}\}$

$\{\mathsf{R}_1, \mathsf{R}_{1a}, \mathsf{R}_2, \mathsf{R}_{2a}\}$
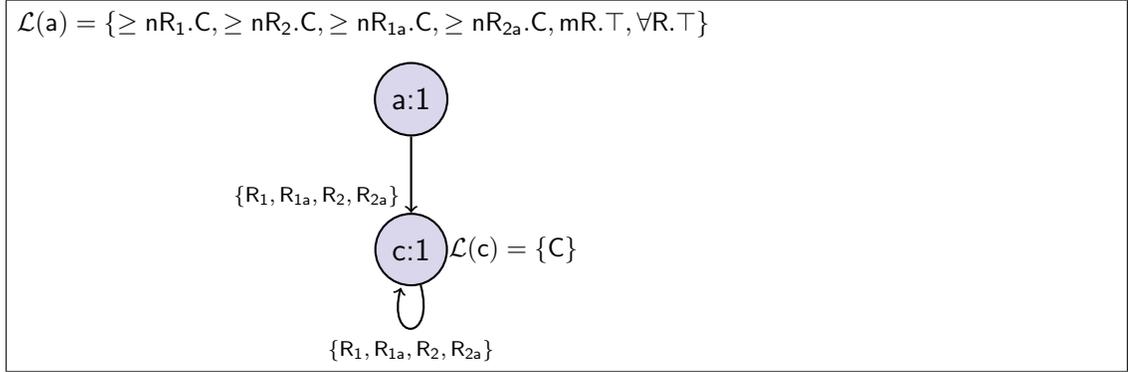
Figure 91: Clash free compressed completion graph for $\mathsf{C}_{\mathsf{cyclic}-\mathcal{ALCHQ}}$.

C.4   Optimizations Effects

This section provides additional information about the test cases used to analyze the effect of the LAB optimizations in Section 7.3.2. Table 28 summarizes the characteristics of the test cases used. Table 29 shows the performance results when one of more LAB optimization is turned OFF. Table 30 shows the speed up factor of the optimizations used.

Test case	size of $\mathcal{DS}$ ADR + ANO
$\mathsf{C}_{\mathsf{SAT}-\mathsf{lin}-\mathcal{ALCQ}(\mathsf{i}=10)}$	4
$\mathsf{C}_{\mathsf{UnSAT}-\mathsf{lin}-\mathcal{ALCQ}(\mathsf{i}=10)}$	4
$\mathsf{C}_{\mathsf{QCR}-\mathcal{ALCQ}(\mathsf{i}=4)}$	10
$\mathsf{C}_{\mathsf{QCR}-\mathsf{var}-\mathcal{ALCQ}(\mathsf{i}=4)}$	8
$\mathsf{C}_{\mathsf{QCR}-\mathsf{disjunctive}-\mathsf{atMost}-\mathcal{ALCQ}(\mathsf{i}=4)}$	6
$\mathsf{C}_{\mathsf{QCR}-\mathsf{disjunctive}-\mathsf{Least}-\mathcal{ALCQ}(\mathsf{i}=4)}$	6
$\mathsf{C}_{\mathsf{QRatio}-\mathcal{ALCQ}(\mathsf{i}=5)}$	14
$\mathsf{C}_{\mathsf{Back}-\mathsf{disjunctive}-\mathcal{ALCQ}(\mathsf{i}=10)}$	11
$\mathsf{C}_{\mathsf{UnSAT}-\mathsf{nested}-\mathcal{ALCHOQ}}$	5

Table 28: Characteristics of the synthetic test cases used to evaluate the LAB optimizations.

Test case	LAB	LA-	L-B	-AB	- -B	-A-	L- -	- - -
$C_{SAT-lin-\mathcal{ALCQ}(i=10)}$	313	484	422	1046	1172	453	564	954
$C_{UnSAT-lin-\mathcal{ALCQ}(i=10)}$	1385	734	438	313	421	3890	751	3687
$C_{QCR-\mathcal{ALCQ}(i=4)}$	10867	49734	157484	49531	23672	50843	TO	TO
$C_{QCR-var-\mathcal{ALCQ}(i=4)}$	69585	TO	236001	246282	123171	TO	TO	TO
$C_{QCR-disjunctive-atMost-\mathcal{ALCQ}(i=4)}$	2218	21296	10266	11625	10188	11298	2094	2531
$C_{QCR-disjunctive-Least-\mathcal{ALCQ}(i=4)}$	76182	423	49031	15219	16155	173265	25219	TO
$C_{QRatio-\mathcal{ALCQ}(i=5)}$	11578	15468	ERR[2]	TO	TO	TO	TO	TO
$C_{Back-disjunctive-\mathcal{ALCQ}(i=10)}$	72553	85937	28173	66015	27985	76750	30062	28797
$C_{UnSAT-nested-\mathcal{ALCHOQ}}$	516	452	1954	421	1952	421	2000	1984

Table 29: Runtimes in milliseconds with the synthetic test cases where one or more LAB optimization(s) is(are) turned OFF.

Test case	S(LAB)	S(LA)	S(LB)	S(AB)	S(B)	S(A)	S(L)
$C_{SAT-lin-\mathcal{ALCQ}(i=10)}$	3	4	1	2	2	1	3
$C_{UnSAT-lin-\mathcal{ALCQ}(i=10)}$	3	0	3	1	1	0	0
$C_{QCR-\mathcal{ALCQ}(i=4)}$	55	2	5	55	5	14	5
$C_{QCR-var-\mathcal{ALCQ}(i=4)}$	9	2	9	9	9	3	4
$C_{QCR-disjunctive-atMost-\mathcal{ALCQ}(i=4)}$	1	5	5	1	10	5	5
$C_{QCR-disjunctive-Least-\mathcal{ALCQ}(i=4)}$	8	0	2	0	0	1	0
$C_{QRatio-\mathcal{ALCQ}(i=5)}$	52	52	52	52	1	N/A	52
$C_{Back-disjunctive-\mathcal{ALCQ}(i=10)}$	0	0	1	0	1	0	1
$C_{UnSAT-nested-\mathcal{ALCHOQ}}$	4	4	1	4	1	4	1

Table 30: Speed up factor of the LAB optimizations used for enhanced Back-jumping.