Consequence-based Algebraic Reasoning for $\mathcal{SHOQ}$

Nikoo Zolfaghar Karahroodi

A Thesis

In the Department of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of

Doctor of Philosophy (Computer Science)

at Concordia University

Montréal, Quebec, Canada

January 2024

© Nikoo Z. Karahroodi, 2024

CONCORDIA UNIVERSITY

SCHOOL OF GRADUATE STUDIES

This is to certify that the thesis prepared

By: **Nikoo Zolfaghar Karahroodi**

Entitled: **Consequence-based Algebraic Reasoning for $\mathcal{SHOQ}$**

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

Doctor of Philosophy (Computer Science)

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

Signed by the final examining committee:

_____ Chair

Dr. Adel Hanna

_____ External Examiner

Dr. Grant Weddell

_____ Examiner

Dr. Wahab Hamou-Lhadj

_____ Examiner

Dr. Gregory Butler

_____ Examiner

Dr. Leila Kosseim

_____ Supervisor

Dr. Volker Haarslev

Approved by _____

Dr. Leila Kosseim, Graduate Program Director

April 3, 2024

_____

Dr. Mourad Debbabi, Dean of Engineering and Computer Science

ABSTRACT

Consequence-based Algebraic Reasoning for $\mathcal{SHOQ}$

Nikoo Z. Karahroodi, Ph.D.

Concordia University, 2024

Qualified Cardinality Restrictions (QCRs) and nominals are the two constructors in OWL 2 DL to apply numerical restrictions on domain concepts and relations. Utilizing these constructors in designing real-world ontologies is unavoidable in many domains, particularly in modelling structures with complex objects. Most existing DL reasoners employ arithmetically uninformed processes to reason about these numeric restrictions by exploring all possible cases.

Meanwhile, Consequence-Based (CB) reasoning algorithms have proven to have a phenomenal performance in practice. However, they have yet to be extended for the expressive DL $\mathcal{SHOQ}$ - a DL that supports *named individuals* and *cardinality restrictions*. This research presents a novel consequence-based algorithm to classify $\mathcal{SHOQ}$ while handling the arithmetic interaction between QCRs and Nominals using atomic decomposition and integer linear programming. The proposed calculus can classify the whole ontology in one round. We have implemented our calculus in a prototype reasoner called CARON.

Empirical evaluation of our implementation demonstrates that CARON outperforms existing reasoners in handling numerical restrictions. At the same time, it offers a competitive performance compared to other state-of-the-art systems. Our results also show that CARON nicely complements existing reasoners for handling numerical restrictions since it provides an arithmetically informed process for handling these constructors. Accordingly, the calculus and implementation presented in this thesis are critical to improving practical reasoning with expressive DLs, including numerical restrictions.

Acknowledgements

My most profound appreciation goes to Professor Volker Haarslev, whose unwavering support and guidance have been the bedrock of my journey. From dissecting the intricacies of description logic in countless hours of discussions to offering sage advice as a life mentor, his belief in me has been a constant source of strength. When I was hesitant to tackle the implementation of integer linear programming on my own, Professor Haarslev's insightful feedback steered me in the right direction and instilled in me the confidence to navigate future complexities. His faith in my potential not only propelled me through tough situations but also paved the way for building a career based on ontologies. I am forever grateful for his mentorship, which has shaped my academic pursuits and taught me invaluable life lessons.

I sincerely appreciate our graduate program director, Professor Leila Kosseim, as her belief in me pulled me back from the brink of abandoning my lifelong dream. Her vote of confidence rekindled a fire within me, which got me through that final gruelling step.

I owe a deep debt of gratitude to my manager, Sarah Deyoung, for her encouragement, support, and understanding in pursuing this personal endeavour. Beyond words of encouragement, she provided me with the flexibility and resources I needed to succeed. Thanks to her, I was able to balance my responsibilities and make this dream a reality.

My biggest thanks go to my family for their boundless love, support, and encouragement. My husband Khalegh, my pillar of strength, deserves immense gratitude for his unconditional love and unyielding support. Whether it was managing life events to give me the time and space to focus on my studies or simply holding my hand through challenging moments, his unwavering presence was my safe harbour. This work is also dedicated to my son Hossein, whose infectious laughter and innocent curiosity provided

a constant source of joy and motivation throughout my studies. Witnessing him blossom from a baby to a young man during this time has been one of life's greatest blessings, and dedicating this achievement to him fills me with immense pride.

Contents

List of Figures

List of Tables

List of Examples

List of Abbreviations

ARM	**AR**ithmetic **M**odule
BCP	**B**oolean **C**onstraint **P**ropagation
CB	**C**onsequence **B**ased
CG	**C**olumn **G**eneration
CQ	**C**ounting **Q**uantifier
DL	**D**escription **L**ogic
DLP	**D**escription **L**ogic **P**rogramming
FOL	**F**irst **O**rder **L**ogic
GCI	**G**eneral **C**oncept **I**nclusion
ILP	**I**nteger **L**inear **P**rogramming
KR	**K**nowledge **R**epresentation
NNF	**N**egation **N**ormal **F**orm
OBDA	**O**ntology **B**ased **D**ata **A**ccess
OWL	**W**eb **O**ntology **L**anguage
QCR	**Q**ualified **C**ardinality **R**estriction
RIA	**R**ole **I**nclusion **A**xiom

Glossary

arithmetic solution An arithmetic solution $\xi(v)$ is a set of tuples $\langle \rho, \sigma_\rho \rangle$ produced by ARM corresponding to $\mathcal{A}(v)$. The *partition* ρ is a set of unary $A(x)$ and binary predicates $R(x, y)$ and $\sigma_p \in \mathbb{N}, \sigma_p \geq 1$ is the cardinality of the partition. 77, 85

arithmetic label arithmetic label is the ordered union of local and global arithmetic labels. 74, 85, 100, 101, 106

completion graph A Digraph constructed by our CB algorithm. 72, 73

conflict set A *Conflict Set* set ($\mathcal{CS}$) is returned by ARM if an arithmetic label $\mathcal{A}(v)$ is unsatisfiable. A $\mathcal{CS}$ is a subset $\mathcal{A}(v)$ constraints, including disjunct CQs of the form $\bigvee Q \in \mathcal{Q}(v)$ and clauses of the form $K \rightarrow M \in \mathcal{C}(v)$ that their integration causes unsatisfiability. 77, 88, 100, 111

decomposition set An ordered set of all predicates occurring in a node's arithmetic label $\mathcal{A} * (v)$. 101, 103, 105

DL-literal A DL-atom or a DL-quantifier. 65

DL-atom A unary predicate $C \in S_{\mathcal{U}}$ or a constant $o \in S_{\mathcal{C}}$. 65

DL-quantifier Either an *at-least* ($\exists_{\geq n} y$), an *at-most* ($\exists_{\leq m} y$) or a *universal* quantifier ($\forall y$). 64

global arithmetic label A *global-arithmetic-label* is a tuple $\mathcal{A} = (\mathcal{Q}, \mathcal{C})$, where $\mathcal{Q}$ is the set of all CQs over unary predicates and $\mathcal{C}$ is a set of normal clauses. 76

local arithmetic label A *local arithmetic label* $\mathcal{A}(v)$ is defined for each node $v \in \mathcal{V}$ as a tuple $\mathcal{A}(v) = (\mathcal{Q}(v), \mathcal{C}(v))$, where $\mathcal{Q}(v)$ is a set of disjunction of CQs and $\mathcal{C}(v)$ is a set of normal clauses. 75

node atom A node atom is a unary predicate $C \in S_{\mathcal{U}} \cup S_O$. 72

node quantifier A node quantifier is either a CQ of the form $\exists_{\bowtie n} y \varphi(y)$, a *universal* quantifier of the form $\forall y \varphi(y)$ or an exact quantifier of the form $\exists_{=1} x \varphi(x)$, where $\varphi(y) = R(x, y) \wedge C(y)$, $\varphi(x) = O(x)$ and $\bowtie \in \{\geq, \leq\}$. 72

node predicate A node predicate has the form $C(x), C(o)$ or $R(x, y)$. 72

node term A node term is either x, y or o . 72

normal DL-clause A DL clause that contains only DL-atoms in the body and DL-literals in the head. 65

partition Mutually disjoint atomic components generated by the atomic decomposition technique. 102, 104, 105

query clause A DL-clause in which all the literal are atoms. 65

simple role A role that is not transitive and does not have transitive sub-roles.. 69

strengthening A clause $K' \rightarrow M'$ is a strengthening of a clause $K \rightarrow M$ if $K' \subseteq K$ and $M' \subseteq M$. xix, 65, 72

List of Symbols

$K \to M \in_* \mathcal{CL}$	The set of clauses $\mathcal{CL}$ contains at least one strengthening of $K \to M$
β_ρ	The number of positive elements in each partition element, which is used as its weight in the MP
$\forall x \exists_{\geq n} y \ \varphi(x, y)$	Counting Quantifiers in First-Order Logic
$\mathcal{A}(v)$	Arithmetic label of the node v
$\mathcal{A}(v)$	Local arithmetic label of the node v
$\mathcal{A}$	Global arithmetic label
$\mathcal{CS}(v)$	Conflict Set, the smallest infeasible subset of arithmetic label
$\mathcal{DS}(v)$	The decomposition set defined for node v
$\mathcal{G}$	Completion Graph
$\mathcal{L}(v)$	A function that assigns a finite set of node clauses to each node v.
Ω	The set of all ground atoms that occur in $\mathcal{G}$ or $\mathcal{L}$.

π	A function to map every nominal o occurring in the ontology to a fresh unary predicate $\pi(o) = O(x)$
$\rho(x, y)$	A partition element of the form $\rho(x, y) = \bigwedge_{\ell=1}^{k} \lambda_\ell(y)$, where each $\lambda_\ell(y)$ is either $e_\ell(x, y)$ or its negation
σ	Substitution which binds a term to a variable
σ_ρ	The cardinality of the partition ρ
$\mathsf{core}(v)$	A function that assigns a unary predicate to each node v.
ϖ	First Order Logic sentence
$\xi(v)$	The *arithmetic solution* return by ARM corresponding to the arithmetic label $\mathcal{A}(v)$, a set of tuples $\langle \rho, \sigma_\rho \rangle$.
ζ	$\zeta \subseteq S_{\mathcal{U}} \cap \rho$ is a set of unary predicates which are represented by the node v.
A, B, C	Unary predicates
K	Conjunction of literals
L	Literal which is a quantifier or an atom
M	Disjunction of Literals
N_C	Set of *atomic concepts*
n_c	The number of feasible partition elements
N_I	Set of *named individual*

N_L	Set of *all literals* in the ontology $\mathcal{O}$
n_o	The number of all Nominals exist in the ontology
n_q	The total number of CQs occurring in $\mathcal{Q}(v)$
N_R	Set of *atomic roles*
o	A constant or nominal
R, S	Binary predicates
S_L	Set of all literals over the signature $S = (S_\mathcal{P}, S_\mathcal{C})$
S_O	The set of all fresh unary predicates regarding nominals defined as $S_O = \{\pi(o) \mid o \in S_\mathcal{C}\}$
$S_\mathcal{B}$	Set of binary predicate symbols (atomic roles) defined in the signature S (occurring in ontology $\mathcal{O}$)
$S_\mathcal{C}$	Set of constant symbols (all nominal) defined in the signature S (occurring in ontology $\mathcal{O}$)
$S_\mathcal{P}$	Set of predicate symbols defined in the signature S
$S_\mathcal{U}$	Set of unary predicate symbols (atomic concepts) defined in the signature S (occurring in ontology $\mathcal{O}$)

Chapter 1

Introduction

Description logics (DLs) are a family of formal representation languages used to represent the terminological knowledge of an application domain in a structured way. They are considered one of the primary foundations of the *Semantic Web* and the *Web Ontology Language* (OWL). They also serve as the basis for developing *ontology reasoning algorithms*. The elements of a domain are described by atomic concepts (unary predicates), atomic roles (binary predicates), and the concept and role constructors provided by the particular DL. For example, the concept of a bank having a branch whose employees are full-time can be defined using conjunction ($\sqcap$), universal ($\forall$) and existential ($\exists$) restrictions over the roles *hasBranch* and *hasEmployee* as follows: Bank $\sqcap \exists$*hasBranch*.($\forall$*hasEmployee*.Full-Time).

Statements in DLs are divided into two groups, namely TBox statements and ABox statements. The TBox contains terminological knowledge, while the ABox lists all assertional knowledge about instances (i.e., individuals). An *ontology* consists of an ABox and a TBox. Once an ontology is represented, DL reasoning services can infer implicit knowledge from explicitly expressed one. The computation of these inferences is called *reasoning*. The basic inference service provided by DLs is named *concept satisfiability*. It

checks whether a concept description denotes the empty set.

The concepts involved in modern ontologies typically are organized in a hierarchical structure called *taxonomy*, which reflects *subsumption* relations between concepts. In a medical ontology, for example, the concept Chickenpox would be *subsumed by* the concept Disease. To use reasoning services, an ontology engineer must provide definitions of concepts and their general properties rather than explicitly stating all relations between the concepts. An appropriate reasoning algorithm would compute the subsumption relations between concepts based on explicitly represented knowledge. An important DL reasoning service is ontology classification, which aims to compute the taxonomy.

1.1 Problem Statement

A description logic must be *decidable* (enable accurate reasoning) and *expressive* enough to be practical for representing knowledge of a particular domain. A DL must be equipped with applicable reasoning algorithms. Having a DL with such requirements is challenging because a higher expressivity of a DL often coincides with a higher time complexity for reasoning services.

Various DLs have been investigated to meet the needs of application domains with different levels of expressive power. The $\mathcal{ALC}$ DL is usually considered as the basic DL and $\mathcal{SROIQ}$ is one of the most expressive DL languages which extends $\mathcal{ALC}$ with *transitive Roles* $(\mathcal{R})$, *Role Compositions* $(\mathcal{R})$, *Nominals* $(\mathcal{O})$, *inverse Roles* $(\mathcal{I})$ and *Qualified Number Restrictions* $(\mathcal{Q})$.

The study of *efficient* DLs continued in two directions, either restricting the expressivity to preserve a low reasoning time complexity or developing new practical reasoning

procedures for expressive DLs [31, 26]. We follow the second approach in our research by introducing an efficient reasoning algorithm for an expressive DL named $\mathcal{SHOQ}$, which is a subset of OWL 2 and covers *nominals* and *Qualified Number Restrictions* (QCRs).

Tableau-based algorithms were proposed for the description logic $\mathcal{ALC}$ (attributive concept description language with complements) [7, 50] and later extended for more expressive DLs. Most modern ontology reasoners, such as FaCT++[1] [65], HermiT[2] [41], Pellet[3] [60], and RacerPro[4] [25], implement optimized *tableau-based algorithms*, or its variations.

The main idea of tableau-based algorithms for classification is to systematically construct a representation for a model of the input ontology plus the negation of each subsumption candidate. If all the representations considered by the procedure contain an *obvious contradiction (clash)*, it is concluded that the subsumption candidate holds. Usually, some heuristics and optimization techniques are used for selecting subsumption candidates. Otherwise, all subsumption relations between concepts should be checked for classifying the ontology.

There is another type of reasoning algorithm, called *Consequence-Based (CB) algorithms*, which in contrast to tableau-based algorithms, rather than building counter-models, directly derive logical consequences of axioms in the ontology using inference rules. The inference rules should be designed to derive all implied subsumptions while deriving only a limited number of consequences, not to overload the system with irrelevant clauses.

CB algorithms were first introduced for the DL $\mathcal{EL}$ [4], which is a tractable and simple

[1] http://owl.man.ac.uk/factplusplus/
[2] http://www.hermit-reasoner.com/
[3] http://pellet.owldl.org/
[4] https://www.ifis.uni-luebeck.de/index.php?id=385

DL that supports conjunction $C \sqcap D$ and existential restriction as the only concept constructors. However, some of the largest currently available ontologies, such as SNOMED CT [52] (a medical ontology including more than 300,000 concepts), are covered mainly by $\mathcal{EL}$.

CB classification algorithms are not limited to $\mathcal{EL}$ but can be extended to more expressive DLs. The extended algorithms are proven to be worst-case optimal and show 'pay-as-you-go' behaviour [8, 54, 57]. However, until now, no CB algorithm could handle *number restrictions* and *nominals* efficiently employing linear programming and algebraic algorithms.

1.2 Motivation

The hierarchical structure of modern ontologies helps people effectively understand subsumption relations between concepts in an ontology. Using modern ontology languages such as OWL-DL, the ontology designer needs only to state basic definitions of concepts and their general and unique properties instead of explicitly stating each of the subsumption relations between the concepts. The taxonomy of the ontology can then be computed by running a suitable reasoning algorithm on the presented knowledge.

Reasoning services, including concept satisfiability and ontology classification, play an essential role during the life cycle of an ontology. Concept satisfiability is crucial during the designing phase of an ontology for detecting inconsistencies and other modelling errors, which typically happen due to unintended or missing subsumption relationships. Ontology classification is required for answering domain questions [45] in the application phase of an ontology.

Some features of CB algorithms cause them to have a better practical performance than conventional tableau-based algorithms. As their name suggests, CB algorithms derive the consequences of an ontology, so they never check subsumptions that are not entailed. Typically, the number of entailed subsumptions is much smaller than the number of potential subsumptions between ontology concepts.

Furthermore, these algorithms classify the whole ontology in one round; they find all entailed subsumptions by following the consequences of axioms. They prevent redundancy caused by checking each subsumption separately, so they reduce the total number of operations required for the whole ontology classification. Finally, CB algorithms are shown to be relatively easily parallelizable. The CB algorithms proposed so far are not well equipped to efficiently reason about the expressive description logic $\mathcal{SHOQ}$, which allows the interaction of nominals and QCRs as they follow an arithmetically uninformed reasoning process.

Cardinality restrictions are a part of OWL-DL that can be used for imposing arithmetic restrictions via the notation: $\texttt{FootballTeam} \sqsubseteq \geq 11\texttt{hasPlayers}$. This statement says that each football team has at least 11 players. Using large values in cardinality restrictions is very natural in many domains. For example, in medical fields, the human anatomy distinguishes hundreds of different kinds of bones as part of the human skeleton [20].

QCRs are a generalization of cardinality restrictions that allows specifying to which class the *role successors* (i.e., individuals related via a particular role) belong. For example, the statement

$$\texttt{GradStudent} \sqsubseteq \geq 130\texttt{hasCredit.Science} \tag{1.1}$$

states that every graduate student must have at least 130 science credits. Cardinality restrictions are the only constructors that allow imposing a cardinality restriction on the number of role successors. Most existing DL reasoners cannot classify ontologies containing a large number of QCRs or if large values are used in QCRs.

Nominals is another part of OWL-DL, also known as named individuals. They capture the notion of uniqueness and identity in knowledge representation. *Nominals* are interpreted as atomic concepts with exactly one element. In general, they can be used to express enumerations of the form: $\mathsf{SolarSystemPlanets} \equiv \{\mathsf{Mercury}, \mathsf{Venus}, \mathsf{Earth}, \mathsf{Mars}, \mathsf{Jupiter}, \mathsf{Saturn}, \mathsf{Uranus}, \mathsf{Neptune}\}$. Furthermore, the nominals allow individuals to be referenced within concept descriptions. For example, Ph.D. students studying at Concordia can be defined as: $\texttt{PhDStudent} \sqcap \exists\ \mathit{studyAt}.\texttt{Concordia}$ where Concordia is a *nominal*. *Nominals* are naturally used in real-world knowledge bases, and some ontologies such as the *Wine* Ontology[5], which was originally designed for exploring DL constructors.

Without *nominals*, referring to individuals within concept descriptions is not possible because there is always a clear separation between data related to concepts and roles, which are represented as *Terminological Axioms* in a TBox (Terminological Box) and data about individuals which are defined as *Membership Assertions* in an ABox (Assertional Box). Since there is always a trade-off between the expressivity of the language and the complexity of reasoning, DLs would enjoy additional expressive power of *nominals* for the price of higher computational complexity.

However, in practice, nominals are hardly used in OWL ontologies. Ontology designers avoid using nominals, even Galen[6] models `maleSex` as an atomic concept, which seems unintuitive since there is only one male sex. One reason for this apparent lack of

[5]https://www.w3.org/TR/owl-guide/wine.rdf
[6]http://www.co-ode.org/galen/

nominals in current ontologies is the minimal tool support for nominals. Implementing algorithms that can handle the interaction of nominals and QCRs efficiently turned out to be challenging. These challenges will be discussed briefly in the next Section.

1.3 Challenges

QCRs are highly non-deterministic constructors; the level of non-determinism goes even higher if they contain large values. Current DL reasoners try to satisfy imposed numerical constraints by exhausting all possibilities. But searching for a model in such an arithmetically uninformed or blind way is usually highly inefficient. Therefore, using arithmetic methods can improve the average case performance of reasoning about QCRs. In this research, we propose a CB algorithm that benefits from *Integer Linear Programming* to handle the numerical features of the language properly.

On the other hand, the syntax of nominals allows referring to ABox individuals in the Tbox, so the TBox and ABox are no longer separated. Having them separated is usually more desirable because it will enable the development of different reasoning processes for ABoxes and TBoxes. The semantics of nominals cause even more challenges because each nominal is interpreted either as one individual or as a concept with the cardinality of exactly one. So, nominals impose global and implicit numerical restrictions, which have to be appropriately handled. For example, if the concept description $\texttt{john} \sqcap \texttt{Student}$ is not empty, then one can conclude that `john` is a student.

The interaction of nominals and QCRs violates the tree model property, such that the search space to find the model is no longer in the form of a tree. By losing this property, the optimization techniques that rely on this property are no longer applicable. Besides,

in a DL that supports the interaction of nominals and QCRs, there are two sources of numerical restrictions that impact each other and must be considered together. QCRs impose explicit numerical restrictions by setting a lower (upper) bound on the number of *role successor*. All these restrictions are local because they only affect the related elements via a particular role. *Nominals* impose implicit numerical restrictions by naming and also counting the number of individuals. Numerical restrictions imposed by nominals are global since they affect all elements of the *interpretation domain*.

For example, the concept $\mathsf{Season} \equiv \{\mathsf{Spring}, \mathsf{Summer}, \mathsf{Fall}, \mathsf{Winter}\}$ where all season names are nominals, means that the instances of the Season concept can be only one of the four enumerated ones. Nominals are the only constructors that can limit the number of instances of a particular concept. Furthermore, they may interact with QCRs to restrict the number of successors of a specific role.

1.4 Research Objectives

This research aims to devise a practical and efficient reasoning algorithm for classifying expressive ontologies containing the interaction of nominals and QCRs while having a competitive performance compared to existing reasoners. Our reasoning algorithm extends CB algorithms combined with an algebraic reasoning technique. The main objectives of the research can be summarized as follows:

- **Extending Consequence-based Reasoners:** CB reasoners were mostly used for reasoning the $\mathcal{EL}$ or Horn family of DLs, which do not support non-deterministic constructors. They have only recently been extended to more expressive DLs. This

research aims to design a CB algorithm for an expressive DL that can efficiently handle the interaction of complex numerical restrictions imposed by the combination of QCRs and Nominals. CB reasoners derive all entailed subsumptions by applying inference rules without requiring backtracking.

- **Supporting an expressive DL:** The work presented in this thesis focuses on providing reasoning support for $\mathcal{SHOQ}$ which supports nominals ($\mathcal{O}$) and number restrictions ($\mathcal{N}$ or $\mathcal{Q}$). These two constructors both impose numerical constraints that may interact with each other.

- **Arithmetic Reasoning:** QCRs and nominals are two constructors for imposing numerical restrictions. So, our reasoning approach must benefit an arithmetic encoding for mapping these restrictions into a set of inequalities, the feasibility of which can be determined using standard Integer Linear Programming algorithms.

- **Minimum Unsatisfiable Subsets** An unsatisfiable set of numerical constraints could be resolved to derive many consequences. However, finding the minimum core set that causes the unsatisfiability, requires checking the satisfiability of all its subsets. We employ the MARCO algorithm to discover all unsatisfiable minimum subsets without checking the satisfiability of all subsets. The MARCO algorithm defines a search map, which is explored by defining a corresponding *Integer Linear Problem*.

- **Prove Correctness** The reasoning procedure must ensure soundness and completeness. A correct classification procedure must be *sound* and *complete*. A procedure is sound if every derived subsumption is implied by the input ontology $\mathcal{O}$, and it is complete if every subsumption implied by $\mathcal{O}$ can be obtained by the procedure.

- **Termination and Efficiency** A practical decision procedure should terminate in a reasonable time to be useful. So, a DL reasoner must be equipped with a set of optimization techniques to ensure efficiency without breaking *correctness* or *termination*.

- **Implementation and Evaluation** We must implement our reasoning procedure to evaluate its performance in practice. We will compare the response time of the implemented CB procedure with existing reasoners to examine its efficiency.

1.5 Outline

The thesis is divided into three main parts. Part I provides the necessary background knowledge, while Part II presents the new contributions of my research. Finally, Part III illustrates the practical implications of the research and discusses future directions for this research. The thesis is structured to allow readers to easily follow the main points of my research.

Part I : **Foundation**

Chapter 2 (**Background**) introduces DLs, their inference services, and reasoning algorithms. This chapter can be skipped by readers who are already familiar with DLs and their reasoning algorithms.

Chapter 3 (**Related Work**) provides a comprehensive survey of state-of-the-art DL reasoners. We highlight the strengths of existing reasoners and identify areas for improvement. Our new arithmetic reasoning algorithm addresses some of the limitations of existing algorithms.

Part II : **Calculus and Applications**

Chapter 4 (**Preliminaries**) provides the preliminary notions needed to formally define our algorithm. It reviews first-order and description logic, their conversion rules, and the notations used throughout the thesis. Finally, it reviews the ontology normalization process to obtain standard ontologies for the algorithm.

Chapter 5 (**CB Reasoning in $\mathcal{SHOQ}$**) treats *arithmetic module* (ARM) as a black box and formally defines the proposed CB algorithm. The CB algorithm relies on ARM to reason about the numerical restrictions imposed by QCRs and Nominals. This chapter defines the graph-based framework constructed during the reasoning process and introduces the CB inference rules. It also includes numerous examples to clarify the newly introduced notions.

Chapter 6 (**Arithmetic Module**) formally defines ARM as a standalone unit that uses linear programming to find a solution that satisfies the numerical restrictions. This chapter also formalizes the expected input and output of the module, which the calculus will use to interact with ARM. This chapter starts with introducing the notion of atomic decomposition to encode QCRs, nominals, and their affiliated axioms as inequalities. It then discusses solving the derived integer linear problem using the column generation technique. Finally, it introduces an efficient algorithm for finding the minimum unsatisfiable subsets when the derived inequality system is infeasible.

Chapter 7 (**Key Properties**) formally proves the soundness and completeness of the proposed CB algorithm. Soundness is proved for each of the inference rules, following the soundness of the hyperresolution rule. For completeness, we show

that a counter-model can be built for every subsumption that does not appear in the complete completion graph. This chapter also analyzes the worst-case time complexity of the CB calculus by introducing an upper bound on the number of steps required to complete the algorithm.

Part III : **Evaluation and Conclusion**

Chapter 8 (**Implementation**) discusses the high-level design of the implemented reasoning prototype called CARON (Consequence-based Algebraic Reasoning for $\mathcal{O}$ (nominals) and $\mathcal{N}$ (number restrictions)). This chapter overviews the architecture and main features of CARON. Also, it describes the underlying defined data structure and finally discusses the inference rules and Arithmetic Module (ARM) implementation.

Chapter 9 (**Evaluation**) presents the practical result of applying the reasoning algorithm on test ontologies and illustrates its performance in comparison with existing reasoners. It finally analyzes the obtained result to better understand the strengths and weaknesses of the proposed approach and future works.

Chapter 10 (**Conclusion**) summarizes this research, highlights the achievements and proposes the next steps and future works.

Part I

Foundation

Chapter 2

Background

This chapter provides an overview of the essential definitions of description logic and the background knowledge required for the remainder of this thesis. It is included to make this thesis self-contained, but readers who are already familiar with this topic may skip this chapter.

It begins with a formal introduction to description logics in Section 2.1.1 and continues by presenting their syntax and semantics in Section 2.1.2. Section 2.2 investigates DL constructors and introduces description logics with different levels of expressivity. Inference services provided by DLs are discussed in Section 2.3, and Section 2.4 introduces the two main types of reasoning algorithms in description logics: *Tableau-based* and *Consequence-based* reasoning algorithms. Finally, the complexity of DL reasoning and the need for a highly optimized and efficient reasoning algorithm is discussed in Section 2.5.

2.1 Description Logics

In this section, we briefly review the basics of description logic and recapitulate the syntax and semantics of the DL $\mathcal{SROIQ}$ and its relevant fragments to this thesis. Furthermore, we describe the DL reasoning problems and highlight the ones our calculus aims to solve.

2.1.1 Basics of Description Logic

Description Logics (DLs) [5] is a family of knowledge representation languages with formal syntax and semantics, which are used for representing and reasoning about the elements of an application domain. DLs were first used in knowledge representation systems to provide formal and logic-based semantics for frames [39] and semantic networks [48]. Description Logics are well-known for their logic-based semantics and inference capabilities. They equip a knowledge representation system with facilities to establish knowledge bases and reason about their content.

DLs are used to model the relationships between entities in an intended domain. There are three types of entities in DLs: *concepts*, *roles*, and *individuals*. Concepts denote sets of individuals, roles denote binary relations between individuals, and individual names indicate single individuals in the domain. They can be considered unary predicates, binary predicates, and constants in first-order logic. A DL ontology consists of a set of statements called *axioms* and is known to be true in the particular domain described by the ontology. The axioms of an ontology are separated into two categories, the *TBox* and the *ABox*. The TBox introduces the terminology and the vocabulary of the domain including terminological axioms which describe general knowledge about concepts and roles, while the ABox contains assertions about named domain elements in terms of TBox vocabulary, such as

membership of an individual in a concept or a relationship between two individuals via a role.

For example, an ontology that is modelling the domain of people and their family relationships might use the concept Man referring to all the persons that are male, roles such as $\mathsf{hasChild}$ to represent a (binary) relationship between parents and their children and named individuals such as John and Sara to denote the people named John and Sara in the domain. The ontology might include TBox axioms such as $\mathsf{Father} \equiv \mathsf{Man} \sqcap \exists\ \mathsf{hasChild}.\mathsf{Person}$ to state that fathers are those individuals who are men and have at least one child who is a person and ABox axioms such as $\mathsf{John}{:}\mathsf{Man}$ and $(\mathsf{John}, \mathsf{Sara}) : \mathsf{hasChild}$ to assert that John is a man and John has a child named Sara respectively.

An important feature of DLs is their capability of *inferring* implicit knowledge from explicitly expressed knowledge. The computation of inferences is called *reasoning*. Designing reasoning algorithms with high performance has been one of the main concerns of DL research. The expressive power of the DL must be restricted as much as possible in accordance with the intended application to have a low reasoning complexity. This is one of the reasons for having DLs with different levels of expressivity. The expressivity of a DL language is determined by the constructors it supports.

$\mathcal{ALC}$ is usually considered the basic DL with which other varieties are usually compared. $\mathcal{ALC}$ supports *negation*, *conjunction*, *disjunction*, *value restriction*, and *existential restriction* constructors for defining complex concept descriptions. The formal syntax, semantics, and inference services of $\mathcal{ALC}$ are introduced throughout the rest of this section. The following section introduces DLs with a higher expressivity, such as the DL $\mathcal{SHOQ}$ which extends $\mathcal{ALC}$ with *transitive Roles* ($\mathcal{R}$), *Role Hierarchy* ($\mathcal{H}$), *Nominals* ($\mathcal{O}$), and *Qualified Number Restrictions* ($\mathcal{Q}$) and is the main focus of our research.

2.1.2 Syntax and Semantics

The formal syntax and semantics of DLs are introduced in this section. DLs are defined w.r.t. three non-empty and pair-wise disjoint sets of *atomic concepts* N_C, *atomic roles* N_R, and *named individuals* N_I. Complex *concepts* and *roles* are defined recursively using the syntax rules in Table 2.1 and 2.2. Unless otherwise explicitly stated, the letters A, B denote atomic concepts (concept names), C, D, E denote concepts, R, S denote roles, and a, b, c denote individuals. The concepts $\top$ and $\bot$ are abbreviations for $(C \sqcup \neg C)$ and $(C \sqcap \neg C)$, respectively.

The semantics of concept descriptions can be defined in terms of standard Tarski-style semantics based on an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set of individuals called the domain of interpretation and $\cdot^{\mathcal{I}}$ is an interpretation function. The interpretation function $\cdot^{\mathcal{I}}$ maps each atomic concept $A \in N_C$ to a subset of $\Delta^{\mathcal{I}}$, and each role $R \in N_R$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Table 2.1 shows the extension of this assignment to $\mathcal{ALC}$ constructors.

Definition 2.1 (Satisfiability)**.** A concept description C is said to be *satisfiable* by an interpretation $\mathcal{I}$ *iff* $C^{\mathcal{I}} \neq \emptyset$, i.e., there exists an individual $x \in \Delta^{\mathcal{I}}$ as an instance of C such that $x \in C^{\mathcal{I}}$.

Definition 2.2 (Subsumption)**.** A concept description D *subsumes* a concept description C (written as $C \sqsubseteq D$) *iff* $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$, i.e., the first concept description is always interpreted as a subset of the second one.

DLs which support full negation, subsumption, and satisfiability can be converted to one another. A concept description C is subsumed by D, $C \sqsubseteq D$, *iff* $C \sqcap \neg D$ is unsatisfiable, and a concept description C is satisfiable *iff* $C \not\sqsubseteq \bot$.

Definition 2.3 (Concept Inclusion Axiom)**.** A concept inclusion axiom is either a *concept subsumption* ($C \sqsubseteq D$) or a *concept equivalence* ($C \equiv D$) where C, D are concept descriptions and $C \equiv D$ is an abbreviation for $\{C \sqsubseteq D, D \sqsubseteq C\}$. A concept inclusion axiom is referred to as a *General Concept Inclusion* (GCI) axiom if C is not a concept name or C is top ($\top$).

Definition 2.4 (TBox $\mathcal{T}$)**.** A TBox $\mathcal{T}$ is a finite set of concept inclusion axioms (an example TBox is shown in Figure 2.1). A TBox $\mathcal{T}$ is satisfiable by an interpretation $\mathcal{I}$ *iff* $\mathcal{I}$ satisfies all the axioms in $\mathcal{T}$; in this case, $\mathcal{I}$ is called a model of the given TBox $\mathcal{T}$ and the TBox is said to be consistent.

A TBox $\mathcal{T}$ is called *acyclic* if it does not contain multiple definitions like $A \equiv C$, $A \equiv D$ for distinct concept description C, D and direct or indirect *cyclic* definitions such as $\mathcal{T} = \{A \equiv \exists R.B,\ B \equiv C,\ C \equiv \forall S.A\}$. In contrast, a *general* TBox may contain concept axioms of any type, with no restriction. Acyclic TBoxes can be *unfolded* or *expanded* by eliminating

TBox Axioms
Woman ≡ Person ⊓ Female
Parent ≡ Person ⊓ ∃ hasChild.Person
Mother ≡ Parent ⊓ Female
ABox Assertions
Sara: Woman
(Sara, John): hasChild

FIGURE 2.1: A Sample DL Knowledge Base Consisting of a TBox and an ABox

all defined names from the right-hand side of all axioms. In this way, a reasoning problem w.r.t an acyclic TBox can always be reduced to a reasoning problem w.r.t the *empty TBox* [5].

Definition 2.5 (ABox $\mathcal{A}$)**.** An Abox $\mathcal{A}$ is a finite set of *assertions* of the form $a : C$, $(a, b) : R$, where C is a concept description, R is a role and a, b are individuals. A model $\mathcal{I}$ satisfies $a : C$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and it satisfies $(a, b) : R$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An ABox $\mathcal{A}$ is said to be satisfiable w.r.t a TBox $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ that satisfies all assertions in $\mathcal{A}$.

Concept satisfiability and subsumption testing can be reduced to ABox consistency problems. A concept C is satisfiable *iff* the ABox $\{a : C\}$ is satisfiable for some $a \in N_I$ and $C \sqsubseteq D$ holds *iff* the ABox $\{a : C \sqcap \neg D\}$ is unsatisfiable.

If the *unique name assumption* is imposed on an ABox $\mathcal{A}$, different individual names consistently denote different elements in the domain. It requires the mapping between individual names and elements of $\Delta^{\mathcal{I}}$ to be injective, so $a^{\mathcal{I}} \neq b^{\mathcal{I}}$ is true for all pairs of individuals. But this assumption is usually relaxed because it is too strict. In this case, the inequalities between individuals should be asserted explicitly in the ABox whenever they hold, as $a \not\approx b$.

$\mathcal{I}$ is a *model* of an ontology $\mathcal{O}$ (written $\mathcal{I} \models \mathcal{O}$) if $\mathcal{I}$ satisfies all axioms in $\mathcal{O}$. An ontology is *consistent* if it has at least one model; otherwise, it is *inconsistent*. An axiom α is said to be a consequence of an ontology $\mathcal{O}$, or an ontology $\mathcal{O}$ entails an axiom α (written $\mathcal{O} \models \alpha$) if every model of $\mathcal{O}$ satisfies the axiom α.

TABLE 2.1: Syntax and Semantics of DL $\mathcal{ALC}$.

	Syntax	**Semantics**
named ind.	a	$a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$
atomic role	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
atomic concept	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
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	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
existential res.	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : \langle x, y \rangle \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
universal res.	$\forall R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : \langle x, y \rangle \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$

2.2 The DL Family

DLs with various levels of expressivity have been introduced to meet the requirements of certain application domains. The expressivity of different DLs is typically specified by the type of constructors they allow. For example, the basic description logic $\mathcal{ALC}$ only allows the $\top, \bot, \sqcap, \sqcup, \neg, \exists, \forall$ constructors and concept inclusion axioms. A DL constructor is either a concept constructor or a role constructor. Different concepts and role constructors are introduced in the two following subsections.

2.2.1 Concept Constructors

As their name suggests, concept constructors operate on concepts to inductively build complex descriptions from atomic concepts.

Nominals

Nominals are known as named individuals. They can be used for naming ABox individuals so that they can be present within concept descriptions in the TBox. Without *nominals*, the TBox axioms and the ABox assertions are separated, and there is no way to limit the maximum number of instances of a concept.

Nominals are also considered concepts with exactly one instance that will be interpreted as singleton sets. Nominals enable DLs with the notion of uniqueness and identity. There exist many concepts in the real world that need to be modelled using *nominals* such as "Concordia University", "Earth" or "Canada". The interpretation of a nominal $o \in N_o$ by a function $\cdot^{\mathcal{I}}$ is presented in Table 2.2, where N_o indicates the set of nominals. *Nominals* can entail *concept cardinalities*; for example, the concept Continent, which is defined as follows, has exactly 6 instances where the continent names are nominal. They are assumed to be pairwise disjoint:

$$\mathsf{Continent} \equiv \mathsf{Asia} \sqcup \mathsf{Africa} \sqcup \mathsf{America} \sqcup \mathsf{Antarctica} \sqcup \mathsf{Europe} \sqcup \mathsf{Australia}$$

In the presence of nominals, an ABox can be considered as syntactic sugar, as ABox assertions can be expressed as TBox axioms. For example, assuming that sara is an individual and an instance of the concept Student. The membership assertion sara: Student can be presented using the TBox axiom $\mathsf{sara} \sqsubseteq \mathsf{Student}$ where sara is nominal. Also, assuming that sara and computer are two individuals, the role assertion $(\mathsf{sara}, \mathsf{computer})$: Studies can be translated to the axiom $\mathsf{sara} \sqsubseteq \exists \mathsf{Studies.computer}$, where sara and computer are two nominals.

Qualified Cardinality Restriction

Qualified Cardinality Restrictions (QCRs) can be used for specifying a lower (at-least restriction) or upper (at-most restriction) bound on the number of *R-successors* belonging to a certain concept, where $R \in N_R$. For example the axiom ConcordiaGradCourse $\sqsubseteq \geq 15$ `hasStudent.ConcordiaStudent` specifies that at least 15 ConcordiaStudents must be enrolled in a ConcordiaGradCourse for the course to be offered. On the semantics side, QCRs are extended by the interpretation function $\cdot^{\mathcal{I}}$ as shown in Table 2.2.

Definition 2.6 (R-successor)**.** A domain element $y \in \Delta^{\mathcal{I}}$ is said to be an R-successor if there exists a domain element $x \in \Delta^{\mathcal{I}}$ such that x and y are related through the role R, $\langle x, y \rangle \in R^{\mathcal{I}}$. The set of all R-successors for a given role R is defined as $Succ(R) = \{y \in \Delta^{\mathcal{I}} \mid \exists x \in \Delta^{\mathcal{I}} : \langle x, y \rangle \in R^{\mathcal{I}}\}$.

Definition 2.7 (Qualifying Concept)**.** A qualifying concept A is a concept name that occurs in a QCR of the form $\leq R.A$ or $\geq R.A$ to impose a minimum or maximum on the number of R-successors for a role $R \in N_R$ that are instances of A.

2.2.2 Role Constructors

Role constructors are used for defining complex roles based on atomic ones. Role constructors that will be discussed through our research are *role hierarchies*, *transitive roles*, and *inverse roles*.

TABLE 2.2: Syntax and Semantics of DL Constructors.

Name	**Syntax**	**Semantics**
Role Constructors:		
Transitive Role	$trans(R)$	if $\langle x,y\rangle \in R^{\mathcal{I}} \wedge \langle y,z\rangle \in R^{\mathcal{I}}$ then $\langle x,z\rangle \in R^{\mathcal{I}}$
Role Subsumption	$R \sqsubseteq S$	$R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$
Inverse Role	R^{-}	$\{\langle x,y\rangle \mid \langle y,x\rangle \in R^{\mathcal{I}}\}$
Concept Constructors:		
At-least Restriction	$\geq nR.C$	$\{x \in \Delta^{\mathcal{I}} \mid \ ||\{y \mid \langle x,y\rangle \in R^{\mathcal{I}}\}|| \geq n \wedge y \in C^{\mathcal{I}}\}$
At-most Restriction	$\leq nR.C$	$\{x \in \Delta^{\mathcal{I}} \mid \ ||\{y \mid \langle x,y\rangle \in R^{\mathcal{I}}\}|| \leq n \wedge y \in C^{\mathcal{I}}\}$
Nominal	o	$\|\|o^{\mathcal{I}}\|\| = 1$

Role Hierarchies

Role hierarchies define *subrole* and *superrole* relationships between the roles used in the TBox. A role hierarchy consists of a set of Role Inclusion Axioms (RIA) of the form $R \sqsubseteq S$ where $R, S \in N_R$ and it indicates that if $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in R^{\mathcal{I}}$ then $\langle a^{\mathcal{I}}, b^{\mathcal{I}}\rangle \in S^{\mathcal{I}}$. For example, the axiom $\mathsf{isBrotherOf} \sqsubseteq \mathsf{isSiblingOf}$ indicates that if there is an individual $a \in N_I$ who is a brother of $b \in N_I$, then a is also a sibling of b. If the relationship $R \sqsubseteq_* S$ holds between two roles, S is said to be a super role of R, and R is called a sub role of S where $\sqsubseteq_*$ is the transitive reflexive closure of $\sqsubseteq$.

Transitive Roles

Transitivity is a role characteristic that is used to define transitive relations between concepts. Assuming that a role R is declared to be transitive, $\mathsf{Trans}(R)$, if $\langle a^{\mathcal{I}}, b^{\mathcal{I}} \rangle \in R^{\mathcal{I}}$ and $\langle b^{\mathcal{I}}, c^{\mathcal{I}} \rangle : R$ then $\langle a^{\mathcal{I}}, c^{\mathcal{I}} \rangle : R^{\mathcal{I}}$. For example, if the role isFriendOf is defined as a transitive role and if John is a friend of Mary and Mary is a friend of George, then John is also a friend of George.

Definition 2.8 (Simple Role)**.** A role R is a simple role if it is not transitive and has no transitive sub-role.

Inverse Roles

Inverse roles are used to present converse relations between individuals and are indicated by the $(^{-})$ operator. For example $\mathsf{hasChild} \equiv \mathsf{hasParent}^{-}$ express that hasChild is the converse relation of hasParent. Their semantics are preserved by extending the interpretation function $\cdot^{\mathcal{I}}$ as shown in Table 2.2.

2.2.3 Light-Weight DL

Subsumption checking is ExpTime-complete for most of DLs extended from $\mathcal{ALC}$ and even higher for more expressive DLs such as $\mathcal{SHOIQ}$ or $\mathcal{SROIQ}$. Light-weight DLs have been obtained by restricting expressivity in order to acquire a better reasoning complexity. There are three main categories of light-weight DLs: $\mathcal{EL}$ [3, 4], DL-Lite [2, 12], and DLP [23], which respectively correspond to language fragments OWL EL, OWL QL, and OWL RL of the Web Ontology Language.

$\mathcal{EL}$ is a simple tractable fragment of $\mathcal{ALC}$ which only allows top, conjunctions, and existential restrictions [3]. $\mathcal{EL}^{++}$ is an extension of $\mathcal{EL}$ that allows bottom, nominals, a restricted form of concrete domains, and complex role inclusions [4]. Despite allowing arbitrary role compositions, subsumption checking in $\mathcal{EL}^{++}$ can still be solved in polynomial time [4]. The $\mathcal{EL}$ family has mostly been used for modeling large but lightweight ontologies, which mainly consist of terminological axioms, such as SNOMED CT [51].

DL-Lite is a family of DLs which is commonly used in combination with traditional relational databases containing large volumes of data to increase the expressivity of a query language used for data retrieval. In this approach, known as *Ontology Based Data Access* (OBDA), an ontology defines a view on top of already existing data sources and provides a vocabulary for user queries [46]. An OBDA system rewrites such queries to standard query languages such as SQL using ontological information [12].

DLP is short for *Description Logic Programs* and can be considered as a kind of rule language contained in DLs. Because DLPs comprise various DLs that allow syntactically restricted axioms which could also be read as rules in first-order Horn logic without function symbols [23].

Definition 2.9 (Positive and Negative Polarities)**.** The polarity of concept occurrence in concept descriptions and axioms are recursively defined as follows: A concept C occurs positively (negatively) if it occurs positively (negatively) in $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\geq nR.C$ and $D \sqsubseteq C$ or if it occurs negatively (positively) in $\neg C$, $\leq nR.C$ and $C \sqsubseteq D$. Note that a concept may occur both positively and negatively in an axiom or ontology.

The Horn Fragment of DL

The Horn DL is a fragment of DL that does not allow non-deterministic constructors. An ontology is Horn if:

- no disjunction ($C \sqcup D$) or at-most restriction ($\leq mR.C$) with $m > 1$ occurs positively in its axioms.
- no negation ($\neg C$), universal restriction ($\forall R.C$), at-least restriction ($\geq nR.C$) with $n > 1$ or at-most restriction ($\leq mR.C$) occurs negatively in its axioms.

2.2.4 More Expressive DLs

The extension of $\mathcal{ALC}$ with *transitive roles* is traditionally indicated by the letter $\mathcal{S}$. Additional letters in the name of a particular DL hint for the constructors it allows (Table 2.2). For example $\mathcal{I}$ stands for *inverse roles*, $\mathcal{Q}$ for *qualified cardinality restrictions*, $\mathcal{O}$ for *nominals* and $\mathcal{H}$ for *role hierarchies*. So the DL $\mathcal{SHI}$ extends $\mathcal{S}$ with role hierarchies and inverse roles. The letter $\mathcal{R}$ is commonly used for a combination of *complex role inclusions*. This naming policy explains the name of $\mathcal{SROIQ}$, which is one of the most expressive DLs and also the DL underlying OWL 2 [40].

2.3 DL Inference Services

The most important feature of DLs is their knowledge extraction capability which allows them to infer implicit knowledge from explicitly represented one using inference services.

Inference services are categorized into two groups: TBox reasoning and ABox Reasoning. This section briefly discusses standard DL reasoning services.

2.3.1 TBox Reasoning

TBox inference services are defined for concept descriptions. These services include:

1. **Satisfiability Checking** determines if a concept is satisfiable.
2. **Subsumption Checking** infers whether there is a subsumption relation between two concepts (determines if one of them is subsumed by the other one).
3. **Classification** results in a taxonomy where all the concepts occurring in the ontology are organized in a subsumption hierarchy from the most general to the most specific.

2.3.2 ABox Reasoning

ABox inference services consider all axioms of an ontology (TBox axioms and ABox assertions) in their reasoning process. The ABox inference services include:

1. **Ontology Consistency** checks whether an ontology $\mathcal{O}$ is consistent. To have a consistent ontology, there must exist a model that satisfies $\mathcal{A}$ and $\mathcal{T}$.
2. **Instance Checking** determines whether an ABox individual a is an instance of a concept C w.r.t. $\mathcal{T}$ and $\mathcal{A}$.

In the presence of *nominals*, any concept satisfiability or ABox consistency problem can be mapped to a TBox consistency problem. A concept C is satisfiable w.r.t a TBox $\mathcal{T}$ if a

new nominal $o \in N_o$ exists such that $o \sqsubseteq C$ is consistent w.r.t. the TBox $\mathcal{T}$. For checking the consistency of an ABox $\mathcal{A}$ every assertion of the form $(a : C) \in \mathcal{A}$ is rewritten to the TBox axiom $a \sqsubseteq C$ and every assertion of the form $(a, b) : R \in \mathcal{A}$ is rewritten to the TBox axiom $a \sqsubseteq \exists R.b$.

2.4 DL Reasoning

Different reasoning algorithms were proposed for providing DL inference services such as *structural subsumption* (the early 90s), *tableau-based* (1991), *automata-based* (2003), *semantic binary tree* (2005) and *resolution-based* (2006) [18]. Among these, *tableau-based* and *resolution-based* algorithms are the most widely used, but each suffers from some drawbacks. *Consequence-based algorithms* try to overcome these drawbacks by combing these two. This section first discusses *tableau-based* algorithms and their drawbacks and then introduces CB algorithms, which are the main focus of this research.

2.4.1 Tableau-based Algorithms

Tableau-based algorithms were first designed for the DL $\mathcal{ALC}$ in 1991 [50] and extended later for more expressive DLs discussed in Chapter 3. In general, tableau-based algorithms follow a goal-directed procedure. They induce that a concept C is satisfiable if an interpretation $\mathcal{I}$ exists such that $C^{\mathcal{I}} \neq \emptyset$. In this section, we briefly discuss the idea of tableau-based algorithms, which was originally proposed for $\mathcal{ALC}$. Some tableau-based algorithms for more expressive DLs are reviewed in Chapter 3.

Tableau-based algorithms assume that all concept descriptions are in *Negation Normal Form* (NNF) for the sake of simplicity. The $\mathsf{NNF}(C)$ is obtained by moving all negation symbols down into the descriptions until they only occur in front of concept names. The NNF function works based on *De Morgan*'s rules, conversion rules of existential and universal restrictions[1] and the rules for quantifiers[2].

Tableau-based algorithms use a *completion graph* for presenting an abstraction of a model. A completion graph is a directed graph G shown as a tuple $(V, E, \mathcal{L}, \neq)$ where V is a set of nodes representing individuals in the domain, and E is a set of edges representing relations between individuals. Each node $x \in V$ has a label $\mathcal{L}(x)$, which is a set of concepts, such that the individuals represented by x are instances of concepts in $\mathcal{L}(x)$. Every edge between nodes x and y, $\langle x, y \rangle \in E$ is labeled with $\mathcal{L}(x, y)$, which is a set of role names satisfying the relations between two nodes. The symmetric binary relation $\neq$ keeps track of the inequalities that hold between pairs of nodes. Given an edge $\langle x, y \rangle$ connecting x to y, the individual x is called *predecessor* of y and y is the *successor* of x and the transitively closed set of successors (predecessors) is called *descendants* (*ancestors*) respectively.

In order to construct a model for checking the satisfiability of a concept C, most tableau-based algorithms initialize the completion graph G by creating a node x_0 and adding concept C to $\mathcal{L}(x)$. The algorithm proceeds by applying the completion rules from Figure 2.2 to the initial graph.

The algorithm terminates if either a *clash* is detected or no more rules are applicable; in this case, the derived completion graph is called *complete*. For example, a node x is said to contain a clash when $\mathcal{L}(x)$ contains C and $\neg C$ at the same time.

[1] The NNF of $\neg(\forall R.C)$ $(\neg(\exists R.C))$ is defined as $\exists R.\neg C$ $(\forall R.\neg C)$.
[2] The NNF of $\neg(\geq nR.C)$ $(\neg(\leq nR.C))$ is defined as $\leq (n-1)R.C$ $(\geq (n+1)R.C)$.

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

FIGURE 2.2: Completion Rules for $\mathcal{ALC}$ Tableau Algorithm

The $\sqcup$-Rule is *non-deterministic*, which means that applying this rule would produce more than one branch to follow. Tableau-based algorithms need to introduce more *non-deterministic* rules to support DLs with higher expressivity. For example, a concept C is considered unsatisfiable if all the branches come to a contradiction known as a clash. However, if at least one expansion leads to a complete clash-free completion graph, then the concept C is satisfiable.

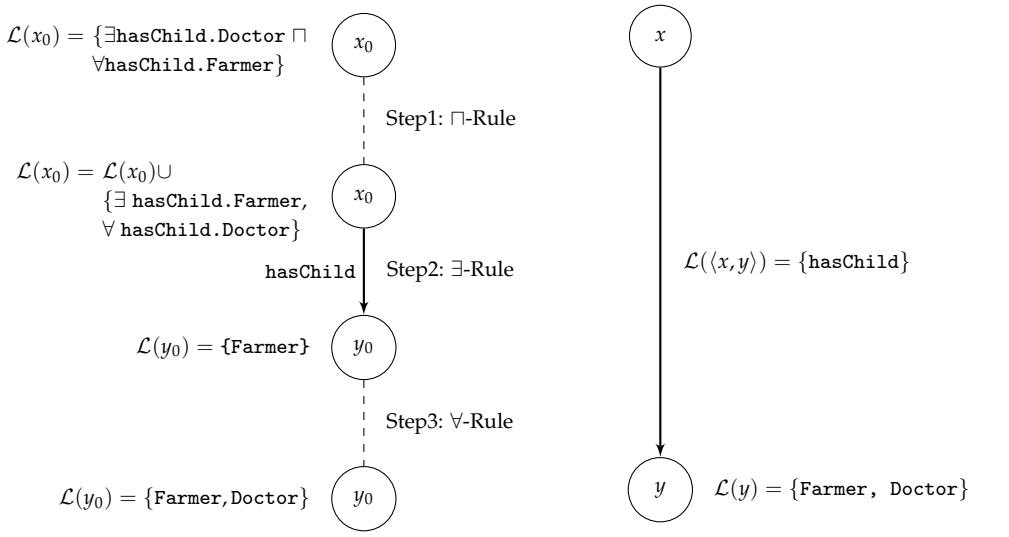
The $\mathcal{ALC}$ tableau algorithm requires a *blocking* technique to ensure termination. Since the completion rules may only be applied on nodes that are not blocked, the algorithm terminates after finitely many steps of applying completion rules. This algorithm is also proven to be sound and complete [50].

Definition 2.10 (Blocked node)**.** A node x is *directly blocked* by a node y if y is as an ancestor of x such that $\mathcal{L}(x) \subseteq \mathcal{L}(y)$. A node x is called a *blocked node* if it is blocked directly or if one of its ancestors is blocked.

Tableau-based algorithms can also reason about the satisfiability of an ABox, TBox, or the whole ontology by reducing it to a concept satisfiability problem. Assuming that the ABox is empty, to check the satisfiability of the TBox $\mathcal{T}$ the tableau algorithm converts all subsumption relations in the form $C \sqsubseteq D \in \mathcal{T}$ to their equivalent $\mathsf{NNF}(\neg C \sqcup D)$ than all the concept axioms in $\mathcal{T}$ can be reduced to a single axiom $\top \sqsubseteq C_{\mathcal{T}}$ where $C_{\mathcal{T}}$ is the conjunction of $\mathsf{NNF}(\neg C \sqcup D)$ for all $C \sqsubseteq D \in \mathcal{T}$. Since $\top$ implies $C_{\mathcal{T}}$ and as indicated by TBox-Rule in Figure 2.2, the concept $C_{\mathcal{T}}$ should be added to every created node of the completion graph G.

To check the satisfiability of an ABox $\mathcal{A}$ w.r.t. the TBox $\mathcal{T}$, the completion graph should be initialized based on ABox assertions. A node x is created in completion graph G corresponding to each ABox individual, and all concepts C for which $x : C \in \mathcal{A}$ are added to $\mathcal{L}(x)$ and an edge is created between each pair of related nodes, labelled with $\mathcal{L}(x, y)$ which contains all role names R that $(x, y) : R \in \mathcal{A}$.

Figure 2.3 shows the step-by-step expansion of the completion graph (obtained by applying completion rules) for checking the satisfiability of the concept $\exists$ hasChild.Farmer $\sqcap \forall$ `hasChild.Doctor`. This concept describes a set of people with a child who is a farmer while all their children are doctors. Although it may initially seem unsatisfiable, one can easily verify that this concept is satisfiable. Because there is no assumption that a doctor can not be a farmer, as Figure 2.3b shows, a possible model for this concept is a person with a child who is a doctor and a farmer.



(A) Steps of applying the expansion rules

(B) Final completion graph

FIGURE 2.3: Tableau Expansion During a Satisfiability Test

2.4.2 Consequence-based Algorithms

Consequence-based a.k.a *saturation-based* algorithms are closely related to the *completion-based* procedures proposed for $\mathcal{EL}^{++}$ ontologies in [4]. In contrast to tableau-based algorithms that enumerate pairs of concept names and build counter-models for subsumption relations, the CB algorithm derives logical consequences of axioms in the ontology using inference rules.

The main advantage of this method is computing subsumption relations "all at once" in a goal-directed way, eliminating the costly enumerations. In other words, these algorithms classify the whole ontology by simultaneously calculating all implied subsumption relations while guaranteeing that only a limited number of axioms is derived. In this section, we present the general approach of CB algorithms for $\mathcal{EL}$. Their extended versions for more expressive DLs are reviewed in Chapter 3.

Most CB algorithms require a *normalization* step before applying their subsumption

derivation procedure [4, 34, 56]. An ontology $\mathcal{O}$ is *normalized* if each GCI in $\mathcal{O}$ is in normal form. The ontology $\mathcal{O}$ can be transformed to a normalized ontology $\mathcal{O}'$ in linear time such that $\mathcal{O}'$ is a conservative extension of $\mathcal{O}$. Several such transformations have been proposed; we discuss our normalization procedure in Section 4.1. The transformation used in the [4] for $\mathcal{EL}^{++}$ ensures that all GCIs have one of the following forms:

$$A \sqsubseteq B \qquad\qquad A \sqsubseteq \exists R.B$$

$$A_1 \sqcap A_2 \sqsubseteq B \qquad\qquad \exists R.A \sqsubseteq B$$

The original CB algorithm does not need any particular *framework* [4, 35]. This algorithm uses a set $\boldsymbol{S}$ for storing axioms, which is closed under the rules in Figure 2.4. Applying the inference rules would add new axioms to the set $\boldsymbol{S}$.

Intuitively these algorithms compute all implied subsumption relations between a *goal concept* G and all concepts occurring in $\mathcal{O}$ by adding the initial axiom $G \sqsubseteq G$ to $\boldsymbol{S}$ and applying the inference rules in Figure 2.4 until no more rules apply. In the end, for each concept B occurring in $\mathcal{O}$ having $\mathcal{O} \models G \sqsubseteq B$ implies $G \sqsubseteq B \in \boldsymbol{S}$. It is important to note that the axioms in $\mathcal{O}$ are not used as premises of the inference rules in Figure 2.4, but rather as side conditions. This means that the ontology axioms are not applied arbitrarily, but only in specific cases where they are relevant.

Figure 2.5b shows the steps of applying the inference rules (Figure 2.4) to check if the query $q = \texttt{Grandparent} \sqsubseteq \texttt{Parent}$ is entailed from the ontology $\mathcal{O}$ shown in Figure 2.5a. One can easily verify that the query q is a consequence of the ontology $\mathcal{O}$.

As can be seen in Figure 2.5, in order to classify an ontology, it is sufficient to repeat

$R_{\sqsubseteq}$**-Rule**	**if** $C \sqsubseteq D \in \boldsymbol{S}$ and $D \sqsubseteq E \in \mathcal{O}$ and $C \sqsubseteq E \notin \boldsymbol{S}$ **then** add $C \sqsubseteq E$ to $\boldsymbol{S}$
$R_{\sqcap}^{-}$**-Rule**	**if** $C \sqsubseteq C_1 \in \boldsymbol{S}$ and $C \sqsubseteq C_2 \in \boldsymbol{S}$ and $C_1 \sqcap C_2 \sqsubseteq D \in \mathcal{O}$ and $C \sqsubseteq D \notin \boldsymbol{S}$ **then** add $C \sqsubseteq D$ to $\boldsymbol{S}$
$R_{\exists}^{-}$**-Rule**	**if** $C \sqsubseteq \exists R.D \in \boldsymbol{S}$ and $D \sqsubseteq D \notin \boldsymbol{S}$ **then** add $D \sqsubseteq D$ to $\boldsymbol{S}$
$R_{\bot}^{+}$**-Rule**	**if** $C \sqsubseteq \exists R.D \in \boldsymbol{S}$ and $D \sqsubseteq \bot \in \boldsymbol{S}$ and $C \sqsubseteq \bot \notin \boldsymbol{S}$ **then** add $C \sqsubseteq \bot$ to $\boldsymbol{S}$
$R_{\top}$**-Rule**	**if** $C \sqsubseteq C \in \boldsymbol{S}$ and $C \sqsubseteq \top \notin \boldsymbol{S}$ **then** add $C \sqsubseteq \top$ to $\boldsymbol{S}$
$R_{\exists}^{+}$**-Rule**	**if** $C \sqsubseteq \exists R.D \in \boldsymbol{S}$ and $D \sqsubseteq D_1 \in \boldsymbol{S}$, $\exists R.D_1 \sqsubseteq E \in \mathcal{O}$ and $C \sqsubseteq E \notin \boldsymbol{S}$ **then** add $C \sqsubseteq E$ to $\boldsymbol{S}$

FIGURE 2.4: Inference Rules for $\mathcal{EL}$ CB Algorithm [4, 35]

a similar reasoning process for all atomic concepts A occurring in the ontology $\mathcal{O}$ as goal concepts, i.e., to compute all conclusions of the inference rules in Figure 2.4 from the initial axioms $A \sqsubseteq A$, where A is an atomic concept.

2.5 Reasoning Complexity

Considering the complexity of a DL is a way of studying the difficulty of reasoning about it. A distinction should be made between the computational complexity of an inference service and the complexity of reasoning techniques proposed for providing an inference service.

The computational complexity of a DL can be determined based on analyzing the size of a completion model in the worst case and the time of constructing such a model. Obviously, in DLs enjoying more expressive power, reasoning requires a higher computational

Ontology $\mathcal{O}$

$\texttt{Grandparent} \sqsubseteq \exists\, \texttt{hasChild.Parent}$	(1)
$\texttt{Parent} \sqsubseteq \texttt{Person}$	(2)
$\exists \texttt{hasChild.Person} \sqsubseteq \texttt{Parent}$	(3)

(A) The ontology $\mathcal{O}$ consists of the above axioms

initial axiom:	$\texttt{Grandparent} \sqsubseteq \texttt{Grandparent}$	(4)
$R_{\sqsubseteq}[1,4]:$	$\texttt{Grandparent} \sqsubseteq \exists \texttt{hasChild.Parent}$	(5)
$R_{\exists}^{-}[5]:$	$\texttt{Parent} \sqsubseteq \texttt{Parent}$	(6)
$R_{\sqsubseteq}[6,2]$	$\texttt{Parent} \sqsubseteq \texttt{Person}$	(7)
$R_{\exists}^{+}[3,5,7]$	$\texttt{Grandparent} \sqsubseteq \texttt{Parent}$	(8)

(B) Steps of applying the inference rules

FIGURE 2.5: $\mathcal{EL}$ CB Algorithm for Goal Goncept `Grandparent`

complexity. For example, the use of general concept inclusions (GCIs) in TBoxes results in an ExpTime-complete satisfiability problem in most DLs extending $\mathcal{ALC}$. Table 2.3 illustrates the complexity of DLs with different expressive power.

TABLE 2.3: Worst-case Complexity of Some DL Inference Services

DL Languages	**Concept Satisfiability (Empty TBox)**	**Ontology Consistency (General TBox)**
$\mathcal{EL}^{++}$	P-complete	P-complete
$\mathcal{ALC}, \mathcal{ALCQ}, \mathcal{ALCOQ}$	PSpace-complete	ExpTime-complete
$\mathcal{SHQ}, \mathcal{SHOQ}$	ExpTime-complete	ExpTime-complete
$\mathcal{SHOIQ}$	NExpTime-complete	NExpTime-complete
$\mathcal{SROIQ}$	N2ExpTime-complete	N2ExpTime-complete

Initially, the high worst-case complexity may seem a rational reason for limiting the practical applicability of expensive constructors such as nominals. However, empirical experiences reveal that the average-case complexity is acceptable for real knowledge bases

because reasoning algorithms are usually equipped with optimization techniques and benefit from a set of intelligent heuristics. Therefore, a worst-case complexity analysis may serve as a theoretical estimation to prove the termination of a reasoning algorithm, but the claim that a proposed algorithm is practical must always be supported by empirical evaluations.

2.6 Conclusion

This chapter provided a formal description of DL languages by introducing their syntax and semantics. We also discussed members of the DL family with different levels of expressivity, which should be selected based on the requirement of a particular application domain. The original idea of *tableau-based* and *consequence-based* algorithms was introduced in this chapter. The following chapter investigates reasoning in more expressive DLs. It is worth mentioning that naive implementations of these algorithms are not practical because of their high level of non-determinism. A practical reasoning algorithm needs to be equipped with optimization techniques and heuristics.

Chapter 3

Literature Review

The previous chapter introduced two main types of DL reasoning algorithms: Tableau-based and consequence-based. This chapter reviews related research for adapting these algorithms to more expressive DLs, primarily focusing on those which support *QCRs*, *nominals* and their interaction.

While handling QCRs (mainly if containing big numbers) in an effective way is still an ongoing research area, supporting their interaction with nominals introduces even more challenges (Section 1.3). There are not many practical algorithms for reasoning in the presence of *nominals* and QCRs when big numbers are used in QCRs, or there are many nominals.

Through the rest of this chapter, we first review the idea of tableau procedures for more expressive DLs. Afterward, we introduce different varieties of optimization techniques employed to improve reasoning performance. Ultimately, we investigate CB algorithms proposed for reasoning in more expressive DLs alongside a brief analysis of their strengths and weaknesses.

3.1 Extending Tableau-based Algorithms

To deal with more expressive DLs, tableau-based algorithms need to be extended to preserve the semantics of the supported constructors [31].

- The data structure might need some changes to reflect the semantics of additional constructors. For example, in the presence of nominals, a distinction needs to be made between regular nodes and *nominal nodes*, which contain a nominal in their label. *Nominal nodes* can be arbitrarily interconnected and form complex graphs, while regular nodes can only be found on tree-like graphs rooted at *nominal nodes*. The completion graph is initialized with one node for every nominal in the ontology. So the graph would be a forest of tree-like structures rooted at *nominal nodes*.

- More extension rules should be introduced to support new constructors. In Standard tableau-based algorithms, the $\geq$-Rule creates n individuals as R-successors for satisfying each at-least restriction of the form $\geq nR.C \in \mathcal{L}(x)$. These individuals are asserted to be mutually distinct, so they can not be merged later.

  Then the choose-Rule assigns D or $\neg D$ to each of the created individuals according to every at-most restriction of the form $\leq mS.D \in \mathcal{L}(x)$. Any at-most restriction in $\mathcal{L}(x)$ may be violated if node x has K successors in D such that $K > m$. In this case, the $\leq$-Rule is applied to satisfy the violated at-most restriction by checking all possible ways of pairwise merging the K successors of x.

  Another rule should be defined to ensure that the semantics of nominals is never violated by merging any nodes containing the same nominal in their labels.

- There may be different types of clashes that should be detected, which may be due to number restrictions or nominals. For example, a node x must satisfy an at-most restriction of the form $\leq nR$ and already has m distinct R-successors with $m > n$, or two distinct nodes with the same nominal in their label.

- Expressive DLs require more sophisticated blocking strategies to ensure termination. For example, in the presence of nominals, none of the nodes between a blocking node and the blocked one can be a *nominal node* because repeating the cycle violates the semantics of nominals.

3.2 Optimizing Tableau-based Algorithms

A naive implementation of tableau-based algorithms is not practical [31, 30], so adopting a set of optimization techniques is crucial to be efficient for practical applications. This section reviews some of the optimization techniques that most modern DL reasoners use. We mainly focus on those designed to deal with nominals and QCRs. We also provide an informal analysis of their practical performance.

3.2.1 Absorption

General Concept Inclusion (GCI) axioms are hard for reasoning because they are highly non-deterministic. Therefore it is always desirable to eliminate or at least minimize GCIs. *Absorption* is a rewriting technique proposed for reducing the number of GCIs in a TBox by converting them to axioms of the form $A \sqsubseteq C$ where A is a concept name and C a concept expression, using the following conversion [29]:

$$C \sqcap D \sqsubseteq E \rightarrow C \sqsubseteq \neg D \sqcup E \tag{3.1}$$

Standard tableau reasoning converts every GCI of the form $C \sqsubseteq D$ to $\top \sqsubseteq \neg C \sqcup D$ and adds it to every node created in the completion graph. But using *Absorption* reduces the effect of GCIs and ensures they apply to a smaller range of nodes. But generally, it is not always possible to absorb all the GCIs in a TBox. *Absorption* can significantly improve reasoners' performance if applied appropriately. There might be more than one way to absorb a GCI, and finding the best way of absorbing a GCI is subject to many research activities [61, 32].

Nominal Absorption

In the presence of nominals, the standard absorption technique discussed previously is no longer applicable. Two absorption techniques have been introduced in [58] for absorbing GCIs with nominals. These techniques are categorized based on the roles of nominals in GCIs, either as enumerations or role fillers. Their goal is to eliminate the negative effect of disjunction at the expense of adding more assertions to the ABox.

- **Nominals as Enumeration** axiom (3.2) in the following example uses *nominals* for enumerating all instances of the concept `HairColor`:

Example 3.1 (Nominals for Enumerations)**.** Assume that a TBox $\mathcal{T}$ includes the following axioms:

$$\texttt{HairColor} \equiv \{\texttt{black, brown, red, blonde, grey}\} \tag{3.2}$$

$$\texttt{HairColor} \sqsubseteq \texttt{PhisicalFeature} \tag{3.3}$$

Applying regular absorption to these axioms is inefficient since it introduces many disjunctions, and every disjunction adds a backtracking point in the search space of a tableau-based algorithm.

An absorption technique, called “One-of Absorption", proposed in [58] tries to rewrite this type of GCIs using *nominals* characteristics and the following equivalence:

$$C \equiv \{a_1, a_2, ..., a_n\} \quad \Leftrightarrow \quad \begin{cases} C \sqsubseteq \{a_1, a_2, ..., a_n\} \\ a_1 : C, ..., a_n : C \end{cases} \tag{3.4}$$

For example, using the "One-of Absorption" technique, axiom (3.2) can be represented by the TBox axiom (3.5) and the ABox assertions (3.6) to (3.10).

$$\texttt{HairColor} \sqsubseteq \{\texttt{black, brown, red, blonde, grey}\} \tag{3.5}$$

$$\texttt{black} : \texttt{HairColor} \tag{3.6}$$

$$\texttt{brown} : \texttt{HairColor} \tag{3.7}$$

$$\texttt{red} : \texttt{HairColor} \tag{3.8}$$

$$\texttt{blonde} : \texttt{HairColor} \tag{3.9}$$

$$\texttt{grey} : \texttt{HairColor} \tag{3.10}$$

This technique is not applicable to reasoning algorithms that do not support ABox reasoning.

- **Nominals as Role fillers** The following axiom shows an example where nominals appear as role fillers.

$$\texttt{RedHead} \equiv \texttt{Person} \sqcap \leq 1\ \texttt{hasHair} \sqcap \exists\ \texttt{hasHair.red} \tag{3.11}$$

Axiom (3.11) is equivalent to the following two axioms:

$$\texttt{RedHead} \sqsubseteq \texttt{Person} \sqcap \leq 1\ \texttt{hasHair} \sqcap \exists\ \texttt{hasHair.red} \tag{3.12}$$

$$\texttt{Person} \sqcap \leq 1\ \texttt{hasHair} \sqcap \exists\ \texttt{hasHair.red} \sqsubseteq \texttt{RedHead} \tag{3.13}$$

We can absorb axiom (3.13) into the definition of `Person` as follows:

$$\texttt{Person} \sqsubseteq \texttt{RedHead} \sqcup \forall\, \texttt{hasHair}.\neg\, \texttt{red} \sqcup \geq 2\, \texttt{hasHair} \tag{3.14}$$

This concept should be considered for each node containing `Person` in its label. However, the "Has Value" absorption technique introduced in [58] is a more effective way of taking care of these cases. This technique takes advantage of nominal semantics and the following equivalence:

$$\exists R.o \sqsubseteq C \qquad \Leftrightarrow \qquad o \sqsubseteq \forall R^{-}.C \tag{3.15}$$

Based on "Has Value" absorption [58], the axiom (3.13) can be re-written by the ABox assertion:

$$\texttt{red} : \forall\, \texttt{hasHair}^{-}.(\texttt{ReadHead} \sqcup \neg\, \texttt{Person} \sqcup \geq 2\, \texttt{hasHair}) \tag{3.16}$$

Although the resulting axiom still contains the same number of disjunctions, this time, they are localized to the individuals related to `red` through the role `hasHair`. The efficiency of this technique is very case specific because changing the effect of disjunctions to a different set of individuals does not always result in reducing their impact. In the previous example, this absorption technique is only efficient if the number of individuals related to `red` through `hasHair` is considerably less than the number of `Person` instances. This technique is only available for reasoners supporting inverse roles.

3.2.2 Boolean Constraint Propagation

Boolean Constraint Propagation (BCP) is a simplification technique that examines disjunct concept descriptions and simplifies them where possible. Therefore it reduces the number of backtracking points in the search space and decreases the probability of expanding a branch that will end up with a clash. The most common simplification is to expand a disjunction that has only one expansion possibility and detect a clash if there is a disjunction in $\mathcal{L}(x)$ with no expansion possibilities [11]. For example, having the following literals in the label of node x [5]:

$$\{C \sqcup (D_1 \sqcap D_2), (\neg D_1 \sqcup \neg D_2 \sqcup C), \neg C\} \subseteq \mathcal{L}(x) \tag{3.17}$$

BCP deterministically expands the disjunction $(C \sqcup (D_1 \sqcap D_2))$, adding $(D_1 \sqcap D_2)$ to $\mathcal{L}(x)$, because of $\neg C \in \mathcal{L}(x)$. The deterministic expansion of $(D_1 \sqcap D_2)$ adds both D_1 and D_2 to $\mathcal{L}(x)$ and allows BCP to identify $(\neg D_1 \sqcup \neg D_2 \sqcup C)$ as a clash, because $D_1, D_2, \neg C \subseteq \mathcal{L}(x)$.

This technique can be adapted to CB algorithms. It is applicable to a wide range of description logic languages without causing much overhead. However, BCP is more efficient with randomly generated problems and does not work well with real-world problems [18].

3.2.3 Dependency Directed Backtracking

Dependency Directed Backtracking is an effective optimization technique used for pruning irrelevant branches created by applying non-deterministic rules. The approach proposed

in [62] benefits from a more informed dependency-directed backtracking strategy to improve caching and *back-jumping*. The main idea is based on the fact that if a branching point is not involved in the occurrence of a clash, exploring other alternatives to this branching point would not eliminate the cause of the clash either. Labelling all the facts in the graph with dependency information allows the algorithm to identify the most recent non-deterministic branching points where an alternative choice might alleviate the cause of the clash. Back-jumping is a general form of directed backtracking in which each fact has a dependency set where all the branching points that it depends on are collected.

3.2.4 Caching

Caching is another well-known optimization technique used for improving the performance of tableau-based reasoners [5]. There are two types of caching, *satisfiability* and *unsatisfiability* caching. The former caches concept sets of node labels that are recognized to be satisfiable. In contrast, unsatisfiability caching creates cache entries containing sets of concepts known to be unsatisfiable.

In unsatisfiable caching, any *superset* of the cached set is also unsatisfiable. Therefore it is always desirable to cache the smallest set of concepts which is still unsatisfiable. Many of the concepts in the label of a node are not involved in a clash. A *Precise caching technique* allows the algorithm to extract the smallest subset of the concepts such that their combination is known to be unsatisfiable. A less precise caching may lead to performance degradation associated with dependency-directed backtracking. If the entire label of a node is added to the cache instead of adding the subset which causes the clash, in future cache retrievals, more dependency-directed backtracking may be required to check the

satisfiability.

[62] investigates creating unsatisfiable cache entries during backtracking. Roughly speaking, all the facts involved in a clash are back-traced to the node where the clash occurs, whereby an unsatisfiable cache entry may be generated according to the collected facts.

In the presence of nominals, the satisfiability status of a node can not be directly used for producing cache entries because different nodes of the completion graph can refer to a nominal; new concepts may be propagated to a previously cached node. The *forest caching* technique [58, 59] may be utilized instead, saving the completion graph's state after an initial consistency check.

3.2.5 Signature Calculus

The *Signature calculus* proposed for $\mathcal{ALCQH}$ [24] tries to address the inefficiency of standard tableau algorithms caused by large values in QCRs. It uses a compact representation, *a signature (proxy individual)*, to denote a set of individuals.

The signature calculus creates one proxy individual for satisfying each at-least restriction $\geq nR.C$. A proxy individual represents n identical individuals, which are R successors and share a common restriction C. A proxy individual x may be split into more than one if the signature is extended for a subset of individuals presented by x. Two proxy individuals may be merged if an at-most restriction $\leq mS.D$ is violated due to an excessive number of potential S-successors.

Although the goal of this algorithm was to address the inefficiency of reasoning about QCRs containing large numbers, the algorithm's complexity still depends on N, where N

is the sum of all the numbers occurring in QCRs. However, it has a major improvement compared with standard tableau algorithms by producing a well-structured search space, which allows pruning in the case of a clash occurrence.

3.2.6 Algebraic Method

Most DL reasoners proposed so far deal with number restrictions and their interaction with nominals in a blind way. These algorithms try to find a solution that satisfies all restrictions imposed by QCRs by exhausting all possible cases. So they are highly non-deterministic and inefficient, especially in handling QCRs containing large numbers. None of the optimization techniques discussed previously can address this problem efficiently.

The algorithm presented in [27] was the first to address the problem of having large numbers in QCRs by combining tableau-based algorithms with algebraic reasoners. The basic idea of [27] is to reduce a concept satisfiability problem to a feasibility testing problem that can be solved using arithmetic methods.

The successful implementation of algebraic reasoning in [20] shows that it can effectively improve reasoning performance in DLs supporting QCRs. [19] extends this idea for reasoning in $\mathcal{SHOQ}$ which supports QCRs and nominals.

The algorithm presented in this proposal benefits from the advantages of algebraic reasoning while avoiding the drawbacks of generating a large number of variables using the column generation technique. The main difference is that our algorithm also incorporates features of CB algorithms, as discussed in Chapter 2.

3.3 Extending Consequence-based Algorithms

Consequence-based (CB) algorithms were proposed for reasoning in less expressive description logic (DLs). Reasoners developed based on these algorithms for the EL family of DLs outperform even highly optimized tableau-based reasoners in handling large ontologies, such as SNOMED CT [34]. However, extending CB procedures to more expressive DLs is accompanied by difficulties, which are briefly discussed in Section 1.3.

Supporting new constructors requires introducing new inference rules to preserve their semantics. The inference rules must be capable of handling all restrictions imposed by new constructors while considering their implicit interactions with current constructors. We presented the basic idea of CB algorithms introduced for $\mathcal{EL}$ in Section 2.4.2. The following subsections discuss and analyze the extensions proposed for handling more expressive DLs.

3.3.1 CB Reasoning for Horn ontologies

After the successful development of CB reasoners for $\mathcal{EL}$, extending these reasoners to more expressive DLs became trending research. One of the first efforts was adopting this reasoning technique to Horn ontologies [34, 44]. The algorithm proposed in [34] extends the $\mathcal{EL}^{++}$ CB reasoner [4] to classify Horn-$\mathcal{SHIQ}$ ontologies. For this purpose, the procedure first applies normalization rules to obtain an ontology containing only axioms of form $A_1 \sqsubseteq A_2$, $A \sqsubseteq C_+$ and $C_- \sqsubseteq A$ where C_+ can only be of the form $\top$, $\bot$, A, $\neg C$, $C \sqcap D$, $\exists R.C$, $\forall R.C$, $\geq nS.C$ or $\leq 1S.B$ and C_- can only be of the form $\top$, $\bot$, A, $C \sqcap D$, $C \sqcup D$, $\exists R.C$ or $\geq 1R.C$.

Then the procedure converts universal restrictions to inverse rules using the equivalence $C \sqsubseteq \forall R.D \equiv \exists R^{-}.C \sqsubseteq D$ and introduces four new inference rules for handling the interaction of inverse roles, functional roles and universal restrictions. The procedure uses the inference rules to produce axioms of the form $M \sqsubseteq C$ and $M \sqsubseteq \exists R.N$, where M and N are conjunction of concepts, and C is an atomic concept. They are applied exhaustively until no more rule is applicable. The resulting ontology $\mathcal{O}'$ is called the *saturation* of $\mathcal{O}$, which is why CB reasoners are also known as saturation-based.

The number of produced axioms of the form $H \sqsubseteq C$ and $H \sqsubseteq \exists R.K$ can be exponential in the number of atomic concepts occurring in ontology $\mathcal{O}$. The worst-case time complexity of this algorithm is exponential in the size of the input ontology.

3.3.2 CB Reasoning Beyond Horn Ontologies

Consequence-based procedures discussed so far were only applicable to Horn DLs. The main difficulty in extending these algorithms is dealing with non-determinism caused by non-Horn axioms, which require reasoning by case. The algorithm proposed in [56] addressed this problem by combining CB algorithms with ordered resolutions.

To handle axioms containing disjunctions, the $\mathcal{ALCH}$ CB algorithm [56] generalizes the form of result axioms from $H \sqsubseteq C$ and $H \sqsubseteq \exists R.K$ to $H \sqsubseteq M$ and $H \sqsubseteq N \sqcup \exists R.K$ where M, N are disjunctions of atomic concepts. The reasoning procedure consists of two stages: A *normalization stage* during which a *structural transformation* is used to simplify the axioms [54], and a *saturation stage*, which derives new axioms by applying inference rules. The runtime of this algorithm is exponential in the size of the input ontology.

To use this procedure in practical reasoning systems, a set of optimization techniques are employed based on introducing a new notion, *contexts*. A context is a conjunction on the left-hand side of derived axioms. Introducing new concepts in a goal-directed way by maintaining a list of active contexts, using the same context to represent its super contexts, and dividing the set of contexts into several partitions are the optimization techniques used in [56] to improve practical results.

3.3.3 CB Reasoning with Nominals

After successfully developing a CB reasoning procedure for DL $\mathcal{EL}$, continued research strove to support additional features on top of $\mathcal{EL}$ preferably with preserving its polynomial complexity. One of the most interesting features that $\mathcal{EL}^{++}$ adds to $\mathcal{EL}$ are *nominals*. In general, one can use them to enumerate all possible instances of a concept. But since disjunctions are not allowed in $\mathcal{EL}$, the use of nominals becomes very restricted and is limited to singleton concepts.

In practice, however, ontology designers avoid using nominals because of minimal practical support for nominals in OWL. Amongst the currently available $\mathcal{EL}$ reasoners, Snorocket provides no support for nominals [37], CEL only supports ABox assertions [6], and the support for nominals in jCEL is incomplete [17].

Designing an efficient algorithm for handling nominals turns out to be challenging because even in low expressive description logics such as $\mathcal{EL}$, mere non-emptiness of concepts may result in new entailments. In other words, solely knowing that a concept has at least one instance can lead to a new subsumption between atomic concepts. Reasoning with nominals gets much more complicated in the presence of more expressive

constructors, especially regarding the interaction of nominals and QCRs (Chapter 1.3).

The first consequences-based algorithm proposed for reasoning about nominals is in the $\mathcal{EL}$ family of DLs [35]. The idea is to deal with nominals by tracking the non-emptiness of a concept to others. For this purpose, the $\mathcal{ELO}$ reasoner introduces a new type of axioms $C \rightsquigarrow D$ called *reachability axioms*. Also, they use *conditional axioms* $G : C \sqsubseteq D$ to state that the subsumption $C \sqsubseteq D$ only holds if the concept G has at least one instance (is not empty).

This sound, complete, and goal-oriented algorithm computes all subsumption relations between a goal concept G and all other atomic concepts occurring in the ontology $\mathcal{O}$ in a single round. The worst-case time complexity of the algorithm is polynomial, and the maximum number of derived axioms is quadratic in the size of the ontology.

3.3.4 Framework for CB Reasoning

The first general framework for CB algorithms was introduced in [57] for reasoning in $\mathcal{ALCI}$ and $\mathcal{SHI}$ which support inverse roles in addition to the $\mathcal{ALC}$ constructors.

We believe that the proposed structure has the potential to improve the practical performance of CB algorithms. We have adopted this idea in our research, but we have extended it to the much more expressive description logic SHOQ, which supports the interaction of nominals and QCRs. We have also applied advanced optimization techniques to make our reasoning system efficient in practice, as discussed in Chapter 5.

The proposed framework is a *decomposition* $\mathcal{D}$ of an ontology $\mathcal{O}$ and queries $\mathcal{Q}$ (i.e., axioms of a particular form). Roughly speaking, $\mathcal{D}$ is a graph-like structure that represents the models of $\mathcal{O}$ in relevance with the queries in $\mathcal{Q}$. Each vertex of $\mathcal{D}$ represents a set of

concepts always held in that vertex, and each edge of $\mathcal{D}$ identifies a relation between such concepts.

The presented algorithm in [57] takes a *normalized* $\mathcal{ALCI}$ ontology $\mathcal{O}$ (i.e., only atomic concepts are allowed on the left-hand side of axioms) and a set of queries $\mathcal{Q}$ and computes the status of $O \models K \sqsubseteq M$ for each query $K \sqsubseteq M \in \mathcal{Q}$ in exponential time.

3.3.5 Extending CB Reasoning to $\mathcal{SRIQ}$ and $\mathcal{SROIQ}$

Despite the excellent practical performance of CB reasoners, they had never been extended for expressive DLs containing QCRs. The first effort for developing CB algorithms to deal with numerical restrictions implied by QCRs has been proposed in [10] for reasoning in the DL $\mathcal{SRIQ}$.

The $\mathcal{SRIQ}$ consequence-based reasoning algorithm extends the framework introduced in [57] to handle QCRs. Since counting quantifiers require equality reasoning, the $\mathcal{SRIQ}$ reasoner [10] encodes ontology axioms into DL clauses, which allow equality reasoning. The proposed calculus [10] must be capable of capturing constraint (3.18) and its consequences, while standard DL axioms cannot explicitly refer to specific successors and predecessors.

So [10] uses DL clauses over terms x, $f_i(x)$, and y, where the variable x represents the ground term, $f_i(x)$ represents the f_i-successor of x, and y represents the predecessor of x. Thus, the predecessor and the successors of x can be identified by name. Number restrictions can be compiled into DL clauses by applying the following translations based on the correspondence between DLs and first-order logic.

$$B \sqsubseteq \geq n\ R.C \quad \rightsquigarrow \quad \begin{cases} B(x) \rightarrow R(x, f_i(x)) & \text{for } 1 \leq i \leq n \\ B(x) \rightarrow C(f_i(x)) & \text{for } 1 \leq i \leq n \\ B(x) \rightarrow f_i(x) \not\approx f_j(x) & \text{for } 1 \leq i < j \leq n \end{cases} \tag{3.18}$$

$$B \sqsubseteq \leq n\ R.C \quad \rightsquigarrow \quad \begin{cases} R(z_1, x) \wedge C(x) \rightarrow R_C(z_1, x) & \text{for fresh } R_C \\ B(x) \wedge \bigwedge_{1 \leq i \leq n+1} R_C(x, z_i) \rightarrow \bigvee_{1 \leq i < j \leq n+1} z_i \approx z_j \end{cases} \tag{3.19}$$

In 2018, Cucala et al. extended CB algorithms to the description logic $\mathcal{ALCHOIQ}$ [16], following the work of Bate et al. [8]. Their algorithm represents all derived consequences in contexts as context clauses in many-sorted equational logic rather than DL-style axioms. Context clauses are defined analogously to [8], with the main difference being that they allow context literals to mention named individuals. Additionally, the algorithm defines a distinguished root context where most inferences involving such literals take place. This root context represents the non-tree-like part of the model and exchanges information with other contexts using some newly devised inference rules.

Compiling away number restrictions by enumerating all equalities is not applicable for ontologies with a large number of QCRs nor QCRs containing large values. For classifying an ontology which contains p at-least restrictions of the form $\geq n_i\ R_i.C_i$ and q at-most restrictions of the form $\leq m_i\ S_i.D_i$ the algorithms would transform them to $3N$ clauses where $N = \Sigma_{i=1}^{p} n_i$, each clause introduces a new variable $f_i(x)$. For satisfying every violated at-most restriction $\leq m\ S.D$, the algorithm would create $N - m$ clauses that each consist of $\binom{N}{2} =$ equality disjunctions. The algorithm's runtime highly depends on the number of QCRs and the values of numbers in at-most and at-least restrictions. It is worth

$$\texttt{GraduateStudent} \sqsubseteq \geq 2\ \texttt{hasCourse.GraduateCourse} \tag{3.20}$$
$$\texttt{GraduateStudent} \sqsubseteq \geq 2\ \texttt{hasCourse.ComputerCourse} \tag{3.21}$$
$$\texttt{GraduateStudent} \sqsubseteq \geq 2\ \texttt{hasCourse.MathCourse} \tag{3.22}$$
$$\texttt{GraduateStudent} \sqsubseteq \leq 5\ \texttt{hasCourse} \tag{3.23}$$

(A) Original ontology $\mathcal{O}$

$$\texttt{GraduateStudent}(x) \rightarrow \texttt{hasCourse}(x, f_i(x)) \quad \text{for } 1 \leq i \leq 2 \tag{3.24}$$
$$\texttt{GraduateStudent}(x) \rightarrow \texttt{GraduateCourse}(f_i(x)) \quad \text{for } 1 \leq i \leq 2 \tag{3.25}$$
$$\texttt{GraduateStudent}(x) \rightarrow f_i(x) \not\approx f_j(x) \quad \text{for } 1 \leq i < j \leq 2 \tag{3.26}$$
$$\texttt{GraduateStudent}(x) \rightarrow \texttt{hasCourse}(x, f_i(x)) \quad \text{for } 3 \leq i \leq 4 \tag{3.27}$$
$$\texttt{GraduateStudent}(x) \rightarrow \texttt{ComputerCourse}(f_i(x)) \quad \text{for } 3 \leq i \leq 4 \tag{3.28}$$
$$\texttt{GraduateStudent}(x) \rightarrow f_i(x) \not\approx f_j(x) \quad \text{for } 3 \leq i < j \leq 4 \tag{3.29}$$
$$\texttt{GraduateStudent}(x) \rightarrow \texttt{hasCourse}(x, f_i(x)) \quad \text{for } 5 \leq i \leq 6 \tag{3.30}$$
$$\texttt{GraduateStudent}(x) \rightarrow \texttt{MathCourse}(f_i(x)) \quad \text{for } 5 \leq i \leq 6 \tag{3.31}$$
$$\texttt{GraduateStudent}(x) \rightarrow f_i(x) \not\approx f_j(x) \quad \text{for } 5 \leq i < j \leq 6 \tag{3.32}$$
$$\texttt{GraduateStudent}(x) \wedge \bigwedge_{1 \leq i \leq 6} \texttt{hasCourse}(x, f_i(x)) \rightarrow \bigvee_{1 \leq i < j \leq 6} f_i(x) \approx f_j(x) \tag{3.33}$$

(B) The ontology $\mathcal{O}'$ produced by $\mathcal{SRIQ}$ algorithm during preprocessing step

FIGURE 3.1: High Non-determinism Problem of $\mathcal{SRIQ}$ algorithm

mentioning that this procedure is not a worst case, but it is a regular case that is very likely to happen.

For example, considering a minimal ontology $\mathcal{O}$ shown in Figure 3.1a, this $\mathcal{SRIQ}$ algorithm would translate it to ontology O' in Figure 3.1b. Clause (3.27) includes fifteen disjunct equalities, which impose a high volume of non-determinism.

We try to overcome this problem by handling QCRs directly (without expanding them) using *Integer Linear Programming* techniques and, more specifically, *row and column generation* optimizations. The other main difference is that our proposed reasoner aims at handling the interaction of nominals and QCRs. These two constructors are used for imposing different types of numerical restrictions. Since their consequences may highly affect each other, these constraints need to be satisfied altogether. Also, nominals are global in the whole ontology, so the numerical restrictions may not be handled locally anymore. Our algorithm was the first CB calculus for reasoning about $\mathcal{SHOQ}$ [33].

3.4 Summary and Conclusion

In this chapter, we have introduced the state-of-the-art techniques for reasoning about the interaction of *nominals* and QCRs. One can categorize DL reasoners into two categories: Tableau-based and consequence-based.

Although tableau-based reasoners can reason in the presence of expressive DL constructors, they are not practically applicable unless they are highly optimized. We have classified and discussed possible optimization techniques to improve tableau-based algorithms. However, there are still some inherent deficiencies that cannot be eliminated by applying optimization techniques, such as the need to build a counter-model for testing

subsumption relations between each pair of concepts occurring in the ontology. Many of these optimization techniques are also applicable to CB algorithms.

CB reasoners were proposed to address the weakness of tableau-based reasoners by introducing a new methodology. Initially, they were developed for lightweight DLs such as $\mathcal{EL}$, but they have since been extended to more expressive DLs. This chapter traces their evolution process all the way to $\mathcal{SROIQ}$. It can be seen that the proposed approach for reasoning about QCRs is not efficient in practice for QCRs containing large values or ontologies with a large number of QCRs. In the next chapter, we present our proposed algorithm, which was the first extension of CB algorithms to $\mathcal{SHOQ}$ that allows the interaction of nominals and QCRs. It also employs linear programming to handle numerical restrictions imposed by these expressive constructors.

Part II

Calculus and Applications

Chapter 4

Preliminaries

This chapter begins with a motivating example that will be used throughout the thesis to illustrate the notions and functionality of the proposed algorithm. Next, we describe the normalization process, which is a prerequisite for the reasoning process. Finally, we review some well-known definitions and review the notations that will be used in the remainder of the thesis.

Example 4.1 (Motivating Example)**.** The partial information about graduate students in a computer science department (CS) is presented as:

(i) CS has at most 45 students

(ii) It has at least 30 students who are not course-based

(iii) It has at least 20 students in Lab-A who are independent, supervised, or PhD students

(iv) It has at most 2 PhD students in Lab-A

(v) Every independent student is course based

(vi) CS has at least 3 supervised students in Lab-A

(vii) CS has at most 2 supervised students in Lab-A

It can be concluded that (i)-(v) are satisfiable and entail (vi). This holds because (vii) is the negation of (vi) and (i)-(v) and (vii) are unsatisfiable. We need to find a satisfying solution or the minimum set of constraints causing the unsatisfiability, which later can be used in the reasoning process.

4.1 Normalization

In this section, we show how to transform an arbitrary $\mathcal{SHOQ}$ ontology $\mathcal{O}$ into a normalized $\mathcal{ALCHOQ}$ ontology $\mathcal{O}'$ such that $\mathcal{O}'$ entails the same subsumption relations as $\mathcal{O}$ over atomic concepts occurring in $\mathcal{O}$. The derived ontology $\mathcal{O}'$ is proved to be a conservative extension of $\mathcal{O}$ [55].

The normalization method consists of two main phases: first, normalizing an arbitrary $\mathcal{SHOQ}$ ontology $\mathcal{O}$ by applying a *structural transformation* [34] and producing a normalized $\mathcal{SHOQ}$ ontology $\mathcal{O}'$. Second, eliminating transitivity axioms from $\mathcal{O}'$ and producing a normalized $\mathcal{ALCHOQ}$ ontology $\mathcal{O}''$. We describe the two normalization steps below.

4.1.1 Structural Transformation

Recall from Chapter 2 (Definition 2.9) that a concept C occurs positively (negatively) in C, if it occurs positively (negatively) in $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\geq nR.C$ and $D \sqsubseteq C$ or if it occurs negatively (positively) in $\neg C$, $\leq nR.C$ and $C \sqsubseteq D$.

Let $\mathcal{O}$ be a $\mathcal{SHOQ}$ ontology. The transformation introduces a fresh atomic concept A_C (does not exist in $\mathcal{O}$) for each concept description C occurring in $\mathcal{O}$. The *structural transformation* of a concept C is denoted by $st(C)$ and is defined as follows:

$$
\begin{aligned}
st(A) &= A & st(C \sqcap D) &= A_C \sqcap A_D \\
st(\top) &= \top & st(C \sqcup D) &= A_C \sqcup A_D \\
st(\bot) &= \bot & st(\exists R.C) &= \exists R.A_C \\
st(\neg C) &= \neg A_C & st(\forall R.C) &= \forall R.A_C \\
st(o) &= o & st(\leq nR.C) &= \leq nR.A_C \\
& & st(\geq nR.C) &= \geq nR.A_C
\end{aligned}
$$

The result of applying the *structural transformation* to $\mathcal{O}$ is a new ontology $\mathcal{O}'$ which contains all role inclusion and role transitivity axioms in $\mathcal{O}$ in addition to the following axioms:

- $st(C) \sqsubseteq A_C$ for every concept C occurring negatively in O
- $A_C \sqsubseteq st(C)$ for every concept C occurring positively in $\mathcal{O}$
- $A_C \sqsubseteq A_D$ for every concept inclusion $C \sqsubseteq D \in \mathcal{O}$.

Note that $\mathcal{O}'$ is not still fully normalized, because its concept inclusion axioms are of the form $st(C) \sqsubseteq A_C$, $A_C \sqsubseteq st(C)$, and $A_C \sqsubseteq A_D$, which may include axioms like $A_{C \sqcap D} \sqsubseteq A_C \sqcap A_D$ that are not normal clauses. We rewrite such axioms into normal clauses

using the following equivalences:

$$A_{C\sqcap D} \sqsubseteq A_C \sqcap A_D \quad \Rightarrow \; A_{C\sqcap D} \sqsubseteq A_C \text{ and } A_{C\sqcap D} \sqsubseteq A_C \tag{4.1}$$

$$A_C \sqcup A_D \sqsubseteq A_{C\sqcup D} \quad \Rightarrow \; A_C \sqsubseteq A_{C\sqcup D} \text{ and } A_D \sqsubseteq A_{C\sqcup D} \tag{4.2}$$

$$A_{\neg C} \sqsubseteq \neg A_C \quad \Rightarrow \; A_{\neg C} \sqcap A_C \sqsubseteq \bot \tag{4.3}$$

$$\neg A_C \sqsubseteq A_{\neg C} \quad \Rightarrow \; \top \sqsubseteq A_{\neg C} \sqcup A_C \tag{4.4}$$

$$\exists R.A_C \sqsubseteq A_{\exists R.C} \quad \Rightarrow \; \top \sqsubseteq A_{\exists R.C} \sqcup \forall R.A_{\neg C} \text{ and } A_C \sqcap A_{\neg C} \sqsubseteq \bot \tag{4.5}$$

$$\forall R.A_C \sqsubseteq A_{\forall R.C} \quad \Rightarrow \; \top \sqsubseteq A_{\forall R.C} \sqcup \exists R.A_{\neg C} \text{ and } A_C \sqcap A_{\neg C} \sqsubseteq \bot \tag{4.6}$$

$$\geq nR.A_C \sqsubseteq A_{\geq nR.C} \Rightarrow \; \top \sqsubseteq A_{\geq nR.C} \sqcup \leq (n-1)\; R.A_C \tag{4.7}$$

$$\leq nR.A_C \sqsubseteq A_{\leq nR.C} \Rightarrow \; \top \sqsubseteq A_{\leq nR.C} \sqcup \geq (n+1)\; R.A_C \tag{4.8}$$

Proposition 4.1. Let $\mathcal{O}'$ be a normalized $\mathcal{SHOQ}$ ontology obtained by applying the *structural transformation* on $\mathcal{O}$. Then for every query $q \in \mathcal{Q}$, we have $\mathcal{O} \models q$ iff $\mathcal{O}' \models q$. Moreover, the size of $\mathcal{O}'$ is linear in the size of $\mathcal{O}$ and $\mathcal{O}'$ can be computed from $\mathcal{O}$ in polynomial time.

4.1.2 Transform a $\mathcal{SHOQ}$ ontology to an $\mathcal{ALCHOQ}$ ontology

To transform a normalized $\mathcal{SHOQ}$ ontology into a normalized $\mathcal{ALCHOQ}$, we need to eliminate all role transitivity axioms. Let $A_{R,C}$ and $B_{R,C}$ denote fresh atomic concepts unique for role R and concept C, where C is an atomic concept occurring in $\mathcal{O}$. The elimination process contains two steps: step 1 restricts the occurrence of universal restrictions to clauses of the form $A \sqsubseteq \forall R.C$ and step 2 uses the well-known "box pushing" technique

to encode transitivity axioms [57]. We explain these steps by introducing two ontologies obtained by transferring $\mathcal{O}$ as follows.

- Ontology $\mathcal{O}_1$ is obtained from $\mathcal{O}$ by replacing each literal $\forall R.C \in M$ with $A_{R,C}$ in each clause $K \sqsubseteq M \in \mathcal{O}$. The normalization process also adds the axiom $A_{R,C} \sqsubseteq \forall R.C$ to ontology $\mathcal{O}_1$ concerning each literal $\forall R.C$.

- Ontology $\mathcal{O}_2$ is obtained from $\mathcal{O}_1$ by adding the following clauses relating to each clause $A \sqsubseteq \forall R.C \in \mathcal{O}_1$ and each role S such that $S \sqsubseteq_* R$ and $\texttt{Trans}(S) \in \mathcal{O}$. Intuitively, we have used $B_{S,C}$ to propagate C to all elements which are reachable from A elements through a S-chain.

$$A \sqsubseteq \forall S.B_{S,C} \qquad B_{S,C} \sqsubseteq \forall S.B_{S,C} \qquad B_{S,C} \sqsubseteq C$$

In this section, we showed that by applying the structural transformation and eliminating transitivity, one could transfer every $\mathcal{SHOQ}$ ontology $\mathcal{O}$ into a normalized $\mathcal{ALCHOQ}$ ontology $\mathcal{O}'$ in polynomial time. Such that $\mathcal{O}'$ is a conservative extension of $\mathcal{O}$, i.e., entails the same subsumption relations between atomic concepts occurring in $\mathcal{O}$.

Through the rest of this thesis, our algorithm accepts a normalized $\mathcal{ALCHOQ}$ ontology $\mathcal{O}$ and a finite set of queries $\mathcal{Q}$ and computes the status of $\mathcal{O} \models q$ for each query $q \in \mathcal{Q}$. Note that the algorithm handles existential restrictions of the form $\exists\ T.A$ by converting them to their equivalent at-least restrictions of the form $\geq 1\ T.A$.

4.2 First-Order Logic

We use the two-variable fragment of first-order logic (without equality) that admits *Counting Quantifiers* (CQs) rather than just universal and existential. *First-Order Logic* (FOL) is able to express facts about the number of objects that have a certain property using equality [22]. However, a more succinct formalization is using CQs, which have the forms $\exists_{\geq n}$ or $\exists_{\leq n}$, with $n \in \mathbb{N}$. The formula $\forall x \exists_{\geq n} y \ \varphi(x, y)$ expresses that there are at least n mappings for variable y in the domain such that the formula $\varphi(x, y)$ is satisfied [53].

A signature S is a pair of finite sets $(S_{\mathcal{P}}, S_{\mathcal{C}})$ where $S_{\mathcal{P}}$ is a set of *predicate symbols* and $S_{\mathcal{C}}$ is a set of *constant symbols*. The set of predicate symbols $S_{\mathcal{P}}$ includes two disjoint subsets, unary predicates $S_{\mathcal{U}}$ and binary predicates $S_{\mathcal{B}}$. A *term* is a constant or variable; an *atom* is a constant or a unary predicate; and a *literal* L is a quantifier or an atom, and S_L is the set of all literals.

A *clause* is a formula of the form $\forall x(K \rightarrow M)$, where the *body* K is a conjunction of atoms, the *head* M is a disjunction of literals and the variable x is occurring in the clause. The universal quantifier $\forall x$ is usually omitted, but the quantifiers over the variable y are always expressed explicitly. Conjunctions and disjunctions are identified as possibly empty sets which are used in standard set operations. The empty conjunction (disjunction) is abbreviated as $\top$ ($\bot$).

A substitution σ binds a term to each variable, which can be expressed as $\{x \mapsto t_1, y \mapsto t_2\}$. The result of applying a substitution to clause $K \rightarrow M$ or unary predicate C is written as $(K \rightarrow M)_\sigma$ and C_σ, respectively. The substitution $\sigma = \{x \mapsto t\}$ may be represented as σ_t. A unary predicate is *ground* if it has no variable.

An interpretation $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set of elements called

the domain, and $\cdot^{\mathcal{I}}$ is an interpretation function that maps every $C \in S_{\mathcal{U}}$ to a subset of $\Delta^{\mathcal{I}}$, every $R \in S_{\mathcal{B}}$ to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and each $o \in S_{\mathcal{C}}$ to $o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The interpretation $\mathcal{I}$ is extended to complex constructs as shown in Table 4.1, where $\#R^{\mathcal{I}}(a, C)$ denotes the cardinality of $\{a \mid (a, b) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$. Each FOL sentence φ is interpreted as a truth-value $\varphi^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies a sentence φ (written as $\mathcal{I} \models \varphi$) if $\varphi^{\mathcal{I}} = \mathit{true}$.

4.3 Description Logics Clauses

DL expressions can be transformed into *DL-clauses* in polynomial time while preserving satisfiability and entailment, as indicated in Table 4.1. DL clauses are a subset of FOL clauses that we use for expressing DL axioms and are defined formally in the following. Our signature uses only two variables, the *central* variable x and a neighbour variable y, which might represent $n \in \mathbb{N}$ individuals in the domain. The set $S_{\mathcal{U}}$ contains all atomic concepts in $\mathcal{O}$ and $S_{\mathcal{B}}$ is the set of atomic roles occurring in $\mathcal{O}$, corresponding to the set of unary and binary predicates in FOL, respectively. The set of constant symbols $S_{\mathcal{C}}$ contains all nominals occurring in the ontology.

A *DL-term* is a constant or variable that has the form x, y or o; and a *DL-predicate* has the form $C(x), C(o), C(y)$ or $R(x, y)$. A *DL-quantifier* is either an *at-least* ($\exists_{\geq n} y$), an *at-most* ($\exists_{\leq m} y$) or a *universal* quantifier ($\forall y$). Based on the above semantics, the negation normal form of CQs is $\neg\exists_{\geq n}\varphi \equiv \exists_{\leq n-1}\varphi$ and $\neg\exists_{\leq n}\varphi \equiv \exists_{\geq n+1}\varphi$. *Existential-quantifiers* $\exists y$ are a particular form of at-least quantifiers where $n = 1$ and the *exact quantifier* $\exists_{=n} y$ abbreviates $\{\exists_{\geq n} y, \exists_{\leq n} y\}$. An element $y \in \Delta^{\mathcal{I}}$ is called an *R-successor*, if an element $x \in \Delta^{\mathcal{I}}$ exists such that $(x, y) \in R^{\mathcal{I}}$, where $R \in S_{\mathcal{B}}$;

TABLE 4.1: Translating $\mathcal{SHOQ}$ Syntax to DL-clauses and Semantics

$^*\varphi(x,y) = (R(x,y) \wedge C(y))$

DL Syntax	DL-Clause	Semantics
Concepts		
$C_1 \sqcap C_2$	$C_1(x) \wedge C_2(x)$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
$C_1 \sqcup C_2$	$C_1(x) \vee C_2(x)$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
$\exists R.C$	$\exists y\ \varphi(x,y)$	$\{a \mid \exists b : (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$
$\forall R.C$	$\forall y\ \varphi(x,y)$	$\{a \mid \forall b : (a,b) \in R^{\mathcal{I}} \Rightarrow b \in C^{\mathcal{I}}\}$
$\geq n\ R.C$	$\exists_{\geq n} y\ \varphi(x,y)$	$\{a \mid \#R^{\mathcal{I}}(a,C) \geq n\}$
$\leq n\ R.C$	$\exists_{\leq n} y\ \varphi(x,y)$	$\{a \mid \#R^{\mathcal{I}}(a,C) \leq n\}$
o	$\exists_{=1} x\ O(x)$	$\#\{o^{\mathcal{I}}\} = 1$
Axioms		
$C_1 \sqsubseteq C_2$	$C_1(x) \rightarrow C_2(x)$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
$R_1 \sqsubseteq R_2$	$R_1(x,y) \rightarrow R_2(x,y)$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$
$\{o\} \sqsubseteq C$	$\top \rightarrow C(o)$	$o^{\mathcal{I}} \in C^{\mathcal{I}}$

A *DL-atom* is a unary predicate $C \in S_{\mathcal{U}}$ or a constant $o \in S_{\mathcal{C}}$. A *DL-literal* is a DL-atom or a DL-quantifier. A *normal DL-clause* contains only DL-atoms in the body and DL-literals in the head. An *ontology* is a finite set of DL-clauses. A DL-clause is called atomic if it contains only atoms, and a *query clause* is an atomic clause. A clause $K' \rightarrow M'$ is a strengthening of a clause $K \rightarrow M$ if $K' \subseteq K$ and $M' \subseteq M$. We use $K \rightarrow M \in_* \mathcal{CL}$ to show that a set of clauses $\mathcal{CL}$ contains at least one strengthening of $K \rightarrow M$.

Normal DL-clauses are of the form $\bigwedge_{i=1}^{n} L_i \sqsubseteq \bigvee_{j=1}^{m} L_j$ where each L_i is either $C(x)$ or $O(x)$ and each L_j is $C(x)$, $O(x)$, $\exists y\ \varphi(x,y)$, $\forall y\ \varphi(x,y)$, $\exists_{\geq n} y\ \varphi(x,y)$ or $\exists_{\leq n} y\ \varphi(x,y)$, where $\varphi(x,y) = (R(x,y) \wedge C(y))$, $C(x)$ is an atomic concept, $O(x)$ is a nominal and $R(x,y)$ is an atomic role. A $\mathcal{SHOQ}$ ontology can be converted to a *normalized* ontology, which contains only Normal DL-clauses using the *structural transformation* discussed in Section 4.1.1. A normalized $\mathcal{SHOQ}$ ontology can be rewritten to a normal $\mathcal{ALCHOQ}$ ontology

by encoding role transitivity axioms [55]. Afterward, we translate normalized axioms to DL-clauses as shown in Table 4.1. In description logic, *nominals* are defined as atomic concepts with exactly one instance that will be interpreted as singleton sets. Following this definition, we introduce a function π to map every nominal o occurring in the ontology to a fresh unary predicate $\pi(o) = O(x)$, and we add $\top \rightarrow \exists_{=1} x\ O(x)$ to the ontology, for each unary predicate $O(x)$. We denote the set of all these fresh unary predicates as $S_O = \{\pi(o) \mid o \in S_o\}$ of size n_o.

Example 4.2 (Ontology Normalization)**.** This example shows the normalization of the formalized version of Example 4.1:

$$
\begin{array}{lll}
(i) & cs \sqsubseteq \leq 45\, h.s & \rightsquigarrow \forall x\ cs(x) \rightarrow \exists_{\leq 45} y\ (h(x,y) \wedge s(y)) \\
(ii) & cs \sqsubseteq \geq 30\, h.(s \sqcap \neg c) & \rightsquigarrow \forall x\ cs(x) \rightarrow \exists_{\geq 30} y\ (h(x,y) \wedge s(y) \\
 & & \quad \wedge \neg c(y)) \\
(iii) & cs \sqsubseteq \geq 20\, h.(s \sqcap a \sqcap (i \sqcup u \sqcup p)) & \rightsquigarrow \forall x\ cs(x) \rightarrow \exists_{\geq 20} y\ (h(x,y) \wedge s(y) \\
 & & \quad \wedge a(y) \wedge (i(y) \vee u(y) \vee p(y))) \\
(iv) & cs \sqsubseteq \leq 2\, h.(p \sqcap a) & \rightsquigarrow \forall x\ cs(x) \rightarrow \exists_{\leq 2} y\ (h(x,y) \wedge p(y) \\
 & & \quad \wedge a(y)) \\
(v) & i \sqsubseteq c & \rightsquigarrow \forall x\ i(x) \rightarrow c(x)
\end{array}
$$

The set of CQs ($\mathcal{Q}$) includes the statements $(i) - (iv)$, and the set of related clauses ($\mathcal{C}$) only contains statement (v). Following the normalization process discussed above, we

introduce new predicates, p_1, p_2, p_3, p_4, corresponding to the CQs in $\mathcal{Q}$, so that CQs are over atoms only. After this step, we have:

$$\begin{aligned}
\mathcal{Q} &= \{\exists_{\leq 45}\, y\ p_1(x,y), \exists_{\geq 30}\, y\ p_2(x,y), \exists_{\geq 20}\, y\ p_3(x,y), \exists_{\leq 2}\, x\ p_4(x)\} \\
\mathcal{C} &= \{\forall x,y\ h(x,y) \wedge s(y) \rightarrow p_1(x,y), \forall x\ p(x) \wedge a(x) \rightarrow p_4(x), \\
&\quad \forall x\ i(x) \rightarrow c(x), \forall x,y\ p_2(x,y) \rightarrow h(x,y) \wedge s(y) \wedge \neg c(y), \\
&\quad \forall x,y\ p_3(x,y) \rightarrow h(x,y) \wedge s(y) \wedge a(y) \wedge (i(y) \vee u(y) \vee p(y))\}
\end{aligned}$$

Applying the structural transformation to the last two clauses in $\mathcal{C}$ results in ($\forall x, y$ is omitted for the sake of brevity): $p_2(x,y) \rightarrow h(x,y)$, $p_2(x,y) \rightarrow s(y)$, $p_2(x,y) \wedge c(y) \rightarrow \bot$, $p_3(x,y) \rightarrow h(x,y)$, $p_3(x,y) \rightarrow s(y)$, $p_3(x,y) \rightarrow a(y)$, $p_3(x,y) \rightarrow i(y) \vee u(y) \vee p(y)$.

4.4 Used Notations

Description Logics $\mathcal{ALCHOQ}$ and $\mathcal{SHOQ}$ are defined w.r.t. non-empty and disjoint sets of *atomic concepts* N_C, *atomic roles* N_R and *named-individuals* (a.k.a *nominals*) N_I. A *DL-literal* is a concept of the form $A(x)$ or a *DL-quantifier* of the form $\exists_{\leq n} y\, \varphi(x,y)$, $\exists_{\geq n} y\, \varphi(x,y)$, $\forall y\, \varphi(x,y)$ or a nominal of the form $O(x)$, where $A(x)$ is an atomic concept (unary predicate), $R(x,y)$ is an atomic role (binary predicate), $O(x)$ is nominal and $\varphi(x,y) = R(x,y) \wedge A(y)$. Then, N_L shows the set of all literals. Unless otherwise stated, individual letters of the alphabet (possibly with sub- and/or superscripts) denote distinct notions as specified below:

- letters $A(x), B(x)$ denote atomic concepts,
- letters $C(x), D(x), E(x)$ denote concepts,
- letter L denotes a literal,
- letters $R(x, y), S(x, y)$ denote atomic roles,
- letter $O(x)$ denotes nominals,
- letter o denotes constants associated with nominals,
- letter K denotes the conjunction of literals, and
- letter M denotes a disjunction of literals.

Since disjunction and conjunction of literals are unordered and without repetition, we treat conjunction and disjunction of literals as the sets of literals, so we may use them in standard set operations. The conjunction and disjunction of literals may be empty, which are abbreviated to $\top$ and $\bot$, respectively.

Recall from Chapter 2 that an *axiom* is either an expression of the form $C_1(x) \to C_2(x)$ (*general concept inclusion*), $R_1(x, y) \to R_2(x, y)$ (*role inclusion*), or $Trans(R)$ (*role transitivity axiom*). A $\mathcal{SHOQ}$ ontology $\mathcal{O}$ is a set of axioms.

A *clause* is a general concept inclusion of the form $\bigwedge_{i=1}^{m} L_i \to \bigvee_{i=m+1}^{n} L_i$ where $0 \leq m \leq n$ and each L_i is a literal. A clause is *normal* if each L_i with $1 \leq i \leq m$ is an atomic concept, and a clause is a *query* if each L_i is an atomic concept. In a clause of the form $K \to M$, the conjunction K is called the *antecedent*, and the disjunction M is called the *consequence*. An ontology $\mathcal{O}$ is *normalized* if each GCI in $\mathcal{O}$ is a normal clause. A clause $K' \to M'$ is a strengthening of a clause $K \to M$ if $K' \subseteq K$ and $M' \subset M$. We use $K \to M \in_* \mathcal{C}$ to show that a set of clauses $\mathcal{C}$ contains at least one strengthening of $K \to M$.

Due to the presence of nominals, ABox assertions can be transformed to TBox axioms; a concept assertion $a : C$ can be written as $a \sqsubseteq C$ where a is a nominal and a role assertion

$R(a, b)$, can be written as $a \sqsubseteq \exists R.b$ where a and b are two nominals. So we only consider terminological reasoning in this research.

In DL $\mathcal{SHOQ}$, the interaction of transitive roles with number restrictions would cause undecidability. To avoid this interaction, $\mathcal{SHOQ}$ allows number restrictions only with *simple roles*, which are neither transitive nor have transitive sub-roles.

Chapter 5

Consequence-based Reasoning in SHOQ

In order to improve the performance of reasoning, one can reduce language expressivity to make reasoning easier. For example, the Horn family of DLs is obtained by removing disjunctions, which eliminates all disjunctions (a.k.a *or-branching* effect). However, reasoning with Horn-$\mathcal{ALCI}$ is still ExpTime-hard due to the interaction between existential and universal quantifiers (a.k.a *and-branching* effect). By further removing universal restrictions and inverse roles, one can obtain the $\mathcal{EL}$ family of DL for which polynomial time reasoning is ensured. However, there are many application domains that require expressive constructors, such as disjunctions, QCRs, and *nominals*. Thus, designing efficient reasoners for less expressive DLs is not always helpful.

An alternative is to explore new reasoning techniques and optimize reasoning processes. Although CB reasoners have proved to be very efficient in practice [4, 35], they are not optimized to deal with more expressive DLs such as $\mathcal{SHOQ}$. The CB reasoners which consider reasoning about QCRs [10, 16] do not efficiently handle QCRs containing large values or ontologies with a relatively large number of QCRs (See Figure 3.1). We believe that our CB algorithm would significantly improve the performance of reasoning

in expressive DLs such as $\mathcal{SHOQ}$ because it handles the numeric restrictions imposed by nominal and QCRs via a more arithmetically informed approach employing *Integer Linear Programming* (ILP).

In this chapter, we present our CB algorithm for efficiently handling the interaction of *nominals* and *qualified cardinality restrictions*. The algorithm receives a normalized $\mathcal{SHOQ}$ ontology $\mathcal{O}$ and a finite set of queries $\mathcal{Q}$ as input and determines whether $\mathcal{O} \models A(x) \rightarrow B(x)$ holds for each query $A(x) \rightarrow B(x) \in Q$. Since the goal of this algorithm is to *classify the ontology*, $A(x)$ and $B(x)$ are atomic concepts from $\mathcal{O}$. Section 5.1 formally defines the algorithm, introducing all necessary concepts. Section 5.2 captures the inference rules that govern the algorithm's logic and functionality, which constitute the core of the reasoning process. Section 5.3 reviews the notions of soundness and completeness for the proposed algorithm. Section 5.4 clarifies the algorithm's intuition by applying the reasoning process to sample ontologies.

5.1 Definition of the Calculus

Throughout this section, all theorems and definitions are implicitly related to a fixed, arbitrary ontology $\mathcal{O}$. Recall that in Section 4, we introduced $S_{\mathcal{U}}$, $S_{\mathcal{B}}$ and $S_{\mathcal{C}}$ respectively to be the sets of atomic concepts, introduce nominal concepts, atomic roles, and nominals in $\mathcal{O}$. Moreover, we introduced S_O as the set of all fresh unary predicates regarding nominals and S_L as the set of all literals over the signature $S = (S_{\mathcal{P}}, S_{\mathcal{C}})$, where $S_{\mathcal{P}}$ includes two disjoint subsets, unary predicates $S_{\mathcal{U}}$ and binary predicates $S_{\mathcal{B}}$.

Following other CB calculi [10, 15, 16, 33], our calculus represents consequences in *nodes* as *node clauses* in first-order logic. The main difference is that instead of encoding

away number restrictions using the equality predicate, we model them directly by CQs of FOL. Besides, our calculus allows mentioning named individuals only in unary predicates. We use a two-variable fragment of FOL, which allows only variables x and y to occur in node clauses, each carrying a special meaning. Intuitively, each node represents a set of similar elements in a model of the ontology; when variable x corresponds to such an element, the variable y corresponds to its successor if it exists. Representing QCRs by FOL counting quantifiers allows the calculus to handle QCRs directly by deriving their corresponding inequalities and applying linear programming algorithms, which will significantly speed up the reasoning process.

Definition 5.1. Assume we have $C \in S_{\mathcal{U}} \cup S_O$, $R \in S_{\mathcal{B}}$ and $o \in S_{\mathcal{C}}$. A *node term* is either x, y or o. A *node predicate* has the form $C(x), C(o), C(y)$ or $R(x, y)$. A *node quantifier* is either a CQ of the form $\exists_{\bowtie n} y\varphi(y)$, a *universal* quantifier of the form $\forall y\varphi(y)$ or an exact quantifier of the form $\exists_{=1} x\varphi(x)$, where $\varphi(y) = R(x, y) \wedge C(y)$, $\varphi(x) = O(x)$ and $\bowtie \in \{\geq, \leq\}$. A *node atom* is a unary predicate $C \in S_{\mathcal{U}} \cup S_O$.

In accordance with previous works, we use a notion of redundancy elimination to ensure termination and reduce the number of clauses derived by the algorithm [16].

Definition 5.2. A set of clauses $\mathcal{CL}$ contains a clause $K \rightarrow M$ *up to redundancy*, denoted as $K \rightarrow M \in_* \mathcal{CL}$, if there is a clause $K' \rightarrow M' \in \mathcal{CL}$ such that the clause $K' \rightarrow M'$ is a strengthening of the clause $K \rightarrow M$.

Our CB algorithm constructs a weighted digraph, named *Completion graph*, whose vertices are called *nodes*. Each node describes a set of elements in a model of the ontology. A

node literal is a node atom or a node quantifier. A *node clause* contains only node atoms in the body and node literals in the head. Each node v is labelled with a set of clauses $\mathcal{L}(v)$ and is associated with a unary predicate $\mathsf{core}(v)$ that holds for every term in the model described by v. It means that the unary predicate $\mathsf{core}(v)$ is implicitly included in the body of all clauses in $\mathcal{L}(v)$, so each clause $K \to M \in \mathcal{L}(v)$ is interpreted as $\mathsf{core}(v) \wedge K \to M$.

Definition 5.3. A node clause $K \to M$ is called *known* if either $K \equiv \top$, $M \equiv \bot$ or if K contains only ground predicates; otherwise, it is called a *possible clause*.

The node labels $\mathcal{L}(v)$ are used to decide query entailment: for each query $K \to M$ and each node v, such that $\mathsf{core}(v) \in K$ and $K \to L \in^* \mathcal{L}(v)$ holds for each literal $L \in K$, if $K \to M \in \mathcal{L}(v)$ then we have $\mathcal{O} \models K \to M$.

The weighted edges between nodes represent *role successor* relations between the corresponding elements. If an edge $\langle u, v \rangle$ exists between two nodes, then u (v) is a predecessor (successor) of v (u). Each edge $\langle u, v \rangle$ is labelled with a tuple (R, n, ζ), where R is the binary predicate holding between the two nodes, $n \in \mathbb{N}$ is the cardinality of the edge and ζ is a set of unary predicates that hold in node v.

Definition 5.4. A *Completion graph* for $\mathcal{O}$ is a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathsf{core}, \mathcal{L})$ where $\mathcal{V}$ is a finite set of nodes and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is a finite set of edges; function $\mathsf{core}(v)$ assigns a unary predicate $C(x) \in S_{\mathcal{U}} \cup S_O$ to each node $v \in \mathcal{V}$; and function $\mathcal{L}(v)$ assigns a finite set of node clauses to each node v. Every clause $K \to M \in \mathcal{L}(v)$ is interpreted as $\mathsf{core}(v) \wedge K \to M$. The function $\mathcal{L}(\langle u, v \rangle)$ assigns a tuple (R, n, ζ) to each edge $\langle u, v \rangle \in \mathcal{E}$, where $R(x, y) \in S_{\mathcal{B}}$ is a binary predicate, $n \in \mathbb{N}$ is the cardinality of the edge and $\zeta \subseteq S_{\mathcal{U}}$ is a set of unary predicates which hold in the node v.

We also need to introduce an *arithmetic label* $\mathcal{A}(v)$ for each node, which is a tuple consisting of the CQs occurring in $\mathcal{L}(v)$ and the related clauses of the qualifying concepts. The arithmetic label is the input of the *Arithmetic Module (ARM)*. This chapter uses ARM as a black box ILP solver. ARM accepts a set of restrictions as input and returns a solution if the set is feasible. Otherwise, it will return the *conflict sets* ($\mathcal{CS}$) of the input restrictions, which caused infeasibility.

Let $Q = \exists_{\bowtie n} y \ \varphi(x, y)$, where φ has the form $R(x, y) \wedge C(y)$; and $\bowtie \in \{\leq, \geq\}$. We define *set of counting quantifiers* occurring in $\mathcal{L}(v)$ locally for a node $v \in \mathcal{V}$ as:

$$\mathcal{Q}_l(v) = \{ \bigvee_{t=1}^{k} Q_t \mid K \rightarrow M \vee \bigvee_{t=1}^{k} Q_t \in \mathcal{L}(v) \} \tag{5.1}$$

Assuming that $\mathcal{Q}(v)$ contains m disjunctions of the form $\bigvee_{t=1}^{k} Q_t$, there are $n_q = \sum_{d=1}^{m} k_d$ CQs occurring in $\mathcal{Q}(v)$. We also need to define a closure set $clos(v)$ for each node $v \in \mathcal{V}$ as Definition 5.5.

Definition 5.5. Assume that $K \rightarrow M$ is an atomic clause, $u, v \in \mathcal{V}$ and $\langle v, u \rangle \in \mathcal{E}$, we define the closure set $clos(v)$ as the smallest set of unary and binary predicates such that:

$$\exists_{\bowtie n}\, y\; R(x,y) \wedge C(y) \in \mathcal{Q}_l(v) \Rightarrow \begin{array}{l} R(x,y) \in clos(v) \\ C(y) \in clos(v) \end{array} \tag{5.2}$$

$$\begin{array}{l} K \rightarrow M \in \mathcal{O} \\ K \subseteq clos(v) \end{array} \Rightarrow\; M \subseteq clos(v) \tag{5.3}$$

$$\begin{array}{l} K \rightarrow M \in \mathcal{L}(u) \\ \mathsf{core}(u) \cup K \subseteq clos(v) \end{array} \Rightarrow\; M \subseteq clos(v) \tag{5.4}$$

$$\begin{array}{l} R_1(x,y) \rightarrow R_2(x,y) \in \mathcal{O} \\ R_1(x,y) \in clos(v) \end{array} \Rightarrow\; R_2(x,y) \in clos(v) \tag{5.5}$$

Definition 5.6. Let $K \rightarrow M$ be an atomic clause and $u, v \in \mathcal{V}$, where there is an edge $\langle u, v \rangle \in \mathcal{E}$. A *local arithmetic label* $\mathcal{A}_l(v)$ is defined for each node $v \in \mathcal{V}$ as a tuple $\mathcal{A}_l(v) = (\mathcal{Q}_l(v), \mathcal{C}_l(v))$, where $\mathcal{Q}_l(v)$ is a set of disjunction of CQs as defined in (5.1) and $\mathcal{C}_l(v)$ is a set containing the following clauses:

- $K \rightarrow M \in \mathcal{O}$ such that $K \subseteq clos(v)$.
- $\mathsf{core}(u) \wedge K \rightarrow M$ such that $K \rightarrow M \in \mathcal{L}(u)$ and $\mathsf{core}(u) \cup K \subseteq clos(v)$.

Since nominals and role hierarchies are non-local, we define a *global arithmetic label* $\mathcal{A}_g$ to deal with the non-local restrictions they impose. As its name suggests, the satisfiability

of the global arithmetic label should be ensured globally, so it has to be considered in the context of every node.

Definition 5.7. A *global arithmetic label* is a tuple $\mathcal{A}_g = (\mathcal{Q}_g, \mathcal{C}_g)$, where $\mathcal{Q}_g$ is the set of all CQs over unary predicates of size n_o, defined as:

$$\{(\exists_{=1} x\ O(x))_{\sigma_y} \mid \top \to \exists_{=1} x\ O(x) \in \mathcal{O}\} \tag{5.6}$$

$\mathcal{C}_g$ contains the following sets of clauses:

- **Nominal Clause**: The set of clauses of the form $\mathsf{core}(u) \to M$ where $K \to M$ is an atomic clause in $\mathcal{L}(u)$ and $\mathsf{core}(u) \in S_O$.

- **Role Subsumptions**: The set of subsumptions of the form $R_1(x, y) \to R_2(x, y) \in \mathcal{O}$, where $R_1(x, y)$ is a binary predicate.

Accordingly, the *arithmetic label* $\mathcal{A}(v)$ is defined for every node $v \in \mathcal{V}$ as $\mathcal{A}(v) = (\mathcal{Q}(v), \mathcal{C}(v))$ with $\mathcal{Q}(v) = \mathcal{Q}_l(v) \cup \mathcal{Q}_g$ and $\mathcal{C}(v) = \mathcal{C}_l(v) \cup \mathcal{C}_g$. The arithmetic label $\mathcal{A}(v)$ is the input of ARM to determine its satisfiability and return a solution of the conflict sets. This chapter works with ARM as a black box, the details of which are discussed in Chapter 6.

ARM generates an inequality system based on the arithmetic label and determines its satisfiability. If the arithmetic label is satisfiable, ARM returns a solution that satisfies all the constraints in $\mathcal{A}(v)$. The returned arithmetic solution assigns a non-negative integer

n to the role successors and their qualifications (atomic concepts that the role successors belong).

Definition 5.8 (Arithmetic Solution)**.** An *Arithmetic solution* $\xi(v)$ is a set of tuples $\langle \rho, \sigma_\rho \rangle$ produced by ARM corresponding to $\mathcal{A}(v)$. The *partition* ρ is a set of unary $A(x)$ and binary predicates $R(x, y)$ and $\sigma_\rho \in \mathbb{N}, \sigma_\rho \geq 1$ is the cardinality of the partition.

For example if $\sigma_{\{R,C\}} = 1$, this means that the interpretation of partition $\{R, C\}$ must have only one element. If the arithmetic label is unsatisfiable, then ARM returns the smallest set of clauses that makes it unsatisfiable called *Conflict Set* (CS) (Definition 5.9). Note that a conflict set is not unique, and an unsatisfiable arithmetic label might have multiple conflict sets. For example, let $\mathcal{Q}(v) = \{\exists_{\geq 2} y\, \varphi(y), \exists_{\geq 3} y\, \varphi(y), \exists_{\leq 1} y\, \varphi(y)\}$, where $\varphi(y) = R(x, y) \wedge C(y)$. There are two conflict sets for this arithmetic label as $\mathcal{CS}_1 = \{\exists_{\geq 3} y\, \varphi(y), \exists_{\leq 1} y\, \varphi(y)\}$ and $\mathcal{CS}_2 = \{\exists_{\geq 2} y\, \varphi(y), \exists_{\leq 1} y\, \varphi(y)\}$.

Definition 5.9 (Conflict Sets)**.** A *Conflict set* ($\mathcal{CS}$) is returned by ARM if an arithmetic label $\mathcal{A}(v)$ is unsatisfiable. A $\mathcal{CS}$ is a subset of $\mathcal{A}(v)$ constraints that their integration causes unsatisfiability. A conflict set might include disjunct CQs of the form $\bigvee Q \in \mathcal{Q}(v)$ and clauses of the form $K \rightarrow M \in \mathcal{C}(v)$.

Proposition 5.1. The satisfiability problem for a node's arithmetic label $\mathcal{A}(v)$ is NEXP-TIME-complete [47].

We say that a completion graph is sound w.r.t. $\mathcal{O}$ if all the clauses derived by the calculus are logical consequences of $\mathcal{O}$.

Definition 5.10. A completion graph $\mathcal{G}$ is *sound* for $\mathcal{O}$ if the following conditions are both satisfied:

(i) for each node $v \in \mathcal{V}$ and each clause $K \rightarrow M \in \mathcal{L}(v)$ we have:

$\mathcal{O} \models \mathsf{core}\,(v) \wedge K \rightarrow M$

(ii) for each edge $(R, n, \zeta) \in \mathcal{L}(\langle u, v \rangle)$ and $\langle u, v \rangle \in \mathcal{E}$, let $A(x), B(x)$ and $C(x)$ be unary predicates such that $A(x) = \mathsf{core}(u)$, $B(x) = \mathsf{core}(v)$ and $C(x) \in \zeta$, we have :

$$\begin{aligned} O \models A(x) \wedge \exists_{\geq n} y \quad & (R(x,y) \wedge \bigwedge C(y)) \rightarrow \\ & \exists_{\geq n} y \quad (R(x,y) \wedge B(y) \wedge \bigwedge C(y)) \end{aligned} \tag{5.7}$$

Definition 5.11. A literal ordering $\prec$ is a *strict partial order* (i.e., an irreflexive and transitive binary relation) on the set of all literals S_L. A literal $L \in S_L$ is $\prec_{min}$ if there is no literal $L' \in S_L$ such that $L' \prec L$; moreover, a set of literals N is $\prec_{min}$ if all the literals in N are $\prec_{min}$. A literal $L \in S_L$ is $\prec_{max}$ with respect to a set of literals N, written as $L \not\prec N$ if there is no $L' \in S_L$ such that $L \prec L'$.

5.2 Inference Rules

Table 5.1 shows the inference rules of our CB algorithm. As in other CB calculi, a rule is triggered only if the corresponding nodes do not contain the derived clauses up to

redundancy. There is no ordering for applying these rules, and the applicable rules may be applied nondeterministically.

The inference process starts with an initialization step according to the target query. Such that, for every query $q = A(x) \rightarrow C(x) \in \boldsymbol{Q}$, we introduce a node v_A where $\mathsf{core}(v_A) = A(x)$, and add clause $\top \rightarrow A(x)$ to $\mathcal{L}(v_A)$. Moreover, we create one node v_o to represent each nominal $O(x)$ occurring in $\mathcal{O}$. The process continues by applying the applicable rules from Table 5.1. The arithmetic labels are updated synchronously, and whenever there is a change in an arithmetic label, ARM re-evaluates its satisfiability and updates the solution or returns the conflict sets. The Sigma and Strict rules are applicable if an arithmetic solution is returned, while the Bottom rule is applicable whenever a conflict set is found. The algorithm proceeds until no further rules can be applied. A query $q = A(x) \rightarrow C(x)$ is entailed if, after the termination of the algorithm, $\top \rightarrow C(x)$ has been derived in $\mathcal{L}(v_A)$. The rest of this section discusses the functionality of each inference rule and exemplifies some of their practical applications. The next section walks through the step-by-step application of the reasoning algorithm on sample ontologies.

The Subs rule performs resolution between ontology and node clauses. The Join and Fct rules resolve ground atoms in the label of unary predicate and nominal nodes, respectively. The Glob rule is used to reflect the information of nominal nodes in any other nodes, not just a neighbouring one; this is due to the fact that reasoning in the presence of nominals is intrinsically non-local.

The rest of our rules are particularly designed for arithmetic handling of $\mathcal{SHOQ}$ DL. Rule Sigma extends the completion graph according to the ARM solution for the arithmetic label. Rule Strict verifies if a nominal and concept have to accompany one another at all times. Rule Bottom is applicable if the arithmetic label is unsatisfiable and resolves

TABLE 5.1: Inference Rules for Reasoning in $\mathcal{ALCHOQ}$

Subs	**If**	$\bigwedge_{i=1}^{n} A_i(x) \rightarrow M \in \mathcal{O}$
		$K_i \rightarrow M_i \vee A_i(x) \in \mathcal{L}(v)$, with $A_i \not\prec M_i$ for $1 \leqslant i \leqslant n$,
	then	add $\bigwedge_{i=1}^{n} K_i \rightarrow \bigvee_{i=1}^{n} M_i \vee M$ to $\mathcal{L}(v)$
Join	**If**	$K \rightarrow M \vee C(o) \in \mathcal{L}(v)$, with $C(o) \not\prec M$ and $C(o) \wedge K' \rightarrow M' \in \mathcal{L}(v)$
	then	add $K \wedge K' \rightarrow M \vee M'$
Fct	**If**	$K \rightarrow M \vee C(o) \in \mathcal{L}(v_o)$, where $\mathsf{core}(v_o) = O(x)$
	then	add $K \rightarrow M \vee C(x)$ to $\mathcal{L}(v_o)$
Elim	**If**	$K \rightarrow M \in \mathcal{L}(v)$, and $K \rightarrow M \in_* \mathcal{L}(v) \setminus K \rightarrow M$
	then	remove $K \rightarrow M$ from $\mathcal{L}(v)$
Glob	**If**	$\bigwedge_{i=1}^{n} A_i(x) \wedge K \rightarrow M \in \mathcal{L}(v_o)$, where $\mathsf{core}(v_o) = O(x)$,
	then	if $K' \rightarrow M' \vee O(x) \in \mathcal{L}(v)$, with $O(x) \not\prec M'$,
		and $K_i \rightarrow M_i \vee A_i(x) \in \mathcal{L}(v)$, with $A_i(x) \not\prec M_i$ for $1 \leqslant i \leqslant n$,
		add $\bigwedge_{i=1}^{n} K_i \wedge K' \wedge K\sigma_o \rightarrow \bigvee_{i=1}^{n} M_i \vee M \vee M'$ to $\mathcal{L}(v)$
		otherwise, add $\bigwedge_{i=1}^{n} A_i(o) \wedge K\sigma_o \rightarrow M\sigma_o$ to $\mathcal{L}(v)$.
Sigma	**If**	$\langle \rho, n \rangle \in \xi(u)$ such that $R \in \rho$, $\zeta = \{C(x) \mid C(x) \in \rho \cap S_{\mathcal{U}}\}$
		and no edge $\langle u, v \rangle \in \mathcal{E}$ exists such that $\mathcal{L}(\langle u, v \rangle) = (R, n, \zeta)$
	then	let $B(x) = \mathsf{select}\ (\rho)$, if no node $v \in \mathcal{V}$ which $\mathsf{core}(v) = B(x)$,
		then create v and let $\mathsf{core}(v) = B(x)$, and $\mathcal{L}(v) = \{\top \rightarrow B(x)\}$;
		if $\langle u, v \rangle$ not exist, create $\langle u, v \rangle$, set the label $\mathcal{L}(\langle u, v \rangle)$ to (R, n, ζ)
		and $C(x) \rightarrow C(x)$ to $\mathcal{L}(v)$ for each $C(x) \in \zeta \backslash B(x)$,
Strict	**If**	$\langle \rho, n \rangle \in \xi(u)$ such that for $O(x) \in \rho$ and $D(x) \in \rho$
		and $\mathcal{CS}(\mathcal{A}(v) \cup \{O(x) \wedge D(x) \rightarrow \bot\}) = \{O(x) \wedge D(x) \rightarrow \bot\}$
	then	add $\top \rightarrow D(o)$ to $\mathcal{L}(v)$.
Bottom	**If**	$Q_i \in \mathcal{CS}(v)$ for $0 \leqslant i \leqslant m$, where Q_i is a disjunction of CQs,
		and $K_i \rightarrow M_i \vee Q_i \in \mathcal{L}(v)$, and $K_j \rightarrow M_j \in \mathcal{L}(v_o)$ for $0 \leqslant j \leqslant n$
		and $O(x) \rightarrow M_j \in \mathcal{CS}(v)$, where M_j contains only unary atoms,
	then	add $\bigwedge_{i=0}^{m} K_i \wedge \bigwedge_{j=0}^{n} K_j\sigma_o \rightarrow \bigvee_{i=0}^{m} M_i$ to $\mathcal{L}(v)$.
Reach	**If**	$\bigwedge_{i=1}^{n} A_i \wedge \bigwedge_{i=1}^{m} B_i \rightarrow M \in \mathcal{L}(v)$, $\mathsf{core}(v) = A_0$
		where each B_i is ground, and each A_i is non-ground,
		and M contains only ground unary atoms,
		and $\langle u, v \rangle \in \mathcal{E}$, with $(R, m, \zeta) \in \mathcal{L}(\langle u, v \rangle)$, where $A_i \in \zeta$
		and $K_i \rightarrow M_i \vee \exists_{\bowtie n_i} y\ (R(x, y) \wedge A_i(y)) \in \mathcal{L}(u)$ for $0 \leqslant i \leqslant n$
	then	add $\bigwedge_{i=0}^{n} K_i \wedge \bigwedge_{i=0}^{m} B_i \rightarrow \bigvee_{i=0}^{m} M_i \vee M$ to $\mathcal{L}(u)$
Nom	**If**	$K \rightarrow M \vee O(x) \in \mathcal{L}(v)$ with $O(x) \not\prec M$,
		and $K' \rightarrow M' \vee C(x) \in \mathcal{L}(v)$ with with $C(x) \not\prec M'$,
	then	add $K' \wedge K \rightarrow M' \vee M \vee C(o)$ to $\mathcal{L}(v)$
		and $C(x) \rightarrow C(x)$ to $\mathcal{L}(v_o)$

the node clauses based on the conflict set of the arithmetic label. Rule Reach propagates information from a node to its predecessor, and rule Nom adds possible clauses to the label of nominal nodes in order to explore the consequences of a nominal being an instance of an atomic concept or two nominals being merged together.

5.2.1 Subs Rule

This rule is a more complete version of the subsumption rule in traditional CB algorithms. This rule is designed for unfolding and resolving purposes.

In the general form, this rule is used for combining a set of axioms as follows:

$$\left.\begin{array}{l} A_1(x) \wedge A_2(x) \wedge A_3(x) \rightarrow M \in \mathcal{O} \\ K_1 \rightarrow M_1 \vee A_1(x) \in \mathcal{L}(v) \\ K_2 \rightarrow M_2 \vee A_2(x) \in \mathcal{L}(v) \\ K_3 \rightarrow M_3 \vee A_3 \in \mathcal{L}(v) \end{array}\right\} \Rightarrow K_1 \wedge K_2 \wedge K_3 \rightarrow M_1 \vee M_2 \vee M_3 \vee M$$

Accordingly, QCRs are explicitly expressed on the right-hand side of axioms so that they can be treated appropriately and their potential consequences can be verified by ARM. Additionally, new subsumptions will be extracted based on the transitive property of subsumption. For example, if $A(x) \rightarrow B(x)$ and $B(x) \rightarrow C(x)$, then applying this rule can conclude that $A(x) \rightarrow C(x)$.

The Subs rule can be applied to resolve disjoint literals in the clause's right-hand side, as shown in the following example:

$$\left.\begin{array}{l} B(x) \to C(x) \in \mathcal{L}(v) \\ B(x) \to D(x) \vee F(x) \in \mathcal{L}(v) \\ F(x) \wedge C(x) \to \bot \in \mathcal{O} \end{array}\right\} \Rightarrow B(x) \to D(x) \in \mathcal{L}(v)$$

The Subs rule can also extend a concept on the right-hand side of a clause, to ensure that all QCRs on the right-hand side of axioms are unfolded. The following example shows how this rule can be used for unfolding.

$$\left.\begin{array}{l} A(x) \to B(x) \vee C(x) \in \mathcal{L}(v) \\ C(x) \to D(x) \vee F(x) \in \mathcal{O} \end{array}\right\} \Rightarrow A(x) \to B(x) \vee D(x) \vee F(x) \in \mathcal{L}(v)$$

This rule is also applicable to resolve a disjunction's literals with a common super-concept, as shown in the following example.

$$\left.\begin{array}{l} C(x) \to A(x) \vee B(x) \in \mathcal{L}(v) \\ A(x) \to D(x) \in \mathcal{O} \\ B(x) \to D(x) \in \mathcal{O} \end{array}\right\} \Rightarrow C(x) \to D(x) \in \mathcal{L}(v)$$

The formal definition of this rule is presented in Table 5.1.

5.2.2 Join Rule and Fct Rule

The Join and Fct rules resolve ground atoms in the label of regular and nominal nodes, respectively.

The ground atoms $C(o)$ are interpreted as $O(x) \to C(x)$ accordingly, a clause $A(x) \wedge C(o) \to B(x)$ is a conditional subsumption clause which states that if $o : C$ then $A(x) \to$

$B(x)$. If the condition appears on different sides of node clauses, then the *Join* rule can be used to resolve the ground atom and infer an unconditional (a.k.a general) axiom.

The Join rule is designed to resolve ground atoms from different sides of the node clauses as follows:

$$\left.\begin{array}{l} K \to M \vee C(o) \in \mathcal{L}(v) \\ C(o) \wedge K' \to M' \in \mathcal{O} \in \mathcal{L}(v) \end{array}\right\} \Rightarrow K \wedge K' \to M \vee M' \in \mathcal{L}(v)$$

Accordingly, new general subsumptions could be inferred within the label of a node by resolving the ground atoms. For example, based on the clauses $A(x) \to C(o)$ and $C(o) \to B(x)$, the join rule derives $A(x) \to B(x)$. Similarly, $A(o) \to \bot$ and $B(x) \to C(x) \vee A(o)$ is resolved to $B(x) \to C(x)$.

The Fct rule is only applicable in nominal nodes to implement the fact that nominal are singleton concepts. In a nominal node, v_o with $\mathsf{core}(v_o) = O(x)$ every clause $K \to M \in \mathcal{L}(v_o)$ is interpreted as $O(x) \wedge K \to M$ so we have $x = o$ in the context of v_o. The Fct rule is required to derive nominal subsumption axioms such as $O(x) \to C(x)$ as shown in the following example:

$$\top \to C(o) \in \mathcal{L}(v_o) \Rightarrow \top \to C(x) \in \mathcal{L}(v_o)$$

5.2.3 Elim Rule

The Elim rule removes the clauses from the node's label for which there is a stronger clause in the label to eliminate redundancy and update the arithmetic label. For example, if $\mathcal{L}(v_A)$ contains both node clauses $B(x) \to C(x)$ and $B(x) \to C(x) \vee \exists\, y\, (R(x,y) \wedge A(x))$

then the Elim rule will eliminate the latter as the former is stronger and having both is redundant. As a result, the CQ $\exists\, y\,(R(x,y) \wedge A(x))$ will also be removed from the arithmetic label $\mathcal{A}(v_A)$.

5.2.4 Glob Rule

The Glob rule reflects the information of nominal nodes in any other nodes in the completion graph, not just a neighbouring one. The Glob rule applies the logic of reasoning in the presence of nominal, which is intrinsically *non-local (global)*.

The Glob is also founded on the fact that all the clauses in the label of a nominal node are interpreted in the context of the core nominal. Therefore, the clause $A(x) \rightarrow C(x) \in \mathcal{L}(v_o)$ reflects as $A(o) \rightarrow C(o)$ in the label of any other node. This inference is translated as if $A(x) \rightarrow C(x)$ and $O(x) \rightarrow A(x)$ then $O(x) \rightarrow C(x)$, which is applied by the second branch of Glob rule.

The first branch applies the same notion to the nodes where $O(x)$ is already entailed and could directly participate in the resolution. For example Glob rule drives $K_1 \wedge K_2 \wedge K' \wedge C(o) \rightarrow D(x) \vee M_1 \vee M_2 \vee M' \in \mathcal{L}(v)$ based on the following clauses:

$$\left.\begin{array}{l} A_1(x) \wedge A_2(x) \wedge C(x) \rightarrow D(x) \in \mathcal{L}(v_o) \\ K' \rightarrow M' \vee O(x) \in \mathcal{L}(v) \\ K_1 \rightarrow M_1 \vee A_1 \in \mathcal{L}(v) \\ K_2 \rightarrow M_2 \vee A_2 \in \mathcal{L}(v) \end{array}\right\}$$

5.2.5 Sigma Rule

The Sigma rule is applied on the *arithmetic solution* returned by ARM corresponding to the *arithmetic label*. Sigma rule reflects the solution in the completion graph. An arithmetic solution $\xi(v)$ is a set of tuples $\langle \rho, \sigma_\rho \rangle$, where *partition* ρ is a set of unary $A(x)$ and binary predicates $R(x, y)$ and $\sigma_\rho \in \mathbb{N}, \sigma_\rho \geq 1$ is the cardinality of the partition (Definition 5.8).

Definition 5.12. The select function chooses one unary predicate from the partition ρ as its representative. The select function chooses the $\prec_{Max}$ concept w.r.t. the set of unary predicates $\zeta \subseteq \rho$. The method select is called a function because it returns a unique representative for all partitions containing the same set of concepts. In other words, if there are two partitions rho_1 and rho_2 containing the same unary predicates, then $\mathsf{select}(\rho_1) \equiv \mathsf{select}(\rho_2)$. The Sigma rule uses the select function to choose a unary predicate as the representative of a partition.

If there exists no node v in the graph such that $core(v) \equiv \mathsf{select}(\rho)$, then Sigma rule creates a new node; otherwise, it adds the rest of unary predicates $C(x) \in \zeta \backslash \mathsf{select}(\rho)$ to the label of v as possibilities. The Sigma rule also adds the clause $C(x) \rightarrow C(x)$ to $\mathcal{L}(v)$, to consider the consequences of having $C(x)$ for all domain elements corresponding to a node v. The formal definition of this rule is presented in Table 5.1. The following example demonstrates a sample application of the Sigma rule.

Example 5.1 (Sigma Rule Application)**.** Assume that ARM returns the solution $\xi(v_A) = \{\langle \{B(x), C(x), R(x, y)\}, 2\rangle, \langle \{C(x), D(x), R(x, y)\}, 2\rangle\}$ based on the arithmetic label $\mathcal{A}(v_A)$. So the Sigma rule is applicable to the completion graph $\mathcal{G}$, which has only one node v_A.

The select function chooses the unary predicate $B(x)$ as a representative for the first partition. So the rule creates a new node v_B, with $\mathsf{core}(v_B) = B(x)$ and adds the initialization clause $\top \rightarrow B(x)$ to its label $\mathcal{L}(v_B)$. Also creates the edge $\langle v_A, v_B \rangle$ and adds $(R, 2, \{B, C\})$ to $\mathcal{L}\langle v_A, v_B \rangle$. Furthermore, it adds the clause $C(x) \rightarrow C(x)$ to $\mathcal{L}(v_B)$ to check the consequences of holding $B(x)$ and $C(x)$ together.

Another node v_C with $\mathsf{core}(v_C) = C(x)$ is created corresponding to the second partition. Moreover, the rule adds the clauses $\top \rightarrow C(x)$ and $D(x) \rightarrow D(x)$ to $\mathcal{L}(v_C)$ and generates $(R, 2, \{C, D\})$ in $\mathcal{L}\langle v_A, v_C \rangle$. Figure 5.1 shows the completion graph after applying the Sigma rule.

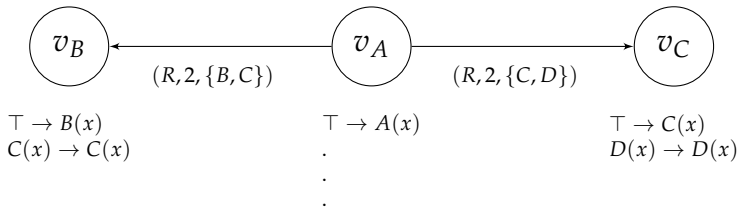


FIGURE 5.1: Sigma Rule Example Application

Intuitively, this rule helps the algorithm to determine which concepts may hold together and need to be considered in one node. Considering a concept $A(x)$ in the node v_B, which is presented by the clause $A(x) \rightarrow A(x)$, implies that there might be at least one individual a in our model, which is $a : A$ and $a : B$. Therefore if two concepts exist in one partition based on an arithmetic solution returned by ARM, our algorithm checks the conjunction of these two concepts and discovers any potential conflicts.

5.2.6 Strict Rule

As we know, the cardinality of each nominal is exactly one, so their inclusion in the same partition could derive more consequences. For example, considering the axiom $A \rightarrow \exists R(x,y) \wedge B(y) \wedge O(y)$, mere satisfiability of $A(x)$ derives $O(x) \rightarrow B(x)$. However, having a concept $B(x)$ and a nominal $O(x)$ in the same partition is not sufficient to conclude the subsumption $O(x) \rightarrow B(x)$. There might exist other solutions for the arithmetic label in which the nominal $O(x)$ and the concept $B(x)$ are not present in the same partition. The example 5.2 specifically illustrates this scenario.

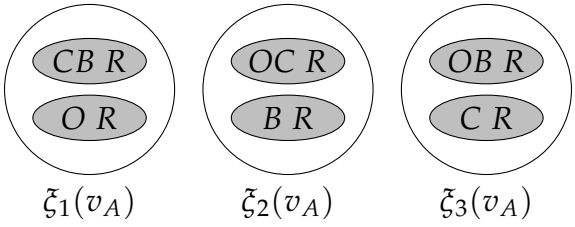
Example 5.2 (Strict Rule Application)**.** Let $O(x)$ be a nominal and $A(x), B(x)$ and $C(x)$ be atomic concepts. The ontology $\mathcal{O}$ contains the following axioms:

$$A(x) \rightarrow \exists y\, R(x,y) \wedge O(x) \tag{5.8}$$

$$A(x) \rightarrow \exists y\, R(x,y) \wedge B(y) \tag{5.9}$$

$$A(x) \rightarrow \exists y\, R(x,y) \wedge C(y) \tag{5.10}$$

$$A(x) \rightarrow \exists_{\leq 2} y\, R(x,y) \wedge (B(x) \vee C(x) \vee O(x)) \tag{5.11}$$



There are three possible models for satisfying these restrictions shown as $\xi_1(v_A), \xi_2(v_A)$ and $\xi_3(v_A)$. Only in one solution does the nominal $O(x)$ exist in a partition with either concept $A(x)$ or $B(x)$. Therefore, the Strict rule is not applicable to this example.

In this case, the subsumption $O(x) \rightarrow B(x)$ is a possibility that is handled using the Sigma rule. Consequently, we need to determine if the concept $A(x)$ and the nominal $O(x)$ are required to belong to the same partition within all possible arithmetic solutions. Instead of finding and investigating all possible solutions, we check if a solution could exist in which the concept $B(x)$ and the nominal $O(x)$ do not occur in the same partition. Accordingly, we check whether, assuming that $O(x)$ and $B(x)$ are disjoint, causes a *strict* unsatisfiability. If so, then the subsumption $O(x) \rightarrow B(x)$ is concluded by the Strict rule. An unsatisfiability is *strict* if it is caused only by known information and there is no possible clause in the conflict set.

This rule is also responsible for situations where two nominals must appear in a partition for all possible solutions. If that is the case, these two nominals would be equal to each other, and reflexive subsumption in their labels would reflect this fact.

5.2.7 Bottom Rule

Intuitively, the Bottom rule propagates obtained information from a successor to its predecessor. If ARM determines that the arithmetic label is unsatisfiable, then there's no solution that satisfies all the constraints imposed by nominals and QCRs. In this case, ARM returns the conflict sets as the unsatisfiability root.

There might be multiple conflict sets of the same size for an arithmetic label. The consequences of resolving each conflict set have to be derived in the construction graph.

Accordingly, the Bottom rule is applicable to each conflict set $\mathcal{CS}$. For example, assume $\mathcal{Q}_l(v) = \exists_{\geq 3}\, y\, \varphi(y), \exists_{\geq 3}\, y\, \varphi(y)$ and $\exists_{\leq 1}\, y\, \varphi(y)$, where $\varphi(y) = R(x,y) \wedge C(y)$. Obviously, the conjunction of these QCRs causes a conflict, so the arithmetic label is not feasible. One can recognize two sets of conflict sets (two different sets with the same number of literals) involved in the clash, $\mathcal{CS}_1 = \{\exists_{\geq 2}\, y\, \varphi(y), \exists_{\leq 1}\, y\, \varphi(y)\}$ and $\mathcal{CS}_2 = \{\exists_{\geq 3}\, y\, \varphi(y), \exists_{\leq 1}\, y\, \varphi(y)\}$.

The Bottom rule only impacts the possible clauses in a conflict set, such that if all the conflict sets are known, then there is an unsatisfiability in the ontology. For example let $\mathcal{Q}(v_B) = \{\exists_{\geq 2}\, y\, R(x,y) \wedge C(y), \exists_{\leq 1}\, y\, R(x,y) \wedge D(y)\}$ and $\mathcal{C}(v_B) = \{C(x) \rightarrow D(x)\}$. This arithmetic label is evidently unsatisfiable, with the conflict set $\mathcal{CS} = \{\{\exists_{\geq 2}\, y\, R(x,y) \wedge C(y), \exists_{\leq 1}\, y\, R(x,y), C(x) \rightarrow D(x)\}$. Since all the clauses in the conflict set are known information, the Bottom rule will resolve these clauses by adding $\top \rightarrow \bot \in \mathcal{L}(v_B)$. This clause shows that the node v_B and consequently the concept $B(x)$ are unsatisfiable.

5.2.8 Reach Rule

A mentioned before, in the presence of nominals, mere satisfiability (non-emptiness) of concepts can lead to new entailments, e.g., asserting that a particular concept has at least one instance may lead to a new subsumption between atomic concepts.

Reach rule implements the non-emptiness tracking adapted from [35] by propagating clauses containing ground atoms from a node to its predecessors. This rule follows the successor relation by resolving the underlying CQs.

Example 5.3 (Reach Rule Application)**.** Assume a completion graph $\mathcal{G}$ as shown in Figure 5.2. Having $\top \rightarrow C(o) \in \mathcal{L}(v_B)$ interprets as *iff* $B(x)$ is satisfiable ($B^{\mathcal{I}} \neq \emptyset$) then $O(x) \rightarrow C(x)$. Meanwhile, non-emptiness of $A(x)$ and $A(x) \rightarrow \exists\, y\, R(x,y) \wedge B(y)$ implies

non-emptiness of $B(x)$, in other words $B(x)$ is reachable from $A(x)$. Accordingly non-emptiness of $A(x)$ also entails $O(x) \rightarrow C(x)$. Such that the Reach rules add $\top \rightarrow C(o) \in \mathcal{L}(v_A)$.

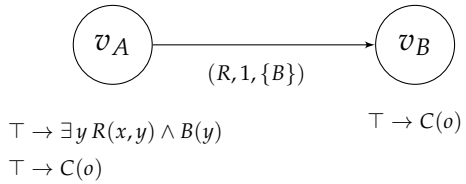


FIGURE 5.2: Reach Rule Example Application

The original form of this rule is generalized to handle more complex cases where the condition clause in the successor's label ($\top \rightarrow C(o)$ in Figure 5.2) has ground/non-ground atoms in its head (e.g. $D(x) \wedge F(o) \rightarrow C(o)$).

5.2.9 Nom Rule

Having a node clause of the form $K \rightarrow M \wedge O(x) \in \mathcal{L}(v)$ could infer the possibility of $x = o$ in the context of node v. The Nom rule reflects this possible equality within other unary predicates which exist in node clauses of $\mathcal{L}(v)$. The following example represents a simple application of this rule:

$$\left.\begin{array}{l} \top \rightarrow C(x) \in \mathcal{L}(v_A) \\ \top \rightarrow O(x) \in \mathcal{L}(v_A) \end{array}\right\} \Rightarrow \top \rightarrow C(o) \in \mathcal{L}(v_A)$$

The clauses in $\mathcal{L}(v_A)$ are interpreted in regard to $A(x)$, such that having $A(x) \rightarrow C(x)$ and $A(x) \rightarrow O(x)$ entail $A(x) \rightarrow C(o)$, which is interpreted as *iff* $A(x)$ is satisfiable ($A^{\mathcal{I}} \neq \emptyset$) *then* $O(x) \rightarrow C(x)$.

Rule Nom also adds possible clauses to the label of nominal nodes in order to explore the consequences of a nominal being an instance of an atomic concept or two nominals being merged together. In the above example, the Nom rule also adds $C(x) \rightarrow C(x) \in \mathcal{L}(v_o)$, which allows tracing the consequences of the possible subsumption $O(x) \rightarrow C(x)$. This possibility is added to the arithmetic label of the relayed nodes as discussed in Definition 5.7 to discover any potential conflicts and derived entailments.

5.3 Soundness and Completeness

Our arithmetic CB algorithm is *sound* and *complete* based on the following theorems, which are formally proved in Chapter 7. Let $\mathcal{G}$ be a sound completion graph w.r.t. $\mathcal{O}$. The completion graph obtained by applying the inference rules from Table 5.1 to $\mathcal{G}$ is sound for $\mathcal{O}$.

Theorem 5.1 (Soundness)**.** Let $\mathcal{G}_1 = (\mathcal{V}, \mathcal{E}, \mathsf{core}, \mathcal{L})$ be a completions graph and let $\mathcal{L}_1$ be the clause system defined for $\mathcal{G}_1$. Additionally, let $\mathcal{G}_1$ and $\mathcal{L}_2$ be the completion graph and the clause system, respectively, obtained by applying an inference rule from Table 5.1 to $\mathcal{G}_1$ and $\mathcal{L}_1$. If $\mathcal{G}_1$ and $\mathcal{L}_1$ are both sound for $\mathcal{O}$, then both $\mathcal{G}_2$ and $\mathcal{L}_2$ are sound for $\mathcal{O}$.

Theorem 5.2 (Completeness)**.** Let $\mathcal{G}$ be a sound completion graph such that no inference rule from Table 5.1 is applicable to it. Then $\top \rightarrow C(x) \in \mathcal{L}(v_A)$ holds for each query $q = A(x) \rightarrow C(x)$ and node $v_A \in \mathcal{V}$ with $\mathsf{core}(v_A) = A(x)$ if $\mathcal{O} \models A(x) \rightarrow C(x)$.

5.4 Examples

This section illustrates the application of the proposed CB calculus with examples. Each example applies our CB algorithm to a normalized $\mathcal{ALCHOQ}$ ontology $\mathcal{O}$ and a finite set of queries $\mathcal{Q}$ to determine whether $\mathcal{O} \models q$ for each query $q = K \rightarrow M \in \mathcal{Q}$. Since the goal of the algorithm is to obtain *ontology classification*, we only consider queries of the form $A(x) \rightarrow B(x)$ where $A(x)$ and $B(x)$ are unary predicates from $\mathcal{O}$. The examples demonstrate the complete reasoning process from the initialization to the query entailments employing the applicable inference rules from Table 5.1. In the following examples, each node is illustrated as a circle with its core inside the circle, and the clauses in $\mathcal{L}(v)$ above/below the circle.

Example 5.4 (Reasoning Process O_1)**.** Let $\mathcal{O}_1$ be an ontology containing axioms (5.12) – (5.22), and $q = A(x) \rightarrow F(x)$. Figure 5.3 summarizes the inferences relevant to deriving $\mathcal{O}_1 \models q$ using our CB algorithm.

The completion graph is initialized by creating a node v_A with core $A(x)$ corresponding to the query clause and one node $v_{O_i}, 1 \leq i \leq 4$ for each nominal occurring in $\mathcal{O}_1$. The created nodes are initialized by clauses (5.27), (5.38), (5.40), (5.42) and (5.44) and Subs rule adds clauses (5.28) – (5.31). Accordingly, $\mathcal{A}(v_A)$ is updated as $\mathcal{Q}_l(v_A) = \{\exists_{\geq 2}\ y\ (R(x,y) \wedge B(y)), \exists\ y\ (R(x,y) \wedge D(y)), \forall\, y\, (R(x,y) \wedge C'(y)), \forall\, y\, (R(x,y) \wedge F'(y))\}$ while $\mathcal{C}_l(v_A) = \{B(x) \rightarrow O_2(x) \vee O_3(x), C(x) \wedge C'(x) \rightarrow \bot, F(x) \wedge F'(x) \rightarrow \bot\}$ and $\mathcal{Q}_g = \{\exists_{=1} y(O_i) \mid 2 \leq i \leq 3\}$. The *arithmetic module* calculates a solution that satisfies $\mathcal{A}(v_A)$ as $\xi(v_A) = \{(\langle R, B, O_2, C', F'\rangle, 1), (\langle R, B, O_3, C', F'\rangle, 1), (\langle R, D\rangle, 1)\}$.

TABLE 5.2: Clauses in Ontology $\mathcal{O}_1$, Example 5.4

$A \sqsubseteq \geq 2\ R.B$	$\rightsquigarrow$	$A(x) \rightarrow \exists_{\geq 2}\ y\ (R(x,y) \wedge B(y))$	(5.12)
$A \sqsubseteq \exists R.D$	$\rightsquigarrow$	$A(x) \rightarrow \exists\ y\ (R(x,y) \wedge D(y))$	(5.13)
$D \sqsubseteq \exists R.o_1$	$\rightsquigarrow$	$D(x) \rightarrow \exists\ y\ R(x,y) \wedge O_1(y)$	(5.14)
$o_4 \sqsubseteq \geq 2\ R.C$	$\rightsquigarrow$	$O_4(x) \rightarrow \exists_{\geq 2}\ y\ (R(x,y) \wedge C(y))$	(5.15)
$B \sqsubseteq o_2 \sqcup o_3$	$\rightsquigarrow$	$B(x) \rightarrow O_2(x) \vee O_3(x)$	(5.16)
$C \sqsubseteq o_1 \sqcup o_2 \sqcup o_3$	$\rightsquigarrow$	$C(x) \rightarrow O_1(x) \vee O_2(x) \vee O_3(x)$	(5.17)
$\top \sqsubseteq \forall R.C' \sqcup F$	$\rightsquigarrow$	$\top \rightarrow \forall y\ (R(x,y) \wedge C'(y)) \vee F(x)$	(5.18)
$\top \sqsubseteq \forall R.F' \sqcup F$	$\rightsquigarrow$	$\top \rightarrow \forall y\ (R(x,y) \wedge F'(y)) \vee F(x)$	(5.19)
$C \sqcap C' \sqsubseteq \bot$	$\rightsquigarrow$	$C(x) \wedge C'(x) \rightarrow \bot$	(5.20)
$F \sqcap F' \sqsubseteq \bot$	$\rightsquigarrow$	$F(x) \wedge F'(x) \rightarrow \bot$	(5.21)
$\top \rightarrow \exists_{=1}\ x\ O_i(x)$		$1 \leq i \leq 4$	(5.22)

Applying the Sigma rule, create nodes v_B and v_D initialized with (5.36) and (5.23); and connects v_A with v_B and v_D through edges (5.51) and (5.53).

Preceding analogously in node $\xi(v_{O_4})$, the Bottom rule is applicable on the conflict set $\mathcal{CS}(v_{o_4}) = \{\forall\, y\, (R(x,y) \wedge C'(y))\}$ which produces (5.46) and Elim rule removes $\top \rightarrow \forall\, y\, (R(x,y) \wedge C'(y)) \vee F(x)$ from $\mathcal{L}(v_{o_4})$. The Sigma rule then creates a new node with core $C(x)$, adds clause (5.48) to $\mathcal{L}(v_C)$ and connects v_{O_4} and v_C through an edge labeled with (5.55) and (5.56). After derivation of (5.49) by Subs rule, the Nom rule becomes applicable. Triple applications of this rule yields clause (5.50); and also clauses (5.39), (5.41) and (5.43). Accordingly, $O_1(x) \rightarrow C(x)$, $O_2(x) \rightarrow C(x)$ and $O_3(x) \rightarrow C(x)$ are added to $\mathcal{C}_g$ which makes the arithmetic label $\mathcal{A}(v_D)$ infeasible so ARM returns $CS(v_D) = \{O_1(x) \rightarrow$

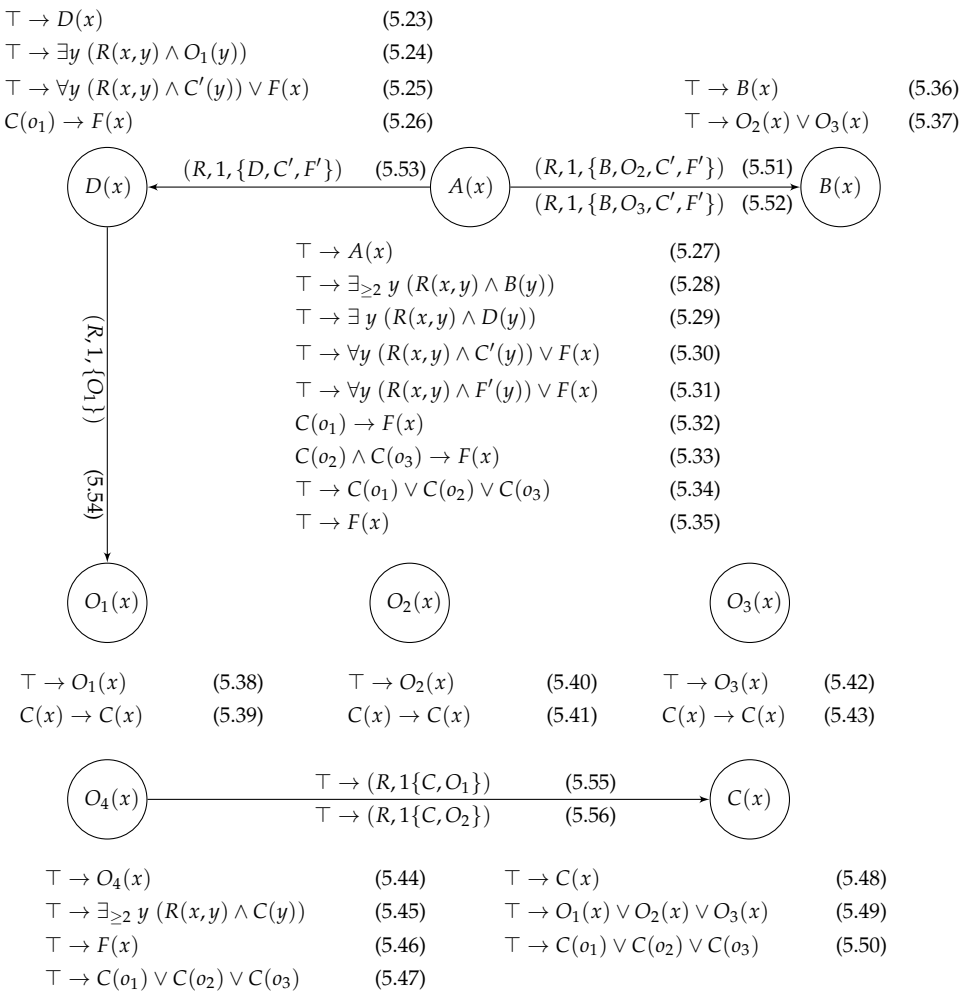


FIGURE 5.3: Algorithm Execution for Example 5.4

$C(x), \forall y\,(R(x,y) \wedge C'(y))\}$ based on which the Bottom rule derives (5.26). Similarly, applying the Bottom rule to the $CS(v_A)$ derives (5.33).

On the other hand, clause (5.50) can be back-propagated by Reach rule to produce clause (5.47) in v_{O_4}. While Reach rule infers (5.32) by propagating (5.26). Consecutively, Glob rule reflects this clause to deduce (5.34). Triple applications of Join rule results in

TABLE 5.3: Clauses in Ontology $\mathcal{O}_2$, Example 5.5

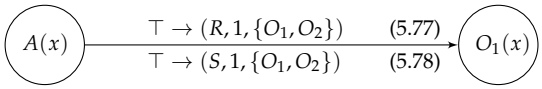
$A \sqsubseteq \exists R.o_1$	$\rightsquigarrow$	$A(x) \rightarrow \exists\, y\ (R(x,y) \wedge O_1(y))$	(5.57)
$A \sqsubseteq \exists R.o_2$	$\rightsquigarrow$	$A(x) \rightarrow \exists\, y\ (R(x,y) \wedge O_2(y))$	(5.58)
$A \sqsubseteq \exists R.o_3 \sqcup D$	$\rightsquigarrow$	$A(x) \rightarrow \exists\, y\ (R(x,y) \wedge O_3(y)) \vee D(x)$	(5.59)
$A \sqsubseteq \leq 1\ S.\top$	$\rightsquigarrow$	$A \rightarrow \exists_{\leq 1}\, y\ R(x,y)$	(5.60)
$O_1 \sqsubseteq \geq 2S.B$	$\rightsquigarrow$	$O_1(x) \rightarrow \exists_{\geq 2}\, y\ (R(x,y) \wedge B(y))$	(5.61)
$B \sqsubseteq o_1 \sqcup o_2 \sqcup o_3$	$\rightsquigarrow$	$B(x) \rightarrow O_1(x) \vee O_2(x) \vee O_3(x)$	(5.62)
$R \sqsubseteq S$	$\rightsquigarrow$	$R(x,y) \rightarrow S(x,y)$	(5.63)
$\top \rightarrow \exists_{=1}\, x\ O_i(x)$		$1 \leq i \leq 3$	(5.64)

(5.35), which proves our query clause.

Example 5.5 (Reasoning Process O_2)**.** Let $\mathcal{O}_2$ be an ontology containing axioms (5.57) – (5.64), and $q = A(x) \rightarrow D(x)$. Our CB algorithm verifies whether $\mathcal{O}_2 \models q$. Figure 5.4 summarizes the inferences relevant to deriving the query q.

The completion graph is initialized by creating a node v_A with core $A(x)$ and one node $v_{O_i}, 1 \leq i \leq 3$ for each nominal occurring in $\mathcal{O}_2$[1]. The created nodes are initialized with the clause (5.65) and (5.71). The Subs rule then adds clauses (5.66) – (5.69). ARM calculates the arithmetic solution as $\xi(v_A) = \{(\langle R, O_1, O_2, O_3\rangle, 1)\}$, based on which the Sigma rule adds (5.72) and (5.73). Adding (5.74) by the Subs rule makes the arithmetic label $\mathcal{A}(v_{o_1})$ infeasible. ARM calculates $\mathcal{CS}(v_{o_1}) = \{O_1 \rightarrow O_2, O_1 \rightarrow O_3\}$, which infers (5.75) by applying the Bottom rule. The Fact rule resolves (5.75) to (5.76), which updates

[1]For simplicity only node O_1 is illustrated in Figure 5.4 as it is required for deriving the query clause.



$\top \to A(x)$		(5.65)	$\top \to O_1(x)$	(5.71)
$\top \to \exists y\,(R(x,y) \wedge O_1(y))$		(5.66)	$O_2(x) \to O_2(x)$	(5.72)
$\top \to \exists y\,(R(x,y) \wedge O_2(y))$		(5.67)	$O_3(x) \to O_3(x)$	(5.73)
$\top \to \exists y\,(R(x,y) \wedge O_3(y)) \vee D(x)$		(5.68)	$\top \to \exists_{\geq 2} y\,(R(x,y) \wedge B(y))$	(5.74)
$\top \to \exists_{\leq 1} y\,R(x,y)$		(5.69)	$O_2(o_1) \wedge O_3(o_1) \to \bot$	(5.75)
$\top \to D(x)$		(5.70)	$O_2(x) \wedge O_3(x) \to \bot$	(5.76)

FIGURE 5.4: Algorithm Execution for Example 5.5

the global arithmetic label and makes $\mathcal{A}(v_A)$ infeasible. Bottom rule derives (5.70) based on the conflict set as $\mathcal{CS}(v_A) = \{\exists y\,(R(x,y) \wedge O3(y))\}$ returned by ARM, which proves the entailment of our query clause.

Example 5.6 (Reasoning Process O_3)**.** Let $\mathcal{O}_3$ be an ontology containing axioms (5.79) – (5.87), and $q = A(x) \to H(x)$. Our CB algorithm verifies whether $\mathcal{O}_3 \models q$. Figure 5.5 summarizes the relevant inferences for deriving the query q.

The completion graph is initialized by creating a node v_A with core $A(x)$ corresponding to the query clause and a node v_O for the nominal O occurring in $\mathcal{O}_3$. The created nodes are initialized by clauses (5.88) and (5.99), respectively. Afterwards, clauses (5.89) – (5.91) are obtained by applying Subs rule. Accordingly, the arithmetic label $\mathcal{A}(v_A)$ is updated as $\mathcal{Q}_l(v_A) = \{\exists_{\geq 2}\, y\,(S(x,y) \wedge C(y)), \exists_{\leq 1}\, y\,(S(x,y) \wedge D(y)), \exists\, y\,(R(x,y) \wedge F(y))\}$ while $\mathcal{C}_g(v_A) = \{F(x) \to O(x), F(x) \to B(x)\}$ and $\mathcal{Q}_g = \{\exists_{=1} y\, O(y)\}$.

ARM then calculates the arithmetic solution as $\xi(v_A) = \{(\langle S, C, \rangle, 2), (\langle R, F, O, B\rangle, 1)\}$ to satisfy $\mathcal{A}(v_A)$. Applying the Strict rule to $\xi(v_A)$ derives (5.92); while applying the Sigma

TABLE 5.4: Clauses in Ontology $\mathcal{O}_3$, Example 5.6

$A \sqsubseteq \geq 2S.C \sqcup H$	$\rightsquigarrow$	$A(x) \rightarrow \exists_{\geq 2}\, y\ (S(x,y) \wedge C(y)) \vee H(x)$	(5.79)
$A \sqsubseteq \leq 1\ S.D \sqcup H$	$\rightsquigarrow$	$A \rightarrow \exists_{\leq 1}\, y\ (S(x,y) \wedge D(y)) \vee H(x)$	(5.80)
$A \sqsubseteq \exists R.F$	$\rightsquigarrow$	$A(x) \rightarrow \exists\, y\ (R(x,y) \wedge F(y))$	(5.81)
$C \sqsubseteq \exists S.o$	$\rightsquigarrow$	$C(x) \rightarrow \exists\, y\ (S(x,y) \wedge O(y))$	(5.82)
$C \sqsubseteq \forall R.B' \sqcup D$	$\rightsquigarrow$	$C(x) \rightarrow \forall\, y\ (R(x,y) \wedge B'(y)) \vee D(x)$	(5.83)
$F \sqsubseteq o$	$\rightsquigarrow$	$F(x) \rightarrow O(x)$	(5.84)
$F \sqsubseteq B$	$\rightsquigarrow$	$F(x) \rightarrow B(x)$	(5.85)
$B \sqcap B' \sqsubseteq \bot$	$\rightsquigarrow$	$B(x) \wedge B'(x) \rightarrow \bot$	(5.86)
		$\top \rightarrow \exists_{=1}\, x\ O(x)$	(5.87)

rule, creates node v_C initialized with (5.95); and connects v_A with v_C and v_O through edges (5.103) and (5.104) and also adds the (5.101) to $\mathcal{L}(v_O)$. Consequently, Sub rule may be applied in newly created nodes to derive clauses (5.96), (5.97), and (5.102).

The arithmetic label is updated as $\mathcal{Q}(v_C) = \{\exists\ y\ (R(x,y) \wedge O(y)), \forall\ y\ (R(x,y) \wedge B'(y))\}$ while $\mathcal{C}(v_C) = \{O(x) \rightarrow B(x)\}$ and $\mathcal{Q} = \{\exists_{=1} y\, O(y)\}$. Accordingly, ARM determines that $\mathcal{A}(v_C)$ is infeasible and calculates the conflict set as $CS(v_C) = \{O(x) \rightarrow B(x)\}$, so applying the Bottom rule to $CS(v_C)$ derives (5.98). Consequently, $C(x) \rightarrow D(x)$ is added to $\mathcal{C}_l(v_C)$ as a possible clause, which makes $\mathcal{A}(v_C)$ infeasible, with $CS(v_A) = \{C(x) \rightarrow D(x), \exists_{\geq 2} y\, (S(x,y) \wedge C(y)), \exists_{\leq 1} y\, (S(x,y) \wedge D(y))\}$. The Bottom rule is applied to $CS(v_A)$ to obtain (5.93). Finally, the Fct rule resolves $B(x)$ in (5.92) and (5.93) results in (5.94), which proves that our query clause is entailed from $\mathcal{O}_3$.

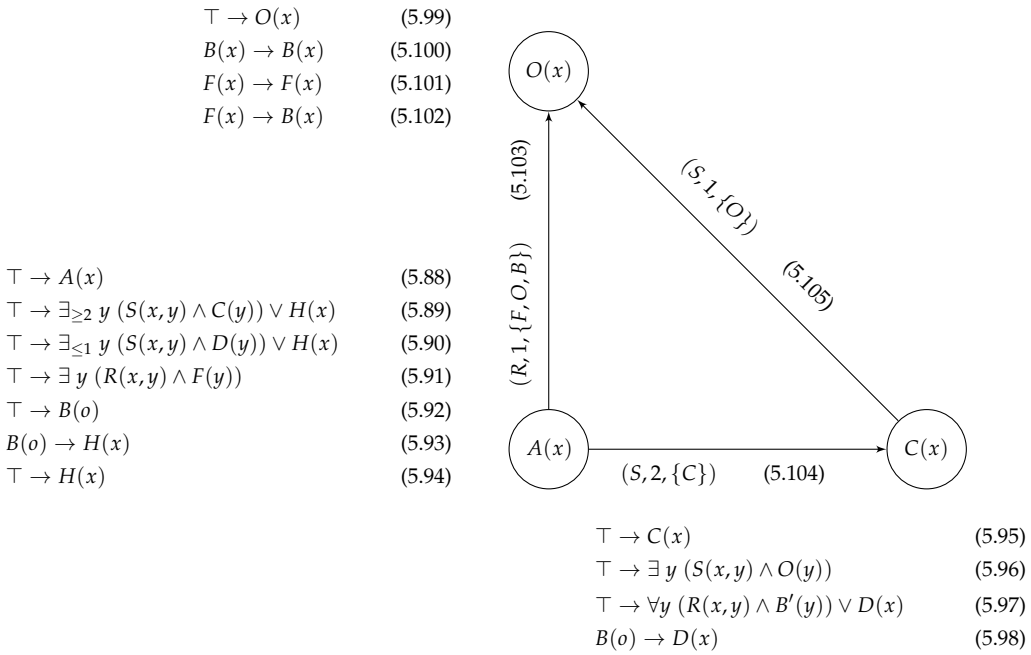


FIGURE 5.5: Algorithm Execution for Example 5.6

5.5 Summary and Conclusion

This chapter presents an algebraic CB reasoning algorithm for $\mathcal{SHOQ}$, which relies on ARM (Arithmetic Module) for reasoning about the number restrictions imposed by nominal and counting quantifiers. To the best of our knowledge, the proposed algorithm was the first CB calculus for the expressive description logic $\mathcal{SHOQ}$ [33].

During the reasoning process, the algorithm creates a single node to represent all elements associated with each concept. Using a representative node for each concept reduces the size of the generated framework and allows re-using elements. In contrast to tableau-based reasoners, our calculus naturally handles cyclic descriptions without the need for any blocking strategies to ensure termination.

Unlike most reasoning algorithms for $\mathcal{SHOQ}$, the algebraic CB method allows informed reasoning about the numerical restrictions on domain elements. In this chapter, we used ARM as a black box for finding an arithmetic solution that satisfies the constraints imposed by CQs and nominal.

ARM maps numerical restrictions to inequalities and handles obtained inequality systems using *integer linear programming methods*. The consequence-based $\mathcal{SHOQ}$ reasoner is based on the atomic decomposition technique, allowing the calculus to handle the various interactions between nominals, role successors, and their qualifications, the details of which are discussed in the next chapter.

The inference rules are designed to derive all consequences of presented axioms while benefiting *lazy unfolding* to avoid overwhelming the framework with unnecessary axioms. The inference rules are inspired by resolution to resolve complement literals.

So far, we have designed our calculus formally, and we tested it for a set of challenging examples. The proposed algorithm could successfully classify all designed ontologies. The next chapter covers the details of mapping numeric constraints to inequalities, determining their satisfiability and returning a solution to satisfy them or the minimum unsatisfiable sets of constraints.

Chapter 6

Arithmetic Module

Most DL reasoning techniques employ arithmetically uninformed methods for dealing with QCRs. They try to find a model that satisfies a particular concept by exhausting all possibilities. Searching for a model in such a blind way, especially if there are a large number of QCRs or if they contain large values, causes a tremendous performance degradation. The problem gets even worse when considering the interaction of QCRs with nominals. In this research, we use *Integer Linear Programming* (ILP), which is a mathematical solution for this problem.

The idea is to convert related number restrictions to inequalities. Since all obtained inequalities are linear, the problem is a *Linear Programming* problem, and since all the variables are integers, it can be considered an ILP problem. So, it can be solved using well-known ILP methods, namely *branch-and-bound algorithm*.

We implement the ILP method in a separate unit called *ARithmetic Module* (ARM), which accepts an arithmetic label as input. If the derived inequalities from the arithmetic label are feasible, ARM returns a solution that satisfies all these constraints. Otherwise, it returns the minimum literal sets that cause a contradiction as *conflict sets*.

The functionality of ARM is comprised of three main steps. ARM first produces the required variables employing *atomic decomposition* technique (Section 6.2). Then, it encodes the numerical restrictions into inequalities using generated variables (Section 6.3). Most importantly, it checks the satisfiability of derived inequalities and either returns a solution or the minimum unsatisfiable sets that cause the infeasibility (Section 6.4).

6.1 Arithmetic Label as Input

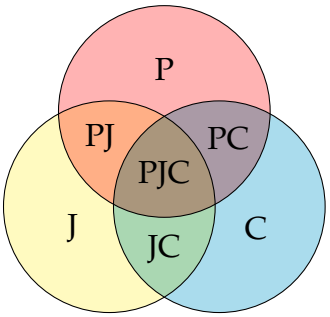
An arithmetic label contains two types of formulas. The first type is a *disjunction of CQs* of the form $\bigvee \exists_{\bowtie n} y \ \varphi(x,y)$, where $\varphi(x,y)$ is a Boolean combination of atoms or unary predicates. The second type is a clause of the form $\forall x,y \ \varphi(x,y) \rightarrow \psi(x,y)$. Both types satisfy the restriction that if a formula contains a binary predicate $R(x,y)$, then all included unary predicates are imposed on the variable y. Every other binary predicate is imposed on the ordered pair (x,y). Recall that an arithmetic label $\mathcal{A} = (\mathcal{Q},\mathcal{C})$ is a conjunction of elements of $\mathcal{Q}$ and $\mathcal{C}$, where $\mathcal{Q}$ is a set of disjunctions of CQs and $\mathcal{C}$ is a set of clauses, while the goal is to determine the satisfiability of $\mathcal{A}$.

6.2 Atomic Decomposition

Atomic decomposition was initially proposed in [42] for reasoning about sets. Later, it has been adapted to concept languages such as DL for reasoning about role successors of CQs [43]. This technique decomposes a finite set of elements, called decomposition set ($\mathcal{DS}$),

into mutually disjoint atomic components, called the *partitions*. Since the cardinality function of disjoint sets is additive, we have $|A \cup B| = |A| + |B|$ for every two subsets (partitions) A and B where $|\cdot|$ denotes the set cardinality. Accordingly, atomic decomposition allows converting the CQ feasibility problem to an ILP problem. For example, assume the following set of CQs:

$$\begin{cases} \exists_{\leq 5}\, y\, \mathsf{hasPublication}(x, y) \\ \exists_{\geq 3}\, y\, \mathsf{hasJournalPaper}(x, y) \\ \exists_{\geq 2}\, y\, \mathsf{hasConferencePaper}(x, y) \end{cases} \tag{6.1}$$



P = Publication, not Journal, not Conference
J = Jouranl, not Publication, not Conference
C = Conference, not Journal, not Publication
PJ = Publication, Journal, not Conference
PC = Publication, Conference, not Jouranl
JC = Jouranl, Conference, not Publication
PJC = Publication, Jouranl, Conference

FIGURE 6.1: Atomic Decomposition Example

Different partitions for translating these number restrictions to arithmetic inequalities are defined in Figure 6.1. In this example, the atomic decomposition is applied to the role successors of hasPublication, hasJouranlPaper, and hasConferencePaper, which are presented using the overlapping sets: publication, journal, and conference papers. In Figure 6.1, we illustrate all mutually disjoint subsets of $\mathcal{DS} = \{\texttt{Publication}, \texttt{Journal}, \texttt{Conference}\}$ as different overlapping areas. For conciseness, we assume that if a partition does not include a particular concept A, then it implicitly includes its negation $\neg A$.

Since the decomposed subsets are mutually disjoint, one can translate each number restriction expression into its corresponding inequality as follows:

$$\begin{aligned}
\exists_{\leq 5}\, y\, \mathsf{hasPublication}(x,y) &\quad\Rightarrow\quad |P| + |PJ| + |PC| + |PJC| \;\leq\; 5 \\
\exists_{\geq 3}\, y\, \mathsf{hasJournalPaper}(x,y) &\quad\Rightarrow\quad |J| + |PJ| + |JC| + |PJC| \;\geq\; 3 \\
\exists_{\geq 2}\, y\, \mathsf{hasConferencePaper}(x,y) &\quad\Rightarrow\quad |C| + |PC| + |JC| + |PJC| \;\geq\; 2
\end{aligned}$$

Having a decomposition set $\mathcal{DS}$ of size n, atomic decomposition produces 2^n disjoint subsets based on the elements of $\mathcal{DS}$. However, reflecting the concept relations such as disjointness and subsumption into arithmetic equations may reduce the number of logically allowed atomic subsets. This is the task of the so-called *infeasible-partition* operator to set the cardinality of some partitions to zero.

For example, the fact that a conference paper can not be a journal paper, $Journal \wedge Conference \rightarrow \bot$, means that having a partition that contains *Journal* and *Conference* is not logically allowed. It can be encoded to arithmetic inequalities by setting $|JC|$ and $|PJC|$ to zero. In addition, arithmetic encoding of the subsumption relation between *Journal* and *Publication* stating that all "journals" are "publications" ($Journal \rightarrow Publication$), similarly *Conference* and *Publication* ($Conference \rightarrow Publication$) makes $|J| = 0$ and $|C| = 0$, because a *Journal* or *Conference* paper is also a *Publication*. At last, $|P| = 0$ assuming that a *Publication* is either a *Journal* paper or *Conference* paper ($P(x) \rightarrow J(x) \vee C(x)$).

Definition 6.1 (Decomposition Set)**.** A *decomposition set* $\mathcal{DS}(v) = \{e_\ell(x,y) \mid 1 \leq \ell \leq k\}$ is an ordered set of all predicates occurring in $\mathcal{A}(v)$, of size k, where e_ℓ denotes a binary $R(x,y)$ or unary predicate $C(y)$. Let $\ell = h(e)$ be a function which maps every predicate e_ℓ

to its index $1 \leq \ell \leq k$ in $\mathcal{DS}(v)$.

Each partition ρ is a subset of $\mathcal{DS}$ and $\mathcal{P}$ is the set of disjoint partitions defined for the decomposition set $\mathcal{DS}$, $\mathcal{P} = \{\rho \mid \rho \subseteq \mathcal{DS}\}$. Therefore, a partition ρ is a set of unary and binary predicates. For example, $p_1 = \{A, B, C, R\}$ is a valid partition, where A, B and C are unary predicates and R is a binary predicate. If there is no unary predicate in a partition, then the top concept ($\top$) will be added by default.

Each partition ρ is associated with a variable $\sigma_p = |\rho^{\mathcal{I}}|$, which is equal to the cardinality of $\rho^{\mathcal{I}}$. For instance, σ_{AR} refers to the cardinality of the partition $p_1 = \{A, R\}$.

Definition 6.2 (Partitioning)**.** There are 2^k partitions of the form $\rho(x, y) = \bigwedge_{\ell=1}^{k} \lambda_\ell(x, y)$, where each $\lambda_\ell(x, y)$ is either $e_\ell(x, y)$ or its negation $\neg e_\ell(x, y)$ occurring in $\mathcal{A}(v)$. A partition $\rho(x, y)$ is called *feasible* if it is consistent with the clauses in $\mathcal{C}(v)$.

Assume $\mathcal{I}$ as an arbitrary model of $\mathcal{O}$. Each partition is interpreted as $\rho^{\mathcal{I}} \subseteq \Delta$, where $\rho^{\mathcal{I}} = \bigcap_{\ell=1}^{k} \lambda_\ell^{\mathcal{I}}$ and each $\lambda_\ell^{\mathcal{I}}$ is either $e_\ell^{\mathcal{I}}$ or its complement w.r.t. the domain Δ. When a partition ρ does not include a predicate, then its complement is part of the partition interpretation $\rho^{\mathcal{I}}$.

6.3 Deriving the Inequalities

An integer linear presentation of $\mathcal{A}(v)$ follows from encoding each partition as a $\{0, 1\}$-vector ρ of size k, in which for $1 \leq \ell \leq k$ the element $\rho_\ell = 1$, if in the partition $\rho(x, y)$ the element λ_ℓ is e_ℓ, and $\rho_\ell = 0$ otherwise. We denote $\rho[h(e)]$, the corresponding value of

the predicate e in the vector ρ as ρ_e. The number of positive elements β_p in each $\rho(x, y)$ is calculated as $\beta_p = \sum_{\ell=1}^{k} \rho_\ell$.

Only n_c *feasible partitions* ($n_c \leq 2^k$) are considered in constructing the ILP formulation. We can obtain a propositional formula based on $\mathcal{C}(v)$ by considering every predicate $p(x)$ or $p(x, y)$ as a propositional symbol p. Afterward, we can generate the set of all feasible partitions by applying any SAT solver on the propositional formula obtained from $\mathcal{C}(v)$ using similar techniques introduced by Finger and De Bona [21]. The following example illustrates how to transform CQs in Example 4.2 into inequalities.

Example 6.1 (Transform CQs to Inequalities)**.** The *decomposition set* $\mathcal{DS} = \{p_1, p_2, p_3, p_4\}$, so the feasible partitions are $\{p_1p_2, p_1p_2p_3, p_1p_2p_3p_4\}$, where the absence of a letter indicates the implied presence of its negation, e.g. $(p_1p_2)^{\mathcal{I}} = p_1^{\mathcal{I}} \cap p_2^{\mathcal{I}} \cap (\neg p_3)^{\mathcal{I}} \cap (\neg p_4)^{\mathcal{I}}$. In atomic decomposition, the subsets are semantically mutually disjoint, so we can map the above CQs to their corresponding inequalities as follows, where $|\cdot|$ designates set cardinality:

$$|p_1p_2| + |p_1p_2p_3| + |p_1p_2p_3p_4| \leq 45 \tag{6.2}$$

$$|p_1p_2| + |p_1p_2p_3| + |p_1p_2p_3p_4| \geq 30 \tag{6.3}$$

$$|p_1p_2p_3| + |p_1p_2p_3p_4| \geq 20 \tag{6.4}$$

$$|p_1p_2p_3p_4| \leq 2 \tag{6.5}$$

Based on the formal definition of an arithmetic label, we can formulate $\mathcal{A}(v)$ as the following integer linear problem, called *Master Problem* (MP):

Master Problem (MP):

$$\text{Min} \quad \sum_{j=1}^{n_c} \beta_{\rho^j} \sigma_j + \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} \boldsymbol{\gamma_i} \tag{6.6}$$

$$\text{S. T.} \quad \sum_{j=1}^{n_c} A_{ij}\sigma_j + \boldsymbol{B_i}\boldsymbol{\gamma_i} \bowtie b_i, \quad 1 \leqslant i \leqslant n_q \tag{6.7}$$

$$\sum_{j=1}^{n_c} A_{ij}\sigma_j = 1, \qquad n_q + 1 \leqslant i \leqslant n \tag{6.8}$$

$$\sum_{\gamma_i \in \bar{\gamma}_c} \boldsymbol{\gamma_i} \leq \boldsymbol{(k_c - 1)}, \quad 1 \leq c \leq m \tag{6.9}$$

$$\sigma_j \in \mathbb{N}^+, \gamma_i \in \{0, 1\} \tag{6.10}$$

In the Master Problem, A is a $n \times n_c$ binary matrix, where $n = n_q + n_o$, each column $1 \leqslant j \leqslant n_c$ corresponds to a feasible partition ρ^j, each row $1 \leqslant i \leqslant n_q$ corresponds to the ith counting quantifier in $\mathcal{Q}(v)$ of the form $\exists_{\bowtie_i n_i}\ R_i(x, y) \wedge C_i(y)$, where $\bowtie \in \{\leq, \geq\}$ and each row $n_q + 1 \leqslant i \leqslant n$ corresponds to a counting quantifier in $\mathcal{Q}$ of the form $\exists_{=1} y\ O_{i'}(y)$. Elements of matrix A for $1 \leqslant i \leqslant n_q$ are defined in (6.11) and for $n_q + 1 \leqslant i \leqslant n$ in (6.12) :

$$A_{ij} = \begin{cases} 1 & \text{if } \rho^j[h(C_i)] = 1 \text{ and } \rho^j[h(R_i)] = 1 \\ 0 & \text{otherwise} \end{cases} \qquad 1 \leqslant i \leqslant n_q \tag{6.11}$$

$$A_{ij} = \begin{cases} 1 & \text{if } \rho^j[h(O_{i'})] = 1, \text{where } i' = i - n_q \\ 0 & \text{otherwise} \end{cases} \qquad n_q + 1 \leqslant i \leqslant n \tag{6.12}$$

Let $\boldsymbol{b}$ be an $n \times 1$ integer vector, such that $b_i = n_i$ and $\boldsymbol{\sigma}$ be a $1 \times n_c$ vector of integer variables. The consequences of having two interacting predicates in the same partition must be checked in CB algorithms. In this regard, a desired solution of MP is one with fewer positive predicates in each partition. Accordingly, we use β_p as the coefficients in the *objective function* (6.6) to reflect the cost of each partition, i.e., a smaller number of predicates in a partition means a lower cost.

To model the or-relation between disjunct CQs in each $\mathcal{Q}_d$, we introduce binary variables γ_i corresponding to every CQ. Thus, $\boldsymbol{\gamma}$ is a $n \times 1$ binary vector, where $\bar{\gamma}_c = \{\gamma_i \mid \sum_{q=1}^{k_{c-1}} + 1 \leqslant i \leqslant \sum_{q=1}^{k_c}\}$ corresponds to CQs in $\mathcal{Q}_d$. Accordingly, $B_i\gamma_i$ is added to constraint (6.7) associated with each CQ, while assuming that M is a sufficiently big value, $B_i = M$ for every at-least CQ and $B_i = -M$ for every at-most CQ to model that regardless of the σ_j values, the ith constraint, $1 \leqslant i \leqslant n_q$, is always stratified if $\gamma_i = 1$.

Also, the constraint (6.9) needs to be included in the formulation corresponding to every disjunction of CQs to handle their or-relation, which works as follows for CQs in an arbitrary $\mathcal{Q}_d$: Since we need at least one out of k_d constraints to hold, there can be at most $k_d - 1$ constraints that *do not* hold. Therefore, constraint (6.9) ensures that at most $k_d - 1$

of the γ_i variables take the value 1 so that the associated B_i values are turned on, thereby eliminating the constraint. Alternative constraints are discussed in [13, Chapter 13].

Example 6.2 (Integer Linear Formalization)**.** The integer linear presentation of the normalized clauses in Example 4.2 is rendered as follows:

$$
\begin{array}{l|ccc}
\boldsymbol{h} & 1 & 1 & 1 \\
\boldsymbol{s} & 1 & 1 & 1 \\
\boldsymbol{\rho} & 1 & 0 & 0 \\
\boldsymbol{a} & 1 & 1 & 0 \\
\boldsymbol{u} & 1 & 1 & 0 \\
\boldsymbol{i} & 0 & 0 & 0 \\
\boldsymbol{c} & 0 & 0 & 0 \\
\hline
\boldsymbol{p_1} & \multicolumn{3}{c}{\multirow{4}{*}{$\underbrace{\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}}_{A}$}} & \multirow{4}{*}{$\cdot \underbrace{\begin{bmatrix} 1 \\ 19 \\ 10 \end{bmatrix}}_{x}$} & \multirow{4}{*}{$\begin{matrix} \leq \\ \geq \\ \geq \\ \leq \end{matrix} \underbrace{\begin{matrix} 45 \\ 30 \\ 20 \\ 2 \end{matrix}}_{b}$} \\
\boldsymbol{p_2} & & & & & \\
\boldsymbol{p_3} & & & & & \\
\boldsymbol{p_4} & & & & &
\end{array}
$$

Each row of A represents a predicate as indicated on the left, and each column is a feasible partition. The last four rows are associated with the CQs in $\mathcal{Q}$. Vector x specifies a possible feasible solution that satisfies all the corresponding inequalities. Obviously, there is a feasible solution for these inequalities, which means this problem is satisfiable.

The following example shows modelling a disjunction of CQs in ILP.

Example 6.3 (Modelling Disjunction of CQs)**.** In Example 4.2, we add a disjunct CQ to (i) as $\exists_{\leq 15}\, y\, p_2(x, y)$. Accordingly, we will have:

$$Q = \begin{cases} (i) & \exists_{\leq 45}\, y\; p_1(x,y) \vee \exists_{\leq 15}\, y\; p_2(x,y) \\ (ii) & \exists_{\geq 30}\, y\; p_2(x,y) \\ (iii) & \exists_{\geq 20}\, y\; p_3(x,y) \\ (iv) & \exists_{\leq 2}\, y\; p_4(x,y) \end{cases}$$

The above constraints may be formulated as:

$$\text{Min} \quad 2\sigma_{p_1p_2} + 3\sigma_{p_1p_2p_3} + 4\sigma_{p_1p_2p_3p_4} + \sum_{i=1}^{2} \gamma_i \tag{6.13}$$

$$\text{S.T.} \quad \sigma_{p_1p_2} + \sigma_{p_1p_2p_3} + \sigma_{p_1p_2p_3p_4} + 100\gamma_1 \leq 45 \tag{6.14}$$

$$\sigma_{p_1p_2} + \sigma_{p_1p_2p_3} + \sigma_{p_1p_2p_3p_4} + 100\gamma_2 \leq 15 \tag{6.15}$$

$$\sigma_{p_1p_2} + \sigma_{p_1p_2p_3} + \sigma_{p_1p_2p_3p_4} \geq 30 \tag{6.16}$$

$$\sigma_{p_1p_2p_3} + \sigma_{p_1p_2p_3p_4} \geq 20 \tag{6.17}$$

$$\sigma_{p_1p_2p_3p_4} \leq 2 \tag{6.18}$$

$$\gamma_1 + \gamma_2 \leq 1 \tag{6.19}$$

$$\sigma_j \in \mathbb{N}^+, \gamma_i \in \{0,1\} \tag{6.20}$$

The inequalities corresponding to (i) are still satisfiable, with $\gamma_1 = 1$ or $\gamma_2 = 1$. Note that to satisfy constraint (6.19), at least one of them has to be zero. Accordingly, one can ensure that at least one of the CQs in the disjunction (i) holds. One can readily see that $\gamma_2 = 1$ should be in the above ILP.

Lemma 6.1. The arithmetic label $\mathcal{A}(v) = (\mathcal{Q}(v), \mathcal{C}(v))$ of node $v \in \mathcal{V}$ is satisfiable if and only of its corresponding MP is feasible.

Proof. We assume the mapping presented in the previous paragraphs is a one-to-one mapping between axioms and linear inequalities.

($\Rightarrow$) Assuming that $\mathcal{A}(v)$ is satisfiable, there exists a non-empty interpretation $\mathcal{I}$. Due to the fact that partitions are semantically disjoint, every $\alpha \in \Delta^{\mathcal{I}}$ is a member of at most one partition interpretation. Let n_j be the number of elements in p_j; one can obtain a feasible solution for MP by setting $\sigma_j = n_j$, for $1 \leq j \leq n_c$. Since the clauses in $\mathcal{C}(v)$ are universal restrictions, for every $\alpha \in \Delta^{\mathcal{I}}$, we have $\alpha \in (\mathcal{C}(v))^{\mathcal{I}}$, thus only feasible partitions would have non-zero values.

($\Leftarrow$) Assume that MP has a feasible solution σ. One can construct an interpretation $\mathcal{I}$ for $\mathcal{A}(v)$ by creating σ_j elements in each $\rho_j^{\mathcal{I}} \in \Delta^{\mathcal{I}}$, where ρ_j is a feasible partition satisfying the clauses in $\mathcal{C}(v)$. □

6.4 Returning Solution or Conflict Sets

After formulating the arithmetic label as an ILP problem in MP, the primary goal is determining its satisfiability. To pursue this goal, one needs to discover whether there is a solution to the inequality system.

However, enumerating all feasible partitions makes MP intractable because an exponential number of columns need to be processed and possibly added to A for solving MP.

To address this problem, we solve ILP via a branch-and-bound algorithm, which relies on *Column Generation* (CG). CG is an efficient, iterative algorithm for solving *linear program* (LP) problems with a large number of variables.

6.4.1 Solving the MP via Branch-and-Bound

Algorithm 1 shows our branch-and-bound method at a high level. In step 2, CG is called. If there exists no solution, CG returns false, and the corresponding *conflict sets* ($\mathcal{CS}$) are calculated by calling the MARCO algorithm 5. If there exists an integral solution, the algorithm terminates and returns the integer solution (ξ), steps 3-6.

Algorithm 1 Branch-and-Bound (B&B Algorithm)

Input: An arithmetic label $\mathcal{A} = (\mathcal{Q}, \mathcal{C})$
Output: An integer solution for MP; or the conflict set

1: **procedure** BRANCH-AND-BOUND($\mathcal{Q}, \mathcal{C}$)
2: *solution* = column-generation($\mathcal{Q}, \mathcal{C}$)
3: **if** *solution* is infeasible **then**
4: **return** *false*, $\mathcal{CS} = ConflictSets(\mathcal{Q}, \mathcal{C})$
5: **else if** *solution* is integral **then**
6: **return** *true*, ξ = *integer solution*
7: **else**
8: *var* = branchVar(*solution*)
9: ($\mathcal{Q}_1$, $\mathcal{Q}_2$, $\mathcal{C}_{new}$) = bounded-problems($\mathcal{Q}, \mathcal{C}$, *var*)
10: **return** B&B($\mathcal{Q}_1, \mathcal{C}_{new}$) **or** B&B($\mathcal{Q}_2, \mathcal{C}_{new}$)
11: **end if**
12: **end procedure**

Otherwise, there exists a solution with at least one non-integral variable. In step 8, a heuristic is used to find a variable σ_j with a non-integral value z_j on which the branching is applied. The heuristic chooses σ_j such that its non-integral value z_j is closer to either

$\lfloor z_j \rfloor$ or $\lceil z_j \rceil$. Afterwards in step 10, the algorithm is called recursively on the two generated sub-problems $\mathcal{A}_1 = (\mathcal{Q}_1, \mathcal{C}_{new})$ and $\mathcal{A}_2 = (\mathcal{Q}_2, \mathcal{C}_{new})$, where it is assumed that p^* is a new unary predicate such that $\mathcal{Q}_1 = \mathcal{Q} \cup \{\exists_{\leq \lfloor z_j \rfloor} y \ p^*(y)\}$, $\mathcal{Q}_2 = \mathcal{Q} \cup \{\exists_{\geq \lceil z_j \rceil} y \ p^*(y)\}$ and $\mathcal{C}_{new} = \mathcal{C} \cup \{\{p^*(y) \rightarrow \rho_j(x,y), \rho_j(x,y) \rightarrow p^*(y)\}$, while $\rho_j(x,y)$ is the partition corresponding to column j.

Column Generation

The column generation algorithm consists of two separate LP problems. The first LP problem is the *restricted* master problem (RMP) (defined in (6.21) – (6.24)), in which integrality constraints on σ and γ are relaxed from $\mathbb{N}$ to $\mathbb{R}$. The second LP is the *pricing problem* (PP) (defined in (6.25) – (6.31), which generates the columns of A as needed to decrease RMP's objective value.

Restricted Master Problem (RMP)

$$OV = \text{Min} \sum_{j=1}^{n_c} \beta_{\rho^j} \sigma_j + \sum_{i=1}^{n} \gamma_i + M \sum_{i=1}^{n} h_i \tag{6.21}$$

$$\text{S. T.} \sum_{j=1}^{n_c} A_{ij}\sigma_j + B_i\gamma_i + h_i \bowtie b_i, \quad 1 \leq i \leq n \tag{6.22}$$

$$\sum_{\gamma_i \in \bar{\gamma}_d} \gamma_i \leq (k_d - 1), \qquad 1 \leq d \leq m \tag{6.23}$$

$$\sigma_j, \gamma_i \in \mathbb{R}^+, \ 0 \leq \gamma_i \leq 1 \tag{6.24}$$

CG accepts an arithmetic label $\mathcal{A} = (\mathcal{Q}, \mathcal{C})$ as input and returns the generated partitions corresponding to the columns of A and the final values of σ. Algorithm 2 presents a high-level version of CG. In order to start CG, it is essential to find an *initial feasible solution* with $\mathcal{C}$-satisfying columns for the constraints in RMP.

Algorithm 2 Column-Generation($\mathcal{Q}, \mathcal{C}$)

Input: An arithmetic label $\mathcal{A} = (\mathcal{Q}, \mathcal{C})$
Output: Generated partitions and a solution for RMP, if it is feasible; *infeasible* otherwise

procedure COLUMN-GENERATION($\mathcal{Q}, \mathcal{C}$)
  Initialize A; $\beta = \emptyset$; $\mathcal{P} = \emptyset$; $RC = -1$
  while $RC < 0$ **do**
    $(OV, x, h) = RMP.solve(A, \beta)$
    $\omega = RMP.duals(A, \beta)$
    $(\rho, d, RC) = PP.solve(\omega, \mathcal{C})$
    if $RC < 0$ **then**
      $A = merge(A, d)$
      $\mathcal{P} = merge(\mathcal{P}, \rho)$
      $\beta = append(\beta, sum(\rho))$
    end if
  end while
  if $h == 0$ **and** $OV < M$ **then**
    return $\sigma, \mathcal{P}$
  else
    return *infeasible*
  end if
end procedure

We resolve this issue by introducing an $n \times 1$ vector $\boldsymbol{h}$ of artificial variables h_i, each associated with a constraint in (6.22). By assigning a sufficiently large cost M to each h_i in the objective function (6.21), since it is a minimization problem, one can ensure that if there exists a feasible solution for the original MP, all these artificial variables would be equal to zero. Otherwise, if some of these variables are still non-zero on the termination

of CG, it means that the original problem is infeasible.

In step 2, Algorithm 2 initializes $\boldsymbol{A}$ as an empty matrix; the artificial initial solution is returned as $\boldsymbol{h} = \boldsymbol{b}$ while all elements of $\boldsymbol{\sigma}$ and $\boldsymbol{\gamma}$ are equal to zero. Afterwards, in each iteration (steps 4-10) RMP is solved for $\boldsymbol{A}$ and $\boldsymbol{\beta}$. Consequently, the *primal solution* $\boldsymbol{\sigma}$ and the *dual values* $\boldsymbol{\omega}$ are calculated and $\boldsymbol{\omega}$ is used to parameterize the *pricing problem* (PP). If CG terminates (steps 13-16), it means RC is no longer negative, RMP's current objective value (OV) cannot be decreased anymore, and the generated columns and their final values of σ_j are returned. Otherwise, the generated column d is merged into $\boldsymbol{A}$ and $\boldsymbol{\beta}$ is updated accordingly (steps 8-10).

Pricing Problem (PP)

$$RC = \text{Min} \quad \sum_{\ell=1}^{k} \rho_\ell - \sum_{i=1}^{n} \omega_i d_i \tag{6.25}$$

$$\text{S. T.} \quad \sum_{s=1}^{n'} \rho_{p_s} \leq \sum_{t=1}^{m'} \rho_{p_t} + (n' - 1) \tag{6.26}$$

$$d_i = \rho_{p_i} \tag{6.27}$$

$$\rho_C \leq \rho_\top, \qquad C(x) \in \mathcal{D} \tag{6.28}$$

$$\rho_R \leq \rho_C \tag{6.29}$$

$$\rho_\perp \leq 0,\ \rho_\top \geq 1 \tag{6.30}$$

$$\rho_\ell, d_i \in \{0, 1\} \tag{6.31}$$

The Pricing Problem (PP) generates a new column $\boldsymbol{d}$ with a negative *Reduced Cost* (*RC*) (6.25) to be merged into matrix $\boldsymbol{A}$ in order to decrease RMP's objective value (6.21). In constraint (6.25) $\sum_{\ell=1}^{k} \rho_\ell$ is the cost associated with $\boldsymbol{d} = [d_1, ..., d_n]^T$, in which $\rho_1, ..., \rho_k$ are binary values representing the partition corresponding to $\boldsymbol{d}$, and $\boldsymbol{\omega} = [\omega_1, ..., \omega_n]$ is the *dual solution* of RMP. Constraints (6.28) and (6.30) are added to PP if $\top \in \mathcal{DS}$ and $\bot \in \mathcal{DS}$, respectively, and constraint (6.27) is added for each CQ of the form $\exists_{\bowtie n} p_i$ that occurs in $\mathcal{Q}$.

Providing more information to ARM reduces the risk of returning a solution that will fail later during the reasoning process. Presenting other information, such as subsumption and disjointness to ARM, prevents generating infeasible partitions by CG. The variables ρ_ℓ are used for modelling feasible partitions. Constraints are added to PP to prevent generating partitions $\boldsymbol{\rho}$ and columns $\boldsymbol{d}$ that would be infeasible w.r.t. $\mathcal{C}$.

- **Atomic clauses** of the form $\forall x, y \ \bigwedge_{s=1}^{n'} p_s(x, y) \rightarrow \bigvee_{t=1}^{m'} p'_t(x, y) \in \mathcal{C}$ are added to PP as the (6.26) inequality. These constraints model Boolean relations between predicates, such that if all the binary variables ρ_{p_s} associated with the predicates in the clause's body are equal to one, then at least one of the binary variables ρ_{p_t} associated with the predicates in the clause's head has to be one; otherwise all ρ_{p_t}, for $1 \leq t \leq m'$ have to be zero. For example, the atomic clause $A(x) \rightarrow B(x) \in \mathcal{C}$ produces $\rho_A \leq \rho_B$.

- **Role Subsumptions** of the form $R(x, y) \rightarrow S(x, y)$ are also added to PP encoded as (6.26) inequality which produces $\rho_R \leq \rho_S$.

- **Universal Restrictions** of the form $\forall\, y\, R(x, y) \wedge C(y)$ are encoded as (6.29) constraint. The semantics of universal restriction implies that all $R(x, y)$ successors are

instances of $C(y)$. The generated inequality satisfies this semantic by implying that a partition containing the binary predicate R should also contain the unary predicate C.

Example 6.4 (Column Generation)**.** This example describes the column generation process for solving the constraints in Example 4.2. The process starts by producing the RMP based on artificial variables h_i and assigning an arbitrarily large value $M = 1000$ as their cost. Accordingly, the RMP 1 would be defined as:

$$\begin{aligned} \text{Min} \quad & 1000 \sum_{i=1}^{5} h_i \\ \text{S. T.} \quad & h_1 \leq 45 \\ & h_2 \geq 30 \\ & h_3 \geq 20 \\ & h_4 \leq 2 \\ & h_5 \geq 3 \end{aligned} \tag{RMP1}$$

Solution: $OV = 5000, \boldsymbol{h} = [0, 30, 20, 0, 3]^T$;

Dual values: $\boldsymbol{\omega} = [0, 1000, 1000, 0, 1000]$.

Dual values are used as coefficients of d variables in the objective function of the pricing problem (reduced cost). The ρ variables show whether a particular member of the decomposition set positively occurs in the newly added partition. These variables are introduced to model the semantics of clauses in pricing problems to prevent the generation of semantically infeasible partitions. The d variables are associated with the inequalities

and indicate the constraint to which the variable $\sigma_{\boldsymbol{\rho}}$ has to be added. The variable $\sigma_{\boldsymbol{\rho}}$ corresponds to the newly generated partition $\boldsymbol{\rho}$.

Through the rest of this example, all variables with a value equal to zero or a zero coefficient are omitted for simplicity.

$$
\begin{aligned}
\text{Min} \quad & \rho_{p_1} + \rho_{p_2} + \rho_{p_3} + \rho_{p_4} + \rho_{p_5} + \rho_s && \text{(PP1)}\\
& + \rho_u + \rho_\perp + \rho_h + \rho_c + \rho_p + \rho_a + \rho_i \\
& - 1000d_2 - 1000d_3 - 1000d_5 \\
\text{S.T.} \quad & d_i = \rho_{p_i} \qquad 1 \leq i \leq 4 \\
& \rho_\perp \leq 0 \\
& \rho_s + \rho_h \leq \rho_{p_1} \\
& \rho_{p_3} \leq \rho_h \\
& \rho_p + \rho_a \leq \rho_{p_4} \\
& \rho_{p_2} \leq \rho_h \\
& \rho_{p_3} \leq \rho_a \\
& \rho_i \leq \rho_c \\
& \rho_{p_2} \leq \rho_s \\
& \rho_{p_2} + \rho_c \leq 1 \\
& \rho_{p_3} \leq \rho_s \\
& \rho_{p_3} \leq \rho_p + \rho_i + \rho_u
\end{aligned}
$$

Solution: $RC = -2992$, $d = [1,1,1,0,0]$, $\rho = [1,1,1,1,0,0,1,0,1,0,0,1,1]$.

We solve the pricing problem by modelling it in CPLEX. We fix the order of elements in the decomposition so that the returned binary vector $\boldsymbol{\rho}$ could be interpreted as $\rho_{p_1} = 1, \rho_a = 1, \rho_{p_2} = 1, \rho_{p_3} = 1, \rho_u = 1, \rho_{p_5} = 1$. As a result, we generate the partition

$\langle p_1(x,y), p_5(x,y), a(x), p_2(x,y), p_3(x,y), s(x), u(x), h(x,y)\rangle$ and add its associated variable σ_1 to RMP based on the value of d variables. So RMP is updated as:

$$\begin{aligned} \text{Min} \quad & 1000\sum_{i=1}^{5} h_i + \mathbf{8\sigma_1} \\ \text{S.T.} \quad & h1 + \sigma_1 \leq 45 \\ & h2 + \boldsymbol{\sigma_1} \geq 30 \\ & h3 + \boldsymbol{\sigma_1} \geq 20 \\ & h4 \leq 2 \\ & h_5 + \boldsymbol{\sigma_1} \geq 3 \end{aligned} \tag{RMP2}$$

Solution: $OV = 240, \boldsymbol{h} = [0,0,0,0,0]^T, \boldsymbol{x} = [30]$; Dual Values $\boldsymbol{\omega} = [0,8,0,0,0]$.

In the objective function of (PP2), the only non-zero dual value is $\omega_2 = 8$.

$$\text{minimize} \quad \sum_{pr\in\mathcal{D}} \rho_{pr} - 8d_2 \tag{PP2}$$

Solution: $RC = -4.0, d = [1,1,0,0,0], \rho = [1,0,1,0,0,0,0,0,1,0,0,1,0]$.

Since only the value of d_1 and d_2 is 1, we add the variable σ_2 to the first two inequalities of RMP2 whose cost is further reduced, from 240 in (RMP2) to 200 in (RMP3). This variable is associated with the partition $\boldsymbol{\rho} = [p_1(x,y), p_2(x,y), s(x), h(x,y)]$.

$$\begin{aligned} \text{Min} \quad & 1000 \sum_{i=1}^{5} h_i + 8\sigma_1 + \mathbf{4\sigma_2} \\ \text{S.T.} \quad & h1 + \sigma_1 + \boldsymbol{\sigma_2} \leq 45 \\ & h_2 + \sigma_1 + \boldsymbol{\sigma_2} \geq 30 \\ & h_3 + \sigma_1 \geq 20 \\ & h_4 \leq 2 \\ & h_5 + \sigma_1 \geq 3 \end{aligned} \tag{RMP3}$$

Solution: $OV = 200, \boldsymbol{h} = [0,0,0,0,0], \boldsymbol{x} = [20,10]$;

Dual values: $\boldsymbol{\omega} = [0,4,4,0,0]$

In the objective function of (PP3) the only non-zero dual values are $\omega_2 = 4$ and $\omega_3 = 4$.

$$\text{minimize} \quad \sum_{pr \in \mathcal{D}} \rho_{pr} - 4d2 - 4d3 \tag{PP3}$$

Solution: $RC = -1, d = [1.0, 1.0, 1.0, 0.0, 0.0]$,

$\rho = [1,1,1,1,0,0,0,0,1,0,0,1,1]$.

Since the values of d_1, d_2 and d_3 are 1 we add the variable σ_3 to the next version of our RMP which corresponds to $\langle p1(x,y), p_1(x,y), h(x,y), s(y) \rangle$ and reduces the objective value from 200 to 183 in (RMP3).

$$
\begin{aligned}
\text{Min} \quad & 1000 \sum_{i=1}^{5} h_i + 8\sigma_1 + 4\sigma_2 + \mathbf{7\sigma_3} \\
\text{S.T.} \quad & h1 + \sigma_1 + \sigma_2 + \boldsymbol{\sigma_3} \leq 45 \\
& h_2 + \sigma_1 + \sigma_2 + \boldsymbol{\sigma_3} \geq 30 \\
& h_3 + \sigma_1 + \boldsymbol{\sigma_3} \geq 20 \\
& h_4 \leq 2 \\
& h_5 + \sigma_1 \geq 3
\end{aligned}
\tag{RMP3}
$$

Solution: $OV = 183, \boldsymbol{h} = [0.0, 0.0, 0.0, 0.0, 0.0], \boldsymbol{x} = [3, 10, 17]$;

Dual values $\boldsymbol{\omega} = [0.0, 4.0, 3.0, 0.0, 1.0]$.

$$
\text{minimize} \quad \sum_{pr \in \mathcal{D}} \rho_{pr} - 3d_3 - d_5 \tag{PP4}
$$

Solution: $RC = 4.44E - 16$

All d variables are zero, and the value of the reduced cost is not negative anymore. Therefore, no new variable can be added to further decrease the objective value of (RMP3).

6.4.2 Finding the Conflict Sets

When the inequality system associated with the arithmetic label is unsatisfiable, we need to extract the *Minimum Unsatisfiable Sets* (*Conflict Set*). The axioms associated with this set can then be resolved using the resolution rules. To enumerate all conflict sets, we deploy an optimized algorithm that is based on the close relation between Maximal Satisfiable

Subsets (MSSes) and Minimal Unsatisfiable Subsets (MUSes) [38]. As their name suggests, the known axioms always hold and will be fully resolved as part of the resolution process if they are part of the conflict set. Accordingly, we only consider the constraints associated with the possible axioms as part of the unsatisfiability analysis.

Assume that $C = \{C_1, C_2, ..., C_n\}$ is the ordered set of constraints associated with possible axioms. A given constraint C_i places restrictions on assignments to a problem's variables, and C_i is satisfied by any assignment that meets its restrictions. If there exists some assignment to C's variables that satisfies every constraint, C is said to be satisfiable; otherwise, it is unsatisfiable.

The Maximal Satisfiable Subset (MSS) produces a satisfiable subset of C with the greatest possible cardinality. Generalizing MaxSAT by considering maximality instead of maximum cardinality yields the concept of a Maximal Satisfiable Subset (MSS):

$$M \subseteq C \text{ is an } MSS \iff M \text{ is SAT and } \forall c \in C \setminus M : M \cup \{c\} \text{ is UNSAT}$$

An MSS is essentially a satisfiable subset of C that cannot be expanded without becoming unsatisfiable. While any solution to the MaxSAT problem is an MSS, some MSSes may be smaller than that maximum size. The complement of an MSS is often more directly useful, and we call such a minimal set (whose removal from C makes it satisfiable or "corrects" it) a Minimal Correction Set (MCS):

$$M \subseteq C \text{ is an MCS} \iff C \setminus M \text{ is SAT} \text{ and } \forall c \in M : (C \setminus M) \cup \{c\} \text{ is UNSAT}$$

Again, the minimality is not in terms of cardinality, but rather it requires that no proper

Algorithm 3 grow$(seed, C)$

Input: Unsatisfiable constraint set C
Input: Satisfiable subset $seed \subset C$
Output: An *MMS* of C

```
for c ∈ C \ seed do
    if seed ∪ {c} is satisfiable then
        seed = seed ∪ {c}
    end if
end for
return seed
```

Algorithm 4 shrink$(seed, C)$

Input: Unsatisfiable constraint set C
Input: Unsatisfiable subset $seed \subseteq C$
Output: An *MMS* of C

```
for c ∈ C \ seed do
    if seed ∪ {c} is unsatisfiable then
        seed = seed ∪ {c}
    end if
end for
return seed
```

FIGURE 6.2: The **grow** and **shrink** Methods for Finding a MSS or a MUS, Respectively, of a Constraint Set

subset of M be capable of "correcting" the infeasibility. A related concept, which is required by our resolution algorithm, is the Minimal Unsatisfiable Subset (MUS):

$$M \subseteq C \text{ is an } MUS \iff M \text{ is UNSAT and } \forall c \in M : M \setminus \{c\} \text{ is SAT}$$

We consider MUSes in terms of minimizing an unsatisfiable constraint set down to a "core" reason for its unsatisfiability.

The algorithms for finding MSSes and MUSes of a constraint set C are shown in Figure 6.2. To find an MSS (MUS), the **grow** (**shrink**) method starts from some satisfiable (unsatisfiable) subset $seed \subseteq C$ and iteratively attempts to add (remove) constraints, checking each new set for satisfiability and keeping any changes that leave the set satisfiable (unsatisfiable).

Note that the input *seed* can take simple default values if no particular subset is given. The **grow** method can begin its construction with $seed = \emptyset$ (guaranteed to be satisfiable), while shrink can start with $seed = C$ (guaranteed UNSAT). Therefore, *seed* can be considered an optional parameter for both.

Algorithm 5 MARCO - (ConglictSets (Q, C))

Input: Unsatisfiable constraint set $C = \{C_1, C_2, C_3, C, ..., C_n\}$
Output: MSSes and MUSes of C as they are discovered

```
1:  Map ← BoolFormula(nvars = |C|)          ▷ Empty formula over |C| Boolean variables
2:  while Map is satisfiable do
3:      m ← getModel(Map)
4:      seed ← {C_i ∈ C : m[x_i] = True}     ▷ Project the assignment m onto C
5:      if seed is unsatisfiable then
6:          MSS ← grow (seed, C)
7:          yield MMS
8:          Map ← Map ∧ blockDown(MMS)
9:      else
10:         MUS ← shrink (seed, C)
11:         yield MUS
12:         Map ← ∧ blockUp(MUS)
13:     end if
14: end while
15: return seed
```

FIGURE 6.3: The MARCO algorithm for enumerating MSSes & MUSes of a constraint set

In this research, we deploy the MARCO algorithm [38] for enumerating all MUSes of an unsatisfiable constraint set C which efficiently explores the power set $P(C)$ by exploiting the idea that any power set can be analyzed and manipulated as a Boolean algebra.

The idea is that any function $f : P(C) \rightarrow \{0, 1\}$ can be represented by a propositional formula over $|C|$ variables. This algorithm maintains the function f that tracks "unexplored" subsets $C' \subseteq C$ such that $f(C') = 1$ iff the satisfiability of C' is unknown and it remains to be checked. This function can be viewed as a "map" of $P(C)$ showing which "regions" have been explored and which have not. Overall, MARCO enumerates MUSes by repeatedly selecting an unexplored subset $C' \in P(C)$ from the map, checking whether C' is satisfiable, minimizing or maximizing it into a MUS or an MSS, and marking a region of the map as explored based on that result.

Figure 6.3 contains pseudo-code for the MARCO algorithm. The formula Map is created to represent the "mapping" function described above, with a variable x_i for every constraint C_i in C. Since the formula is initially a tautology, it holds in every model, which means that every subset of C is still unexplored. Given its semantics, any model of Map can be projected onto C (lines 3 and 4) to identify a yet unexplored element of $P(C)$ whose satisfiability is currently unknown. If this subset, *seed*, is satisfiable, then it must be a subset of some MSS, and it can be "grown" into an MSS. Likewise, if it is unsatisfiable, *seed* must contain at least one MUS, and it can be "shrunk" to produce one. In either case, the result is reported (via yield in the pseudocode, indicating that the result is returned, but the algorithm may continue).

Each result provides information about some region of $P(C)$ that is either satisfiable or unsatisfiable, and so a clause is added to Map to represent that region as "explored." For an MSS M, all subsets of M are now known to be satisfiable, and so models corresponding to any subset of M are eliminated by requiring that later models of Map include at least one constraint not in M:

$$\textbf{blockDown}(M) \equiv \bigvee_{i\,:\,C_i \notin M} x_i$$

Similarly, all supersets of any MUS M are known to be unsatisfiable; supersets of M are blocked by requiring models to exclude at least one of its constraints:

$$\textbf{blockUp}(M) \equiv \bigvee_{i\,:\,C_i \in M} \neg x_i$$

Eventually, all MSSes and MUSes are enumerated, the satisfiability of every element in $P(C)$ is known, and MARCO terminates when Map has no further models.

6.5 Summary and Conclusion

The first step is to convert all constraints to inequalities using atomic decomposition. We then discussed how to find a solution to the resulting ILP without generating an exponential number of variables in advance. Column generation techniques add variables as needed during the solution process. If no solutions are found by the end of the process, then the inequality system is unsatisfiable. To capture the consequences of this unsatisfiability, we need to find the minimal unsatisfiable subset that caused it. The next chapter proves that the proposed algorithm is sound and complete, meaning that it finds all entailed clauses and only entailed clauses.

Chapter 7

Key Properties of the Calculus

This section contains proofs for the main properties of the calculus presented in the previous chapters. Namely, in Section 7.1, we prove the soundness of the proposed calculus and Section 7.2 proves the completeness of our calculus.

7.1 Proof of Soundness

This section proves the Theorem 5.1, assuming that both $\mathcal{G}_1$ and $\mathcal{L}_1$ are sound, we next show that both $\mathcal{G}_2$ and $\mathcal{L}_2$ are sound as well. To this end, let $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ be an arbitrary model of $\mathcal{O}$, and consider all possible inference rules that derive a clause in $\mathcal{L}_2$ or modify $\mathcal{G}_2$.

(Subs Rule) Assume that the rule is applied as in Table 5.1; we will show that $\mathcal{O} \models \mathsf{core}(v) \wedge \bigwedge_{i=1}^{n} K_i \rightarrow \bigvee_{i=1}^{n} M_i \vee M$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (\mathsf{core}(v) \wedge \bigwedge_{i=1}^{n} K_i)^{\mathcal{I}}$. Now $\mathcal{L}_1$ is sound for $\mathcal{O}$ so, for each $1 \leqslant i \leqslant n$, we have $\mathcal{O} \models \mathsf{core}(v) \wedge K_i \rightarrow M_i \vee A_i$, which implies $\delta \in (M_i \vee A_i)^{\mathcal{I}}$. If $\delta \in M_i^{\mathcal{I}}$ for some $1 \leqslant i \leqslant n$, then clearly $\delta \in (\bigvee_{i=1}^{n} M_i)^{\mathcal{I}}$. Otherwise, we have $\delta \in (\bigwedge_{i=1}^{n} A_i)^{\mathcal{I}}$, but then

$\bigwedge_{i=1}^{n} A_i \rightarrow M \in \mathcal{O}$ implies $\delta \in M^{\mathcal{I}}$. In either case, we have $\delta \in (M \wedge \bigwedge_{i=1}^{n} M_i)^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

(Join Rule) Assume that the rule is applied as in Table 5.1; we will show that $\mathcal{O} \models \mathsf{core}(v) \wedge K \wedge K' \rightarrow M \vee M'$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (\mathsf{core}(v) \wedge K \wedge K')^{\mathcal{I}}$. Now $\mathcal{L}_1$ is sound for $\mathcal{O}$ so, we have $\mathcal{O} \models \mathsf{core}(v) \wedge K \rightarrow M \vee C(o)$, which implies $\delta \in (M \vee C(o))^{\mathcal{I}}$. If $\delta \in M^{\mathcal{I}}$, then clearly $\delta \in (M \vee M')^{\mathcal{I}}$, otherwise, we have $\delta \in (C(o))^{\mathcal{I}}$. But then, following our initial assumption, we have $\delta \in (C(o) \wedge K')^{\mathcal{I}}$. Once again, since $\mathcal{L}_1$ is sound for $\mathcal{O}$, we have $\mathcal{O} \models \mathit{core}(v) \wedge C(o) \wedge K' \rightarrow M'$, which implies that $\delta \in (M')^{\mathcal{I}}$. So we proved that $\delta \in (M \wedge M')^{\mathcal{I}}$ holds in either case. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

(Fct Rule) Assume that the rule is applied as in Table 5.1; we will show that $\mathcal{O} \models O(x) \wedge K \rightarrow M \vee C(x)$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (O(x) \wedge K)^{\mathcal{I}}$. Now $\mathcal{L}_1$ is sound for $\mathcal{O}$ so, we have $\mathcal{O} \models O(x) \wedge K \rightarrow M \vee C(o)$, which implies $\delta \in (M \vee C(o))^{\mathcal{I}}$. If $\delta \in M^{\mathcal{I}}$, then clearly $\delta \in (M \vee C(x))^{\mathcal{I}}$. Otherwise, we have $\delta \in (C(o))^{\mathcal{I}}$, which implies that $\delta \in (O(x) \rightarrow C(x))^{\mathcal{I}}$. Following our assumption $\delta \in (O(x))^{\mathcal{I}}$, so we have $\delta \in (C(x))^{\mathcal{I}}$. Therefore, we proved that in either case, $\delta \in (M \vee C(x))^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

(Glob Rule) This rule is divided into two branches depending on the preconditions:

> **Branch 1**: Assume that the rule is applied as in Table 5.1; we will show that $\mathcal{O} \models \mathsf{core}(v) \wedge \bigwedge_{i=1}^{n} K_i \wedge K' \wedge K\sigma_o \rightarrow \bigvee_{i=1}^{n} M_i \vee M \vee M'$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (\mathsf{core}(v) \wedge \bigwedge_{i=1}^{n} K_i \wedge K' \wedge K\sigma_o)^{\mathcal{I}}$. Now $\mathcal{L}_1$ is sound for $\mathcal{O}$ so, for each $1 \leqslant i \leqslant n$, we have $\mathcal{O} \models \mathsf{core}(v) \wedge K_i \rightarrow M_i \vee A_i$, which implies $\delta \in (M_i \vee A_i)^{\mathcal{I}}$. If $\delta \in M_i^{\mathcal{I}}$ for some $1 \leqslant i \leqslant n$, then clearly $\delta \in (\bigvee_{i=1}^{n} M_i)^{\mathcal{I}}$.

Otherwise, we have:

$$\delta \in (\bigwedge_{i=1}^{n} A_i)^{\mathcal{I}} \tag{7.1}$$

Meanwhile, $\mathcal{L}_1$ soundness for $\mathcal{O}$ entails $\mathcal{O} \models \mathsf{core}(v) \wedge K' \rightarrow M' \vee O(x)$, which implies $\delta \in (M' \vee O(x))^{\mathcal{I}}$. If $\delta \in (M')^{\mathcal{I}}$, then obviously $\delta \in (\bigvee_{i=1}^{n} M_i \vee M \vee M')^{\mathcal{I}}$, otherwise we have:

$$\delta \in O(x)^{\mathcal{I}}, \text{ which implies } \delta = o \tag{7.2}$$

Then again, considering $\bigwedge_{i=1}^{n} A_i \wedge K \rightarrow M \in \mathcal{L}(v_o)$, $\mathcal{L}_1$ soundness leads to $\mathcal{O} \models O(x) \wedge \bigwedge_{i=1}^{n} A_i \wedge K \rightarrow M$. Moreover, (7.2) implies $K \equiv K\sigma_o$; so based in the initial assumption, we have $\delta \in K^{\mathcal{I}}$; which alongside (7.1), (7.2) results in $\delta \in M^{\mathcal{I}}$. Accordingly, we proved that in either case, $\delta \in (\bigvee_{i=1}^{n} M_i \vee M \vee M')^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

Branch 2: Assume that the rule is applied as in Table 5.1; we will show that $\mathcal{O} \models \bigwedge_{i=1}^{n} A_i(o) \wedge K\sigma_o \rightarrow M\sigma_o$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (\bigwedge_{i=1}^{n} A_i(o) \wedge K\sigma_o \rightarrow M\sigma_o)^{\mathcal{I}}$. Now $\mathcal{L}_1$ is sound for $\mathcal{O}$ so, we have $\mathcal{O} \models O(x) \wedge \bigwedge_{i=1}^{n} A_i(x) \wedge K \rightarrow M$, which implies $\delta \in (O(x) \wedge \bigwedge_{i=1}^{n} A_i(x) \wedge K)^{\mathcal{I}}$. Following the nominals semantics, $\delta \in O(x)^{\mathcal{I}}$ implies that $\delta = o$, therefore we can readily conclude that $\delta \in (M\sigma_o)^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

(Sigma Rule) Assume the rule is applied as in Table 5.1. Each clause introduced by the rule is of the form $L \rightarrow L$, so clearly, $\mathcal{I} \models \mathsf{core}(v) \wedge L \rightarrow L$. We next show that the new edge $\langle u, v \rangle \in \mathcal{E}$, with $(R, n, \zeta) \in \mathcal{L}(\langle u, v \rangle)$ introduced by the rule satisfies the (5.7) clause in Definition 5.10. To simplify the notation, we denote $\mathsf{core}(u)$ and $\mathsf{core}(v)$ by unary

predicates $A(x)$ and $B(x)$, respectively. The select function (Definition 5.12) chooses $\prec_{max}$ unary predicate $B(x) \in \rho_k$. All the unary predicates in ρ_k (directly/indirectly[1]) originate from known clauses of the from $\top \rightarrow \exists_{\geq n} y(R(x,y) \wedge B(y)) \in \mathcal{L}(u)$. Accordingly, we have $\mathcal{O} \models \mathsf{core}(u) \rightarrow \exists_{\geq n} y(R(x,y) \wedge B(y))$ as $\mathcal{L}_1$ is sound for $\mathcal{O}$. Meanwhile, every unary predicate $C(x) \in \zeta$ originates from clauses of the from $K \rightarrow M \wedge \exists_{\bowtie n} y(R(x,y) \wedge C(y)) \in \mathcal{L}(u)$ or $C(x) \rightarrow F(x)$ both of which are included in $\mathcal{A}(u)$ so they are satisfied by the arithmetic solution $\xi(v)$. But then, we have $\mathcal{O} \models A(x) \wedge \exists_{\geq n} y(R(x,y) \wedge \bigwedge_{C \in \zeta} C(y)) \rightarrow \exists_{\geq n} y(R(x,y) \wedge B(y) \wedge \bigwedge C_{\in \zeta}(y))$ as required.

(Strict Rule) Assume that the rule is applied as in Table 5.1; we will show that $\mathcal{O} \models \mathsf{core}(v) \rightarrow D(o)$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (\mathsf{core}(v))^{\mathcal{I}}$. Now $\mathcal{L}_1$ is sound for $\mathcal{O}$ so, $\delta \in \mathcal{A}(v)$. Additionally, following Definition 5.9 $O(x) \wedge D(x) \rightarrow \bot$ is unsatisfiable; so based on Nominals' semantics, we conclude that $\delta \in (O(x) \rightarrow D(x))^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

(Bottom Rule) Assume that the rule is applied as in Table 5.1; we will show that $\mathcal{O} \models \mathsf{core}(v) \wedge \bigwedge_{i=0}^{m} K_i \wedge \bigwedge_{j=0}^{n} K_j\sigma_o \rightarrow \bigvee_{i=0}^{m} M_i$. To this end, let $\delta \in \Delta^{\mathcal{I}}$ be an arbitrary element such that $\delta \in (\mathsf{core}(v) \wedge \bigwedge_{i=0}^{m} K_i \wedge \bigwedge_{j=0}^{n} K_j\sigma_o)^{\mathcal{I}}$. Since $\mathcal{L}_1$ is sound for $\mathcal{O}$, for each $0 \leqslant j \leqslant n$ we also have $\mathcal{O} \models O(x) \wedge K_j \rightarrow M_j$ which implies $\delta \in (\bigwedge_{j=0}^{n} M_j\sigma_o)^{\mathcal{I}}$. Due to $\mathcal{L}_1$ soundness, for each $0 \leqslant i \leqslant n$, we have $\mathcal{O} \models \mathsf{core}(v) \wedge K_i \rightarrow M_i \vee Q_i$, which implies $\delta \in (M_i \vee Q_i)^{\mathcal{I}}$. If $\delta \in M_i^{\mathcal{I}}$ for some $0 \leqslant i \leqslant m$, then clearly $\delta \in (\bigvee_{m=0}^{n} M_i)^{\mathcal{I}}$. Otherwise, we have $\delta \in (\bigwedge_{i=0}^{m} Q_i)^{\mathcal{I}}$. But then Definition 5.9 implies that $(\bigwedge_{j=0}^{n} M_j\sigma_o \wedge \bigwedge_{i=0}^{m} Q_i)^{\mathcal{I}} \equiv \emptyset$, which is a conflict. Accordingly, $\delta \in (\bigvee_{i=0}^{n} M_i)^{\mathcal{I}}$ has to be correct. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

[1]Indirect entailment means that $\top \rightarrow \exists_{\geq n} y R(x,y) \wedge F(y) \in \mathcal{L}(u)$ while there is a known clause $F(x) \rightarrow C(x) \in \mathcal{A}(u)$

(Reach Rule) Assume that the rule is applied as in Table 5.1; we will show that $\mathcal{O} \models \mathsf{core}(u) \wedge \bigwedge_{i=0}^{n} K_i \wedge \bigwedge_{i=0}^{m} B_i \rightarrow \bigvee_{i=0}^{m} M_i \vee M$. To this end, consider an arbitrary element $\delta \in (\mathsf{core}(u) \wedge \bigwedge_{i=0}^{n} K_i \wedge \bigwedge_{i=0}^{m} B_i)^{\mathcal{I}}$. Now $\mathcal{L}_1$ is sound for $\mathcal{O}$, so for each $1 \leqslant i \leqslant n$, we have $\mathcal{O} \models \mathsf{core}(u) \wedge K_i \rightarrow M_i \vee \exists_{\bowtie n_i} y\ (R(x,y) \wedge A_i(y))$, which implies $\delta \in (M_i \vee \exists_{\bowtie n_i} y\ (R(x,y) \wedge A_i(y)))^{\mathcal{I}}$. If $\delta \in M_i^{\mathcal{I}}$ for some $0 \leqslant i \leqslant n$, then clearly $\delta \in (\bigvee_{i=0}^{n} M_i)^{\mathcal{I}}$. Otherwise, we have $\delta \in (\bigwedge_{i=0}^{n} \exists_{\bowtie n_i} y\ (R(x,y) \wedge A_i(y)))^{\mathcal{I}}$. Hence, an element $\gamma \in \Delta^{\mathcal{I}}$ exists such that $\langle \delta, \gamma \rangle \in R^{\mathcal{I}}$ and $\gamma \in (\bigwedge_{i=0}^{n} nA_i)^{\mathcal{I}}$. Additionally, since B_i are ground atoms for $1 \leqslant i \leqslant n$ then $\delta \in (\bigwedge_{i=1}^{n} B_i)^{\mathcal{I}}$ implies $\gamma \in (\bigwedge_{i=1}^{n} B_i)^{\mathcal{I}}$. Accordingly, as $\mathcal{L}_1$ is sound for $\mathcal{O}$, then $\mathcal{O} \models \bigwedge_{i=0}^{n} A_i \wedge \bigwedge_{i=1}^{m} B_i \rightarrow M$ and as $\mathcal{G}_1$ is sound we have $\gamma \in M^{\mathcal{I}}$. Once again, since M only contains ground atoms, then $\delta \in M^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

(Nom Rule) Assume the rule is applied as in Table 5.1. The clauses introduced in Nominal nodes are of the form $C(x) \rightarrow C(x)$, so clearly, $\mathcal{I} \models \mathsf{core}(v_o) \wedge C(x) \rightarrow C(x)$. We will show that $\mathcal{O} \models \mathsf{core}(v) \wedge K' \wedge K \rightarrow M' \vee M \vee C(o)$. To this end, consider an arbitrary element $\delta \in \Delta^{\mathcal{I}}$ such that $\delta \in (\mathsf{core}(v) \wedge K' \wedge K)^{\mathcal{I}}$. Now $\mathcal{L}_1$ is sound for $\mathcal{O}$ so, we have $\mathcal{O} \models \mathsf{core}(v) \wedge K \rightarrow M \vee C(x)$, which implies $\delta \in (M \vee C(x))^{\mathcal{I}}$. If $\delta \in M^{\mathcal{I}}$, then the rule's conclusion is readily satisfied, so let $\delta \in (C(x))^{\mathcal{I}}$. The $\mathcal{L}_1$ soundness for $\mathcal{O}$ also implies that $\mathcal{O} \models \mathsf{core}(v) \wedge K' \rightarrow M' \vee O(x)$, which entails $\delta \in (M' \vee O(x))^{\mathcal{I}}$. Once again, if $\delta \in (M')^{\mathcal{I}}$, then the rule's conclusion is readily satisfied, so we assume $\delta \in (O(x))^{\mathcal{I}}$. Since, $\delta \in (O(x))^{\mathcal{I}}$ and $\delta \in (C(x))^{\mathcal{I}}$, based on the nominal semantics we have $\delta \in (O(x) \rightarrow C(x))^{\mathcal{I}}$. Since δ was chosen arbitrarily, $\mathcal{I}$ satisfies the rule's conclusion, as required.

7.2 Proof of Completeness

Through the rest of this section, we assume $\mathcal{O}$ to be normalized $\mathcal{ALCHOQ}$ ontology, $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ to be a completion graph and $\mathcal{L}$ to be a complete clause system for $\mathcal{G}$ such that none of the inference rules from Table 5.1 applies to $\mathcal{L}$. Furthermore, let $\boldsymbol{L}$ contain the set of all literals occurring in $\mathcal{O}$, $\mathcal{G}$ or $\mathcal{L}$ and let Ω contain all ground atoms that occur in $\mathcal{G}$ or $\mathcal{L}$. We prove Theorem 5.2 by adapting the standard theorem-proving techniques from the first-order resolution and existing CB algorithms [49, 57, 16]. The idea is to prove that if a query clause $q = K \rightarrow M$, which satisfies all the conditions (listed below), is not derived in the completions graph $\mathcal{G}$; then it is refuted in the ontology $\mathcal{O}$. To this end, we prove that the following contra-positive claim: $\mathcal{O} \not\models K \rightarrow M$ holds for each query $K \rightarrow M$ and each node $v \in \mathcal{V}$, where $\mathsf{core}(v) \equiv K$ such that:

- For each atom $A(x) \in K$ and each atom $A'(x)$ of the form $B(x)$ such that $A'(x) \prec A(x)$, we have $A'(x) \in K$.
- $K \rightarrow L \in_* \mathcal{L}(v)$ for each $L \in K$, and
- $K \rightarrow M \notin_* \mathcal{L}(v)$

We prove the above claim by constructing a model $\mathcal{I}$ of $\mathcal{O}$ that refutes each random query stratifying the above conditions. For simplicity, instead of directly constructing a model, we first create a *pre-model*. Our pre-model is a directed graph structure in that its vertices represent domain elements labelled with sets of literals, and its edges express role-successor relations between them. In the following, we first introduce the notion of a pre-model and explain how to convert it into a model in Section 7.2.1. Afterward, we show how to construct a propositional interpretation in Section 7.2.2 and discuss how

our inference rules allow us to use this construction in Section 7.2.3. Finally, we use these propositional interpretations to build our pre-model of $\mathcal{O}$ based on a complete completion graph $\mathcal{G}$ in Section 7.2.4.

7.2.1 Pre-interpretations and pre-models

This section introduces the notion of pre-interpretations and pre-models of an ontology. We use these notions to present the subsequent proofs.

Definition 7.1. A set of literals J satisfies a clause $K \rightarrow M$, shown as $J \models K \rightarrow M$ if $K \subseteq J$ implies $M \cap J \neq \emptyset$; otherwise J refutes $K \rightarrow M$, written as $J \not\models K \rightarrow M$.

Definition 7.2. A pre-interpretation is a labelled graph $I = (V, E, J)$ where V is a non-empty set of nodes and $E \subseteq V \times V$ is a set of edges. Every node $a \in V$ is labelled with a set of literals $J(a) \subseteq S_L$, and each edge $\langle a, b \rangle \in E$ is labelled with a set of roles $J(\langle a, b \rangle) \subseteq S_{\mathcal{B}}$. The pre-interpretation I satisfies the following three conditions, where $\varphi(a, b) = (R(a, b) \wedge C(b))$.

($I_{\geq}$) For each $a \in V$ and each literal $\exists_{\geq n} b \ \varphi(a, b)$, there are at least n edges $\langle a, b \rangle \in E$ such that $R \in J(\langle a, b \rangle)$ and $C \in J(b)$.

($I_{\leq}$) For each $a \in V$ and each literal $\exists_{\leq n} b \ \varphi(a, b)$, there are at most n edges $\langle a, b \rangle \in E$ such that $R \in J(\langle a, b \rangle)$ and $C \in J(b)$.

(I_o) There is exactly one node $a \in V$ such that $o \in J(a)$.

Pre-interpretation $I = (V, E, J)$ satisfies a clause $K \to M$ (written as $I \models K \to M$) if $J(a) \models K \to M$ holds for each node $a \in V$; otherwise I refutes $K \to M$ (written as $I \not\models K \to M$). Moreover, I is called a pre-model of $\mathcal{O}$ if all (normal) clauses in $\mathcal{O}$ are satisfied by I.

Given a pre-interpretation $I = (V, E, J)$, we can construct an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the non-empty domain and $\cdot^{\mathcal{I}}$ is the interpretation function. The construction of $\mathcal{I}$ for all atomic concepts $C \in S_{\mathcal{U}}$ and all atomic role $R \in S_{\mathcal{B}}$ is performed by the following equivalences:

$$\Delta^{\mathcal{I}} := V \tag{7.3}$$

$$C^{\mathcal{I}} := \{a \mid C \in J(x)\} \tag{7.4}$$

$$R^{\mathcal{I}} := \{\langle a, b\rangle \mid \langle a, b\rangle \in E \text{ and } R \in J(\langle a, b\rangle)\} \tag{7.5}$$

Interpretation $\mathcal{I}$ satisfies the following property for each literal $L \in M$ and each node $a \in V$:

$$L \in J(a) \text{ implies } a \in L^{\mathcal{I}} \tag{7.6}$$

The property (7.6) holds for atomic concepts and nominals by the definition of $C^{\mathcal{I}}$ and the condition (I_o). Moreover, this property holds for literals of the form $\exists_{\geq n} b\ \varphi(a, b)$ and $\exists_{\leq n} b\ \varphi(a, b)$ due to the conditions $(I_{\geq})$ and $(I_{\leq})$ respectively. However, the converse of the property (7.6) holds only for atomic concepts and nominals. That is why all normal clauses in $\mathcal{O}$ must have only atomic concepts and nominals in their antecedents, and

queries must have only atomic concepts and nominals in their consequences. Next, we show that the obtained interpretation $\mathcal{I}$ is a model of ontology $\mathcal{O}$.

Lemma 7.1. Assume that $\mathcal{I}$ is an interpretation obtained from pre-interpretation I using equations (7.3) - (7.5). If a clause α is satisfied in I (written as $I \models \alpha$), then it is satisfied in $\mathcal{I}$ as well ($\mathcal{I} \models \alpha$). Moreover, if pre-interpretation I refutes a query q (written as $I \not\models q$) then the interpretation $\mathcal{I}$ refutes it as well ($\mathcal{I} \not\models q$).

Proof. Assume an arbitrary normal clause α of the form $\bigwedge_{i=1}^{m} C_i \rightarrow \bigvee_{j=1}^{n} L_j$ that is satisfied in I, where $C_i(x)$ is an atomic concept or nominal. Also, let $a \in V$ be an arbitrary node such that $a \in C_i^{\mathcal{I}}$ holds for each $1 \leqslant i \leqslant m$. Based on (7.4), we have $C_i \in J(a)$ for each $1 \leqslant i \leqslant m$. But then, following our assumption, pre-interpretation I satisfies α, so $L_j \in J(a)$ holds for some $1 \leqslant j \leqslant n$. Therefore, by (7.6), we have $a \in L_j^{\mathcal{I}}$ for some $1 \leqslant i \leqslant n$. Since we chose a arbitrary, the interpretation $\mathcal{I}$ satisfies α.

For the second part, assume an arbitrary query q of the form $\bigwedge_{i=1}^{m} L_i \rightarrow \bigvee_{j=1}^{n} C_j(x)$ that is refuted in I, where C_j is an atomic concept or nominals. Also, let $a \in V$ be in an arbitrary node such that $L_i \in J(a)$ for each $1 \leqslant i \leqslant m$, and $C_j \notin J(a)$ for each $1 \leqslant j \leqslant n$. But then, $a \in L_i^{\mathcal{I}}$ holds for each $1 \leqslant i \leqslant m$ by (7.6), and we have $a \notin C_j^{\mathcal{I}}$ for each $1 \leqslant j \leqslant n$ by (7.4); therefore, the interpretation $\mathcal{I}$ refutes q. □

Corollary 7.1. If there exists a pre-model I of an ontology $\mathcal{O}$ that refutes a query $q = K \rightarrow M$, then $\mathcal{O}$ refutes the query as well, $I \not\models q \implies \mathcal{O} \not\models q$.

7.2.2 Construction of literal interpretations

This section shows how we can use the sets of clauses $\mathcal{L}(v)$ to construct the sets $J(a)$ of a pre-model $I = (V, E, J)$ that satisfies all clause in ontology $\mathcal{O}$ but refutes a query $K_q \rightarrow M_q$. Since our construction is independent of the selected node v; we only consider a single set of clauses N_t in the first part of this section. We apply the construction on each node $v_t \in \mathcal{V}$ for $1 \leqslant t \leqslant \eta$ based on the following ordering:

Let $C_1, C_2, \ldots, C_\eta$ be an arbitrary total ordering over $S_\mathcal{U} \cup S_O$ that extends $\prec$ such that $C_l(x) \equiv \mathsf{core}(v_l)$, for all $1 \leqslant l, m \leqslant \eta$ we have that:

$$C_l \prec C_m \text{ implies } l < m, \text{ and} \tag{7.7}$$

$$C_l \prec C_m \text{ implies } v_l \prec v_m \tag{7.8}$$

$$C_l \in K_q \text{ implies } C_l \prec C_m \tag{7.9}$$

$$C_l \in S_\mathcal{U} \text{ and } C_m \in S_O \text{ implies } C_l \not\prec C_m \tag{7.10}$$

For the construction, we always select the next minimal node $v_t \in \mathcal{V}, 1 \leqslant t \leqslant \eta$, so we only consider a single set of clauses $\mathcal{N}_t$ at any time. Throughout the appendix, allow $\prec$ to be a literal ordering, $\mathcal{N}_t$ to be a set of clauses over $\boldsymbol{L} \cup \Omega$ and $K \rightarrow M$ to be a query clause over $\boldsymbol{L} \cup \Omega$ such that:

$$\begin{aligned}&\text{For each atom } A(x) \in M_q \text{ and each atom } A'(x)\\&\text{such that } A'(x) \prec A(x), \text{ we have } A'(x) \in M.\end{aligned} \tag{7.11}$$

$$K \to L \in_* \mathcal{N}_t \text{ for each literal } L \in K, \text{ and} \tag{7.12}$$

$$K \to M \notin_* \mathcal{N}_t \tag{7.13}$$

The following definition introduces a notion to determine whether set $\mathcal{N}_t$ contains "sufficiently many" hyperresolution consequences w.r.t literal ordering $\prec$:

Definition 7.3. Set $\mathcal{N}_t$ is closed under hyperresolution with a clause $\bigwedge_{i=1}^{n} L_i \to M'$ (*not necessarily contained in* $\mathcal{N}_t$) if, for each set of n clauses $K_i \to M_i\ L_i \in \mathcal{N}_t$, with $1 \leqslant i \leqslant n$ and $L_i \not\prec M_i$, we have $\bigwedge_{i=1}^{n} K_i \to M' \vee \bigvee_{i=1}^{n} M_i \in_* \mathcal{N}_t$.

Lemma 7.2. Allow α_1 and α_2 to be arbitrary clauses such that α_1 is a strengthening of α_2, then if $\mathcal{N}_t$ is closed under hyperresolution with α_1, it would be closed under hyperresolution with α_2 as well.

Proof. Let $\alpha_1 = \bigwedge_{i=1}^{n} L_i \to M_1'$, then following the notion of strengthening $\alpha_2 = \bigwedge_{i=1}^{m} L_i \to M_2'$, where $n \leq m$ and $M_1' \subseteq M_2'$. We need to prove that for each set of m clauses $K_i \to M_i \vee L_i \in \mathcal{N}_t$, with $1 \leqslant i \leqslant m$ and $L_i \not\prec M_i$, there exists a strengthening of clause $\alpha = \bigwedge_{i=1}^{m} K_i \to M_2' \vee \bigvee_{i=1}^{m} M_i$ in $\mathcal{N}_t$. But then, as $\mathcal{N}_t$ is closed under hyperresolution with α_1, we have $\bigwedge_{i=1}^{n} K_i \to M_1' \vee \bigvee_{i=1}^{n} M_i \in_* \mathcal{N}_t$ which is a strengthening of α. □

Next, we show how to construct a set of literals J that refutes $K \rightarrow M$ but satisfies every clause $\alpha = \bigwedge_{i=1}^{n} L_i \rightarrow M'$, such that $\mathcal{N}_t$ is closed under hyperresolution with α. To this end, we essentially adopt the standard resolution theorem-proving technique [49] followed by [57]. Let $L_1, L_2, \ldots, L_\ell$ be an arbitrary total ordering of $\boldsymbol{L}$ that extends $\prec$ such that the elements of M precede all the remaining literals from $\boldsymbol{L}$. Accordingly, for all i and j we have:

$$L_i \prec L_i \text{ implies } i < j, \text{ and} \tag{7.14}$$

$$i < j \text{ and } L_j \in M \text{ implies } L_j \in M \tag{7.15}$$

Since M is $\prec$-min by (7.11) and $\boldsymbol{L}$ is finite, there is at least one such total ordering. For each $0 < k \leq \ell$, let $\boldsymbol{L}_k := \{L_1, ..., L_k\}$, $J_0 := \emptyset$, then for each $0 < k \leq \ell$ we inductively define J_k as follows:

$$\begin{cases} J_{k-1} \cup \{L_k\} & \text{if a clause } K' \rightarrow M' \vee L_k \in \mathcal{N}_t \\ & \text{exists such that } K' \subseteq K, \\ & M' \cap J_{k-1} = \emptyset, \text{ and} \\ & M' \subseteq \boldsymbol{L}_{k-1}, \\ \\ J_{k-1} & \text{otherwise.} \end{cases} \tag{7.16}$$

Assume $J := J_\ell \setminus \alpha$. Each set J that is obtained by such a construction is called a *literal interpretation* of $\boldsymbol{L}$ w.r.t. $\mathcal{N}_t$ and $\prec$ refuting $K \rightarrow M$. Each clause $K' \rightarrow M' \vee L_k \in \mathcal{N}_t$ that

satisfies the first condition in (7.16) for some k is called a *productive clause*, which *produces* L_k in J. Obviously, every literal in J has to be produced by at least one productive clause in $\mathcal{N}_t$.

We need to keep track of produced ground atoms to build a pre-model. To this end, let $\mathbf{g}$ be defined as follows:

$$\begin{cases} J_\ell\{x \mapsto o\} \cap \alpha & \text{if } \mathsf{core}(v_t) = \pi(o) \in S_O \\ J_\ell \cap \alpha & \text{otherwise.} \end{cases} \tag{7.17}$$

Then we define $\mathbf{g}_{v_t}$ as the set of all ground atoms generated w.r.t. $\mathcal{N}_t = \mathcal{L}(v_t)$ for constructing each J. We define the set of *global ground atom* as $\mathbf{G}_t = \{\mathbf{g}_{v_1} \cup \cdots \cup \mathbf{g}_{v_t}\}$ for every $1 \leqslant t \leqslant \eta$.

Throughout the rest, we proceed as follows: in Lemma 7.3, we prove specific properties of productive clauses in $\mathcal{N}_t$; in Lemma 7.4 we show that J satisfies every clause $K' \to M'$ for which a strengthening exist in $\mathcal{N}_t$ and that satisfies $K' \subseteq K$; in Lemma 7.5 we show that J refutes $K \to M$; and finally in Lemma 7.6 we prove that if $\mathcal{N}_t$ is closed under $\prec$-hyperresolution with clause α, then $J \models \alpha$, which is the desired result.

Lemma 7.3. Each productive clause $K' \to M' \vee L_k \in \mathcal{N}_t$ satisfies $K' \subseteq K$, $M' \cap J = \emptyset$, and $L_k \not\prec M'$.

Proof. Assume $K' \to M' \vee L_k \in \mathcal{N}_t$ is an arbitrary clause that produces literal L_k in J for some integer k, so it is a productive clause and satisfies the first condition in (7.16). Accordingly, we have $K' \subseteq K$, $M' \cap J_{k-1} = \emptyset$, and $M' \subseteq \boldsymbol{L}_{k-1}$. Following the inductive definition of J we have $J_{k-1} = J \cap \boldsymbol{L}_{k-1}$, so $M' \cap J \cap \boldsymbol{L}_{k-1} = \emptyset$, which can be simplified as

$M' \cap J = \emptyset$. Finally, due to 7.14, $L_k \not\prec L$ holds for each $L \in \boldsymbol{L}_k$. Therefore, since $M' \subseteq \boldsymbol{L}_{k-1}$, we have $L_k \not\prec M'$. □

Lemma 7.4. Each non-ground clause $K' \rightarrow M'$ such that $K' \rightarrow M' \in_* \mathcal{N}_t$ and $K' \subseteq K$ implies $M' \cap J \neq \emptyset$

Proof. Assume that $K' \rightarrow M'$ is an arbitrary clause such that $K' \rightarrow M' \in_* \mathcal{N}_t$ and $K' \subseteq K$. So, $\mathcal{N}_t$ contains a strengthening clause $K_1 \rightarrow M_1$ of $K' \rightarrow M'$, for which $K_1 \subseteq K'$ and $M_1 \subseteq M'$ hold. If $M_1 = \emptyset$, then clearly, the clause $K_1 \rightarrow \bot$ is a strengthening of $K \rightarrow M$. But it is in contradiction with the assumption (7.13) $K \rightarrow M \notin_* \mathcal{N}_t$, so $M_1 \neq \emptyset$. Therefore, $M_1 \subseteq M'$ should contain at least one literal L_k, let k be the largest integer such that $L_k \in M'$. So, $K_1 \rightarrow M_1$ has to be of the form $K_1 \rightarrow M_2 \vee L_k$ where $M_2 \subseteq \boldsymbol{L}_{k-1}$. If $M_2 \cap J \neq \emptyset$, then clearly, $M' \cap J \neq \emptyset$ and the lemma is proved. Likewise, if $M_2 \cap J = \emptyset$, then $L_k \in J$ would be produced by clause $K_1 \rightarrow M_2 \vee L_k$ based on (7.16), which again implies that $M' \cap J \neq \emptyset$, as required. □

Lemma 7.5. J refutes $K \rightarrow M$, $J \not\models K \rightarrow M$, which means that $K \subseteq J$ and $M \cap J = \emptyset$.

Proof. To prove $K \subseteq J$, assume to have an arbitrary literal $L \in K$. Then, by (7.12) we have $K \rightarrow L \in_* \mathcal{N}_t$. So, a conjunction $K' \subseteq K$ exists for which either $K' \rightarrow \bot$ or $K' \rightarrow L \in \mathcal{N}_t$. The former is a strengthening of $K \rightarrow M$, so it contradicts the assumption $K \rightarrow M \notin_* \mathcal{N}_t$ from (7.13). Thus, $K' \rightarrow L \in \mathcal{N}_t$ should hold, which produces $L \in J$ by (7.16) for $M' = \emptyset$.

We prove $M \cap J = \emptyset$ by showing that the contra-positive claim $M \cap J \neq \emptyset$ causes a contradiction. To this end, assume a literal $L_k \in M \cap J$ exists. So, there exists a clause $K' \to M' \vee L_k \in \mathcal{N}_t$ that produces $L_k \in J$. Thus, it is a productive clause that satisfies the first condition of (7.16) as $M' \subseteq \boldsymbol{L}_{k-1}$ and $K' \subseteq K$. Additionally, since $L_k \in M$, by (7.15) we have $\boldsymbol{L}_{k-1} \subseteq M$; thus $M' \cup \{L_k\} \subseteq M$, which implies that, $K' \to M' \vee L_k$ is a strengthening of $K \to M$. But then, $K' \to M' \vee L_k \in \mathcal{N}_t$ contradicts the assumption $K \to M \not\in_* \mathcal{N}_t$ from (7.13). □

Lemma 7.6. For each clause α for which $\mathcal{N}$ is closed under $\prec$-hyperresolution with α, we have $J \models \alpha$.

Proof. Let $\alpha = \bigwedge_{i=1}^{n} L_i \to M'$. Assume that there exist $L_i \in J$ and a productive clause $K_i \to M_i \vee L_i$ that produces $L_i \in J$ for each $1 \leqslant i \leqslant n$. By Lemma 7.3, we have $K_i \subseteq K$, $M_i \wedge J = \emptyset$, and $L_i \not\prec M_i$. Since $\mathcal{N}_t$ is closed under hyperresolution with α, a strengthening of clause $\bigwedge_{i=1}^{n} K_i \to M' \vee \bigvee_{i=1}^{n} M_i$ exists in $\mathcal{N}_t$. Furthermore, by Lemma 7.4 and considering that $(\bigwedge_{i=1}^{n} K_i) \subseteq K$, we have $(M' \vee \bigvee_{i=1}^{n} M_i) \cap J \neq \emptyset$. Finally, since $M_i \cap J = \emptyset$ holds for each $1 \leqslant i \leqslant n$, we have $M' \cap J \neq \emptyset$. Consequently, as required, we proved that $J \models \alpha$, as required. □

7.2.3 Properties of the CB inference rules

The following two auxiliary lemmas allow us to apply the construction discussed in (7.14) – (7.16) to obtain J from the clause system $\mathcal{L}$ and the completion graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$.

Lemma 7.7 shows that applying the above construction to $\mathcal{L}(v)$ produces a set of literals that satisfies each clause in $\mathcal{O}$. Likewise, Lemma 7.8 states a similar property about the clauses that participate in the `Bottom` rule.

Lemma 7.7. For each $v \in \mathcal{V}$, set $\mathcal{L}(v)$ is closed under hyperresolution with each clause $\alpha \in \mathcal{O}$.

Proof. Lemma 7.7 follows trivially from Definition 7.3 and the fact that the `Subs` rule is no more applicable to $\mathcal{L}(v)$. □

Lemma 7.8. For arbitrary clauses $\alpha_i = K_i \rightarrow M_i \vee Q_i \in \mathcal{L}(v)$ for $0 \leqslant i \leqslant m$ and $\psi_j = K_j \rightarrow M_j \in \mathcal{L}(v_o)$ for $0 \leqslant j \leqslant n$, where Q_i is a disjunction of CQs and M_j contains only unary atoms, such that $Q_i \in \mathcal{CS}(v)$ and $O(x) \rightarrow M_j \in \mathcal{CS}(v)$, set $\mathcal{L}(u)$ is closed under hyperresolution with $\bigwedge_{i=0}^{n} M_j\sigma_o \wedge \bigwedge_{i=0}^{m} Q_i \rightarrow \bot$.

Proof. Consider arbitrary clauses $\alpha_i = K_i \rightarrow M_i \vee Q_i \in \mathcal{L}(v)$ for $0 \leqslant i \leqslant m$ and $\psi_j = K_j \rightarrow M_j \in \mathcal{L}(v_o)$ for $0 \leqslant j \leqslant n$. By definition of clause strengthening, we have subsets $\{K'_i\}_{i=0}^{m'} \subseteq \{K_i\}_{i=0}^{m}$ and $\{M'_i\}_{i=0}^{m'} \subseteq \{M_i\}_{i=0}^{m}$ such that $Q'_i \in \mathcal{CS}(v)$. Likewise, following the clause strengthening, there exist $\{K'_j\}_{j=0}^{n'} \subseteq \{K_j\}_{j=0}^{n}$ and $\{M'_j\}_{j=0}^{n'} \subseteq \{M_j\}_{j=0}^{n}$ such that $O'(x) \rightarrow M'_j \in \mathcal{CS}(v)$. Since the Bottom rule is not applicable to the clause system $\mathcal{L}$; we have $\bigwedge_{i=0}^{m'} K'_i \wedge \bigwedge_{j=0}^{n'} K'_j\sigma_o \rightarrow \bigvee_{i=0}^{m'} M'_i$ in $\mathcal{L}(v)$. So $\mathcal{L}(v)$ is closed under hyperresolution with clause $\bigwedge_{i=0}^{n'} M'_j\sigma_o \wedge \bigwedge_{i=0}^{m'} Q'_i \rightarrow \bot$. According to Lemma 7.2, $\mathcal{L}(u)$ is also close under hyperresolution with $\bigwedge_{i=0}^{n} M_j\sigma_o \wedge \bigwedge_{i=0}^{m} Q_i \rightarrow \bot$ because the former is a strengthening of the latter clause. □

Claim 7.1. Considering each node $v \in \mathcal{V}$ and each clause $K \rightarrow M$, if $K \rightarrow L \in_* \mathcal{L}(v)$ for each $L \in K$ and $K \rightarrow M \notin_* \mathcal{L}(v)$, then $K \subseteq \boldsymbol{L}$.

Proof. We prove this lemma by showing that the contra-positive claim $K \nsubseteq \boldsymbol{L}$ can not hold. So there is at least one literal $L \in K$ while $L \notin \boldsymbol{L}$; thus L does not occur in $\mathcal{L}(v)$. Accordingly, $K \rightarrow L \in_* \mathcal{L}(v)$ implies $K \rightarrow \bot \in \mathcal{L}(v)$ which contradicts the assumption that $K \rightarrow M \notin_* \mathcal{L}(v)$. Therefore, $K \subseteq \boldsymbol{L}$ should hold. □

7.2.4 Constructing a pre-model

Let $\mathbf{G}_0 = \emptyset$, we define a set of clauses for each node $v_t \in \mathcal{V}$, where $1 \leqslant t \leqslant \eta$ which will be used for constructing the pre-model:

If $\mathsf{core}(v_t) \equiv O(x) \in S_O$, we define a clause $K_t \rightarrow M_t$ such that:

$$\begin{cases} K_t = K_q \text{ and } M_t = M_q & t = 1 \\ \\ K_t = O(x) \cup \mathbf{G}_{t-1}\{o \mapsto x\} \text{ and} & \\ M_t = \{C(x) \in S_{\mathcal{U}} \cup S_O\} & \text{otherwise} \\ \text{such that } C(x) \prec O(x) \text{ and } C(o) \in \mathbf{G}_{t-1} & \end{cases} \tag{7.18}$$

If $\mathsf{core}(v_t) \equiv C(x) \in S_{\mathcal{U}}$:

$$\begin{cases} \text{define a clause } K_t \to M_t & t = 1 \\ \text{such that } K_t = K_q \text{ and } M_t = M_q & \\ & \\ \text{define a clause } K_t^\varepsilon \to M_t^\varepsilon & \text{for each incoming} \\ \text{such that } K_t^\varepsilon = \zeta \cup \mathbf{G}_{t-1} & \text{edge } \varepsilon = \langle w, v_t \rangle \in \mathcal{E} \\ M_t^\varepsilon = \{\mathcal{DS}(v_t) \setminus \zeta\} \cup \{O(x) \in S_O \mid O(x) \prec C(x)\} & (R, n, \zeta) \in \mathcal{L}(\varepsilon) \end{cases} \tag{7.19}$$

Theorem 5.2 holds trivially if there is no node $v \in \mathcal{V}$ and no query $K_q \to M_q$ such that $v \nvdash K_q \to M_q$. Thus, we assume that there exists at least one node $v \in \mathcal{V}$ and at least one query $K_q \to M_q$ that satisfies this condition.

Let's define $I = (V, E, J)$ as follows:

- Set V is the smallest set that contains a distinguished element $b(v_t)$ for each node $v_t \in V$ and each clause $K_t \to M_t$ such that $v_t \nvdash K_t \to M_t$ and also n distinguished elements $\beta_i^\varepsilon(v_t)$ for each node $v_t \in \mathcal{V}$, each edge $\langle u, v \rangle \in \mathcal{E}$, with $(R, n, \zeta) \in \mathcal{L}(\langle u, v \rangle)$ and each clause $K_t^\varepsilon \to M_t^\varepsilon$ such that $v_t \nvdash K_t^\varepsilon \to M_t^\varepsilon$.

- For each $b(v_t) \in V$, let $J(b(v_t))$ be an arbitrary literal interpretation over $\boldsymbol{L}$ w.r.t. $\mathcal{L}(v_t)$ and $\mathbf{G}$ that refutes $K_t \to M_t$ and for each $\beta_i^\varepsilon(v_t) \in V$, let $J(\beta_i^\varepsilon(v_t))$ be an arbitrary literal interpretation over $\boldsymbol{L}$ w.r.t. $\mathcal{L}(v_t)$ and $\mathbf{G}$ that refutes $K_t^\varepsilon \to M_t^\varepsilon$.

- Set E is the smallest set that contains n edges $\langle b(u), b_i^\varepsilon(v_t), R \rangle$ for each element $b(u) \in V$ and each incoming edge $\varepsilon = \langle u, v_t \rangle \in \mathcal{E}$ with $(R, n, \zeta) \in \mathcal{L}(\varepsilon)$ such that $v_t \nvdash K_t^\varepsilon \to$

M_t^ε, where k_t^ε and M_t^ε are specified below.

$$\bigwedge K_t^\varepsilon = \zeta \cup \mathbf{G}_{t-1} \tag{7.20}$$

$$\bigvee M_t^\varepsilon = \{\mathcal{DS}(v_t) - \rho\} \tag{7.21}$$

Lemma 7.9. If $S_O \neq \emptyset$, there is at least one element in V.

Proof. The literal ordering among nominals indicates at least one nominal $O_m(x) \not\preceq S_O$, for which $M_{o_m} \equiv \bot$. Based on the construction introduced in 7.2.4, $K_{o_m} \rightarrow M_{o_m}$ has to be refuted. Otherwise, the ontology is inconsistent. Based on the definition of V there exists an element $b(v_{o_m}) \in V$ w.r.t. v_{o_m}. □

Claim 7.2. The set E is correctly defined – that is, for each element $b(u) \in V$ and for each edge $\varepsilon = \langle w, v_t \rangle \in \mathcal{E}$ with $(R, n, \zeta) \in \mathcal{L}(\varepsilon)$, there are n elements $b_i^\varepsilon(v_t)$ such that $J(b_i^\varepsilon(v_t) \not\models K_t^\varepsilon \rightarrow M_t^\varepsilon$, where k_t^ε and M_t^ε are specified above.

Proof. Following the definition of V, we create n elements $b_i^\varepsilon(v_t) \in V$ corresponding to each incoming edge ε of v_t, unless $\bigwedge \zeta \cup \mathbf{G}_{t-1} \rightarrow \bigvee\{O(x) \in S_O \mid O(x) \prec C(x)\} \in_* \mathcal{L}(v_t)$, which by Definition 5.6 should be included in $\mathcal{A}(v_t)$, so it is satisfied by the solution returned by ARM. If this set is empty, the claim is proven trivially, so let's assume there is a nominal in the set, so the nominal has to be part of the partition. Constraint (6.7) in MP ensures that $\varepsilon : n = 1$.

Based on the ordering Definition 5.11, there is already an entity $b(v)$ in the graph such that $O(x) \in J(b(v))$. Since the Reach rule and Nom rule are not applicable, $\zeta \subseteq J(b(v))$ and $J(b(v)) \cap (\mathcal{DS}(v) \setminus \zeta) = \emptyset$. So, we have proved that E is correctly defined. □

Claim 7.3. For each nominal $O(x) \in S_O$, there is at least one $\gamma \in V$ such that $O(x) \in J(\gamma)$.

Proof. Following the initialization step, there is a node $v_o \in \mathcal{G}$, for each nominal $O(x) \in S_O$, such that $\mathsf{core}(v_o) = O(x)$. Based on the construction defined in 7.2.4:

- If we have $v_o \nvdash K_t \rightarrow M_t$, then an entity $b^{k_t \rightarrow M_t}(v_o)$ is created in V. Following the initialization step, there is a productive clause $\top \rightarrow O(x) \in \mathcal{L}(v_o)$. Therefore, $O(x) \in J(b^{k_t \rightarrow M_t}(v_o))$, as required.

- If $v_o \vdash K_t \rightarrow M_t$, then there is a clause $K' \rightarrow M' \in \mathcal{L}(v_o)$ such that $M' \subseteq \{C(x) \in S_\mathcal{U} \cup S_O \mid C(x) \prec O(x)$ and $C(o) \in \mathbf{G}_{t-1}\}$.

  According to Definition 5.11, one of the two following cases is possible:

  - $C(x) \equiv K_q$: Based on the construction discussed in Section 7.2.4, there always exists an entity $b^{K_q \rightarrow M_q}(v)$ that refutes the query clause $K_q \rightarrow M_q$. Following the Definition 5.11, v_c where $\mathsf{core}(v_c) = C(x)$ is the first node in the model construction procedure, therefore $C(o) \in \boldsymbol{G}_{t-1}$ is produced by the Nom rule. Accordingly, we have a productive clause $K' \rightarrow M' \vee O(x) \in \mathcal{L}(v_c)$, that will add $O(x)$ to $J(b^{K_q \rightarrow M_q}(v))$, as required.

– $C(x) \equiv O'(x) \in S_O$: Following the assumption $O'(x) \prec O(x)$ and $O'(o) \in \mathbf{G}_{t-1}$, an entity $b(v_{o'})$. By the assumption, there exists a clause $K' \to O'(x) \in \mathcal{L}(v_o)$, since the Glob rule is not applicable, we also have a productive clause $K \to O(x) \in \mathcal{L}(v_{o'})$, which produces $O(x) \in J(b(v_{o'}))$, as required.

□

Claim 7.4. For each $O(x) \in S_O$ there is at most one $\gamma \in V$ such that $O(x) \in J(\gamma)$.

Proof. Assume there are two entities γ_1 and γ_2 such that $O(x) \in J(\gamma_1)$ and $O(x) \in J(\gamma_2)$. Let $K_1 \to M_1 \vee O(x) \in \mathcal{L}(v_1)$ and $K_2 \to M_2 \vee O(x) \in \mathcal{L}(v_2)$ be arbitrary productive clauses generating $O(x)$ in $J(\gamma_1)$ and $J(\gamma_2)$, respectively. By Lemma 7.3, we have $O(x) \not\prec M_1$ and $O(x) \not\prec M_2$. To simplify the proofs annotation, let $\mathsf{core}(v_1) = C_1(x)$ and $\mathsf{core}(v_2) = C_2(x)$, where $\{C_1(x), C_2(x)\} \in S_{\mathcal{U}} \cup S_O$. Without loss of generality, we assume that $C_1(x) \prec C_2(x)$.

The initialization step has produced $\top \to C_1(x) \in \mathcal{L}(v_1)$. Since Nom rule is not applicable, we have $K_1 \to C_1(o) \vee M_1$ which creates $C_1(o) \in \boldsymbol{G}$. Also $C_1(x) \to C_1(x) \in \mathcal{L}(v_o)$. On the other hand, the initialization step also has produced $\top \to C_2(x) \in \mathcal{L}(v_2)$. Since Glob rule is not applicable, we have $K_2 \wedge C_1(o) \to C_1(x) \vee M_2 \in \mathcal{L}(v_2)$. Following Lemma 7.3 and (7.18), we have $K_2 \wedge C_1(o) \subseteq K_t(v_2)$. In addition, based on the ordering defined in (7.7) - (7.10) and Lemma 7.3 the node clause definition (7.18) implies that

$C_1(x) \vee M_2 \subseteq M_t(v_2)$. We proved that $K_t \to M_t$ is not refuted in v_2, so there can not be an entity γ_2 corresponding to v_2 in the graph, which contradicts our initial assumption. □

Claim 7.5. Structure I satisfies conditions $(I_{\leq})$, $(I_{\geq})$ and (I_o) of pre-interpretation from Definition 7.2.

Proof. (Property $I_{\geq}$) Consider an arbitrary element $b^{\varepsilon}(u) \in V$ and an arbitrary literal $\exists_{\geq n}\, y\, (R(x,y) \wedge C(y)) \in J(b^{\varepsilon}(u))$; We next show that there are at least n edges $\langle b^{\varepsilon}(u), \gamma_i, R\rangle$ such that $1 \leqslant i \leqslant n$ and $C \in J(\gamma_i)$. The Sigma rule does not apply to $\mathcal{L}(u)$ so there are m edges $\varepsilon = \langle u, v_j\rangle \in \mathcal{E}$ with $(R_j, n_j, \zeta_j) \in \mathcal{L}(\varepsilon)$ that satisfy the following conditions for $1 \leqslant j \leqslant m$:

$$\sum_{j=1}^{m} n_j \geqslant n, C(x) \in \zeta_j \text{ and,} \tag{7.22}$$

$$L \to L \in_* \mathcal{L}(v_j) \text{ for each } L \in \zeta_j \tag{7.23}$$

□

Let $K^{\varepsilon}_{v_j}$ and $M^{\varepsilon}_{v_j}$ be as specified in (7.20) and (7.21). The following observation shows that $v_j \nvdash K^{\varepsilon}_{v_j} \to M^{\varepsilon}_{v_j}$.

- Consider an arbitrary literal $L \in K^{\varepsilon}_{v_j}$; we show that $K^{\varepsilon}_{v_j} \to L \in \mathcal{L}(u_j)$. According to (7.20), we have $L \in \zeta$. But then $L \to L \in_* \mathcal{L}(u_j)$ holds by (7.23), which is stronger than the required $K^{\varepsilon}_{v_j} \to L \in \mathcal{L}(u_j)$.

- Assume for contradiction that $K^{\varepsilon}_{v_j} \rightarrow M^{\varepsilon}_{v_j} \in_* \mathcal{L}(v_j)$. According to (5.4) in Definition 6.1 and Definition 5.6, we have clause $\alpha = K^{\varepsilon}_{v_j} \rightarrow M^{\varepsilon}_{v_j} \in_* \mathcal{C}(u)$. Following Lemma 6.1, the found solution satisfies this clause, which clearly implies that α is satisfied in $J(\beta^{\epsilon}_i(v_j))$, however by the definition of V in Section 7.2.4 this clause is refuted in $J(\beta^{\epsilon}_i(v_j))$, which is a contradiction. Consequently we have $K^{\varepsilon}_{v_j} \rightarrow M^{\varepsilon}_{v_j} \notin_* \mathcal{L}(v_j)$ as required.

Proof. (Property $I_{\leq}$) For contradiction, assume there exist $n+1$ arbitrary edges $\langle b(u), b^{\varepsilon}_i(v_t), R\rangle \in E$ as in the definition of E, where $1 \leqslant i \leqslant n+1$ as well as an arbitrary literal $\leq n\, R.C \in J(b(u))$ and $C \in J(b^{\varepsilon}_i(v_t))$.

Let $K^{\varepsilon}_{v_j}$ and $M^{\varepsilon}_{v_j}$ be as specified in (7.20) and (7.21). Following Lemma 7.5 we have $J(b^{\varepsilon}_i(v_t)) \not\models K^{\varepsilon}_{v_j} \rightarrow M^{\varepsilon}_{v_j}$, so $C \in \zeta$ holds. Based on the definition of E there should exist n' edges $\varepsilon_t = (\langle u, v_t\rangle)$ with $(R, n_t, \zeta_t) \in \mathcal{L}(\varepsilon_t)$, $1 \leqslant t \leqslant n$ where $\sum_{t=1}^{n'} n_t \geqslant n+1$.

On the other hand, $\exists_{\leq n}\, y\, (R(x,y) \wedge C(y)) \in J(b(u))$ has to be produced by a productive clause $K \rightarrow M \vee \exists_{\leq n}\, y\, (R(x,y) \wedge C(y)) \in \mathcal{L}(u)$ based on the definition of the arithmetic label, $\exists_{\leq n}\, y\, (R(x,y) \wedge C(y)) \in \mathcal{C}(u)$ holds. So, the found solution satisfies the $\exists_{\leq n}\, y\, R(x,y) \wedge C(y)$. Since the Sigma Rule does not apply to the graph, one edge is created regarding each partition $\langle \rho, n\rangle \in \xi(u)$ the sum of all outgoing edges should be less than n, which mean that two of the edges in the model are the same.

□

Proof. (Property I_o) This property is derived from Claim 7.3 and Claim 7.4. □

Claim 7.6. Pre-interpretation I is a pre-model of $\mathcal{O}$.

Proof. Consider an arbitrary element $b^{K_q \to M_q}(v) \in \Delta$ and arbitrary normal clause $\alpha \in \mathcal{O}$. By Lemma 7.7 set $\mathcal{L}(v)$ is closed under hyperresolution with α, so $J(b^{K_q \to M_q}(v)) \models \alpha$ by Lemma 7.6. This holds for arbitrary $b^{K_q \to M_q}(v) \in \Delta$, so we have $I \models \alpha$. Finally, the latter holds for arbitrary $\alpha \in \mathcal{O}$, so I is a pre-model of $\mathcal{O}$. □

Claim 7.7. Pre-interpretation I refutes the query clause $K_q \to M_q$ for which there exists a node $v \in \mathcal{V}$ such that v is complete for $K_q \to M_q$, $K_q \to L \in_* \mathcal{L}(v)$ for each $L \in K$, and $K_q \to M_q \notin_* \mathcal{L}(v)$.

Proof. Let $K_q \to M_q$ be an arbitrary query satisfying the preconditions of the claim. According to the ordering defined in (7.7) – (7.10) node v is the first node in the pre-model construction, $t = 0$. Following the pre-model construction discussed in 7.2.4, the set of global ground atoms for this node $G_v = \emptyset$.

Regardless of the node type, when $t = 0$, then $v_t \nvdash K_t \to M_t$. So, an element $b(v_t)$ is created, by Lemma 7.13 we have $J(b(v_t)) \nvdash K_q \to M_q$. Therefore, as required, we proved that I refutes $K_q \to M_q$.

□

This section ends with an example illustrating the construction of a pre-model based on the complete completion graph in Example 5.6.

Example 7.1 (Pre-model Construction)**.** This example shows how to obtain the pre-model, for Example 5.6 that refutes the query clause $q = A(x) \rightarrow C(x)$, following the construction introduced in Section 7.2.4. The pre-model is constructed based on the result of applying the reasoning algorithm on the completion graph illustrated in Figure 5.5. To prove the contra-positive claim, we need to show that for any arbitrary query clause, $K_q \rightarrow M_q$, that is not derived in the completion graph $\mathcal{G}$, we can construct a pre-model of the ontology that refutes the query clause. Since this query clause is chosen arbitrarily, this holds for every clause not derived in the completion graph. As proven, the ontology refutes every clause that is not derived in G. We conclude that all query clauses entailed by the ontology are derived in the graph.

The construction is an iterative process based on the ordering introduced in Section 7.2.4. The query concept $A(x)$ has the highest priority by (7.9). By (7.19), the query clause $A(x) \rightarrow C(x)$ is accepted as the nodes query $K_1 \rightarrow M_1$. Since $v_A \nvdash A(x) \rightarrow C(x)$, the pre-model's element set V, defined in 7.2.4, contains the entity $\beta(v_A)$. Based on the literal interpretation definition (7.16), the label of this entity $J(\beta(v_A))$ contains $\exists\ y\ (R(x,y) \wedge F(y)$ and $H(x)$. While the set of global ground atoms $G_1 = \{B(o), F(o)\}$. Note that Elim rule removes the clauses (5.89) and (5.90) because of their strengthening (5.94). The update in $\mathcal{A}(v_A)$ triggers the ARM unit, and the completion graph would be updated based on the new solution by removing $\langle A, C\rangle$.

The nominal $O(x)$ is next in the ordering introduced in Section 7.2.4. The pre-model

construction (7.18) introduces one clause $O(x) \wedge B(x) \wedge F(x) \rightarrow \bot$ corresponding to this node. Since $v_O \nvdash O(x) \wedge B(x) \wedge F(x) \rightarrow \bot$, the element set, V contains the entity $\beta(v_O)$, by 7.2.4. The literal interpretation (7.16) adds $O(x), B(x)$ and $F(x)$ to the label of this entity $J(\beta(v_O))$. Note that (7.19) does not introduce any clauses for node v_C, so there are no entities in V corresponding to this node because it has no incoming edges. By the definition of E in 7.2.4, $\langle \beta(v_A), \beta(v_O), R \rangle$ is the only edge in the pre-model.

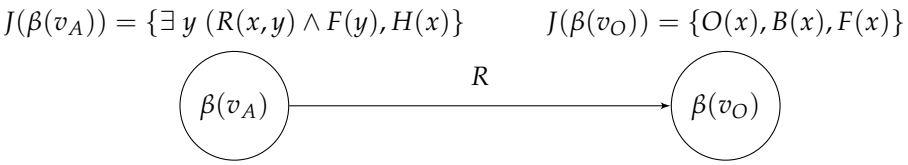


FIGURE 7.1: Pre-model of example 5.6
refutes the query clause $q = A(x) \rightarrow C(x)$

7.3 Time Complexity

Due to the use of the atomic decomposition technique, our algorithm runs in worst-case double exponential time. However, in DL reasoning, the emphasis is more on practical performance and scalability than worst-case theoretical complexity. Even worst-case optimal reasoners for expressive DLs with an exponential time complexity might be impractical.

Proposition 7.1 (time complexity)**.** The proposed consequenced-based algorithm has worst-case double exponential complexity.

Proof. The algorithm can generate $\wp$ of different clauses in each node, using the symbols in $\mathcal{O}$, which is exponential to the size of $\mathcal{O}$. Also, at most m clauses might participate in each inference, which is linear in the size of O. Accordingly, if k is the number of nodes in the graph, the number of inferences is bound by $k \times \wp \times m$, which is exponential. However, deriving each inference might rely on calling the Arithmetic Module, which is NExpTime-Complete following Proposition 5.1 (detailed time complexity analysis [66]). Therefore, in the worst case our algorithm runs in double exponential time. □

Part III

Implementation and Evaluation

Chapter 8

Implementation

To validate the practicality of the proposed calculus described in Chapter 5 and Chapter 6, we implemented it in a new reasoning system called CARON (Consequence-based Algebraic Reasoning for $\mathcal{O}$ (nominals) and $\mathcal{N}$ (number restrictions)). CARON currently targets the $\mathcal{SHOQ}$ subset of OWL 2 DL (i.e., it does not support datatypes) for which it can decide concept satisfiability and concept subsumption and classify an ontology. The system is written in Java[1] and relies on OWL API[2] and IBM ILOG CPLEX optimizer[3] [14].

Through the rest of this chapter, in Section 8.1, we overview the architecture and main features of CARON 1.0. In Section 8.2, we describe the *Completion Graph* data structure and discuss the inference rules implementation details which enable CARON to reason with nominals and cardinality restrictions. Finally, Section 8.3 presents the implementation details of the Arithmetic Module using CPLEX.

[1] java.version: 17.0.9, java.system.library: JavaSE-1.8
[2] owl-api version: 5.1.6
[3] cplex studio version: V12.8.0

8.1 Overall Architecture

Figure 8.1 depicts the architecture of CARON, where the reasoning engine operates not directly on OWL API representations but through a dedicated set of OWL API Bindings that facilitate seamless translation back and forth.

The CARON Reasoning Engine lies at the heart of the system (Figure 8.1). CARON 1.0 accepts OWL 2 DL ontologies without datatypes and HasKey axioms as input. It functions in a best-effort mode by default, ignoring unsupported axioms, but can be configured to raise an exception and halt the reasoning process upon encountering them.

The Ontology Loader is the first stage, transforming OWL axioms into normalized DL-clauses (defined in Section 4.3). This transformation involves two key steps: converting axioms to Negation Normal Form (NNF) and applying a structural transformation variant to generate a set of normalized DL-clauses. Like most DL reasoners, CARON leverages indices to identify clauses relevant to inferences or simplification rules efficiently. Therefore, the Ontology Indexing phase immediately follows the normalization routine, indexing the generated DL-clauses.

Once an ontology $\mathcal{O}$ is loaded, transformed, and indexed, the *Completion Graph Manager* creates and saturates a completion graph $\mathcal{G}$ to classify the ontology. Following Step [S1] of Algorithm 6, the *Completion Graph Manager* initializes a node v_C with *core* $C(x)$ for each named concept $C \in \mathcal{O}$. Furthermore, it defines a node ordering for each node, ensuring that a direct comparison between named concepts (excluding auxiliary predicates) is not possible. This constraint is paramount for achieving completeness. The node order also mandates that ground atoms assume a position of greater precedence relative to terms. In contrast, fresh concepts introduced during normalization have a lower priority

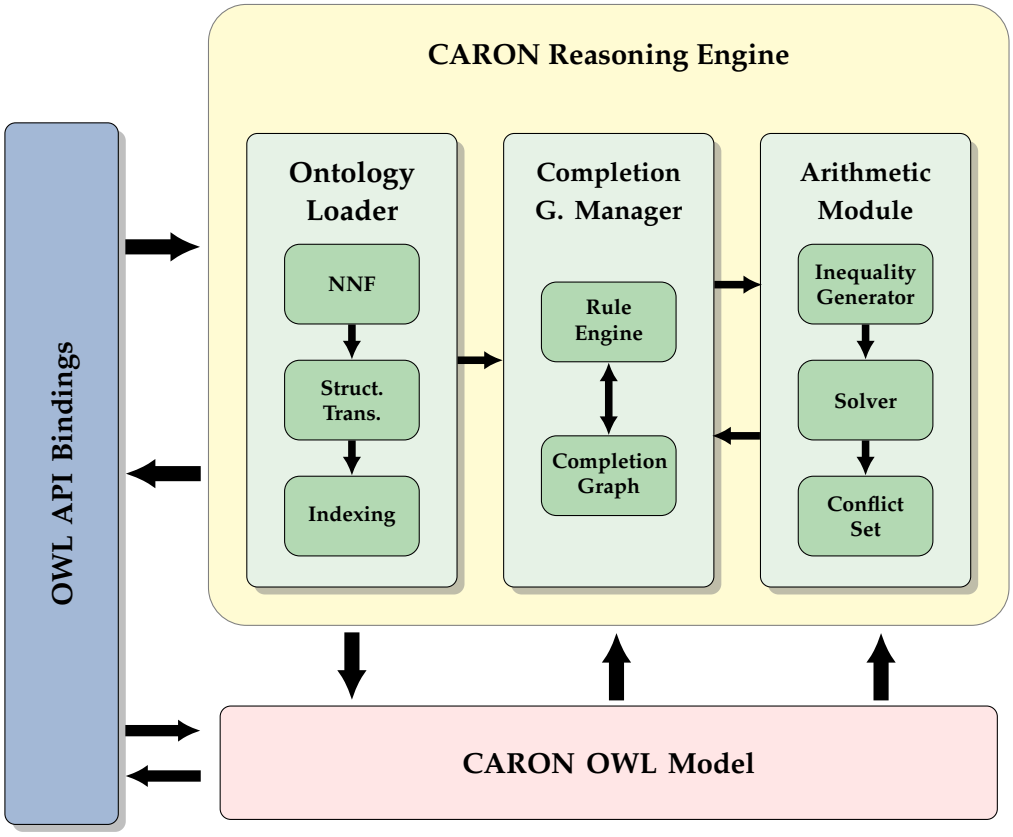


FIGURE 8.1: Architecture of CARON. Boxes represent key components, and arrows indicate the flow of information.

compared to those found within the original input ontology. These strategically implemented restrictions collectively contribute to the enhancement of the reasoner's overall performance.

Indexing is crucial to speed up the application of the Subs rule. CARON maintains several indexes that can quickly identify the clauses that can participate in the inference rule. During the ontology indexing phase, ontology $\mathcal{O}$ is indexed using index ind_1, which is a multi-map hash table that maps each atomic concept to a set of all DL-clauses of $\mathcal{O}$ that contains that concept in their body. Note that the set of all DL-clauses of $\mathcal{O}$ whose

body is empty are also indexed using the $\top$(OWLThing) concept as their key.

Finally, for each node $v \in \mathcal{V}$, we index the node's label $\mathcal{L}(v)$ using a multi-map hash table ind_2 that maps every atomic concept to the set of all clauses in $\mathcal{L}(v)$ where that atomic concept is the maximal atom in their head. Index ind_1 contains the DL-clauses in $\mathcal{O}$ and are thus immutable, while index ind_2 is updated whenever $\mathcal{L}(v)$ is updated.

These indexes are used as follows. Assume we wish to apply the Subs rule to a head atom A of a clause C. Then, $ind_1[A]$ returns the set of clauses whose bodies contain A. For each clause $K \rightarrow M$ and each atom $A' \in K$, we query $ind_2[A']$ to identify all the premises that can participate in the inference. Moreover, the clauses in $ind_1[\top]$ are added to each node as soon as it is created.

8.2 Completion Graph Manager

The construction of the completion graph $\mathcal{G}$ commences with initializing nodes representing atomic concepts occurring in the bodies of the query set $\mathcal{Q}$ clauses. Additionally, a node is generated for each nominal within the ontology. Notably, $\mathcal{G}$ distinguishes between two node types: nominal nodes and regular nodes. As their names imply, the *core* of a nominal node is a nominal, and the core of a regular node is a regular atomic concept. Upon creating a node v, we initialize an empty set $\mathcal{L}(v)$ to store derived clauses, an auxiliary empty queue $\mathcal{U}(v)$ for active clauses awaiting processing, and an arithmetic label set $\mathcal{A}(v)$ for relevant ARM clauses. Additionally, an empty predecessor list $\mathcal{P}(v)$ is created and updated through completion graph edges. This list facilitates the efficient application of the Reach rule later.

Algorithm 6 Node v processing steps

UpdateLabels(CLs)**:** If $\mathcal{L}(v)$ does not already contain strengthenings of the inference clauses CLs, add them to $\mathcal{L}(v)$ and $\mathcal{U}(v)$. Also, update the arithmetic label $\mathcal{A}$ if the inference clauses are relevant to ARM.

[S1] Apply the Initialization Step by adding the initialization clause to $\mathcal{L}(v)$ and $\mathcal{U}(v)$.

[S2] Apply the Subs rule with ontology clauses of the form $\top \rightarrow M$; add all derived clauses to $\mathcal{L}(v)$ and $\mathcal{U}(v)$. Also, add the relevant clauses to $\mathcal{A}(v)$.

[S3] While $\mathcal{U}(v)$ is not empty:

(a) Pick a clause C from $\mathcal{U}(v)$; let $\mathcal{M}$ be the set of maximal literals in C.

(b) Apply all Subs rule inferences involving a literal in $\mathcal{M}$, ontology clauses, and clauses in $\mathcal{L}(v)$. Assuming CLs is the set of all inference clauses, **UpdateLabels**(CLs).

(c) Apply all inferences with the Nom, Glob and Join rules involving a literal in $\mathcal{M}$ and clauses in $\mathcal{L}(v)$; **UpdateLabels**(CLs) with the inference clauses.

(d) Erase C from $\mathcal{U}(v)$.

[S4] If the arithmetic label $\mathcal{A}(v)$ has been updated, call ARM.

(a) If the arithmetic label is feasible, apply the Sigma rule on the solution by propagating the relevant information to the successor nodes and updating the edges (list $\mathcal{P}$ is updated accordingly). Clauses are propagated through the connection between nodes, so if a connection cannot be established with a node, this means that the node or the connecting edge has not been created yet, so the Completion Graph Manager initializes an appropriate node/edge.

(b) If the returned solution contains any Nominals in a partition, apply the Strict rule and add the inferences to the to $\mathcal{L}(v)$ and $\mathcal{U}(v)$ and update the arithmetic label accordingly.

(c) If the arithmetic label is infeasible, apply the Bottom rule on each returned conflict set. Add the inference clauses to $\mathcal{L}(v)$ and $\mathcal{U}(v)$ and update the arithmetic label accordingly.

[S5] For each new clause added to $\mathcal{L}(v)$, propagate any relevant inference(s) to the precedessor nodes $\mathcal{P}$ using the Reach rule.

Following node creation, v undergoes the steps outlined in Algorithm 6. Notably, the Fct rule, responsible for immediate simplifications after deriving a new clause C, is exclusive to nominal nodes. Set $\mathcal{L}(v)$ utilizes a redundancy index to prevent redundancy. This ensures that C is only added to $\mathcal{L}(v)$ (or $\mathcal{U}(v)$ and $\mathcal{A}(v)$) if it is not subsumed by an existing clause C'. Furthermore, any clauses in $L(v)$ subsumed by C are removed upon C's addition, implementing the Elim rule. Since the derivation of new inferences within each node is independent of the state of other nodes, this part of the algorithm is easily parallelizable.

When $\mathcal{U}(v) = \emptyset$ for all nodes $v \in \mathcal{V}$, the completion graph is fully constructed, and no active nodes remain in the processing queue. Consequently, the algorithm terminates and extracts the derived subsumption relations from the completion graph. For atomic concepts C and D, the inference $C \sqsubseteq D$ is made if $\top \rightarrow D(x) \in \mathcal{L}(v_C)$. CARON further utilizes standard criteria within the graph to determine the consistency of the ontology $\mathcal{O}$. Specifically, $\mathcal{O}$ is inconsistent if $\top \rightarrow \bot$ appears in the label of all nodes or is inferred within the label of a nominal or the *Thing* node; otherwise, the ontology is consistent.

8.3 Arithmetic Module Implementation

In the proposed algorithm, ARM is critical in handling the numerical restrictions imposed by QCRs and nominals. It starts by generating inequalities corresponding to the numerical restrictions in the arithmetic label. Then, ARM verifies the feasibility of the resulting integer linear problem by either finding a solution to satisfy all these constraints or pinpointing the smallest group of unsatisfiable constraints that causes infeasibility.

Integer Linear Programming (ILP) models are characterized by an objective function that must be optimized while adhering to a system of linear constraints. Notably, ILP falls within the broader domain of Linear Programming (LP), with the distinguishing feature of imposing integrality constraints on all variables. Integer Programming (IP) problems can be solved using the widely known Simplex method for LP, extended with the branch and bound technique to solve the integer constraints. ARM uses IBM ILOG CPLEX optimizer [14] API in Java to formalize and solve linear problems.

ARM receives $\mathcal{A}(v)$ as input, which contains a set of disjunct cardinality restrictions $\mathcal{Q}(v)$ occurring in $\mathcal{L}(v)$ and a set of relevant clauses $\mathcal{C}(v)$. Completion Graph manager updates $\mathcal{Q}(v)$, whenever a clause containing cardinality restrictions is added or removed from $\mathcal{L}(v)$. Consequently, $clos(v)$ and $\mathcal{C}$ are updated. If the ontology contains any nominals, we also create a set of related nominals $RO(v)$ for each regular node $v \in \mathcal{V}$, which contains the set of nominals $clos(v)$. On the other hand, every nominal node has a set of related concepts $RC(v)$ to which it is related. These sets are required for efficiently updating the clauses in $\mathcal{C}(v)$ with nominal clauses (Definition 5.7).

Primarily, ARM starts with assigning a positive integer id number to each cardinality restriction in $\mathcal{Q}(v)$, which is used as their index. It proceeds with defining a Master Problem and generating LE (less than equal) or GE (greater than or equal) CPLEX ranges corresponding to at-least or at-most restrictions, respectively. Similarly, it assigns id numbers to members of the decomposition set, and ARM continues with defining the Pricing Problem based on the clauses in $\mathcal{C}$. It then calls the Branch-and-Bound method defined in Algorithm 1. The $B\&B$ algorithm determines the feasibility of the arithmetic label and returns a solution or the conflict sets, which are handled by Sigma rule and Bottom rule, respectively.

Chapter 9

Evaluation

In this chapter, we discuss the methodology employed to evaluate the performance of CARON on classification and consistency checking of real-world ontologies. We also evaluate CARON's performance on interesting designed ontologies containing entailments due to reasoning about QCRs and nominals. In these experiments, we utilize the same methodology introduced by [62] and followed by [8] and [9]. Section 9.1 describes our evaluation methodology, including the input repositories and the systems used. Section 9.2 delves into the key findings of our experiment and offers a critical analysis, dissecting both the successes and shortcomings of the proposed reasoning algorithm, which provides valuable insights for future research.

9.1 Methodology

We pursue three main objectives in the designed evaluation experiments:

O1 Show that the final implementation is sound and complete. The employed heuristics and optimizations have not jeopardized these foundational requirements.

O2 Evaluate the efficiency of the proposed ILP-based algebraic reasoning on handling numerical restriction, namely QCRs, nominals and their interaction.

O3 Compare the efficiency of the proposed algorithm in handling real-world ontologies with state-of-the-art DL reasoners.

To achieve these objectives, we have selected the following repositories for our experiments:

- Challenging *Test Ontologies Repository* consists of 155 small ontologies containing tricky entailments. These ontologies were designed and gathered to test the completeness and soundness of DL reasoners. We have adapted these ontologies to match CARON's supported expressively level, $\mathcal{SHOQ}$.
- The *HARD Ontology Repository* consists of 886 ontologies with $\mathcal{SHOQ}$ expressivity. These ontologies are gathered and adapted to evaluate the performance of the HARD reasoner [19]. Find a more detailed description of these ontologies in [18].
- The *Canadian Parliament (CP) Ontologies* contains 38 ontologies, which are variants of Canadian Parliament ontology introduced in [18]. The expressiveness of these ontologies ranges from $\mathcal{ELQ}$ to $\mathcal{ALCOQ}$. Some of these ontologies are intentionally designed to be inconsistent to verify the soundness and completeness of the reasoners and to evaluate the impact of ontology consistency on their processing time.
- The *Oxford Ontology Repository* consists of 799 ontologies. Sizes of the ontologies vary from ∼1KB to ∼1GB, and their expressiveness ranges from lightweight languages such as DL-Lite and $\mathcal{EL}$ to $\mathcal{SROIQ}(\mathcal{D})$ [63].

We ended up with 762 ontologies in the Oxford Repository after preprocessing each ontology by performing the following steps:

1. **Syntax Verification**. We verified that each ontology could be successfully loaded by the OWL API 5.1.6 and eliminated all empty ontologies.

2. **Decidability Check**. We verified the use of complex object properties in QCRs to ensure decidability. We ignored the complex object properties definitions to obtain decidable ontologies. A total of 7 ontologies contained axioms violating these restrictions.

3. **Datatypes Elimination**. CARON does not currently support datatype reasoning; thus, we eliminated datatypes and also ignored all HasKey axioms.

4. **ABox Elimination**. It is known that in the absence of TBox nominals, eliminating the ABox from knowledge bases does not affect the result of classification or TBox consistency. Accordingly, we removed all the ABox individuals that are not TBox nominals.

We used version CARON 1.0 in our evaluation, which has been described in Chapter 8. The state-of-the-art reasoners used in the evaluations are HermiT 1.3.8, Konclude v0.6.2, FaCT++ 1.6.3 and JFact 1.2.2. We performed all the evaluations on an HP DL580 Scientific Linux SMP server with four 15-core processors and a total of 1 TB RAM. Konclude supports parallel reasoning; however, since this feature is not yet available in CARON 1.0, we compared all reasoners with their default configuration and used CPU time to register timeouts.

For each ontology and each reasoner, we created a fresh process which performs the following steps:

(i) **Load Ontology:** Loads the ontology using OWL API

(ii) **Instantiate Reasoner:** Creates a new instance of the reasoner for the ontology

(iii) **Consistency Check:** Asks the reasoner to return the TBox consistency result

We set 1000 seconds (∼16.5 minutes) of CPU runtime as the timeout threshold, checking the ontology consistency (Step (iii) above); if the reasoning continued beyond the threshold, the process was killed and recorded as timeout. We measured the wall-clock duration (elapsed time) of the whole process as processing time (i.e. Steps (i) - (iii) above). The consistency check results were compared to verify correctness. All systems and ontologies used in the experiment are available online. We run the TBox consistency check for all reasoners; however, since CARON is a consequence-based reasoner, it is designed to derive consistency by performing classification. Accordingly, it is not possible to completely disable classification and only verify the consistency. To compare consistency check timing with other reasoners, we have implemented some heuristics to withdraw the classification process and return the consistency result as soon as a known consistency result is derived (e.g. $\top \rightarrow \bot$ in the label of a nominal node entails that the ontology is inconsistent).

9.2 Experiment Results

The processing times for each repository introduced in Section 9.1 are summarised in Figures 9.1 to 9.4, respectively. For each reasoner, we sorted the processing times in ascending

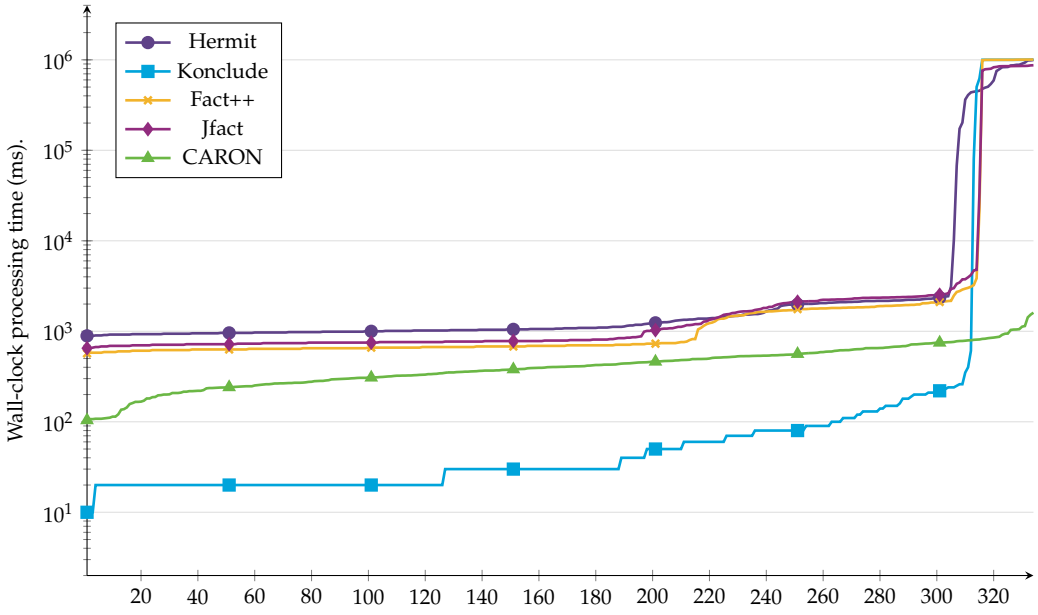


FIGURE 9.1: Processing Times for 155 Ontologies in *Test Repository*

order; a value point (x, y) in a line plot represents that the x-th smallest processing time for that reasoner was y milliseconds. Points where $y \simeq 1000s$ represent timeouts. The ontologies in *Test*, *Hard*, and *Canadian Parliament* Repositories are manually labelled with TBox consistency results. CARON correctly determined the consistency of all the ontologies in these repositories.

Figure 9.1 shows that on the *Test Repository* benchmark, CARON is outperforming all other reasoners by roughly an order of magnitude. CARON's processing time is independent of the number of QCRs and the value within at-most and at-least restrictions. Although Konclude has the lowest processing time for simple ontologies, CARON is the only reasoner capable of handling all the ontologies in the *Test Repository*. While all other reasoners, including Konclude, end with a timeout for ∼6% of ontologies. As a result, CARON has sped up reasoning about *Test Repository* by a factor of 125, on average, compared to other reasoners. The experiment shows that CARON's performance has a direct

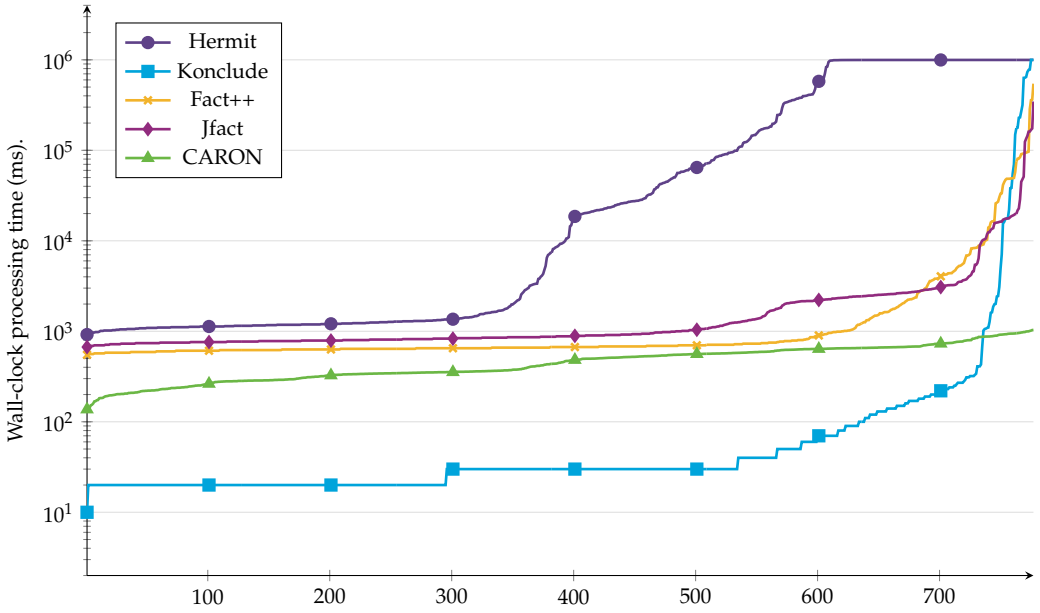


FIGURE 9.2: Processing Times for 886 Ontologies in *HARD Repository*

relation with the number of axioms in the ontologies. We believe this is due to the inefficient application of the Subs rule, which highly impacts its runtime and degrades its efficiency in handling larger ontologies.

The ontologies in *HARD repository* are mainly designed to illustrate a reasoner's capabilities in handling a large number of QCRs with large values. These are small ontologies (∼50 axioms on average) that contain interesting entailments involving the interaction of QCRs and Nominals. Figure 9.2 summarizes the processing times of ontologies in this benchmark. This diagram also supports the results from the previous experiment. CARON is the most stable reasoner to derive entailments that concern number restrictions, with no timeouts for ontologies in this repository. There are 80 ontologies in this repository where all other reasoners timeout while CARON successfully terminates in less than 1*s*. Although Konclude has the lowest processing time for simpler ontologies (low number of QCRs with small values), CARON is the most stable reasoner across all

the ontologies in the repository. Accordingly, CARON is speeding up the reasoning process for ontologies in *HARD repository* by several orders of magnitude (on average by a factor of ∼200). Table 9.1 summarizes the total processing times for *HARD repository*.

Reasoner	**processing time (s)**
Konclude	118,539
CARON	785
Fact++	113,006
Hermit	306,981
Jfact	98,313

TABLE 9.1: Total Processing Time for 886 Ontologies in *HARD Repository*

The *Canadian Parliament repository* represents the members of the Canadian parliament based on their distribution over Canadian provinces [18]. Figure 9.3 compares the performance of DL reasoners in testing the TBox consistency of variants of this ontology. The diagram shows that CARON successfully verifies the consistency of all the variants. CARON performance is a flat line independent of the number and value of QCRs, while all other reasoners will start timing out by increasing the number/value in the QCRs. Figure 9.3 shows that increasing the number of nominals still has a high impact on CARON's performance due to generating an extensive number of clauses containing ground atoms by Nom rule. Accordingly, further optimizations are required to make the application of this rule restrictive.

Figure 9.4 shows that the performance of CARON is competitive with that of well-established DL reasoners. Except for Konclude, CARON was the only reasoner capable of

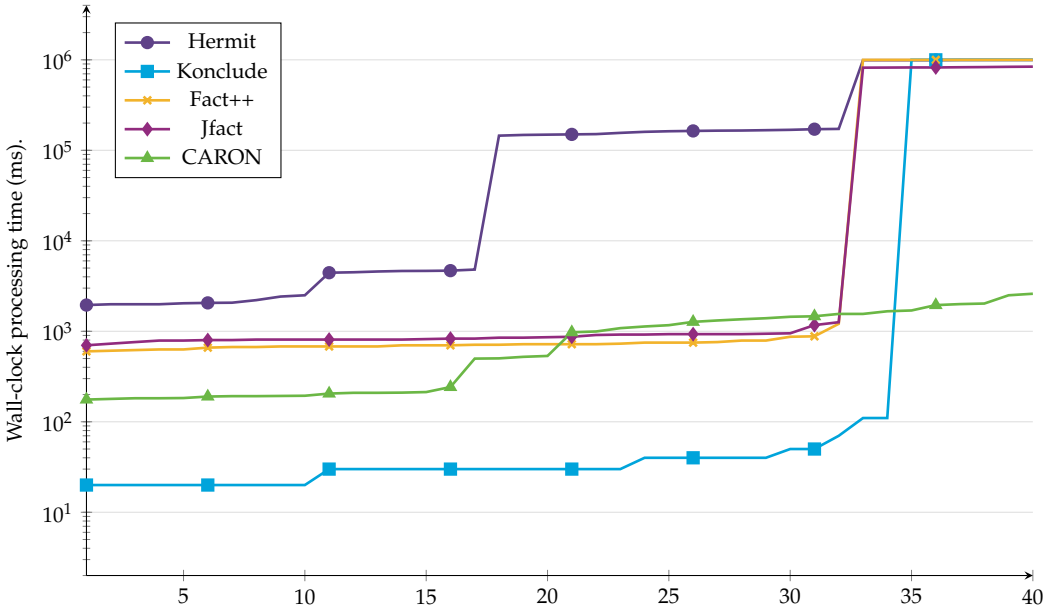


FIGURE 9.3: Processing Times for 38 Ontologies *CP Repository*

classifying ontology ID 667 in ∼50s, which is one of the ontologies with the highest number of nominals. CARONs performance degrades by increasing the number of concepts in the ontology. We believe this is caused by the large number of nodes in the completion graph. Ontologies with large numbers of disjuncts in the axioms were also significant sources of inefficiency, as the saturation process produced many clauses. This is a well-known source of performance issues in CB algorithms [9].

In essence, CARON is designed to accelerate deriving entailments involving nominals and QCRs. However, only 56 normalized ontologies in the *Oxford Repository* contain QCRs with values greater than one, out of which only 41 also contain nominals. Besides, there is no guarantee that there are any entailments that involve these number restrictions.

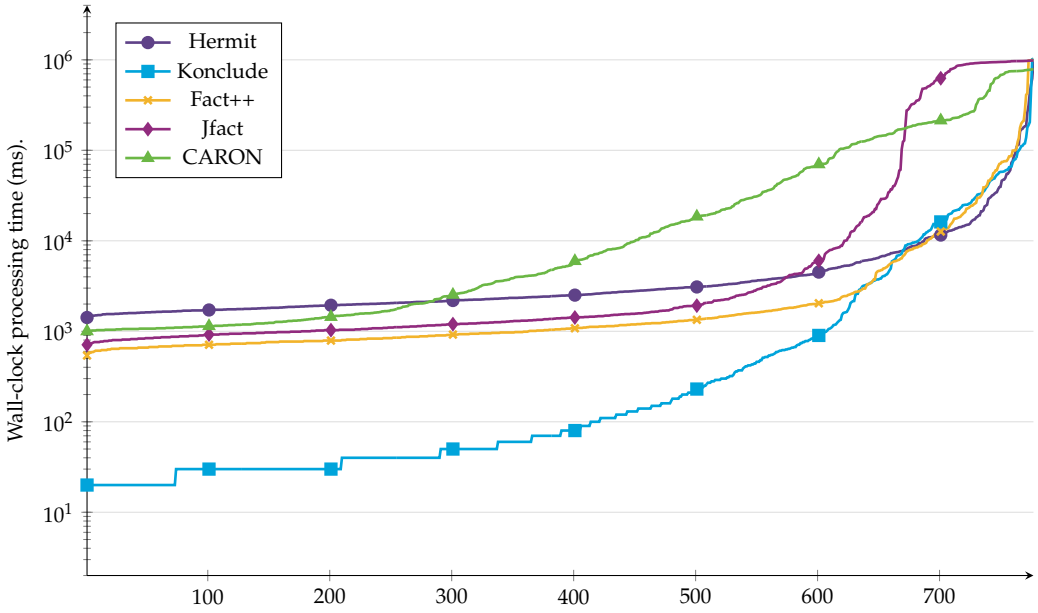


FIGURE 9.4: Processing Times for 762 Ontologies in *Oxford Repository*

Although most existing real-world ontologies do not include a large number of number restrictions, using cardinality restrictions and nominals in designing real-world ontologies is an unavoidable requirement in many domains, particularly in modelling structures with complex objects. The most simple example of QCRs is a model of a hand: A hand has five fingers, one of which is a thumb. There are numerous real-world domains where using QCRs and nominals is the most natural way and is considered the best practice by the OWL community, from modelling interstellar space to hockey tournament rules [1].

So far, expert ontologists have been avoiding the use of these numeric constraints as most well-known DL reasoners fall short in handling these constructors, especially with

high values in QCRs. Researchers have repeatedly reported that an arithmetically uniformed calculus cannot efficiently handle QCRs. This holds regardless of the underlying reasoning technique: tableau-based DL reasoning [28], resolution-based DL reasoning [36], hyper tableau-based DL reasoning [41] and consequence-based reasoning [64]. As a workaround for this problem, some ontology designers compensate the use of QCRs with concrete datatypes.

In summary, our implementation illustrates that CARON has a promising performance while it is not as mature as other well-established reasoners. Our approach has proved to be highly effective for handling QCRs and nominals. So, it can be a great complement for other reasoners to avoid arithmetically uninformed processes of numerical restrictions imposed by nominals and QCRs. There still appears to be lots of room for optimizing indexing techniques employed in implementation and better handling large ontologies with (hundreds of concepts). Note that although the size of an ontology greatly impacts the reasoning complexity, CARON prototype was not intended to tackle such a complexity.

Chapter 10

Conclusions and Future Work

This chapter summarizes the contributions accomplished in this research and proposes several possible future lines of studies that have opened up as a result of this work.

10.1 Conclusion

In this research, we proposed a theoretically sound and complete reasoning algorithm and demonstrated practical application by implementing it in the CARON prototype. This section captures the contributions of this thesis from theoretical and practical perspectives.

10.1.1 Theoretical Contributions

- **Extending Consequence-based Reasoners:** We proposed the first CB reasoning algorithm for an expressive DL that can efficiently handle the interaction of complex numerical restrictions imposed by the combination of QCRs and Nominals in [33]. Like other CB reasoners, the proposed calculi can derive all entailed subsumptions

by applying inference rules without backtracking. Primarily, we addressed the challenge of reasoning with global restrictions imposed by Nominals in the context of a consequence-based approach based on the derivation of local clauses. To allow having ground atoms and individual names in node clauses, we had to switch to a stronger logic than DL (a two-variable fragment of first-order logic). We introduced the required inference rules to exchange information between nominal nodes and the rest of the nodes in the graph that are not connected via edges.

- **Supporting an expressive DL:** The proposed calculi provides reasoning support for $\mathcal{SHOQ}$ which supports nominals ($\mathcal{O}$) and number restrictions ($\mathcal{N}$ or $\mathcal{Q}$). These two constructors both impose numerical constraints that may interact with each other. In [33], we proposed the first CB algorithm for reasoning in expressive description logic $\mathcal{SHOQ}$.

- **Arithmetic Reasoning:** QCRs and nominals are two constructors for imposing numerical restrictions. So, our reasoning approach highly benefits an arithmetic encoding for mapping these restrictions into a set of inequalities, the feasibility of which can be determined using standard Integer Linear Programming algorithms. We also employed Column Generation (an ILP optimization technique) to resolve the inefficiency by producing an exponential number of partitions. We formalized the interaction between the Arithmetic Module (ARM) and the Consequence-based inference engine by introducing secondary (arithmetic) labels for each node to accumulate numerical restrictions and their related knowledge. It also introduced suitable inference rules for reflecting the result of ARM in the completion graph.

- **Minimum Unsatisfiable Subsets** We can resolve an unsatisfiable set of numerical constraints to derive many consequences. However, finding the minimum core unsatisfiable set has proven challenging as it requires checking the satisfiability of all its subsets. We employed an innovative algorithm to discover all unsatisfiable minimum subsets without enumerating them by introducing a search map explored by solving its corresponding *Integer Linear Problem*.

- **Prove Correctness** The reasoning procedure must ensure soundness and completeness. A correct classification procedure must be *sound* and *complete*. A procedure is sound if every derived subsumption is implied by the input ontology $\mathcal{O}$, and it is complete if every subsumption implied by $\mathcal{O}$ can be obtained by the procedure. We formally proved the soundness and completeness of the proposed algorithm in Chapter 7.

10.1.2 Practical Contributions

- We developed CARON as a running prototype reasoner and evaluated its performance compared to well-established DL reasoners. To our knowledge, CARON is the first CB reasoner that employs arithmetically informed reasoning to handle the interaction of nominals and QCRs.

- We have compared the performance of CARON on checking TBox constancy against other well-established DL reasoners. Based on the empirical results, CARON offers competitive performance. At the same time, the algebraic approach proposed in this research can be considered a suitable complement for other reasoners to enable arithmetically informed reasoning.

- The practical evaluations in Chapter 9 showed that in about 85% of the test cases, CARON outperforms existing well-established reasoners in testing the consistency of ontologies that contain interesting entailments involving QCRs and nominals and, at the same time, speeding up the runtime by 2-3 orders of magnitude. The tests also show that the current implementation allows ample room for further optimization to improve reasoning with large ontologies containing large disjunctions in the head.

10.2 Future Work

Based on our research, we believe many open challenges can be addressed via future studies in this field. This section briefly discusses different open areas for extending this research.

- **More Expressive DLs**. An open future next step is to extend algebraic CB reasoning to more expressive DL languages such as $\mathcal{SROIQ}$, which is the logic foundation for OWL 2 DL. The DL $\mathcal{SROIQ}$ also allows inverse role and role relations in addition to the constructors supported by CARON. Handling the interaction of nominals, QCRs, and inverse roles (I) would be the main challenge of such an extension.

- **Datatypes**. To completely cover OWL 2 DL, another outstanding task is to extend our proposed calculus with datatypes. So far, there has been little progress in reasoning about expressive DLs with concrete domains. However, we believe that this step is well aligned with the arithmetic technique proposed in this thesis. CARON reasoner should also be updated to become equipped with datatype-handling capabilities to support Datatypes.

- **Optimisations for CARON**. The empirical evaluations have illustrated that there is still plenty of room for optimization; in particular, Nominals can also be a source of problems for CARON due to the derivation of large numbers of ground clauses in numerous nodes. A simple application of Nom rule imposes a high overhead by producing many clauses with ground atoms in their head.

- **Parallel Reasoning**. The introduced completion graph data structure should make it relatively straightforward to support parallel reasoning. While all local rule applications can be performed completely in parallel, one needs to set up standing points to ensure proper data propagation throughout the completion graph.

- **Optimized unsatisfiable set**. MARCO is an efficient algorithm for finding all minimum unsatisfiable sets. However, our current implementation relies on basic *grow* and **shrink** methods to find the core subsets, which are not particularly efficient, and many improvements can be made on both. This can be accomplished by replacing the *grow* and **shrink** algorithms with some of the state-of-the-art single minimum unsatisfiable set extraction algorithms.

Based on our observations, we firmly believe that using an algebraic approach is an absolute requirement to handle reasoning with numerical restrictions imposed by nominals and QCRs in OWL 2 DL. Combining this technique with existing well-established reasoners will provide efficient and scalable reasoning with numerical restrictions, allowing proper use of these constructors when naturally required within a domain.

Bibliography

[1] Allemang, D., Hendler, J., and Gandon, F. (2020). *Semantic Web for the Working Ontologist: Effective Modeling for Linked Data, RDFS, and OWL*, volume 33. Association for Computing Machinery, New York, NY, USA, 3 edition.

[2] Artale, A., Calvanese, D., Kontchakov, R., and Zakharyaschev, M. (2009). The DL-lite family and relations. *Journal of Artificial Intelligence Research*, 36(1):1–69.

[3] Baader, F. (2003). Terminological cycles in a description logic with existential restrictions. In *International Joint Conference on Artificial Intelligence*, pages 325–330.

[4] Baader, F., Brandt, S., and Lutz, C. (2005). Pushing the $\mathcal{EL}$ envelope. In *International Joint Conference on Artificial Intelligence*, pages 364–369.

[5] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook*. CUP.

[6] Baader, F., Lutz, C., and Suntisrivaraporn, B. (2006). *CEL — A Polynomial-Time Reasoner for Life Science Ontologies*, pages 287–291. Springer Berlin Heidelberg.

[7] Baader, F. and Sattler, U. (2001). An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40.

[8] Bate, A., Motik, B., Cuenca Grau, B., Simančík, F., and Horrocks, I. (2015). Extending Consequence-Based Reasoning to $\mathcal{SHIQ}$. In Calvanese, D. and Konev, B., editors, *Description Logics Workshop*, volume 1350. CEUR-WS.org.

[9] Bate, A., Motik, B., Grau, B. C., Cucala, D. T., Simancik, F., and Horrocks, I. (2018). Consequence-based reasoning for description logics with disjunctions and number restrictions. *Journal of Artificial Intelligence Research*, 63:625–690.

[10] Bate, A., Motik, B., Grau, B. C., Simančík, F., and Horrocks, I. (2016). Extending consequence-based reasoning to $\mathcal{SRIQ}$. In *Principals of Knowledge Representation and Reasoning*, pages 187–196.

[11] Borning, A. and Freeman-Benson, B. N. (1995). The OTI constraint solver: A constraint library for constructing interactive graphical user interfaces. In Montanari, U. and Rossi, F., editors, *Constraint Programming*, volume 976 of *Lecture Notes in Computer Science*, pages 624–628. Springer.

[12] Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., and Rosati, R. (2007). Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *Journal of Automated Reasoning*, 39(3):385–429.

[13] Chinneck, J. W. (2016). *Practical Optimization: A Gentle Introduction*, chapter 13. Carleton University.

[14] Cplex, I. I. (2009). V12. 1: User's manual for cplex. *International Business Machines Corporation*, 46(53):157.

[15] Cucala, D. T., Grau, B. C., and Horrocks, I. (2017). Consequence-based reasoning for description logics with disjunction, inverse roles, and nominals. In Artale, A., Glimm,

B., and Kontchakov, R., editors, *Proceedings of the 30th International Workshop on Description Logics, Montpellier, France, July 18-21, 2017*, volume 1879. CEUR-WS.org.

[16] Cucala, D. T., Grau, B. C., and Horrocks, I. (2018). Consequence-based reasoning for description logics with disjunction, inverse roles, number restrictions, and nominals. In *International Joint Conference on Artificial Intelligence*, pages 1970–1976.

[17] Ecke, A., Peñaloza, R., and Turhan, A.-Y. (2014). Completion-based generalization inferences for the description logic ELOR with subjective probabilities. *Journal of Approximate Reasoning*, 55(9):1939–1970.

[18] Faddoul, J. (2011). *Reasoning Algebraically with Description Logics*. PhD thesis, Concordia University.

[19] Faddoul, J. and Haarslev, V. (2010). Algebraic tableau reasoning for the description logic $\mathcal{SHOQ}$. *Journal of Applied Logic, Special Issue on Hybrid Logics*, 8(4):334–355.

[20] Farsiniamarj, N. and Haarslev, V. (2010). Practical reasoning with qualified number restrictions: A hybrid ABox calculus for the description logic $\mathcal{SHQ}$. *AI Communications*, 23(2-3):205–240.

[21] Finger, M. and De Bona, G. (2017). Algorithms for deciding counting quantifiers over unary predicates. In *Conference on Artificial Intelligence*, pages 3878–3884, San Francisco, California, USA.

[22] Graedel, E., Otto, M., and Rosen, E. (1997). Two-variable logic with counting is decidable. In *Logic in Computer Science*, pages 306–317. IEEE Computer Society.

[23] Grosof, B. N., Horrocks, I., Volz, R., and Decker, S. (2003). Description logic programs: Combining logic programs with description logic. In *International World Wide Web Conference*, pages 48–57. ACM.

[24] Haarslev, V. and Möller, R. (2001). Optimizing reasoning in description logics with qualified number restrictions. In *Description Logics Workshop*, pages 142–151.

[25] Haarslev, V. and Möller, R. (2001). Racer system description. In *International Joint Conference on Automated Deduction*, pages 701–705. Springer.

[26] Haarslev, V., Möller, R., and Turhan, A.-Y. (2001a). Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In *International Joint Conference on Automated Deduction*, pages 61–75. Springer-Verlag.

[27] Haarslev, V., Timmann, M., and Möller, R. (2001b). Combining tableaux and algebraic methods for reasoning with qualified number restrictions. In *Description Logics Workshop*, pages 152–161.

[28] Horrocks, I. (2002a). Backtracking and qualified number restrictions: Some preliminary results. In *Description Logics Workshop)*, volume 63 of *CEUR*, pages 99–106.

[29] Horrocks, I. (2002b). Reasoning with expressive description logics: Theory and practice. In *International Conference on Automated Deduction*, pages 1–15. Springer.

[30] Horrocks, I., Kutz, O., and Sattler, U. (2006). The even more irresistible $\mathcal{SROIQ}$. In *Principals of Knowledge Representation and Reasoning*, pages 57–67.

[31] Horrocks, I. and Sattler, U. (2007). A tableau decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 39(3):249–279.

[32] Hudek, A. K. and Weddell, G. E. (2006). Binary absorption in tableaux-based reasoning for description logics. In *Description Logics Workshop*.

[33] Karahroodi, N. Z. and Haarslev, V. (2017). A consequence-based algebraic calculus for $\mathcal{SHOQ}$. In Artale, A., Glimm, B., and Kontchakov, R., editors, *International Workshop on Description Logics*, volume 1879, Montpellier, France.

[34] Kazakov, Y. (2009). Consequence-driven reasoning for Horn-$\mathcal{SHIQ}$ ontologies. In Boutilier, C., editor, *International Joint Conference on Artificial Intelligence*, pages 2040–2045.

[35] Kazakov, Y., Krötzsch, M., and Simančík, F. (2012). Practical reasoning with nominals in the $\mathcal{EL}$ family of description logics. In *Principals of Knowledge Representation and Reasoning*, pages 264–274.

[36] Kazakov, Y. and Motik, B. (2008). A resolution-based decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 40(2):89–116.

[37] Lawley, M. J. and Bousquet, C. (2010). Fast classification in Protégé: Snorocket as an OWL 2 $\mathcal{EL}$ reasoner. In *The International Association for Ontology and its Applications*, volume 122, pages 45–49.

[38] Liffiton, M. H. and Malik, A. (2013). Enumerating infeasibility: Finding multiple MUSes quickly. In *Integration of AI and OR Techniques in Constraint Programming*.

[39] Minsky, M. (1981). A framework for representing knowledge. In Haugeland, J., editor, *Mind Design: Philosophy, Psychology, Artificial Intelligence*, pages 95–128. MIT Press.

[40] Motik, B., Patel-Schneider, P. F., and Grau, B. C. (2008). OWL 2 web ontology language: Direct semantics.

[41] Motik, B., Shearer, R., and Horrocks, I. (2009). Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research*, 36:165–228.

[42] Ohlbach, H. J. and Köhler, J. (1998). *How to Augment a Formal System with a Boolean Algebra Component*, pages 57–75. Springer Netherlands.

[43] Ohlbach, H. J. and Köhler, J. (1999). Modal logics, description logics and arithmetic reasoning. *Journal of Artificial Intelligence*, 109(1-2):1–31.

[44] Ortiz, M., Rudolph, S., and Simkus, M. (2010). Worst-case optimal reasoning for the Horn-DL fragments of OWL 1 and 2. In Lin, F., Sattler, U., and Truszczynski, M., editors, *Principals of Knowledge Representation and Reasoning*, pages 269–279.

[45] Osumi-Sutherland, D., Reeve, S., Mungall, C. J., Neuhaus, F., Ruttenberg, A., Jefferis, G. S. X. E., and Armstrong, J. D. (2012). A strategy for building neuroanatomy ontologies. *Bioinformatics*, 28(9):1262–1269.

[46] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., and Rosati, R. (2008). *Linking Data to Ontologies*, pages 133–173. Springer Berlin Heidelberg.

[47] Pratt-Hartmann, I. (2008). On the computational complexity of the numerically definite syllogistic and related logics. *Bulletin of Symbolic Logic*, 14(1):1–28.

[48] Quillian, M. R. (1967). Word concepts: A theory and simulation of some basic semantic capabilities. *Behavioral Science*, 12:410–430.

[49] Robinson, J. A. and Voronkov, A., editors (2001). *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press.

[50] Schmidt-Schauß, M. and Smolka, G. (1991). Attributive concept descriptions with complements. *Journal of Artificial Intelligence*, 1(48):1–26.

[51] Schulz, S., Cornet, R., and Spackman, K. A. (2011). Consolidating SNOMED CT's ontological commitment. *Applied Ontology*, 6:1–11.

[52] Schulz, S., Suntisrivaraporn, B., and Baader, F. (2007). SNOMED CT's problem list: ontologists' and logicians' therapy suggestions. *Studies in health technology and informatics*, 129(Pt 1):802—806.

[53] Schweikardt, N. (2005). Arithmetic, first-order logic, and counting quantifiers. *ACM Trans. Comput. Logic*, 6(3):634–671.

[54] Simancik, F., Kazakov, Y., and Horrocks, I. (2011). Consequence-based reasoning beyond Horn ontologies. Technical report, Oxford University.

[55] Simančík, F. (2013). *Consequence-Based Reasoning for Ontology Classification*. PhD thesis, University of Oxford.

[56] Simančík, F., Kazakov, Y., and Horrocks, I. (2011). Consequence-based reasoning beyond Horn ontologies. In *International Joint Conference on Artificial Intelligence*, pages 1093–1098.

[57] Simančík, F., Motik, B., and Horrocks, I. (2014). Consequence-based and fixed-parameter tractable reasoning in description logics. *Artificial Intelligence*, 209:29–77.

[58] Sirin, E. (2006). *Combining Description Logic Reasoning with AI Planning for Composition of Web Services*. PhD thesis, University of Maryland.

[59] Sirin, E., Grau, B. C., and Parsia, B. (2006). From wine to water: Optimizing description logic reasoning for nominals. In Doherty, P., Mylopoulos, J., and Welty, C. A., editors, *Principals of Knowledge Representation and Reasoning*, pages 90–99.

[60] Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical OWL-DL reasoner. *Web Semantics*, 5(2):51–53.

[61] Steigmiller, A., Glimm, B., and Liebig, T. (2014). Optimised absorption for expressive description logics. In *Description Logics Workshop*, pages 324–335.

[62] Steigmiller, A., Liebig, T., and Glimm, B. (2012). Extended caching and backjumping for expressive description logics. In *Description Logics Workshop*, pages 514–529.

[63] Tena Cucala, D. (2019). *Consequence-based reasoning for the Description Logic $\mathcal{SROIQ}$*. PhD thesis, University of Oxford.

[64] Tena Cucala, D., Cuenca Grau, B., and Horrocks, I. (2021). Pay-as-you-go consequence-based reasoning for the description logic $\mathcal{SROIQ}$. *Journal of Artificial Intelligence*, 298:103518.

[65] Tsarkov, D. and Horrocks, I. (2006). FaCT++ description logic reasoner: System description. In *International Joint Conference on Automated Deduction*, pages 292–297.

[66] Vlasenko, J., Haarslev, V., and Jaumard, B. (2017). Pushing the boundaries of reasoning about qualified cardinality restrictions. In *Frontiers of Combining Systems*, pages 95–112, Brasília, Brazil.