Efficient Classification of Complex Ontologies

by

Weihong Song

Bachelor in Architecture, Tsinghua University, China, 1990
The Second Bachelor in Software, Tsinghua University, China, 2004
Master of Software, Tsinghua University, China, 2008

A DISSERTATION SUBMITTED IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF

Doctor of Philosophy

In the Graduate Academic Unit of Computer Science

Supervisor(s): Weichang Du, Ph.D., Faculty of Computer Science, UNB
Bruce Spencer, Ph.D., Faculty of Computer Science, UNB

Examining Board: Harold Boley, Ph.D., Faculty of Computer Science, UNB
Joseph D. Horton, Ph.D., Faculty of Computer Science, UNB
Donglei Du, Ph.D., Business Administration, UNB

External Examiner: Volker Haarslev, Ph.D., Dept. of Computer Science and
Software Engineering, Concordia University

This dissertation is accepted

Dean of Graduate Studies

THE UNIVERSITY OF NEW BRUNSWICK

May, 2015

©Weihong Song, 2015
Abstract

Description logics (DLs) are knowledge representation languages that provide the theoretical underpinning for modern ontology languages such as OWL and serve as the basis for the development of ontology reasoners. The task of ontology classification is to compute the subsumption relationships between all pairs of atomic concepts in an ontology, which is the foundation for other ontology reasoning problems. There are two types of mainstream reasoners to perform ontology classification: 1) tableau-based reasoners usually support very expressive DLs, but they may not be efficient for large and highly cyclic ontologies. 2) consequence-based reasoners are typically significantly faster than tableau-based reasoners, but they support less expressive DLs. It is difficult to extend the consequence-based reasoners directly to support more expressive DLs.

In the present thesis, we propose a weakening and strengthening based approach for ontology classification, which aims to extend the capability of an efficient reasoner for a less expressive base language $L_b$ ($L_b$-reasoner) to support a more expressive language. The approach approximates the target ontology by a weakened version and a strengthened version in $L_b$. Their subsumptions are a subset and a superset of the subsumptions of the target ontology. There are two cases:
(1) the subsumptions of the strengthened ontology are the same as that of the target ontology; (2) there may be more subsumptions in the strengthened ontology, which is therefore unsound. In case (1) which we call soundness-preserved strengthening, we classify only the strengthened ontology with the $L_b$-reasoner to get the final classification results. In case (2) which we call soundness-relaxed strengthening, a hybrid approach is employed – we first classify both the weakened and strengthened ontologies with the $L_b$-reasoner, and then use a full-fledged (hyper)tableau-based assistant reasoner to check whether the subsumptions implied by the strengthened ontology are also implied by the target ontology.

We first study the general principles to apply weakening and strengthening to extend an $L_b$-reasoner for a DL language that has one more constructor than $L_b$, i.e., single extension. Then we study the combination of several single extensions for multiple extended constructors for the reasoner, i.e., multiple extension.

Based on the general principles, we investigate two single extensions from the $\text{ALCH}$ description language to $\text{ALCH}(D)^-$ and $\text{ALCHI}$ with soundness-preserved strengthening, as well as a single extension from $\text{ALCH}$ to $\text{ALCHO}$ with soundness-relaxed strengthening. Then, we show how to combine them into multiple extensions from $\text{ALCH}$ to $\text{ALCHI}(D)^-$, $\text{ALCHOI}$, $\text{ALCHO}(D)^-$, and $\text{ALCHOI}(D)^-$. The soundness and completeness of all the single and multiple extensions are proved.

We also develop a prototype $\text{ALCHOI}(D)^-$-reasoner, WSClassifier, based on the proposed approach. We experiment with and evaluate WSClassifier on large and highly cyclic real-world ontologies such as FMA and Galen ontologies, the
required languages of which are beyond the capability of the current efficient consequence-based reasoners. The experiments show that on most of these ontologies, WSClassifier outperforms or significantly outperforms conventional (hyper)tableau-based reasoners.
Acknowledgements

It has been a great journey to pursue my Ph.D degree with my two supervisors Dr. Weichang Du and Dr. Bruce Spencer. I am deeply grateful to Professor Weichang Du, who not only gave me an entire new perception about my research from methodological and systematic views, but also directed me effectively to drill down to my research. His systematic theoretical guidance, rigorous logic, and brilliant advices substantially influence this thesis. I am deeply indebted to professor Bruce Spencer, who not only introduced me to tableau-based reasoning and encouraged me to the essential research problem, but also dug profoundly into my tangled research problems, directing me to accomplish the core theoretical proofs through numerous insightful discussions. The research presented in this thesis would not have been possible without the systematic methodological and in-depth guidance and various supports from both my supervisors.
I am also very grateful to Dr. Volker Haarslev, Dr. Harold Boley, Dr. Joseph Horton, Dr. Yevgen Biletskiy and Dr. Donglei Du, who are the members of my examining committee, as well as Dr. Christopher Baker, Dr. Alexandre Riazanov and Mr. Alan Meech for their many valuable and constructive suggestions.
I would like to express my gratitude to František Simančík, Birte Glimm, Yevgeny Kazakov, Ian Horrocks, and Boris Motik from University of Oxford for providing great assistance on the technical details of the reasoner ConDOR and HermiT employed in our prototype.

I also thank all the ORE2013 organizers – Ernesto Jimenez from University of Oxford, Birte Glimm from University of Ulm, Bijan Parsia and Rafael Goncalves from University of Manchester, and other organizers for their diligent work and great assistance in helping us to participate in the ORE2013 competition. And special thanks to Ian Horrocks, Yevgeny Kazakov, Birte Glimm and Uli Sattler for their valuable comments about our research on ORE2013.

Thanks are also due to the members of the Faculty of Computer Science of UNB and the advisor of International Student Advisor’s Office. Their support provides me a pleasant working environment. Special thanks to Jodi O’Nell and Tania Serrano for their great help.

I thank my former colleagues at the National Research Council Canada: Ebrahim Bagheri, Harold Boley, Sandy Liu, Yonghua You, Liqiang Geng, Yunli Wang, Yuanyuan Guo, Hongyu Liu, Stephen MacKay, Scott Buffett, Danny D’Amours and Serge Leger. I had great time with all of them.

Thanks are also extended to my house-mates Gen Zou and Shijie Xu, with whom I have some wonderful discussion.

Finally, I owe my deepest gratitude to my husband Yuanlong Gao and my daughters Luxi Gao and Zhenya Gao. Without their unconditional love and support, this thesis would not have been completed.
Table of Contents

Abstract ii

Acknowledgments v

Table of Contents xii

List of Tables xiv

List of Figures xv

List of Symbols xvi

List of Abbreviations xvii

1 Introduction 1

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 1

1.2 Thesis Objectives . . . . . . . . . . . . . . . . . . . . . . . . . . 9

1.3 Thesis Outline . . . . . . . . . . . . . . . . . . . . . . . . . . . . 10

2 Background and Related Work 13

2.1 Description Logic . . . . . . . . . . . . . . . . . . . . . . . . . . 13
6.1.2 Creating Weakened and Strengthened Ontologies in Multiple Extension .................................................. 116

6.2 Several Composed Multiple Extensions .......................................................... 117

6.3 ALCXI(D)− Ontology Classification ..................................................................... 120

6.4 Chapter Summary ................................................................................................ 122

7 System Design and Implementation ........................................................................ 123

7.1 System Design ..................................................................................................... 124

7.1.1 Overarching Architecture ............................................................................. 124

7.1.2 Index .............................................................................................................. 126

7.1.3 Hierarchy ....................................................................................................... 127

7.1.4 Optimization .................................................................................................. 133

7.2 Implementation .................................................................................................. 134

7.2.1 Java Classes .................................................................................................... 134

7.2.2 Implementation of Classification Procedure .................................................. 136

7.3 Chapter Summary ............................................................................................... 145

8 Experiments ............................................................................................................ 146

8.1 Introduction ........................................................................................................ 146

8.2 Experiments with ALCXI(D)− Ontologies ........................................................... 147

8.3 Experiments with ALCXI Ontologies .................................................................. 149

8.4 Experiments with ALCXI(D) Ontologies ............................................................ 152

8.5 ORE2013 Competition ....................................................................................... 155

8.5.1 Methodology .................................................................................................. 156
8.5.2 OWL2 DL Offline Competition .......................... 157
8.5.3 OWL2 DL Live Competition ............................. 159
8.5.4 Comparing HermiT and WSClассifier ................. 159
8.6 Chapter Summary .............................................. 162

9 Methodology Evaluation ......................................... 164
  9.1 Reasoning Characteristics ................................. 165
  9.2 Practical Usability .......................................... 166
  9.3 Reasoners ...................................................... 167
  9.4 Comparison of Reasoners ................................. 169
    9.4.1 Reasoning Characteristics ......................... 169
    9.4.2 Practical Usability .................................... 170
  9.5 Characteristics of Extension-Based Reasoners ............ 173
    9.5.1 Characteristics .......................................... 173
    9.5.2 Comparison of Extension-Based Reasoners .......... 175
  9.6 Chapter Summary .............................................. 179

10 Conclusions ..................................................... 180
  10.1 Summary ..................................................... 180
  10.2 Contributions .............................................. 181
    10.2.1 Theoretical Contributions ......................... 182
    10.2.2 Practical Contributions .............................. 184
  10.3 Future Work ............................................... 184
List of Tables

1.1 An Ontology Example ........................................ 3
1.2 Comparison of classification performance .................. 8

2.1 Basic DL knowledge base consisting of a TBox and an ABox ... 15
2.2 Syntax and Semantics of Data Ranges ....................... 17
2.3 Syntax and semantics of $SROIQ(D)$ constructors .......... 20
2.4 Syntax and semantics of $SROIQ(D)$ axioms ................ 21
2.5 Part of DLs considered in this thesis ....................... 24
2.6 Examples of derivation rules ................................ 28
2.7 Classification Time of Several Tableau Reasoners .......... 33

4.1 Encoding $\varphi$ for atomic concepts/roles/features/data ranges 60
4.2 Structural transformation ..................................... 72
4.3 Axioms added in structural transformation .................. 73
4.4 Rewritten Axioms ............................................ 73

5.1 Rewriting Nominals in OWL 2 ................................ 87
5.2 Complete Inference Rules for Normalized $\mathcal{ALCH}$ ontologies 89
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1 The Multiple Extensions to $(D)^{-}, I$ and $O$ Based on $\text{ALCH}$</td>
<td>118</td>
</tr>
<tr>
<td>7.1 The Index</td>
<td>127</td>
</tr>
<tr>
<td>7.2 Major Classes in Each Component</td>
<td>137</td>
</tr>
<tr>
<td>7.3 Major Classes in Each Component</td>
<td>138</td>
</tr>
<tr>
<td>8.1 Comparison of classification performance of $\text{ALCHI}(D)^{-}$ ontologies</td>
<td>148</td>
</tr>
<tr>
<td>8.2 Comparison of classification performance of $\text{ALCHO}$ ontologies</td>
<td>150</td>
</tr>
<tr>
<td>8.3 Comparison of classification performance of $\text{ALCHOI}(D)$ ontologies</td>
<td>153</td>
</tr>
<tr>
<td>8.4 Comparison of classification performance of $\text{ALCHOI}(D)^{-}$ ontologies with Konclude</td>
<td>153</td>
</tr>
<tr>
<td>8.5 Offline Competition Record</td>
<td>159</td>
</tr>
<tr>
<td>8.6 Live Competition Record</td>
<td>161</td>
</tr>
<tr>
<td>9.1 Worst-Case Complexities of Concept Satisfiability Checking</td>
<td>166</td>
</tr>
<tr>
<td>9.2 Reasoning characteristics</td>
<td>171</td>
</tr>
<tr>
<td>9.3 Practical Usability</td>
<td>172</td>
</tr>
<tr>
<td>9.4 Characteristics of Extension-based Approaches Table</td>
<td>176</td>
</tr>
</tbody>
</table>
List of Figures

1.1 Hierarchy before reasoning .............................................. 5
1.2 Hierarchy after reasoning ................................................ 5

7.1 Key components of WSClassifier ...................................... 125
7.2 Add a New Subsumption in Hierarchy ................................. 132
7.3 Overarching Class Diagram ............................................. 135

8.1 Offline Competition: OWL2 DL Classification (from ore 2013 official report [30]) ....................................................... 158
8.2 Live Competition: OWL2 DL Classification (from ore 2013 official report [30]) ....................................................... 160
List of Symbols

$L_b$ The base language
$L_o$ The original language
$O_w$ The weakened ontology
$O_s$ The strengthened ontology
$\mathcal{H}_O$ The classification result of the original ontology
$\mathcal{H}_w$ The classification result of the weakened ontology
$\mathcal{H}_s$ The classification result of the strengthened ontology
$\mathcal{H}_{ws}$ The classification result based on $\mathcal{H}_w$ and gradually supplemented with verified subsumptions from $\mathcal{H}_s$

$\text{Sig}(O)$ The signature of ontology $O$

MR The base reasoner
AR The assistant reasoner
$N_C$ Atomic concepts
$N_{C,T,\bot}$ $N_C \cup \{T, \bot\}$
$N_R$ Atomic roles
$N_I$ Individuals
$N_F$ Features
List of Abbreviations

DL    Description Logic
WS    Weakening and Strengthening
OWL   Web Ontology Language
TBox  Terminological axioms
ABox  Assertional axioms
Chapter 1

Introduction

This chapter is organized in three sections. Section 1.1 provides the motivation of this research; Section 1.2 introduces the objective of the thesis and Section 1.3 shows the outline of the whole thesis.

1.1 Motivation

During the first decade of the existence of the World Wide Web, most of the information on the Web was designed for only human consumption. Humans could read Web pages and understand them, but their inherent meaning was not shown in a way that allows their interpretation by computers [14]. The huge amount of information available on the Web has expanded beyond the capability of human beings. Such problems have made it necessary to find efficient ways to automatically locate meaningful information on the Web. Furthermore, it is important that
the information on the Web can be used by computers for interoperability and integration between systems and applications.

The objective of the Semantic Web, which is an extension of the World Wide Web, is to provide the information in such a way that computers can understand it and can process it using deductive reasoning and inference, thereby obtaining more meaningful results and performing automated information gathering and research. This is why the Semantic Web is an important infrastructure for interoperability.

As the foundation of the Semantic Web, ontology, is a shared, formal conceptualization of a domain as a description of concepts and their relationships [70, 93]. Ontologies are domain models with two special characteristics, which lead to the notion of shared meaning or semantics [57]:

1. Ontologies are expressed in formal languages with well-defined semantics;
2. Ontologies build upon a shared understanding within a community.

The first point underlines that an ontology needs to be modeled using languages with formal semantics. The second point reminds us that there is no such thing as a “personal ontology”. For example, the schema of a database or a UML class diagram that we have created for the design of our own application is not an ontology because there is no commitment toward this schema from anyone else but us.

The most popular ontology modeling language is the Web Ontology Language (OWL), which is built upon logic-based knowledge representation formalisms, known as Description Logics (DLs). Due to the connection of OWL to DLs, the basic reasoning services available for DL knowledge bases also apply to OWL ontologies.
Table 1.1 illustrates part of the axioms of a wine ontology, which defines concepts such as Wine, TableWine, WineSugar and DryRiesling and their relationship in the domain of wine. For decades, ontologies have been extensively employed in the fields of biology [65, 77, 82] and medicine [29, 75]. More recently, ontologies have also been applied to more diverse fields, such as astronomy [17], geography [31], defence [52], and agriculture [83].

Table 1.1: An Ontology Example

<table>
<thead>
<tr>
<th>Concept</th>
<th>Definition</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wine</td>
<td>⊑ ∃ hasSugar. WineSugar</td>
<td>(1)</td>
</tr>
<tr>
<td>TableWine</td>
<td>≡ Wine ⊓ ∃ hasSugar. {Dry}</td>
<td>(2)</td>
</tr>
<tr>
<td>WineSugar</td>
<td>≡ {Dry, OffDry, Sweet}</td>
<td>(3)</td>
</tr>
<tr>
<td>DryRiesling</td>
<td>⊑ Wine</td>
<td>(4)</td>
</tr>
<tr>
<td>DryRiesling</td>
<td>⊑ ∀ hasSugar. {Dry}</td>
<td>(5)</td>
</tr>
</tbody>
</table>

In order to help people effectively understand the relationship between the concepts in ontology, the concepts are usually organized in a hierarchical structure called a taxonomy which demonstrates the subsumption relationship between the concepts. In the ontology of Table 1.1, for example, the concept DryRiesling would be subsumed by the concept Wine and would be placed under this concept in the taxonomy. DL based modern ontology languages such as OWL-DL provide the capability to concisely model the target domain: instead of stating each of the subsumption relationship between the concepts explicitly, an ontology just states the basic definitions of concepts and their general and unique properties, from which the taxonomy can be computed by running a relevant reasoning algorithm [22]. Reasoning is critical in the creation phase of an ontology to detect
inconsistences or other modeling errors, which typically manifest themselves as unintended or missing subsumption relationships. Reasoning is also employed to answer sophisticated domain questions\[66\] in the application phase of an ontology. Ontology classification, of which the purpose is to compute the taxonomy, is a fundamental terminological reasoning task in DLs. In the following, we illustrate the taxonomy of the ontology from Table1.1.

Figure 1.1 shows an initial taxonomy that arises from an inspection of two axioms (2) and (4) in Table1.1 that explicitly show subsumption relationships between atomic concepts, i.e., TableWine is a subclass (type) of Wine from axiom (2); DryRiesling is a subclass(type) of Wine from axiom (4). However, other axioms can give further information. From axiom (1) we know all types of Wine must have sugar which belongs to WineSugar. Then, based on (4) and (1), we know that DryRiesling must have sugar which belongs to WineSugar. Axiom (5) means that if DryRiesling has sugar, the sugar type must belong to \{Dry\}, which is an nominal concept with only one instance Dry (3) shows that instance Dry is also a type of WineSugar. According to the above results, we know that DryRiesling is a type of Wine and DryRiesling must have sugar which is Dry. This result satisfies axiom (2) from the right to the left side, and thus we can draw the conclusion that DryRiesling is a subclass(type) of TableWine, which is implicit and can be obtained after reasoning on all the axioms in Table1.1. The final taxonomy is shown in Figure 1.2. In this thesis, we regard an ontology as complex if (1) it uses an expressive language, i.e. supporting more logic constructors in the language so
that it can express more complex meaning, or (2) it is highly cyclic. We say an ontology is cyclic if there are concepts defined in terms of themselves directly or indirectly through other concepts.

Most modern ontology reasoners, such as FaCT++ [91], HermiT [63], Pellet [81], and RacerPro [54] [35], implement optimized tableau-based algorithms, or variations thereof, which perform classification by trying to build counter-models for candidate subsumptions. Current (hyper)tableau-based reasoners are able to classify ontologies in very expressive DL, such as $SROIQ(D)$ covering the full OWL 2 DL profile. However, despite various optimizations having been applied, efficiently classifying certain existing and emerging ontologies is still a challenge for the (hyper)tableau-based reasoners. When the ontology is large and highly cyclic, (hyper)tableau-based reasoners tend to build large models. It takes very long time to construct the model, and computationally expensive blocking techniques need to be employed to ensure termination of the process. These factors make it extremely difficult for (hyper)tableau-based reasoners to classify large and highly
cyclic ontologies, such as various versions of Galen\textsuperscript{1} and FMA\textsuperscript{2} ontologies.

In order to obtain efficient reasoning, consequence-based reasoning algorithms have been invented, which directly derive logical consequences of axioms in the ontology using inference rules rather than building counter-models. The inference rules are designed to derive all implied subsumptions, while guaranteeing that only a bounded number of consequences is derived \textsuperscript{[80]}. All the subsumptions are obtained by applying the inference rules iteratively until no more rules can be applied. As a result, consequence-based reasoning is significantly different from tableau-based reasoning in two aspects. Firstly, it never checks the subsumption pairs that are not entailed by the ontology; while tableau-based reasoning tries to inspect all the pairs unless the relationship of one pair is already implied by a previous check. Since the number of entailed subsumptions is typically much smaller than the number of all pairs of concepts, consequence-based reasoning needs to perform much less work. For example, SNOMED CT \textsuperscript{[75]} entails only about 5 million subsumption relations, which is less than 0.01\% of the total number of possible subsumptions. Secondly, the classification procedure computes all subsumptions at once in ‘one pass,’ which requires fewer operations than testing the same number of subsumptions separately \textsuperscript{[51]}. These reasons explain why the consequence-based reasoning approach is significantly faster than tableau-based reasoning. However, consequence-based reasoning supports less expressive DLs

\footnotesize\textsuperscript{1}http://www.opengalen.org/index.html
\footnotesize\textsuperscript{2}Foundational Model of Anatomy, http://sig.biostr.washington.edu/projects/fm/index.html
like the OWL EL family. The most expressive languages that are supported by consequence-based reasoners are Horn-\textit{SHIQ} \cite{46} and \textit{ALCH} \cite{78}, which are less expressive than OWL Full \textit{SROIQ(D)}. Therefore, when an ontology uses an expressive language, the consequence based reasoners usually cannot completely classify it since they accept less expressive languages. The constructors which are partially supported by the current consequence-based reasoners are as follows: (1) inverse role, (2) transitive role, and (3) number restriction are supported by CB reasoner, which was developped for language Horn-SHIQ; 1)safe nominals, 2)numeric data type and 3)complex role chains are supported by ELK reasoner which was developped for OWL2 EL language. Among all the non-supported or partially supported constructors, inverse roles, datatypes and nominals are commonly used constructors.

Table \ref{tab:classification} gives classification results of several existing consequence- and tableau-based reasoners on some ontologies with expressive DLs. As is shown, consequence-based reasoners are fast but are unable to get complete results on some of the ontologies beyond their languages, while tableau-based reasoners are unable to efficiently classify the large and complex Galen and FMA ontologies, or are unable to get the results. In real applications, reasoning needs to be done under time-pressure and also there are other limited resources besides time, such as memory. So in this thesis, we regard the classification as practical if the task can be completed in reasonable time, usually within minutes. If the classification is completed in many days, then the reasoner can hardly be used in many real-world applications.
Table 1.2: Comparison of classification performance

MS: (# of subsumption pairs missing) / (# of total subsumption pairs)

T: Time(seconds)

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Criteria</th>
<th>(Hyper) tableau</th>
<th>Consequence-based</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>HermiT</td>
<td>Pellet</td>
</tr>
<tr>
<td>Wine</td>
<td>T</td>
<td>29.11</td>
<td>377.88</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>0/968</td>
<td>0/968</td>
</tr>
<tr>
<td>DOLCE</td>
<td>T</td>
<td>5.64</td>
<td>8.40</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>0/2595</td>
<td>0/2595</td>
</tr>
<tr>
<td>Galen-Heart-YN1</td>
<td>T</td>
<td>115.10</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>0/45,513</td>
<td>-</td>
</tr>
<tr>
<td>Galen-Heart-YN2</td>
<td>T</td>
<td>63.52</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>0/45,914</td>
<td>-</td>
</tr>
<tr>
<td>Galen-EL-YN1</td>
<td>T</td>
<td>197,090</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>0/431,990</td>
<td>-</td>
</tr>
<tr>
<td>Galen-EL-YN2</td>
<td>T</td>
<td>289,637</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>0/457,779</td>
<td>-</td>
</tr>
<tr>
<td>Galen-EL-LN1</td>
<td>T</td>
<td>188,018</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>0/431,990</td>
<td>-</td>
</tr>
<tr>
<td>Galen-Full-UnionN2</td>
<td>T</td>
<td>604,800+</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>/</td>
<td>-</td>
</tr>
<tr>
<td>FMA-cPFNS</td>
<td>T</td>
<td>667,430</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>MS</td>
<td>0/481,967</td>
<td>-</td>
</tr>
</tbody>
</table>

Note: ‘-’ entry means that the reasoner was unable to classify the ontology due to some problems. “/” entry means the number is not available. ‘+’ sign indicates the tasks are not completed within the time shown before ‘+’

Current consequence-based reasoners can meet the performance requirements of efficient practical classification, but may miss many subsumptions; while (hyper) tableau-based reasoners can provide complete results theoretically, but their performance may be impractical on large and highly cyclic ontologies. Each of the techniques has their advantages and disadvantages. We usually encounter expressive and highly cyclic ontologies. Due to the language limitation of consequence-
based reasoners and the performance issue for (hyper)tableau-based reasoners, the complete hierarchy of such ontologies cannot be obtained independently from any available reasoners in acceptable time.

1.2 Thesis Objectives

This thesis aims to develop practical reasoning techniques for classifying large and complex ontologies to have a similar performance to consequence-based reasoners, or a better performance than tableau-based reasoners in most cases, while providing more results than consequence-based reasoners.

The main objective of the research can be summarized as follows:

1) Propose a methodology to extend an efficient $L_b$-reasoner for language $L_b$ to classify ontologies in a more expressive language $L_o$. We will study the general principles of single extensions of an $L_b$-reasoner for languages that have one more constructor than $L_b$ and multiple extensions for languages that have two or more constructors than $L_b$. The methodology is based on a weakening and strengthening approach.

2) Based on the methodology, develop three single extension procedures to extend a fast $ALCH$ reasoner with different constructors: nominals ($O$), inverse roles ($I$), and limited datatypes ($D^-$). The three procedures are used to classify ontologies in languages $ALCHO$, $ALCHI$, and $ALCH(D^-)$, respectively.
(3) Combine the three single extension procedures into four multiple extension procedures for classifying $A\text{LCLI}(D)^{-}$, $A\text{LCHO}(D)^{-}$, $A\text{LCHOI}$ and $A\text{LCHOI}(D)^{-}$ ontologies.

(4) Prove the correctness of the proposed procedures including all the single extensions and multiple extensions. A correct classification procedure must be sound and complete. A procedure is sound if every subsumption it gets is implied by the input ontology $O$. A procedure is complete if every subsumption implied by $O$ can be obtained by the procedure.

(5) Implement the procedures and evaluate their performance. We will implement the procedures and compare their performance with other tableau-based reasoners on classifying large and complex ontologies.

1.3 Thesis Outline

The remainder of the thesis is organized as follows:

In Chapter 2, we present background and related work. We first introduce the relevant knowledge of description logic. Next, we explain the two mainstream reasoning procedures: tableau-based reasoning and consequence-based reasoning. Then we introduce the complexity of classification on different languages. At the end of this chapter we list the related work.

In Chapter 3, we introduce the overall methodology adopted throughout the thesis. We first briefly summarize our approach about single extension, then introduce our
core techniques in this thesis – weakening and strengthening.

In Chapters 4, we illustrate two soundness-preserved single extensions from a base language $\mathcal{ALCH}$ to $\mathcal{ALCH}(D)^-$ and $\mathcal{ALCHI}$. The proofs of soundness and completeness of the two single extensions are also presented.

In Chapter 5, we illustrate a soundness-relaxed single extension from a base language $\mathcal{ALCH}$ to $\mathcal{ALCHO}$. The proof of soundness and completeness of the final result is also provided.

In Chapter 6, we present the general principle of multiple extensions. Then, we explain how to create weakened and strengthened versions of ontologies in multiple extensions. We demonstrate the examples on how to combine three single extensions of $\mathcal{ALCH}(D)^-$, $\mathcal{ALCHI}$ and $\mathcal{ALCHO}$ to do multiple extensions for supporting $\mathcal{ALCHI}(D)^-$, $\mathcal{ALCHO}(D)^-$, $\mathcal{ALCHOI}$ and $\mathcal{ALCHOI}(D)^-$ ontologies. Finally, we prove the soundness and completeness of the multiple extensions.

In Chapter 7, we present the system design and the concrete implementation of our prototype $\mathcal{ALCHOI}(D)^-$-reasoner WSClassifier.

In Chapter 8, we present an empirical evaluation to compare the performance of WSClassifier on $\mathcal{ALCHI}(D)^-$, $\mathcal{ALCHO}$ and $\mathcal{ALCHOI}(D)^-$ ontologies against existing state-of-the-art reasoners and shows the effectiveness of the proposed approaches. We also demonstrate and give a brief analysis on the outcome of the DL classification of the 2013 international OWL Reasoner Performance Competition, in which WSClassifier participated.

In Chapter 9, we first propose some characteristics in two dimensions for evalu-
ating reasoners. Then, based on these characteristics, we evaluate WSClassifier against state-of-the-art reasoners. We also summarize specific characteristics for extension-based reasoners and compare WSClassifier with other extension-based reasoners.

In Chapter 10, we conclude our research and summarize the theoretical and practical contribution of this thesis. We end the thesis with future work.
Chapter 2

Background and Related Work

This chapter is about the background and related work of this research. It contains four sections. Section 2.1 introduces the preliminary knowledge of Description Logic; Section 2.2 and Section 2.3 introduces the core techniques of the two mainstream reasoning approaches – tableau-based and consequence-based reasoning, respectively. Finally, Section 2.4 summarizes the related work w.r.t. our research.

2.1 Description Logic

Description logics (DLs) are a family of logic-based languages that are widely used for representing ontological knowledge. They were first introduced as knowledge representation languages in the mid 1980s. Many DLs can be regarded as decidable fragments of first-order logic [3] and also as syntactic variants of modal logics [72]. Since the emergence of semantic web, DLs become one of the main
underpinnings of the Web Ontology Language (OWL), a standard developed by the World Wide Web Consortium (W3C).

DLs employ three kinds of building blocks to model the relationships between entities within a domain of interest: concepts, roles, and individuals. Individuals denote entities in the domain of interest. Concepts denote sets of individuals sharing common characteristics. Roles denote binary relations between the individuals. An ontology described in DL consists of a set of axioms, which are statements known to be true in the domain. Typically, these axioms reflect only partial knowledge about the domain that the ontology describes, and there may be many different states of the world that are consistent with the ontology. Conventionally, the axioms are separated into two subsets: assertional axioms (ABox) and terminological axioms (TBox). ABox contains facts about individuals like the membership of an individual in a concept or a relationship between two individuals via a role. TBox contains general knowledge about relationships between concepts and roles, and it is a finite set of concept inclusion axioms.

**Concept Inclusion Axiom** Assume \( C, D \) represent concepts. A Concept Inclusion Axiom is an expression of the form:

- \( C \sqsubseteq D \) referred to as a concept subsumption axiom, or

- \( C \equiv D \) referred to as a concept definition axiom which is an abbreviation for \( C \sqsubseteq D \) and \( D \sqsubseteq C \).

An example of TBox and ABox is shown in Table 2.1.

Suppose one wants to build an ontology that models the domain of wine prod-
Table 2.1: Basic DL knowledge base consisting of a TBox and an ABox

<table>
<thead>
<tr>
<th>TBox axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wine ⊑ ∃ hasSugar. WineSugar</td>
</tr>
<tr>
<td>TableWine ≡ Wine ⊓ ∃ hasSugar. {Dry}</td>
</tr>
<tr>
<td>WineSugar ≡ {Dry, OffDry, Sweet}</td>
</tr>
<tr>
<td>DryRiesling ⊑ Wine</td>
</tr>
<tr>
<td>DryRiesling ⊑ ∀ hasSugar. {Dry}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ABox axioms</th>
</tr>
</thead>
<tbody>
<tr>
<td>DryRiesling(PacificRimRieslingDry2012)</td>
</tr>
<tr>
<td>WineSugar(Dry)</td>
</tr>
<tr>
<td>hasSugar(PacificRimRieslingDry2012, Dry)</td>
</tr>
</tbody>
</table>

ucts and the relationship between different types of wine and their ingredients. We can use concepts such as TableWine to represent the set of all entities of table wine, roles such as hasSugar to denote the (binary) relationship between entities of table wine and their sugar, and individual names such as PacificRimRieslingDry2012 and Dry to denote the Pacific Rim dry riesling produced in 2012 and dry sugar, respectively. The ontology might include ABox axioms such as DryRiesling (PacificRimRieslingDry2012), WineSugar(Dry), and hasSugar(PacificRimRieslingDry2012, Dry) to assert PacificRimRieslingDry2012 is a DryRiesling, Dry is a WineSugar, and that a PacificRimRieslingDry2012 has sugar Dry, and TBox axioms such as TableWine ≡ Wine ⊓ ∃ hasSugar.{Dry} to state that TableWine is defined as those wine entities that have at least one type of sugar – Dry.

DLs are equipped with a formal semantics that precisely specifies the meaning of DL ontologies. This semantics allows the exchange of ontologies between hu-
mans and computer systems without ambiguity in their intended meaning. It also enables logical deduction, which infers implicit knowledge from explicitly stated facts and axioms in an ontology. The process of computing inferred knowledge is called reasoning. In the design of DL languages, one of the major targets is to ensure the availability of reasoning algorithms with good performance. This is one of the reasons why there is not just a single DL: the best balance between expressivity of the language and complexity of reasoning depends on the intended application.

In this section, we introduce some of the DLs that are most commonly used in practice, we also list and discuss the DLs that are used later in this thesis. Finally, we describe the relationship of DL to OWL.

### 2.1.1 Syntax and Semantics

In this subsection we give the definition of the syntax and the semantics of a DL named $\text{SROIQ}(\mathcal{D})$. This DL covers the expressivity of the OWL 2 standard and is one of the most expressive DLs commonly considered today. $\text{SROIQ}(\mathcal{D})$ has a parameter $\mathcal{D}$, defined in the following:

**Definition 2.1.1.** A datatype map $\mathcal{D}$ is a tuple $(N_{DT}, N_{LS}, N_{FS}, \cdot^{D})$, where

- $N_{DT}$ is a set of datatype names;
- $N_{LS}$ is a function assigning to each $d \in N_{DT}$ a set of constants $N_{LS}(d)$;
- $N_{FS}$ is a function assigning to each $d \in N_{DT}$ a set of facets $N_{FS}(d)$, and each
$f \in N_{FS}(d)$ has the form $(p_f, v)$. A facet is a restriction to a data type, e.g., minExclusive($>$), maxInclusive($\leq$) are facets of Real datatype.

- $\cdot^D$ is a function assigning a datatype interpretation $d^D$ to each $d \in N_{DT}$ called the value space of $d$, a data value $v^D \in d^D$ for each $v \in N_{LS}(d)$, and a facet interpretation $f^D$ for each $f \in \bigcup_{d \in N_{DT}} N_{FS}(d)$.

On top of a datatype map $D$ data ranges – expressions representing a set of data values in $D$ – can be defined. An atomic data range, denoted by $ADR$ in the thesis, is of the form $\top_D, d, d[f]$ or $\{v\}$, where $v \in N_{LS}(d)$. A data range $dr$ is either an atomic data range or defined recursively using $\cap$, $\cup$, and $\neg$. Their interpretations are shown in Table 2.2.

<table>
<thead>
<tr>
<th>Semantics of Data Ranges</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(\top_D)^D = \Delta^D$</td>
</tr>
<tr>
<td>${v}^D = {v^D}$</td>
</tr>
<tr>
<td>$(d \cap f)^D = d^D \cap f^D$</td>
</tr>
<tr>
<td>$(\neg dr)^D = \Delta^D \setminus dr^D$</td>
</tr>
<tr>
<td>$(dr_1 \cup dr_2)^D = dr_1^D \cup dr_2^D$</td>
</tr>
</tbody>
</table>

The syntax of $SROIQ(D)$ uses mutually disjoint sets of atomic concepts $N_C$, atomic roles $N_R$, individuals $N_I$, and features $N_F$. The union of $N_C$, $N_R$, $N_I$, and $N_F$ in the ontology $O$ is called the signature of $O$, denoted by $\text{Sig}(O)$.

The set of concepts and roles of $SROIQ(D)$ are recursively defined from constructors listed in Table 2.3, where $\bot$ is the bottom concept, $A$ is an atomic concept, $C$ and $D$ are concepts, $r$ is an atomic role, $R$ is a role, $F$ is a feature, $a$ is an individual,

\[\text{Since one facet may be shared by multiple datatypes, we define its interpretation as containing subsets of all the relevant datatypes.}\]
$n$ is a nonnegative integer, and $dr$ is a data range, which will be explained later. In the remaining of this thesis, we use $A$ and $B$ for atomic concepts, $C$ and $D$ for concepts, $r$ and $s$ for atomic roles, $R$ and $S$ for roles, $a$ and $b$ for individuals, and $F$ and $G$ for features.

An $SROIQ(D)$ ontology consists of three groups of axioms: (1) terminological axioms (TBox) $T$, which contains concept inclusions $C \sqsubseteq D$ and concept equivalence $C = D$; (2) relational axioms (RBox) $R$, which contains role inclusions $R \sqsubseteq S$, role equivalence $R = S$, role disjointness $\text{Disj}(R, S)$ and complex role inclusions $R_1 \circ R_2 \sqsubseteq S$; (3) assertional axioms (ABox) $A$, which contains assertions in the forms of $C(a)$, $R(a, b)$, $a \approx b$, and $a \neq b$, where $\approx$ and $\neq$ represent equality and inequality relationships, respectively.

The semantics of $SROIQ(D)$ is defined via an interpretation $I = (\Delta^I, \Delta^D, \cdot^I, \cdot^D)$, where $\Delta^I$ and $\Delta^D$ are disjoint non-empty sets called object domain and data domain. $\Delta^D$ satisfies $\Delta^D \supseteq d^D$ for each $d \in N_{DT}$. The data interpretation function $\cdot^D$ will be described in the following paragraph. The object interpretation function $\cdot^I$ assigns an element $a^I \in \Delta^I$ to every individual $a \in N_I$, a set $A^I \subseteq \Delta^I$ to each $A \in N_C$, a relation $R^I \subseteq \Delta^I \times \Delta^I$ to each $R \in N_R$, and a relation $F^I \subseteq \Delta^I \times \Delta^D$ to each $F \in N_F$. The interpretation of all the constructors are defined in Table 2.3. If the language of an ontology does not use datatype maps, $\Delta^D$ and $\cdot^D$ does not exist, and the interpretation is represented by $(\Delta^I, \cdot^I)$. An interpretation $I$ is a model of an ontology $O$ if it satisfies all the axioms in $O$ according to the semantics in Table 2.4.

An ontology is consistent if it has at least one model, otherwise it is inconsistent.
We say that an axiom is a consequence of an ontology \( O \), or also that \( O \) entails \( \alpha \) (written \( O \models \alpha \)), if every model of \( O \) satisfies \( \alpha \). Note that an inconsistent ontology entails every axiom. A concept \( C \) is unsatisfiable w.r.t. \( O \) if \( O \models C \sqsubseteq \bot \), otherwise \( C \) is satisfiable w.r.t. \( O \). A concept \( C \) is subsumed by \( D \) w.r.t. \( O \) if \( O \models C \sqsubseteq D \). Concepts \( C \) and \( D \) are equivalent w.r.t. \( O \) if \( O \models C \equiv D \). An individual \( a \) is an instance of a concept \( C \) w.r.t. \( O \) if \( O \models C(a) \).

A general reasoning problem in DLs is checking entailment of axioms from ontologies: given an ontology \( O \) and an axiom \( \alpha \), check if \( O \models \alpha \). If both \( O \) and \( \alpha \) consist only of terminological axioms, we speak about terminological reasoning. In case \( \alpha \) is a concept inclusion \((\alpha = C \sqsubseteq D)\), the problem is known as subsumption checking. Many other reasoning problems can be reduced to subsumption checking. In any DL that supports \( \top \) and \( \bot \), consistency checking \((\top \sqsubseteq \bot)\) and satisfiability checking \( (\alpha = C \sqsubseteq \bot) \) are just special cases of subsumption checking. In any DL that supports existential restrictions, entailment of a role inclusion \( R \sqsubseteq S \) can be checked by checking the subsumption \( \exists R.A \sqsubseteq \exists S.A \) where \( A \) is a new atomic concept not occurring in the ontology (see, e.g., [25]).

In practice one often does not check entailment of a single axiom, but performs a reasoning task that consists of checking multiple entailments at once. The goal of the ontology classification task is to compute the taxonomy: an acyclic graph representing direct subsumptions between equivalence classes of atomic concepts occurring in \( O \).

A correct classification procedure must be sound and complete. We use \( \mathcal{H}_O \) to denote the classification result of \( O \).
Table 2.3: Syntax and semantics of $SROIQ(D)$ constructors

<table>
<thead>
<tr>
<th>Constructor</th>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Individuals:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>individual name</td>
<td>$a$</td>
<td>$a^I$</td>
</tr>
<tr>
<td>Roles:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atomic role</td>
<td>$r$</td>
<td>$r^I$</td>
</tr>
<tr>
<td>inverse role</td>
<td>$R^-$</td>
<td>${(y,x)</td>
</tr>
<tr>
<td>universal role</td>
<td>$U$</td>
<td>$\Delta^I \times \Delta^I$</td>
</tr>
<tr>
<td>Concepts:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>atomic concept</td>
<td>$A$</td>
<td>$A^I$</td>
</tr>
<tr>
<td>top</td>
<td>$\top$</td>
<td>$\Delta^I$</td>
</tr>
<tr>
<td>bottom</td>
<td>$\bot$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>negation</td>
<td>$\neg C$</td>
<td>$\Delta^I \setminus C^I$</td>
</tr>
<tr>
<td>conjunction</td>
<td>$C \sqcap D$</td>
<td>$C^I \cap D^I$</td>
</tr>
<tr>
<td>disjunction</td>
<td>$C \sqcup D$</td>
<td>$C^I \cup D^I$</td>
</tr>
<tr>
<td>existential restriction</td>
<td>$\exists R.C$</td>
<td>${x</td>
</tr>
<tr>
<td>universal restriction</td>
<td>$\forall R.C$</td>
<td>${x</td>
</tr>
<tr>
<td>nominal</td>
<td>${a}$</td>
<td>${a^I}$</td>
</tr>
<tr>
<td>at-most restriction</td>
<td>$\leq nR.C$</td>
<td>${x</td>
</tr>
<tr>
<td>at-least restriction</td>
<td>$\geq nR.C$</td>
<td>${x</td>
</tr>
<tr>
<td>local reflexivity</td>
<td>$\exists R.Self$</td>
<td>${x</td>
</tr>
<tr>
<td></td>
<td>$\exists F.dr$</td>
<td>${x</td>
</tr>
<tr>
<td></td>
<td>$\forall F.dr$</td>
<td>${x</td>
</tr>
</tbody>
</table>

Definition 2.1.2. A classification procedure is sound if for every $A, B \in N_C \cap \text{Sig}(O)$, and $A \subseteq B \in H_O$, $O \models A \sqsubseteq B$.

Definition 2.1.3. A classification procedure is complete if for every $A, B \in N_C \cap \text{Sig}(O)$ and $O \models A \sqsubseteq B$, $A \sqsubseteq B \in H_O$.

2.1.2 The DL Family

In order to meet the expressivity needs of certain application domains, various DL constructors have been investigated in terms of expressivity and decidabil-
Table 2.4: Syntax and semantics of $SROIQ(\mathcal{D})$ axioms

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>TBox:</td>
<td></td>
</tr>
<tr>
<td>concept inclusion</td>
<td>$C \sqsubseteq D$</td>
</tr>
<tr>
<td>concept equivalence</td>
<td>$C = D$</td>
</tr>
<tr>
<td>RBox:</td>
<td></td>
</tr>
<tr>
<td>role inclusion</td>
<td>$R \sqsubseteq S$</td>
</tr>
<tr>
<td>role equivalence</td>
<td>$R = S$</td>
</tr>
<tr>
<td>complex role inclusion</td>
<td>$R_1 \circ R_2 \sqsubseteq S$</td>
</tr>
<tr>
<td>role disjointness</td>
<td>$\text{Disj}(R,S)$</td>
</tr>
<tr>
<td>ABox:</td>
<td></td>
</tr>
<tr>
<td>concept assertion</td>
<td>$C(a)$</td>
</tr>
<tr>
<td>role assertion</td>
<td>$R(a,b)$</td>
</tr>
<tr>
<td>equality assertion</td>
<td>$a \approx b$</td>
</tr>
<tr>
<td>inequality assertion</td>
<td>$a \neq b$</td>
</tr>
</tbody>
</table>

ity of their corresponding inference services. Different DLs are defined through the types of constructs and axioms allowed. The corresponding languages are identified by a string of the form $\mathcal{ALC}[S][R][H][O][I][Q][\mathcal{D}]$, where every letter stands for a certain constructor. $\mathcal{ALC}$ stands for basic DL. In paper [73], it was shown that $\mathcal{ALC}$ is a syntactic variant of modal logic K [3], where all roles are atomic and complex concepts can be built using boolean operators ($\sqcap, \sqcup, \neg$), universal restriction($\forall$), and existential($\exists$)value restriction on atomic concepts as follows:

$\mathcal{ALC}\text{concepts} \rightarrow \top | \bot | A | \neg A | (C \sqcap D) | (C \sqcup D) | (\exists R.C) | (\forall R.C)$

One of the most expressive DL languages is $SROIQ(\mathcal{D})$ [42], which is now the DL underlying OWL2 [62], extends $\mathcal{ALC}$ with role inclusions $\mathcal{H}$, the combination of complex role inclusions, role characteristics, and local reflexivity $\mathcal{R}$, nominals $\mathcal{O}$, inverse roles $\mathcal{I}$, and qualified cardinality restrictions $\mathcal{Q}$. The letter
\( S \) denotes the extension of \( \mathcal{ALC} \) with transitive roles \(^{[43]}\). Furthermore, each logic can be extended with a number of datatypes such as booleans, integers, and strings. This is commonly denoted by the letter \( D \) in parentheses. On the other hand, a less expressive fragment of \( \mathcal{ALC} \), the DL \( \mathcal{EL} \) \(^{[6]}\), has drawn considerable attention for modelling large scale biomedical KBs. Each logic can also be considered with or without ABox axioms. This thesis only considers extensions of \( \mathcal{ALCH} \) enabling \( D, I \) and \( O \). We have formally introduced \( D, I \) and \( O \) in section \(^{2.1.1}\). Here we will explain these three constructors intuitively.

**Datatypes** \((D)\) Datatypes are used to represent literal values such as numbers and strings. They can be used to describe concepts such as ”an infant is a child whose age is between 0 and 1” \((\text{Child} \sqcap \exists \text{hasAge}. (\text{max}1) \sqcap \text{hasAge}. (\text{min}0))\) where \((\text{max}1)\) and \((\text{min}0)\) are datatypes derived by adding minimum and maximum value constraints on an integer datatype.

**Inverse Roles** \((I)\) Inverse Roles are used to express converse relations between individuals using \(-\) operator. For example, \((\text{hasComponent} = \text{isComponentOf}^-)\) can be used to express that \(\text{hasComponent} \) is the converse relation of \(\text{isComponentOf} \).

**Nominals** \((O)\) Nominals, known as named individuals, are studied in the areas of hybrid logics \(^{[9]}\) as well as DLs \(^{[71]}\). Nominals play an important role in DL as they allow one to express the notion of uniqueness and identity; nominals must
be interpreted as singleton sets. Many ontologies, for instance in the geospatial domain, use nominals as names for persons, countries, colours, etc. For example, “God”, “University of New Brunswick”, etc, each of such concept can have only one identical instance, so they need to be modelled using nominals. Take “God” as an example, its nominal axiom is expressed as \( \text{God} \equiv \{\text{god}\} \), i.e., concept \( \text{God} \) has only one individual which is \( \text{god} \). In all DLs without nominals, the presence of ABox axioms in an ontology can have no effect on the entailment of TBox axioms apart from possibly making the whole ontology inconsistent.

In this thesis, our base language is \( \text{ALCH} \). We single-extend it to \( \text{ALCH(D)}^- \), \( \text{ALCHI} \) and \( \text{ALCHO} \), and multiple-extend it to \( \text{ALCHI(D)}^- \), \( \text{ALCHO(D)}^- \), \( \text{ALCHOI} \) and \( \text{ALCHOI(D)}^- \) based on the three single extensions, see Table 2.5. The reason we select \( \text{ALCH} \) as the base language is that \( \text{ALCH} \) contains the very critical constructor–disjunction. Disjunction cannot be supported by consequence-based reasoners except the \( \text{ALCH} \) reasoner ConDOR. Our goal in this research is for practical reasoning. It aims to obtain as many subsumption relationships as possible in an acceptable time such as in minutes rather than having complete result with unacceptable time such as in hours or days. So we will try to extend the \( \text{ALCH} \) language to more expressive languages which contains some significant constructors that frequently appear in the common ontologies and may greatly affect the final classification result. We do not pursue to extend the base language to the full OWL2 language \( \text{SROIQ(D)} \). Some constructors such as number restriction \( Q \) and the combination of complex role inclusions, role char-
Table 2.5: Part of DLs considered in this thesis

<table>
<thead>
<tr>
<th>Logic</th>
<th>$\mathcal{ALCH}$</th>
<th>$\mathcal{ALCH}(D)^-$</th>
<th>$\mathcal{ALCHI}$</th>
<th>$\mathcal{ALCHO}$</th>
<th>$\mathcal{ALCHOI}(D)^-$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roles</td>
<td>$R$, $\top$, $\bot$, $\neg\top$, $\neg\bot$</td>
<td>$R$, $R^-$</td>
<td>$R$, $R^-$</td>
<td>$R$, $R^-$</td>
<td>$R$, $R^-$</td>
</tr>
<tr>
<td>Concepts</td>
<td>$C \sqcap D$, $C \sqcup D$</td>
<td>$C \sqcap D$, $C \sqcup D$, $D \sqcap C$, $D \sqcup C$</td>
<td>$C \sqcap D$, $C \sqcup D$, $D \sqcap C$, $D \sqcup C$</td>
<td>$C \sqcap D$, $C \sqcup D$, $D \sqcap C$, $D \sqcup C$</td>
<td>$C \sqcap D$, $C \sqcup D$, $D \sqcap C$, $D \sqcup C$</td>
</tr>
<tr>
<td>Axioms</td>
<td>$C \sqsubseteq D$, $R \sqsubseteq S$</td>
<td>$C \sqsubseteq D$, $R \sqsubseteq S$</td>
<td>$C \sqsubseteq D$, $R \sqsubseteq S$</td>
<td>$C \sqsubseteq D$, $R \sqsubseteq S$</td>
<td>$C \sqsubseteq D$, $R \sqsubseteq S$</td>
</tr>
<tr>
<td>Complexity</td>
<td>ExpTIME same</td>
<td>same</td>
<td>same</td>
<td>same</td>
<td>same</td>
</tr>
</tbody>
</table>

characteristics, and local reflexivity $R$, they do not occur very often in the ontologies, but the complexity of language with them will increase dramatically. See section 9.1 Datatypes, inverse roles and nominals are frequently used. For ontologies in language $\mathcal{ALCH}(D)^-$, $\mathcal{ALCHI}$, $\mathcal{ALCHO}$, $\mathcal{ALCHI}(D)^-$, $\mathcal{ALCHO}(D)^-$, $\mathcal{ALCHOI}$ and $\mathcal{ALCHOI}(D)^-$, as we have mentioned, no consequence-based reasoners can fully support these languages, while the tableau-based reasoners can support them. If the ontology is large and highly cyclic, it may require an unacceptably long time for the tableau-based reasoners to classify them. Thus our goal in this thesis is to take these languages as examples, investigate new approach to classify ontologies in these languages and obtain sound and complete result in acceptable time.

2.1.3  Relationship to OWL

The Web Ontology Language OWL is a knowledge representation language standardized by the World Wide Web Consortium (W3C). OWL is one of the lan-
guages most commonly used on the Semantic Web. The current version of the
OWL specification is OWL 2 [92]. OWL is one of the most important appli-
cations of DLs today. In this section we briefly outline the relationship of the
two languages. The main building blocks of OWL are very similar to those of
DLs, with the main difference being that concepts are called classes and roles
are called properties. Historically, however, OWL has also been conceived as an
extension of RDF [56], a Web data modeling language whose expressivity is com-
parable to DL ABoxes. The formal semantics of RDF [37] is subtly different from
that of DLs, even though both lead to the same consequences in many common
cases. For this reason, there are two styles of formal semantics for OWL: the
Direct Semantics [60] based on DLs and the RDF-Based Semantics [37]. The Di-
rect Semantics of OWL is only defined for a certain syntactic fragment of OWL
called OWL DL. In contrast, the OWL language without any syntactic constraints
is called OWL Full; it comprises ontologies that can only be interpreted under
the RDF-based Semantics. Under the Direct Semantics, large parts of OWL DL
can indeed be considered as a syntactic variant of DLs. For example, the axiom
\( \text{Mother} \equiv \text{Female} \cap \text{Parent} \) would be written as follows in the OWL Functional
Style Syntax:
EquivalentClasses( Mother ObjectIntersectionOf( Female Parent ) )
where the symbols Mother, Female, and Parent would be identifier strings that
conform to the OWL specification which is based on Uniform Resource Identifi-
cers (URIs). The above example illustrates the close relationship between the
syntax of DLs and that of OWL. In many cases, it is enough to translate an oper-
ator symbol of DLs into the corresponding operator name in OWL, which is then written in prefix notation like a function. This is also why the above form of syntax is called Functional-Style Syntax [61]. The OWL standard provides a number of syntactic forms that can be used to express OWL ontologies. The most prominent among these is the RDF/XML Syntax [8] since it is the only format that all conforming OWL tools need to support. On the other hand, it is more difficult for humans to read, and we do not present it here. The OWL API [38], a Java API for manipulating and reasoning with OWL ontologies, provides parsers and writers for all standard OWL syntaxes as well as interfaces for OWL reasoners. Popular reasoners for large parts of OWL DL include FaCT++ [91], HermiT [63], Pellet [81], and RacerPro [34]. Up-to-date lists of current OWL reasoners are best found online.

The expressivity of OWL DL corresponds approximately to the DL SROIQ(D) [33]. The OWL standard defines three fragments (also known as profiles) [58] that trade expressive power for favorable computation properties. These are closely related to and have similar properties as the three families of light-weight DLs that we discussed above: OWL EL is based on the EL family of DLs, OWL QL (for Query Language) is based on the DL-Lite family of DLs, and OWL RL (for Rule Language) is based on Description Logic Programs. OWL also provides a number of non-logical features that are not considered in DLs. These include the ability to give a name and a version to an ontology, import axioms from one ontology into another, declare identifiers, and annotate axioms and entities with extra information, e.g., provenance.
2.2 Tableau Based Classification

(Hyper) Tableau-based techniques classify an ontology using (1) a model construction procedure which builds a model to check the satisfiability of a given concept, e.g. a model for \( A \cap \neg B \) for testing whether \( O \models A \sqsubseteq B \) holds; (2) a classification procedure which selectively tests the satisfiability of certain concepts using the model construction procedure in order to obtain \( \mathcal{H}_O \). Thus the total classification time is determined by (1) the efficiency of every satisfiability test and (2) the number of subsumption tests performed in the classification procedure. Next we will discuss these two procedures and their performance.

2.2.1 Model Construction Procedure

In an ontology \( O \), checking if \( O \models A \sqsubseteq B \) is equivalent to checking whether the concept \( A \cap \neg B \) is unsatisfiable. That is to say, if a model cannot be constructed for \( O \cup \{A(s_0), \neg B(s_0)\} \) with some fresh individual \( s_0 \notin N_I \cap \text{Sig}(O) \), we can conclude that \( O \models A \cap \neg B \sqsubseteq \bot \), and \( O \models A \sqsubseteq B \) holds. To build a model for an ontology, a tableau algorithm expands the concept and role assertions in its ABox \( \mathcal{A} \) by repeatedly applying derivation rules until either of the two cases occurs: 1) no more rules can be applied; 2) an obvious contradiction occurs. The purpose of applying derivation rules is to try to satisfy the axioms in \( O \), and thus evolve the initial assertions towards a (representation of a) model of the ontology. In case 1), the model construction succeeds and we obtain a pre-model, indicating that the ontology is satisfiable. In case 2), the model cannot be constructed and
the ontology is unsatisfiable. Table 2.6 shows some examples of derivation rules commonly used in DL tableau calculi.

<table>
<thead>
<tr>
<th>Rule</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>⊔-rule</td>
<td>Given ((C_1 \sqcup C_2)(s)), derive either (C_1(s)) or (C_2(s))</td>
</tr>
<tr>
<td>⊓-rule</td>
<td>Given ((C_1 \sqcap C_2)(s)), derive (C_1(s)) and (C_2(s))</td>
</tr>
<tr>
<td>∃-rule</td>
<td>Given ((\exists R.C)(s)), derive (R(s,t)) and (C(t)) for (t) a fresh individual</td>
</tr>
<tr>
<td>∀-rule</td>
<td>Given ((\forall R.C)(s)) and (R(s,t)), derive (C(t))</td>
</tr>
</tbody>
</table>

The \(\sqcup\)-rule is nondeterministic: if \((C_1 \sqcup C_2)(s)\) is true, then \(C_1(s)\) or \(C_2(s)\) or both are true. Therefore, tableau calculi make a nondeterministic guess and choose either \(C_1\) or \(C_2\); if one choice leads to a contradiction, the algorithm must backtrack and try the other choice. The ontology is unsatisfiable only if all choices lead to a contradiction \[63\]. In this case, the calculi may need to backtrack many branches; it is one of the important reasons why (hyper)tableau calculus is inefficient.

2.2.2 Classification Procedure and Optimization

Given a sound and complete satisfiability test procedure, a naive algorithm to classify \(O\) is to check whether \(O \models A \sqsubseteq B\) holds for every pair \(\langle A, B \rangle\) in \(O\). However, if the number of the atomic concepts in \(O\) is \(n\), then such a method needs \(n^2\) tests and is inefficient already for medium-sized ontologies. ET \[5\] and KP \[25\] are two prominent algorithms designed for reducing the number of tests. The ET algorithm builds the concept hierarchy \(\mathcal{H}_O\) for \(O\) by first initializing it with \(\{\langle \bot, \top \rangle\}\) and then inserting every concept \(A\) into the hierarchy. The process employs a top-down search followed by a bottom-up search. Each insertion step
typically requires one or more subsumption tests. This algorithm significantly reduces the number of subsumption tests from the theoretical upper bound of \( n^2 \).

For example, in the top-down phase, if \( O \not\models C \sqsubseteq D \), then the algorithm does not need to check \( C \) against the subclasses of \( D \). Nevertheless, it is still not efficient when the ontologies have a large number of leaf nodes, since in the bottom-up phase, many concepts may need to be checked against these nodes.

In contrast to ET, the KP algorithm does not build hierarchies directly, but maintains a set of known subsumptions \( K \) and a set of remaining possible subsumptions \( P \). The algorithm performs subsumption tests to augment \( K \) and reduce \( P \) until \( K \) contains all the subsumptions of \( O \) and \( P = \emptyset \). Such a representation of the hierarchy allows one to manipulate \( K \) and \( P \) using highly-tuned algorithms, for example the ones that compute the transitive closure and the transitive reduction of a relation. Furthermore, KP also exploits the transitivity of the subclass relation to propagate (non)-subsumption information and thus speeds up the process of augmenting \( K \) and reducing \( P \). The implementation of KP algorithms in the OWL reasoner HermiT has shown significant performance improvements over ET.

### 2.2.3 Blocking Techniques for Reasoning on Cyclic Ontologies

As we mentioned in Section 1.1, cyclic ontologies turn out to be difficult for tableau-based reasoners. This is because a naive way of expanding the model for cyclic ontologies may not terminate, and a relatively expensive cycle detection technique called *blocking* needs to be applied to ensure termination. The follow-
The ontology given above is cyclic, since the concept \textit{Lung} is defined in terms of itself transitively through concepts \textit{PulmonaryVein}, \textit{Mediastinum}, and \textit{PulmonaryArtery}. Now let us see why a “naive” model expansion procedure cannot terminate. Note that all the axioms are of the form $A \sqsubseteq \exists R.B$, so that the derivation process expands the ABox $\mathcal{A}$ using only two types of derivations: (1) If $A(x) \in \mathcal{A}$ and $\exists R.B(x) \notin \mathcal{A}$, then add $\exists R.B(x)$ to $\mathcal{A}$; (2) if $\exists R.B(x) \in \mathcal{A}$ and there is no $y$ such that $R(x,y) \in \mathcal{A}$ and $B(y) \in \mathcal{A}$, then create a fresh instance $y$ which is not in $\mathcal{A}$, and add $R(x,y)$ and $B(y)$ to $\mathcal{A}$.

\begin{align*}
\text{Lung} & \sqsubseteq \exists \text{hasStructuralComponent}.\text{PulmonaryVein} & (6) \\
\text{Lung} & \sqsubseteq \exists \text{isServedBy}.\text{PulmonaryVein} & (7) \\
\text{PulmonaryVein} & \sqsubseteq \exists \text{isNonPartitivelyContainedIn}.\text{Mediastinum} & (8) \\
\text{Mediastinum} & \sqsubseteq \exists \text{nonPartitivelyContains}.\text{PulmonaryArtery} & (9) \\
\text{PulmonaryArtery} & \sqsubseteq \exists \text{Serve}.\text{Lung} & (10) \\
\text{PulmonaryArtery} & \sqsubseteq \exists \text{isStructuralComponentOf}.\text{Lung} & (11)
\end{align*}
Here is part of the derivation sequence starting with $\mathcal{A}_0 = \{\text{Lung}(a_0)\}$:

\[
\begin{align*}
\mathcal{A}_1 &= \mathcal{A}_0 \cup \{\exists \text{hasStructuralComponent}.\text{PulmonaryVein}(a_0), \\
&\quad \exists \text{isServedBy}.\text{PulmonaryVein}(a_0)\} \\
\mathcal{A}_2 &= \mathcal{A}_1 \cup \{\text{hasStructuralComponent}(a_0, b_0), \text{PulmonaryVein}(b_0), \\
&\quad \text{isServedBy}(a_0, b_1), \text{PulmonaryVein}(b_1)\} \\
\mathcal{A}_3 &= \mathcal{A}_2 \cup \{\exists \text{isNonPartitivelyContainedIn}.\text{Mediastinum}(b_0), \ldots\} \\
\mathcal{A}_4 &= \mathcal{A}_3 \cup \{\exists \text{isNonPartitivelyContainedIn}(b_0, c_0), \text{Mediastinum}(c_0), \ldots\} \\
&\ldots \\
\mathcal{A}_6 &= \mathcal{A}_5 \cup \{\text{nonPartitivelyContains}(c_0, d_0), \text{PulmonaryArtery}(d_0), \ldots\} \\
&\ldots \\
\mathcal{A}_8 &= \mathcal{A}_7 \cup \{\text{Serve}(d_0, a_1), \text{Lung}(a_1), \\
&\quad \text{hasStructuralComponent}(d_0, a_2), \text{Lung}(a_2), \ldots\}
\end{align*}
\]

As is shown, two new instances $a_1$ and $a_2$ of $\text{Lung}$ are added to $\mathcal{A}_8$ in the ABox expansion. The derivations from $\mathcal{A}_0$ to $\mathcal{A}_8$ will repeat forever, and the naive algorithm will not terminate. In order to ensure termination, we need to block instances $a_1$ and $a_2$ by $a_0$, meaning that the expansion for $a_1$ is exactly a repetition of what has been or will be done for $a_0$. Here $a_1$ and $a_2$ are called blocked nodes\footnote{we interchangeably use “instance” and “node” to refer to an individual in the ABox} and $a_0$ is called a blocker. Once $a_1$ and $a_2$ are blocked, all the derivations for them and their successors are prohibited. Such techniques ensure termination of the ABox expansion process. The resulting ABox with blocked instances is a pre-model of $O$, and it can be unraveled to a model $O$ by copying the tree of successors.
from every blocker to its relevant blocked nodes.

If $O$ uses a less expressive language, checking whether one instance $t$ can be a blocker for $s$ requires a comparison of all their labels, i.e. all the concepts that they belong to. If $O$ is expressive, a comparison of their predecessors’ labels are also needed. Thus finding a blocker for a node requires a lot of computation, even if some indexing techniques are used for enhancing efficiency. Moreover, such a blocker-finding process needs to be done for every node after every derivation, since a node $t$ may become a blocker for $s$ only after several derivations has been done, therefore the blocker-finding is very time consuming.

In the early years, pairwise blocking techniques were used, in which the checking condition is very strong and a potential blocker $t$ for $s$ may become valid long after $s$ has been created and a lot of derivations have been for $s$, causing a great waste of time and constructing a large model. With the development of the HermiT reasoner, blocking strategies were improved by using core blocking \cite{26}, where a weaker condition is used, leading to a earlier stopping of the model construction process. However, it is not guaranteed that the constructed pre-model can be unraveled to a model. Thus, a validation process is needed in the following stages of the algorithm to ensure the validity of each blocking. Therefore, the reasoner still spends a lot of time in validation. That is why this optimization is still not enough for handling some highly cyclic ontologies.
2.2.4 Implementation

The main reasoners using tableau based techniques include HermiT, Fact++, Pellet and RacerPro. All of them employ various optimization techniques to ensure practical performance. Table 2.7 shows the performance of the three reasoners on some representative ontologies. For about half of the ontologies, their performance is within several minutes, and for many of them it is within several seconds. Some versions of FMA and Galen ontologies are highly cyclic, which turn out to be difficult in different ways for all the tableau reasoners.

Table 2.7: Classification Time of Several Tableau Reasoners

<table>
<thead>
<tr>
<th>Ontology Name</th>
<th>Classification Times(seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>HermiT</td>
</tr>
<tr>
<td>EMap(Feb09)</td>
<td>1.1</td>
</tr>
<tr>
<td>GO Term DB(Feb06)</td>
<td>1.3</td>
</tr>
<tr>
<td>DLP ExtDnS 397</td>
<td>1.3</td>
</tr>
<tr>
<td>LUBM(one university)</td>
<td>1.7</td>
</tr>
<tr>
<td>Biological Process(Feb09)</td>
<td>1.8</td>
</tr>
<tr>
<td>MGED Ontology</td>
<td>2.1</td>
</tr>
<tr>
<td>RNA With Individuals(Dec09)</td>
<td>2.7</td>
</tr>
<tr>
<td>NCI Thesaurus(Feb09)</td>
<td>58.2</td>
</tr>
<tr>
<td>OBI(Mar10)</td>
<td>150.0</td>
</tr>
<tr>
<td>FMA Lite(Feb09)</td>
<td>211.1</td>
</tr>
<tr>
<td>FMA-constitutional part(Feb06)</td>
<td>1638.3</td>
</tr>
<tr>
<td>GALEN-doctored</td>
<td>1.8</td>
</tr>
<tr>
<td>GALEN-undoctored</td>
<td>6.7</td>
</tr>
<tr>
<td>GALEN-module1</td>
<td>out of mem</td>
</tr>
<tr>
<td>GALEN-full</td>
<td>out of mem</td>
</tr>
</tbody>
</table>
2.3 Consequence-Based Classification

Despite the wide expressivity of OWL DL, the reasoning problem becomes more difficult when some advanced constructors interact with each other. Starting from Brandt [10], languages with less expressive power but high reasoning performance have become an important research topic. Investigations of these tractable DL fragments such as $\mathcal{EL}$ [10], $\mathcal{ELH}$ [10] and $\mathcal{EL}^{++}$ [6] led to the discovery of another type of reasoning procedure called consequence-based procedures.

2.3.1 Core Techniques and Procedure

Instead of enumerating pairs of classes and building counter-models for candidate subsumptions, consequence-based procedures derive subsumption relations directly using inference rules. In this procedure, the original ontology $O$ is first normalized. Then, while reasoning, a resulting ontology $O'$ is initialized by some tautological concept inclusions, such as $A \sqsubseteq A$ and $A \sqsubseteq \top$. In each round, all the inference rules are applied to axioms in $O$ and $O'$, and new subsumption consequences are added into $O'$. This process iterates until no more subsumption relationships are found. The resulting ontology $O'$, which contains all the subsumption relationships of $O$, is called a saturation of $O$. This kind of approach has two advantages: (1) the reasoning process is goal-oriented, meaning that the search can be limited to computing all of the superclasses of a given concept; (2) the subsumption relationships are computed “all at once” for the goal, which means all of its superclasses are obtained after one pass of the saturation process.
Current consequence-based reasoner implementations include CEL [6,7], CB [46], ELK [49], ConDOR [78], and TrOWL [69]. The languages supported by these reasoners are $\mathcal{EL}^+$, Horn-$SHIQ$, $\mathcal{EL}H_{R+}$, $SH$, and $SROIQ$ respectively. In CEL, CB, ELK, and ConDOR, the classification results are complete within their specified languages. In contrast, TrOWL employs an approximation approach to encode $SROIQ$ ontologies into $\mathcal{EL}^{++}$ with additional data structures, and the completeness of classification results are not ensured. According to experiments [46, 69, 78] and our previous experiment results shown in Table 1.2, these reasoners significantly outperform other types of reasoners.

The consequence-based procedures are fast in that (1) the procedure is designed to achieve optimal complexity of classification tasks in target languages; (2) all the derived subsumptions can be easily reused in future derivations. In contrast, in the tableau model construction procedure, reusing fragments of models is much more difficult.

Extending consequence-based reasoning techniques to include new constructors is not an easy task for three reasons.

1. One needs to carefully analyze how the new constructors will interact with existing constructors and how to ensure completeness.

2. For certain constructors, like union, universal quantification, and nominals,
which cause an increase in the complexity of the classification problem, the form of derived axioms need to be extended or changed, leading to a complete change of inference rules. For example, in $\mathcal{EL}$ the derived axioms have two forms $A \sqsubseteq B$ and $A \sqsubseteq \exists R. B$. However, when the language is extended to $\mathcal{ALCH}$ with the \texttt{\forall}, $\sqcup$, and more constructors, the forms of the derived axioms have been changed to $H \sqsubseteq M$ and $H \sqsubseteq N \sqcup \exists R.K$. The significant differences between the derived axioms of $\mathcal{ALCH}$ and the derived axioms of $\mathcal{EL}$ are that the new axioms contain both conjunctions and disjunctions, where $H$ and $K$ are conjunctions, and $M$ and $N$ are disjunctions.

3. When new constructors are added, reasoning time and memory usage of the new procedure may increase dramatically. If further optimization strategies cannot be found, then it will no longer be efficient. However, finding the effective optimization strategy may be difficult. For example, the complete inference rules for $\mathcal{ELO}$ were presented in 2005 [6, 7], but unfortunately, the intermediate inference result is prohibitively large in many practical cases. In 2011, Kazakov [45] found some ways to optimize the algorithm, which allow consequence-based reasoners to practically classify $\mathcal{ELO}$ ontologies with efficiency.

2.3.2 Implementation

The currently most active and competitive $\mathcal{EL}$ reasoner ELK[51] which is an exemplary OWL2 EL profile reasoner, supports nominals and part of numeric data
types in Horn features. But these two features of nominals and data types have not been extended to non-Horn consequence-based reasoning up to now. For inverse role, it is supported by Horn $SHIQ$ consequence-based reasoner CB. Besides published open-source consequence-based $ALCH$ reasoner ConDOR, Frantisek etc. also proved theoretically that they could classify $ALCI$ ontology through fixed parameter tractable reasoning via decomposition [79], and this result is applicable to $SHI$ as well. For the $ALCI$ and $SHI$ reasoning, they have not published implementation reasoner. Besides the above mentioned active consequence-based reasoners ELK, CB and ConDOR, there are still some others, but for the three constructors inverse role, nominals and data types which we extended to the consequence-based $ALCH$ reasoning, none of the current consequence-based reasoners can fully support them together.

2.4 Related Work

In this section, we will evaluate our approach through three aspects: Firstly, we overview the previous work on datatypes, inverse roles and nominals; then we introduce the approximation approach which offers tractable reasoning support for all the expressive power of OWL 2 by using approximate reasoning. Finally, we present the hybrid reasoning trend which combines fast consequence-based reasoning and slow tableau-based reasoning, and intends to delegate the majority of computation to the more efficient, profile specific consequence-based reasoning and let the tableau-based reasoning do as few computation as possible.
2.4.1 Datatypes, Inverse Roles and Nominals

DL researchers have been working on combining DLs and datatypes for quite a long time. In the DL literature, datatypes are best known as concrete domains [54]. Baader and Hanschke [4] first presented the concrete domain approach. Lutz [53] studied the effect on complexity of adding concrete domains to a range of DLs. Haarslev et al. [36] extended ALCNHR with concrete domains. Horrocks and Sattler [40] proposed the SHOQ(D) DL which combines DLs and type systems (e.g. the XML Schema type system). Pan and Horrocks [67, 96, 59] presented the SHOQ(D_n) DL, which extends SHOQ(D) with n-ary datatype predicates. More recently, Motik et al. [59] analyze the datatype system of OWL and OWL 2 in tableau-based reasoning system. While in consequence-based reasoning, Magka et al. [55] consider extensions of the lightweight description logic (DL) EL with numerical datatypes such as naturals, integers, rationals and reals equipped with relations such as equality and inequalities; Oleksandr [64] works on improving the support for tractable reasoning with datatype expressions in EL reasoner ELK. For other consequence-based reasoners which language are beyond EL^++, none of them supports datatypes currently.

De Giacomo [15] eliminates the “converse” operator (inverse roles in DLs) from the propositional dynamic logic CPDL (Converse PDL). The approach encodes CPDL formulae into PDL and adds enough information so that the soundness and completeness of inference is not compromised. The author also points out that the technique can be applied to nominals as well. Calvanese et al. [12] introduces a general approach to eliminate inverse roles and functional restrictions.
from $\mathcal{ALCFI}$ to $\mathcal{ALC}$. For eliminating $I$, the approach needs to add one axiom for each inverse role and each concept. So the number of axioms added can be very large. Ding et al. [19] introduces a new mapping from $\mathcal{ALCFI}$ to $\mathcal{ALC}$ and further extends it to a mapping from $\mathcal{SHI}$ to $\mathcal{SH}$ in [18]. The approach allows tableau-based decision procedures to use some caching techniques and improve the reasoning performance in practice. Both approaches in [12, 18] preserve the soundness and completeness of inference after elimination of $I$. Our approach is similar to the one in [19, 18]. However, the NNF normalized form in [19, 18] in which $\top$ appears in the left side of all axioms will dramatically degrade the performance of our consequence-based $\mathcal{ALCH}$ reasoner. Thus we eliminate the inverse role based on our own normalized form and our approach is more suitable for consequence-based reasoners.

For nominals, adding this constructor to DL often increases the complexity of reasoning, hence, DL researchers mainly focus on studying the complexity and the optimization on practical reasoning in DLs with nominals. Calvanese et al. [13] prove the complexity of entailment problem of $\mathcal{SROI}$, $\mathcal{SROQ}$ and $\mathcal{SRIQ}$ is 2Exp-Time. Kazakov [44] proves that the classical reasoning problems are N2ExpTime-complete for $\mathcal{SROIQ}$. Tableau-based reasoners have been developed for the most expressive OWL2 DL $\mathcal{SROIQ}(\mathcal{D})$ which includes nominals. However, the practical tool support for nominals in OWL EL, which usually uses consequence-based reasoning, is limited. Baader et al. [6] first propose to add nominals to EL by computing a reachability relation. Then optimizations for consequence-based classification of $\mathcal{ELO}$ ontologies were studied [50], and the most effective technique is
overestimation. Firstly, the algorithm saturates the ontology using inference rules for $\mathcal{EL}$ and obtains sound subsumptions. Next, potential subsumptions are obtained by continuing saturation with a new overestimation rule added. Finally, the potential subsumptions are checked using a sound and complete but slower procedure for $\mathcal{ELO}$. Comparing with this procedure, our approach also builds upon consequence-based reasoning, and we support a more expressive DL $\mathcal{ALCHO}$.

### 2.4.2 Theory Approximation Approach

Selman and Kautz [76] illustrated an idea of rewriting a propositional logical theory into approximate tractable forms. It provides a theory approximation setting where a set of clauses (the original theory) $\Sigma$ is approximated by a Horn lower-bound $\Sigma_{lb}$ whose models are a subset of the models of $\Sigma$ and a Horn upper-bound $\Sigma_{ub}$ whose models are a superset. Our methodology is analogous to this Theory Approximation setting. Our weakening step creates $O_w$ which is an upper bound $\Sigma_{ub}$. Instead of creating a lower bound $\Sigma_{lb}$, the target of our strengthening step is to generate an $O_s$ of which models can be transformed to models of $O$ so that completeness can be achieved. Subsumption results from $O_w$ are guaranteed to be sound, exactly as queries asked of $\Sigma_{ub}$ that return “yes” can be taken also as “yes” from $\Sigma$. New candidate subsumption results from $O_s$ need to be checked, analogously as queries $\Sigma_{lb}$ that return “yes” need to be checked.

Yuan et al. [69] encoded $SROIQ$ ontologies into $\mathcal{EL}^{++}$ with additional data structures, and classified by a tractable, sound but incomplete algorithm [69]. More precisely, 1) they replaced the original concepts which contain either univer-
sal quantification or disjunction with a new named concept and added the comple-
ment expression of that original concept to the final $\mathcal{EL}^{++}$ ontology and maintain such relations in a separate complement table ($CT$). Meanwhile, they need to add some new inference rules to utilize the complementary relations in $CT$. The purpose is to eliminate universal quantification and disjunction in the final $\mathcal{EL}^{++}$ ontology and still keep as much semantics as the original one.

2) They replaced the original concepts which contain cardinality with a new named concept and added the relations among the filler concepts, the role and the cardinality values into a cardinality table ($QT$), also added extra axioms to the final $\mathcal{EL}^{++}$ ontology and more inference rules to the original $\mathcal{EL}^{++}$ inference rules. Both of us did weakening and strengthening, but our encoding different from theirs in two ways:

1) the language of our target ontology is $\mathcal{ALCHOI}(D)^{-}$ and our classification result is sound and complete with $\mathcal{ALCHOI}(D)^{-}$ while theirs is $\mathcal{SROI}(D)$ which is more expressive than ours, and their classification is sound but not complete for $\mathcal{SROI}(D)$

2) we encoded datatypes, inverse roles and nominals, while they encoded universal quantification, disjunction and number restriction; Among their three encoded constructors, two of them, i.e., universal quantification and disjunction are commonly used in ontologies. Since they did not ensure the completeness for the encoded constructors, they may miss some results in the ontologies which intensively use these two constructors.

3) we based our approach on consequence-based $\mathcal{ALCH}$ reasoner conDOR and treated conDOR as a black box, while they used existing $\mathcal{EL}^{++}$ inference rules to implement a reasoner and add extra inference rules on it plus additional data structure to store extra information.
2.4.3 Hybrid Reasoning

Similar to our work, the very recent work of Armas Romero et al. \[1\] proposes an ontology classification approach using a modular based combination of a fully-fledged OWL 2 reasoner \( R \) and an efficient \( \mathcal{L} \)-reasoner \( R_\mathcal{L} \) supporting a fragment \( \mathcal{L} \) of OWL 2. That is to say, \( R \) supports a more expressive language and \( R_\mathcal{L} \) supports a less expressive language. The approach first finds a set of classes \( \Sigma^\mathcal{L} \subseteq N_C \cap \text{Sig}(\mathcal{O}) \) and a subontology \( M^\mathcal{L} \subseteq \mathcal{O} \) written in \( \mathcal{L} \) such that the classes in \( \Sigma^\mathcal{L} \) can be completely classified by \( R_\mathcal{L} \) using only the axioms in \( M^\mathcal{L} \). Then, the approach computes an ontology \( M^\mathcal{E} \subseteq \mathcal{O} \) such that the classes in \( \mathcal{O} \setminus \Sigma^\mathcal{E} \) can be fully classified using only the axioms in \( M^\mathcal{E} \). The reasoner \( R \) is used to classify the remaining concepts in \( M^\mathcal{E} \). Finally the classification result from \( R \) is combined with \( M^\mathcal{E} \) and fed into \( R_\mathcal{L} \) to obtain \( H_\mathcal{O} \). Roughly speaking, \( H_\mathcal{O} \) is a combination of the results from two reasoners \( R \) and \( R_\mathcal{L} \). Comparing with this approach, ours uses a different way to combine the two reasoners. We classify two approximate versions \( \mathcal{O}_w \) and \( \mathcal{O}_s \) of \( \mathcal{O} \) using \( R_\mathcal{L} \), and use \( R \) to check the pairs in \( H_s \setminus H_w \), which may be unsound. Regarding the language support, our approach currently supports \( \text{ALCHOI}(D)^- \) while theirs supports more expressive \( \text{SROIQ} \). \text{MORe} is based on modular extraction. When the ontology is highly cyclic, this approach cannot work effectively. Because the sub module extracted for the efficient reasoner is very small, the bulk of the workload is still assigned to the slower tableau-based reasoner. In contrast, our approach is not based on a module approach and is not affected by the cyclic definitions in the ontologies. Moreover, the modular extraction approach has some limitations, i.e., when the ontology contains nominals, all
the nominal axioms will be extracted into modules for the efficient reasoner. If the efficient reasoner cannot handle nominals, then all the work will be assigned to the slower tableau-based reasoner. Since none of the current consequence-based reasoners can completely handle unsafe nominals. The MORe reasoner directly uses the slower tableau-based reasoner when ontologies contain unsafe nominals.

Unlike MORe’s approach of combing two independent reasoners into one reasoning system where the reasoners are used as black boxes, Andreas et al.[88] proposed a white-box approach to tightly couple tableau algorithms and completion-based saturation procedures into one reasoning system for DL $SROIQ^V$ called Konclude[89]. Concretely, the saturation algorithm is implemented into a tableau-based reasoning system. These two algorithms operate on compatible data structures so that they can exchange complex intermediate results, allowing more sophisticated optimization techniques. In the meantime, the system has implemented parallel processing at three different levels and integrated several core optimisations. The resulting system achieves remarkable overall reasoning performance.

Yujiao et al.[94] introduced how an $\mathcal{RL}$ reasoner can be used to efficiently compute lower bound and upper bound query answers on a huge datasets and arbitrary OWL 2 $SROIQ$ ontologies. They first converts the input ontology into Datalog$^{+,\vee}$ rules, and then transforms the existential-headed and disjunction-headed rules into stronger Datalog rules. These correspond to OWL 2 axioms using existentials and
disjunctions as superclasses. Their result upper bound query answer is complete, but may be unsound. Yujiao et al.’s further work [95] compute exact answers to queries over datasets and ontologies using hybrid reasoning, which built on their above work [94] and combines an OWL2 \( \mathcal{RL} \) reasoner based on a highly scalable RDF triple store with a fully-fledged OWL2 reasoner such that most of the computational workload can be delegated to the \( \mathcal{RL} \) reasoner, and the OWL reasoner is used only as necessary to ensure completeness. Their contribution in this paper is to propose a technique for extracting a subset of the dataset and ontology that is sufficient for the fully-fledged OWL2 reasoner to check with and fill out all the unsound answers from the upper bound query answers, and then obtain sound and complete result.

Different from the above hybrid approaches, Faddoul et al. [23, 24] presents a new algebraic tableau reasoning procedure for DL \( SHOQ \), which combines tableau-based DL reasoning with algebraic methods for efficient handling of nominals and number restrictions. The approach modifies and extends a standard tableau algorithm for \( \mathcal{ALC} \) to work with an algebraic reasoning component – an in-equation solver. The semantics of nominal and number restrictions are encoded into a set of in-equations and processed by the in-equation solver. The output of the solver can be used for constructing the tableau model.

Recently, Yevgeny et. al. [48] proposed another novel approach to incorporate Tableau-style backtracking into consequence-based reasoning. Concretely, the
paper presented a non-deterministic consequence-based reasoning procedure for DL $\mathcal{ALCHI}$. The procedure retains many nice properties of previously known consequence-based methods, such as optimal worst-case complexity, but also employs some tableau like features, such as backtracking, without the need for blocking to achieve termination.

Our approach shares some common features with Armas Romero et al. [1] and Yujiao et al. [95], i.e. the employed internal reasoners are all used as black-boxes. However, our approach achieves this goal so that the full reasoner tests as few pairs of concepts as possible. We do not need to extract subset of ontology. Thus, when the ontology is highly cyclic and the required extra tests are few, which is often the case, our approach can work very well.
Chapter 3

Methodology Description

In this chapter, we first introduce the deficiency of existing approaches for efficient classification in Section 3.1. After that, we propose a new methodology for practical classification on expressive and cyclic ontologies in Section 3.2. Then, we introduce the core techniques and procedures employed in our methodology - the weakening and strengthening approach in Section 3.3.

3.1 Introduction

As introduced in Sections 2.2 and 2.3, the two mainstream reasoning techniques both have their advantages and disadvantages. None of their current implementations can get complete classification results of large, expressive, and highly cyclic ontologies in acceptable reasoning time. In order to apply efficient consequence-based reasoning techniques for more expressive ontologies, approximate reason-
ing and hybrid reasoning techniques, explained in Sections 2.4.2 and 2.4.3 respectively, have been developed. Current approximate reasoning techniques, of which TrOWL is a representative, tend to get incomplete classification results if the ontology contains many non-Horn axioms. Current hybrid reasoning techniques, of which MORE is a representative, tend to be ineffective if the input ontology is expressive and highly cyclic, because of the inherent problem of the module extraction-based approach – the extracted module for the slower tableau-based reasoner may have a similar size to the original ontology, therefore the bulk of the computation is not delegated to the more efficient, profile specific reasoner, but still to the fully-fledged and slower tableau-based reasoner, which has difficulty to reason on the expressive and highly cyclic ontologies.

Due to the problems of existing work, we try to investigate a new hybrid approach to improve the reasoning efficiency on the expressive, cyclic ontologies. We try to mainly rely on a more expressive fast non-Horn consequence-based reasoner to take advantage of its efficiency and non-Horn features, and employ a hybrid approach different from modular extraction - the weakening and strengthening approach, using the full-fledged tableau-based reasoner to assist when necessary. In this chapter, we introduce the overall methodology of our weakening and strengthening approach. In later chapters we will explain how to apply the approach to specific languages.
3.2 General Methodology

We first introduce two definitions.

**Definition 3.2.1.** Given a constructor $F$ beyond $L_b$, if we could extend a $L_b$-reasoner to classify $L_bF$ ontologies with sound and complete results, we call this *single extension from $L_b$ to $L_bF$.*

**Definition 3.2.2.** Given multiple constructors $F_1$, $F_2$, ..., and $F_i$ beyond $L_b$, if we could extend a $L_b$-reasoner to classify $L_bF_1F_2...F_i$ ontologies with sound and complete results, we call this *multiple extension from $L_b$ to $L_bF_1F_2...F_i$.*

The basic idea of our approach is as follows:

- Select a language $L_b$ so that ontologies in $L_b$ can be classified by a fast $L_b$-reasoner, e.g., a consequence-based reasoner.

- Do single-extension. Chapter 4 and 5 illustrates single extension examples from the base language $\mathcal{ALCH}$ to target languages $\mathcal{ALCH}(D)^-$, $\mathcal{ALCHI}$, and $\mathcal{ALCHO}$, respectively.

- Do multiple-extension. Single extension is regarded as the special case of multiple extension. Section 6.1.1 presents the principle on how to do multiple extension based on several single extensions. In Section 6.2, we demonstrate the concrete example of how to do multiple extension $\mathcal{ALCHOI}(D)^-$ from three single extensions $\mathcal{ALCH}(D)^-$, $\mathcal{ALCHI}$ and $\mathcal{ALCHO}$.
• For a given ontology \( O \) in language \( L_b \mathcal{F}_1 \mathcal{F}_2 ... \mathcal{F}_i \) for which multiple extension can be done from \( L_b \) to \( L_b \mathcal{F}_1 \mathcal{F}_2 ... \mathcal{F}_i \), we adopt a weakening and strengthening approach, which is presented in section 3.3 for single extension and section 6.1.2 for multiple extension, to realize the goal of multiple extension.

### 3.3 Weakening and Strengthening Approach

The goal of weakening and strengthening is to extend the capability of a reasoner for language \( L_b \) (an \( L_b \)-reasoner) to a reasoner for a more expressive language \( L_o \). In the rest of the thesis we call \( L_b \) the **base language** and \( L_o \) the **target language**.

In this section, we introduce the weakening and strengthening approach for single extension, i.e., \( L_o = L_b \mathcal{F}_1 \). We will present the weakening and strengthening approach for multiple extension in Section 6.1.2. The basic idea of weakening and strengthening is to create estimated versions of the input ontology \( O \) that can be classified by an efficient consequence-based \( L_b \)-reasoner to get exact or estimated classification result(s) of \( O \), so that the classification performance can be greatly improved.

Before we present the overall weakening and strengthening approach, some concepts used in this thesis should be clarified.

**Definition 3.3.1.** An ontology \( O' \) is sound w.r.t. \( O \), if for every \( A, B \in N_C \cap \text{Sig}(O) \), \( O' \models A \subseteq B \) implies \( O \models A \subseteq B \).
Definition 3.3.2. An ontology $O'$ is complete w.r.t. $O$, if for every $A, B \in N_C \cap \text{Sig}(O)$, $O \models A \subseteq B$ implies $O' \models A \subseteq B$.

Based on Definitions 2.1.2 and 3.3.1 (resp. Definitions 2.1.3 and 3.3.2), a classification procedure that constructs and classifies an ontology $O'$ which is sound (resp. complete) w.r.t. $O$ is also sound (resp. complete).

So in the following, we will introduce our approach to construct a weakened ontology $O_w$ which is sound w.r.t $O$ and a strengthened ontology $O_s$ which is complete but may be sound or unsound w.r.t. $O$. Then we will employ different strategies to obtain the final sound and complete classification result.

Given an ontology $O$ in $L_o$, the weakening and strengthening approach computes estimated versions of $O$ in $L_b$ by the following steps:

- **Preprocessing** (Optional) We rewrite $O$ so that all constructors beyond $L_b$ are put in separated axioms.

- **Weakening** We create a weakened version $O_w$ of $O$ by removing axioms containing constructors beyond $L_b$. The classification result of $O_w$ will always be sound w.r.t. that of $O$. We will prove this conclusion in Theorem 3.3.3 of this section.

- **Strengthening** We create a strengthened version $O_s$ of $O$ by adding strengthening axioms into $O_w$ to compensate for the removed axioms. Both $O_s$ and $O_w$ are in language $L_b$. We require that the classification result $\mathcal{H}_s$ of $O_s$ should be complete but may be unsound w.r.t. that of $O$. To achieve this, Property 3.3.4 is presented in later this section. For some constructors,
\( \mathcal{H}_s \) is also sound w.r.t. \( O \), while for some others, \( \mathcal{H}_s \) may not be sound.

We call the two cases soundness-preserved strengthening and soundness-relaxed strengthening, respectively.

For soundness-preserved strengthening, we simply classify \( O_s \) with an \( \mathcal{L}_b \)-reasoner to get a sound and complete classification of \( O \). The strengthenings of \( \mathcal{ALCH}(D)^- \) and \( \mathcal{ALCHI} \) from \( \mathcal{ALCH} \) in chapter 4 are two examples of soundness-preserved strengthening. For soundness-relaxed strengthening, we classify both \( O_w \) and \( O_s \) with a main \( \mathcal{L}_b \)-reasoner \( MR \) and use an assistant \( \mathcal{L}_o \)-reasoner \( AR \) to filter out unsound pairs in \( \mathcal{H}_s \). The strengthening of \( \mathcal{ALCHO} \) from \( \mathcal{ALCH} \) in chapter 5 is an example of soundness-relaxed strengthening. Algorithm 3.1 gives the details of the classification procedure based on the soundness-relaxed strengthening.

**Algorithm 3.1: HybridClassify(O)**

| Input: An \( \mathcal{ALCHX} \) ontology \( O \) |
| Output: The classification result of \( O \) |
| 1 Encoding and Normalize \( O \) and collect all axioms \( O' \) beyond \( \mathcal{L}_b \); |
| 2 \( O_w \leftarrow O \setminus O' \); |
| 3 \( O_s \leftarrow \text{getStrAx}(O_w,O') \); |
| 4 \( \mathcal{H}_w \leftarrow \text{MR.classify}(O_w) \); |
| 5 \( \mathcal{H}_s \leftarrow \text{MR.classify}(O_s) \); |
| 6 \( \mathcal{H}_{ws} \leftarrow \mathcal{H}_w \); |
| 7 foreach \( E \subseteq \bot \in \mathcal{H}_s \setminus \mathcal{H}_w \) do |
| 8 if \( \text{AR.isSatisfiable}(O,E) \) then return \( \text{AR.classify}(O) \); |
| 9 else add \( E \subseteq \bot \) into \( \mathcal{H}_{ws} \); |
| 10 foreach \( E \subseteq F \in \mathcal{H}_s \setminus \mathcal{H}_w \) where \( F \neq \bot \) do |
| 11 if not \( \text{AR.isSatisfiable}(O,E \sqcap \neg F) \) then add \( E \subseteq F \) into \( \mathcal{H}_{ws} \); |
| 12 return \( \mathcal{H}_{ws} \) |
• Line 1 covers the encoding and normalization steps;

• Line 2 obtains the weakened ontology $O_w$;

• Line 3 calculates the strengthened ontology $O_s$;

• Line 4 and 5 compute the hierarchy $H_w$ and $H_s$ of $O_w$ and $O_s$, respectively;

• Line 6 assigns the value of $H_w$ to $H_{ws}$ which represents the return value of the entire classification result and may be added some value in the followed steps;

The following steps (Lines 7-11) only apply to soundness relaxing case.

• Line 7 selects each of the unsatisfiable concept $E$ generated from $O_s$ but not from $O_w$;

• Line 8 checks whether this concept $E$ is satisfiable in $O$ using the assistant reasoner AR. If $E$ is satisfiable in $O$, which means that our strengthened step makes this concept unsatisfiable. In such case, it means our strengthening is not good, we give up the weakening and strengthening approach and return to the AR to do the whole classification;

• If this concept $E$ is unsatisfiable in $O$, that means $H_w$ misses this result, so we add this pair back to the classification result to $H_{ws}$ in Line 9;

• Line 10 finds each of the subsumption pair which is derived from $O_s$ but not from $O_w$;
• Line 11 checks whether the counterpart concept $E \sqcap \neg F$ of that subsumption $E \sqsubseteq F$ is satisfiable in $O$ using the assistant reasoner $AR$. If $E \sqcap \neg F$ is unsatisfiable in $O$, then $E \sqsubseteq F$ holds in $O$. So we add this pair to $\mathcal{H}_{ws}$; otherwise, it means $E \sqsubseteq F$ is unsound, it needs to be filtered out, then the algorithm does not need to do anything.

• Line 12 returns the entire classification result;

In the following, we will present Theorem 3.3.3 for the soundness of $O_w$ and Property 3.3.4 for the completeness of $O_s$.

Theorem 3.3.3 shows the classification result of $O_w$ is sound w.r.t. $O$.

Theorem 3.3.3. For any $A, B \in N_C^{T, \perp} \cap \text{Sig}(O)$, if $O_w \models A \sqsubseteq B$, then $O \models A \sqsubseteq B$.

Proof. Let $I$ be a model of $O$. Since $O_w \subseteq O$, $I \models O_w$. Because $O_w \models A \sqsubseteq B$, $I \models A \sqsubseteq B$. So for any $I \models O$, $I \models A \sqsubseteq B$, thus $O \models A \sqsubseteq B$ holds.

To make $\mathcal{H}_s$ complete w.r.t. $O$, we need to ensure that for any $A, B \in N_C^{T, \perp} \cap \text{Sig}(O)$ s.t. $O \models A \sqsubseteq B$, $O_s \models A \sqsubseteq B$. A contrapositive statement is that for $A, B \in N_C^{T, \perp} \cap \text{Sig}(O)$ such that $O_s \not\models A \sqsubseteq B$, $O \not\models A \sqsubseteq B$. By the model-theoretic semantics, we have the following necessary condition for the completeness of $\mathcal{H}_s$.

Property 3.3.4. For any $A, B \in N_C^{T, \perp} \cap \text{Sig}(O)$ such that $O_s \not\models A \sqsubseteq B$, if there exists a model $I'$ of $O_s$ where $(A \sqcap \neg B)^{I'} \neq \emptyset$, it can be transformed to a model $I$ of $O$ such that $(A \sqcap \neg B)^{I} \neq \emptyset$.

The target of the transformation of model $I'$ is to make them satisfy the removed axioms, i.e., to make the transformed interpretation $I$ a model of $O$. The function
of the strengthening axioms is to ensure that after transformation, in model $I$, the other axioms except the removed axioms in $O$ will not be violated because of the change of interpretation on $I'$ in order to satisfy the removed axioms.

### 3.4 Chapter Summary

In this chapter, we analyzed the problems of existing work and proposed a new (hybrid) methodology for improving the reasoning efficiency of expressive and cyclic ontologies. Our approach is to extend a reasoner for a less expressive language to a reasoner for a more expressive language based on the weakening and strengthening approach. The main idea of the weakening and strengthening approach and its properties were explained.
Chapter 4

Classification of \( \mathcal{ALCH}(D)^- \) and \( \mathcal{ALCHI} \) with Soundness-Preserved Strengthening

In this chapter, we introduce two single extensions from \( \mathcal{ALCH} \): adding parameter datatype \( (D) \) supporting limited datatypes \( (D)^- \) and adding constructor inverse role \( (I) \). The corresponding target languages are \( \mathcal{ALCH}(D)^- \) and \( \mathcal{ALCHI} \), respectively. Both of the extensions preserve soundness and completeness. So we only need to use the main \( L_b \)-reasoner MR to classify the strengthened ontology \( O_s \) and get the final classification result \( H_O \). In the next two sections, we will illustrate the detail for each of the extension.
4.1 Classification of $\mathcal{ALCH}(\mathcal{D})$

In this section we will show how to extend an $\mathcal{ALCH}$-reasoner to classify an $\mathcal{ALCH}(\mathcal{D})$ ontology with a given datatype map $\mathcal{D}$. In the following, $ADR$ or $adr$ refers to atomic data range, see Section 2.1.1 for its definition.

For convenience, we assume all the datatypes in $\mathcal{D}$ are disjoint, as do in Motik et al. [59]. If two datatypes $d_1$ and $d_2$ are not disjoint, they can be united to become a new datatype $d_{1+2}$ with $d_1$ and $d_2$ as facets. For example, in the OWL 2 datatype map, $owl: \text{rational}$ and $owl: \text{real}$ are not disjoint. Since $owl: \text{rational}$ is a subset of $owl: \text{real}$, we can provide a facet $rat$ for $owl: \text{real}$ instead. Our approach in this thesis is limited to some commonly used datatypes and facets: (1) the real datatype and rational, decimal, integer, $>_{a}$, $\geq_{a}$, $<_{a}$ and $\leq_{a}$ facets; (2) the string datatype and equality facet; (3) the boolean datatype. This approach can also be applied to other datatypes with no facets or facets similar to the number comparing facets. Since all datatypes are disjoint, we can deal with them independently without their affecting each other.

Algorithm 4.1 shows the overall procedure computing strengthened ontology $O_s$, which consists of three steps explained in Section 3.3: 1) (Line 1–4) preprocessing, 2) (Line 5) weakening, and 3) (Line 6–8) strengthening. We will give the details in Sections 4.1.1 and 4.1.2 respectively.
Algorithm 4.1: Datatype Transformation(ProcessWeakenStrenDT)

Input: An $\mathcal{ALCH(D)}$-ontology $O$
Output: An $\mathcal{ALCH}$-ontology $O_s$ which is sound and complete w.r.t. $O$

1. $O_p \leftarrow O$;
2. foreach $d \in O$ do
3.  foreach $adr \in O$ do
4.    Replace $adr$ with $\text{transform}_d(adr)$ in $O_p$;
5.    $O_w \leftarrow O_p \setminus O_r$;
6.    $O_s \leftarrow O_w \cup \varphi(O_r)$;
7. foreach $d_1, d_2 \in N_{DT}(O), d_1 \neq d_2$ do $O_s \leftarrow O_s \cup \{\varphi(d_1) \cap \varphi(d_2) \subseteq \bot\}$;
8. foreach $d \in N_{DT}(O)$ do $O_s \leftarrow O_s \cup \text{getAxioms}_d(ADR_d(O_p), \varphi)$;
9. return $O_s$;

4.1.1 Preprocessing and Weakening

In this phase, we rewrite data ranges of the form $d[f]$ for numeric datatypes into transformed forms to reduce the kinds of facets used. We denote the procedure by $\text{transform}_d$ in the rest of this section.

Next we discuss the implementation of $\text{transform}_d$ for some commonly used datatypes. For the boolean type, we do not have any facets, so $\text{transform}_d$ does nothing. For the string type, currently we do not support any facets, so $\text{transform}_d$ still does nothing. Numeric datatypes are the most commonly used datatypes in ontologies. Here we discuss the implementation for $\text{owl:real}$, which we denote by $\mathbb{R}$. $\text{owl:real}, \text{xsd:decimal}, \text{and xsd:integer}$ are treated as facets $\text{rat}, \text{dec},$ and $\text{int}$ of $\mathbb{R}$ and denoted by $\mathbb{R[rat]}, \mathbb{R[dec]},$ and $\mathbb{R[int]}$, respectively. Comparison facets of the forms $>_a, <_a, \geq_a,$ and $\leq_a$ are supported. For $\text{transform}_d$ with input $adr$: (1) if $adr = \mathbb{R}[f]$, we transform it to equivalent data ranges using only facets of the form $>_a$, e.g. $\mathbb{R[<_a]} = \mathbb{R} \cap \neg(\mathbb{R[>_a]} \cup \{a\})$; (2) we replace any constant
a used in _adr_ with a normal form, so that any constants having the same interpretation becomes the same after normalization, e.g. integer constants +3 and 3 are both interpreted as real number 3, so they are normalized into the same form "3"^xsd : integer. Different from inverse roles in section 4.2 and nominals in chapter 5 for datatypes, we cannot put all the constructors beyond \(L_b\) in separate axioms. The result ontology from this step is called \(O_p\), which is an equivalent change to the input ontology \(O\). We will use the \(O_p\) form instead of \(O\) if necessary in some of the following algorithms or proofs. From \(O_p\), we remove all the axioms that contain datatypes and obtain the weakened version \(O_w\), which is sound based on Theorem 3.3.3. We record all the removed axioms \(O_r\) in some data structures. \(O_r\) is defined to be all of the removed axioms.

**Example 4.1.1.** Consider the following ontology \(O_{ed}\) which we will use as a running example for the weakening and strengthening for datatypes:

\[
(1) \quad \exists Fr.(\mathbb{R}[\text{int}] \cap \mathbb{R}[>0] \cap \mathbb{R}[\leq 2]) \subseteq B \\
(2) \quad B_1 \subseteq \exists Fr.(\mathbb{R}[\text{int}] \cap \{1\})
\]

\(Fr\) is a feature in axiom (1) and (2), e.g., \(Fr\) can represent hasAge in real world.

- In axiom (1), \(\mathbb{R}[\text{int}] \cap \mathbb{R}[>0] \cap \mathbb{R}[\leq 2]\) means the range of \(Fr\) is the set of integers that are greater than zero and less equal than two.

- In axioms (2), \(\mathbb{R}[\text{int}] \cap \{1\}\) means the range of \(Fr\) is integer one.

After preprocessing:

\[
\text{transform}_d(\mathbb{R}[\leq 2]) = \mathbb{R} \cap \neg \mathbb{R}[>2];
\]

\[
ADR_e(O_{ed}) = \{\mathbb{R}[\text{int}], \mathbb{R}[>0], \{1\}, \mathbb{R}[>2]\}
\]
Then, $O_p$ contains the following two axioms (3) and (2):

(3) $\exists Fr.(\mathbb{R}[int] \cap \mathbb{R}[>0] \cap \mathbb{R} \cap \neg \mathbb{R}[>2]) \subseteq B$

(2) $B_1 \subseteq \exists Fr.(\mathbb{R}[int] \cap \{1\})$

The classification result:

$$H_{O_p} = H_{O_{ed}} = \{B_1 \subseteq B\};$$

The subsumption $B_1 \subseteq B$ holds because integer one is contained in the set of integers that are greater than zero and less equal than two:

$$\mathbb{R}[int] \cap \{1\} \subseteq \mathbb{R}[int] \cap \mathbb{R}[>0] \cap \mathbb{R} \cap \neg \mathbb{R}[>2]$$

Both axioms (2) and (3) contain datatypes and they should be removed in the weakening step. So $O_r = O_p$. After weakening, $O_w = \emptyset$ and also $H_w = \emptyset$.

4.1.2 Strengthening

In the strengthening stage, we first encode features into roles and data ranges into concepts for the removed axioms. We initialize $O_s$ with a union of $O_w$ and encodings of axioms in $O_p \setminus O_w$. Then we further add to $O_s$ extra subsumption axioms between the encoded concepts to reflect the implicit relationship between the corresponding data ranges.

**Task 1: Add the encoding axioms**

Table 4.1 gives the definition of encoding function $\varphi$ over atomic elements in $O_p$, where $A_d, A_f,$ and $A_v$ are fresh concepts and $R_F$ is a fresh role. $\varphi$ over complex data
ranges, roles, concepts and axioms are defined recursively using corresponding constructors.

Table 4.1: Encoding $\phi$ for atomic concepts/roles/features/data ranges

$$
\begin{align*}
\phi(\top_D) &= \top \\
\phi(d[f]) &= A_d \cap A_f \\
\phi(\top) &= \top \\
\phi(A) &= A \\
\phi(d) &= A_d \\
\phi([v]) &= A_v \\
\phi(R) &= R \\
\phi(F) &= R_F
\end{align*}
$$

Example 4.1.2. The following shows the result of applying the above encoding function $\phi$ to the atomic data ranges, concepts and feature to $O_r$:

$$
\begin{align*}
\phi(\mathbb{R}[\text{int}]) &= A_\mathbb{R} \cap A_{\text{int}} \\
\phi(\mathbb{R}[>0]) &= A_\mathbb{R} \cap A_{[>0]} \\
\phi([1]) &= A_{[1]} \\
\phi(\mathbb{R}[>2]) &= A_\mathbb{R} \cap A_{[>2]} \\
\phi(B) &= B \\
\phi(B_1) &= B_1 \\
\phi(Fr) &= R_{Fr}
\end{align*}
$$

We add the result encoding axioms of $O_r$ to $O_w$ and obtain the initial $O_s$ as the following two axioms (4) and (5). Note for the encoding axioms, we have removed repeated conjuncts in a conjunction. For example, $A_\mathbb{R} \cap A_{\text{int}} \cap A_\mathbb{R} \cap A_{[>0]} \cap A_{[>2]}$, $A_\mathbb{R}$ is simplified to $A_\mathbb{R} \cap A_{\text{int}} \cap A_{[>0]}$.

$$
\begin{align*}
(4) & \exists R_{Fr}. (A_\mathbb{R} \cap A_{\text{int}} \cap A_{[>0]} \cap \neg A_{[>2]} ) \subseteq B \\
(5) & B_1 \subseteq \exists R_{Fr}. (A_\mathbb{R} \cap A_{\text{int}} \cap \{1\})
\end{align*}
$$

$\diamond$
Task 2: Add axioms to reflect the relationship between data ranges

Next, we introduce how to add axioms to initial $O_d$ to reflect the implicit relationship between data ranges for some commonly used datatypes. The process is represented by function $getAxioms_d$. Since the only atomic data ranges of boolean are $xsd:boolean$, $\{true\}$ and $\{false\}$, $getAxioms_d$ only needs to return two axioms

$\varphi(xsd:boolean) \equiv \varphi(\{true\}) \sqcup \varphi(\{false\})$ and $\varphi(\{true\}) \sqcap \varphi(\{false\}) \sqsubseteq \bot$.

For the string type, atomic data ranges are either $xsd:string$ or of the form $\{c\}$, where $c$ is a constant. We need to add $\varphi(\{c\}) \sqsubseteq \varphi(xsd:string)$ for each $\{c\} \in ADR(O)$, as well as pairwise disjoint axioms for all such $\varphi(\{c\})$.

Numeric datatypes are the most commonly used datatypes in ontologies. Algorithm 4.2 gives the details of $getAxioms_R$ for real numbers.


- (Line 1) Initialize the returned set $O_R^+$ to be empty.
- (Line 2–5) For each atomic data range $\{v\}$ representing a single real value in $O_p$, add to $O_R^+$ axioms representing subsumptions between $\{v\}$ and $R[int]$, $R[dec]$, and $R[rat]$.
- (Line 6–8) Add to $O_R^+$ axioms representing subsumptions among $R[int]$, $R[dec]$, and $R[rat]$ if they are used in the $O_p$.
- (Line 9–18) Add to $O_R^+$ axioms representing relationships among the set of all data ranges of the form $R[>\_\_\_]$ or $\{v\}$. There are four kinds of axioms:
Algorithm 4.2: getAxiomsₚ for ℝ

Input: A set of atomic data ranges ADRₚ(𝒪ₚ) of type ℝ, encoding function ϕ

Output: A set of axioms 𝒪⁺ₚ

1. 𝒪⁺ₚ ← ∅;

2. foreach {v} ∈ ADRₚ(𝒪ₚ) do

   3. if ℝ[int] ∈ ADRₚ(𝒪ₚ) and v ∈ (ℝ[int]) then add φ(⟨v⟩) ⊆ ϕ(ℝ[int]) to 𝒪⁺ₚ;

   4. if ℝ[dec] ∈ ADRₚ(𝒪ₚ) and v ∈ (ℝ[dec]) then add φ(⟨v⟩) ⊆ ϕ(ℝ[dec]) to 𝒪⁺ₚ;

   5. if ℝ[rat] ∈ ADRₚ(𝒪ₚ) and v ∈ (ℝ[rat]) then add φ(⟨v⟩) ⊆ ϕ(ℝ[rat]) to 𝒪⁺ₚ;

6. if ℝ[int], ℝ[dec] ∈ ADRₚ(𝒪ₚ) then add ϕ(int) ⊆ ϕ(dec) to 𝒪⁺ₚ;

7. if ℝ[int], ℝ[rat] ∈ ADRₚ(𝒪ₚ) then add ϕ(int) ⊆ ϕ(rat) to 𝒪⁺ₚ;

8. if ℝ[dec], ℝ[rat] ∈ ADRₚ(𝒪ₚ) then add ϕ(dec) ⊆ ϕ(rat) to 𝒪⁺ₚ;

9. Put all ℝ[>a] ∈ ADRₚ(𝒪ₚ) in fArray with ascending order of a;

10. foreach pair of adjacent elements ℝ[>a] and ℝ[>b] (a < b) in fArray do

    11. add ϕ(⟨a⟩) ⊆ ϕ(⟨b⟩) to 𝒪⁺ₚ;

    12. if ℝ[int] ∈ ADRₚ(𝒪ₚ) then

        13. M ← ⟨[aᵢ]⟩, where a₁, . . . , aₙ are all integer constants in (a, b);

        14. if M ∈ ADRₚ(𝒪ₚ) then

            15. add ϕ(int) ⊆ ϕ(⟨aᵢ⟩) ⊆ ⊥ to 𝒪⁺ₚ;

    16. Let N be all v such that {v} ∈ ADRₚ(𝒪ₚ) and v ∈ (a, b];

17. foreach v ∈ N do add ϕ(⟨v⟩) ⊆ ϕ(⟨v⟩) ⊆ ⊥ to 𝒪⁺ₚ;

18. foreach v₁, v₂ ∈ N, v₁ ≠ v₂ do add ϕ(⟨v₁⟩) ⊆ ϕ(⟨v₂⟩) ⊆ ⊥ to 𝒪⁺ₚ;

19. return 𝒪⁺ₚ;

62
• (Line 11) Axioms representing subsumptions between data ranges $\mathbb{R}[^{>a}]$ and $\mathbb{R}[^{>b}]$;

• (Line 12–15) Axioms representing relationships between each interval $(a, b]$ and the integers within the interval, e.g. the axiom

$$\varphi(\mathbb{R}[^{int}]) \sqcap \varphi(\mathbb{R}[^{>0.5}]) \sqcap \neg \varphi(\mathbb{R}[^{>2}]) \sqcap \neg \varphi({1}) \sqcap \neg \varphi({2}) \sqcap \bot$$

represents $\{1,2\}$ are the only integers in the interval $(0.5,2]$.

• (Line 17) Axioms representing subsumptions and disjointness between data ranges $\{v\}$ and $\mathbb{R}[^{>a}]$;

• (Line 18) Axioms representing disjointness between pairs of unequal real values $\{v_1\}$ and $\{v_2\}$;

In order to reduce the size of $O_s$, we sort all $\mathbb{R}[^{>a}]$ in ascending order in Line 9 and add only strengthening axioms for each interval $(a, b]$. This is sufficient to imply all relationships among all data ranges of the form $\mathbb{R}[^{>a}]$ or $\{v\}$. For example, $\varphi(\mathbb{R}[^{>1}]) \subseteq \varphi(\mathbb{R}[^{>0}])$ and $\varphi(\mathbb{R}[^{>2}]) \subseteq \varphi(\mathbb{R}[^{>1}])$ implies $\varphi(\mathbb{R}[^{>2}]) \subseteq \varphi(\mathbb{R}[^{>0}])$, thus the implied axioms do not need to be explicitly added to $O_s$.

**Example 4.1.3.** The following shows the strengthening axioms computed by Algorithm 4.2. And they are added to the initial $O_s$ to obtain the final $O_s$. The line number refers to the corresponding line in the algorithm. Line 3 adds one strengthening axiom:

$$(6) \ A_{[1]} \subseteq A_R \sqcap A_{int}$$
Line 11 adds one strengthening axiom:

(7) $A_{[>2]} \subseteq A_{[>0]}$

Line 17 adds two strengthening axioms:

(8) $A_{[1]} \subseteq A_{[>0]}$

(9) $A_{[1]} \cap A_{[>2]} \subseteq \bot$

So the final $O_s$ is composed of axioms (4),(5),(6),(7),(8),(9)

$H_s$ contains the following subsumptions:

(6) $A_{[1]} \subseteq A_R \cap A_{int}$  (7) $A_{[>2]} \subseteq A_{[>0]}$  (8) $A_{[1]} \subseteq A_{[>0]}$  (10) $B_1 \subseteq B$

Subsumptions (6), (7), (8) comes from the strengthening axioms. Subsumption (9) is equivalent to axiom (11) $A_{[1]} \subseteq \neg A_{[>2]}$. Based on axioms (8) and (11), $A_{[1]} \subseteq A_{[>0]} \cap \neg A_{[>2]}$ holds. Then $A_R \cap A_{int} \cap A_{[1]} \subseteq A_R \cap A_{int} \cap A_{[>0]} \cap \neg A_{[>2]}$ holds.

Hence, subsumption (10) is derived based on axioms (4), (5), (6), (8) and (9).

Among the subsumptions (6), (7), (8) and (10), the three (6), (7) and (8) do not satisfy $\text{Sig}(\alpha) \subseteq \text{Sig}(O)$ and are eliminated from $H_s$. Only (10) is preserved in $H_s$. The resulting $H_s$ is the same as $H_{O_s}$. This validates our conclusion that $O_s$ is sound and complete w.r.t. $O_{\text{eq}}$. ♦

64
4.1.3 Proof of Soundness and Completeness

The following theorem shows that the classification of $\varphi(O) = \{\varphi(\alpha) \mid \alpha \in O\}$ is sound w.r.t. $O$.

Lemma 4.1.4. Let $\alpha$ be an axiom s.t. $\text{Sig}(\alpha) \subseteq \text{Sig}(O)$. If $\varphi(O) \models \varphi(\alpha)$, then $O \models \alpha$.

Proof. It is sufficient to prove every model $I$ of $O$ can be turned into a model $I'$ of $\varphi(O)$, where $I'$ preserves interpretations for symbols in $\text{Sig}(O)$ after encoding. Let $I'$ be an interpretation of $\varphi(O)$ where:

- $\Delta^{I'} = \Delta^I \cup \Delta^D$.
- For each $A \in N_C \cap \text{Sig}(\varphi(O))$, if $A \in \text{Sig}(O)$, then $A^{I'} = A^I$; otherwise $A$ is of the form $A_d, A_f$, or $A_v$, which is interpreted as $d^D, f^D$ or $\{v^D\}$, respectively.
- For each $r \in N_R \cap \text{Sig}(\varphi(O))$, $r^{I'} = r^I$ if $r \in \text{Sig}(O)$, or $r^{I'} = F^{I'}$ if $r = \varphi(F)$ and $F \in N_F \cap \text{Sig}(O)$.

It is easy to see that under such definitions, for every data range $dr$ in $O$, $\varphi(dr)^{I'} = dr^D$; for every role $R$ in $O$, $\varphi(R)^{I'} = R^I$; and so for every concept $C$, $\varphi(C)^{I'} = C^I$.

Thus, concept subsumption and role subsumption axioms in $O$ are also satisfied after the encoding. Feature subsumption axioms $F \sqsubseteq G$ in $O$ are encoded into $\varphi(F) \sqsubseteq \varphi(G)$ in $\varphi(O)$, which is trivially satisfied by $I'$. Thus $I'$ is a model of $\varphi(O)$, and the theorem is proved. $\Box$

Theorem 4.1.5. Let $\alpha$ be an axiom s.t. $\text{Sig}(\alpha) \subseteq \text{Sig}(O)$. $O_s$ is the strengthened ontology returned from algorithm 4.1. If $O_s \models \varphi(\alpha)$, then $O \models \alpha$. 
Proof. According to the computation of strengthening axioms, we can see that the model $I'$ created in Lemma 4.1.4 satisfies also the strengthening axioms. Hence $I'$ is also a model of $O_s$. Thus the theorem holds.

Applying the theorem to concept subsumption axioms $\alpha = A \sqsubseteq B, A, B \in N_C \cap \text{Sig}(O)$, we get $O_s \models A \sqsubseteq B$ implies $O \models A \sqsubseteq B$. Hence, by Definition 3.3.1 $O_s$ is sound w.r.t. $O$. Thus, our procedure that classifies $O_s$ to get $H_O$ is also sound.

Although $\varphi(O)$'s classification is sound w.r.t. $O$, it may not be complete. In order to preserve completeness, extra axioms need to be added to $\varphi(O)$ to get $O_s$. Next, we give a sufficient condition for $O_s$ to be complete w.r.t. $O$.

**Definition 4.1.6.** Let $ADR_d$ be a set of atomic data ranges of the forms $d$, $d[d']$, or $\{v\}$, where $v \in N_{LS}(d)$, and let $\varphi$ be an encoding function. We say $\text{getAxioms}_d$ preserves data range relationships if for any $ADR_d$, $\varphi$, and different $ar_1, \ldots, ar_n, ar'_1, \ldots, ar'_m \in ADR_d, m, n \geq 0$, such that $(\bigcap_{i=1}^{n} ar_i) \cap (\bigcap_{j=1}^{m} \neg ar'_j)$ is unsatisfiable, $(\bigcap_{i=1}^{n} \varphi(ar_i)) \cap (\bigcap_{j=1}^{m} \neg \varphi(ar'_j))$ is unsatisfiable in the ontology $O_d^+ = \text{getAxioms}_d(ADR_d, \varphi)$.

**Lemma 4.1.7.** Let $O_s$ be the ontology computed from $O_p$ using Algorithm 4.1. If for every datatype $d \in N_{DT}(O_p)$, $\text{getAxioms}_d$ preserves data range relationships, then for any model $I'$ of $O_s$, there exists a $\Delta^D$ such that for every $x \in \Delta^I$, there is a data value $t(x) \in \Delta^D$ such that $x \in \varphi(dr)^I' \iff t(x) \in dr^D$ for any data range $dr$ appeared in $O_p$.

Proof. Let $ADR_d(O_p)$ be all the atomic data ranges in $O_p$ which are of datatype

66
We define $DR^+_d(x)$, $DR^-_d(x)$ and $DR_d(x)$ as

\[
DR^+_d(x) = \{ ar \mid ar \in ADR_d(O_p) \land x \in \varphi(ar)^T \}
\]
\[
DR^-_d(x) = \{ \neg ar \mid ar \in ADR_d(O_p) \land x \notin \varphi(ar)^T \}
\]
\[
DR_d(x) = DR^+_d(x) \cup DR^-_d(x)
\]

We first show that there is at most one $d^* \in N_{DT}(O_p)$ such that $DR^+_d(x) \neq \emptyset$. Note that if $ar \in DR^+_d(x)$, then $x \in \varphi(ar)^T$ and $ar \subseteq d^D$. Since $getAxioms_d$ preserves data range relationships, $\varphi(ar) \land \neg \varphi(d) \subseteq \bot$. Hence, if $DR^+_d(x) \neq \emptyset$, then $x \in \varphi(d)^T$. Line 7 of Algorithm 4.1 ensures $O_s = \varphi(d_1) \cap \varphi(d_2) \subseteq \bot$ for any different datatypes $d_1$ and $d_2$; thus, $x$ cannot belong to both $\varphi(d_1)^T$ and $\varphi(d_2)^T$.

If $DR^+_d(x) = \emptyset$ for all datatypes $d \in N_{DT}(O_p)$, then we can choose some $\Delta_D \supseteq \bigcup_{d \in N_{DT}(O_p)} d^D$, and $\tau(x)$ to be a value which is not in any $d^D$. Otherwise, there can only be one $d^* \in N_{DT}(O_p)$ such that $DR^+_d(x) \neq \emptyset$. Let $h = \bigcap_{d \in DR^+_d(x)} dr$. Since $x \in \varphi(h)^T$, $\varphi(h)$ is satisfiable. Since $getAxioms_d$ preserves data range relationships, by Definition 4.1.6, there exists a $\Delta_D$ and $\tau(x) \in \Delta_D$ such that $\tau(x) \in h^D$. Next we prove $x \in \varphi(dr)^T \Leftrightarrow \tau(x) \in dr^D$ by induction over $dr$:

- $dr = ar$ If $DR^+_d(x) = \emptyset$ for all datatypes, then $x \notin \varphi(ar)^T$ and $\tau(x) \notin ar^D$.

Otherwise, both $d^*$ and $h$ exist. If $x \in \varphi(ar)^T$, then $ar$ is a conjunct of $h$. Hence, $h^D \subseteq ar^D$ and $\tau(x) \in ar^D$. Otherwise, either $\neg ar$ is a conjunct of $h$ or $ar$ is not of type $d^*$. In both cases, $\tau(x) \notin ar^D$. So in all cases, $x \in \varphi(ar)^T \Leftrightarrow \tau(x) \in ar^D$ holds.
• $dr = \neg dr'$

\[
x \in \varphi(dr)^I \iff x \notin \varphi(dr')^I \iff t(x) \notin dr^D \iff t(x) \in dr^D
\]

• $dr = dr_1 \cap dr_2$

\[
x \in \varphi(dr)^I \iff x \in \varphi(dr_1)^I \land x \in \varphi(dr_2)^I \iff t(x) \in dr_1^D \land t(x) \in dr_2^D \iff t(x) \in dr^D
\]

• $dr = dr_1 \cup dr_2$

\[
x \in \varphi(dr)^I \iff x \in \varphi(dr_1)^I \lor x \in \varphi(dr_2)^I \iff t(x) \in dr_1^D \lor t(x) \in dr_2^D \iff t(x) \in dr^D
\]

Therefore, the lemma holds. \(\square\)

**Theorem 4.1.8.** Let $O_s$ be the ontology computed from $O_p$ using Algorithm 4.1, and $\alpha$ be an axiom using symbols in $\text{Sig}(O_p)$ or data ranges in $O_p$. If for every datatype $d \in \mathbb{N}_{DT}(O_p)$, $\text{getAxioms}_d$ preserves data range relationships, then $O_p \models \alpha$ implies $O_s \models \varphi(\alpha)$.

**Proof.** We will show that if $O_s \not\models \varphi(\alpha)$, then $O_p \not\models \alpha$. Let $I'$ be a model of $O_s$ where $I' \not\models \varphi(\alpha)$. Since all $\text{getAxioms}_d$ preserve data range relationships, by Lemma 4.1.7 we can find a $\Delta^D$, and for each $x \in \Delta^I$, there exists $t(x) \in \Delta^D$ such that $x \in \varphi(dr)^I \Rightarrow t(x) \in dr^D$. Using this $\Delta^D$, we define an interpretation $I' = (\Delta^I, \Delta^D, \cdot)^I$ of $O_p$, where the domain, atomic concepts and roles are interpreted...
the same as $I'$, while any feature $F$ is interpreted as

$$F^T = \{(o, t(x)) \mid (o, x) \in \varphi(F)^T\}$$

Since $x \in \varphi(dr)^T \iff t(x) \in dr^D$, it is easy to see that $(\exists F.dr)^T = (\varphi(\exists F.dr))^T$ and $(\forall F.dr)^T = (\varphi(\forall F.dr))^T$. Since $I$ preserves interpretations of all atomic concepts and roles from $I'$, for any concept $C$ in $O_p$, $C^I = \varphi(C)^I$. Thus, $I$ satisfies all concept and role subsumption axioms in $O_p$. For feature subsumptions $F \sqsubseteq G$, we have

$$(o, t(x)) \in F^T \rightarrow (o, x) \in \varphi(F)^T \rightarrow (o, x) \in \varphi(G)^T \rightarrow (o, t(x)) \in G^T$$

Thus, all feature subsumptions in $O_p$ are satisfied, and so $I$ is a model of $O_p$. Since $\alpha$ use only symbols in $\text{Sig}(O_p)$ and data ranges in $O_p$, $I \not \models \alpha$. Hence, $O_p \not \models \alpha$ and the theorem is proved. □

For the boolean and string datatypes, it is obvious that the corresponding $\text{getAxioms}_d$ preserves data range relationships. For $\text{getAxioms}_r$, we prove this property in the following lemma.

**Lemma 4.1.9.** Algorithm 4.2’s implementation of $\text{getAxioms}_r$ preserves data range relationship for owl:real.

**Proof.** Let $h = \bigcap_{i=1}^m dr_i, m > 0$ where $dr_i$ is $ar$ or $\neg ar$. After normalization for the real datatype, $ar$ can only be of the forms $\mathbb{R}$, $\mathbb{R}[dec]$, $\mathbb{R}[rat]$, $\mathbb{R}[int]$, $\mathbb{R}[>_a]$, or $\{v\}$. We write $h^+$ and $h^-$ for the set of all positive and negative conjuncts of $h$, 69
respectively. Next we perform a case-by-case analysis over all possible cases that $h^D = \emptyset$. Without loss of generality, we assume $h$ is minimal, i.e. if we remove any conjunct from $h$, then $h^D \neq \emptyset$.

Case 1 $h^+ \subseteq \{\mathbb{R}, \mathbb{R}\{\text{dec}\}, \mathbb{R}\{\text{rat}\}, \mathbb{R}\{\text{int}\}\}$. There are two cases for $h^D = \emptyset$:

Case 1.1 $\neg \mathbb{R} \in h^-$ and $\varphi(\mathbb{R}[f]) \cap \neg \varphi(\mathbb{R}) \subseteq \perp$ is implied by the definition of $\varphi$.

Case 1.2 There exists $f_1, f_2 \in \{\text{rat}, \text{dec}, \text{itg}\}$ such that $(\mathbb{R}[f_1] \cap \neg \mathbb{R}[f_2])^D = \emptyset$. All such cases are captured by lines 6 to 8 and $\varphi(\mathbb{R}[f_1]) \cap \neg \varphi(\mathbb{R}[f_2]) \subseteq \perp$.

Case 2 There exists $\mathbb{R}[>a] \in h^+$. There are 3 cases for $h^D = \emptyset$:

Case 2.1 $\neg \mathbb{R}[>a] \in h^-$ and $a > a'$. In this case, axioms added by line 11 where $a = a'$ and $b = a$, ensures that $O^+_R \models \varphi(\mathbb{R}[>a]) \cap \varphi(\neg \mathbb{R}[>a']) \subseteq \perp$ for any $a, a'$ such that $a > a'$.

Case 2.2 $\mathbb{R}\{\text{int}\} \in h^+$, $\neg \mathbb{R}[>b], \neg\{a_1\}, \ldots, \neg\{a_n\} \in h^-$ such that $a_1, \ldots, a_n$ are all integers in the range $(a, b]$. This case is captured by lines 12 to 15.

Case 2.3 $\{v\} \in h^+$, and $v \leq a$. Line 17 ensures that $\varphi(\{v\}) \cap \varphi(\mathbb{R}[>a]) \subseteq \perp$ for the minimum $a'$ such that $v \leq a'$. Axioms added by line 11 ensures $O^+_R \models \varphi(\mathbb{R}[>a]) \subseteq \varphi(\mathbb{R}[>a'])$. Thus, $O^+_R \models \varphi(\{v\}) \cap \varphi(\mathbb{R}[>a]) \subseteq \perp$.

Case 3 There exists $v$ such that $\{v\} \in h^+$, and no $\mathbb{R}[>a] \in h^+$. There are 3 cases where $h^D = \emptyset$:

Case 3.1 There exists $f \in \{\text{rat}, \text{dec}, \text{itg}\}$ such that $\neg \mathbb{R}[f] \in h^-$ and $v^D \in \mathbb{R}[f]$. $O^+_R \models \varphi(\{v\}) \cap \neg \varphi(\mathbb{R}[f]) \subseteq \perp$ is ensured by lines 2 to 5.
Case 3.2 \( \neg R \in h^- \). Since in OWL 2, no real constants beyond \textit{owl: rational}, \textit{xsd: decimal}, \textit{xsd: integer} are supported. So there exists \( f \in \{ \text{rat}, \text{dec}, \text{itg} \} \) such that \( v^D \in \mathbb{R}[f] \). Lines 2 to 5 ensure \( \varphi(\{v\}) \subseteq \varphi(\mathbb{R}[f]) \subseteq \varphi(\mathbb{R}) \), and so \( O_R^+ \models \varphi(\{v\}) \sqcap \neg \varphi(R) \sqsubseteq \bot \).

Case 3.3 There exists \( \{v'\} \in h^+ \) and \( v^D \neq v'^D \). Lines 17 to 18 ensure \( O_R^+ \models \varphi(\{v\}) \sqcap \varphi(\{v'\}) \sqsubseteq \bot \).

Applying Lemma 4.1.9 and Theorem 4.1.8 to concept subsumption axioms \( \alpha = A \sqsubseteq B \), \( A, B \in \mathcal{N}_C \cap \text{Sig}(O) \), we obtain \( O_p \models \alpha \) implies \( O_s \models \alpha \). Hence, by Definition 3.3.2 \( O_s \) is complete w.r.t. \( O_p \). Thus, our procedure that classifies \( O_s \) to get \( \mathcal{H}_O \) is also complete.

4.2 Classification of \( \mathcal{ALCHI} \)

In this section, we discuss how we transform an \( \mathcal{ALCHI} \) ontology \( O \) into an \( \mathcal{ALCH} \) ontology \( O_s \), such that \( O \models A \sqsubseteq B \) iff \( O_s \models A \sqsubseteq B \). The procedures are in the following:

4.2.1 Preprocessing and Weakening

In this step, we have two tasks. Task 1): we first replace each of the anonymous role “the inverse role of \( r \), i.e., \( r^- \)” with an atomic role \( r' \), and add an inverse properties axiom \( r^- = r' \). where \( r \) denotes an atomic role. This is an equivalent change to the original ontology \( O \). After that, except the two types of role related
axioms $r \sqsubseteq s$ and $r^{-} = r'$, the other axioms are in $\mathcal{ALCH}$. Task 2): We normalize the $\mathcal{ALCH}$ axioms to contain only axioms of the forms $\bigcap A_j \sqsubseteq \bigcup B_j$, $A \sqsubseteq \exists r.B$, $\exists r.A \subseteq B$, $A \subseteq \forall r.B$, $R \sqsubseteq S$. The normalization procedure is demonstrated in the following and which follows the structural transformation shown in previous work [78]. The result ontology from this step is called $O_p$, which is an equivalent change to the input ontology $O$. We will use the $O_p$ form instead of $O$ if necessary in some of the following algorithms or proofs.

From $O_p$, we remove all the inverse role properties axioms and obtain the weakened version $O_w$, which is sound based on Theorem 3.3.3. We record all the removed axioms $O_r$ in some data structures.

**Normalization for $\mathcal{ALCH}$ Axioms**

Given an $\mathcal{ALCH}$ ontology $O$, the normalization is performed by recursively traversing through the structure of axioms. The process introduces a new atomic concept $[C]$ for each concept $C$ occurring in $O$. Structure transformation of $C$, denoted by $st(C)$, is defined in Table 4.2.

The structural transformation of $O$ is a new ontology $st(O)$ containing the same role inclusion axioms as $O$ and the axioms shown in Table 4.3.

<table>
<thead>
<tr>
<th>$st(A)$</th>
<th>$st(C \sqcap D)$</th>
<th>$st(\exists r.C)$</th>
<th>$st(\forall r.C)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$</td>
<td>$[C] \sqcap [D]$</td>
<td>$\exists r.[C]$</td>
<td>$\forall r.[C]$</td>
</tr>
<tr>
<td>$\top$</td>
<td>$[C] \sqcup [D]$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\bot$</td>
<td>$\neg [C]$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

72
Table 4.3: Axioms added in structural transformation

- $st(C) \subseteq [C]$ for every $C$ occurring negatively in $O$,
- $[D] \subseteq st(D)$ for every $D$ occurring positively in $O$,
- $[C] \subseteq [D]$ for each axiom $C \subseteq D$ in $O$.

Table 4.4: Rewritten Axioms

\[
\begin{align*}
[C \cap D] &\subseteq [C] \cap [D] \leadsto [C \cap D] \subseteq [C] \text{ and } [C \cap D] \subseteq [D] \\
[C] \cup [D] &\subseteq [C \cup D] \leadsto [C] \subseteq [C \cup D] \text{ and } [D] \subseteq [C \cup D] \\
[-C] &\subseteq [-C] \leadsto [-C] \cap [C] \subseteq \bot \\
\neg[C] &\subseteq [\neg C] \leadsto \top \subseteq [-C] \cup [C]
\end{align*}
\]

Moreover, some types of axioms are not normalized but can be rewritten into normalized axioms as indicated in Table 4.4. Notice that the size of $st(O)$ is linear in the size of $O$ and $st(O)$ can be computed from $O$ in polynomial time. The proof of the correctness of this structural transformation can be found in [78].

**Example 4.2.1.** Consider the following ontology $O_{ei}$ which we will use as a running example for the weakening and strengthening for inverse role:

1. $A_1 \sqsubseteq \exists R'. A_2$
2. $A_2 \sqsubseteq A_3$
3. $A_3 \sqsubseteq \forall S. A_4$
4. $R^- = R'$
5. $R \sqsubseteq S$

For the above ontology $O_{ei}$, we do nothing for the preprocessing. So $O_p = O_{ei}$.  

73
The classification result $H_{O_p}$ contains the following two subsumptions:

\[(2) \ A_2 \sqsubseteq A_3 \quad \text{and} \quad (6) \ A_1 \sqsubseteq A_4\]

Subsumption (2) comes from $O_{e_i}$. Axiom (3) $A_3 \sqsubseteq \forall S.A_4 \leftrightarrow \exists S^-.A_3 \sqsubseteq A_4$ is a well-known equivalence. Subsumption (6) can be derived from $\exists S_.A_3 \sqsubseteq A_4$ and axioms (1), (2), (4) and (5). Axiom (4) and (5) implies $R' \sqsubseteq S^-$. And $A_1 \sqsubseteq \exists R'.A_2 \sqsubseteq \exists S^-.A_3 \sqsubseteq A_4$. So subsumption (6) holds.

Axiom (4) contains the inverse role axiom and should be removed in the weakening step. So $O_r$ contains axiom (4), and $O_w$ includes all the other axioms in $O_p$ except (4). $H_w = \{A_2 \sqsubseteq A_3\}$ and subsumption (6) is missing in $H_w$ because the inverse role axiom (4) is removed in $O_w$. Then $R' \sqsubseteq S^-$ cannot be derived. Without this bridge, $R'.A_2 \sqsubseteq \exists S^-_.A_3$ cannot be derived, either. So we cannot derive (6). ♦

### 4.2.2 Strengthening

In the strengthening step, we will add axioms to address the impact of the removed inverse properties axioms. We analyze that an inverse role axiom $r^- = r'$ can affect the inference of $\mathcal{ALCH}$ through

- The role hierarchy, where (1) $r^- = r'$ and $r^- = r^*$ imply $r' = r^*$;
  (2) $r \sqsubseteq s$, $r^- = r'$ and $s^- = s'$ imply $r' \sqsubseteq s'$.

- The well-known equivalence $A \sqsubseteq \forall R.B \iff \exists R^-.A \sqsubseteq B$, i.e., $r^- = r'$ and $A \sqsubseteq \forall r.B$ imply $\exists r'.A \sqsubseteq B$; and $r^- = r'$ and $\exists r.A \sqsubseteq B$ imply $A \sqsubseteq \forall r'.B$. 

74
• The role hierarchy,
  \[
  \text{if } r^{-} = r' \text{ and } r^{-} = r^* \text{ add } r' = r^*; \\
  \text{(2) if } r \subseteq s, r^{-} = r' \text{ and } s^{-} = s' \text{ add } r' \subseteq s'.
  \]

• The well-known equivalence \( A \subseteq \forall R.B \iff \exists R^- . A \subseteq B \), i.e.,
  \[
  \text{if } r^{-} = r' \text{ and } A \subseteq \forall r.B \text{ add } \exists r'.A \subseteq B; \text{ and} \\
  \text{if } r^{-} = r' \text{ and } \exists r.A \subseteq B \text{ add } A \subseteq \forall r'.B.
  \]

Algorithm 4.3 shows the details of transformation for inverse roles and the strengthening axioms added. The procedure \( \text{Inv}_r \) contains the set of atomic roles which are the inverses of \( r \). Line 1 initializes \( O_w \) with \( \text{ALCH} \) axioms in \( O_p \) excluding the inverse role axioms; Lines 2 to 6 initialize \( \text{Inv}_r \) and put all \( r \) where \( \text{Inv}_r \neq \emptyset \) into \( \text{RolesToBeProcessed} \). Line 7 initializes \( O_s \) with \( O_w \). Lines 8 to 17 process each role in \( \text{RolesToBeProcessed} \) and add axioms into \( O_s \) to address the effect of inverse role axioms mentioned above. Line 10 ensures all roles in \( \text{Inv}_r \) are equivalent. Lines 11 to 15 ensure that if a role in \( \text{RolesToBeProcessed} \) has a role subsumption relationship with another role, then their corresponding inverse roles also have a role subsumption relationship. Concretely, we add \( r' \subseteq s' \) if \( r \subseteq s, r' \in \text{Inv}_r \) and \( s' \in \text{Inv}_s \). Note that if \( r' \subseteq s^- \) where \( \text{Inv}_s = \emptyset \), we need to introduce a new atomic role \( s' \) for \( s^- \), because an axiom \( \exists s.A \subseteq B \), equivalent to \( A \subseteq \forall s'.B \), can interact with \( A' \subseteq \exists r'.B \) in the inference. Lines 16 and 17 add \( A \subseteq \forall r'.B \) or \( \exists r'.A \subseteq B \).

**Example 4.2.2.** The following shows the strengthening axioms (7) and (8) computed by Algorithm 4.3. The line numbers refer to the corresponding line in the
Algorithm 4.3: Transformation for inverse roles

**Input:** Normalized ontology $\mathcal{ALCHI}$ ontology $O_p$

**Output:** An $\mathcal{ALCH}$ ontology $O_s$ having the same classification result as $O_p$

1. Initialize $O_w$ with all $\mathcal{ALCH}$ axioms in $O_p$, excluding inverse role axioms;
2. $\textbf{foreach } r \in N_R \cap \text{Sig}(O) \textbf{ do } \text{Inv}_r \leftarrow \emptyset$;
3. RolesToBeProcessed $\leftarrow \emptyset$;
4. $\textbf{foreach } r' = r^- \in O_p \textbf{ do }$
   5. $\text{Inv}_r \leftarrow \text{Inv}_r \cup \{r'\}; \text{Inv}_{r'} \leftarrow \text{Inv}_{r'} \cup \{r\};$
   6. RolesToBeProcessed $\leftarrow$ RolesToBeProcessed $\cup \{r, r'\}$;
    $O_s \leftarrow O_w$;
8. $\textbf{while }$ RolesToBeProcessed $\neq \emptyset$ $\textbf{ do }$
   9. remove a role $r$ from RolesToBeProcessed and pick a role $r'$ from $\text{Inv}_r$;
   10. $\textbf{foreach } r^* \in \text{Inv}_r \text{ where } r^* \text{ is not } r' \textbf{ do add } r' \equiv r^* \text{ to } O_s$;
   11. $\textbf{foreach } r \sqsubseteq s \in O_p \textbf{ do }$
      12. $\textbf{if } \text{Inv}_s = \emptyset \text{ then }$
          13. add a fresh atomic role $s'$ to $\text{Inv}_s$;
          14. RolesToBeProcessed $\leftarrow$ RolesToBeProcessed $\cup \{s\}$;
      15. pick a role $s'$ from $\text{Inv}_s$ and add $r' \sqsubseteq s'$ to $O_s$;
   16. $\textbf{foreach } \exists r.A \sqsubseteq B \in O_p \textbf{ do add } A \sqsubseteq \forall r'.B \text{ to } O_s$;
   17. $\textbf{foreach } A \sqsubseteq \forall r.B \in O_p \textbf{ do add } \exists r'.A \sqsubseteq B \text{ to } O_s$;
18. return $O_s$
algorithm. Line 13 adds a fresh atomic role $S'$ to Inv$_s$. Line 15 adds one strengthening axiom

\[(7) \quad R' \sqsubseteq S'\]

Line 17 adds another strengthening axiom:

\[(8) \quad \exists S'. A_3 \sqsubseteq A_4\]

So the final $O_s$ is composed of axioms (1), (2), (3), (5), (7) and (8).

$H_s$ contains the following subsumptions:

\[(2) \quad A_2 \sqsubseteq A_3 \quad (6) \quad A_1 \sqsubseteq A_4\]

In $H_s$, subsumption (6) is derived for the following reason: axiom (4) is not in $O_s$, however, the strengthening axiom (7) reestablishes the relationship of $R'$ and $S'$. And the strengthening axiom (8) acts as the equivalence axiom of axiom (3). So based on axiom (1), (2), (7) and (8), subsumption (6) can be derived.

\[A_1 \sqsubseteq \exists R'. A_2 \sqsubseteq \exists S'. A_3 \sqsubseteq A_4\]

So $H_s = H_{O_s}$. This validates our conclusion that $O_s$ is sound and complete w.r.t. $O_{el}$.  

\[\diamond\]
4.2.3 Proof of Soundness and Completeness

The following theorem shows that our transformation for inverse role preserves the subsumptions in $O$.

**Theorem 4.2.3.** For every $A, B \in N_{C}^{\top, \bot} \cap \text{Sig}(O)$, $O_{p} \models A \sqsubseteq B$ iff $O_{s} \models A \sqsubseteq B$.

**Proof.** Let $O' = O_{p} \cup \{s' = s^- \mid s'$ is a fresh role introduced for $s$ in line 13]. Since the new axioms only define fresh roles, it is trivial that $O_{p} \models A \sqsubseteq B$ iff $O' \models A \sqsubseteq B$. Hence, we only need to prove $O' \models A \sqsubseteq B$ iff $O_{s} \models A \sqsubseteq B$.

According to Algorithm 4.3, it is easy to see $O' \models O_{s}$, thus $O_{s} \models A \sqsubseteq B$ implies $O' \models A \sqsubseteq B$. Next we will prove if $O' \models A \sqsubseteq B$ then $O_{s} \models A \sqsubseteq B$ by its contraposition: if $O_{s} \n \models A \sqsubseteq B$, then $O' \n \models A \sqsubseteq B$. Let $I$ be a model of $O_{s}$ where $(A \sqcap \neg B)^I \neq \emptyset$, and $I'$ be an interpretation of $O'$ such that

- For every atomic concept $A$, $A^{I'} = A^I$.

- For every atomic role $r$, $r^{I'} = r^I \cup (\bigcup_{r \in s \in inv} (r'^I))$.

Now we show $I'$ is a model of $O'$. Since the only change from $I$ to $I'$ are the interpretations of roles $r$ where $\text{inv}_r \neq \emptyset$, we only need to prove $I'$ satisfies every axiom that has such a role. Note that $\text{inv}_r$ is made nonempty by an operation in line 5 or 13 of Algorithm 4.3, and $r$ is added to $\text{RolesToBeProcessed}$ by line 6 or 14 respectively in both cases. So $r$ will be processed in the loop from lines 8 to 17 and there exists a representative role $r_i$ which is picked from $\text{inv}_r$ in line 9. Note that according to line 10 of Algorithm 4.3 all roles in $\text{inv}_r$ are equivalent to $r_i$ in $O_{s}$. Thus, the following statement (*) holds:

78
If $\text{Inv}_r \neq \emptyset$, $r'^I = r^I$ for every $r' \in \text{Inv}_r$, and $r'^I = r^I \cup (r_i^-)^I$ \hspace{1cm} (*)

Now we prove $I'$ satisfies every axiom that has a role $r$ s.t. $\text{Inv}_r \neq \emptyset$, on a case-by-case analysis.

- **$r' = r^-$** According to line 5, $r \in \text{Inv}_{r'}$ and $r' \in \text{Inv}_r$. Let $r_i'$ be the representative role in $\text{Inv}_{r'}$. By (*), we have

$$(r^-)^I = (r^-)^I \cup ((r_i^-)^I) = (r^-)^I \cup r_i'^I = (r_i'^-)^I \cup r'^I = r'^I'$$

- **$r \subseteq s$** Since $r \subseteq s \in O_s$, $r^I \subseteq s^I$. If $\text{Inv}_s \neq \emptyset$ and $\text{Inv}_r = \emptyset$, we have $r'^I = r^I \subseteq s^I \subseteq s'^I$. Otherwise $\text{Inv}_r \neq \emptyset$. By line 11 to 15 we know that there exists some $s' \in \text{Inv}_s$ such that $r_i \subseteq s' \in O_s$. Let $s_i$ be the representative role in $\text{Inv}_s$, by (*) we have

$$r'^I = r^I \cup (r_i^-)^I \subseteq s^I \cup (s_i^-)^I = s^I \cup (s_i^-)^I = s'^I$$

- **$A \subseteq \exists r.B$** By (*) and $B'^I = B^I$, we have $(\exists r.B)^I = (\exists r.B)^I \cup (\exists r_i^- . B)^I$. Thus, $A'^I = A^I \subseteq (\exists r.B)^I \subseteq (\exists r.B)^I$.

- **$\exists r.A \subseteq B$** By (*) and $A'^I = A^I$, we have $(\exists r.A)^I = (\exists r.A)^I \cup (\exists r_i^- . A)^I$. Since $\exists r.A \subseteq B \in O_s$, $(\exists r.A)^I \subseteq B^I$. By line 16, $A \subseteq \forall r_i . B \in O_s$; thus, $O_s \models \exists r_i^- . A \subseteq B$, and so $(\exists r_i^- . A)^I \subseteq B^I$. Hence, $(\exists r.A)^I \subseteq B^I = B'^I$.

- **$A \subseteq \forall r.B$** By (*) and $B'^I = B^I$, we have the following equivalence, where
\[ r^T(x) = \{ y \mid (x, y) \in r^T \} \]

\[(\forall r.B)^T = \{ x \mid r^T(x) \subseteq B^T \} \]

\[= \{ x \mid (r^T(x) \cup (r_i)^T(x)) \subseteq B^T \} \]

\[= \{ x \mid r^T(x) \subseteq B^T \land (r_i)^T(x) \subseteq B^T \} \]

\[= \{ x \mid r^T(x) \subseteq B^T \} \cap \{ x \mid (r_i)^T(x) \subseteq B^T \} \]

\[= (\forall r.B)^T \cap (\forall r_i.B)^T \]

Since \( A \subseteq \forall r.B \in O_s \), \( A^T \subseteq (\forall r.B)^T \). By line [17], \( \exists r_i.A \subseteq B \in O_s \); thus, \( O_s \models A \subseteq \forall r_i.B \), and so \( A^T \subseteq (\forall r_i.B)^T \). Hence, \( A^T = A^T \subseteq (\forall r.B)^T \).

Hence, \( T' \) is a model of \( O' \), and \( (A \land \neg B)^T = (A \land \neg B)^T \neq \emptyset \). So \( O' \not\models A \subseteq B \).

Thus for every \( A, B \in N_{\text{C}}^{\top, \bot} \cap \text{Sig}(O) \), \( O' \models A \subseteq B \) implies \( O_s \models A \subseteq B \), and the theorem is proved. \( \square \)

### 4.3 Chapter Summary

This chapter demonstrates classification of \( \mathcal{ALCH}(D)^- \) and \( \mathcal{ALCHI} \) ontologies based on the weakening and strengthening approach using a fast \( \mathcal{ALCH} \) reasoner. We explained all the steps of the WS approach: preprocessing, weakening and strengthening.

For \( \mathcal{ALCH}(D)^- \) ontologies, we first rewrite the data range of numeric datatype to
simplify its format in the preprocessing step and then remove the datatype axioms to obtain the weakened $\text{ALCH}$ ontology. After that, we add encoded axioms of the removed axioms and axioms to reflect the relationship between data ranges, both in $\text{ALCH}$, to the weakened ontology for compensating the semantics of the removed datatype axioms.

For $\text{ALCHI}$ ontologies, we first rewrite the ontology and put the inverse roles in separated axioms. Then we normalize the $\text{ALCH}$ axioms to contain only five types of axioms in the preprocessing step. After that, we remove the inverse roles axioms to obtain the weakened $\text{ALCH}$ ontology. Finally, we add the $\text{ALCH}$ strengthening axioms to the weakened ontology for compensating the semantics of the removed inverse role axioms.

For the resulting strengthened ontologies from the above $\text{ALCH(D)}^-$ or $\text{ALCHI}$ classification procedure, we use only the MR reasoner to classify them.

Also, we proved the soundness and completeness of our approach for the classification of $\text{ALCH(D)}^-$ and $\text{ALCHI}$ ontologies.
Chapter 5

Classification of \textit{ALCHO} with Soundness-Relaxed Strengthening

In this chapter, we introduce a single extension from \textit{ALCH} to \textit{ALCHO}, adding nominals. Unlike the two extensions in chapter 4 that are based on soundness-preserved strengthening, \textit{ALCHO} is based on soundness-relaxed strengthening. The procedure first creates $O_w$ and $O_s$, and then classifies each of them with a main $L_b$-reasoner MR and produce $H_w$ and $H_s$, correspondingly. Subsumptions in $H_w$ are sound but may not be complete w.r.t $O$, whereas subsumptions in $H_s$ are complete but may not be sound, i.e., soundness-relaxed strengthening. Then, an assistant $L_o$-reasoner AR is used to filter out unsound subsumptions in $H_s \setminus H_w$. Those that remain are added to $H_w$ resulting in the sound and complete classification of $O$. Since we employ two reasoners MR and AR in the procedure, we call it hybrid reasoning.
The rest of the chapter is organized as follows: Section 5.1 gives an overview of our hybrid classification procedure for \texttt{ALCHO} ontologies. Section 5.2 presents the preprocessing and weakening step. Section 5.3 presents the strengthening step. Section 5.4 explains the detail of canonical model used in previous sections and Section 5.5 demonstrates the detail proofs of lemmas and theorems in Section 5.3.

### 5.1 Overall Procedure of Hybrid Reasoning

Algorithm 5.1 describes our hybrid procedure for classifying $O$. It consists of three stages: (1) a \textit{preprocessing stage} (line 1) during which the ontology is rewritten to simplify the forms of axioms in it; (2) a \textit{main classification stage} (lines 2 to 8) in which $O_w$ and $O_s$ are generated and classified using the MR; and (3) a \textit{verification stage} (lines 9 to 17) in which the subsumptions arising from just $O_s$ are verified using AR. We use notations $N_C \cap \text{Sig}(O)$ and $N_C \cap \text{Sig}(o)$ to denote the set of atomic concepts in $O$ before and after normalization, respectively, and $N_{C}^{\top,\bot} \cap \text{Sig}(O) = N_C \cap \text{Sig}(O) \cup \{\top, \bot\}$.

In the preprocessing stage, the \texttt{ALCHO} ontology $O$ is rewritten to contain only axioms of forms $\bigcap A_i \subseteq \bigcup B_j$, $A \subseteq \exists R.B$, $\exists R.A \subseteq B$, $A \subseteq \forall R.B$, $R \subseteq S$, or $N_a \equiv \{a\}$. The rewritten axioms preserve subsumptions in $O$. See section 5.2 for the transformation detail.

In the verification stage, there are some cases we hand over the classification work to AR: (1) $N_a \equiv \bot \in \mathcal{H}_s$; (2) $E \equiv \bot \in \mathcal{H}_s$ but $O \not\models E \equiv \bot$, then for every $F \in N_{C}^{\top,\bot} \cap \text{Sig}(O)$, $E \subseteq F \in \mathcal{H}_s$, while likely only few of them are in $\mathcal{H}_w$, thus...
Algorithm 5.1: HybridClassify($\mathcal{O}$)

**Input:** An $\mathcal{ALCHO}$ ontology $\mathcal{O}$

**Output:** The classification result of $\mathcal{O}$

1. preprocess $\mathcal{O}$;
2. $\mathcal{O}_w \leftarrow \mathcal{O}$ with nominal axioms $N_a = \{a\}$ removed; /* Weakening */
3. $\mathcal{H}_w \leftarrow \text{MR.classify}($ $\mathcal{O}_w$ $);$ /* Classify the weakened ontology. */
4. $\mathcal{O}^+ \leftarrow \text{getNominalStrAx}($ $\mathcal{O}_w$, $\mathcal{NP}$, $N_C^{\perp \perp}$ $\cap$ $\text{Sig}(\mathcal{O})$ $);$ /* from Algorithm 5.4 */
5. remove all $E \sqsubseteq F$ from $\mathcal{H}_w$ where $\langle E, F \rangle \notin N_C^{\perp \perp}$ $\cap$ $\text{Sig}(\mathcal{O})$ $\times$ $N_C^{\perp \perp}$ $\cap$ $\text{Sig}(\mathcal{O})$ $;$
6. if $\mathcal{O}^+ = \emptyset$ then return $\mathcal{H}_w$ $;$
7. $\mathcal{O}_s \leftarrow \mathcal{O}_w \cup \mathcal{O}^+$ $;$
8. $\mathcal{H}_s \leftarrow \text{MR.classify}($ $\mathcal{O}_s$ $)$ $;$ /* Classify the strengthened ontology. */
9. if $N_a \sqsubseteq \bot \in \mathcal{H}_s$ for some $N_a \in \mathcal{NP}$ then return $\text{AR.classify}(\mathcal{O})$ $;$
10. remove all $E \sqsubseteq F$ from $\mathcal{H}_s$ where $\langle E, F \rangle \notin N_C^{\perp \perp}$ $\cap$ $\text{Sig}(\mathcal{O})$ $\times$ $N_C^{\perp \perp}$ $\cap$ $\text{Sig}(\mathcal{O})$ $;$
11. if $\| \mathcal{H}_s \setminus \mathcal{H}_w \| / \| N_C \cap \text{Sig}(\mathcal{O}) \|$ $>$ $d$ then return $\text{AR.classify}(\mathcal{O})$ $;$
12. $\mathcal{H}_{ws} \leftarrow \mathcal{H}_w$ $;$
13. foreach $E \sqsubseteq \bot \in \mathcal{H}_s \setminus \mathcal{H}_w$ do
14.  if $\text{AR.isSatisfiable}(\mathcal{O}, E)$ then return $\text{AR.classify}(\mathcal{O})$ $;$
15.  else add $E \sqsubseteq \bot$ into $\mathcal{H}_{ws}$ $;$
16.  foreach $E \sqsubseteq F \in \mathcal{H}_s \setminus \mathcal{H}_w$ where $F \neq \bot$ do
17.  if not $\text{AR.isSatisfiable}(\mathcal{O}, E \sqcap \neg F)$ then add $E \sqsubseteq F$ into $\mathcal{H}_{ws}$ $;$
18. return $\mathcal{H}_{ws}$

$\mathcal{H}_s \setminus \mathcal{H}_w$ may be huge; (3) the fraction $\| \mathcal{H}_s \setminus \mathcal{H}_w \| / \| N_C \cap \text{Sig}(\mathcal{O}) \|$ is greater than a threshold $d$. In the latter two cases, the estimated work for the stage is more than using AR to classify $\mathcal{O}$. For (3) we set $d = 1.5$ in our implementation based on the experiments in [27].

In the main classification stage, the major work is to generate the $\mathcal{ALCH}$ ontologies $\mathcal{O}_w$ and $\mathcal{O}_s$. $\mathcal{O}_w$ is produced by simply removing all the nominal axioms of the form $N_a = \{a\}$ from preprocessed $\mathcal{O}$. Since $\mathcal{O}_w \subseteq \mathcal{O}$, $\mathcal{O} \vDash \mathcal{O}_w$ and so $\mathcal{H}_w \subseteq \mathcal{H}_o$, 

84
i.e. the classification result of \( O_w \) is sound w.r.t. \( O \), see Theorem 3.3.3 for proof.

**Example 5.1.1.** Consider the following normalized ontology \( O_{eq} \) which we will use as a running example:

\[
\begin{align*}
(1) & \quad A \sqsubseteq C \\
(2) & \quad A \sqsubseteq \exists R.E \\
(3) & \quad E \sqsubseteq N_a \\
(4) & \quad C \sqsubseteq \forall R.D \\
(5) & \quad D \sqsubseteq G \\
(6) & \quad A \sqsubseteq \exists S.N_a \\
(7) & \quad N_a \equiv \{a\} \\
(8) & \quad \exists S.D \sqsubseteq F
\end{align*}
\]

Classification result of \( O_{eq} \) is \( \mathcal{H}_{o,ex} = \{ A \sqsubseteq F, A \sqsubseteq C, E \sqsubseteq N_a, D \sqsubseteq G \} \), where \( A \sqsubseteq F \) is implied by axioms (1)–(4),(6)–(8). The weakened version \( O_{w,ex} \) of \( O_{eq} \) is obtained by removing nominal axiom (7). And its classification result \( \mathcal{H}_{w,ex} = \{ A \sqsubseteq C, E \sqsubseteq N_a, D \sqsubseteq G \} \). We can see that \( A \sqsubseteq F \), which requires (7) to imply, is missing in \( \mathcal{H}_{w,ex} \). We will see later how we add strengthening axioms to get \( A \sqsubseteq F \) in \( \mathcal{H}_{S,ex} \).

The most difficult part of the procedure is to find \( O_s \) which entails no fewer subsumptions than \( O \). A sufficient condition is that for any \( A, B \in N^T_C \cap \text{Sig}(O) \) such that \( O_s \not\models A \sqsubseteq B \), there exists a model \( I \) of \( O_s \) satisfying \( A \sqcap \neg B \), and it can be transformed to a model \( I' \) of \( O \) satisfying \( A \sqcap \neg B \). Since \( O_s \) is obtained from \( O_w \) by adding strengthening axioms, every model \( I \) of \( O_s \) satisfies all axioms in \( O \) except possibly the nominal axioms. The interpretation of each of these requires each \( N_a \in \mathbb{NP} \) to have exactly one instance, whereas for an arbitrary model \( I \) of \( O_s \), \( N^I_a \) could have zero or multiple instances. However, if for each \( N_a, N^I_a \neq \emptyset \) and all the instances can be interpreted to be identical, they can be replaced by a single instance. Concretely, these instance have the same label set:

85
Definition 5.1.2. Given an interpretation $I = (\Delta^I, \cdot^I)$, an atomic concept $A$ is called a **label** of an instance $x$ if $x \in A^I$. The set of all the labels of $x$ is named the **label set** of $x$ in $I$, denoted by $LS(x, I)$.

Given an interpretation in which two instances have the same label set, and suppose a new interpretation is built from the old one in which both instances become the same instance. Such a replacement is called a **condensation** – it condenses all of the different instances into one instance, and thus it transforms $I$ into a model that satisfies the nominal axiom for $N_a$. If such a condensation can be done for all nominal placeholders, then we can create a model for $O$.

The strengthening axioms are designed to make these condensations possible. They have the form $N_a \sqsubseteq X$ and $N_a \sqcap X \sqsubseteq \bot$ computed by Algorithm 5.4, and thus they force $X$ to be a label of the nominal instance $N_a$, or not to be one, respectively. By “manipulating” labels of nominal instances through these strengthening axioms, we can force them all to be identical in certain models, so that the condensations can occur.

The models we construct for transformation are variants of the canonical model constructed for $\mathcal{ALCH}$ ontologies [78]. Our model construction is introduced in Section 5.4. Given $F \in N^T_C \cap \text{Sig}(O)$, our approach ensures we can build a model of $O_s$, which satisfies every $E \sqcap \neg F$ where $O_s \not\models E \sqsubseteq F$. And every such model can be condensed to a model of $O$ satisfying $E \sqcap \neg F$. Stating the contrapositive: if $O \models E \sqsubseteq F$ then $O_s \not\models E \sqsubseteq F$. Thus classification of $O_s$ is complete. Soundness of our approach is ensured by the verification stage, and completeness is proved in the following section 5.3.1.
5.2 Preprocessing and Weakening

There are two tasks in this step. Task 1): we first rewrite the original ontology to make nominals occur only in the axioms of the forms $N_a = \{a\}$. We rewrite the related OWL 2 DL class expressions and axioms by their equivalent forms containing only singleton nominal concepts, according to Table 5.1. After that, for each $\{a\}$, we replace all its occurrences by a new concept $N_a$ and add an axiom $N_a = \{a\}$. We call $N_a$ a nominal placeholder for $a$ in the following sections. We write $\mathbb{NP}$ for the set of all nominal placeholders after normalization.

Table 5.1: Rewriting Nominals in OWL 2

<table>
<thead>
<tr>
<th>OWL 2 Syntax</th>
<th>Equivalent Forms</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class Expressions</strong></td>
<td></td>
</tr>
<tr>
<td>ObjectHasValue ($R a$)</td>
<td>$\exists R.{a}$</td>
</tr>
<tr>
<td>ObjectOneOf ($a_1 \ldots a_n$)</td>
<td>${a_1} \sqcup \ldots \sqcup {a_n}$</td>
</tr>
<tr>
<td><strong>Axioms</strong></td>
<td></td>
</tr>
<tr>
<td>ClassAssertion ($C a$)</td>
<td>${a} \sqsubseteq C$</td>
</tr>
<tr>
<td>SameIndividual ($a_1 \ldots a_n$)</td>
<td>${a_1} = \ldots = {a_n}$</td>
</tr>
<tr>
<td>DifferentIndividuals ($a_1 \ldots a_n$)</td>
<td>${a_i} \cap {a_j} = \bot$</td>
</tr>
<tr>
<td>ObjectPropertyAssertion ($R a b$)</td>
<td>${a} \sqsubseteq \exists R.{b}$</td>
</tr>
<tr>
<td>NegativeObjectPropertyAssertion ($R a b$)</td>
<td>${a} \sqcap \exists R.{b} \sqsubseteq \bot$</td>
</tr>
</tbody>
</table>

Task 2): Apart from the axioms of the forms $N_a = \{a\}$, the remaining axioms in the ontology are in DL $\mathcal{ALCH}$. These axioms are normalized using the normalization procedure explained in 4.2.1. The result ontology contains axioms of the forms $\prod A_i \sqsubseteq \bigcup B_j$, $A \sqsubseteq \exists R.B$, $\exists R.A \sqsubseteq B$, $A \sqsubseteq \forall R.B$, $R \sqsubseteq S$, $N_a = \{a\}$. From it we remove all the nominal axioms $N_a = \{a\}$, and obtain the weakened version of ontology $O_w$, which is sound based on Theorem 3.3.3.
5.3 Strengthening

The strengthening step is very complex. First in subsection 5.3.1, we analyze theoretically what kind of properties that a strengthened ontology must satisfy so that it can be complete w.r.t $O$. Then in subsection 5.3.2, we demonstrate the concrete process on how to compute the strengthened ontology which is complete.

5.3.1 Sufficient Properties for Completeness of Strengthening

In this section, we will introduce some important definitions, lemmas, property and theorems. We use these to define conditions that extension procedure from $\mathcal{ALCH}$ to $\mathcal{ALCHO}$ must satisfy so that the completeness of the procedure can be ensured. All the proofs of the lemmas, property and theorems are in section 5.5.

Here we briefly introduce some notations used later. A series of strengthening transformation is required. We write $O^+$ for any intermediate (including final) version of strengthening axioms, $O^+_w$ for the corresponding intermediate (including final) version of strengthened ontology where $O^+_w = O_w \cup O^+$. We write $O_s$ for the final version of strengthened ontology. Given $E, F \in N^T, \bot \cap \text{Sig}(O)$ such that $O^+_w \not\models E \subseteq F$, a canonical model $I_{[O^+_w, <_F]}$ of $O^+_w$ is constructed by first computing a saturation $S_{O^+_w}$ of $O^+_w$ and then defining a model based on a total order $<_F$. The definition of construction of the canonical model is in section 5.4.

The computation of saturation is as follows: Given $O^+_w$, the saturation $S_{O^+_w}$ is initialized as

$$\{\text{init}(A) \mid A \in N_C \cap \text{Sig}(O)\} \cup \{\text{init}(N_a) \mid N_a \in \mathbb{NP}\}$$
Then $S_{O^+}$ is expanded by iteratively applying the inference rules in Table 5.2 and adding the conclusions into $S_{O^+}$ until reaching a fixpoint. During this process, existing axioms in $S_{O^+}$ are used as premises and axioms in $O^+_w$ are used as side conditions. $S_{O^+}$ contains axioms of the forms $\text{init}(H), H \subseteq M \cup A$ and $H \subseteq M \cup \exists R.K$. Note Table 5.2 is modified from Table 3 in [78] by using $R^+_A$ and $R_{\text{init}}$ to initialize contexts whenever necessary. The conjunction $H$ that occurs in the premises or conclusions of the inference rules is called the context of the inference.

Table 5.2: Complete Inference Rules for Normalized $\mathcal{ALCH}$ ontologies

<table>
<thead>
<tr>
<th>Rule</th>
<th>Premise</th>
<th>Conclusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R^+_A$</td>
<td>$H \subseteq A$</td>
<td>$A \in H$</td>
</tr>
<tr>
<td>$R^-_A$</td>
<td>$H \subseteq N \cup A$</td>
<td>$H \subseteq N$</td>
</tr>
<tr>
<td>$R_{\text{init}}$</td>
<td>$H \subseteq M \cup \exists R.K$</td>
<td>$\text{init}(K)$</td>
</tr>
<tr>
<td>$R^n_i$</td>
<td>$H \subseteq \bigsqcup_{i=1}^n N_i \cup M$</td>
<td>$\prod_{i=1}^n A_i \subseteq M \in O^+_w$</td>
</tr>
<tr>
<td>$R^+_3$</td>
<td>$H \subseteq M \cup \exists R.K, K \subseteq N \cup A$</td>
<td>$\exists S.A \subseteq B \in O^+_w$</td>
</tr>
<tr>
<td>$R^+_3$</td>
<td>$H \subseteq M \cup B \cup \exists R.(K \cap \neg A)$</td>
<td>$R \subseteq^*_O S$</td>
</tr>
<tr>
<td>$R^+_3$</td>
<td>$H \subseteq N \cup A$</td>
<td>$A \subseteq \exists R.B \in O^+_w$</td>
</tr>
<tr>
<td>$R^+_3$</td>
<td>$H \subseteq N \cup \exists R.B$</td>
<td>$R \subseteq^*_O S$</td>
</tr>
</tbody>
</table>

Definition 5.3.1. In an interpretation $I = (\Delta^I, \cdot^I)$, an atomic concept $L$ is called a **condensing label** if (1) $L^I \neq \emptyset$ and (2) for any $x, y \in L^I$, $LS(x, I) = LS(y, I)$.

If a label applied to some instance is a condensing label then every instance to which it applies has the same label set. This means the label sets of all such instances are identical and can be condensed into one instance.
Definition 5.3.2. Given a model $I$ of an $\mathcal{ALCHO}$ ontology $O$ and an individual name $x_L$, we define a condensation function $\text{condense}(L, x_L, I)$ that transforms $I$ into an interpretation $I' = (\Delta I', r_I')$ as follows:

1). Let $n$ be a fresh instance which is not in $\Delta I$, and $r$ be a replacement function $r(x) = \begin{cases} n & x \in L_I \\ x & \text{otherwise} \end{cases}$

2). $\Delta I' = \{r(x) \mid x \in \Delta I\}$

3). For each concept $A$, role $R$ and individual $o$ in $O$,

$A_{I'} = \{r(x) \mid x \in A_I\}, R_{I'} = \{(r(x), r(y)) \mid (x, y) \in R_I\}, o_{I'} = r(o_I), x_L' = n$

We say that each $x \in L_I$ is condensed into $n$. We also say $I$ is condensed to $I'$.

Definition 5.3.3. $H$ is a potentially supporting context of $A$ in $O_w^+$ if $H \subseteq M \cup A \in S_{O_w^+}$.

Definition 5.3.4. A concept $X$ is called a Potentially Supporting concept (PS) of some $A$ in $O_w^+$ if either $X$ or $\neg X$ is a conjunct of a potentially supporting context $H$ of $A$ in $O_w^+$. The set of all PS of $A$ in $O_w^+$ is denoted by $\text{PS}_A = \{X \mid X \subseteq O_w^+ \text{ is a PS of } A\}$.

Example 5.3.5. (PS) Consider the $\mathcal{ALCH}$ ontology $O_{w,ex}$ in Example 5.1.1, it can be seen as an ontology $O_{w,ex}^+$ where $O_{\emptyset}^+ = \emptyset$, and $\mathbb{M} = \{N_a\}$. We obtain three axioms in $O_{w,ex}^+$ which contains potentially supporting context of $N_a$. The axioms are: 1). $E \subseteq N_a$, 2). $N_a \subseteq N_a$ and 3). $E \cap D \subseteq N_a$. The three potentially supporting contexts $H$ of $N_a$ are: $H_1 = E$, $H_2 = N_a$ and $H_3 = E \cap D$. $H_1$ contains one PS of $N_a$: $E$; $H_2$ contains one PS of $N_a$: $N_a$; $H_3$ contains two PS of $N_a$: $E$ and $D$; So the set of PS of $N_a$ is $\text{PS}_{[N_a, O_{w,ex}]} = \{N_a, D, E\}$.  

90
Property 5.3.6. We say $O^+_w$ is decisive if for each $N_a \in \mathbb{NP}$, each $X \in \text{PS}_{[N_a, O^+_w]}$, either $O_{w,ex} \models N_a \sqsubseteq X$ or $O_{w,ex} \models N_a \sqcap X \sqsubseteq \bot$ holds.

Example 5.3.7. (decisive) for the above $\text{PS}_{[N_a, O^+_w]}$, if each $X \in \{E, N_a, D\}$, either $O_{w,ex} \models N_a \sqsubseteq X$ or $O_{w,ex} \models N_a \sqcap X \sqsubseteq \bot$ holds, then $O_{w,ex}$ is decisive.

Lemma 5.3.8. Given $O^+_w$ and $N_a \in \mathbb{NP}$, if (1) $O^+_w$ is decisive, and (2) $O^+_w \models E \sqsubseteq \bot$. Then for any $F \in N^T_\bot \cap \text{Sig}(O)$, $N_a$ is a condensing label in $I_{[O^+_w, \prec]}$.

Lemma 5.3.9. Let $I$ be a model of an $\mathcal{ALCHO}$ ontology $O$ satisfying $E \sqcap \neg F, E, F \in N^T_\bot \cap \text{Sig}(O)$, where $L$ is a condensing label in $I$. Then $I' = \text{condense}(L, x_L, I)$ is a model of $O \cup \{L = \{x_L\}\}$ satisfying $E \sqcap \neg F$.

Lemma 5.3.10. Given some $O^+_w$ and $F \in N^T_\bot \cap \text{Sig}(O)$, if in $I_{[O^+_w, \prec]}$ every $N_a \in \mathbb{NP}$ is a condensing label, then for each $E \in N^T_\bot \cap \text{Sig}(O)$ s.t. $O^+_w \models E \sqsubseteq F$, there is a model of $O$ satisfying $E \sqcap \neg F$.

Theorem 5.3.11. If the $O^+_w$ we compute satisfies: (1) $O^+_w$ is decisive and (2) all $N_a \in \mathbb{NP}$ are satisfiable in $O^+_w$. Then the classification result of $O^+_w$ is complete w.r.t. $O$.

Proof. Let $E, F \in N^T_\bot \cap \text{Sig}(O)$ be concepts such that $O^+_w \models E \sqsubseteq F$. Since conditions (1) and (2) of Lemma 5.3.8 are satisfied, all $N_a \in \mathbb{NP}$ are condensing labels in the canonical model $I_{[O^+_w, \prec]}$. By Lemma 5.3.10, the model can be condensed to a model $I'$ of $O$ for $E \sqcap \neg F$, proving $O \models E \sqsubseteq F$. So the classification result of $O^+_w$ is complete w.r.t. $O$. □
In the next section, we demonstrate how we will compute an $O^+_w$ satisfying condition (1) in Theorem 5.3.11 i.e., is decisive. Such an $O^+_w$ can also satisfy condition (2) in most cases as in our experiments, therefore completeness is achieved. If in a few cases that such an $O^+_w$ ends up not satisfying condition (2), then we hand over the work to AR. The classification result is still complete.

5.3.2 Generating the Strengthened Ontology

In this section we show how to create a decisive $O^+_w$. The property requires that the strengthening axioms should be of the form $N_a \sqsubseteq X$ or $N_a \sqcap X \sqsubseteq \bot$. We first briefly introduce the strengthening axioms selection function $\text{chooseStrAxiom}$. Based on that, we explain an initial idea to create a decisive $O^+_w$. Then, we introduce how to compute $\text{OPS}_{[N_a, O^+_w]}$ which is an overestimation of $\text{PS}_{[N_a, O^+_w]}$ from $O^+_w$ without doing saturation, and give the naive and improved algorithms to compute $O^+_w$.

Definition 5.3.12. A strengthening axiom selection $\text{chooseStrAxiom}$ is a function that takes an $N_a \in \mathbb{NP}$, $X \in N_C \cap \text{Sig}(a)$, and $O_w$ and returns:

1. $\emptyset$ only if $O_w \models N_a \sqsubseteq X$ or $O_w \models N_a \sqcap X \sqsubseteq \bot$;

2. a choice between $N_a \sqsubseteq X$ or $N_a \sqcap X \sqsubseteq \bot$.

Initial idea for generating a decisive $O^+_w$: To create such an ontology, a straightforward idea is to start from $O_w$, generate saturation $S_{O^+_w}$ of $O_w$, obtain $\text{PS}_{[N_a, O^+_w]}$ from $S_{O^+_w}$ for all $N_a \in \mathbb{NP}$. For each pair $(N_a, X)$, $X \in \text{PS}_{[N_a, O^+_w]}$, we add a strengthening axiom determined by $\text{chooseStrAxiom}$ function into $O^+_w$. Then, add $O^+_w$
into $O_w$ and obtain $O_w^{1+}$. Next, we generate saturation again based on $O_w^{1+}$. Since the impact of $O_w^{1+}$, $PS_{[N_a, O_w^{1+}]} \supseteq PS_{[N_a, O_w]}$. Then we repeat the above process until a fixpoint at which $PS_{[N_a, O_w^{j+1}]} = PS_{[N_a, O_w^{j}]}$ for all $N_a \in \mathbb{NP}$, we call the final $O_w^{j+1}$ as $O_s$ which is decisive. However to implement such a procedure generating the saturation for $PS_{[N_a, O_w]}$ is costly. Thus, instead of $PS_{[N_a, O_w]}$, we compute the overestimated $PS_{[N_a, O_w^{1+}]}$ called $OPS_{[N_a, O_w^{1+}]}$.

**Definition 5.3.13.** Given an ontology $O_w^+$, $OPS_{[N_a, O_w^+]}$ is a set of overestimated potentially supporting concepts we compute such that $OPS_{[N_a, O_w^+]} \supseteq PS_{[N_a, O_w^+]}$ for each $N_a \in \mathbb{NP}$.

We use Algorithm 5.2 to compute $OPS_{[X, O_w^+]}$. Based on Definitions 5.3.3 and 5.3.4, $PS_{[X, O_w^+]}$ includes all the atoms of $H$ such that $H \sqsubseteq M \sqcup X \in S_{O_w^+}$. Without a real procedure generating the saturation for $PS_{[X, O_w^+]}$ as explained above, we actually compute $OPS_{[X, O_w^+]}$ based on analyzing the relationships among the premises, side conditions and conclusions of the inference rules in Table 5.2. The procedure of Algorithm 5.2 can be divided into three parts:

1. (line 2 to 9) We first conduct a search in the converse direction of all possible derivation paths (Definition 5.5.1 in Section 5.5) of a conclusion $H \sqsubseteq M \sqcup X$ for all possible $H$s. In the search we maintain a set $Pri_{[X, O_w^+]}$. Each concept $W \in Pri_{[X, O_w^+]}$ may be necessary to the derivation of $X$, and appears prior to $X$ in the derivation path. More precisely, $W$ corresponds to some potential intermediate conclusion $H \sqsubseteq M' \sqcup W$ which is a necessary conclusion for deriving $H \sqsubseteq M \sqcup X$. Since for any context $H, H \sqsubseteq A$ is the only conclusion
Algorithm 5.2: getOPS

**Input**: Normalized $\mathcal{ALCH}$ ontology $O^+_w$, a concept $X$, a set of nominal placeholders $NP$, the set of atomic classes $U$ in the original ontology

**Output**: A pair $(OPS_{[X,O^+_w]}, Pri_{[X,O^+_w]})$

1. $OPS_{[X,O^+_w]} ← \emptyset$; $Pri_{[X,O^+_w]} ← \emptyset$; $ToProcess ← \{X\}$; $Exists ← \emptyset$; $ToProcess \neq \emptyset$ do
2. take out a label $W$ from $ToProcess$;
3. if $W \not\in Pri_{[X,O^+_w]}$ then
4. add $W$ to $Pri_{[X,O^+_w]}$;
5. if $W = \top$ then stop the procedure and use AR to do the classification work;
6. foreach $\bigwedge A_i \sqsubseteq M \sqcup W \in O^+_w$ do select one $A_i$ and add it into $ToProcess$;
7. foreach $\exists S.Y \sqsubseteq W \in O^+_w$ and $R \sqsubseteq^* O_S$ and $B \sqsubseteq \exists R.Z \in O^+_w$ do
8. add $B$ into $ToProcess$;
9. foreach $W \in Pri_{[X,O^+_w]}$ do
10. if $W \in U$ or $W \in NP$ then add $W$ to $OPS_{[X,O^+_w]}$;
11. foreach $Y \sqsubseteq \forall S.W \in O^+_w$ and $R \sqsubseteq^* S$ and $B \sqsubseteq \exists R.Z \in O^+_w$ do add $\exists R.Z$ to $Exists$;
12. foreach $B \sqsubseteq \exists R.W \in O^+_w$ do add $\exists R.W$ to $Exists$;
13. foreach $\exists R.W \in Exists$ and $R \sqsubseteq^* S$ do
14. add $W$ to $OPS_{[X,O^+_w]}$;
15. foreach $Y \sqsubseteq \forall S.Z \in O^+_w$ do add $Z$ to $OPS_{[X,O^+_w]}$;
16. foreach $\exists S.Z \sqsubseteq Y \in O^+_w$ do add $Z$ to $OPS_{[X,O^+_w]}$;
17. return $(OPS_{[X,O^+_w]}, Pri_{[X,O^+_w]})$
that can be derived from init(H), at least one of such A, which is a conjunct of H, is in Pri_{X,O;w}. We write \( C^X_H \) for such A. Pri_{X,O;w} contains at least one conjunct \( C^X_H \) for any potentially supporting context \( H \) of \( X \).

2. (lines 10 to 13) Check each concept \( W \in \text{Pri}_{X,O;w} \) to see whether it can be a conjunct \( C^X_H \) of some potential supporting context \( H \) of \( X \). If it can, we find the first conjunct \( C^1_H \) of \( H \) from \( C^X_H \) (see \( H^* \) in Lemma 5.5.14 in Section 5.5), which is either in \( U \) or \( NP \) in line 11 or the filler of concepts in Exists from line 12 to 13, is the initial concept \( H \) starts from.

3. (line 14 to 17) Find all the other conjuncts of \( H \) based on \( C^1_H \) by searching along the derivation paths.

**Lemma 5.3.14.** Given \( O^+_w \) and a concept \( A \), \( \text{OPS}_{[A,O^+_w]} \) returned by Algorithm 5.2 preserves \( \text{OPS}_{[A,O;w]} \supseteq \text{PS}_{[A,O;w]} \).

**Example 5.3.15.** (OPS) Consider again the ontology \( O_{w,ex} \) in Example 5.1.1

The Execution of Alg. 5.2: getOPS\((O_{w,ex}, N_a, \{N_a\}, \{A, C, D, E, F, G\})\)

<table>
<thead>
<tr>
<th>Line</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 to 9</td>
<td>( \text{Pri}<em>{[N_a,O</em>{w,ex}]} = {N_a, E} )</td>
</tr>
<tr>
<td>10</td>
<td>( \text{OPS}<em>{[N_a,O</em>{w,ex}]} = {N_a, E} )</td>
</tr>
<tr>
<td>11</td>
<td>Exists = ( \exists R.E )</td>
</tr>
<tr>
<td>12</td>
<td>( \text{OPS}<em>{[N_a,O</em>{w,ex}]} = {N_a, E, D} )</td>
</tr>
<tr>
<td>32</td>
<td>Return ( \text{OPS}<em>{[N_a,O</em>{w,ex}]} = {N_a, E, D}, \text{Pri}<em>{[N_a,O</em>{w,ex}]} = {N_a, E} )</td>
</tr>
</tbody>
</table>

For each \( X \in \text{OPS}_{[X,O^+_w]} \), we will use function chooseStrAxiom to return a strengthening axiom. Note in Definition 5.3.12 for chooseStrAxiom, how can we know
Actually in Algorithm 5.1 line 4 executes getOPS($O_w, N_a, \mathbb{N}_p, U$) internally in getNominalStrAx before line 3. Then for each $N_a \in \mathbb{N}_p$, each $X \in \text{OPS}_{\{N_a, O_w\}}$, we add a testing axiom $X_a \subseteq N_a \cap X$ into $O_w$ in the first round classification and obtain $\mathcal{H}_w$, where $X_a$ is a fresh concept for $O_w$.

In Algorithm 5.1 and Definition 5.3.12 we ignore this detail and only mention using $O_w$ just for simplifying explanation. If $X_a \subseteq \bot$ is found in $\mathcal{H}_w$, then we simply say $O_w \models N_a \cap X \subseteq \bot$. If $N_a \subseteq X$ or $N_a \cap X \subseteq \bot$ is implied by $\mathcal{H}_w$, we do not add any strengthening axiom for $X$ into $O^{i+1}$. We remove the extra testing results from $\mathcal{H}_w$ in line 5 of Algorithm 5.1 When choosing between $N_a \subseteq X$ and $N_a \cap X \subseteq \bot$, we use the heuristics that if $X$ corresponds to a union concept before normalization, and $N_a \subseteq X$ is not implied, then we add $N_a \cap X \subseteq \bot$. For other cases, we add $N_a \subseteq X$.

**Example 5.3.16.** For the concepts in $\text{OPS}_{\{N_a, O_w, \text{ex}\}}$, since none of the 4 axioms $N_a \subseteq E, E_a \subseteq \bot, N_a \subseteq D$ or $D_a \subseteq \bot$ is in $\mathcal{H}_w, \text{ex}$, and assume $E$ and $D$ do not contain union concepts before normalization, we add $N_a \subseteq E$ and $N_a \subseteq D$ as strengthening axioms $O^{1+}_{\text{ex}}$.

Using $\text{OPS}_{\{X, O^+_w\}}$ instead of $\text{PS}_{\{X, O^+_w\}}$, we design a naive Algorithm 5.3 based on the above Initial Idea to generate $O^+$ for $O^+_w$ so that $O^+_w$ is decisive. The execution process is as follows:

1) $O^{0+} = \emptyset$.
2) Compute $\text{OPS}_{\{N_a, O^+_w\}}$ for $\forall N_a, N_a \in \mathbb{N}_p$ for the $O^{i+}$.
3) $O^{i+1} = \{\text{chooseStrAxiom}(N_a, X) \mid \forall N_a, X, N_a \in \mathbb{N}_p, X \in \text{OPS}_{\{N_a, O^+_w\}}\}$
Algorithm 5.3: getNominalStrAxNaive (Naive algorithm for computing strengthening axioms)

**Input:** Normalized $\mathcal{ALCH}$ ontology $O_w$, a set of nominal placeholders $\mathcal{NP}$, the set of atomic classes $U$ in the original ontology

**Output:** Strengthening axioms $O^+$

1. $O^+ \leftarrow \emptyset$;
2. repeat
   3. newAxioms = $\emptyset$;
   4. foreach $N_a \in \mathcal{NP}$ do
      5. $O^+ \leftarrow O_w \cup O^+$;
      6. $(\text{OPS}_{[X,O^+_w]}, \text{Pri}_{[X,O^+_w]}) \leftarrow \text{getOPS}(O^+_w, N_a, \mathcal{NP}, U)$; /* from Algorithm 5.2 */
      7. foreach $X \in \text{OPS}_{[X,O^+_w]}$ do
         8. newAxioms $\leftarrow$ newAxioms $\cup$ chooseStrAxiom($N_a$, $X$);
      9. $O^+ \leftarrow O^+ \cup$ newAxioms;
   10. until newAxioms = $\emptyset$;
11. return $O^+$

In each loop, we add the new $O^+_i$ into $O_w$ and generate $O^+_w$. When the process converges, $O^{i+1}_w = O^+$. Since $O^{i+1}_w$ is computed from $\text{OPS}_{[N_a,O^+_w]}$, and $\text{OPS}_{[N_a,O^+_w]} \supseteq \text{PS}_{[N_a,O^+_w]}$ for all $N_a$ and for any $i$, thus Algorithm 5.3 guarantees that $O^+_w$ is decisive.

**Theorem 5.3.17.** Let $O^+$ be strengthening axiom produced from Algorithm 5.3, $O^+_w = O_w \cup O^+$ is decisive.

**Theorem 5.3.18.** Let $O^+$ be strengthening axioms computed from Algorithm 5.4, $O^+_w = O_w \cup O^+$ is decisive. (Proof see Appendix 5.5, the following is just an intuitive explanation)

Algorithm 5.4 improves Algorithm 5.3 by avoiding repetitive search process. In each loop of Algorithm 5.3, Algorithm 5.2 getOPS uses the axioms of the input.
$O_w^{i+1}$ to compute $OPS_{[N_a, O_w^{i+1}]}$ for $N_a$. During the computation, only searching on $O_w^{i+1} \setminus O_w^i$ is new. The majority work which is searching on $O_w^i$ is repeated in each iteration. Thus Algorithm 5.4 improves this and only executes $getOPS$ based on $O_w^i$ for once in line 3. In Algorithm 5.2 and 5.3, a strengthening axiom $\alpha$ can take effect only in the following situation: If $\alpha = \{N_b \subseteq X \} \in O_w^i$, $X \in Pri_{[N_a, O_w^i]}$, then $N_b \in OPS_{[N_a, O_w^i]}$. If chooseStrAxiom($N_a, N_b$) return $N_a \nsubseteq N_b$ in $O_w^{i+1}$, then $N_a \in OPS_{[N_b, O_w^{i+1}]}$. That means because of $\alpha$, in the end, $OPS_{[N_a, O_w^i]} = OPS_{[N_b, O_w^i]}$. In Algorithm 5.4 without really computing each of $O_w^i$ and repeatedly search on them, we achieve similar results based on the merge criteria in line 6 and the merge operation in line 7. Assume $N_b \in g_i.nominals$ and $N_a \in g_j.nominals$, $X \in g_i.ops \cap g_j.pri \neq \emptyset$ in line 6. Then, $X \in g_i.ops$ means $N_b \subseteq X$ will possibly be a strengthening axiom, $X \in g_j.pri$ means $X$ is possibly in $N_a$’s $Pri$, then $g_i$ and $g_j$ are merged into one group, and finally $OPS_{[N_a, O_w^i]} = OPS_{[N_b, O_w^i]}$.

**Example 5.3.19.** In Algorithm 5.3 $O^{1+}_ex = O^{2+}_ex$. Thus, the loop from line 2 to 9 repeats twice. In Algorithm 5.4 since $NP = \{N_a\}$, the merge process from line 6 to 7 does not happen. For both algorithms, that means $O^{1+}_ex$ does not have impact in later saturation.

Note the naive Algorithm 5.3 is only used for demonstrating our initial idea. In our implementation, we used the improved Algorithm 5.4. All three Algorithms 5.2, 5.3, and 5.4 have polynomial complexity. In Algorithm 5.2 the number of iterations in all levels of loops are bounded by the number of axioms or concepts in $O_w$, and thus have polynomial complexity. In Algorithm 5.3 the number of iterations is bounded by the size of $O^+$, which is also polynomial. In Algorithm 5.4

98
**Algorithm 5.4: getNominalStrAx** (Calculate strengthening axioms for nominals)

**Input**: Normalized \( \forall LCH \) ontology \( O_w \), a set of nominal placeholders \( \mathbb{NP} \), the set of atomic classes \( U \) in the original ontology

**Output**: Strengthening axioms \( O^+ \)

1. groups \( \leftarrow \emptyset \);
2. \( \textbf{foreach} ~ N_a \in \mathbb{NP} \textbf{ do} \)
   3. \( \langle \text{OPS}_{[N_a,O_w]}, \text{Pri}_{[N_a,O_w]} \rangle \leftarrow \text{getOPS}(O_w, N_a, \mathbb{NP}, U); \quad \text{/* from Algorithm 5.2 */} \)
   4. create a group \( g \) with \( g.\text{nominals} = \{N_a\}, g.\text{ops} = \text{OPS}_{[N_a,O_w]}, g.\text{pri} = \text{Pri}_{[N_a,O_w]}; \)
   5. add \( g \) into groups;
6. \( \textbf{while} \) there exists \( g_i, g_j \in \text{groups} \) such that \( g_i.\text{ops} \cap g_j.\text{pri}, \emptyset \) \( \textbf{do} \)
   7. merge \( g_i, g_j \) into one group \( g \), whose properties are unions of corresponding properties of \( g_i \) and \( g_j \); remove \( g_i, g_j \) from groups and add \( g \);
8. \( \textbf{foreach} \ g \in \text{groups} \textbf{ do} \)
   9. \( \textbf{foreach} \ N_a \in g.\text{nominals} \textbf{ do} \)
   10. \( \textbf{foreach} \ X \in g.\text{ops} \textbf{ do} \)
   11. \( O^+ \leftarrow O^+ \cup \text{chooseStrAxiom}(N_a,X); \)
12. \( \textbf{return} \ O^+ \)

The merge loop can only continue for at most \( ||\mathbb{NP}|| \) times. So all the algorithms are polynomial of the size of \( O_w \) and terminate.

**Example 5.3.20.** : The Overall Execution Results from Alg. 5.1

1. **Weakened Ontology** \( O_w \): \( N_a \equiv \{a\} \) removed \( \text{Line 2 of Alg. 5.1} \)
2. **Class Hierarchy** \( H_w \): \( A \sqsubseteq C \quad E \sqsubseteq N_a \quad D \sqsubseteq G \) \( \text{Line 3 of Alg. 5.1} \)
3. **Strengthening axioms** \( O^+ \): two axioms added: \( N_a \sqsubseteq E \quad N_a \sqsubseteq D \)
4. **For (3):** Line 4 of Alg. 5.1 return from Alg. 5.4 \( \text{For (5): Line 8 of Alg. 5.1} \)
5. **Class Hierarchy** \( H_x \): \( A \sqsubseteq C \quad E \sqsubseteq N_a \quad D \sqsubseteq G \quad A \sqsubseteq F \quad N_a \sqsubseteq E \quad N_a \sqsubseteq D \)
6. **Verify the following 6 pairs (indirect subsumptions included):** Line 16 to 17
(7) \( A \subseteq F \quad E \subseteq D \quad E \subseteq G \quad N_a \subseteq D \quad N_a \subseteq E \quad N_a \subseteq G \), 1 pair validated:
\( A \subseteq F \)

(8) Class Hierarchy \( H_{w=H} : A \subseteq C \quad A \subseteq F \quad E \subseteq N_a \quad D \subseteq G \). Line 18 of Alg. 5.1

5.4 Canonical Model Construction

Given \( F \in N_C^{\top,\bot} \cap \text{Sig}(O) \) and \( O_w^+ \), our target is to construct a model \( I_{[O_w^+,\prec_F]} \) for \( O_w^+ \) satisfying \( E \sqcap \neg F \) for any \( E \in N_C^{\top,\bot} \cap \text{Sig}(O) \) such that \( O_w^+ \not\models E \sqsubseteq F \). We write \( I \) for \( I_{[O_w^+,\prec_F]} \) when the parameters are not important. The construction is done by first computing a saturation \( S_{O_w^+} \) of \( O_w^+ \) and then defining a model based on it. \( S_{O_w^+} \) contains axioms of the forms \( \text{init}(H) \), \( H \sqsubseteq M \sqcup A \) and \( H \sqsubseteq M \sqcup \exists R.K \) derived using the inference rules.

(1) Computation of saturation

Given an \( \mathcal{ALCH} \) ontology \( O_w^+ \), the saturation \( S_{O_w^+} \) is initialized as

\[
\{ \text{init}(A) \mid A \in N_C^{\top} \} \cup \{ \text{init}(N_a) \mid N_a \in \mathbb{NP} \}
\]

Then \( S_{O_w^+} \) is expanded by iteratively applying the inference rules in Table 5.2 and adding the conclusions into \( S_{O_w^+} \) until reaching a fixpoint. An inference rule is applied by using existing axioms in \( S_{O_w^+} \) as premises and axioms in \( O_w^+ \) as side conditions. We write \( O_w^+ \vdash \alpha \) for every \( \alpha \in S_{O_w^+} \) derived from \( O_w^+ \).

The inference process is obviously sound, i.e. if \( O_w^+ \vdash \alpha \) then \( O_w^+ \models \alpha \).
Model construction

We define a total order \( \prec \) over all the concepts in \( O^+_w \) such that \( F \) has the least order. If \( F \) is \( \bot \) or \( \top \), the order can be any arbitrary order. We define the domain \( \Delta^\mathcal{I} \) of \( \mathcal{I}_{[O^+_w, \prec]} \) as

\[
\Delta^\mathcal{I} := \{ x_H \mid \text{init}(H) \in S_{O^+_w} \text{ and } H \subseteq \bot \not\in S_{O^+_w} \}
\]

where \( x_H \) is an instance introduced for \( H \). \( \Delta^\mathcal{I} \) is nonempty because if \( O^+_w \) is consistent, \( \top \subseteq \bot \not\in S_{O^+_w} \). Since \( \text{init}(\top) \in S_{O^+_w} \), \( x_\top \) exists.

To define the interpretation for atomic concepts, we first construct the label set \( LS(x_H, \mathcal{I}) \) for each instance \( x_H \). In this section, we write \( \mathcal{I}_H \) for \( LS(x_H, \mathcal{I}) \). Let \( A_i \) be the concept with the \( i \)th order from the smallest to the largest according to \( \prec \). For convenience we write \( M \prec \mathcal{I} A_i \) if for each disjunct \( A \) in \( M \), \( A \prec \mathcal{I} A_i \). Let \( \mathcal{I}^i_H \) be a sequence where \( \mathcal{I}^0_H := \emptyset \), and \( \mathcal{I}^i_H \) is defined as

\[
\mathcal{I}^i_H := \begin{cases} 
\mathcal{I}^{i-1}_H \cup \{ A_i \} & \text{if there exists } M \prec \mathcal{I} A_i \text{ such that } \text{init}(M) + H \subseteq M \cup A_i \text{ and } M \cap \mathcal{I}^{i-1}_H = \emptyset \\
\mathcal{I}^{i-1}_H & \text{otherwise}
\end{cases}
\]

The last element of the sequence is defined as \( LS(x_H, \mathcal{I}) \). With the \( LS(x_H, \mathcal{I}) \) defined, the interpretation of an atomic concept \( A \) is defined as

\[
A^\mathcal{I} := \{ x_H \mid A \in LS(x_H, \mathcal{I}) \}
\]
The roles are interpreted to satisfy the axioms $H \subseteq M \cup \exists R.K$. For each role $R$ and each $H$ such that $x_H \in \Delta^I$, define

$$I^R_H := \{ K | \exists M : O^+_w \vdash H \subseteq M \cup \exists R.K, M \cap I_H = \emptyset \}$$

A conjunction $K$ is said to be maximal in $LS^R(x_H, I)$ if there is no $K' \in I^R_H$ with a superset of conjuncts of $K$. Since $H \subseteq \bot \notin S_{O^+_w}$, by $R^\bot$ rule we have $K \subseteq \bot \notin S_{O^+_w}$. And by $R_{\text{init}}$ rule we have $\text{init}(K) \in S_{O^+_w}$. So $x_K$ is well-defined. The interpretation of roles is defined as

$$R^I := \bigcup_{R \subseteq O^+_w} \{ (x_H, x_K) | K \text{ is maximal in } I^R_H \}$$

The inference rules in Table 5.2 is modified from Table 3 in [78] by using $R^+_A$ and $R_{\text{init}}$ to initialize contexts only when necessary. The change affects only the validity of $x_K$ in the construction for $R^I$ which has been explained above, and the proof that $I$ satisfies each type of axiom can be kept unchanged from Simancik et. al. [78]. So $I$ is a model of the $\mathcal{ALCH}$ ontology $O^+_w$. Moreover, for any $E \in N_{C, \bot} \cap \text{Sig}(O)$, if $O^+_w \not\models E \subseteq F$, $O^+_w \not\models E \subseteq F$. Since $F$ has the least order in $\prec_F$, by the definition of $LS(x_E, I)$ we know $x_E \notin F^I$. Thus $x_E \in (E \cap \neg F)^I$, and $I_{[O^+_w, \prec_F]}^E$ satisfies $E \cap \neg F$. 

102


5.5 Detail Proofs of Lemmas and Theorems for Completeness

In this section, we will illustrate the concrete proofs of Lemmas and Theorems for completeness presented in section 5.3.1.

**Lemma 5.5.8.** Given $O'_w$ and $N_a \in \mathbb{NP}$, if (1) $O'_w$ is decisive, and (2) $O'_w \not\models N_a \subseteq \bot$.

Then for any $F \in N^\top_C \cap \text{Sig}(O)$, $N_a$ is a condensing label in $I_{[O'_w, \prec F]}$.

**Proof.** For simplicity we write $I$ for $I_{[O'_w, \prec F]}$ in this proof. Since $\text{init}(N_a) \in S_{O_w}$ and $O'_w \not\models N_a \subseteq \bot$, by the construction of $I$, $x_{N_a}$ exists and $x_{N_a} \in N^I_a$. Hence it is equivalent to show that for each $x_H \in N^I_a$, $LS(x_{N_a}, I) = LS(x_H, I)$.

We first show that for each $H$ such that $x_H \in N^I_a$, $x_{N_a} \in H^I$. Since $x_H \in N^I_a$, $H$ is a potentially supporting context of $N_a$. Let $H = \prod_{i=1}^n C^i_H$, where $C^i_H$ is $A$ or $\neg A$.

Since $O'_w$ is decisive, we have the following two cases:

- $O'_w \models N_a \subseteq C^i_H$ holds for all $C^i_H$, $1 \leq i \leq n$, then $O'_w \not\models N_a \subseteq H$. Since $x_{N_a} \in N^I_a$, $x_{N_a} \in H^I$.

- There exists some $i$ such that $O'_w \models N_a \cap C^i_H \subseteq \bot$, then $O'_w \not\models N_a \cap H \subseteq \bot$. By Lemma 3 of paper [78], $x_H \in H^I$, which contradicts with our assumption $x_H \in N^I_a$.

Next we prove $LS(x_{N_a}, I) \subseteq LS(x_H, I)$ by contradiction. Assume $LS(x_{N_a}, I) \not\subseteq LS(x_H, I)$, let $X$ be the concept in $LS(x_{N_a}, I) \setminus LS(x_H, I)$ with the smallest order. Since $X \in LS(x_{N_a}, I)$, there exists $N \prec F X$ such that $O'_w \vdash N_a \subseteq N \cup X$ and $N \cap LS(x_{N_a}, I) = \emptyset$. 

103
\[\therefore O_w^+ \vdash N_a \subseteq N \sqcup X \therefore O_w^+ \models N_a \subseteq N \sqcup X\]

\[\therefore x_H \in N_a^f \land x_H \notin X^f \therefore x_H \in N^f \therefore LS(x_H, I) \cap N \neq \emptyset\]

In the above proof, if \( N = \bot \), a contradiction arises with \( x_H \in N^f \). Otherwise, let \( Y \in LS(x_H, I) \cap N \), there must exist \( N' \prec_F Y \) s.t. \( O_w^+ \vdash H \subseteq N' \sqcup Y \) and \( LS(x_H, I) \cap N' = \emptyset \).

\[\therefore O_w^+ \vdash H \subseteq N' \sqcup Y \text{ and } x_{N_a} \in H^f \therefore x_{N_a} \in (N' \sqcup Y)^f\]

\[\therefore N' \prec_F Y \text{ and } Y \in N \text{ and } N \prec_F X \therefore N' \prec_F X\]

Since \( X \) is the smallest in \( LS(x_{N_a}, I) \setminus LS(x_H, I) \), \( N' \prec_F X \) and \( LS(x_H, I) \cap N' = \emptyset \), we have \( LS(x_{N_a}, I) \cap N' = \emptyset \) (it is trivially true if \( N' = \bot \)), and \( x_{N_a} \notin N'^f \). Given \( x_{N_a} \in (N' \sqcup Y)^f \), we have \( Y \in LS(x_{N_a}, I) \), this contradicts with \( N \cap LS(x_{N_a}, I) = \emptyset \).

So we conclude that \( LS(x_{N_a}, I) \setminus LS(x_H, I) = \emptyset \) and \( LS(x_{N_a}, I) \subseteq LS(x_H, I) \).

Finally we need to prove \( LS(x_H, I) \subseteq LS(x_{N_a}, I) \). For each \( X \in LS(x_H, I) \), there exists \( N \prec_F X \) such that \( O_w^+ \vdash H \subseteq N \sqcup X \) and \( N \cap LS(x_H, I) = \emptyset \).

\[\therefore LS(x_{N_a}, I) \subseteq LS(x_H, I) \therefore N \cap LS(x_{N_a}, I) = \emptyset \therefore x_{N_a} \notin N^f\]

\[\therefore x_{N_a} \in H^f \land x_{N_a} \notin N^f \therefore x_{N_a} \in X^f\]

Thus we conclude \( X \in LS(x_{N_a}, I) \). \( \square \)
Lemma 5.5.9. Let $\mathcal{I}$ be a model of an $\mathcal{ALCHO}$ ontology $\mathcal{O}$ satisfying $E \cap \neg F$, and $E, F \in N_{C}^{\vee} \cap \text{Sig}(\mathcal{O})$, where $L$ is a condensing label in $\mathcal{I}$. Then $\mathcal{I}' = \text{condense}(L, x_L, \mathcal{I})$ is a model of $\mathcal{O} \cup \{L \equiv \{x_L\}\}$ satisfying $E \cap \neg F$.

Proof. By the definition of condensing label, we have: (1) $L^I \neq \emptyset$; (2) for all $x \in L^I$, $LS(x, I)$ are the same. By (1) and the definition of $r$ in $\text{condense}(L, x_L, I)$, we have $L'^I = \{x_L'^I\}$, so the axiom $L \equiv \{x_L\}$ is satisfied. By (2), we can further prove $LS(x, I) = LS(r(x), I')$ holds for all $x \in \Delta^I$. Next we need to prove $I' \models \alpha$ from $I \models \alpha$ for any axiom $\alpha$ in $\mathcal{O}$. We do a case-by-case analysis for every possible form of $\alpha$:

- $\alpha = \bigcap A_i \subseteq \bigcup B_j$ Assume $x' \in (\bigcap A_i)^I'$, there exists $x \in \Delta^I$ s.t. $x' = r(x)$. Since $LS(x, I) = LS(x', I')$, we have $x \in \cap_i A_i^I$, so $x \in \cup_j B_j^I$. Hence $x' \in (\bigcup B_j)^I'$.

- $\alpha = A \subseteq \exists R.B$ Assume $x' \in A^I$, there exists $x$ such that $x' = r(x)$ and $x \in A^I$. Since $I \models \alpha$, there exists $y \in \Delta^I$ s.t. $(x, y) \in R^I$ and $y \in B^I$. So $(x', r(y)) \in R^I$ and $r(y) \in B^I$. Hence $x' \in (\exists R.B)^I$.

- $\alpha = \exists R.A \subseteq B$ Assume $x' \in (\exists R.A)^I$, there exists $y'$ such that $(x', y') \in R'^I$ and $y' \in A^I$. So there exists $(x, y) \in R^I$ s.t. $x' = r(x)$ and $y' = r(y)$. Since $r(y) \in A^I$, $y \in A^I$. Because $I \models \alpha$, $x \in B^I$ and thus $x' \in B^I$.

- $\alpha = A \subseteq \forall R.B$ Assume $x', y' \in \Delta^I$ s.t. $(x', y') \in R'^I$ and $x' \in A^I$, there exists $x, y \in \Delta^I$ s.t. $x' = r(x)$, $y' = r(y)$ and $(x, y) \in R^I$. Since $LS(x, I) = LS(x', I')$, we have $x \in A^I$. Because $I \models \alpha$, $y \in B^I$, hence $y' \in B^I$.  

105
• $\alpha = N_a \equiv \{ a \}$ By $I \models \alpha$ we have $N^I_a = \{ a^I \}$. According to the definition of the function $\text{condense}(\cdot)$ we have $N^I_a = \{ r(a^I) \} = \{ a^I \}$.

• $\alpha = R \subseteq S$ If $(x', y') \in R^I$, there exists $x, y \in \Delta^I$ s.t. $x' = r(x)$, $y' = r(y)$ and $(x, y) \in R^I$. Since $I| = \alpha$, $(x, y) \in S^I$ and so $(x', y') \in S^I$.

So $I' \models O \cup \{ L = \{ x_L \} \}$ holds. Assume $x \in (E \cap \neg F)^I$, since $LS(x, I) = LS(r(x), I')$ we know $r(x) \in (E \cap \neg F)^I$, so $(E \cap \neg F)^I \neq \emptyset$. □

Lemma 5.5.10. Given some $O^+_w$ and $F \in N^{\top}_C \cap \text{Sig}(O)$, if in $I_{[O^+_w, <^I]}$ every $N_a \in \mathbb{N}_P$ is a condensing label, then for each $E \in N^{\top}_C \cap \text{Sig}(O)$ s.t. $O^+_w \not\models E \subseteq F$, there is a model of $O$ satisfying $E \cap \neg F$.

Proof. Let $\{ L_i \equiv \{ x_{L_i} \} \}_{i=1}^n$ be all nominal axioms in $O$. We prove $I_{[O^+_w, <^I]}$, which satisfies $E \cap \neg F$ for each non-subclass $E \in N^{\top}_C \cap \text{Sig}(O)$ of $F$, can be transformed to a model $I_n$ of $O^n = O^+_w \cup \{ L_i \equiv \{ x_{L_i} \} \}_{i=1}^n$ such that $(E \cap \neg F)^I_n \neq \emptyset$ by induction on $n$.

By assumption for $n = 0$, $I_0 = I_{[O^+_w, <^I]}$. We need to show a model $I_k$ of $O^k$ satisfying $E \cap \neg F$ can be transformed to a model $I_{k+1}$ of $O^{k+1}$ satisfying $E \cap \neg F$. This step is proved by applying Lemma 5.3.9 where $I = I_k$, $O = O^k$, $L = L_k$ and $x_L = x_{L_k}$.

Then we have transformed the model $I_{[O^+_w, <^I]}$ to a model $I_n$ of $O^n$ satisfying $E \cap \neg F$ where $||\mathbb{N}_P|| = n$. Since $O^n \supseteq O$, we have $I_n \models O$ and $(E \cap \neg F)^I_n \neq \emptyset$. □

Definition 5.5.1. In a saturation $S_{O^+_w}$, the derivation path of a conclusion $\alpha$ of the form $H \subseteq M$ or $H \subseteq N \cup \exists R.K$ is the sequence of all the inference steps

106
\( IS^1_H, \ldots, IS^m_H \) in the context \( H \), where: (1) \( \alpha \in IS^m_H \), conc, and (2) for any \( n < m \), \( IS^n_H \) occurs before \( IS^{n+1}_H \) in the saturation process.

**Lemma 5.5.2.** Given \( O^+ \), a concept \( A \in N_C \cap \operatorname{Sig}(O) \), and an axiom \( \alpha \in SO^+ \) of the form \( H \sqsubseteq M \cup A \) or \( H \sqsubseteq M \cup A \sqcup \exists R.K \), then there exists a conjunct \( B \) of \( H \) such that \( B \in \operatorname{Pri}[A,O^+] \).

**Proof.** According to line 1 and 5 of Algorithm 5.2, \( A \in \operatorname{Pri}[A,O^+] \). Let \( IS^1_H, \ldots, IS^m_H \) be the derivation path of \( \alpha \). We prove the lemma by induction over \( m \).

If \( m = 1 \), then \( IS^1_H \), rule is \( R^+_A \), and by the side condition of \( R^+_A \), \( A \) is a conjunct of \( H \). So the lemma holds where \( B = A \in \operatorname{Pri}[A,O^+] \). Next we show the lemma holds when \( m = k \), if it holds for all \( m < k \). Since \( \alpha \in SO^+ \), there must exist some step \( IS^p_H \) such that \( A \) is a disjunct of the axiom in the conclusion but not in the premise. In this case, \( IS^p_H \), rule can only be \( R^+_A \), \( R^+_n \) or \( R^-_i \), so we can perform a case analysis as follows.

**Case 1** Similarly to the case \( m = 1 \), we can choose \( B = A \) to prove the lemma.

**Case 2** \( IS^p_H \), rule=\( R^+_n \) In this case \( IS^p_H \), sc has a single axiom \( \alpha \) of the form \( \bigcap A_i \sqsubseteq \bigcup B_j \). By line 7, there exists some \( A_i \) s.t. \( A_i \in \operatorname{Pri}[A,O^+] \). Since \( H \sqsubseteq N_i \sqcup A_i \in IS^p_H \), prem, its derivation path \( IS^1_H, \ldots, IS^{p'}_H \) must satisfy \( p' < p \leq k \). By applying the inductive hypothesis to \( m = p' \) and \( H \sqsubseteq N_i \sqcup A_i \), there exists \( H \)'s conjunct \( B \) s.t. \( B \in \operatorname{Pri}[A,O^+] \). Since \( A_i \in \operatorname{Pri}[A,O^+] \), according to the Algorithm 5.2, we can see that \( \operatorname{Pri}[A,O^+] \subseteq \operatorname{Pri}[A,O^+] \). So \( B \in \operatorname{Pri}[A,O^+] \), and the lemma is proved.
Case 3. Let $\mathcal{S}_n^0$.rule$=\mathcal{R}_3^-$ in this case $\mathcal{S}_n^0$.sc has axioms of the forms $R \sqsubseteq^* S$ and $\exists S. Y \sqsubseteq A$, and one of the premises $\mathcal{S}_n^0$.prem is of the form $H \sqsubseteq M' \sqcup \exists R.(\bigcap_{i=1}^n C'_{K_i})$, which is derived by the process:

$$H \sqsubseteq M_1 \sqcup A' \xrightarrow{R_3^-} H \sqsubseteq M_1 \sqcup \exists R.C'_{K_1} \xrightarrow{R_3^-} H \sqsubseteq M' \sqcup \exists R.(\bigcap_{i=1}^n C'_{K_i})$$

The first inference step has a side condition of the form $A' \sqsubseteq \exists R.C'_{K_1}$ and a premise of the form $H \sqsubseteq M_1 \sqcup A'$. By line 9 to 10, $A'$ is added to Pri[A,O;1] where $W = A$ and $Z = C'_{K_i}$. Let $\mathcal{S}_n^1, \ldots, \mathcal{S}_n^{p'}$ be the derivation path of $H \sqsubseteq M_1 \sqcup A'$. We can see $p' < k$ since $H \sqsubseteq M_1 \sqcup A'$ must be derived before the $k$th step. By the inductive hypothesis, there exists $H$'s conjunct $B$ s.t. $B \in \text{Pri}[A',O;1]$. Since $A' \in \text{Pri}[A,O;1]$, $\text{Pri}[A',O;1] \subseteq \text{Pri}[A,O;1]$. So $B \in \text{Pri}[A,O;1]$, and the lemma is proved. □

**Lemma 5.5.14.** Given $O^+_w$ and a concept $A$, $\text{OPS}_{[A,O;1]}$ returned by Algorithm 5.2 preserves $\text{OPS}_{[A,O;1]} \supseteq \text{PS}_{[A,O;1]}$.

**Proof.** By Lemma 5.5.2, we have shown that there is at least one conjunct $B$ of $H$ in Pri[A,O;1]. Since $H$’s conjuncts are all collected during the derivation of init(H), we discuss the two cases how init(H) is derived, and how $H$’s conjuncts are added in each case:

- If init(H) is introduced at initialization stage, then $B$ is the only conjunct in $H$ belonging to $N^+_C \cap \text{Sig}(O)$ or $\mathbb{NP}$, and is added to $\text{OPS}_{[A,O;1]}$ in line 19 of Algorithm 5.2 where $W = B$ and $U = N^+_C \cap \text{Sig}(O)$.

108
• If init(H) is introduced by R_{init} rule, then there is a premise of the form 
\( H^* \subseteq M \sqcup \exists R.H \), where \( H^* \) is a context different from \( H \). Let \( H = \prod_{i=1}^{n} C_{H_i} \),
the derivation process of \( H^* \subseteq M \sqcup \exists R.H \) is:

\[
\begin{align*}
H^* \subseteq M_1 \sqcup A & \xrightarrow{\begin{array}{c}
R_1^+ \\
A \in \exists R.C_{H_i}^1
\end{array}} H^* \subseteq M_1 \sqcup \exists R.C_{H_i}^1 \\
& \cdots \\
& \xrightarrow{\begin{array}{c}
R_r / R_3^- \\
R_{\infty} S \ Y \subseteq \forall S.Z \exists S.Z \subseteq Y
\end{array}} H^* \subseteq M \sqcup \exists R.\left( \prod_{i=1}^{n} C_{H_i}^1 \right)
\end{align*}
\]

The side condition of the first step is \( A \subseteq \exists R.C_{H_i}^1 \). We first prove \( \exists R.C_{H_i}^1 \in \text{Exists} \), where \( \text{Exists} \) is the set produced in the loop from lines 18 to 25. If \( B \) is \( C_{H_i}^1 \), then \( \exists R.C_{H_i}^1 \) is added to \( \text{Exists} \) in line 25 where \( W = B \). If \( B \) is a conjunct of \( H \) other than \( C_{H_i}^1 \), then \( B \) becomes a conjunct after an application of \( R_\forall \) rule, in such case the side condition is \( R \sqsubseteq^* O.S \) and \( Y \subseteq \forall S.B \), so \( \exists R.C_{H_i}^1 \) is added to \( \text{Exists} \) in line ?? where \( W = B \).

Next we show the lemma holds for all three types of conjuncts \( C \) of \( H \):

1. If \( C \) is added to the conjuncts of \( H \) by \( R_3^+ \) rule, then \( C = C_{H_i}^1 \) and is added to \( \text{OPS}_{[A,O^+]} \) in line 27.

2. If \( C \) is added to the conjuncts of \( H \) by \( R_\forall \) rule, then \( C \) is added to \( \text{OPS}_{[A,O^+]} \) in line 29.

3. If \( C \) is added to the conjuncts of \( H \) by \( R_3^- \) rule, then \( C \) is of the form \( \neg Z \), and \( Z \) is added to \( \text{OPS}_{[A,O^+]} \) in line 31.

Hence the lemma is proved. \( \square \)

**Theorem 5.5.17.** Let \( O^+ \) be strengthening axioms computed from Algorithm 5.3, the ontology \( O^*_w = O_w \cup O^+ \) is decisive.
Proof. Since in the last round of the loop $O^{n+} = O^{n-1+}$, we know that for each $N_a \in \mathbb{NP}$ and $X \in \text{OPS}_{[N_a, O^{n+}_w]} = \text{OPS}_{[N_a, O^{n-1+}_w]}$, chooseStrAxiom($N_a, X$) $\subseteq O^{n+} \subseteq O^{n+}_w$. And because $\text{PS}_{[N_a, O^{n+}_w]} \subseteq \text{OPS}_{[N_a, O^{n+}_w]}$, thus $O^{n+}_w = O^{n+}$ is decisive.

\textbf{Theorem 5.5.18.} Let $O^+$ be strengthening axioms computed from Algorithm 5.4, the ontology $O^+_w = O_w \cup O^+$ is decisive.

Proof. For each $N_a \in \mathbb{NP}$, we write $g_{N_a}$ for the final group that $N_a$ belongs to after executing lines 6 to 7 of Algorithm 5.4. We will prove that $\text{OPS}_{[N_a, O^{n+}_w]} \subseteq g_{N_a}$. ops. From lines 18 to 31 of Algorithm 5.2, we can see that each concept $X$ is added to $\text{OPS}_{[N_a, O^{n+}_w]}$ because of some $W \in \text{Pri}_{[N_a, O^{n+}_w]}$ in line 18, for which we write $W_X$. And according to the loop in lines 2 to 17, $W_X$ is added to $\text{Pri}_{[N_a, O^{n+}_w]}$ through a search path $N_a = W_0 \rightarrow W_1 \rightarrow \ldots W_n = W_X$, where $W_{i-1} \rightarrow W_i$, $1 \leq i \leq n$ represents $W_i$ is added to ToProcess while processing $W_{i-1}$ in the loop where $W = W_{i-1}$. Note that in Algorithm 5.2, the only case that a strengthening axiom $\alpha \in O^+$ is used is when $\alpha$ is of the form $N_b \subseteq X'$ and used in line 7. Let $W^s$ be the set of $W_i$ such that $W_i$ is added into ToProcess from $W = W_{i-1}$ in line 7 using such a strengthening axiom $\alpha = N_b \subseteq X'$. We prove the lemma by induction over the size $m = ||W^s||$ of $W^s$. The inductive hypothesis is:

For each $X \in \text{OPS}_{[N_a, O^{n+}_w]}$ and $N_a \in \mathbb{NP}$, let $W_X \in \text{Pri}_{[N_a, O^{n+}_w]}$ be the concept that causes $X$ to be added into $\text{OPS}_{[N_a, O^{n+}_w]}$, $N_a = W_0 \rightarrow W_1 \rightarrow \ldots W_n = W_X$ be the search path of $W_X$, and $W^s_X$ be the set of $W_i$ such that $W_{i-1} \rightarrow W_i$ uses a strengthening axiom. If $||W^s_X|| < m$ then $X \in g_{N_a, ops}$. 

110
• $m = 0$ In this case $W_X \in \text{Pri}_{[N_a, O_w]}$ and $X \in \text{OPS}_{[N_a, O_w]}$. By line 4 we see $X$ is in the initial group of $N_a$, and is merged into $g_{N_a}.ops$ while executing lines 6 to 7.

• $m = k$ Let $i_m = \min\{i \mid W_i \in W^s\}$. In this case we have $W_{i_m-1} = X'$ and $W_{i_m} = N_b$. It is easy to see that $X$ is also added to $\text{OPS}_{[N_b, O_w]}$ through the same $W_X$ and a search path $N_b = W_{i_m} \rightarrow \ldots W_n = W_X$, whose $\|W'\|$ is $k - 1$. By applying the inductive hypothesis to $X$ and $N_b$, we know $X \in g_{N_b}.ops$. And since $X' \in \text{Pri}_{[N_a, O_w]}$, $X' \in g_{N_a}.pri$. So $g_{N_a}.pri \cap g_{N_b}.ops \neq \emptyset$, and $g_{N_a}$ and $g_{N_b}$ must be the same group according to the merge criteria in line 6 of Algorithm 5.4. So $X \in g_{N_b}.ops = g_{N_a}.ops$.

Hence $\text{PS}_{[N_a, O_w]} \subseteq \text{OPS}_{[N_a, O_w]} \subseteq g_{N_a}.ops$ for each $N_a \in \mathbb{NP}$ in $O^+$. According to the construction of $O^+$ in line 11 of Algorithm 5.4 we conclude $O^+_w = O_w \cup O^+$ is decisive.

## 5.6 Chapter Summary

In this chapter, we demonstrated how to classify $\mathcal{ALCHO}$ ontologies based on the weakening and strengthening approach, using a hybrid of a fast $\mathcal{ALCH}$ reasoner and a slower $\mathcal{ALCHO}$ reasoner. We explained all steps of the weakening and strengthening approach: preprocessing, weakening and strengthening.

In the preprocessing step, we rewrite the original ontology $O$ to put the nominals in separated axioms and then normalize the ontology to contain only six types
of axioms. In the weakening step, we remove the nominal axioms to obtain the weakened \textit{ALCH} ontology.

For strengthening, we first theoretically analyzed the desired properties for the strengthened ontology to be complete w.r.t \( O \), and then developed algorithms to compute the strengthening axioms. The strengthening is soundness relaxed. We then used the AR to verify the possible unsound subsumptions in \( \mathcal{H}_s \setminus \mathcal{H}_w \). \( \mathcal{H}_w \) and \( \mathcal{H}_s \) are classification result of \( O_w \) and \( O_s \), respectively, obtained from MR.

The final classification result is sound and complete. We also proved the soundness and completeness of our procedure on \textit{ALCHO} ontology classification.
Chapter 6

Classification of Multiple Extensions to $ALCH$

In this chapter, we first present the general principle to composite single extensions to multiple extensions in Section 6.1. Next, we proved several composed multiple extensions based on the previous three single extensions in Section 6.2. Then, we show how to do two extension of $ALCHI(D)^\sim$ from $ALCH$ in Section 6.3.

In this thesis, we only consider sequential composition of single extensions for constructors, and leave the study of parallel composition procedures for future work.
6.1 General Principle

In this section, we first present the general principle of multiple extensions in subsection 6.1.1. Then, we introduce the strategies for creating intermediate weakened and strengthened ontologies in different conditions in subsection 6.1.2.

6.1.1 General Principle of the Methodology

In the previous two chapters, we discussed the general approach to single-extend a reasoner $R_b$ with a new constructor $\mathcal{F}$. Then a natural question comes up: if we can single-extend $R_b$ with feature $\mathcal{F}_1$, and we can also single-extend $R_b$ with feature $\mathcal{F}_2$, can we also double-extend $R_b$ with both features $\mathcal{F}_1$ and $\mathcal{F}_2$? The answer is not always “yes”. So, in the following we will differentiate the question.

A natural question would be based on single extensions whether we can extend multiple constructors from a base language. Let us first have a look at a simple scenario. Assume we have two single extensions $L_b\mathcal{F}_1$ and $L_b\mathcal{F}_2$ from a base language $L_b$, and we want to classify an ontology $O$ in language $L_b\mathcal{F}_1\mathcal{F}_2$ which has both features of $\mathcal{F}_1$ and $\mathcal{F}_2$. According to the overall weakening and strengthening approach explained in Chapter 3, we can first do normalization, and then compute a weakened ontology $O_w$ by removing all axioms beyond $L_b$. The next step is to do strengthening. A naive way is to compute the strengthening axioms by combining $L_b-L_b\mathcal{F}_1$ strengthening axioms and $L_b-L_b\mathcal{F}_2$ strengthening axioms. Unfortunately, such a strengthening ontology may not preserve the completeness w.r.t. $O$. This is because the two constructors may interact with each other, and so
completeness cannot be achieved through the above approach which only handles each constructor separately without considering their interaction.

Based on the two single extensions $L_b F_1$ and $L_b F_2$ from a base language $L_b$, there are two possible ways to further extend to a language $L_b F_1 F_2$. 1) $L_b F_1$ can single-extend to $L_b F_1 F_2$, or 2) $L_b F_2$ can single-extend to $L_b F_1 F_2$. So for the success of a multiple extension, there exists at least one path that we can based on $L_b$ to single-extend to $L_b F_1$, and then base on $L_b F_1$ to further single-extend to $L_b F_1 F_2$, the process continues in such way until final extension to language $L_b F_1 F_2 ... F_n$. Assume the base language is $L_b$ and the target language is $L_b F_1 F_2 ... F_n$, one needs to find a chain $L_b - L_b F_1, L_b F_1 ... F_{i-1} - L_b F_1 ... F_{i-1} F_i, ..., L_b F_1 F_2 ... F_{n-1} - L_b F_1 F_2 ... F_{n-1} F_n$ such that in each single extension from $L_b F_1 ... F_{i-1}$ to $L_b F_1 ... F_{i-1} F_i$, the strengthened ontology can preserve completeness, and the weakened ontology can preserve soundness. We will further illustrate this problem in section 6.1.2. Here we give some formal definition and property for the multiple extension.

**Definition 6.1.1.** We say language $L_b$ can be WS-extended to $L_b F$ if we can do a single extension from $L_b$ to $L_b F$ using the weakening and strengthening approach.

**Definition 6.1.2.** We define a relation $<_L$ over a language $L$ as follows: $F_1 <_L F_2$ if we can WS-extend $L$ with $F_1$. $F_1 <_L F_2$ if we can WS-extend $L$ with $F_1$ and then WS-extend $L + F_1$ with $F_2$.

**Property 6.1.3.** Given multiple constructors $F_1, F_2, ...,$, and $F_n$ beyond $L_b$, we can do a multiple extension from $L_b$ to $L_b F_1 F_2 ... F_n$ if $F_1 <_L F_2 \cdots <_L F_n$, where $L_n = L_b \cup F_1 \cup \cdots \cup F_{n-1}$.
6.1.2 Creating Weakened and Strengthened Ontologies in Multiple Extension

There are several intermediate versions of weakened and strengthened ontologies before the final $O_w$ and $O_s$ are created in the multiple extensions. If $\mathcal{F}_1 \prec_{L_1} \mathcal{F}_2 \ldots \prec_{L_n} \mathcal{F}_n$ holds, then for the given $\mathcal{L}_b \mathcal{F}_1 \mathcal{F}_2 \ldots \mathcal{F}_n$ ontology, in the WS-approach, we will weaken and strengthen the constructors in the order of $\mathcal{F}_n$, $\ldots$, $\mathcal{F}_2$, $\mathcal{F}_1$, their resulting weakened and strengthened ontologies are named $O_{w,n}$, $O_{s,n}$, $O_{w,2}$, $O_{s,2}$, $O_{w,1}$, $O_{s,1}$, accordingly.

In the single extension of WS-approach, we have proved that $O_w$ is sound, and we require $O_s$ is complete in chapter 3. In the multiple extensions, we require the final $O_w$ is sound, and the final $O_s$ is complete so that WS-approach can still use similar approach as in single extensions to ensure soundness and completeness of the final classification result.

In the chain of doing weakening and strengthening on the constructors, a naive way is to generate $O_{w,j-1}$ based on $O_{w,j}$, generate $O_{s,j-1}$ based on $O_{s,j}$. However, if $\mathcal{F}_n$ is soundness-preserved strengthening, $O_{s,j}$ is both sound and complete. Then we can generate $O_{w,j-1}$ from $O_{s,j}$, not from $O_{w,j}$. And $O_{s,j-1}$ is created from $O_{w,j-1}$. If all the constructors from $\mathcal{F}_n$ to $\mathcal{F}_1$ are soundness-preserved strengthening, then we can generate each $O_{w,j-1}$ from $O_{s,j}$, and $O_{s,j-1}$ from $O_{w,j-1}$, $2 \leq j \leq n$, and each $O_{s,j-1}$ is sound and complete. In the end, we only need to use $\text{MR}$ to classify $O_{s,1}$ (i.e., $O_s$) which will preserve soundness and completeness w.r.t. $O$. This is similar to the soundness-preserved strengthening process in single extensions.
If $\mathcal{F}_i$ is the first soundness-relaxed constructor, i.e., $\mathcal{F}_{i+1}, \ldots, \mathcal{F}_{n-1}, \mathcal{F}_n$ are soundness-preserved, then for the constructors prior and including $\mathcal{F}_i$, each of their weakened ontology still can be generated using the strengthened ontology of its precedent constructor, i.e., for $i + 1 \leq j \leq n$, we generate $O_{w,j-1}$ from $O_{s,j}$, and $O_{s,j-1}$ from $O_{w,j-1}$. However, for the constructors after $\mathcal{F}_i$, the situation is different since $O_{s,j}$ may be unsound, and thus all $O_{s,j}$ after $O_{s,i}$ may be unsound, also. Hence the weakened ontologies after $O_{w,j}$ can only be generated using the weakened ontology of its precedent constructor, which is still sound. So for $2 \leq j \leq i$, we generate $O_{w,j-1}$ based on $O_{w,j}$, and generate $O_{s,j-1}$ based on $O_{s,j}$ from which we first need to remove the axioms containing the $j-1$ constructor. In the end, we will use MR to classify both $O_{w,1}(O_w)$ and $O_{s,1}(O_s)$, and use AR to verify pairs in $H_s \setminus H_w$. This is similar to the soundness-relaxed strengthening process in the single extension.

### 6.2 Several Composed Multiple Extensions

We have introduced three single extensions from $\mathcal{ALCH}$ to $\mathcal{ALCH}(D)^-$, $\mathcal{ALCHI}$ and $\mathcal{ALCHO}$ using the WS-approach in chapter 4 and 5. We also have presented the sufficient condition for multiple extension in property 6.1.3. In this section, we will demonstrate how to do multiple extensions with constructors nominals $O$, inverse roles $I$, and limited parameter datatypes $(D)^-$ based on the previous three single extensions.

From the base language $\mathcal{ALCH}$ extending with $(D)^-$, $I$ and $O$, there are alto-
gether \( P_1 + P_2 + P_3 = 15 \) possible paths of multiple extensions including single extensions, two extensions and three extensions. However, based on the condition in property \[0.6.1.3] we have proved nine paths of multiple extensions in this thesis, which are 1) \( \text{ALCH}(D)^- \), 2) \( \text{ALCHI} \), 3) \( \text{ALCHO} \), 4) \( \text{ALCH}(D)^- \text{- ALCHI}(D)^- \), 5) \( \text{ALCHI} \text{- ALCHI}(D)^- \), 6) \( \text{ALCHO} \text{- ALCHO}(D)^- \), 7) \( \text{ALCHO} \text{- ALCHOI} \), 8) \( \text{ALCHO} \text{- ALCHO}(D)^- \text{- ALCHOI}(D)^- \), 9) \( \text{ALCHO} \text{- ALCHOI} \text{- ALCHOI}(D)^- \). see Table \[0.6.1\] The possibility of the other six paths will leave to future work, which are the paths where adding nominal \( O \) is based on \( \text{ALCH}(D)^- \), \( \text{ALCHI} \), or \( \text{ALCHI}(D)^- \) rather than \( \text{ALCH} \). In such cases, it is still not clear whether we can prove the strengthened ontology can ensure completeness because the proof of completeness of \( \text{ALCHO} \) is based on the canonical model, which is built upon inference rules of \( \text{ALCH} \) ontologies and the model for \( \text{ALCH} \) cannot be easily applied to \( \text{ALCH}(D)^- \), \( \text{ALCHI} \) and \( \text{ALCHI}(D)^- \).

Table 6.1: The Multiple Extensions to \( (D)^- \), \( I \) and \( O \) Based on \( \text{ALCH} \)

<table>
<thead>
<tr>
<th>Class</th>
<th>( \text{ALCH} )</th>
<th>( (D)^- )</th>
<th>( I )</th>
<th>( O )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (D)^- )</td>
<td>( \text{ALCH} ) &amp; ( (D)^- ) &lt; ( \text{ALCH} )</td>
<td>( I ) &lt; ( \text{ALCH} )</td>
<td>( O ) &lt; ( \text{ALCH} )</td>
<td></td>
</tr>
<tr>
<td>( I )</td>
<td>( I ) &lt; ( \text{ALCH}(D)^- ) &amp; ( I ) &lt; ( \text{ALCHO}(D)^- )</td>
<td>( (D)^- ) &lt; ( \text{ALCH} )</td>
<td>( (D)^- ) &lt; ( \text{ALCHO} )</td>
<td></td>
</tr>
<tr>
<td>( O )</td>
<td>( O ) &lt; ( \text{ALCH}(D)^- )</td>
<td>( O ) &lt; ( \text{ALCH} )</td>
<td>( O ) &lt; ( \text{ALCH} )</td>
<td></td>
</tr>
</tbody>
</table>

In the following, we select four most essential proofs in the nine paths, other proofs are very similar and are neglected.
Lemma 6.2.1. $I ≺_{ALCH(D)}$

Proof. Note that in the proof of Lemma 4.1.4, Theorem 4.1.5, and Theorem 4.1.8 if $O$ is an $ALCHI(D)$-ontology, the extra type of axioms $R' = R^-$ are also satisfied in the model transformations in the proofs because (1) these axioms are kept unchanged in $\varphi(O)$ and $O_s$ which is in $ALCHI$; (2) the model transformations does not change the interpretation of roles. So $O_s$ is sound and complete w.r.t. to $O$. Hence, the WS-approach for $(D)^-$ can be applied to extend $ALCHI$-reasoners while preserving soundness and completeness. So $I ≺_{ALCH(D)}^-$ holds.

Lemma 6.2.2. $(D)^- ≺_{ALCH} I$

Proof. The proof is an extension of Theorem 4.2.3 for an $ALCHI(D)$-input $O$. The proof for the “if” part still applies. For the “only-if” part, suppose $I'$ is a model of $O_s$. We apply the same transformation to the $I'$ shown in Theorem 4.2.3. In Theorem 4.2.3, we showed that the interpretation $I$ satisfies all $ALCH$ axioms in $O$. Since in the transformation, the interpretation of all features and data ranges are kept unchanged, $I$ also satisfies all data type related axioms in $O$. So $I$ is a model of $O$, and the conclusion of Theorem 4.2.3 also holds for the $ALCHI(D)$-ontology $O$.

Hence, the WS-approach for $I$ can be applied to extend $ALCH(D)^-$-reasoners while preserving soundness and completeness. So $(D)^- ≺_{ALCH} I$ holds.

Lemma 6.2.3. $O ≺_{ALCH} I$

Proof. The proof is an extension of Theorem 4.2.3. The proof for the “if” part still applies. For the “only-if” part, suppose $I'$ is a model of $O_s$. We apply the
same transformation to the $I'$ shown in Theorem 4.2.3. In Theorem 4.2.3, we showed that the interpretation $I$ satisfies all $\mathcal{ALCH}$ axioms in $O$. Since in the transformation, the interpretation of all concepts and individuals are preserved, $I$ also satisfies axioms of the form $N_a \equiv \{a\}$ in $O$. So $I$ is a model of $O$, and the conclusion of Theorem 4.2.3 also holds for the $\mathcal{ALCHOI}$ ontology $O$.

Lemma 6.2.4. $I \preceq_{\mathcal{ALCHO}(D)^-}$

Proof. Note that in the proof of Lemma 4.1.4, Theorem 4.1.5, and Theorem 4.1.8 if $O$ is an $\mathcal{ALCHOI}(D)^-$ ontology, the extra type of axioms $R' = R^-$ and $N_a = \{a\}$ are also satisfied in the model transformation because (1) these axioms are kept unchanged in $O_s$; and (2) the model transformation does not change the interpretation of roles, atomic concepts and individuals. Hence, the WS-approach for $(D)^-$ can be applied to extend $\mathcal{ALCHOI}$-reasoners while preserving soundness and completeness. So $I \preceq_{\mathcal{ALCHO}(D)^-}$ holds.

6.3 $\mathcal{ALCHI}(D)^-$ Ontology Classification

In this section we will illustrate how to extend an $\mathcal{ALCH}$-reasoner to classify an $\mathcal{ALCHI}(D)^-$ ontology based on single extensions $\mathcal{ALCH}(D)^-$ and $\mathcal{ALCHI}$ in Chapter 4, the principle of multiple extension in Section 6.1 and the Lemmas 6.2.1 and 6.2.2 in Section 6.2.

There are two possible paths to extend an $\mathcal{ALCH}$ reasoner to $\mathcal{ALCHI}(D)^-$: 1) $\mathcal{ALCH}\cdot\mathcal{ALCHI}\cdot\mathcal{ALCHI}(D)^-$ and 2) $\mathcal{ALCH}\cdot\mathcal{ALCH}(D)^-\cdot\mathcal{ALCHI}(D)^-$. In Chapter 4, we have proved the single extensions of $\mathcal{ALCH}(D)^-$ and $\mathcal{ALCHI}$ are
both soundness-preserved strengthening. In Lemmas 6.2.1 and 6.2.2, we further proved that $\text{ALCHI}^{-}\text{ALCHI}(\mathcal{D})^{-}$ and $\text{ALCH}(\mathcal{D})^{-}\text{ALCHI}(\mathcal{D})^{-}$ strengthening also preserves soundness.

We take path 1) as an example to demonstrate how to do the two-extension in Algorithm 6.1. We first apply preprocessing, weakening and strengthening for datatypes explained in Section 4.1 to create the strengthened ontology $O_{\mathcal{D}}^{-}$ for $O$, as shown in Line 1-2 of Algorithm 6.1. Note the input ontology here is $\text{ALCHI}(\mathcal{D})^{-}$, while in Section 4.1 the input ontology is $\text{ALCH}(\mathcal{D})^{-}$. But this does not affect the above procedures as we have proved in Lemmas 6.2.1. Inverse role axioms and anonymous roles are kept unchanged in the process. Then we follow the processing, weakening and strengthening procedures for inverse role explained in Section 4.2 to compute the strengthened ontology $O_{I\mathcal{D}}^{-}$ for $O_{\mathcal{D}}^{-}$, as shown in Line 3-4 in the algorithm. Finally we classify $O_{I\mathcal{D}}^{-}$ with MR to get the classification result of $O$, shown in Line 5 in the algorithm.

Algorithm 6.1: Classification of $\text{ALCHI}(\mathcal{D})^{-}$ Ontologies

| Input: $\text{ALCHI}(\mathcal{D})^{-}$ ontology $O$ |
| Output: Classification result of $O$ |
| 1 Do datatype preprocessing on $O$; |
| 2 Compute strengthened ontology $O_{\mathcal{D}}^{-}$ for the preprocessed ontology based on the weakening and strengthening for datatypes; |
| 3 Do inverse role preprocessing on $O_{\mathcal{D}}^{-}$; |
| 4 Compute strengthened ontology $O_{I\mathcal{D}}^{-}$ for the preprocessed ontology based on the weakening and strengthening for inverse roles; |
| 5 return MR.classify($O_{I\mathcal{D}}^{-}$); |
6.4 Chapter Summary

In this chapter, we first proposed the sufficient condition for multiple extensions in property 6.1.3, and then presented the corresponding methods on how to create the weakened and strengthened ontologies based on the different conditions of the extended constructors which belong to soundness-preserved or soundness-relaxed strengthening. Then, according to property 6.1.3, we proved nine paths of multiple extensions from the base language $\text{ALCH}$ extending with $(D)^-$, $I$ and $O$. Finally, we demonstrated how to do two extension of $\text{ALCHI}(D)^-$ from $\text{ALCH}$ based on the proposed multiple extension method in this chapter.
Chapter 7

System Design and Implementation

Based on the general methodology of weakening and strengthening introduced in Chapter 3 and the multiple extensions introduced in Chapter 6, we have implemented a prototype $\text{ALCHOI}(D)^-$ reasoner, WSClassifier, in Java using the OWL API.\footnote{http://owlapi.sourceforge.net} The classifier uses ConDOR r.12 as the main $\text{ALCH}$ reasoner and the $\text{SROIQ}(D)$ hyper-tableau based HermiT 1.3.8\footnote{http://condor-reasoner.googlecode.com} as the assistant reasoner. In this chapter, we first explain our system design in Section 7.1 and then introduce some implementation details in Section 7.2.
7.1 System Design

The system design needs to consider the approach and principles presented in the previous chapters and also the system performance and optimization in the implementation.

In this section, we introduce our overarching architecture in Section 7.1.1 and indexing in Section 7.1.2. Then, we explain the hierarchy in Section 7.1.3 and the major optimization in Section 7.1.4.

7.1.1 Overarching Architecture

The overall architecture of WSClassifier is shown in Fig. 7.1. The input is an OWL 2 ontology in any syntax supported by the OWL API. The output is the class hierarchy $H_{ws}$ that can be accessed through the OWL API reasoning interfaces. We explain all the components in the following, among which the ones in white boxes are implemented by us.

The arrows in Fig. 7.1 represent the data flow of the overall reasoning procedure. The numbers on the arrow denote the execution order, and the symbols represent the data.

- The Preprocessing and Weakening component rewrites axioms containing constructors that are not supported by the main reasoner, implementing the preprocessing and weakening step of datatypes in Section 4.1.1, inverse roles in Section 4.2.1 and nominals in Section 5.2.

\(^4\)Since our algorithm is designed for DL $\mathcal{ALChO}I$, the unsupported anonymous concepts are replaced with artificial atomic concepts and the unsupported axioms are ignored.
Figure 7.1: Key components of WSClassifier

- The Datatype Strengthening component computes and adds strengthening axioms for the datatypes, implementing the strengthening step of datatype in Section 4.1.2.

- The Indexer component normalizes the input ontology and builds an internal representation of it, which is suitable for finding axioms and concept expressions and speeds up the search for strengthening axioms. Details will be explained in Section 7.1.2.

- The Inverse Role Strengthening component computes and adds strengthening axioms for the inverse roles, implementing the strengthening step of inverse role in Section 4.2.2.

- The Nominal Strengthening component has two separate tasks:
  1. calculating the labels of nominals based on Algorithm 5.2 and adds testing axioms to the ontology $O_{w,dio}$ as explained in Section 5.3.2 and Step 6 of Section 7.2.2. The result ontology is denoted by $O'_{w,dio}$. The
tasks corresponds to the arrow labeled “8” in Fig. [7.1]

2. computing the strengthening axioms for the nominals as described in Algorithm [5.4] in Section [5.3.2] and adding them to $O'_w$ and obtain $O_s$, corresponding to the arrow labeled “10” in Fig. [7.1]

- The main and assistant reasoners perform the main reasoning tasks. They can be customized by the settings in the configuration instance of each WS-Reasoner object.

- The comparer calculates the difference between two concept hierarchies produced by the first and second round of classifications, i.e. $H_w$ and $H_s$ respectively.

7.1.2 Index

The target of indexing is to build an internal representation of the ontology $O$ for efficient look-up of the conditions of each foreach statement in Algorithm [5.2].

For this purpose, we construct Table [7.1] and store them in OntologyIndexer object in our program.

Indexing is performed at the start of classification by recursively traversing through the structure of axioms. After indexing, all necessary normalized axioms that we need to access are stored. posMap and unionMap store normalized axioms of the form $\bigcap A \subseteq \bigcup B$. posUnivCptMap and posUnivRoleMap store axioms of the form $A \subseteq \forall R.B$. posExistCptMap and posExistRoleMap store axioms of the form $A \subseteq \exists R.B$. negExistCptMap and negExistRoleMap store axioms of the
Table 7.1: The Index

<table>
<thead>
<tr>
<th>Table Entry</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>posMap</strong> = {⟨D, C⟩</td>
<td>[C] ⊑ [D] ∈ O}</td>
</tr>
<tr>
<td><strong>unionMap</strong> = {⟨C, ∪C, ⊔C⟩</td>
<td>[∪C] ⊑ [⊔C] ∈ O}</td>
</tr>
<tr>
<td><strong>negExistCptMap</strong> = {⟨C, ∃R.C⟩</td>
<td>∃R.[C] ⊑ [∃R.C] ∈ O}</td>
</tr>
<tr>
<td><strong>negExistRoleMap</strong> = {⟨R, ∃R.C⟩</td>
<td>∃R.[C] ⊑ [∃R.C] ∈ O}</td>
</tr>
</tbody>
</table>

Form \(∃R.A ⊑ B\). Besides these tables, we also build and store the role subsumptions \(⊂^*_O\) in a separate hierarchy, which we will mention in the next section.

With these indices, the backward search in Algorithm 5.2 can be implemented efficiently by Algorithm 7.1.

### 7.1.3 Hierarchy

Like most DL reasoners, we organize the outcome of classification in a **class hierarchy**, which can be seen as a directed acyclic graph consisting of nodes and directed edges. Each node represents a set of equivalent atomic concepts. A edge from node \(V_1\) to \(V_2\) represents any concept in \(V_1\) is a direct subclass of any concept in \(V_2\). The notation \(V_1 \lessdot V_2\) denotes either \(V_1 = V_2\) or there exists a directed path from \(V_1\) to \(V_2\).
Algorithm 7.1: getOPS using Index

**Input:** Normalized $\mathcal{ALCH}$ ontology $O^+_w$, an atomic concept $X$, a set of nominal placeholders $\mathbb{NP}$, the set of atomic classes $U$ in the original ontology

**Output:** A pair $\langle \text{OPS}_{[X,O^+_w]}, \text{Pri}_{[X,O^+_w]} \rangle$

1. $\text{OPS}_{[X,O^+_w]} \leftarrow \emptyset$; $\text{Pri}_{[X,O^+_w]} \leftarrow \emptyset$; $\text{ToProcess} \leftarrow \{X\}$; $\text{Exists} \leftarrow \emptyset$

2. while $\text{ToProcess} \neq \emptyset$ do

3. take out a label $W$ from $\text{ToProcess}$;

4. if $W \not\in \text{Pri}_{[X,O^+_w]}$ then

5. add $W$ to $\text{Pri}_{[X,O^+_w]}$;

6. if $W = \top$ then

7. stop the procedure and use AR to do the classification work;

8. foreach $\langle W, C \rangle \in \text{posMap}$ do

9. add $C$ into $\text{ToProcess}$;

10. foreach $\langle W, C \rangle \in \text{unionMap}$ do

11. add $C$ into $\text{ToProcess}$;

12. if $W$ is of the form $\bigcap_i C_i$ then

13. put one $C_i$ into $\text{ToProcess}$;

14. if $W$ is of the form $\exists S.Y$ then

15. foreach role $R$ such that $R \sqsubseteq^* S$ do

16. if $\langle R, \exists R.C \rangle \in \text{posExistRoleMap}$ then

17. add $\exists R.C$ into $\text{ToProcess}$;

18. foreach $W \in \text{Pri}_{[X,O^+_w]}$ do

19. if $W \in U$ or $W \in \mathbb{NP}$ then add $W$ to $\text{OPS}_{[X,O^+_w]}$;

20. foreach $\langle W, \forall S.W \rangle \in \text{posUnivCptMap}$ do

21. foreach role $R$ such that $R \sqsubseteq^* S$ do

22. foreach $\langle R, \exists R.Z \rangle \in \text{posExistRoleMap}$ do

23. add $\exists R.Z$ to $\text{Exists}$;

24. foreach $\langle W, \exists R.W \rangle \in \text{posExistCptMap}$ do

25. add $\exists R.W$ to $\text{Exists}$;

26. foreach $\exists R.W \in \text{Exists and } R \sqsubseteq^* S$ do

27. add $W$ to $\text{OPS}_{[X,O^+_w]}$;

28. foreach $\langle S, \forall S.Z \rangle \in \text{posUnivRoleMap}$ do

29. add $Z$ to $\text{OPS}_{[X,O^+_w]}$;

30. foreach $\langle S, \exists S.Z \rangle \in \text{negExistRoleMap}$ do

31. add $Z$ to $\text{OPS}_{[X,O^+_w]}$;

32. return $\langle \text{OPS}_{[X,O^+_w]}, \text{Pri}_{[X,O^+_w]} \rangle$
In WSClassifier, we maintain all intermediate and final classification results in a data structure that stores the class hierarchy. The data structure is implemented in the Hierarchy class. The class stores a set of HNodes. Each HNode has three major members: concepts is a set of equivalent concepts, parents is the set of direct superclasses, and children is the set of direct subclasses.

In the classification procedure shown in Algorithm 5.1 we come across three class hierarchies, $H_w$, $H_s$, and $H_w$. If the strengthening preserves both soundness and completeness, i.e. $H_s = H_o$, we only need to classify $O_s$ and return the results. Otherwise, we need to store both $H_w$ and $H_s$ and use the assistant reasoner to filter out unsound results in $H_s$.

As shown in Algorithm 5.1 the final returned hierarchy $H_w$ is first initialized with $H_w$, and then further expanded with verified subsumptions. While adding subsumptions, the hierarchy may need to be adjusted. In order to make this process more efficient, we do a topological sorting on the nodes and maintain the orders after new subsumptions are added. The order of each HNode stored in its order member. We store all the nodes in an array-based linked list in Hierarchy, where each node’s order is also used as its index in the array. Moreover, in each node we also store the order of the previous and next node, so that the data structure acts as a linked list which allows constant time addition/deletion of nodes.

The most difficult part of implementing the hierarchy is to maintain the topological order while adding new subsumptions. Algorithm 7.2 gives the details of the procedure. The inputs of the algorithm are the hierarchy $S$ and two nodes $V_1$ and $V_2$, which we want to add subsumption $V_1 \sqsubseteq V_2$. If $V_1$’s order is less than $V_2$’s, we
Algorithm 7.2: addSubsumption($V_1, V_2, S$)

**Input:** A hierarchy $S$, two nodes $V_1, V_2$ of $S$

**Output:** A new hierarchy with subsumption added

1. if $V_1.order < V_2.order$ then
   2. if $V_1 \not\leq V_2$ then
      3. add $V_2$ to $V_1.parents$;
      4. foreach $P \in V_1.parents$ do
         5. if $V_2 \leadsto P$ then remove $P$ from $V_1.parents$, $V_1$ from $P.children$;
         6. add $V_1$ to $V_2.children$;
      7. foreach $Q \in V_2.children$ do
         8. if $Q \leadsto V_1$ then remove $Q$ from $V_2.children$, $V_2$ from $Q.parents$;
   9. else
      10. // Find all nodes that need to be adjusted
      11. Put all nodes $P$ such that $V_2 \leadsto P$ and $P.order \leq V_1.order$ in $FNodes$;
      12. Put all nodes $Q$ such that $Q \leadsto V_1$ and $Q.order \geq V_2.order$ in $BNodes$;
      13. Collect order numbers of $BNodes$ and $FNodes$, and put them into $OrdersToAllocate$;
      14. NodesToMerge ← $BNodes \cap FNodes$;
      15. $FNodes ← FNodes \setminus NodesToMerge$;
      16. $BNodes ← BNodes \setminus NodesToMerge$;
      17. Assign to $BNodes$ orders in $OrdersToAllocate$ in ascending order;
      18. if NodesToMerge is not empty then
         19. newParents ← $\emptyset$, newChildren ← $\emptyset$;
         20. foreach $V \in NodesToMerge$ do
            21. foreach $P \in V.parents$ do
               22. if $P \not\in NodesToMerge$ then
                  23. add $P$ to newParents and remove $V$ from $P.children$;
               24. foreach $C \in V.children$ do
                  25. if $C \not\in NodesToMerge$ then
                     26. add $C$ to newChildren and remove $V$ from $C.parents$;
            27. Remove every node $V$ from newParents where there exists $V' \in newParents$ such that $V' \neq V$ and $V' \leadsto V$;
            28. Remove every node $V$ from newChildren where there exists $V' \in newChildren$ such that $V' \neq V$ and $V \leadsto V'$;
            29. Create a merged node $V^*$ which contains all concepts from nodes in NodesToMerge;
               31. foreach $P \in newParents$ do add $V^*$ to $P.children$;
               32. foreach $C \in newChildren$ do add $V^*$ to $C.parents$;
               33. assign $V^*$ the next order in $OrdersToAllocate$;
         34. Assign to $FNodes$ orders in $OrdersToAllocate$ in ascending order;
simply update $V_1$.parents and $V_2$.children and do transitive reduction, as shown in lines 1 to 8. If $V_1$’s order is more than $V_2$’s, we need to reassign the orders to some of the nodes in the hierarchy in order to ensure that for each subsumption the subclass node’s order is less than superclass node’s order. The procedure is built on top of a so-called dynamic topological sorting technique developed by Pearce et al. [68]. Basically, we find two sets of nodes BNodes and FNodes whose orders that need to be reassigned. Then we collect all order numbers in these two sets, sort them in ascending order, and reassign them to these nodes according to the subsumption relationships. If the added subsumption creates a cycle in the hierarchy, some of the nodes need to be merged to one node.

**Example 7.1.1.** Assume we have a hierarchy $H = \{V, R\}$:

$$V = \{\bot, \top, W_1, W_2, W_3, W_4, W_5\}$$

$$R = \{\langle \bot, W_1 \rangle, \langle \bot, W_4 \rangle, \langle W_1, W_2 \rangle, \langle W_1, W_3 \rangle, \langle W_2, W_5 \rangle, \langle W_4, W_5 \rangle, \langle W_3, \top \rangle, \langle W_5, \top \rangle\}$$

$\bot$.order = 0

$W_1$.order = 1

$W_2$.order = 2

$W_3$.order = 3

$W_4$.order = 4

$W_5$.order = 5

$\top$.order = 6
Figure 7.2: Add a New Subsumption in Hierarchy

See Hierarchy H in the left of Figure 7.2. The number beside the nodes is the order of the node.

Now we want to add a new subsumption $W_5 \sqsubseteq W_1$. Since $W_1 \sqsubseteq W_5$, we will need to merge some nodes.

By the algorithm, we first find the set $\text{BNodes} = \{W_1, W_2, W_4, W_5\}$ and $\text{FNodes} = \{W_1, W_2, W_3, W_5\}$. We then collect all orders in $\text{BNodes} \cup \text{FNodes} : \{1, 2, 3, 4, 5\}$.

Then we compute the nodes to be merged: $\text{BNodes} \cap \text{FNodes} = \{W_1, W_2, W_3\}$ and remove them from $\text{BNodes}$ and $\text{FNodes}$. Now $\text{BNodes} = \{W_4\}$ and
FNodes = \{W_3\}. Then we start to reassign orders to the nodes: first we take out \(W_4\) from BNodes and assign it the least available order 1. Next we create a merged node \(W_6\) which contains all concepts in \(W_1, W_2,\) and \(W_5\), and assign it with the second least available order 2. Next we take out \(W_3\) from FNodes and assign it the next available order 3. We also need to adjust the parent and children fields for nodes \(W_3, W_4,\) and \(W_6\). Finally we get a new hierarchy: \(H' = \{V', R'\}\). The new orders are displayed in the following. See Hierarchy H’ in the right side of Figure 7.2.

\[
\begin{align*}
V' &= \{W_3, W_4, W_6\} \\
R' &= \{\langle \bot, W_4\rangle, \langle W_4, W_6\rangle, \langle W_6, W_3\rangle, \{W_3, \top}\} \\
\bot \text{. order} &= 0 \\
W_4 \text{. order} &= 1 \\
W_6 \text{. order} &= 2 \\
W_3 \text{. order} &= 3 \\
\top \text{. order} &= 6
\end{align*}
\]

### 7.1.4 Optimization

In the implementation, we use an optimization technique employed in the KP [27] algorithm to reduce the number of pairs to be checked. The approach maintains a set of unchecked pairs \(P\). When one pair is checked, the algorithm propagates
the result for other unchecked pairs: (1) if \( A \sqsubseteq B \) is verified, then for all \( B \sqsubseteq X \), \( A \sqsubseteq X \) is verified; (2) if \( A \sqsubseteq B \) is verified to be false, then for all \( X \sqsubseteq B \), \( A \sqsubseteq X \) is false; (3) if \( A \sqsubseteq B \) is verified to be false, and for \( X, Y \) such that \( A \sqsubseteq X \) and \( Y \sqsubseteq B \), then \( X \not\sqsubseteq Y \) is false.

Another optimization employed is to add known subsumptions to the weakened ontology. When we do \( \text{ALCH-ALCHO} \) implementation, we use the following heuristics: if two nominal concepts have a subsumption relationship, they are equivalent to each other.

### 7.2 Implementation

In this section, we first demonstrate the major classes and their functionality in Section [7.2.1](#). Then we illustrate the implementation of the classification procedure in Section [7.2.2](#).

#### 7.2.1 Java Classes

The overarching class diagram is shown in Figure [7.3](#), which includes the major packages and classes in the project and demonstrates their relationships. Rectangles and “tabbed folders” in the diagram represent Java classes and packages, respectively. There are three different kinds of relationships between classes/packages, distinguished by the lines connecting them: (1) solid lines with no arrows denote the association relationship, e.g. an object of one class is a member of the other class; (2) dashed lines \( \rightarrow \) with an arrow denote the dependency relationship,
Figure 7.3: Overarching Class Diagram
e.g. an object of one class is a parameter of a function of the other class; (3) solid lines —▷ with a triangular arrow head denote the generalization relationship, i.e., subclass and superclass relationship. Please refer to Bernd Brügge et al. [11] for the formal definition of these relationships. The WSReasoner class is the main access point of the classifier and implements all interfaces of the OWLReasoner class in OWL API. The classification procedure can be invoked by calling the precomputeInferences method of WSReasoner class, which overrides standard OWL API interface.

Tables 7.2 and 7.3 describe the functionality of the major classes, which are grouped according to the components in Figure 7.1. In Section 7.2.2, we will explain how these classes are used for classification.

### 7.2.2 Implementation of Classification Procedure

As explained in Section 6.2, $\text{ALCH-ALCHO-ALCHOI-ALCHOI(D)}^-$ is one of the possible paths of multiple-extension from $\text{ALCH}$ to $\text{ALCHOI(D)}^-$. We base on this path to do weakening and strengthening, which is the reverse order of the multiple extension, i.e., $\text{ALCHOI(D)}^-\text{-ALCHOI-ALCHO-ALCH}$.

In this subsection, we introduce the detail weakening and strengthening procedure step by step, which illustrates how the final $O_w$ and $O_s$ are computed through the intermediate ones which are created in the weakening and strengthening process for $(D)^-$, $I$ and $O$, respectively. The procedure is basically according to the general methodology of weakening and strengthening in chapter 5 as well as the multiple extensions in chapter 6. We have some minor changes in the
Table 7.2: Major Classes in Each Component

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component: Preprocessor</strong></td>
<td></td>
</tr>
<tr>
<td>OntologyPreprocessor.java</td>
<td>Preprocess axioms that use constructors beyond $\mathcal{ALCH}$</td>
</tr>
<tr>
<td>RealHandler.java</td>
<td>Preprocess and compute strengthening axioms for the $owl:real$ datatype (Also used in the Datatype Strengthening component)</td>
</tr>
<tr>
<td>NumberUtil.java</td>
<td>Utility class for handling numbers (Also used in the Datatype Strengthening component)</td>
</tr>
<tr>
<td>BigRational.java</td>
<td>Manage rational numbers (Also used in the Datatype Strengthening component)</td>
</tr>
<tr>
<td><strong>Component: Datatype Strengthening</strong></td>
<td></td>
</tr>
<tr>
<td>DataTypeProcessor.java</td>
<td>Compute encodings and strengthening axioms for datatypes</td>
</tr>
<tr>
<td>DataTypeHandler.java</td>
<td>Abstract class implementing general strengthening functions for datatypes</td>
</tr>
<tr>
<td>StringHandler.java</td>
<td>Compute encodings and strengthening axioms for $xsd:string$</td>
</tr>
<tr>
<td>BooleanHandler.java</td>
<td>Compute encodings and strengthening axioms for $xsd:boolean$</td>
</tr>
<tr>
<td><strong>Component: Indexer</strong></td>
<td></td>
</tr>
<tr>
<td>OntolgyIndexer.java</td>
<td>Manage index of axioms in an ontology</td>
</tr>
<tr>
<td><strong>Component: InverseRole Strengthening</strong></td>
<td></td>
</tr>
<tr>
<td>InverseRoleProcessor.java</td>
<td>Compute strengthening axioms for inverse roles</td>
</tr>
<tr>
<td><strong>Component: Main Reasoner</strong></td>
<td></td>
</tr>
<tr>
<td>ConDORReasoner.java</td>
<td>Wrapper for ConDOR, implementing all OWL API interfaces</td>
</tr>
<tr>
<td>ConDORConfiguration.java</td>
<td>Configuration information for creating ConDORReasoner instances</td>
</tr>
<tr>
<td>ConDORReasonerFactory.java</td>
<td>Create ConDORReasoner instances</td>
</tr>
</tbody>
</table>
Table 7.3: Major Classes in Each Component

<table>
<thead>
<tr>
<th>File name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Component: Nominal Strengthening</strong></td>
<td></td>
</tr>
<tr>
<td>LabelCalculator.java</td>
<td>Compute nominal labels and generate strengthening axioms, implementing Algorithms 5.2 and 5.4</td>
</tr>
<tr>
<td><strong>Component: Comparer</strong></td>
<td></td>
</tr>
<tr>
<td>Hierarchy.java</td>
<td>Hierarchy data structure</td>
</tr>
<tr>
<td>ConceptHierarchy.java</td>
<td>Manage concept hierarchies</td>
</tr>
<tr>
<td>HierarchyComparer.java</td>
<td>Compare two concept hierarchies</td>
</tr>
<tr>
<td>HNode.java</td>
<td>Nodes in concept hierarchies</td>
</tr>
<tr>
<td><strong>WSClassifier</strong></td>
<td></td>
</tr>
<tr>
<td>WSReasoner.java</td>
<td>Implement entire multiple extension procedure</td>
</tr>
<tr>
<td>WSReasonerFactory.java</td>
<td>Create WSReasoner instances</td>
</tr>
<tr>
<td>WSReasonerConfig.java</td>
<td>Configuration information of WSReasoner, e.g. for creating the main and the assistant reasoner</td>
</tr>
<tr>
<td><strong>Main Supporting Classes</strong></td>
<td></td>
</tr>
<tr>
<td>InternalEntityManager.java</td>
<td>Manage fresh concepts and roles introduced during classification</td>
</tr>
<tr>
<td>MultiMap.java</td>
<td>Implement multimap data structure</td>
</tr>
<tr>
<td>OntologyProcessor.java</td>
<td>Abstract class implementing functions for processing ontologies</td>
</tr>
<tr>
<td>OntologyClassifier.java</td>
<td>Abstract class implementing part of OWLReasoner interface methods using ClassHierarchy</td>
</tr>
<tr>
<td><strong>Role Hierarchy</strong></td>
<td></td>
</tr>
<tr>
<td>RoleHierarchy.java</td>
<td>Manage role hierarchies</td>
</tr>
<tr>
<td>PropNode.java</td>
<td>Nodes in role hierarchies</td>
</tr>
<tr>
<td><strong>Utility Classes</strong></td>
<td></td>
</tr>
<tr>
<td>WSReasonerStatistic.java</td>
<td>Record statistics for WSReasoner</td>
</tr>
<tr>
<td>ReasonerLogger.java</td>
<td>Logging for WSReasoner</td>
</tr>
<tr>
<td>RunClassifier.java</td>
<td>Entry point of the program</td>
</tr>
</tbody>
</table>
implementation. e.g., in order to build the index which is based on the preprocessed ontologies, we put the preprocessing of inverse role and nominal together in Step 2 before building index; in order to optimize nominal strengthening, we compute nominal labels in Step 5 before first round classification. We show the steps in the follows.

**Step 1: Weakening and Strengthening from $ALCHOI(D)^-\rightarrow ALCHOI$**

**Input:** Ontology $O$ in $ALCHOI(D)^-$

**Output:** Weakened ontology $O_{w,d}$ and the strengthened ontology $O_{s,d}$, both in $ALCHOI$

In this step, based on the input $ALCHOI(D)^-$ ontology $O$, we illustrate the implementation of the two major phases for the weakening and strengthening from $ALCHOI(D)^-$ to $ALCHOI$ – the preprocessing and weakening phase as well as the strengthening phase for datatype $D$, which is similar to the two phases in the single extension from $ALCH$ to the $ALCH(D)^-$ in Section 4.1.1 and Section 4.1.2.

- Preprocessing and weakening for $(D)^-$: this phase uses a normalization procedure $\text{transform}_d$, where only for $d=\text{owl:real}$ (denoted by $\mathbb{R}$) there is work to do. As explained in Section 4.1.1, there are two tasks for $\text{transform}_\mathbb{R}$: (1) transforming all atomic data ranges using comparison facets into normalized ones using only $>_a$. This task is implemented by $\text{RealHandler}$. (2) parsing string representations of numbers into Java objects of the abstract Java built-in class $\text{Number}$. The task is implemented in $\text{NumberUtil}$ and
invoked by RealHandler. Constants of xsd:integer, owl:decimal, and owl:rational are stored as Java objects of instantiatable subclasses of Number: Integer, Long, BigInteger, BigDecimal, or BigRational, where BigRational is implemented by us and the others are Java build-ins. In order to store different syntactic format of constants with the same value into equivalent Java objects, all constants are represented using the least expressive class, e.g. equivalent constants "6/2"^^owl:rational and "3"^^xsd:integer are both stored using Integer instead of using BigRational for "6/2"^^owl:rational. In NumberUtil, we implement a custom comparator to compare objects of these different Java classes based on the comparison function of each class. The comparator is needed for sorting fArray with Number elements in ascending order in the implementation of Algorithm 4.2 After this preprocessing step, the result ontology is O_{p,d}. From O_{p,d}, we remove all the axioms which contain datatypes and obtain the weakened version O_{w,d}.

- Strengthening for (D)^−: From O_{w,d}, we add datatype strengthening axioms to O_{w,d} and obtain O_{s,d} in this step. The strengthening axioms are produced according to Algorithm 4.1 and implemented through DataTypeProcessor. In the algorithm, the strengthening process is from line 6 to 8. For line 6, DataTypeProcessor implements encodings for data properties, datatype-related axioms, and complex data ranges. It calls handlers for specific datatypes, i.e. RealHandler, BooleanHandler, and StringHandler, to encode atomic data ranges of those datatypes. Line 7 and 8 are also implemented in DataTypeProcessor, where the function getAxioms_d
for generating strengthening axioms is implemented in the specific datatype handlers. For example, the implementation of getAxioms$_R$ (Algorithm 4.2), which generates the strengthening axioms for real datatype, is implemented by RealHandler.

**Step 2: Preprocessing on Inverse Role and Nominal**

**Input:** Ontology $O_{s,d}$ in $\text{ALCHOI}$

**Output:** Ontology $O'_{s,d}$ in $\text{ALCHOI}$

In Section 6.2, we have proved that $O_{s,d}$ is sound and complete. In such case, we only need to use $O_{s,d}$ for the weakening and strengthening of the next constructor $I$ as introduced in Section 6.1.2 $O_{w,d}$ is not needed.

- Preprocessing $I$ and $O$: from $O_{s,d}$, we do preprocessing for inverse role $I$ and nominal $O$, see Section 4.2.1 and Section 5.2 for the tasks. For both of them, here we only perform task 1) in each of their preprocessing phase. Task 2) of their preprocessing phases can be achieved in building index explained in the following. Both the preprocessing of the inverse role and nominal is implemented in OntologyPreprocessor. The result ontology is $O'_{s,d}$

**Step 3: Building Role Hierarchy and Index**

**Input:** Ontology $O'_{s,d}$ in $\text{ALCHOI}$

**Output:** Role hierarchy and index

- Extract role axioms, compute the role hierarchy and store the result in a RoleHierarchy instance.
- Build the index and store it in an OntologyIndex instance

Step 4: Weakening and Strengthening from $\text{ALCHOI}$-$\text{ALCHO}$

**Input:** Ontology $O_{s,di}'$ in $\text{ALCHOI}$

**Output:** Ontology $O_{w,di}$ and $O_{s,di}$ in $\text{ALCHO}$

- **Weakening $I$:** we remove all the inverse role axioms from the $O_{s,di}'$ and obtain $O_{w,di}$.

- **Strengthening $I$:** we add inverse role strengthening axioms to $O_{w,di}$ and obtain $O_{s,di}$. The strengthening axioms are produced according to Algorithm 4.3 in Section 4.2.2, which is implemented in InverseRoleProcessor and uses the index as an input and is invoked by WSReasoner.

Step 5: Weakening from $\text{ALCHO}$-$\text{ALCH}$ and Adding Testing Axioms

**Input:** Ontology $O_{s,di}$ in $\text{ALCHO}$

**Output:** Ontology $O_{w,di,o}$ and $O_{w,di,o}'$ in $\text{ALCH}$

- **Weakening $O$:** use the above $O_{s,di}$, remove all the nominal axioms, and obtain $O_{w,di,o}$.

- **Compute nominal labels and add testing axioms:** compute the possible labels for each nominal concept described in algorithm 5.2. This function is implemented in LabelCalculator. For each label of each nominal, one testing axiom to check whether they have disjoint relationship is added to $O_{w,di,o}$ as explained in Section 5.3.2 and obtain $O_{w,di,o}'$. 

142
Step 6: First Round Classification

Input: ontology $O'_{w, dio}$ in $ALCH$

Output: hierarchy $H_w$ in $ALCH$

- Creating the main reasoner MR: In the project, our main reasoner MR is ConDOR, which is written in C++. Our project is java-based and uses ConDOR.exe file. We set its basic configuration in ConDORConfiguration, and ConDORReasonerFactory will use the configuration and $O'_{w, dio}$ as input to generate ConDORReasoner. ConDORReasoner is a subclass of the abstract class OntologyClassifier, which realizes some interfaces of OWLReasoner in OWL API.

- Classifying $O'_{w, dio}$ with MR and obtain $H_w$.

Step 7: Strengthening from $ALCHO$-$ALCH$

Input: ontology $O_{w, dio}$ in $ALCH$, weakened classification result $H_w$

Output: $O_{s, dio}$ in $ALCH$

- Strengthening O: we add the nominal strengthening axioms into $O_{w, dio}$ and obtain $O_{w, dio}$. The strengthening axioms are produced according to algorithm 5.4 which also uses the heuristic rules based on $H_w$ explained in Section 7.1.4. The algorithm is implemented in LabelCalculator, which uses ConceptHierarchy to access concepts stored in HNode of the concept hierarchy $H_w$. The index and role hierarchy computed precedent are also used.
**Step 8: Second Round Classification**

**Input:** ontology $O_{s,dio}$ in $\mathcal{ALCH}$, the main reasoner MR  
**Output:** Hierarchy $H_s$ in $\mathcal{ALCH}$

- Classify $O_{s,dio}$ with MR and obtain $H_s$

**Step 9: Verification**

**Input:** Hierarchy $H_w, H_s$  
**Output:** Final classification result $H_{ws}$

The verification stage is described in algorithm [5.1] and implemented by WSReasoner. The stage contains the following two parts:

- Comparing concept hierarchy: our classification result is stored in concept hierarchy implemented by ConceptHierarchy, which extends class Hierarchy. The basic element in ConceptHierarchy is HNode, each of which contains a group of equivalent concepts and the information of its parents nodes and children nodes. HierarchyComparer can compare two instances of ConceptHierarchy. This function is used in $H_s \setminus H_w$ and produce the different pairs between them.

- Creating the assistant reasoner AR: we use HermiT.Reasoner to create an instance of HermiT reasoner;

- Verifying subsumption pairs: the AR is used to verify the extra pairs in $H_s$ computed above. The pairs which are verified to be true are added to $H_w$ and produce the final classification result $H_{ws}$
7.3 Chapter Summary

In this chapter, we introduced the system design and implementation of our prototype reasoner WSClassifier for classifying $\mathcal{ALCHOI}(\mathcal{D})^-$ ontologies based on the approach proposed in previous chapters.

We first described the overall architecture of the system, including the software modules and information flow for the classification. Next we introduced indexing and optimizations in the system, which greatly influence the performance of the system. We also explained a basic data structure employed in the system – the hierarchy for storing the classification result of the ontologies.

In the implementation section, we first illustrated important Java classes and their functionality. We then demonstrated the concrete steps of classifying $\mathcal{ALCHOI}(\mathcal{D})^-$ ontologies through a three-extension from $\mathcal{ALCH}$ to $\mathcal{ALCHOI}(\mathcal{D})^-$, following the general principles of multiple extensions as well as the weakening and strengthening approach.
Chapter 8

Experiments

In this chapter, we conduct experiments with $\text{ALCHI(D)}^-$, $\text{ALCHO}$ and $\text{ALCHOI(D)}^-$ ontologies and compare the performance of our WSClassifier with available relevant mainstream reasoners in Sections 8.2, 8.3 and 8.4 respectively. Then in Section 8.5 we report the result of ORE 2013 competition in which we participated.

8.1 Introduction

In this chapter, we did experiments and compared the classification time of WSClassifier with tableau-based reasoners HermiT 1.3.8, Fact++ 1.5.3, and Pellet 2.3.0, as well as another hybrid reasoner MORe 0.1.3, which combines ELK and HermiT. All the experiments were run on a laptop with an Intel Core i7-2670QM 2.20GHz quad core CPU and 16GB RAM running Java 1.6 under Windows 7. We
set the Java heap space to 12GB and the time limit to one day for all reasoners on \textit{ALCHI(D)}\textsuperscript{−} classification, nine days on \textit{ALCHO} classification, and four days on \textit{ALCHOI(D)}\textsuperscript{−} classification. For HermiT, we set its configuration to simple core blocking and individual reuse, which is the optimized configuration for running large and complex ontologies.

In the following, we present the experiment on \textit{ALCHI(D)}\textsuperscript{−} ontologies using only MR in Section 8.2; the experiment on \textit{ALCHO} and \textit{ALCHOI(D)}\textsuperscript{−} ontologies using both MR and AR in Section 8.3 and 8.4 respectively. Finally, we present the DL classification result in the 2013 international OWL Reasoner Performance Competition on which WSClassifier participated.

### 8.2 Experiments with \textit{ALCHI(D)}\textsuperscript{−} Ontologies

We have proposed approaches for \textit{ALCH(D)}\textsuperscript{−} and \textit{ALCHI} ontology classification in Chapter 4. Since the classification on both language only need AR and the two features do not interact with each other as explained in Section 6.2, we use the combination of these two features of \textit{ALCHI(D)}\textsuperscript{−} ontologies in this experiment. We compare the runtime of our WSClassifier with all other available \textit{ALCHI(D)}\textsuperscript{−} reasoners HermiT, Fact++, and Pellet, which all happen to be tableau-based reasoners. Resulting from our search for large and highly cyclic ontologies that are publicly available, we were able to find only one, FMA-constitutionalPartForNS(FMA-C), that uses all the constructors of \textit{ALCHI(D)}\textsuperscript{−}. 

147
We remove seven axioms using $xsd:float$. For Full-Galen\(^1\), of which the language is $\mathcal{ALEHIF}^+$ without datatypes, we introduce some new data type axioms by converting some axioms using roles $\text{hasNumber}$ and $\text{hasMagnitude}$ into axioms with new features $\text{hasNumberDT}$ and $\text{hasMagnitudeDT}$. Some concepts, which should be modeled as data ranges, are also converted to data ranges. For Galen-Heart, we did not introduce datatypes to it, but it contains inverse roles. Wine is a small but highly cyclic ontology. We also include two commonly used ontologies ACGT and OBI, which are not highly cyclic. For ACGT and OBI, we change $xsd:int$, $xsd:positiveInteger$, and $xsd:nonNegativeInteger$ to $xsd:integer$, and change $xsd:float$ to $owl:rational$. For all the ontologies, we reduce their language to $\mathcal{ALCHI}(\mathcal{D})^-$. The ontologies are available from our website:\(^2\)

Table 8.1: Comparison of classification performance of $\mathcal{ALCHI}(\mathcal{D})^-$ ontologies

<table>
<thead>
<tr>
<th></th>
<th>HermiT</th>
<th>Pellet</th>
<th>FaCT++</th>
<th>WSClассifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wine</td>
<td>1.16 sec</td>
<td>0.43 sec</td>
<td>0.005 sec</td>
<td>0.40 sec</td>
</tr>
<tr>
<td>ACGT</td>
<td>9.603 sec</td>
<td>2.955 sec</td>
<td>*</td>
<td>1.945 sec</td>
</tr>
<tr>
<td>OBI</td>
<td>3.166 sec</td>
<td>45.261 sec</td>
<td>*</td>
<td>8.835 sec</td>
</tr>
<tr>
<td>Galen-Heart</td>
<td>123.628 sec</td>
<td>–</td>
<td>–</td>
<td>2.779 sec</td>
</tr>
<tr>
<td>Full-Galen</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>16.774 sec</td>
</tr>
<tr>
<td>FMA-C</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>32.74 sec</td>
</tr>
</tbody>
</table>

Note: “–” means out of time or memory and “*” means not support some data types

Table 8.1 summarizes the experiment results. WSClассifier is significantly faster than the tableau-based reasoners on the three highly cyclic large ontologies: Galen-Heart, Full-Galen, and FMA-C. ACGT is not highly cyclic, but WSClассifier is

\(^1\)http://www.co-ode.org/galen
\(^2\)http://isel.cs.unb.ca/~wsong/ORE2013WSClassifierOntologies.zip
still faster. For the other two ontologies where WSClassifier is not the fastest one, Wine is cyclic but small and OBI is not highly cyclic. The classification times of all reasoners on these two ontologies are significantly shorter comparing with the times on large highly cyclic ontologies. Then WSClassifier took a larger percentage of time on the overhead to transmit the ontology to and from ConDOR. Fact++ does not support owl:rational, so it can not classify ACGT and OBI, but it is significantly faster on Wine.

8.3 Experiments with \textit{ALCHO} Ontologies

We have proposed the approach for \textit{ALCHO} ontology classification in Chapter\textsuperscript{5} In this experiment, we evaluated ontology classification in \textit{ALCHO} by WSClassifier and other reasoners on all large and complex ontologies available to us, on the ORE2012 dataset, and on some proposed variants. The only large and complex ontologies included are FMA-constitutionalPartForNS(FMA-C)\textsuperscript{3} and modified versions of Galen in which some concepts starting with a lower case letter and subsumed by \textit{SymbolicValueType} are modeled as nominals. The ontologies containing “EL” in the name are constructed based on Galen-EL\textsuperscript{4} Galen-EL-n1Y and Galen-EL-n2Y were provided \textsuperscript{50}. Galen-Heart-n1 and Galen-Heart-n2 are subontologies extracted from Full-Galen\textsuperscript{5}. Galen-EL-n1YE and Galen-EL-n2YE have some nominals removed from Galen-EL-n1Y and Galen-EL-n2Y,

\textsuperscript{4}http://code.google.com/p/condor-reasoner/downloads/list
\textsuperscript{5}http://www.co-ode.org/galen/
respectively. And Galen-Union-n is made by adding disjunctions of nominals.

We used two common smaller complex ontologies – Wine and DOLCE. We use the ORE2012 dataset\footnote{http://www.cs.ox.ac.uk/isg/conferences/ORE2012/} where two ontologies without axioms are removed. In all cases, we reduce the language to \textit{ALCHO}. The ontologies are available from our website\footnote{http://isel.cs.unb.ca/~wsong/WSClassifierExperimentOntologies.zip}.

Table 8.2: Comparison of classification performance of \textit{ALCHO} ontologies

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Concepts</th>
<th>Nominals</th>
<th>(Hyper) tableau</th>
<th>Hybrid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>HermiT</td>
<td>Pellet</td>
</tr>
<tr>
<td>Wine</td>
<td>146</td>
<td>206</td>
<td>24.6</td>
<td>285.6</td>
</tr>
<tr>
<td>DOLCE</td>
<td>207</td>
<td>39</td>
<td>6.6</td>
<td>7.0</td>
</tr>
<tr>
<td>Galen-Heart-n1</td>
<td>3366</td>
<td>55</td>
<td>264.0</td>
<td>–</td>
</tr>
<tr>
<td>Galen-Heart-n2</td>
<td>3366</td>
<td>92</td>
<td>768.4</td>
<td>–</td>
</tr>
<tr>
<td>Galen-EL-n1Y</td>
<td>23136</td>
<td>739</td>
<td>701,822.0</td>
<td>–</td>
</tr>
<tr>
<td>Galen-EL-n2Y</td>
<td>23136</td>
<td>1113</td>
<td>407,427.0</td>
<td>–</td>
</tr>
<tr>
<td>Galen-EL-n1YE</td>
<td>23136</td>
<td>598</td>
<td>244,146.0</td>
<td>–</td>
</tr>
<tr>
<td>Galen-EL-n2YE</td>
<td>23136</td>
<td>712</td>
<td>289,637.0</td>
<td>–</td>
</tr>
<tr>
<td>Galen-Union-n</td>
<td>23136</td>
<td>598</td>
<td>469,274.3</td>
<td>–</td>
</tr>
<tr>
<td>FMA-C</td>
<td>41648</td>
<td>85</td>
<td>140,882.0</td>
<td>–</td>
</tr>
</tbody>
</table>

ORE-dataset (OWL DL & EL, 113 ontologies) the following refers to average number

| FMA-lite               | 75,141   | 0        | 137,409.0      | –      | –      | –    | 26.0         |
| remaining 112 ontologies | 4293  | 343      | 0.84           | 0.86   | –      | 0.24 | 2.10         |

\textbf{Note:} The time is measured in seconds. “–” means out of time or memory

\textbf{*:} FaCT++ terminates unexpectedly while classifying some ontologies in the ORE-dataset.

For FMA-C, which is the only real-world large and complex ontologies with nominals we have access to, WSClassifier finished classification in 21.2 seconds, while HermiT used 140,882 seconds. Other reasoners did not finish classification on it in nine days. From the results of Table\textsuperscript{8.2}, we can see, except for the ORE dataset, WSClassifier is significantly faster than the tableau-based reasoners on seven out of ten ontologies. For the other three ontologies – Wine, Galen-EL-YN1 and
Galen-EL-YN2 – WSClassifier, incurring a relatively small cost, detected that strengthening axioms made some concepts unsatisfiable in $O_s$, and so failed over to HermiT.

We see a major speedup for WSClassifier on ORE’s FMA-lite. On the remaining 112 ORE ontologies, our average reasoning time is longer than other reasoners. Among these ontologies, 51 have nominals, mostly coming from ABoxes, and only nine of them have strengthening axioms. Of the nine ontologies, eight did not produce any new subsumptions in $\mathcal{H}_s$ and only one which is the variant of Wine ontology introduced new unsatisfiable concepts and fails over to HermiT. Thus, the WS approach does not incur much additional work, and most of the additional time is taken on overheads: computing normalized and strengthening axioms and transmitting the ontology to and from ConDOR. The transmission, which consumes about 60% of the time, is necessary since ConDOR cannot be accessed directly through OWL API.

WSClassifier outperforms MORe on DOLCE and all the Galen ontologies. For the Galen ontologies, MORe assigns all the classification work to a default configured HermiT; fine-tuning may improve its times. However, MORe computes only subsumptions implied by the TBox, ignoring the ABox, thus its classification result is incomplete for some ontologies with ABoxes, such as Wine.

WSClassifier seems most applicable when the ontologies are large and highly cyclic since then tableau reasoners construct large models and employ expensive blocking strategies. On the other hand consequence-based reasoners do not encounter problems on highly cyclic ontologies, and so can classify even cyclic $O_w$. 
and $O$, quickly. If there are up to a few additional subsumptions derived by $O$, AR does at most a little work on the highly cyclic $O$. This improvement is observed for FMA-C which is the only real-world large and complex ontology with nominals we have access to.

### 8.4 Experiments with $\mathcal{ALCHOI}(D)$ Ontologies

Based on the proposed approach for $\mathcal{ALCHOI}(D)^-$ ontology classification in previous chapters, we evaluated WSClassifier and other reasoners on all large and complex $\mathcal{ALCHOI}(D)^-$ ontologies available to us and on some proposed variants of Galen ontologies. We used a combination of the approaches, as explained in Sections 8.2 and 8.3, to produce the Galen variant ontologies. FMA-C is constructed in the same way as in Section 8.2 except nominals are kept. Galen-Union-n-D, Galen-Full-n1YE-D, and Galen-Full-n2YE-D are obtained from Full-Galen.\(^8\) The nominals introduced to the Galen ontologies are similar to the nominals introduced to the corresponding ontologies without “D” in the name in Section 8.3 and the datatypes introduced are similar to that of the corresponding ontologies in Section 8.2. In all cases, we reduce the language to $\mathcal{ALCHOI}(D)^-$. The ontologies are available from our website\(^9\).

From Table 8.3, we can see the performance of WSClassifier on the $\mathcal{ALCHOI}(D)^-$ ontologies is similar to or longer than that of $\mathcal{ALCHO}$ ontologies. This is because the $(D)^-$ and $O$ extension only need MR, which is very fast, see Table 8.1. $O$ may

---

\(^8\)\url{http://www.co-ode.org/galen/}

\(^9\)\url{http://isel.cs.unb.ca/~wsong/WSClassifierExperimentOntologies.zip}
Table 8.3: Comparison of classification performance of $\mathcal{ALCHOI}(D)$ ontologies

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Language</th>
<th>(Hyper) tableau</th>
<th>Hybrid</th>
<th>Couple T+C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>HermiT</td>
<td>Pellet</td>
<td>FaCT++</td>
</tr>
<tr>
<td>Galen-Heart-n1D</td>
<td>$\mathcal{ALCHOI}$</td>
<td>267.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Galen-Heart-n2D</td>
<td>$\mathcal{ALCHOI}$</td>
<td>770.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Galen-Full-n1YED</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Galen-Full-n2YED</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Galen-EL-n1YED</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>244,720</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Galen-EL-n2YED</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>378,648.6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Galen-Union-nD</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>FMA-C-D</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Note: The time is measured in seconds. "–" means out of time or memory.

Table 8.4: Comparison of classification performance of $\mathcal{ALCHOI}(D)^-\$ ontologies with Konclude

<table>
<thead>
<tr>
<th>Ontology</th>
<th>Language</th>
<th>Concepts</th>
<th>Individuals</th>
<th>Inverse axioms</th>
<th>Axioms with datatypes</th>
<th>Axioms</th>
<th>Konclude</th>
<th>WSClassifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Galen-Heart-n1</td>
<td>$\mathcal{ALCHOI}$</td>
<td>3,366</td>
<td>55</td>
<td>94</td>
<td>0</td>
<td>14,290</td>
<td>0.8</td>
<td>4.7</td>
</tr>
<tr>
<td>Galen-Heart-n2D</td>
<td>$\mathcal{ALCHOI}$</td>
<td>3,366</td>
<td>92</td>
<td>94</td>
<td>0</td>
<td>14,290</td>
<td>4.3</td>
<td>10.0</td>
</tr>
<tr>
<td>Galen-Full-n1YED</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>23,142</td>
<td>597</td>
<td>475</td>
<td>45</td>
<td>64,032</td>
<td>68,901.4</td>
<td>32.4</td>
</tr>
<tr>
<td>Galen-Full-n2YED</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>23,142</td>
<td>1,097</td>
<td>475</td>
<td>45</td>
<td>65,311</td>
<td>69,958.6</td>
<td></td>
</tr>
<tr>
<td>Galen-EL-n1YED</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>23,142</td>
<td>597</td>
<td>0</td>
<td>45</td>
<td>64,421</td>
<td>5.0</td>
<td>18.7</td>
</tr>
<tr>
<td>Galen-EL-n2YED</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>23,142</td>
<td>1,113</td>
<td>0</td>
<td>45</td>
<td>62,358</td>
<td>9.728</td>
<td>35,046.0</td>
</tr>
<tr>
<td>Galen-Union-nD</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>23,142</td>
<td>597</td>
<td>475</td>
<td>45</td>
<td>64,115</td>
<td>-</td>
<td>34.8</td>
</tr>
<tr>
<td>FMA-C-D</td>
<td>$\mathcal{ALCHOI}(D)$</td>
<td>41,648</td>
<td>85</td>
<td>28</td>
<td>293</td>
<td>272,486</td>
<td>-</td>
<td>27.5</td>
</tr>
</tbody>
</table>

Note: The time is measured in seconds. "–" means out of time or memory.

need AR to verify many subsumptions, which may take significantly long time, see Table 8.2. So comparing $I$ and $(D)^-$, $O$ affect the performance of WSClassifier more significantly. However, $I$ and $(D)^-$ can aggregate the impact of $O$ when they occur together. If the strengthening axioms of $(D)^-$ and $I$ does not increase the strengthening axioms for $O$, the classification time of WSClassifier on the $\mathcal{ALCHOI}(D)^-$ ontology is similar to the corresponding $\mathcal{ALCHOI}$ ontology. Otherwise, the performance of WSClassifier on the $\mathcal{ALCHOI}(D)^-$ ontology will degrade. Galen-EL-n2YE ontology is such an example: in Table 8.2 the time is 25,630s for the $\mathcal{ALCHOI}$ ontology; and in Table 8.3 the time is 35,046s for the
\(\text{ALCHOI}(\mathcal{D})^{-}\) ontology. Except this ontology, the other six ontologies in the experiment in Table 8.3 have not been affected by \(\mathcal{D}^{-}\) and \(\mathcal{I}\) for WSClassifier. Galen-Full-n2YE-D contains unsatisfiable concepts in \(H_s\) and WSClassifier fails over to HermiT to do the entire classification. Since HermiT cannot obtain result in time limit, WSClassifier did not get result, either.

For other reasoners in the experiments, their performance is mainly affected by the cycles, most of them cannot obtain results. However, for HermiT, we can also see some impact for the combination of \(\mathcal{D}^{-}\), \(\mathcal{I}\) and \(\mathcal{O}\). HermiT does not obtain results for FMA-C-D in \(\text{ALCOI}(\mathcal{D})\) in four days in Table 8.3, but takes only one and a half days to finish classification for FMA-C in \(\text{ALCO}\) in Table 8.2. The interaction of \(\mathcal{I}\) and \(\mathcal{O}\) makes the reasoning procedure for HermiT more complex than that of on \(\text{ALCHO}\). See Section 8.5.4 for more detail about HermiT.

We also conducted an experiment with the winner of ORE 2014 DL classification competition – Konclude, using the same ontologies as in Table 8.3. The result is shown in Table 8.4. Konclude outperforms WSClassifier on four ontologies, while WSClassifier outperforms Konclude on the other four. WSClassifier did not obtain the classification result for Galen-Full-n2YED.owl because the strengthening axioms make some of the concepts unsatisfiable. Thus, WSClassifier fails over to HermiT, which is also unable to classify the ontology. For Galen-EL-n2YED.owl, WSClassifier is slower because there are many subsumptions in \(H_s \backslash H_{wk}\) which needs to be verified by the assistant reasoner HermiT. Konclude did not obtain
results on the last two ontologies since they contain disjunctions. The saturation rules do not handle disjunction, so Konclude goes back to the Tableau algorithm. Since these two ontologies are large and highly cyclic and have disjunctions, the memory-consuming Tableau algorithm ends up running out of memory.

### 8.5 ORE2013 Competition

Our system participated in the 2013 international OWL Reasoner Performance Competitions held in ULM, Germany [20] and won the Live OWL2 DL classification competition [21]. The purpose of the competition is to systematically evaluate reasoners supporting all or various subset of the features of OWL. The competition was organized by the University of Manchester. There are 14 participating reasoners from six countries, which includes the majority of the most active OWL reasoners around the world. WSClassifier participated in the competition of classification OWL 2 DL ontologies.

In the following, we will introduce the methodology of the competition in Section 8.5.1, the result of OWL2 DL Offline Competition in Section 8.5.2, the result of OWL2 DL Live Competition in Section 8.5.3. Finally, we compare the features of the offline competition winner HermiT with the live competition winner WSClassifier in Section 8.5.4.
8.5.1 Methodology

- **Hardware**: The experiments were run on a cluster of identical computers (one reasoner per computer). Each computer had the following configuration: 1) Intel Xeon QuadCore CPU @2.33GHz; 2) 12GB RAM (8GB assigned to the process); 3) Running the Fedora 12 operating system; and 4) Java version 1.6.

- **Timeout**: The reasoners are asked to enforce a five minute timeout.

- **Correctness check**: The reasoner output was checked for correctness by a majority vote, i.e. the result returned by the most reasoners was considered to be correct.

- **Success and Failure**: In the end, the outcome of a reasoning task on an ontology was either success or fail. A reasoner would pass the test (solve the problem) successfully if it met the following three criteria: 1) Process the ontology without throwing an error; 2) Return a result within the allocated timeout; and 3) Return the correct result (based on majority vote).

- **Error**: e.g. parsing error, out of memory, unsupported OWL feature, etc.

- **Test corpus**: 1) the NCBO BioPortal corpus; 2) the Oxford Ontology Library; 3) the corpus of ontologies from the Manchester Ontology Repository. These three are the sample ontology pool, composed of 2499 ontologies; and 4) user-submitted ontologies. Besides these four, the May 2013 version of the National Cancer Institute (NCI) Thesaurus (NCIt) [28] and
the January 2011 version of the Systematized Nomenclature of Medicine (SNOMED) Clinical Terms (SNOMED CT) were also added to the corpus to the DL bin.

- **Ontologies selection approach**: For the OWL2 DL, they chose to include those ontologies that do not fall into any of the subprofiles (i.e. OWL 2 EL, RL, or QL) in order to ensure that features outside the sub-profiles were tested. A stratified random sample was drawn to obtain a set of 200 ontologies: 1) 50 small ontologies (between 100 and 499 logical axioms); 2) 100 medium sized ontologies (between 500 and 4,999 logical axioms); 3) 50 large ontologies (5,000 and more logical axioms). The entire sampling process was performed twice in order to create two complete test sets: Set A for the offline competition and Set B for the live competition. Some ontologies occurred in both Set A and B: 40 ontologies occurred in both Set A and B for the DL category.

### 8.5.2 OWL2 DL Offline Competition

In the offline competition, 204 ontologies were selected. Each reasoner ran until it had finished processing all ontologies. The reasoner that correctly classified the most ontologies won this competition. Figure shows an overview of the number of correctly processed ontologies and the average classification time per ontology per reasoner in the DL offline competition. To deeply analyze the competition result, we may need to investigate all the
features of the total over 400 ontologies in the offline and live competition, which is an impossible work within my Ph.D time limit. However, we still can display some interesting results here and will do an analysis in Section 8.5.4. Among all the reasoners in the competition, HermiT, FacT++, \textit{MOREHermiT}, \textit{MOREPellet} and Trowl are the related reasoners we have introduced in previous chapters and used in our own experiments. So we show the detail competition record of these reasoners and WSClassifier in Table 8.5. The definition on the features can be found in the above 8.5.1.

![Classification OWL 2 DL Ontologies](image)

Figure 8.1: Offline Competition: OWL2 DL Classification (from ore 2013 official report [30])
Table 8.5: Offline Competition Record

<table>
<thead>
<tr>
<th>reasoner</th>
<th>errors</th>
<th>timeouts</th>
<th>missing or incorrect</th>
<th>correct result</th>
<th>time per correctly processed ontology(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermit</td>
<td>37</td>
<td>13</td>
<td>7</td>
<td>147</td>
<td>12.3</td>
</tr>
<tr>
<td>FacT++</td>
<td>11</td>
<td>15</td>
<td>53</td>
<td>125</td>
<td>9.07</td>
</tr>
<tr>
<td>MOReHermiT</td>
<td>22</td>
<td>18</td>
<td>23</td>
<td>141</td>
<td>8.16</td>
</tr>
<tr>
<td>MORePellet</td>
<td>1</td>
<td>25</td>
<td>37</td>
<td>141</td>
<td>2.84</td>
</tr>
<tr>
<td>Trowl</td>
<td>0</td>
<td>3</td>
<td>86</td>
<td>115</td>
<td>5.23</td>
</tr>
<tr>
<td>WSClассifier</td>
<td>15</td>
<td>1</td>
<td>73</td>
<td>115</td>
<td>13.8</td>
</tr>
</tbody>
</table>

8.5.3 OWL2 DL Live Competition

In the live competition, 222 ontologies were selected. The online competition enforces the same five minute timeout per ontology per reasoner as the offline competition, but there is a strict total timeout of two hours. The number of ontologies correctly classified by each reasoner within the time are counted. Figure 8.2 gives an overview of the number of correctly processed ontologies and the average classification time per ontology for per reasoner in the DL live competition. Table 8.6 gives the detail record of more features in the competition for the related reasoners and WSClассifier. Note that for HermiT, the record shows 110 timeouts. However, each timeout needs five minutes, so it is impossible to have this number within total two hours’ limitation. The reason may be because HermiT did not fully follow the error processing and reporting requirement that the competition required and thus other errors of HermiT were reported as timeout.

8.5.4 Comparing HermiT and WSClассifier

In the following we give a comparison of the offline competition winner HermiT and the live competition winner WSClассifier.
There are two cases where HermiT may take longer time, sometimes significantly, than WSClassifier to classify an ontology: 1) the ontology uses expressions beyond $\mathcal{ALCHOI}(\mathcal{D})^-$ language; 2) the ontology is highly cyclic.

1. **Expressive Ontologies**: HermiT is a full OWL 2 reasoner supporting reasoning on $\mathcal{SROIQ}(\mathcal{D})$ ontologies. However, it employs different procedures and optimization strategies for ontologies in different sublanguages of OWL 2. When the ontologies are more expressive than $\mathcal{ALCHOI}(\mathcal{D})^-$, e.g. containing a combination of $O$, $I$, and $Q$, HermiT has to employ more expensive reasoning procedures to ensure the completeness of the results,
Table 8.6: Live Competition Record

<table>
<thead>
<tr>
<th>reasoner</th>
<th>errors</th>
<th>timeouts</th>
<th>not finished</th>
<th>missing or incorrect</th>
<th>correct processed ontology</th>
<th>time per correctly processed ontology(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hermit</td>
<td>0</td>
<td>110</td>
<td>38</td>
<td>5</td>
<td>68</td>
<td>11.26</td>
</tr>
<tr>
<td>FacT++</td>
<td>0</td>
<td>18</td>
<td>18</td>
<td>121</td>
<td>82</td>
<td>11.5</td>
</tr>
<tr>
<td>MORE\textsubscript{HermiT}</td>
<td>0</td>
<td>64</td>
<td>64</td>
<td>53</td>
<td>104</td>
<td>7.37</td>
</tr>
<tr>
<td>MORE\textsubscript{Pellet}</td>
<td>0</td>
<td>25</td>
<td>25</td>
<td>114</td>
<td>82</td>
<td>1.75</td>
</tr>
<tr>
<td>Trowl</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>194</td>
<td>26</td>
<td>10.8</td>
</tr>
<tr>
<td>WSClassifier</td>
<td>10</td>
<td>4</td>
<td>0</td>
<td>54</td>
<td>153</td>
<td>5.05</td>
</tr>
</tbody>
</table>

thus taking more time than WSClassifier. WSClassifier reduces all ontologies to $\mathcal{ALCHOI}(D)^-$ before classification, thus being more efficient at the cost of getting incomplete results on some ontologies.

2. **Highly Cyclic Ontologies**: As mentioned in previous chapters, highly cyclic ontologies are usually difficult for tableau-based reasoners. HermiT employs advanced blocking techniques that improve its classification time for these ontologies, but it still much less efficient than consequence-based reasoners. Since WSClassifier uses a consequence-based reasoner as the main reasoner, it is usually more efficient for those ontologies.

There are two cases where WSClassifier may take more time than HermiT to classify an ontology, but the time usually is not long: 1) the ontology is small; 2) there are too many subsumptions that need to be checked.

1. **Small ontologies**: three are three kinds of overheads for WSClassifier: 1) encoding and calculating strengthening axioms; 2) I/O communication with ConDOR; 3) check possibly unsound subsumption pairs from the strengthened ontology. When the input ontology is small, these overheads take up a
large percentage of total classification time, which sometimes may be even longer than the classification time of weakened and strengthened ontologies. In such cases, WSClassifier may be slower than other reasoners. However, in most cases, the absolute time of the overhead is still short.

2. **Too many subsumptions need to be checked:** If the set $H_s \setminus H_w$ is large, the assistant reasoner AR may take a lot of time to check their soundness. Thus, in some situations, WSClassifier may also fail over the entire classification task to AR, as explained in Section 5.1. If a fail-over occurs, our overhead compares to using AR to classify the ontology is about two rounds of classification using MR, which are usually short comparing with the time of using AR to classify the ontology.

The above analysis provides some hints why WSClassifier did not win in the offline competition but won in the live competition: 1) in the live competition there was a 2-hour total time limit, so that HermiT was not able to process 38 ontologies, while WSClassifier finished processing all 222 ontologies; 2) WSClassifier does not support full OWL 2 DL profile, so that the results may be incomplete. However, the constructors beyond $\text{ALCHOI}(D)^-$ did not produce subsumptions for many ontologies used in the live competition.

### 8.6 Chapter Summary

In this chapter, we conducted experiments on available cyclic ontologies and their variants to compare the performance of WSClassifier with other reasoners. The
results shows that

- on the large and highly cyclic $\mathcal{ALCHI}(\mathcal{D})^{-}$ ontologies, WSClassifier is significantly faster than tableau based reasoners;

- on the large and highly cyclic $\mathcal{ALCHO}$ ontologies, WSClassifier is significantly faster in most cases than tableau based reasoners. When it is slower, the time difference is very small.

- on the large and highly cyclic $\mathcal{ALCHOI}(\mathcal{D})^{-}$ ontologies, the results are similar to $\mathcal{ALCHO}$ ontologies, since the performance is mostly determined by the reasoning time of $\mathcal{ALCHO}$.

Finally, for more systematic testing of WSClassifier’s performance, we also participated in the ORE2013 DL classification performance competitions and won the live DL classification competition. The results are analyzed in Section 8.5.
Chapter 9

Methodology Evaluation

In this chapter, we first evaluate our methodology by comparing WSClassifier with other reasoners in two dimensions [16] – reasoning characteristics and practical usability. Then, we further analyze and compare the characteristics of extension-based reasoners which includes WSClassifier.

In the following, we explain the evaluation criteria of the two dimensions in Sections 9.1 and 9.2 respectively. Next, we briefly introduce the reasoners used for the comparison in Section 9.3. After that, we compare the reasoners based on the two dimensions in Section 9.4. Finally, we compare the characteristics of WSClassifier with other extension-based reasoners in Section 9.5.2.
9.1 Reasoning Characteristics

This dimension consists of the following four aspects.

- **Methodology**
  
  There are three kinds of methodologies employed by OWL reasoners for ontology classification. (Hyper)tableau-based procedures aim at classifying expressive ontologies. Consequence-based procedures, which are originated from completion rules, are developed for efficient reasoning on less expressive ontologies. Hybrid reasoning procedures combine these two techniques and take the advantages of both. The approaches have been explained in Sections 2.2, 2.3, and 2.4.

- **Soundness and Completeness in Theory**
  
  This property evaluates whether the inferences of the employed reasoning methods are sound and complete, defined in Section 2.1 based on the underlying semantics. Some reasoners sacrifice soundness or completeness for the efficiency of reasoning. Note that the soundness and completeness of the underlying theory does not mean the implementation is sound or complete.

- **Expressivity and Computational Complexity**
  
  The tradeoff between the expressiveness of languages and the computational complexity of reasoning, has always been an essential research problem in Description Logics [2]. This property evaluates the expressivity of the
languages supported by different reasoners and the corresponding computational complexity. Table 9.1 lists several DLs with their corresponding worst-case complexities for concept satisfiability checking taken from the literature.

- **Supporting ABox Reasoning**

ABox reasoning is reasoning with individuals and comprises classification, instance checking, (conjunctive)query answering and ABox consistency checking.

### 9.2 Practical Usability

We introduce the practical usability criteria in the following.

- **OWLAPI**

The OWL API is an Application Programming Interface (API) for leveraging OWL ontologies. It supports parsing and rendering in the syntaxes...
defined by the OWL 2 specification. It provides a standard Java interface for OWL reasoners, so that an application can embed different reasoners without having to change its implementation.

- **Licence**

Many reasoners come with a dual license. This means that they are free under certain conditions, and that for different use, arrangements have to be made with their developers. The major distinguishing feature concerning licenses is whether the license is a recognized open source license or not.

- **Further Characteristics**

The remaining characteristics include whether the reasoner is open source, what is the programming language the reasoner is implemented, and the supported platforms (Windows/Linux/Mac OS).

### 9.3 Reasoners

In the following, we briefly introduce some state-of-the-art (Hyper)tableau-based reasoners, consequence-based reasoners and hybrid reasoners, which we will use in the two-dimension comparisons in Section 9.4.

- **FaCT++** (University of Manchester) is a tableaux reasoner written in C++ which supports the full OWL 2 DL profile.

- **HermiT** (University of Oxford) is a Java-based OWL 2 DL reasoner implementing a hypertableau calculus. It has been introduced in Sections 2.2.4.
and 8.5.4.

- Pellet (Clark & Parsia) is Java-based reasoner supporting the full OWL 2 profile.

- RacerPro (Renamed ABox and Concept Expression Reasoner, Racer Systems. Concordia University) implements the description logic $SHIQ(D)^\neg$. Dedicated optimizations for OWL 2 EL have been added to the system. RacerPro was the first system which efficiently supported concrete domains for TBox and ABox reasoning.

- MORe (University of Oxford) is Java-based modular reasoner which integrates a full-fledged (and slower) reasoner with a profile specific (and more efficient) reasoner.

- WSClassifier (University of New Brunswick) is Java-based reasoner for the $ALCHOI(D)^\neg$ fragment of OWL 2 DL, using a hybrid of the consequence-based reasoner ConDOR and hypertableau reasoner HermiT.

- TrOWL (University of Aberdeen) is an approximative OWL 2 DL reasoner. In particular, TrOWL utilizes a semantic approximation to transform OWL 2 DL ontologies into OWL 2 QL for conjunctive query answering and a syntactic approximation from OWL 2 DL to OWL 2 EL for TBox and ABox reasoning.

- CB (University of Oxford) is an implementation of a reasoning procedure [46] for Horn $SHIQ$ ontologies.
• ConDOR (University of Oxford) An experimental reasoner written in C++ for the description logic SH. It is used as the main reasoner in our WSClassifier project.

• ELK (University of Oxford and University of ULM) is a consequence-based Java reasoner which utilises multiple cores/processors by parallelising multiple threads. The goal of ELK is to fully support the OWL 2 EL profile and it currently supports (most of) the OWL 2 EL ontology language.

• Konclude (University of Ulm) is a tableau-based reasoner for $SROIQ^V(D)$ with extensions for the handling of nominal schemas. Konclude also integrates the saturation algorithm covering DL Horn-SRIF into the system. Therefore, we regard Konclude as a hybrid reasoner. It is written in C++.

### 9.4 Comparison of Reasoners

In this section, we compare the above listed reasoners according to their reasoning characteristics and practical usability in Sections 9.4.1 and 9.4.2 respectively.

### 9.4.1 Reasoning Characteristics

Table 9.2 lists the reasoning characteristics of all reasoners.

- **Methodology** The reasoners employ either a (Hyper)tableau calculus, or a consequence-based reasoning or completion rules, or the hybrid of consequence and tableau-based approaches.
• **Soundness and Completeness in Theory** The underlying theories of all reasoners except TrOWL are proved to be sound and complete w.r.t. the languages they support. TrOWL is soundness preserving but possibly incomplete.

• **Expressivity and Computational Complexity** The languages supported by the reasoners are listed in Table 9.2. CB’s reasoning procedure [46] supports Horn-\(SHIQ\), but its implementation supports only Horn-\(SHIF\). TrOWL implements \(EL^{++}\) without datatypes and supports \(SROIQ\) by approximation. If a third-party reasoner is used, then its supported expressivity is the one of this reasoner.

• **Supporting ABox Reasoning** In Table 9.2, all tableau-based reasoners, Conclude, ELK and TrOWL support ABox reasoning. The two consequence-based reasoner CB and ConDOR, as well as the hybrid reasoner MORe do not support ABox reasoning. WSClassifier partially supports ABox reasoning – classification with ABox, which is the basis for the other ABox reasoning tasks 9.1.

### 9.4.2 Practical Usability

Table 9.3 compares the reasoners based on the practical usability criteria.

• **OWL API**
Table 9.2: Reasoning characteristics

<table>
<thead>
<tr>
<th>Reasoner</th>
<th>Methodology</th>
<th>Sound</th>
<th>Complete</th>
<th>Expressivity</th>
<th>Supporting Abox</th>
</tr>
</thead>
<tbody>
<tr>
<td>HermiT</td>
<td>hypertableau</td>
<td>Y</td>
<td>Y</td>
<td>SROIQ(D)</td>
<td>Y</td>
</tr>
<tr>
<td>FaCT++</td>
<td>tableau</td>
<td>Y</td>
<td>Y</td>
<td>SROIQ(D)</td>
<td>Y</td>
</tr>
<tr>
<td>Pellet</td>
<td>tableau</td>
<td>Y</td>
<td>Y</td>
<td>SROIQ(D)</td>
<td>Y</td>
</tr>
<tr>
<td>RacerPro</td>
<td>tableau</td>
<td>Y</td>
<td>Y</td>
<td>SROIQ(D)</td>
<td>Y</td>
</tr>
<tr>
<td>MORe</td>
<td>hybrid</td>
<td>Y</td>
<td>Y</td>
<td>SROIQ(D)</td>
<td>N</td>
</tr>
<tr>
<td>CB</td>
<td>consequence</td>
<td>Y</td>
<td>Y</td>
<td>Horn-SHIF</td>
<td>N</td>
</tr>
<tr>
<td>ConDOR</td>
<td>consequence</td>
<td>Y</td>
<td>Y</td>
<td>SH</td>
<td>N</td>
</tr>
<tr>
<td>ELK</td>
<td>consequence</td>
<td>Y</td>
<td>Y</td>
<td>EL++</td>
<td>Y</td>
</tr>
<tr>
<td>TrOWL</td>
<td>completion</td>
<td>rules</td>
<td>Y</td>
<td>N</td>
<td>approximating SROIQ subset of EL++</td>
</tr>
<tr>
<td>Konclude</td>
<td>hybrid</td>
<td>Y</td>
<td>Y</td>
<td>SROIQV(D)</td>
<td>Y</td>
</tr>
<tr>
<td>WSClassifier</td>
<td>hybrid</td>
<td>Y</td>
<td>Y</td>
<td>ALCHOI(D)</td>
<td>partially</td>
</tr>
</tbody>
</table>

All reasoners except ConDOR support OWL API, so that they can be easily accessed by Java applications through the same interface.

- **Licence**

HermiT, FaCT++, MOReHermiT and MORePellet, Konclude and CB can be redistributed or modified under the terms of GNU Lesser GPL (LGPL). ConDOR and ELK come with Apache License 2.0 (AP 2.0). Pellet comes with a dual license: software that is released under a recognized open source license can use Pellet under the terms of the Affero General Public License (AGPL), for other software, another license has to be arranged. This has the advantage that the community benefits from source code that uses Pellet under its open source license. TrOWL may be used under the terms of the AGPL for open source applications and is available under alternative...
license terms for proprietary, closed-source applications and other commercial applications. RacerPro is distributed under a BSD 3-clause license. WSClassifier comes with MIT. MIT, LGPL, AP 2.0 and BSD 3-clause are open source licence.

- *Further Characteristics*

The remaining columns of Table 9.3 lists further characteristics of the reasoners, including whether they are open source, the languages they are implemented in, and the platform they support.

Table 9.3: Practical Usability

<table>
<thead>
<tr>
<th>Reasoner</th>
<th>OWL API</th>
<th>License</th>
<th>Open Source</th>
<th>Language</th>
<th>Platform</th>
</tr>
</thead>
<tbody>
<tr>
<td>HermiT</td>
<td>Y</td>
<td>LGPL</td>
<td>Y</td>
<td>Java</td>
<td>all</td>
</tr>
<tr>
<td>FaCT++</td>
<td>Y</td>
<td>LGPL</td>
<td>Y</td>
<td>C++</td>
<td>all</td>
</tr>
<tr>
<td>Pellet</td>
<td>Y</td>
<td>DuLi:AGPL</td>
<td>Y</td>
<td>Java</td>
<td>all</td>
</tr>
<tr>
<td>RacerPro</td>
<td>Y</td>
<td>BSD 3-clause</td>
<td>Y</td>
<td>lisp</td>
<td>all</td>
</tr>
<tr>
<td>MORE</td>
<td>Y</td>
<td>LGPL</td>
<td>Y</td>
<td>Java</td>
<td>all</td>
</tr>
<tr>
<td>CB</td>
<td>Y</td>
<td>LGPL</td>
<td>Y</td>
<td>OCaml</td>
<td>all</td>
</tr>
<tr>
<td>ConDOR</td>
<td>N</td>
<td>AP 2.0</td>
<td>Y</td>
<td>C++</td>
<td>all</td>
</tr>
<tr>
<td>ELK</td>
<td>Y</td>
<td>AP 2.0</td>
<td>Y</td>
<td>java</td>
<td>all</td>
</tr>
<tr>
<td>TrOWL</td>
<td>Y</td>
<td>DuLi:AGPL</td>
<td>Y</td>
<td>Java</td>
<td>all</td>
</tr>
<tr>
<td>Konclude</td>
<td>via OWLink</td>
<td>LGPL</td>
<td>Y</td>
<td>C++</td>
<td>all</td>
</tr>
<tr>
<td>WSClassifier</td>
<td>Y</td>
<td>MIT</td>
<td>Y</td>
<td>Java</td>
<td>all</td>
</tr>
</tbody>
</table>

1https://www.ifis.uni-luebeck.de/index.php?id=385
9.5 Characteristics of Extension-Based Reasoners

In this section, we compare the characteristics of reasoners that aim at extending a fast but less expressive reasoner for more expressive languages. All these reasoners use an efficient main reasoner MR to do the majority of classification work. Some of them also use an assistant reasoner AR to help achieve reasoning soundness and completeness. Section 9.5.1 introduces the characteristics to be compared. Section 9.5.2 gives the comparison.

9.5.1 Characteristics

In the section, we introduce the following characteristics of extension-based approaches.

- Hybrid reasoner
  
  This characteristic indicates whether the reasoner uses a combination of two reasoners which usually have different efficiency and support different language expressivity.

- Expressivity reduction approach

  To generate the input ontology for the main reasoner, most of the extension-based reasoners reduce the expressivity of languages of the ontologies by transforming axioms beyond the supported language of the main reasoner into axioms within the language, while some reasoner gains intended expressivity by conducting modular extraction to obtain sub ontologies with
the required constructors.

- Reduced language features
  This characteristic compares the language features that are reduced.

- Black or white box
  The reasoners employed are treated as black boxes or white boxes

- Main reasoner and property of the input ontology
  The main reasoner is expected to do all or most of the reasoning work. It is usually a fast reasoner supporting a less expressive language. We compare the main reasoner for each of the extension-based reasoners, as well as the properties of the input ontology of MR.

- Assistant reasoner and property of the input ontology
  The assistant reasoner is used to achieve soundness and completeness. We compare each employed assistant reasoner and the properties of its input ontology.

- Efficiency on highly cyclic ontologies
  This characteristic discusses whether the approach can handle highly cyclic ontologies efficiently.
9.5.2 Comparison of Extension-Based Reasoners

In this section, we compare state-of-the-art extension-based reasoners in Table 9.4 according to the characteristics listed in the previous Section. There are four reasoners to be compared: TrOWL, MORe, YJ and WSClassifier. YJ refers to the reasoner implemented in [95], which focuses on query answering.

- Hybrid reasoner
  
  MORe, YJ and WSClassifier are hybrid reasoners, while TrOWL is not.

- Expressivity reduction approach
  
  TrOWL, YJ and WSClassifier use transformation and MORe uses modular extraction approach to reduce the expressivity of languages of the input ontologies.

- Reduced language features
  
  TrOWL transforms disjunctions and universal restrictions into negations of conjunctions and existential restrictions, respectively. It transforms number restrictions into weaker axioms, hence not preserving completeness.

  YJ first converts the input ontology into Datalog$^{\pm,\lor}$ rules, and then transforms the existential-headed and disjunction-headed rules in Datalog$^{\pm,\lor}$ into stronger Datalog rules, thus reducing the expressivity to Datalog. These transformed Datalog$^{\pm,\lor}$ rules correspond to OWL 2 subsumption axioms using existential restrictions and disjunctions as superclasses.
Table 9.4: Characteristics of Extension-based Approaches Table

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>TrOWL</th>
<th>MORRe</th>
<th>YJ</th>
<th>WSClassifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hybrid reasoner</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Expressivity reduction method</td>
<td>transformation</td>
<td>modular extraction</td>
<td>transformation</td>
<td>transformation</td>
</tr>
<tr>
<td>Reduced language features</td>
<td>disjunction</td>
<td>universal-restriction complement cardinality</td>
<td>disjunction-headed and existential-headed Datalog±\lor\lor rules</td>
<td>datatypes inverse roles nominals</td>
</tr>
<tr>
<td>Black or white boxes</td>
<td>white</td>
<td>black</td>
<td>black</td>
<td>black</td>
</tr>
<tr>
<td>MR and property of the input ontology</td>
<td>self-developed extension of $E\mathcal{L}^{++}$ reasoner REL, with transformed ontology as input</td>
<td>requires $MR$ to handle nominal, with input $\mathcal{M}^E$, all $A \in \Sigma^E$ are completely classified in $\mathcal{M}^E$</td>
<td>RL reasoner with input $O_L, O_U, cert(Q, O, D)$ $\subseteq$ $cert(Q, O_L, D)$ $\subseteq$ $cert(Q, O_U, D)$</td>
<td>$\mathcal{ALCH}$ reasoner with input $O_w, O_s$, $H_w \subseteq H_o \subseteq H_s$</td>
</tr>
<tr>
<td>AR and property of the input ontology</td>
<td>-</td>
<td>$SROIQ(\mathcal{D})$ reasoner, with input $\mathcal{M}^E$, all $B \in O \setminus \Sigma^E$ are completely classified in $\mathcal{M}^E$</td>
<td>$SROIQ(\mathcal{D})$ reasoner with input $O_f \cup D_f$, to verify whether $O_f \cup D_f \models Q(\bar{a})$ for each $\bar{a} \in G$, $O \cup D \models Q(\bar{a})$ iff $O_f \cup D_f \models Q(\bar{a})$</td>
<td>$\mathcal{ALCHOI}(\mathcal{D})^*$ reasoner with input $O$ to filter out unsound subsumption in $H_f \setminus H_w$</td>
</tr>
<tr>
<td>Efficiency on highly cyclic ontologies</td>
<td>good</td>
<td>fair</td>
<td>unknown</td>
<td>good</td>
</tr>
</tbody>
</table>
WSClassifier transforms datatypes, inverse roles and nominals as explained in Sections 4.1.1, 4.2.1 and 5.2 respectively.

- Black or white box

TrOWL modifies its MR and treats it as a white box. MORe, YJ and WS-Classifier treat their reasoners as black boxes.

- Main reasoner and property of the input ontology

TrOWL develops its own main reasoner called TrOWL REL, which extends the $\mathcal{EL}^{++}$ completion rules to support negation and number restriction. The inference is sound but may be incomplete. The input of TrOWL REL is a transformed ontology which encodes negation and number restrictions. Unlike the MR of the other three reasoners in this comparison, TrOWL’s MR cannot be replaced by other reasoners.

MORe requires its MR having the capability to handle nominal. MORe computes a signature $\Sigma^L \subseteq \text{Sig}(O)$ and an L-ontology $\mathcal{M}^L \subseteq O$ such that the classes in $\Sigma^L$ can be completely classified using only the axioms in $\mathcal{M}^L$.

The MR of YJ is a RL reasoner, of which the input ontologies are the lower bound and upper bound ontologies $O_L, O_U$, respectively. MR computes the lower and upper bound query answers $\text{cert}(Q, O_L, D)$ and $\text{cert}(Q, O_U, D)$ based on $O_L, O_U$, satisfying the property that: $\text{cert}(Q, O_L, D) \subseteq \text{cert}(Q, O, D) \subseteq \text{cert}(Q, O_U, D)$. Q denotes queries, D denotes data sets.

WSClassifier requires its MR to be an $\mathcal{ALCH}$ reasoner, which works on
the $O_w$ and $O_s$ ontologies. The classification of $O_w$ and $O_s$ satisfying the property: $H_w \subseteq H_o \subseteq H_s$.

- Assistant reasoner and property of the input ontology

MORe requires its AR to be a full-fledged OWL 2 SROIQ($\mathcal{D}$) reasoner. MORe computes an ontology $M^L \subseteq O$ such that the classes in $O \setminus \Sigma^L$ can be fully classified using only the axioms in $M^L$.

YJ requires its AR to be a full-fledged OWL 2 SROIQ($\mathcal{D}$) reasoner. YJ computes fragments $O_f$ of $O$ and $D_f$ of $D$ satisfying: $O \cup D \models Q(\bar{a})$ iff $O_f \cup D_f \models Q(\bar{a})$. $G = \text{cert}(Q, O_U, D) \setminus \text{cert}(Q, O_L, D)$, $\bar{a} \in G$. AR verifies whether $O_f \cup D_f \models Q(\bar{a})$ for each tuple $\bar{a} \in G$.

WSClассifier requires its AR to be an $\mathcal{ALCHOI}(\mathcal{D})^-$ reasoner, which works on the original ontology $O$ to filter out the unsound subsumptions in $H_s \setminus H_w$.

- Efficiency on highly cyclic ontologies

As explained in Section 2.4.3, MORe may not be efficient when the ontology is highly cyclic. The efficiency of YJ on cyclic ontologies is unknown. TrOWL and WSClассifier are efficient on many of the highly cyclic ontologies, see experiments in Chapter 8.
9.6 Chapter Summary

In this chapter, we first evaluate our methodology by comparing WSClassifier with other reasoners in two dimensions [16] – reasoning characteristics and practical usability. Then we further analyze and compare the characteristics of reasoners of a similar type to WSClassifier – those aim at extending a fast but less expressive reasoner for more expressive languages.
Chapter 10

Conclusions

In this chapter, we conclude the thesis in Section 10.1. We summarize the contribution of the research in Section 10.2 and present future work in Section 10.3.

10.1 Summary

The popularity of Description Logics in Knowledge Representation and modeling, has made DLs the topic of many research efforts focused on extending their expressivity as well as their reasoning efficiency. It has become essential for a DL not only to handle the expressivity to model all elements within an application’s domain, but also to allow efficient reasoning when the expressivity is fully used [23]. It is shown in Chapter 1 that existing (hyper)tableau-based reasoning procedures are not efficient for classifying large and highly cyclic ontologies of which the languages are beyond the capability of consequence-based reasoning.
Nominals \((O)\) are needed in real world domains as names for concepts with only one instance. Inverse role \((I)\) and datatypes \((D)\) are commonly used constructors. However, there are no consequence-based reasoners that support all of the three constructors.

The main objective of this thesis outlined in Section 1.2 is to design an efficient hybrid reasoning procedure to handle the expressivity of DLs \(\mathcal{ALCHOI}(D)^{-}\) based on a consequence-based \(\mathcal{ALCH}\) reasoning procedure. In the previous chapters, we have presented our general methodology, the procedures and proofs of single and multiple extensions, the system design and implementation as well as optimization, the experiment results and the outcomes in the ORE 2013 international competition, and the evaluation of the methodology. Through these contents, we have demonstrated that we have achieved our overarching goal from Chapter 1.

The next section shows which objectives from Section 1.2 have been met while identifying the main contributions of the thesis.

## 10.2 Contributions

The research methodology adopted consists of devising a theoretically sound, complete and terminating hybrid reasoning procedure, and designing a practical implementation of such a procedure. The contribution of this thesis is therefore twofold: theoretical and practical.
10.2.1 Theoretical Contributions

In this research, we propose a weakening and strengthening based approach, which employs transformation technology and hybrid reasoning when necessary, to extend a fast but less expressive $\mathcal{ALCH}$ reasoner for more expressive languages such as $\mathcal{ALCHOI}(D)^-$ with sound and complete results. We list the concrete contributions in the following:

(1) First, we have proposed the general idea and procedure of the weakening and strengthening approach for extending a reasoner with one constructor, i.e. single extension, as explained in Section 3.3. Hence, the single extension requirement of objective (1) from Section 1.2 is met.

(2) Second, based on (1), we propose two single extensions from $\mathcal{ALCH}$ to $\mathcal{ALCH}(D)^-$ and $\mathcal{ALCHI}$ with soundness-preserved strengthening. In these approaches, the axioms that use datatypes or inverse roles are transformed to obtain a strengthened ontology $O_s$ in $\mathcal{ALCH}$, which is sound and complete with regard to the original ontology. $O_s$ is then classified with the main $\mathcal{ALCH}$ reasoner to get the classification result of the original ontology. The approaches were proposed and published in paper [87, 86, 85]. See Chapter 4 for the detail of procedures and the proofs for soundness and completeness. Hence, the requirement of single extension of $(D)^-$ and $I$ of objective (2) and (4) from Section 1.2 is met.

(3) Third, based on (1), we propose a single extension from $\mathcal{ALCH}$ to $\mathcal{ALCHO}$ with soundness-relaxed strengthening, where the classification results of the
strengthened ontology may be unsound. Thus we employ two reasoners to re-establish the sound and complete results; this use of more than one reasoner is known as hybrid reasoning. The weakened and strengthened ontologies are in $\mathcal{ALCH}$ and classified by the main reasoner. Then a full-fledged tableau-based $\mathcal{SROIQ(D)}$ assistant reasoner is employed to filter out the unsound subsumptions in the classification of strengthened ontology. The approach was proposed and published in paper [84]. See Chapter 5 for the detail of procedure and the proofs for soundness and completeness. Hence, the requirement of single extension for $O$ of objective (2) and (4) from Section 1.2 is met.

Fourth, we propose the general principle of composing single extensions to multiple extensions and the strategy to create the weakened and strengthened ontologies in multiple extensions in Section 6. Then the requirement of multiple extension of objective (1) and (4) from Section 1.2 is met. Based on the principle and the three single extensions in (2) and (3), we demonstrate six paths to do four extensions from $\mathcal{ALCH}$ to $\mathcal{ALCHI(D)}^-$, to $\mathcal{ALCHOI}$, to $\mathcal{ALCHO(D)}^-$, and to $\mathcal{ALCHOI(D)}^-$. We prove the soundness and completeness for these four extensions in Section 6.2. Hence, objective (3) from Section 1.2 is met.
10.2.2 Practical Contributions

We have developed a prototype reasoner WSClassifier for classifying large and complex ontologies in $\mathcal{ALCHOI}(\mathcal{D})^-$. The reasoner implements a classification procedure for $\mathcal{ALCHOI}(\mathcal{D})^-$ explained in Chapters 6 and 7, based on the weakening and strengthening approach for $\mathcal{ALCH}(\mathcal{D})^-$, $\mathcal{ALCHI}$ and $\mathcal{ALCHO}$ explained in Sections 4.1 and 4.2 and Chapters 5. It combines a consequence-based $\mathcal{ALCH}$-reasoner ConDOR and an $\mathcal{ALCHOI}(\mathcal{D})^-$ reasoner HermiT and takes the advantage of both of them. ConDOR does the majority of the work on classifying the weakened and strengthened versions of the ontology. HermiT is used to check the possibly unsound subsumptions and filter out the unsound ones.

We evaluate WSClassifier against existing DL reasoners on available large and highly cyclic ontologies and their variants. The empirical evaluation results show that WSClassifier outperforms or sometimes significantly outperforms other reasoners that support $\mathcal{ALCHOI}(\mathcal{D})^-$ expressivity on most of these ontologies. WSClassifier also participated in the international OWL Reasoner Performance Competition held in 2013 and won the live competition of DL Classification. Hence, the final objective – objective (5) from Section 1.2 is met.

10.3 Future Work

With increasing real-world demands, the DL language may become more and more expressive, and more fast consequence-based reasoners or other efficient
reasoners will be created, which may support less expressive languages than the slower full-fledged reasoners. The weakening and strengthening approach may still be an effective way applied in practical efficient reasoning. So we hope to do the following work in the future:

- extension to other datatypes and facets, and further optimization, e.g. adapting the idea of Magka et al. [55] to WSClassifier to distinguish positive and negative occurrences of data ranges, in order to reduce the number of axioms to be added.

- apply this approach to extend more DL constructors.

- study the parallel processing on the multiple extensions for different constructors.

- extend the approach for other reasoning tasks, e.g. conjunctive query answering.
References


the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009 (Craig Boutilier, ed.), 2009, pp. 714–720.


Vita

Candidate’s full name: Weihong Song

University attended (with dates and degrees obtained):

University of New Brunswick, Fredericton, NB, Canada
Ph.D. Candidate, Faculty of Computer Science, 2015, May

Tsinghua University, China. Master of Software Engineering, 2008, January

Tsinghua University, China. Second Bachelor in Software Engineering, 2004, June

Tsinghua University, China. Bachelor in Architecture, 1990, June

Publications:


**Conference Presentations:**

A transformation approach for classifying ALCHI(D) ontologies with a consequence-based ALCH reasoner, Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation, July, 2013, Ulm, Germany

Complete classification of complex ALCHO ontologies using a hybrid reasoning approach, Proceedings of the 26th International Workshop on Description Logics, July, 2013, Ulm, Germany

Semantic analysis for keywords based user segmentation from internet data, the 9th International Conference on Semantics, Knowledge and Grids, IEEE Computer Society, Oct. 2013, Beijing, China.

WSReasoner: A prototype hybrid reasoner for ALCHOI ontology classification using a weakening and strengthening approach, the first OWL Reasoner Evaluation Workshop, July, 2012, Manchester, UK.

Efficient Classification of Large Complex Ontologies, The 4th Atlantic Workshop on Semantics and Services. May 2012, Halifax, Nova Scotia, Canada

Hybrid reasoning for ontology classification, AI 2011 Graduate Student Symposium, the twenty-fourth Canadian Conference on Artificial Intelligence (2011), May, 2011, St.John’s, Newfoundland, Canada.