Technische Universität Dresden
Faculty of Computer Science
Institute of Theoretical Computer Science
Chair of Automata Theory

A Classification Algorithm For $\mathcal{ELHI}f_{\mathcal{R}^+}$

Master's Thesis
European Master's Program in Computational Logic

Author
Julian Alfredo Mendez

Supervisor
Prof. Dr.-Ing. Franz Baader

Dresden, Germany — March 2011

TECHNISCHE UNIVERSITÄT DRESDEN

Author:	**Julian Alfredo Mendez**
Matriculation number:	**3361365**
Title:	**A Classification Algorithm For ELHIfR+**
Degree:	**Master of Science**
Date of submission:	**2011-03-15**

Declaration

Hereby I certify that this thesis has been written by me. Any help that I have received in my research work has been acknowledged. Additionally, I certify that I have not used any auxiliary sources and literature except those I cited in this thesis.

Julian Alfredo Mendez

Abstract

Description logics are a family of knowledge representation formalisms for representing and reasoning about conceptual knowledge. Every description logic system has reasoning services that infer implicit knowledge from that explicitly given. Standard reasoning problems include concept satisfiability, concept subsumption, ABox consistency and the instance problem. This thesis focus on the concept subsumption service, which is considered to be the most common service.

Some years ago, a polynomial-time algorithm for the subsumption problem in the description logic $\mathcal{EL}$ was developed. After that, algorithms for different problems in tractable extensions of $\mathcal{EL}$ have been developed. These description logics are sufficient to represent many knowledge bases, e.g. a large medical ontology called SNOMED CT. However, there are ontologies requiring extensions of $\mathcal{EL}$ that are not tractable. In particular, GALEN, another important medical ontology, requires $\mathcal{ELHI}f_{\mathcal{R}^+}$, an extension of $\mathcal{EL}$ that includes role hierarchies, inverse, functional and transitive roles.

This thesis presents a classification algorithm for $\mathcal{ELHI}f_{\mathcal{R}^+}$. Together with this thesis there is an implementation available at `http://jcel.sourceforge.net`.

Acknowledgements

First of all, I would like to thank my advisor, Prof. Franz Baader, for sharing his broad experience in description logics with me and for giving me the wonderful possibility of working at the Chair of Automata Theory (LAT).

I would like to thank Barbara Morawska, Ph.D., who kindly read several drafts and gave me her valuable opinion. I would like to express my gratitude to Dr. Rajeev Goré, with whom I discussed how to work with inverse roles. I would like to thank my former advisors, Prof. Carsten Lutz, who taught me how to write a scientific paper in English, and Dr. Carlos Areces, who introduced me in the amazing world of non-classical logics.

I would like to thank my colleagues, Dr. Anni-Yasmin Turhan, for giving me the chance to prove my skills designing and programming; Marcel Lippmann, who had a helpful look at the implementation and gave me his suggestions; and Dr. Rafael Peñaloza, with whom I discussed different properties of description logics. I would like to thank Dr. Boontawee Suntisrivaraporn (Meng), who kindly explained the CEL algorithm to me.

I would like to express my gratitude to the secretaries Kerstin Achtruth and Sylvia Wünsch, who gave me a very important help.

I would like to thank my wife, Silvia, for her company all this time and for accepting the challenge of living abroad. She read many drafts and gave me her suggestions in the use of English. I would like to express my gratitude to my parents, Luis and Julia, my brother Fernando, and my friends across the world, for their constant and important support during my studies.

"*Piano, piano, si va lontano.*" (Popular saying)

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

The purpose of this thesis is to present a classification algorithm and an implementation for a particular description logic called $\mathcal{ELHIF}_{\mathcal{R}^+}$.

Description logics (DL) are a family of logic formalisms that originated in the field of artificial intelligence as a tool for knowledge representation. During the last two decades, description logics have been successfully applied in knowledge representation and in many other areas such as reasoning in formalisms based on classes (UML diagrams and databases), and in ontology engineering for the Semantic Web [Baader et al.(2003)Baader, Calvanese, McGuinness, Nardi, and Patel-Schneider].

The basic entities of description logics are *concepts*, which are built with *concept names* (seen as unary predicates) and *role names* (seen as binary relations), and a set of concept constructors and role constructors provided by the particular description logic.

An *ontology* is a formal vocabulary of terms which refers to a conceptual schema inside a domain. This is a hierarchical data structure with semantics containing entities and relevant relations in a domain, which describe specific topics like those having biological classification. The terms are related using an *ontology language*.

An *ontology reasoner* is a tool that can process an ontology. The main service that a reasoner provides is *classification*. The purpose of classification is to compute a hierarchical relation. Ontologies may contain errors and reasoners can help finding these errors. Reasoners are used in *ontology editors*, like Protégé[1].

Protégé is a free, open source ontology editor, and also a knowledge base framework. Protégé ontologies can be exported to several formats, like RDF, OWL and XML Schema. It uses the OWL API[2], which is a Java application programming interface (API) and reference implementation for creating, manipulating and serializing OWL ontologies. The OWL API 3 is focused on OWL 2, especially version 3.2.2 which is mentioned along this thesis.

These tools are oriented to their use on the Semantic Web[3]. This is a group of methods and technologies to give to machines the possibility of capturing the semantics of information on the World Wide Web. The term was coined by Tim Berners-Lee, defining it as "a web of data that can be processed directly and indirectly by machines."

The Semantic Web is at the present under development. However, there are practical examples

[1] `http://protege.stanford.edu/`
[2] `http://owlapi.sourceforge.net/`
[3] `http://semanticweb.org/`

of application of these tools in everyday life. This is the case of the medical ontologies, which are ontologies used in medical fields. In the following, we will focus on this kind of ontologies.

1.2 Ontologies

The first ontology we discuss is GALEN. Its name stands for General Architecture for Language, Encyclopaedias and Nomenclatures. It is a project that has been funded as part of Framework IV of the EC Health Care Telematics Research Program. It is a paradigmatic ontology presented since it has the main characteristics that motivated the development of the algorithm presented in this thesis. Nowadays, this ontology is distributed by OpenGALEN [4].

Another important ontology is SNOMED CT, an acronym for Systematized Nomenclature of Medicine - Clinical Terms[5]. According to the International Health Terminology Standards Development Organisation[6] it is considered the most comprehensive, multilingual, clinical health care terminology in the world.

1.3 Organization of the thesis

In Chapter 2 we present two challenging medical ontologies, SNOMED CT and GALEN, and other smaller ontologies. These ontologies are the main motivator of this thesis.

In Chapter 3 we present description logics formally. They constitute the theoretical basis of this thesis.

In Chapter 4 we present the normalization and completion rules to classify the ontologies.

In Chapter 5 we present the pseudocode and data structures to implement the rules.

In Chapter 6 we explain how the implementation was done. We present its design, considered alternatives to the current implementation, and experiments showing its performance.

In Chapter 7 we present the conclusion of this thesis and some topics for future work.

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

Chapter 2

Medical Ontologies

In this chapter, we present two medical ontologies that are challenging for the ontology reasoners, GALEN and SNOMED CT, and other smaller ontologies.

2.1 The GALEN ontology

GALEN stands for General Architecture for Language, Encyclopaedias and Nomenclatures in medicine. GALEN is a project funded by the European Union, as part of Framework IV of the EC Healthcare Telematics research program. According to its creators, it is intended to "put the clinical into the clinical workstation". It produces a multilingual coding system for medicine that is different from the one used in the past.

GALEN has been developed to represent clinical information in a new way. The project builds an ontology, the Common Reference Model, to represent medical concepts. Many clinical systems, such as electronic health care records, decision support systems and computer-based multilingual coding systems for medicine, benefit from this ontology.

The GALEN Programme represents the overall development of the technology, which has included several research projects, like Framework III (GALEN project) and Framework IV (GALEN-IN-USE project). In the early stage, the GALEN Programme mainly constructed a concept model language, named GALEN Representation and Integration Language (GRAIL). At the same time, different structures of GALEN Common Reference Model were tested.

The main purpose of GALEN is to achieve the following goals:

1. to create a common medical terminology;
2. to avoid the high costs needed for harmonization of small variations in ontologies;
3. to facilitate clinical applications;
4. to share patient records required for patient care and for general views, like in statistics, management or research;
5. to provide multilingual systems that preserve the meaning and the representation.

To achieve these goals, GALEN brings five fundamental changes in the standard approach.

In the user interface, the user can describe conditions instead of selecting codes. It is possible to have a central concept described by simple forms. When it is necessary, a precise code can be automatically generated later.

In the structure, GALEN uses composite descriptions instead of enumerated codes. GALEN handles terminology making descriptions based on the composition of basic concepts. On the other hand, other systems work as a phrase book, where each possible description has to be sought in a list.

In establishing standards, GALEN uses a standard reference model instead of a standard coding system. There are differences between coding and classifications. These differences arise because they are used for different purposes. Finding a single set of codes usable for all medical terms would be a huge uncoordinated construction. The GALEN Common Reference Model, however, provides means of representing coding and classification in an interrelated fashion. The slogan is: "coherence without uniformity".

In delivery, GALEN uses a dynamic terminology instead of static coding systems. The project CorbaMed is an effort to have a terminology server.

In presentation, GALEN uses multilingual terminologies instead of monolingual terminologies. GALEN has a separation between the concepts and the natural language that presents them.

Nowadays, the GALEN Programme is not funded by the European Union anymore. The members of the GALEN Programme founded a non-profit organization named *OpenGALEN* to expand its results and to find related technologies for the GALEN ontology. This ontology represents a challenge for the reasoners due to the logic expressivity and the use of general concept inclusions.

2.2 The SNOMED CT ontology

SNOMED CT was a joint development between the National Health Service (NHS) [1] in England and the College of American Pathologists (CAP) [2] in the United States. It was formed in 1999, although SNOMED itself was started in 1965 as SNOP (Systematized Nomenclature of Pathology) and then extended to other medical fields. SNOMED brings a way to index, store and retrieve clinical data across different specialties. In addition, it helps organizing the content of medical records. This ontology represents a challenge for the reasoners due to its big size.

2.3 Smaller ontologies

Due to GALEN's characteristics, some simpler ontologies were developed. They are based on GALEN, but simpler in their structure. The ontology NotGALEN is a selective adaptation made in 1995 of an early prototype of the GALEN model, contributed by Ian Horrocks. Its content is not related to or representative of any OpenGALEN release. The ontology CELGalen is a modification of GALEN, but without inverse roles nor functional roles, contributed by Boontawee Suntisrivaraporn.

The Gene Ontology (GO)[3] provides a controlled vocabulary to describe gene and gene product attributes of any organism. Currently, this ontology consists of thousands of concept names and only one transitive role `part-of`.

The (US) National Cancer Institute thesaurus (NCI)[4] provides reference terminology that covers vocabulary for clinical care, translational and basic research, and public information.

Another ontology of interest is the Foundational Model of Anatomy (FMA)[5]. It is a knowledge source for biomedical informatics. It is focused on representation of classes or types and the rela-

[1] `http://www.nhs.uk/`
[2] `http://www.cap.org/`
[3] `http://www.geneontology.org/`
[4] `http://ncit.nci.nih.gov/`
[5] `http://sig.biostr.washington.edu/projects/fm/`

tionships used for the symbolic representation of the phenotypic structure of the human body. It is intended to be understandable to humans, and also navigable, parsable and interpretable by automated systems. Although the FMA is about the human anatomy, its ontological framework can be applied and extended to other species.

2.4 Reasoners

A reasoner is a program that can compute consequences derived from a set of axioms. Reasoners are used to classify ontologies. Using a reasoner, it is possible to decide for two concepts in an ontology, whether one is a subconcept of the other one. With the help of a reasoner, not only is it possible to find errors, like wrong relationships or empty concepts, but also to find implicit relations that are very difficult to find for a human expert. Different reasoners use different techniques to classify ontologies.

Some of the reasoners we mention are the following:

CB[6] (Consequence-based reasoner, University of Oxford) is a reasoner for Horn $\mathcal{SHIF}$ ontologies, i.e. $\mathcal{SHIQ}$ ontologies that can be translated to the Horn fragment of first-order logic. It is implemented in OCaml. It uses a completion-based procedure for $\mathcal{EL}^{++}$ ontologies and works by deriving new consequent axioms [Kazakov(2009)].

CEL[7] (Classifier for EL, Technische Universität Dresden) is a reasoner for $\mathcal{EL}^{+}$. It is implemented in Lisp. It uses a refined polynomial-time algorithm [Baader et al.(2005)Baader, Brandt, and Lutz] [Baader et al.(2008)Baader, Brandt, and Lutz] that can process very large ontologies in reasonable time.

FaCT++[8] (Fast Classification of Terminologies, University of Manchester) is a reasoner for the description logic $\mathcal{SHROIQ}(\mathcal{D})$. It is implemented in C++. It is based on optimized tableaux algorithms [Tsarkov and Horrocks(2006)].

HermiT[9] (University of Oxford) is a resoner for $\mathcal{SHROIQ}(\mathcal{D})$. It is implemented in Java. It is based on a "hypertableau" calculus [Motik et al.(2007)Motik, Shearer, and Horrocks].

Pellet[10] (Clark & Parsia) is a reasoner for $\mathcal{SHROIQ}(\mathcal{D})$. It is implemented in Java.

RacerPro[11] (Renamed ABox and Concept Expression Reasoner, Racer Systems) is a reasoner for the description logic $\mathcal{SHIQ}$. It is implemented in Lisp. It uses a highly optimized tableau calculus for a very expressive description logic [Haarslev and Möller(2001)].

Snorocket[12] (Commonwealth Scientific and Industrial Research Organisation - CSIRO) is a reasoner for $\mathcal{EL}^{+}$. It is implemented in Java. It uses the polynomial-time classification algorithm for $\mathcal{EL}^{+}$, it was optimized for classifying SNOMED CT, and it was licensed to the International Health Terminology Standards Development Organisation (IHTSDO) to maintain and produce SNOMED CT.

2.5 Classification of large ontologies

SNOMED CT has been considered one of the most challenging ontologies due to its big size. Different approaches have been employed to classify this ontology. With more than three hundred thousand

[6] http://cb-reasoner.googlecode.com/
[7] http://cel.googlecode.com/
[8] http://factplusplus.googlecode.com/
[9] http://hermit-reasoner.com/
[10] http://clarkparsia.com/pellet/
[11] http://www.racer-systems.com/
[12] http://research.ict.csiro.au/software/snorocket

concepts, the ontology needs an important amount of memory if it is classified by traditional algorithms.

GALEN is a relatively famous ontology since it is very difficult to classify. Especially the tableaux-based reasoners need to create very large models, due to the fact that large parts of the TBox are cyclically interconnected with each other. This difficult version of GALEN can be classified by a prototypical consequence based reasoner (CB) [Kazakov(2009)].

Many versions based on GALEN have been created. They typically restrict the expressivity to find a fragment that is classifiable. Among them, we will present two: NotGalen and CEL-Galen. The former is a test ontology. The latter is also a test ontology that was developed to test CEL [Baader et al.(2006)Baader, Lutz, and Suntisrivaraporn]. Both were used in the experiments of Chapter 6.

Although SNOMED CT and GALEN are both challenging for ontology reasoners, they have some important structural differences. On the one hand, SNOMED CT is acyclic and bigger than GALEN. On the other hand, GALEN is more compact, requires more expressiveness, and uses general concept inclusions (GCIs).

Chapter 3

Description Logics

In this chapter, we define the description logic $\mathcal{ELHI}f_{\mathcal{R}^+}$. Its name means that this particular description logic has $\mathcal{E}$ (existential restrictions), $\mathcal{H}$ (role hierarchies), $\mathcal{I}$ (inverse roles), f (functional roles), and R^+ (transitive roles). The meaning of these terms, syntax and semantics of $\mathcal{ELHI}f_{\mathcal{R}^+}$ are presented below.

3.1 Syntax of $\mathcal{ELHI}f_{\mathcal{R}^+}$

The sets $\mathsf{N_C}$, $\mathsf{N_R}$ are countable sets of symbols, pairwise disjoint. The elements of $\mathsf{N_C}$ are called *concept names*, the elements of $\mathsf{N_R}$ are called *role names*.

The atomic symbols of a description logic signature can be combined using *concept constructors* and *role constructors*, to construct more complex concept expressions and role expressions respectively. Each description logic is identified by the set of role and concept constructors it has.

Definition 1 (**Concept and roles**). A *role* r' of $\mathcal{ELHI}f_{\mathcal{R}^+}$ is either r or r^-, where $r \in \mathsf{N_R}$. They are called *role name* and *inverse role* respectively.

A *concept* C of $\mathcal{ELHI}f_{\mathcal{R}^+}$ is defined as follows:

$$C ::= A \mid \top \mid \neg C \mid C \sqcap D \mid \exists r'.C$$

where $A \in \mathsf{N_C}$; D is a concept, r' is a role. $\diamond$

Concepts and roles are used to construct *terminological axioms*. These axioms make the TBox, which defines the terminology.

Definition 2 (**TBox**). A *terminological axiom* τ is an expression of the following form:

$$\tau ::= C_1 \sqsubseteq C_2 \mid r'_1 \sqsubseteq r'_2 \mid f(r') \mid r' \circ r' \sqsubseteq r'$$

where C_1 and C_2 are concepts; r', r'_1, r'_2 are roles.

A *TBox* or *terminological box* is a finite set, possibly empty, of terminological axioms.

The expression $C_1 \equiv C_2$ used in a TBox represents an abbreviation for $C_1 \sqsubseteq C_2$ and $C_2 \sqsubseteq C_1$. Analogouosly, the expression $r'_1 \equiv r'_2$ used in a TBox represents an abbreviation for $r'_1 \sqsubseteq r'_2$ and of $r'_2 \sqsubseteq r'_1$. $\diamond$

Notation: In the literature, it is common to find an axiom of the form $\top \sqsubseteq (\leq\ 1\ r')$ to denote functional roles. In this thesis, we use $f(r')$ with the same meaning.

3.2 Semantics of $\mathcal{ELHI}f_{\mathcal{R}^+}$

To define the semantics, we use an interpretation function that maps language symbols to elements and relations of a domain.

Definition 3 (Interpretation). An *interpretation* $\mathcal{I}$ is a tuple $\langle \Delta_\mathcal{I}, \cdot^\mathcal{I} \rangle$, where $\Delta_\mathcal{I}$ is a non-empty set called *domain* and $\cdot^\mathcal{I}$ is an *interpretation function* defined as follows:

$\forall A \in \mathsf{N_C} : A^\mathcal{I} \subseteq \Delta_\mathcal{I}$
$\forall r \in \mathsf{N_R} : r^\mathcal{I} \subseteq \Delta_\mathcal{I} \times \Delta_\mathcal{I}$

The interpretation function is extended to every concept and every role as described in Table 3.1:

$$
\begin{array}{rcl}
\top^\mathcal{I} & := & \Delta_\mathcal{I} \\
(\neg C)^\mathcal{I} & := & \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(C_1 \sqcap C_2)^\mathcal{I} & := & {C_1}^\mathcal{I} \cap {C_2}^\mathcal{I} \\
(\exists r.C)^\mathcal{I} & := & \{d_1 \in \Delta_\mathcal{I} \mid \exists d_2 \in \Delta_\mathcal{I} : ((d_1, d_2) \in r^\mathcal{I} \wedge d_2 \in C^\mathcal{I})\} \\
(r^-)^\mathcal{I} & := & \{(d_1, d_2) \in \Delta_\mathcal{I} \times \Delta_\mathcal{I} \mid (d_2, d_1) \in r^\mathcal{I}\}
\end{array}
$$

where C_1, C_2, C are concepts; r are roles.

Table 3.1: Interpretation function.

$\diamond$

In other words, an interpretation of a description logic is a first-order interpretation, where only unary and binary predicates are allowed and the set of functions is empty.

Definition 4 (Satisfaction of a TBox). For an interpretation $\mathcal{I}$ and a terminological axiom ψ, the relation $\mathcal{I} \models \psi$ is defined in Table 3.2:

$$
\begin{array}{lll}
\mathcal{I} \models (C \sqsubseteq D) & \text{iff} & C^\mathcal{I} \subseteq D^\mathcal{I} \\
\mathcal{I} \models (r \sqsubseteq s) & \text{iff} & r^\mathcal{I} \subseteq s^\mathcal{I} \\
\mathcal{I} \models f(r) & \text{iff} & \forall d_1 \in \Delta_\mathcal{I} : \mid \{d_2 \in \Delta_\mathcal{I} \mid (d_1, d_2) \in r^\mathcal{I}\} \mid \leq 1 \\
\mathcal{I} \models (r \circ r \sqsubseteq r) & \text{iff} & r^\mathcal{I} \circ r^\mathcal{I} \subseteq r^\mathcal{I}
\end{array}
$$

where C, D are concepts; r, s are roles.

Table 3.2: Satisfaction.

Let $\mathcal{T}$ be a TBox, $\mathcal{I}$ an interpretation, C, C_1, C_2 concepts, ψ a terminological axiom. Then:

- We say that $\mathcal{I}$ is a *model* of $\mathcal{T}$, denoted $\mathcal{I} \models \mathcal{T}$, if and only if $\forall \psi \in \mathcal{T} : \mathcal{I} \models \psi$.
- We say that $\mathcal{T}$ *entails* ψ, denoted $\mathcal{T} \models \psi$, if and only if for every interpretation $\mathcal{I}$: if $\mathcal{I} \models \mathcal{T}$, then $\mathcal{I} \models \psi$.

- We say that C_1 is *subsumed* by (or *included* in) C_2 with respect to $\mathcal{T}$, denoted $C_1 \sqsubseteq_{\mathcal{T}} C_2$, if and only if $\mathcal{T} \models C_1 \sqsubseteq C_2$. Thus, C_2 is a *subsumer* of C_1 and C_1 is a *subsumee* of C_2.
- We say that a concept name C_2 is a *direct subsumer* of a concept name C_1 with respect to $\mathcal{T}$ if and only if:
  - $C_1 \sqsubseteq_{\mathcal{T}} C_2$
  - C_2 is not C_1
  - there is no concept name C, different from C_1 and from C_2, such that $C_1 \sqsubseteq_{\mathcal{T}} C$ and $C \sqsubseteq_{\mathcal{T}} C_2$
- We say that r_1 is *included* in r_2 with respect to $\mathcal{T}$, denoted $r_1 \sqsubseteq_{\mathcal{T}} r_2$, if and only if $\mathcal{T} \models r_1 \sqsubseteq r_2$.
- We say that C is *satisfiable* with respect to $\mathcal{T}$ if and only if there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$.

$\diamond$

3.3 Example of $\mathcal{ELHI}f_{\mathcal{R}^+}$

In Table 3.3 we can see a simple example of use of concept and role constructors.

1.	$\mathsf{Mother} \sqsubseteq \mathsf{Woman} \sqcap \mathsf{Parent}$
2.	$\mathsf{Mother} \sqsubseteq \exists \mathsf{isMotherOf}.\mathsf{Human}$
3.	$\mathsf{hasMother} \sqsubseteq \mathsf{hasAncestor}$
4.	$\mathsf{isMotherOf} \equiv \mathsf{hasMother}^{-}$
5.	$\mathsf{hasAncestor} \circ \mathsf{hasAncestor} \sqsubseteq \mathsf{hasAncestor}$
6.	$f(\mathsf{hasMother})$

Table 3.3: Example of $\mathcal{ELHI}f_{\mathcal{R}^+}$.

The intended meaning of this example is as follows:

1. A mother is a woman and a parent.
2. A mother is a mother of some human being.
3. If x has a mother y, then x has an ancestor y.
4. If x is mother of y, then y has x as mother.
5. An ancestor of an ancestor is also an ancestor.
6. If someone has a mother, has at most one mother.

3.4 Less expressive logics

Some of the ontologies presented in Chapter 2 use a less expressive logic than $\mathcal{ELHI}f_{\mathcal{R}^+}$. As explained above, the name of each of these description logics derives from the concept and role constructors it has. Other description logics like $\mathcal{EL}$, $\mathcal{ELH}$, and $\mathcal{ELI}$, have less constructors. Hence, every econcept in $\mathcal{EL}$, $\mathcal{ELH}$ or $\mathcal{ELI}$ is also a concept in $\mathcal{ELHI}f_{\mathcal{R}^+}$. Thus, a TBox which uses concepts for $\mathcal{EL}$, $\mathcal{ELH}$ or $\mathcal{ELI}$ is also a TBox in $\mathcal{ELHI}f_{\mathcal{R}^+}$ and can be classified by the algorithm presented in this thesis.

3.5 Semantic Web terminology

Semantic Web tools use a terminology that is different from the terminology we defined above. These terms follow the OWL 2 standard and can be easily related using Table 3.4.

DL syntax	Name	OWL 2 functional style syntax
$\top$	top	Thing
A	concept name	Class
r	role name	ObjectProperty
C	concept	ClassExpression
r'	role	ObjectPropertyExpression
r^-	inverse role	InverseObjectProperty
$C_1 \sqcap \ldots \sqcap C_n$	conjunction	ObjectIntersectionOf($C_1 \ldots C_n$)
$\exists r.C$	existential restriction	ObjectSomeValuesFrom($r\ C$)
$C_1 \sqsubseteq C_2$	concept inclusion	SubClassOf($C_1\ C_2$)
$C_1 \equiv C_2$	equivalence	EquivalentClasses($C_1\ C_2$)
$r_1 \sqsubseteq r_2$	role inclusion	SubObjectPropertyOf($r_1\ r_2$)
$r_1 \equiv r_2$	role equivalence	EquivalentObjectProperties($r_1\ r_2$)
$r \circ r \sqsubseteq r$	transitive role	TransitiveObjectProperty(r)
$f(r)$	functional role	FunctionalObjectProperty(r)
$f(r^-)$	inverse functional role	InverseFunctionalObjectProperty(r)
$r_1^- \equiv r_2$	inverse role definition	InverseObjectProperties($r_1\ r_2$)

Table 3.4: Terminology defined in the OWL 2 profiles.

Chapter 4

Normalization and Completion Rules

In this chapter, we discuss the rules needed for the algorithm. These rules do not actually constitute an algorithm by themselves since they need a way to be computed. Executing the presented set of rules with a naive approach can take an immense amount of time.

The normalization replaces some of the axioms in the TBox by simpler axioms. After getting the normalized TBox, the TBox is saturated with axioms that can be deduced. For example, if we have $r \sqsubseteq s$ and $s \sqsubseteq t$, one saturation rule adds $r \sqsubseteq t$.

The normalization and completion rules are themselves based on two main sources. The normalization rules are taken from [Suntisrivaraporn(2009)]. The completion rules are based on the completion rules in [Vu(2008)], which need the normalized form of [Suntisrivaraporn(2009)]. It is proved that the completion rules in [Vu(2008)] are correct, meaning soundness, completion, and termination. Since the completion rules presented in this thesis are not exactly the same, we prove the correctness of these rules as well.

4.1 Normal form

In order to apply the rules, it is first necessary to bring the TBox to the conditions of application of the rules. This transformation is called *normalization*, and produces a *normalized TBox*. The normalized form uses only a reduced number of axioms which are described below.

Once the TBox is normalized, the axioms will be as presented in Table 4.1.

4.2 Normalization rules

As said above, the normalization is performed before any other rule is applied. The normalization rules preserve the same models when applied to ontologies. The repetitive application of these rules on a TBox produces the normalized TBox.

The rules in Table 4.2 are taken from [Suntisrivaraporn(2009)]. The $\rightsquigarrow$ means a replacement of the axiom on the left-hand side by the axiom or axioms on the right-hand side. A *fresh* concept name and a *fresh* role name are symbols that are not used in the TBox before the rule is applied.

Lemma 5. *Let $\mathcal{T}$ be an $\mathcal{ELHI}f_{\mathcal{R}^+}$ TBox and $\mathcal{T}'$ after the application of the normalization rules of Table 4.2. Let A, B be concept names occurring in $\mathcal{T}$. Then $A \sqsubseteq_{\mathcal{T}} B$ if and only if $A \sqsubseteq_{\mathcal{T}'} B$.*

Proof. The idea of the proof is to show that the lemma holds for every normalization rule applied. To simplify the individual proofs of each rule, we first show the proofs for NR-7 and NR-8.

GCI-0	$A \sqsubseteq B$
GCI-1	$A_1 \sqcap \cdots \sqcap A_n \sqsubseteq B$
GCI-2	$A \sqsubseteq \exists r.B$
GCI-3	$\exists r.A \sqsubseteq B$
RI-1	$f(r)$
RI-2	$r \sqsubseteq s$
RI-3	$r \circ r \sqsubseteq r$
RI-4	$r^- = s$

where $r, s \in \mathsf{N_R}$;
$A_i, A, B \in \mathsf{N_C}$.

Table 4.1: Normalized axioms.

NR-1	$C \equiv D$	$\rightsquigarrow$	$C \sqsubseteq D, D \sqsubseteq C$
NR-2	$C_1 \sqcap \cdots \sqcap \hat{C} \sqcap \cdots \sqcap C_n \sqsubseteq D$	$\rightsquigarrow$	$\hat{C} \sqsubseteq A, C_1 \sqcap \cdots \sqcap A \sqcap \cdots \sqcap C_n \sqsubseteq D$
NR-3	$\exists r'.\hat{C} \sqsubseteq D$	$\rightsquigarrow$	$\hat{C} \sqsubseteq A, \exists r'.A \sqsubseteq D$
NR-4	$\hat{C} \sqsubseteq \exists r'.D$	$\rightsquigarrow$	$\hat{C} \sqsubseteq A, A \sqsubseteq \exists r'.D$
NR-5	$B \sqsubseteq \exists r'.\hat{C}$	$\rightsquigarrow$	$B \sqsubseteq \exists r'.A, A \sqsubseteq \hat{C}$
NR-6	$D \sqsubseteq C_1 \sqcap C_2$	$\rightsquigarrow$	$D \sqsubseteq C_1, D \sqsubseteq C_2$
NR-7	$C \sqsubseteq \exists r^-.D$	$\rightsquigarrow$	$C \sqsubseteq \exists u.D$, $u \sqsubseteq r^-$, $r^- \sqsubseteq u$
NR-8	$\exists r^-.C \sqsubseteq D$	$\rightsquigarrow$	$\exists u.C \sqsubseteq D$, $u \sqsubseteq r^-$, $r^- \sqsubseteq u$

where
r is a role name;
r' is a role;
C, C_i, D are arbitrary concept descriptions;
$\hat{C}$, $\hat{D}$ are complex concept descriptions;
B is a concept name;
A is a *fresh* concept name;
u is a *fresh* role name.

Table 4.2: Normalization rules.

For NR-7 and NR-8, let u be a fresh role name. Replacing in the original axiom any occurrence of r^- by u, and adding $u \sqsubseteq r'^-$, $r'^- \sqsubseteq u$ to $\mathcal{T}$, produces that any model of the axiom before the substitution is equivalent to the model after the substitution.

For NR-1 and NR-2 the rules are identical to the rules in [Suntisrivaraporn(2009)], where this is already proved.

For NR-3 and NR-5, if we assume that r' is a role name, the proof is shown in [Suntisrivaraporn(2009)]. If r' is an inverse role, we can reduce the formulae to the case of NR-7 and NR-8.

For NR-4 and NR-6 it is sufficient to verify that these rules are variants of the rules shown in Table 4.3. Let us consider the case where these rules differ. This is $\hat{C} \sqsubseteq C_1 \sqcap \ldots \sqcap C_n$, where $\hat{C}$ is

a complex concept description and $C_1 \dots C_n$ are concept names of existential restrictions. The successive application of NR-6 reduces the number of conjuncts on the right-hand side of the inclusion, placing the axioms in the condition of NR-4.

NR-3-1	$\hat{C} \sqsubseteq \hat{D}$	$\rightsquigarrow$	$\hat{C} \sqsubseteq A, A \sqsubseteq \hat{D}$
NR-3-3	$B \sqsubseteq C \sqcap C$	$\rightsquigarrow$	$B \sqsubseteq C, B \sqsubseteq D$

Table 4.3: Some of the original normalization rules.

□

The rules perform a reduction in the axiom complexity by creating more axioms possibly using new auxiliary concepts and roles.

4.3 Normalized TBox

Once the TBox is normalized, it has to be saturated. Unlike the normalization rules, the saturation rules do not remove axioms. After ensuring that $r \sqsubseteq r$ for each role name r, the saturation rules in Table 4.4 are applied. The $\rightsquigarrow$ means an addition of the axioms on the right-hand side when the axioms on the left-hand side are found.

SR-1	$r \sqsubseteq s$	$\rightsquigarrow$	$r^- \sqsubseteq s^-$
SR-2	$r \sqsubseteq s$, $s \sqsubseteq t$	$\rightsquigarrow$	$r \sqsubseteq t$
SR-3	$r \circ r \sqsubseteq r$	$\rightsquigarrow$	$r^- \circ r^- \sqsubseteq r^-$
SR-4	$r \sqsubseteq s$, $f(s)$	$\rightsquigarrow$	$f(r)$

where
r, s, t are role names or inverse roles.

Table 4.4: Saturation rules.

To prove the correctness of the rules in Table 4.4, we prove Lemma 6.

Lemma 6. *If r, s, t are roles and $\mathcal{I}$ is an interpretation, then the following holds:*

1. $\mathcal{I} \models r \sqsubseteq s$ *if and only if* $\mathcal{I} \models r^- \sqsubseteq s^-$
2. *if* $\mathcal{I} \models r \sqsubseteq s$ *and* $\mathcal{I} \models s \sqsubseteq t$, *then* $\mathcal{I} \models r \sqsubseteq t$
3. $\mathcal{I} \models r \circ r \sqsubseteq r$ *if and only if* $\mathcal{I} \models r^- \circ r^- \sqsubseteq r^-$
4. *if* $\mathcal{I} \models r \sqsubseteq s$ *and* $\mathcal{I} \models f(s)$, *then* $\mathcal{I} \models f(r)$

Proof.

1. $\mathcal{I} \models r^- \sqsubseteq s^-$
   if and only if by definition $(r^-)^{\mathcal{I}} \subseteq (s^-)^{\mathcal{I}}$
   if and only if by definition $\{(y,x) \mid (x,y) \in r^{\mathcal{I}}\} \subseteq \{(y,x) \mid (x,y) \in s^{\mathcal{I}}\}$
   if and only if by properties of sets $\forall x \forall y : (x,y) \in r^{\mathcal{I}} \rightarrow (x,y) \in s^{\mathcal{I}}$
   if and only if by properties of sets and definition $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
   if and only if by properties of sets and definition $\mathcal{I} \models r \sqsubseteq s$

2. $\mathcal{I} \models r \sqsubseteq s$ and $\mathcal{I} \models s \sqsubseteq t$
   if and only if by definition $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ and $s^{\mathcal{I}} \subseteq t^{\mathcal{I}}$
   then by transitivity of the subset relation $r^{\mathcal{I}} \subseteq t^{\mathcal{I}}$
   if and only if by definition $\mathcal{I} \models r \sqsubseteq t$

3. $\mathcal{I} \models r^- \circ r^- \sqsubseteq r^-$
   if and only if by definition $(r^-)^{\mathcal{I}} \circ (r^-)^{\mathcal{I}} \subseteq (r^-)^{\mathcal{I}}$
   if and only if by definition $\{(z,y) \mid (y,z) \in r^{\mathcal{I}}\} \circ \{(y,x) \mid (x,y) \in r^{\mathcal{I}}\} \subseteq \{(z,x) \mid (x,z) \in r^{\mathcal{I}}\}$
   if and only if by properties of sets $\forall x \forall y \forall z : (y,z) \in r^{\mathcal{I}} \wedge (x,y) \in r^{\mathcal{I}} \rightarrow (x,z) \in r^{\mathcal{I}}$
   if and only if by properties of sets $\{(x,y) \mid (x,y) \in r^{\mathcal{I}}\} \circ \{(y,z) \mid (y,z) \in r^{\mathcal{I}}\} \subseteq \{(x,z) \mid (x,z) \in r^{\mathcal{I}}\}$
   if and only if $\mathcal{I} \models r \circ r \sqsubseteq r$

4. $\mathcal{I} \models r \sqsubseteq s$ and $\mathcal{I} \models f(s)$
   if and only if by definition $r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$ and $\forall d \in \Delta_{\mathcal{I}} :\mid \{(d,e) \in s^{\mathcal{I}}\} \mid \leq 1$
   then $\{(d,e) \in r^{\mathcal{I}}\} \subseteq \{(d,e) \in s^{\mathcal{I}}\}$ and $\forall d \in \Delta_{\mathcal{I}} : \mid \{(d,e) \in s^{\mathcal{I}}\} \mid \leq 1$
   then $\forall d \in \Delta_{\mathcal{I}} : \mid \{(d,e) \in r^{\mathcal{I}}\} \mid \leq 1$
   if and only if by definition $\mathcal{I} \models f(r)$

□

After applying the saturation rules exhaustively, some role names are equivalent to the inverse of other role names. The following step is to add fresh auxiliary role names that represent the inverse roles for those roles not having them.

The last step in this normalization process is to choose for every role r a unique role s such that $r^- \sqsubseteq s \in \mathcal{T}$, $s^- \sqsubseteq r \in \mathcal{T}$, to add two axioms to $\mathcal{T}$: $r^- = s$ and $s^- = r$, and to replace any other occurrence of r^- by s and any other occurrence of s^- by r.

In the following, the sets $\mathsf{N_R}$ and $\mathsf{N_C}$ will refer respectively to the sets of role names and concept names after the normalization process. We will use the notation r^- to refer to the role s such that $r^- = s \in \mathcal{T}$. In addition, we have the following property: $\mathcal{T} \models r \sqsubseteq s$ if and only if $r \sqsubseteq s \in \mathcal{T}$, where r, s are role names.

Finally, we are in condition to prove Theorem 7.

Theorem 7. *The normalization process is sound and complete, and terminates in polynomial time.*

Proof. Soundness and completeness are direct consequences of Lemma 5.

For showing termination in polynomial time, without loss of generality we assume that NR-7 and NR-8 have been exhaustively applied. This can be applied at most once for each axiom. After this, the TBox is in the conditions of [Suntisrivaraporn(2009)], which is proved to be terminating in polynomial time. □

4.4 Completion rules

Once the TBox is normalized and saturated, the following phase is the application of the completion rules. The completion rules are based on [Vu(2008)]. We first present the sets that remain constant. [1]

- $\Xi := \{\exists r.A \mid r \in \mathsf{N_R}, A \in \mathsf{N_C}\}$
- $\Omega := \{(A, \psi) \mid A \in \mathsf{N_C}, \psi \subseteq \Xi\}$

Based on the sets above, we define the sets that will be modified during the successive application of the completion rules:

- $V \subseteq \Omega$
- $S \subseteq \{(x, A) \mid x \in \Omega, A \in \mathsf{N_C}\}$
- $R \subseteq \{(r, x, y) \mid r \in \mathsf{N_R}, x, y \in \Omega\}$

The elements in set S are called *S-entries*, the elements in set R are called *R-entries*, the elements in V are referred as *nodes*.

By construction, Ξ and Ω are finite sets, and therefore V, S, and R are bounded. This observation is relevant, since the completion rules saturate V, S, and R, and the fact that they are bounded is used to show termination.

The completion process should satisfy the following invariants:

- if $((A, \varphi), C) \in S$, then $(A \sqcap \prod_{E \in \varphi} E) \sqsubseteq_{\mathcal{T}} C$
- if $(r, (A, \varphi), (B, \psi)) \in R$, then $(A \sqcap \prod_{E \in \varphi} E) \sqsubseteq_{\mathcal{T}} \exists r.(B \sqcap \prod_{E \in \psi} E)$

  where each E is of the form $\exists r.X$.

  In addition, we have that $A \sqsubseteq_{\mathcal{T}} B$ if and only if $((A, \emptyset), B) \in S$.

4.4.1 Start conditions

The algorithm starts with the following conditions:

- $S := \{((A, \emptyset), A) \mid A \in \mathsf{N_C}\} \cup \{((A, \emptyset), \top) \mid A \in \mathsf{N_C}\}$
- $R := \emptyset$

4.4.2 Original completion rules

In Table 4.5, we present the original completion rules given by [Vu(2008)]. They preserve the same numbering, although here the rules have an "O" as prefix, to distinguish them from those used in this thesis. There is no OCR-3 since it was removed in the original. There is a small change in the notation, like $A \in S(x)$ is denoted by $(x, A) \in S$ and $(x, r, y) \in E$ is denoted by $(r, x, y) \in R$, and variable name substitutions.[2]

As an abbreviation, $f(r)$ means that there exists an axiom in $\mathcal{T}$ stating that r is functional.

[1] There is a small difference in set Ξ with respect to [Vu(2008)], since here we only use role names and not inverse role names. This is not a problem, since every role name has an assigned inverse role name.

[2] Rule OCR-6 had an omission, Rule OCFR-1 had a typo and an omission, and Rule OCFR-3 had an extra term in [Vu(2008)]. All these issues have been corrected.

4.4.3 Optimized completion rules

The optimized completion rules are presented in Table 4.6. The underlined elements are membership checks for S and R. These conditions have special relevance in Chapter 5 where the algorithm is discussed. In the algorithm these conditions are considered the triggers to activate each rule. Since every applied rule modifies S or R, this automatically triggers the following rules.

We say that a completion rule is *sensitive* to changes in a set, when a rule checks in that set in the condition of application. For example, in Table 4.6, CR-1 is sensitive to changes in S only, CR-7 is sensitive to changes in R only, and CR-4 is sensitive to changes in S and R.

As a consequence of the normalized form presented in this thesis, CR-2 allows having several conjuncts on the left-hand side of the inclusion (GCI-1). This simple optimization reduces the number of auxiliary symbols.

A small change is that OCFR-1 has been numbered as rule CR-3, and the variable matchings for $y = (B, \emptyset)$ and $z = (\top, \{\exists r^-.A\})$ have been replaced by assignments. In CR-9, a condition of $y \neq z$ has been added.

In Lemma 8 we prove that the rules in Table 4.6 are equivalent to the rules in Table 4.5.

Lemma 8. *The completion rules in Table 4.6 produce the same results as the completion rules in Table 4.5.*

Proof. This is proved showing that for each rule in Table 4.5 there is exactly one rule in Table 4.6 producing the same result, and there are no more rules. The rules OCR-n are mapped to CR-n. OCFR-1 is mapped to CR-3, OCFR-2 is mapped to CR-8, and OCFR-3 is mapped to CR-9. There are changes factorizing the **if** in CR-6, CR-7 and CR-9:

if A **then** $B; D$ **else** $C; D$	is rewritten as	**if** A **then** B **else** C D

CR-2 follows the normalization conditions proved in Theorem 7. CR-3 forces the instantiation for y and for v (z in OCR-3) instead of testing possible values. CR-9 checks for $y \neq z$ to avoid the unnecessary execution when $y = z$.

□

The intuition behind the set of completion rules shown in Table 4.6 is explained below.

4.4.4 Observations

The notation r^- occurring in the GCIs in the rules is an abbreviation, since the elements in $\mathcal{T}$ do not have inverse roles in the GCIs. For example, $\exists r^-.A \sqsubseteq B \in \mathcal{T}$ is in fact a notation to say $\exists r'.A \sqsubseteq B \in \mathcal{T}, r^- = r' \in \mathcal{T}$.

As mentioned above, the rules have underlined terms according to what changes in S and R can be relevant to the conditions of application.

Although CR-2 is affected by many changes in S, only one case is considered. This is due to the fact that the other cases are symmetric, and the rule is triggered anyway.

CR-5 combines transitive roles with role hierarchies. Both $(r_1, x, y) \in R$ and $(r_2, y, z) \in R$ need to be considered, since both changes to R are not symmetric.

CR-6 "branches" an edge (r, x, y) to include a new edge (r, x, v) if this edge is new in the graph. CR-7 is very similar to CR-6, but combines the use of transitive roles and role hierarchies.

CR-8 is similar to CR-4, although the existential restriction is on right-hand side of the inclusion, using an inverse role. Role s has to be functional, and therefore r_1 and r_2.

CR-9 is given to support the combination of role hierarchies and functional roles. Although two possible changes to R can trigger the rule, only one is considered since the other one is symmetric.

The completion rules do not define any order of application. This is discussed in Chapter 5.

OCR-1 **if** $A \sqsubseteq B \in \mathcal{T}$, $(x, A) \in S$
then $S := S \cup \{(x, B)\}$

OCR-2 **if** $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{T}$, $(x, A_1) \in S$, $(x, A_2) \in S$
then $S := S \cup \{(x, B)\}$

OCR-4 **if** $\exists s.B \sqsubseteq A \in \mathcal{T}$, $(r, x, y) \in R$, $(y, B) \in S$, $r \sqsubseteq_{\mathcal{T}} s$
then $S := S \cup \{(x, A)\}$

OCR-5 **if** $s \circ s \sqsubseteq s \in \mathcal{T}$, $(r_1, x, y) \in R$, $(r_2, y, z) \in R$, $r_1 \sqsubseteq_{\mathcal{T}} s$, $r_2 \sqsubseteq_{\mathcal{T}} s$
then $R := R \cup \{(s, x, z)\}$

OCR-6 **if** $\exists s^-.A \sqsubseteq B \in \mathcal{T}$, $r \sqsubseteq_{\mathcal{T}} s$, $(r, x, y) \in R$, $(x, A) \in S$, $(y, B) \notin S$, $y = (B', \psi)$
then $v := (B', \psi \cup \{\exists r^-.A\})$
if $v \notin V$ **then** $V := V \cup \{v\}$, $S := S \cup \{(v, k) \mid (y, k) \in S\} \cup \{(v, B)\}$
else $S := S \cup \{(v, B)\}$
$R := R \cup \{(r, x, v)\}$

OCR-7 **if** $\exists s^-.A \sqsubseteq B \in \mathcal{T}$, $(r_2, x, y) \in R$, $x = (A', \varphi)$, $y = (B', \psi)$,
$r \circ r \sqsubseteq r \in \mathcal{T}$, $r_1 \sqsubseteq_{\mathcal{T}} r$, $r_2 \sqsubseteq_{\mathcal{T}} r$, $\exists r_1^-.A \in \varphi$, $r \sqsubseteq_{\mathcal{T}} s$
then $v := (B', \psi \cup \{\exists r^-.A\})$
if $v \notin V$ **then** $V := V \cup \{v\}$, $S := S \cup \{(v, k) \mid (y, k) \in S\} \cup \{(v, B)\}$
else $S := S \cup \{(v, B)\}$
$R := R \cup \{(r_2, x, v)\}$

OCFR-1 **if** $A \sqsubseteq \exists r.B \in \mathcal{T}$, $(x, A) \in S$, $y = (B, \emptyset)$, $z = (\top, \{\exists r^-.A\})$
then if $f(r)$
then $R := R \cup \{(r, x, z)\}$, $S := S \cup \{(z, B)\} \cup \{(z, \top)\}$
else $R := R \cup \{(r, x, y)\}$

OCFR-2 **if** $B \sqsubseteq \exists r_2^-.A \in \mathcal{T}$, $(r_1, x, y) \in R$, $(y, B) \in S$,
$r_1 \sqsubseteq_{\mathcal{T}} s$, $r_2 \sqsubseteq_{\mathcal{T}} s$, $f(s^-)$
then $S := S \cup \{(x, A)\}$

OCFR-3 **if** $(r_1, x, y) \in R$, $(r_2, x, z) \in R$, $r_1 \sqsubseteq_{\mathcal{T}} s$,
$r_2 \sqsubseteq_{\mathcal{T}} s$, $y = (\top, \psi)$, $z = (\top, \varphi)$, $f(s)$
then $v := (\top, \psi \cup \varphi)$
if $v \notin V$ **then** $V := V \cup \{v\}$, $S := S \cup \{(v, k) \mid (y, k) \in S\} \cup \{(v, k) \mid (z, k) \in S\}$
else $S := S \cup \{(v, k) \mid (y, k) \in S\} \cup \{(v, k) \mid (z, k) \in S\}$
$R := R \cup \{(r_1, x, v)\}$

Table 4.5: Original completion rules for $\mathcal{ELHI}f_{\mathcal{R}^+}$ in [Vu(2008)].

CR-1 **if** $A \sqsubseteq B \in \mathcal{T}$, $\underline{(x, A) \in S}$
then $S := S \cup \{(x, B)\}$

CR-2 **if** $A_1 \sqcap \ldots \sqcap A_i \sqcap \ldots \sqcap A_n \sqsubseteq B \in \mathcal{T}$,
$(x, A_1) \in S, \ldots, \underline{(x, A_i) \in S}, \ldots, (x, A_n) \in S$
then $S := S \cup \{(x, B)\}$

CR-3 **if** $A \sqsubseteq \exists r.B \in \mathcal{T}$, $\underline{(x, A) \in S}$
then if $f(r)$
then $v := (\top, \{\exists r^-.A\})$
if $v \notin V$ **then** $V := V \cup \{v\}$
$S := S \cup \{(v, B)\} \cup \{(v, \top)\}$
$R := R \cup \{(r, x, v)\}$
else $y := (B, \emptyset)$
$R := R \cup \{(r, x, y)\}$

CR-4 **if** $\exists s.A \sqsubseteq B \in \mathcal{T}$, $\underline{(r, x, y) \in R}$, $\underline{(y, A) \in S}$, $r \sqsubseteq_{\mathcal{T}} s$
then $S := S \cup \{(x, B)\}$

CR-5 **if** $s \circ s \sqsubseteq s \in \mathcal{T}$, $\underline{(r_1, x, y) \in R}$, $\underline{(r_2, y, z) \in R}$, $r_1 \sqsubseteq_{\mathcal{T}} s$, $r_2 \sqsubseteq_{\mathcal{T}} s$
then $R := R \cup \{(s, x, z)\}$

CR-6 **if** $\exists s^-.A \sqsubseteq B \in \mathcal{T}$, $r \sqsubseteq_{\mathcal{T}} s$, $\underline{(r, x, y) \in R}$, $\underline{(x, A) \in S}$, $(y, B) \notin S$, $y = (B', \psi)$
then $v := (B', \psi \cup \{\exists r^-.A\})$
if $v \notin V$ **then** $V := V \cup \{v\}$, $S := S \cup \{(v, k) \mid (y, k) \in S\}$
$S := S \cup \{(v, B)\}$
$R := R \cup \{(r, x, v)\}$

CR-7 **if** $\exists s^-.A \sqsubseteq B \in \mathcal{T}$, $\underline{(r_2, x, y) \in R}$, $x = (A', \varphi)$, $y = (B', \psi)$,
$r \circ r \sqsubseteq r \in \mathcal{T}$, $r_1 \sqsubseteq_{\mathcal{T}} r$, $r_2 \sqsubseteq_{\mathcal{T}} r$, $\exists r_1^-.A \in \varphi$, $r \sqsubseteq_{\mathcal{T}} s$
then $v := (B', \psi \cup \{\exists r^-.A\})$
if $v \notin V$ **then** $V := V \cup \{v\}$, $S := S \cup \{(v, k) \mid (y, k) \in S\}$
$S := S \cup \{(v, B)\}$
$R := R \cup \{(r_2, x, v)\}$

CR-8 **if** $A \sqsubseteq \exists r_2^-.B \in \mathcal{T}$, $\underline{(r_1, x, y) \in R}$, $\underline{(y, A) \in S}$,
$r_1 \sqsubseteq_{\mathcal{T}} s$, $r_2 \sqsubseteq_{\mathcal{T}} s$, $f(s^-)$
then $S := S \cup \{(x, B)\}$

CR-9 **if** $\underline{(r_1, x, y) \in R}$, $(r_2, x, z) \in R$, $r_1 \sqsubseteq_{\mathcal{T}} s$,
$\underline{r_2 \sqsubseteq_{\mathcal{T}} s}$, $y = (\top, \psi)$, $z = (\top, \varphi)$, $y \neq z$, $f(s)$
then $v := (\top, \psi \cup \varphi)$
if $v \notin V$ **then** $V := V \cup \{v\}$
$S := S \cup \{(v, k) \mid (y, k) \in S\} \cup \{(v, k) \mid (z, k) \in S\}$
$R := R \cup \{(r_1, x, v)\}$

Table 4.6: Optimized completion rules for $\mathcal{ELHI}f_{\mathcal{R}^+}$.

Chapter 5

Algorithm

In this chapter, we describe an algorithm for deciding standard description logic inference problems. Since satisfiability can be reduced to subsumption and subsumption is itself a sub-problem of classification, it suffices to have a classification algorithm. In Chapter 4, it is presented a set of rules that can classify an $\mathcal{ELHI}f_{\mathcal{R}^+}$ TBox. This means that, given a TBox, the subsumption relations between all pairs of concept names become explicit.

An algorithm that classifies the TBox applying these rules could be expensive in time if it is performed by a systematic search. It is also expensive in space if it uses the structures directly as they are described. These two considerations motivate creating an algorithm where these aspects are explicitly managed. We obtained this algorithm by taking a *change propagation* approach.

5.1 General description

The input of the algorithm is a normalized TBox containing axioms as described in Table 4.1. The output is a set S, called *set of subsumptions*, such that for each pair of concept names A, B in $\mathcal{T}$: $(A, B) \in S$ if and only if $\mathcal{T} \models A \sqsubseteq B$.

The algorithm itself has different components. In Figure 5.1 we can observe how these components are interconnected. S and R correspond to the sets described in Chapter 4. We can also see the completion rules, named as CR, which are grouped in two "chains". The "processor", the "duplicates checker" and the "start", are just three different parts of one single unit that processes the entries. The arrows indicate how the data flows, i.e. how S-entries and R-entries and sent from one component to the other one. A dashed line indicates that the duplicates checker uses S or R before sending an entry to the following component.

There are components of two kinds: those which process S-entries (having an S between parentheses), and those which process R-entries (having an R between parentheses). Every rule in any of both groups takes an entry and after applying the operations returns a list of new entries. These entries can be either S-entries or R-entries. They are suggestions of changes in S (for the S-entries) or in R (for the R-entries).

A chain as described in the diagram is a chain of rules. Every entry is applied to all the rules. The result is the union of the sets resulting of the application of the individual rules. The resulting set contains S-entries and R-entries to update set S and set R. If these entries have never been applied, which is checked by the "duplicates checker", they are added to set Q to be taken by the processor, otherwise they are ignored. Thus, Q is a set of proposed changes.

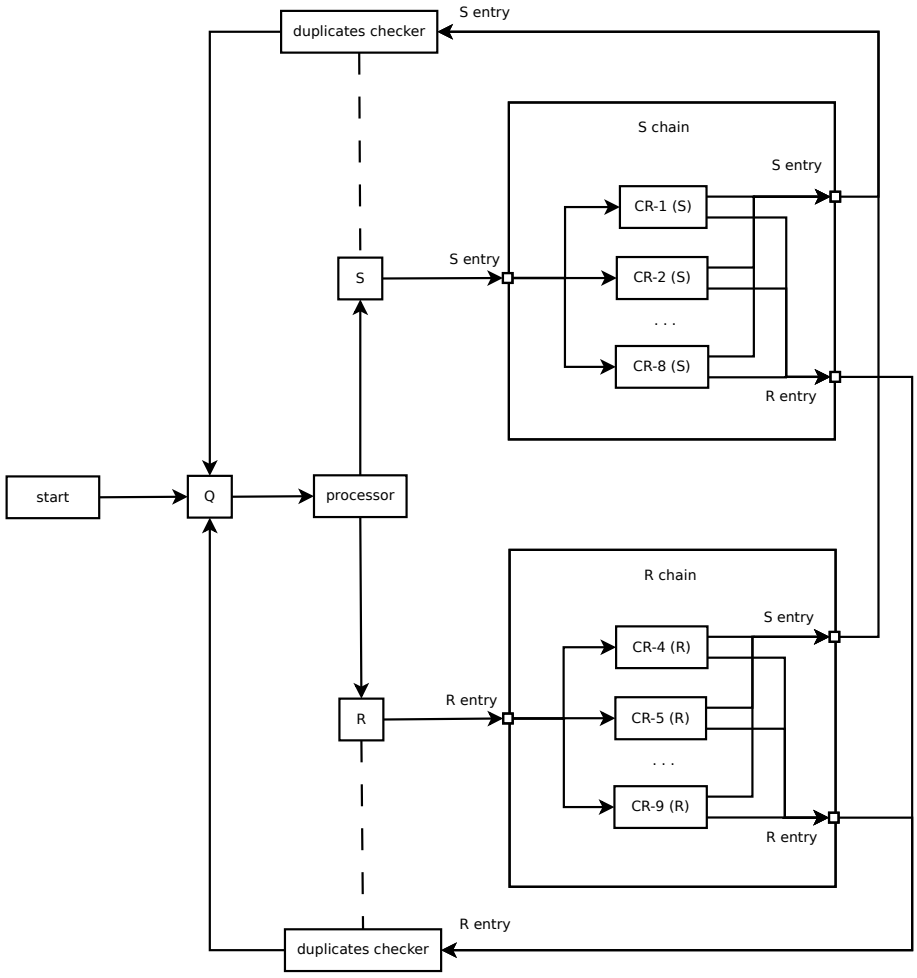

Figure 5.1: Diagram showing the dynamics of the components.

The processor takes entries from Q, changes set S and R, and informs of these changes to the corresponding chain of rules. This procedure is repeated until Q is empty.

The algorithm start condition is slightly modified to include Q as follows:

- $S := \emptyset$
- $R := \emptyset$
- $Q := \{((A, \emptyset), A) \mid A \in \mathsf{N_C}\} \cup \{((A, \emptyset), \top) \mid A \in \mathsf{N_C}\}$

In order to get the appropriate axioms, the TBox, seen as a set of axioms, is stored in multiple maps. Each map has a key and a value, where the value is an axiom. These maps are accessed by functions. For example, the functions $\mathsf{getGCI0}(A)$, $\mathsf{getGCI1}(A)$, and $\mathsf{getGCI2}(A)$ return all the GCIs of type GCI-0, GCI-1 and GCI-2 respectively, where A occurs on the left-hand side. These types are defined in Table 4.1. A full list of the ontology functions is shown in Table 5.1.

The completion rules feed Q with new instances to apply the next rule. Since trying all possibilities is extremely costly, these rules are triggered whenever a relevant structure is changed. This property is

$\mathsf{getGCI0}(A)$	:=	$\{k \in \mathcal{T} \mid k = A \sqsubseteq B\}$
$\mathsf{getGCI1}(A)$	:=	$\{k \in \mathcal{T} \mid k = A_1 \sqcap \ldots \sqcap A_i \sqcap \ldots \sqcap A_n \sqsubseteq B, A_i = A\}$
$\mathsf{getGCI2}(A)$	:=	$\{k \in \mathcal{T} \mid k = A \sqsubseteq \exists r.B\}$
$\mathsf{getGCI3A}(A)$	:=	$\{k \in \mathcal{T} \mid k = \exists r.A \sqsubseteq B\}$
$\mathsf{getGCI3r}(r)$	:=	$\{k \in \mathcal{T} \mid k = \exists r.A \sqsubseteq B\}$
$\mathsf{getGCI3rA}(r, A)$	:=	$\{k \in \mathcal{T} \mid k = \exists r.A \sqsubseteq B\}$
$\mathsf{getSubRoles}(s)$	:=	$\{r \in \mathsf{N_R} \mid r \sqsubseteq s \in \mathcal{T}\}$
$\mathsf{getSuperRoles}(r)$	:=	$\{s \in \mathsf{N_R} \mid r \sqsubseteq s \in \mathcal{T}\}$
$\mathsf{getRolesFunAncestor}(r_1)$	:=	$\{r_2 \in \mathsf{N_R} \mid \exists s \in \mathsf{N_R} : f(s) \in \mathcal{T} \wedge r_1 \sqsubseteq s \in \mathcal{T}, r_2 \sqsubseteq s \in \mathcal{T}\}$
$\mathsf{getSubsumers}(x)$	:=	$\{A \in \mathsf{N_C} \mid (x, A) \in S\}$
$\mathsf{getSecondByFirst}(r, x)$	:=	$\{y \in V \mid (r, x, y) \in R\}$
$\mathsf{getFirstBySecond}(r, y)$	:=	$\{x \in V \mid (r, x, y) \in R\}$
$\mathsf{getRelationsByFirst}(x)$	:=	$\{r \in \mathsf{N_R} \mid \exists y \in V : (r, x, y) \in R\}$
$\mathsf{getRelationsBySecond}(y)$	:=	$\{r \in \mathsf{N_R} \mid \exists y \in V : (r, x, y) \in R\}$
$\mathsf{isFunctional}(r)$	:=	$\begin{cases} \textbf{true} & \text{if } f(r) \in \mathcal{T} \\ \textbf{false} & \text{otherwise} \end{cases}$
$\mathsf{isTransitive}(r)$	:=	$\begin{cases} \textbf{true} & \text{if } r \circ r \sqsubseteq r \in \mathcal{T} \\ \textbf{false} & \text{otherwise} \end{cases}$

Table 5.1: Ontology and status functions.

called the *change propagation*. Each rule suggests the modification of at least one of these structures, producing that new rules can be applied using the new status of the structures. The algorithm is based on a distributed execution of completion rules, where each completion rule is executed in a similar fashion.

A conceptual scheme of the algorithm is presented in Table 5.2.

1. $S, R, Q := \emptyset$
2. **for each** concept name A, add $((A, \emptyset), A)$ and $((A, \emptyset), \top)$ to Q
3. **while** $Q \neq \emptyset$
4. take one element e in Q and remove it from Q
5. **if** e is an S-entry
6. let Q' be the result of applying all the S-rules to e
7. **else if** e is an R-entry
8. let Q' be the result of applying all the R-rules to e
9. **else** $Q' := \emptyset$
10. $Q := Q \cup ((Q' \setminus S) \setminus R)$

Table 5.2: General algorithm.

5.2 Completion rules

The rule descriptions in Table 4.6 do not have a condition to avoid repetition in their specification. This is managed by taking produced values as "suggestions", requiring an extra step of membership check to avoid repetition.

Every completion rule is sensitive to changes in S or sensitive to changes in R. In order to ensure the correctness of the algorithm, every completion rule must propose all the possible changes according to the given piece of information. Moreover, completion rules sensitive to both sets must be subdivided into two different completion rules: one sensitive to S and one sensitive to R. To see how this works, let us take CR-4 as an example. This is shown in Table 5.3.

CR-4	**if** $\exists s.A \sqsubseteq B \in \mathcal{T}$, $\underline{(r,x,y) \in R}$, $\underline{(y,A) \in S}$, $r \sqsubseteq_{\mathcal{T}} s$ **then** $S := S \cup \{(x,B)\}$

Table 5.3: Completion rule CR-4.

Looking at the underlined terms, we know that this rule is sensitive to the changes in both sets, S and R. The algorithm is then designed having two parts, one sensitive to S and one sensitive to R. They are called CR-4-S and CR-4-R respectively.

Please notice that the order of getting the different values can lead to faster or slower algorithms. In fact, some order can be better for some ontologies, and worse for others. Although GALEN was used to approximate the best order in the rules, there might be a better order, and this future work is discussed in Chapter 7.

Looking at CR-4-S, we have that $(y, A) \in S$. This means that we know y and A. The purpose is to get x and B. To get r we try all the role names such that there is a triplet in R with y as second component, using $\mathsf{getRelationsBySecond}(y)$. We find all s that are super roles of r using $\mathsf{getSuperRoles}(r)$.

We can use $\mathsf{getGCI3rA}(s, A)$ to get the axiom $\exists s.A \sqsubseteq B$. Finally, we get x for each triplet $(r, x, y) \in R$ using $\mathsf{getFirstBySecond}(r, y)$, and suggest the addition of (x, B) to S.

The set of known variables goes like this:

$$\{A, y\} \overset{\mathsf{getRelationsBySecond}(y)}{\longrightarrow_*} \{A, r, y\} \overset{\mathsf{getSuperRoles}(r)}{\longrightarrow_*} \{A, r, s, y\} \overset{\mathsf{getGCI3rA}(s,A)}{\longrightarrow_*} \{A, \boldsymbol{B}, r, s, y\}$$

$$\overset{\mathsf{getFirstBySecond}(r,y)}{\longrightarrow_*} \{A, \boldsymbol{B}, r, s, \boldsymbol{x}, y\}$$

In this transition diagram, the bold letters correspond to the required variables.

This process, written with sets of new elements, is:

$$\{A, y\} \overset{\mathsf{getRelationsBySecond}(y)}{\longrightarrow_*} \{r\} \overset{\mathsf{getSuperRoles}(r)}{\longrightarrow_*} \{s\} \overset{\mathsf{getGCI3rA}(s,A)}{\longrightarrow_*} \{\boldsymbol{B}\} \overset{\mathsf{getFirstBySecond}(r,y)}{\longrightarrow_*} \{\boldsymbol{x}\}$$

In the case of CR-4-R, the provided entry is $(r, x, y) \in R$. In this case, we know r, x and y and we need to find B. We use $\mathsf{getSubsumers}(y)$ to get A. We find all s that are super roles of r using $\mathsf{getSuperRoles}(r)$. Then, using $\mathsf{getGCI3rA}(s, A)$ we get $\exists s.A \sqsubseteq B$. Finally we suggest the addition of (x, B) to S.

$$\{r, \boldsymbol{x}, y\} \overset{\mathsf{getSubsumers}(y)}{\longrightarrow_*} \{A\} \overset{\mathsf{getSuperRoles}(r)}{\longrightarrow_*} \{s\} \overset{\mathsf{getGCI3rA}(s,A)}{\longrightarrow_*} \{\boldsymbol{B}\}$$

The other rules follow the same idea. Given the information of a new tuple in the sets, each rule computes the remaining variables to trigger the rule itself.

The function $\mathsf{addToV}(x)$ is the only function that modifies the status from inside the rule. All the other modifications are postponed to be processed later on. The application of addToV does not alter the other rules since only adds fresh elements that are not connected to any other one. The relations in S and R are processed together with the other elements.

The look-up process presented in Table 5.4 is a guide to design the algorithm of each rule.

CR-1	$\{A, \boldsymbol{x}\} \overset{\mathsf{getGCI0}(A)}{\longrightarrow_*} \{\boldsymbol{B}\}$
CR-2	$\{A_i, \boldsymbol{x}\} \overset{\mathsf{getGCI1}(A_i)}{\longrightarrow_*} \{A_1, \ldots, A_n, \boldsymbol{B}\}$
CR-3	$\{A, \boldsymbol{x}\} \overset{\mathsf{getGCI2}(A)}{\longrightarrow_*} \{\boldsymbol{B}, \boldsymbol{r}\} \longrightarrow \{\boldsymbol{v}, \boldsymbol{y}\}$
CR-4-S	$\{A, y\} \overset{\mathsf{getRelationsBySecond}(y)}{\longrightarrow_*} \{r\} \overset{\mathsf{getSuperRoles}(r)}{\longrightarrow_*} \{s\} \overset{\mathsf{getGCI3rA}(s,A)}{\longrightarrow_*} \{\boldsymbol{B}\} \overset{\mathsf{getFirstBySecond}(r,y)}{\longrightarrow_*} \{\boldsymbol{x}\}$
CR-4-R	$\{r, \boldsymbol{x}, y\} \overset{\mathsf{getSubsumers}(y)}{\longrightarrow_*} \{A\} \overset{\mathsf{getSuperRoles}(r)}{\longrightarrow_*} \{s\} \overset{\mathsf{getGCI3rA}(s,A)}{\longrightarrow_*} \{\boldsymbol{B}\}$
CR-5-R-1	$\{r_1, \boldsymbol{x}, y\} \overset{\mathsf{getSuperRoles}(r_1)}{\longrightarrow_*} \{\boldsymbol{s}\} \overset{\mathsf{getSubRoles}(s)}{\longrightarrow_*} \{r_2\} \overset{\mathsf{getSecondByFirst}(r_2,y)}{\longrightarrow_*} \{\boldsymbol{z}\}$
CR-5-R-2	$\{r_2, y, \boldsymbol{z}\} \overset{\mathsf{getSuperRoles}(r_2)}{\longrightarrow_*} \{\boldsymbol{s}\} \overset{\mathsf{getSubRoles}(s)}{\longrightarrow_*} \{r_1\} \overset{\mathsf{getFirstBySecond}(r_1,y)}{\longrightarrow_*} \{\boldsymbol{x}\}$
CR-6-S	$\{A, \boldsymbol{x}\} \overset{\mathsf{getGCI3A}(A)}{\longrightarrow_*} \{\boldsymbol{B}, s\} \overset{\mathsf{getSubRoles}(s)}{\longrightarrow_*} \{\boldsymbol{r}\} \overset{\mathsf{getSecondByFirst}(r,x)}{\longrightarrow_*} \{\boldsymbol{y}\} \longrightarrow \{B', \psi\} \longrightarrow \{\boldsymbol{v}\}$
CR-6-R	$\{\boldsymbol{r}, \boldsymbol{x}, \boldsymbol{y}\} \longrightarrow \{B', \psi\} \overset{\mathsf{getSubsumers}(x)}{\longrightarrow_*} \{A\} \overset{\mathsf{getSuperRoles}(r)}{\longrightarrow_*} \{s\} \overset{\mathsf{getGCI3rA}(s^-,A)}{\longrightarrow_*} \{\boldsymbol{B}\} \longrightarrow \{\boldsymbol{v}\}$
CR-7	$\{\boldsymbol{r_2}, \boldsymbol{x}, \boldsymbol{y}\} \overset{\mathsf{getSuperRoles}(r_2)}{\longrightarrow_*} \{r\} \overset{\mathsf{getSuperRoles}(r)}{\longrightarrow_*} \{s\} \longrightarrow \{A', B', \varphi, \psi\} \overset{\mathsf{getGCI3r}(s^-)}{\longrightarrow_*} \{\boldsymbol{A}, \boldsymbol{B}\} \overset{\mathsf{getSubRoles}(r)}{\longrightarrow_*} \{r_1\} \longrightarrow \{\boldsymbol{v}\}$
CR-8-S	$\{A, y\} \overset{\mathsf{getGCI2}(A)}{\longrightarrow_*} \{\boldsymbol{B}, r_2\} \overset{\mathsf{getSuperRoles}(r_2)}{\longrightarrow_*} \{s\} \overset{\mathsf{getSubRoles}(s)}{\longrightarrow_*} \{r_1\} \overset{\mathsf{getFirstBySecond}(r_1,y)}{\longrightarrow_*} \{\boldsymbol{x}\}$
CR-8-R	$\{r_1, \boldsymbol{x}, y\} \overset{\mathsf{getSuperRoles}(r_1)}{\longrightarrow_*} \{s\} \overset{\mathsf{getSubsumers}(y)}{\longrightarrow_*} \{A\} \overset{\mathsf{getGCI2}(A)}{\longrightarrow_*} \{\boldsymbol{B}, s\}$
CR-9	$\{\boldsymbol{r_1}, \boldsymbol{x}, \boldsymbol{y}\} \overset{\mathsf{getSuperRoles}(r_1)}{\longrightarrow_*} \{s\} \overset{\mathsf{getSubRoles}(s)}{\longrightarrow_*} \{r_2\} \longrightarrow \{\psi\} \overset{\mathsf{getSecondByFirst}(r_2,x)}{\longrightarrow_*} \{\boldsymbol{z}\} \longrightarrow \{\boldsymbol{v}, \varphi\}$

Table 5.4: Variable look-up process.

The general way of producing the algorithms consists of replacing the $\longrightarrow_*$ by loops (**for each**) trying the valid combinations, and placing the **if** to verify the conditions.

The result of the application is a set of tuples (S-entries and R-entries) to be processed afterwards. The rules can also modify set V, but the entries related to the new members of V are processed together with the other entries.

Table 5.5, Table 5.6 and Table 5.7 have all the algorithms according to what is described above. In the pseudocode we use a notation of pattern matching when getting the components of a complex

data structure. For example, the notation $k := A \sqsubseteq B$ means to create a new axiom using A and B, and assigning this axiom to k. Whereas, $k = A \sqsubseteq B$ means to assign values to A and B such that for a given axiom k, the equality holds. The function $\mathsf{oneElemOf}(Q)$ returns one element of a non-empty set Q.

5.3 Correctness and Complexity

Correctness and complexity are shown below.

Theorem 9. *The algorithm in Table 5.2 is correct.*

Proof. To be correct, the algorithm needs to produce all the entries required by the completion rules, it needs to execute all the completion rules fairly, and it needs to terminate.

To prove that all the required entries are produced, we can observe that each completion rule considers the given entry and completes all the remaining variables in order to be in conditions of application. In this way, no possible combination of values is left apart.

To prove that all rules are applied fairly, we can observe that all the rules are applied for each given entry, thanks to the chain construction.

To prove that the algorithm terminates, it is enough to observe that the duplicates checker prevents the execution of an entry twice, saturating bounded sets.

By Lemma 8, the algorithm can use the rules in Table 4.6 producing the same result as in Table 4.5, which are proved to be correct.

□

Theorem 10. *For a normalized $\mathcal{ELHI}f_{\mathcal{R}^+}$ general TBox $\mathcal{T}$, the algorithm runs in exponential time.*

Proof. The proof in [Vu(2008)] is based on the supporting sets of the algorithm: V, S, and R, finding a bound for their complete saturation. In this thesis, we use the same supporting sets, and the bound is the same. □

5.4 Direct subsumers

Once the completion rules cannot be applied anymore, the auxiliary symbols are removed. The following step is to compute the direct subsumers. For doing that, we present an algorithm based on the enhanced traversal method in [Suntisrivaraporn(2009)]. The algorithm is shown in Table 5.8.

The intuition of the algorithm is to traverse a graph in levels, where **TOP** is the only element in level 0 and it is the starting node. The algorithm starts in level 0 and computes the direct subsumees of **TOP**. The direct subsumees of **TOP** are those having only elements in level 0 (**TOP**) as a subsumer. Once it gets the direct subsumees of **TOP**, it marks them as belonging to level 1. Then, it computes the direct subsumees of every element in level 1. They are those having only subsumers of level 1 and level 0. In this way the algorithm detects the elements of level 2, and then continues with the remaining levels until it marks all the nodes in the graph.

These subsumees are named *children*, and the *parents* are obtained by reversing the edges. During the process, the *equivalents* are obtained by detecting mutual subsumption.

procedure main()
$S := \emptyset$
$R := \emptyset$
$Q := \emptyset$
for each $A \in N_C$
  $Q := Q \cup \{((A, \emptyset), A)\} \cup \{((A, \emptyset), \top)\}$
while $Q \neq \emptyset$
  $e :=$ oneElemOf(Q)
  $Q := (Q \setminus \{e\}) \cup$ apply(e)

function apply$(e) \rightarrow$ *ret*
ret $:= \emptyset$
if $e = (x, A)$
  $S := S \cup \{(x, A)\}$
  ret $:=$ chainS(x, A)
if $e = (r, x, y)$
  $R := R \cup \{(r, x, y)\}$
  ret $:=$ chainR(r, x, y)
ret $:=$ (*ret* $\setminus S) \setminus R$

function chainS$(x, A) \rightarrow$ *ret*
ret $:=$ CR-1$(x, A) \cup$ CR-2(x, A)
  $\cup$ CR-3$(x, A) \cup$ CR-4-S(x, A)
  $\cup$ CR-6-S$(x, A) \cup$ CR-8-S(x, A)

function chainR$(r, x, y) \rightarrow$ *ret*
ret $:=$ CR-4-R(r, x, y)
  $\cup$ CR-5-R-1$(r, x, y) \cup$ CR-5-R-2(r, x, y)
  $\cup$ CR-6-R$(r, x, y) \cup$ CR-7(r, x, y)
  $\cup$ CR-8-R$(r, x, y) \cup$ CR-9(r, x, y)

function CR-1$(x, A) \rightarrow$ *ret*
ret $:= \emptyset$
for each $k \in$ getGCI0(A)
  $k = A \sqsubseteq B$
  ret $:=$ *ret* $\cup \{(x, B)\}$

function CR-2$(x, A) \rightarrow$ *ret*
ret $:= \emptyset$
for each $k \in$ getGCI1(A)
  $k = A_1 \sqcap \ldots \sqcap A \sqcap \ldots \sqcap A_n \sqsubseteq B$
  if $\{(x, A_1), \ldots, (x, A_n)\} \subseteq S$
    ret $:=$ *ret* $\cup \{(x, B)\}$

function CR-3$(x, A) \rightarrow$ *ret*
ret $:= \emptyset$
for each $k \in$ getGCI2(A)
  $k = A \sqsubseteq \exists r.B$
  if isFunctional(r)
    $v := (\top, \{\exists r^-.A\})$
    if $v \notin V$
      addToV(v)
    ret $:=$ *ret* $\cup \{(v, B)\}$
    ret $:=$ *ret* $\cup \{(v, \top)\}$
    ret $:=$ *ret* $\cup \{(r, x, v)\}$
  else
    $y := (B, \emptyset)$
    ret $:=$ *ret* $\cup \{(r, x, y)\}$

Table 5.5: Algorithm (Table 1 / 3).

function CR-4-S$(y, A) \rightarrow$ *ret*
ret $:= \emptyset$
for each $r \in$ getRelationsBySecond(y)
  for each $s \in$ getSuperRoles(r)
    for each $k \in$ getGCI3rA(s, A)
      $k = \exists s.A \sqsubseteq B$
      for each $x \in$ getFirstBySecond(r, y)
        ret $:=$ *ret* $\cup \{(x, B)\}$

function CR-4-R$(r, x, y) \rightarrow$ *ret*
ret $:= \emptyset$
for each $A \in$ getSubsumers(y)
  for each $s \in$ getSuperRoles(r)
    for each $k \in$ getGCI3rA(s, A)
      $k = \exists s.A \sqsubseteq B$
      ret $:=$ *ret* $\cup \{(x, B)\}$

function CR-5-R-1$(r_1, x, y) \rightarrow$ *ret*
ret $:= \emptyset$
for each $s \in$ getSuperRoles(r_1)
  if isTransitive(s)
    for each $r_2 \in$ getSubRoles(s)
      for each $z \in$ getSecondByFirst(r_2, y)
        ret $:=$ *ret* $\cup \{(s, x, z)\}$

function CR-5-R-2$(r_2, y, z) \rightarrow$ *ret*
ret $:= \emptyset$
for each $s \in$ getSuperRoles(r_2)
  if isTransitive(s)
    for each $r_1 \in$ getSubRoles(s)
      for each $x \in$ getFirstBySecond(r_1, y)
        ret $:=$ *ret* $\cup \{(s, x, z)\}$

function CR-6-S$(x, A) \rightarrow$ *ret*
ret $:= \emptyset$
for each $k \in$ getGCI3A(A)
  $k = \exists s^{-}.A \sqsubseteq B$
  for each $r \in$ getSubRoles(s)
    for each $y \in$ getSecondByFirst(r, x)
      if $B \notin$ getSubsumers(y)
        $y = (B', \psi)$
        $v := (B', \psi \cup \{\exists r^{-}.A\})$
        if $v \notin V$
          addToV(v)
          for each $p \in$ getSubsumers(y)
            ret $:=$ *ret* $\cup \{(v, p)\}$
        ret $:=$ *ret* $\cup \{(v, B)\}$
        ret $:=$ *ret* $\cup \{(r, x, v)\}$

function CR-6-R$(r, x, y) \rightarrow$ *ret*
ret $:= \emptyset$
for each $A \in$ getSubsumers(x)
  for each $s \in$ getSuperRoles(r)
    for each $k \in$ getGCI3rA(s^{-}, A)
      $k = \exists s^{-}.A \sqsubseteq B$
      if $B \notin$ getSubsumers(y)
        $y = (B', \psi)$
        $v := (B', \psi \cup \{\exists r^{-}.A\})$
        if $v \notin V$
          addToV(v)
          for each $p \in$ getSubsumers(y)
            ret $:=$ *ret* $\cup \{(v, p)\}$
        ret $:=$ *ret* $\cup \{(v, B)\}$
        ret $:=$ *ret* $\cup \{(r, x, v)\}$

Table 5.6: Algorithm (Table 2 / 3).

function CR-8-S$(y, A) \rightarrow$ *ret*
ret $:= \emptyset$
for each $k \in$ getGCI2(A)
  $k = A \sqsubseteq \exists r_2^-.B$
  for each $s \in$ getSuperRoles(r_2)
    if isFunctional(s^-)
      for each $r_1 \in$ getSubRoles(s)
        for each $x \in$ getFirstBySecond(r_1, y)
          ret $:=$ *ret* $\cup \{(x, B)\}$

function CR-8-R$(r_1, x, y) \rightarrow$ *ret*
ret $:= \emptyset$
for each $s \in$ getSuperRoles(r_1)
  if isFunctional(s^-)
    for each $A \in$ getSubsumers(y)
      for each $k \in$ getGCI2(A)
        $k = A \sqsubseteq \exists r_2^-.B$
        if $r_2 \in$ getSubRoles(s)
          ret $:=$ *ret* $\cup \{(x, B)\}$

function CR-7$(r_2, x, y) \rightarrow$ *ret*
ret $:= \emptyset$
$x = (A', \varphi)$
$y = (B', \psi)$
for each $r \in$ getSuperRoles(r_2)
  if isTransitive(r)
    for each $s \in$ getSuperRoles(r)
      for each $k \in$ getGCI3r(s^-)
        $k = \exists s^-.A \sqsubseteq B$
        for each $r_1 \in$ getSubRoles(r)
          if $\exists r_1^-.A \in \varphi$
            $v := (B', \psi \cup \{\exists r^-.A\})$
            if $v \notin V$
              addToV(v)
              for each $p \in$ getSubsumers(y)
                ret $:=$ *ret* $\cup \{(v, p)\}$
            ret $:=$ *ret* $\cup \{(v, B)\}$
            ret $:=$ *ret* $\cup \{(r_2, x, v)\}$

function CR-9$(r_1, x, y) \rightarrow$ *ret*
ret $:= \emptyset$
if $y = (\top, \psi)$
  for each $r_2 \in$ getRolesFunAncestor(r_1)
    for each $z \in$ getSecondByFirst(r_2, x)
      if $z = (\top, \varphi)$
        if $y \neq z$
          $v := (\top, \psi \cup \varphi)$
          if $v \notin V$
            addToV(v)
          for each $p \in$ getSubsumers(y)
            ret $:=$ *ret* $\cup \{(v, p)\}$
          for each $p \in$ getSubsumers(z)
            ret $:=$ *ret* $\cup \{(v, p)\}$
          ret $:=$ *ret* $\cup \{(r_1, x, v)\}$

Table 5.7: Algorithm (Table 3 / 3).

```
procedure computeDag
  classified := classified ∪ (⊤)
  for each A ∈ N_C
    if (A ∉ classified)
      dagClassify(A)
  for each A ∈ N_C
    if children(A) = ∅
      children(A) := {⊥}

procedure dagClassify(A)
  candidates := {}
  for each B ∈ N_C, B ≠ A, B ≠ ⊤, ((A, ∅), B) ∈ S
    if ((B, ∅), A) ∈ S
      classified := classified ∪ {B}
      equivalents := equivalents ∪ {(A, B)}
    else
      if B ∉ classified
        dagClassify(B)
      candidates := candidates ∪ {B}
  dagInsert(A, candidates)
  classified := classified ∪ {A}

procedure dagInsert(A, candidates)
  marked := ∅
  for each X ∈ N_C, B ∈ candidates
    if (B, X) ∈ parents
      marked := marked ∪ {X}
  parents := parents ∪ {(A, B) | B ∈ candidates, B ∉ marked}
  for each (A, B) ∈ parents
    children := children ∪ {(B, A)}
```

Table 5.8: Algorithm for computing direct subsumers.

Chapter 6

Implementation and Experiments

In this chapter, we discuss the implementation of the algorithm presented in Chapter 5. The program is called **jcel** and its source code is available at `http://jcel.sourceforge.net`. We also talk about the experiments we conducted.

6.1 Implementation

The classification algorithm is implemented in Java, which is an object-oriented, platform-independent, multithreaded programming environment. The object-oriented design of this algorithm brings a very low coupling, since each rule can be changed separately. It also brings high cohesion because all the logic of the completion rule is only in the rule itself.

It is implemented using 4 modules:

- **jcel-core** : it is the reasoner itself; it uses integer numbers in the internal representation for concepts and roles, instead of special Java classes
- **jcel-owlapi** : transforms objects from the OWL API 3.2.2 to the internal representation used by the core
- **jcel-adapter** : is an adapter from the OWL API 2.2.0 to the OWL API 3.2.2
- **jcel-protege** : is a module to use **jcel** as a Protégé plug-in, either with the OWL API 3.2.2 or the OWL API 2.2.0

The interconnection of these modules is shown in Figure 6.1. The diagram shows how the problem of compatibility with the OWL API 2.2.0 and the OWL API 3.2.2 is solved. The rectangles are the modules or libraries, and each arrow indicates a *dependency*, i.e. there is a class or interface in one library (the arrow end) that is used by the other library (the arrow start).

6.2 Successful improvements

In this section we discuss successful improvements that played an important role in the implementation.

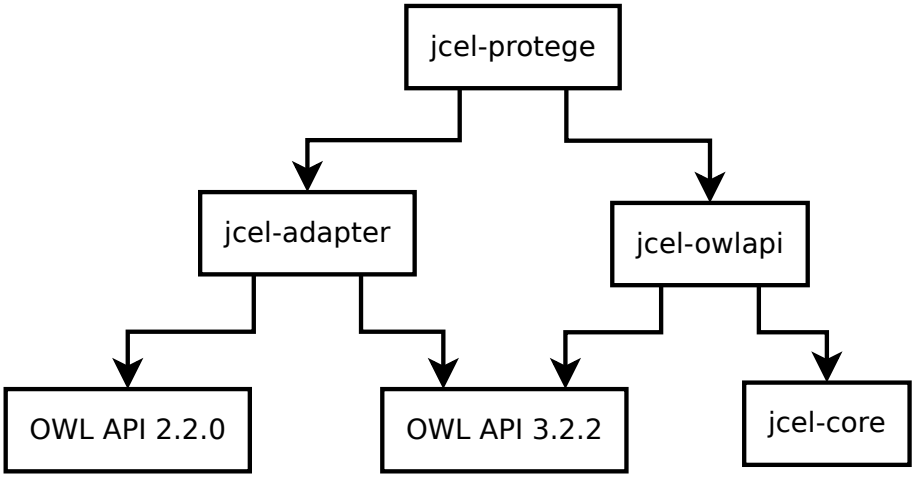

Figure 6.1: Use of modules in the implementation.

6.2.1 Use of collections

When we talk about the representation of S, we often refer to it as a set. However, it is actually implemented by a map of array lists. At an abstract level, these lists are seen as collections, which do not differ much from sets.

The sets in Java, for example `HashSet` or `TreeSet`, are implemented using extra information to make efficient the membership test. This extra information, used in all the elements of S and R, makes a very important use of memory. In addition, the Java structures require the use of classes (like `Integer`), but the primitive type for integer numbers, `int`, is kept to make operations faster and simpler.

Replacing these sets by arrays of `int` of exponential growth in the internal representation of S, changed drastically the use of memory and therefore the use of the garbage collector and the total execution time.

6.2.2 Use of two sets of entries

On the one hand, the algorithm presented by [Suntisrivaraporn(2009)] has a structure that is a list of queues, having one queue for each concept name. On the other hand, in this thesis we talk about one single set Q which has all the entries.

In the real implementation there are two sets, one Q_S for S-entries and one Q_R for R-entries. These sets are processed in order to have a balanced size. Thus, the algorithm chooses the set with more entries, and takes one entry from that set. In the tested ontologies this was faster than having one single set, and faster than processing first one set and then the other one.

6.2.3 Individualized modification of rules

Since every rule is independent, we used a profiler to detect how much time they required. Following the specification, we modified each rule to make it faster. Some of the successful changes were to place the **if** instruction before the **for each** instruction, and to get the axioms in the ontology before taking the elements in S or R.

6.2.4 Integers as identifiers

In the **jcel-core** module every identifier for concepts or roles is an `Integer`. The mapping is done by the **jcel-owlapi** module. Storing integer numbers and comparing them became more efficient in memory and time than using `String` or other more complex structure. The clear disadvantage derives from the fact of not having types for them. Thus, an extra care must be taken to avoid misplacing the order of parameters in methods and constructors due to the absence of distinctive types.

6.3 Discarded implementations

During the development of the implementation, many different approaches were considered. The main restriction was the memory. The system was developed using a 32-bit Java Virtual Machine. The ontology used to test these approaches was SNOMED CT.

In this section, we discuss the different alternative implementations we tried and why we discarded them.

6.3.1 Shared references

In order to save memory, the idea was to share references for sets that were equal. Java internally works mainly with pointers to objects. If two sets are known to be equivalent, they can share the same pointer. Instead of storing the set twice, only one instance is needed.

Set S is formed by sets of subsumers. Many elements may share the same set of subsumers. Some of these sets can be considerably big. Taking this idea into account, the implementation had a set of references. Using hash maps, having a set as key and a reference to itself as value, the system was able to avoid duplication of sets.

The inconvenience of this approach, beyond the overhead of the look-up method for finding the proper pointer, was to keep updated these sets. In the process of completing set S, two sets may be the same, but differ afterwards. For example,

$$S = \{(1,3),(1,4),(2,3),(2,4)\}$$

In the representation,

$$S(1) = \{3,4\}, S(2) = \{3,4\}$$

so they can be

$$S(1) = ptr1, S(2) = ptr1, ptr1 = \{3,4\}$$

but when adding $\{(2,5)\}$, the pointers need to differ, in order to avoid adding $\{(1,5)\}$.

The problem could be overcome by keeping the backlinks, this is, keeping the links to the keys using these sets.

The final result considering time and memory was unfavorable, and therefore this approach was discarded.

6.3.2 Binary compression

Trying to solve the problem of storing set S, a possible approach was to store the data using some compression. Set S is actually a map, where the key is the subsumee and the value is the set of

subsumers. Since each `Integer` in Java (in 32-bit platform) uses 32 bits, and considering that the identifiers were much less than $2^{24} = 16777216$, the idea was to take advantage of the remaining 8 bits. Hence, 3 numbers, using 32 bits each, use 96 bits. Using the 24-bit approach, it is possible to encode 4 numbers of 24 bits using 3 numbers of 32 bits. Although this gain seems to be small, reducing the use of memory, let us say by 20%, would have been enough to make the classification process much faster.

In all these process of compression and decompression, a considerable time was needed. Having the algorithm in mind, this decompression was needed for every time a check $(x, A) \in S$ was considered. The consumed time and the small gain of memory made us discard this approach.

6.3.3 List compression

A list compression algorithm was considered. With the following set:

$$S = \{(1, 3), (1, 5), (1, 7)\}$$

This set can be stored as

$$S(1) = \{3, 5, 7\}$$

where $S(x)$ refers to all the elements y such that $(x, y) \in S$.

It is possible to store the elements in a list. Although lists could contain repeated elements, in the algorithm there is a check of membership that avoids repetition. But, if we store a list and we consider the differences (starting with 0), we have

$$S(1) = [+3, +2, +2]$$

Assuming that we have less than one million identifiers (in fact, less than 2^{20}), the differences between them during the execution may be smaller. Using 15 bits for the number and 1 bit for the sign, we can store the numbers in 16 bits using their differences.

For example:

$$S(1) = [+3, +32000, -10000]$$

would be

$$S(1) = \{3, 32003, 22003\}$$

Considering the bits, the first representation can use 16-bit integers (with positive and negative values), while the second one uses 32-bit integers, with only non-negative numbers.

Since the list represents a set, there is no need to repeat any number. Then, 0 can be used as escape code, to force a 32-bit representation, without using the differences. Thus,

$$S(1) = \{3, 720918\}$$

would be

$$S(1) = [+3, 0, 11, 22]$$

because $11 \cdot 2^{16} + 22 = 720918$.

Like in the case of the binary compression, this approach took too much execution time, and the difference in the use of memory was negligible. For those reasons, we discarded this approach.

6.3.4 External storage

The 32-bit Java Virtual Machine has a limit of 2600 MB. Even if we consider this as a big number, storing numbers uses a big amount of memory. Even for less than one million concepts, the quadratic-style approach of storing S needs an important amount of memory. However, this number is not that big when using an external storage, for example, using files or local sockets.

Sockets opened locally can communicate very fast. However, their communication speed was not fast enough. The big number of operations required for a classification made the transmitted amount of data too much, and hence too slow. For the file, the access was too slow as well. For these reasons, these approaches were discarded.

6.3.5 Transitive data structure

The algorithm we use for classification works saturating two sets, S and R, using a set of entries Q while Q is not empty. Set S basically stores a transitive closure of a reachability graph, where *is reachable from* means *is subsumer of*. Having this idea in mind, we implemented a "slim" graph, a graph where the transitivity is not explicit but implicit.

In this way, $S_t = \{(1,2),(2,3)\}$ actually means $S = \{(1,2),(2,3),(1,3)\}$, where S_t is the implicit representation. The algorithm works completing S in a progressive fashion, though. Thus, it was needed to represent $S = \{(1,2),(2,3)\}$ without really representing $S = \{(1,2),(2,3),(1,3)\}$.

For solving this, two possibilities were considered. One was to store "negative edges". For example, $S = \{(1,2),(2,3)\}$ would be represented as $S_t = \{(1,2),(2,3)\}$ and $S_m = \{(1,3)\}$, where S_m stores the negative edges. This approach needed an explicit representation of S_m bringing an almost identical case as the initial one.

Another possibility was to process the different edges first. In this approach, the negative edges were not really stored, but processed first to avoid the inconsistency. Unfortunately this approach did not work either, since the temporary use of memory was too high.

The post processing was not skipped either because of the auxiliary entities. After the normalization, a number of auxiliary entities are created, and become part of the graph structure. Removing them without creating false direct subsumers was much slower than just running the standard algorithm. For these reasons, we discarded this approach.

6.3.6 Horn-style completion

To determine which rules were consuming more execution time, a profiler was developed. This profiler uses some execution time, and because of that, the total execution time is greater. In Table 6.1, we can see a comparison of the consumed time in the execution, and how it was used. The rule that consumed the most was CR-2.

This characteristic gave us the idea of improving CR-2, which is described in Table 6.2.

In Prolog, each rule has a counter of the number of satisfied preconditions. When this counter reaches 0, the rule is triggered. Adapting an idea of how Prolog works, this rule would be triggered only when all the conditions are satisfied. This means that a counter is kept, and every time it is informed that $(x, A_i) \in S$ for some x and some A_i, the counter is decreased. Reaching zero means that all $A_1 \dots A_n$ are satisfied and necessarily $S := S \cup \{(x, B)\}$ is triggered.

rule	accepted	suggested	total time
CR-1	5461356	139813430	55121 ms
CR-2	2875912	139813430	**149232 ms**
CR-3	2803075	139813430	46201 ms
CR-4-S	15462923	139813430	74227 ms
CR-4-R	1102649	3226022	58358 ms

Table 6.1: Used times for each rule when classifying SNOMED CT.

CR-2 **if** $A_1 \sqcap \ldots \sqcap A_i \sqcap \ldots \sqcap A_n \sqsubseteq B \in \mathcal{T}$, $(x, A_1) \in S$, …, $\underline{(x, A_i) \in S}$, …, $(x, A_n) \in S$
then $S := S \cup \{(x, B)\}$

Table 6.2: Completion rule CR-2.

The idea was implemented using the algorithm described in Table 6.3. In this algorithm we use "maps", which formally could be considered as set of pairs M, where for each x (called *key*) there is at most one y (called *value*) such that $(x, y) \in M$. When there is no such y, we say that the value is **undef**. A map has the following function:

$$\mathsf{get}(m, x) \quad := \quad \begin{cases} y & \text{if } (x, y) \in m \\ \textbf{undef} & \text{otherwise} \end{cases}$$

In the algorithm, *counters* is a map of maps, where the key is x (the same x of CR-2). Each map m has an axiom of type GCI 1 (which is $A_1 \sqcap \ldots \sqcap A_n \sqsubseteq B$) as key, and the value is the number of A_i that are already satisfied.

It is important to remark that in our experiments the maps were hash maps (`HashMap`), and every axiom had an appropriate hash function, although not necessarily optimal.

The algorithm was tested with the Gene Ontology, since the use of memory was very important. Using around the double of memory, and being slower, we discarded this implementation of CR-2. However, we consider that some extra experiments could be performed trying to reduce the used memory, or improving the accessing time to the maps.

6.3.7 Multithreaded processing

The nature of having independent rules sparked the idea of a distributed execution. Even for a short time, and using some overhead, assuming the processing execution can be distributed in several cores, most of the execution would be distributed and the total time would be smaller. This simple idea is actually compatible with the model. To implement this idea, we needed a protocol to communicate the threads themselves and a monitor to avoid inconsistencies. Java provides many high-level elements to manage this in an elegant fashion.

The first approach was to follow exactly the dynamics shown in Figure 5.1. This means, for each entry, either S-entry or R-entry, to branch the execution among the different completion rules. Each completion rule would have an image of the status. Just when all the threads are finished, the modifications to the status are performed. Considering the correctness, this is equivalent to the

```
function CR-2-Horn(x, A) → ret
  ret := ∅
  m := get(counters, x)
  if m = undef
    m := ∅
    counters := counters ∪ {(x, m)}
  for each k ∈ getGCI1(A)
    k = A1 ⊓ ... ⊓ An ⊑ B
    c := get(m, k)
    if c = undef
      c := n
    m := m \ {(k, c)}
    c := c − 1
    m := m ∪ {(k, c)}
    if c = 0
      ret := ret ∪ {(x, B)}
      m := m \ {(k, c)}
      if m = ∅
        counters := counters \ {(x, m)}
```

Table 6.3: Algorithm CR-2 using a Horn-style completion.

sequential processing. There is a small exception, and it is that some rules modify set V during their execution. In the way the rules run, the new elements added to V do not interfere with other rules.

Despite of the fact that this model looked perfect, each step needed the creation of all the threads (one for each completion rule) and a join (of all the threads) after processing each entry. In the case of SNOMED CT, this is run more than 100 million times. Although the functions provided by Java are very fast, if each one takes 1 ms, the execution time can take more than 27 hours. Unfortunately, the execution time provided by Java was not fast enough, which led us to the second approach.

The second approach was designed having all the completion rules already running. This means, no object creation time was needed, and the join was managed using monitors. This second approach made the protocol very complicated. The critical data was prone to enter in inconsistent states very easily. As a result, the only correct execution was a much slower execution that practically put the completion rules in sequential execution, wasting all the advantages of the multithreaded processing. For the reasons above, we discarded this approach.

6.4 Implementation quality

The implementation includes features of quality. These features added some internal redundancy, which according to the tests did not affect the execution time nor the use of memory. These features brought stability and maintainability to the system, and made it more resilient.

6.4.1 No null pointers

A null pointer or a null reference is a reserved value for a variable indicating that it does not refer to any object. Null pointers were invented in 1965 for ALGOL W by C.A.R. Hoare, and since then they have been used in many languages such as C, C++ and Java. In 2009, he referred to his invention as "The Billion Dollar Mistake"[1].

The main problem of using null pointers is that the *intended* meaning can be different in different parts of the source code. In languages like C there is no difference between 0, NULL and **false**. Projecting the same idea in Java, if a variable of type `List` is **null**, this could be interpreted as an empty list; if a variable of type `Integer` is **null**, this could be interpreted as a value "0"; if a variable of type `Boolean` is **null**, this could be interpreted as a **false**. However, in Java executing a method of a variable with null pointer throws an exception (`NullPointerException`). This brings the idea of considering **null** as an *undefined* object.

In the implementation, public and protected methods do not accept null pointers as parameters nor return null pointers as result. In Table 6.4, there is a comparison of the different meanings of null pointers. In the C-like case, **null** is the same as 0, and the minimum between 3 and 0 is 0. If **null** is accepted as an undefined object, the minimum between 3 and *nothing else* is 3. If **null** is considered an undefined object such that the result is also an undefined object, the minimum between 3 and **null** is **null**. Finally, the exception approach refuses to continue the execution, since it assumes that this computation is due to an error. This latter criterion was adopted for all public or protected methods of every class in the implementation, and they do not accept null pointers as a parameter. Every object sent as a parameter needs to be an instance. As a counterpart, methods never return null pointers.

case	expression	value
C-like	$\min(3, \textbf{null})$	0
undefined accepting	$\min(3, \textbf{null})$	3
undefined rejecting	$\min(3, \textbf{null})$	**null**
exception	$\min(3, \textbf{null})$	$-$

Table 6.4: Compared meanings of null pointers.

6.4.2 Unmodifiable collections

In Java, `Integer` and `String` are types with unmodifiable elements. However, this is not the case with collections (`Collection`), like `List`, `Set` and `Map`, which are modifiable. This means that if a public method returns a set, the caller can modify the set. This does not look harmful unless we consider what happens if the returned set is an internal set. In this case, the caller can actually modify the set and violate the invariant of the called object.

If a defective piece of code tries to modify a set assumed immutable, this should be stopped to avoid a wrong computation. This is exactly the purpose of unmodifiable collections. These collections are returned by public and protected methods, giving simplicity in operations to read them, but impeding their modification. Clearly, the methods to modify each collection is available in the object containing it, if the object allows the modification.

[1] `http://qconlondon.com/london-2009/presentation/Null+References:+The+Billion+Dollar+Mistake`

6.4.3 No cyclic dependencies of packages

A package in Java is a set of classes and interfaces. Each of the mentioned modules has several packages. Each package has several classes and interfaces. One package a depends on another package b when at least one class or interface in a depends on at least one class or interface in b.

Avoiding cyclic dependencies facilitates maintenance, since modifications on one package do not alter any other package that does not depend on the former.

6.4.4 Javadoc

Implementation documentation has been specially considered. Every package, every class and every interface has their corresponding javadoc information. All public and protected methods have javadoc unless their function is inherited (like `toString()`).

6.5 Experiments

The implementation of the presented algorithm was tested with ontologies presented in Chapter 2. The experiments were run on a computer with 2 cores Intel(R) Core(TM)2 Duo CPU E8500 @ 3.16GHz, with processors at 1998 MHz, and using 4 GB of RAM memory.

6.5.1 Experiments in $\mathcal{ELH}_{\mathcal{R}^+}$

In Table 6.7 we compare the sizes of the different ontologies mentioned in Chapter 1. The numbers of concepts and roles do not include **TOP** and **BOTTOM** of concepts and roles. From left to right, the columns show the ontology name, the description logic used by the ontology, the number of axioms, the number of normalized axioms, the number of concepts, the number of roles, the number of auxiliary concepts, and the number of auxiliary roles.

ontology	logic	axioms	norm. ax.	concepts	roles	aux. concepts	aux. roles
NCI	$\mathcal{EL}$	74662	47080	27652	70	0	0
GO	$\mathcal{EL}_{\mathcal{R}^+}$	49363	28900	20465	1	0	0
FMA	$\mathcal{EL}_{\mathcal{R}^+}$	150282	119570	75139	2	0	2
SNOMED CT	$\mathcal{ELH}$	962796	1127193	378569	61	354774	56
NotGalen	$\mathcal{ELH}_{\mathcal{R}^+}$	7540	15089	2748	413	3417	413
CELGalen	$\mathcal{ELH}_{\mathcal{R}^+}$	60637	102742	23141	950	24704	950

Table 6.5: Ontologies using $\mathcal{ELH}_{\mathcal{R}^+}$.

In order to compare this implementation with other reasoners, the execution times were compared with the CEL system. CEL is one of the fastest reasoners in the community and also known to have correct results [Mendez and Suntisrivaraporn(2009)]. It is relevant to mention that the generated inferred ontology is identical in both classifications. The results are shown in Table 6.6.

6.5.2 Experiments in $\mathcal{ELHI}f_{\mathcal{R}^+}$

The full version of GALEN is one of the most challenging ontologies since almost no reasoner can classify it. Two GALEN ontologies were considered. One is the original version of GALEN (referred as GALEN-A), and the other one is a newer version of GALEN (referred as GALEN-B).

ontology	entries	jcel 0.13.0	CEL Plug-in 0.5.0	quotient
NCI	346887	8.9 s	10.2 s	0.87
GO	154489	4.4 s	3.5 s	1.26
FMA	9576858	149 s	2388 s	0.06
SNOMED CT	143039451	1108 s	705 s	1.57
NotGalen	224565	2.9 s	5.2 s	0.56
CELGalen	6836237	52 s	134 s	0.39

Table 6.6: Compared times of classification between **jcel** and CEL.

ontology	logic	axioms	norm. ax.	concepts	roles	aux. concepts	aux. roles
GALEN-A	$\mathcal{ELHI}f_{\mathcal{R}^+}$	8140	12930	2748	413	3458	0
GALEN-B	$\mathcal{ELHI}f_{\mathcal{R}^+}$	61787	95789	23143	950	24704	0

Table 6.7: Ontologies using $\mathcal{ELHI}f_{\mathcal{R}^+}$.

For GALEN-A, **jcel** took 1093 s and CB less than 1 s.

For the case of GALEN-B, **jcel** could not finish due to lack of memory. CB classified this ontology in 5 s.

Although GALEN-B was not classified, classifying GALEN-A is already considered a successful achievement.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

We presented not only a classification algorithm for $\mathcal{ELHI}f_{\mathcal{R}^+}$, but also a meta-algorithm that allows the extension of expressivity in a direct way. The presented algorithm is an efficient generalized cohesive modular algorithm with very low coupling. This general algorithm was presented in a multi-level view, going from the more abstract and general aspects to the implementation details. We also presented an effective way of developing efficient completion rules.

In the experimental results the implementation showed an excellent performance. The implementation is modular, resilient and highly extensible. Implemented in a state-of-the-art technology, it is portable and brings an optimal interface with other technologies of the Semantic Web. We have presented a particular configuration of this algorithm for the logic $\mathcal{ELHI}f_{\mathcal{R}^+}$.

7.2 Future work

7.2.1 Different order for different ontologies

The presented algorithm and implementation are based on the fact that each rule executes the commands in a particular order. A possible extension to this work would be to develop the rules finding the elements in a different order. In fact, different implementations for each rule can be dynamically loaded according to the properties of the ontology. An ontology with many concepts, but a minimal use of roles, may need a different implementation of rules from an ontology with a deep role hierarchy but relatively few concepts. A diagnose test could be run in advance and a set of rules expected to be optimal could be choosen for each ontology.

7.2.2 More expressivity

The general algorithm does not prevent from having rules managing more expressive logics. Every constructor that can be processed using a rule-based algorithm, like concrete domains [Lutz(2002)], could be added without interfering with the other rules. A good extension would be to complete the constructor set to be completely compliant with the OWL 2 EL profile.

Bibliography

[Baader et al.(2003)Baader, Calvanese, McGuinness, Nardi, and Patel-Schneider] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003. ISBN 0521781760.

[Baader et al.(2005)Baader, Brandt, and Lutz] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope. In *Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence IJCAI-05*, Edinburgh, UK, 2005. Morgan-Kaufmann Publishers.

[Baader et al.(2006)Baader, Lutz, and Suntisrivaraporn] Franz Baader, Carsten Lutz, and Boontawee Suntisrivaraporn. CEL—a polynomial-time reasoner for life science ontologies. In Ulrich Furbach and Natarajan Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning (IJCAR'06)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 287–291. Springer-Verlag, 2006.

[Baader et al.(2008)Baader, Brandt, and Lutz] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the EL envelope further. In Kendall Clark and Peter F. Patel-Schneider, editors, *In Proceedings of the OWLED 2008 DC Workshop on OWL: Experiences and Directions*, 2008.

[Haarslev and Möller(2001)] Volker Haarslev and Ralf Möller. Racer system description. In Rajeev Goré, Alexander Leitsch, and Tobias Nipkow, editors, *International Joint Conference on Automated Reasoning, IJCAR'2001, June 18-23, Siena, Italy*, pages 701–705. Springer-Verlag, 2001.

[Kazakov(2009)] Yevgeny Kazakov. Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In *Proceedings of the 21st International Conference on Artificial Intelligence IJCAI 2009*, 2009.

[Lutz(2002)] Carsten Lutz. Description logics with concrete domains — a survey. In *Advances in Modal Logic*, volume 4. Wold Scientific Publishing Co. Pte. Ltd., 2002.

[Mendez and Suntisrivaraporn(2009)] Julian Mendez and Boontawee Suntisrivaraporn. Reintroducing CEL as an OWL 2 EL reasoner. In Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, and Ulrike Sattler, editors, *Proceedings of the 2009 International Workshop on Description Logics (DL2009)*, volume 477 of *CEUR-WS*, 2009.

[Motik et al.(2007)Motik, Shearer, and Horrocks] Boris Motik, Rob Shearer, and Ian Horrocks. A Hypertableau Calculus for SHIQ. In Diego Calvanese, Enrico Franconi, Volker Haarslev, Domenico Lembo, Boris Motik, Sergio Tessaris, and Anni-Yasmin Turhan, editors, *Proc. of the 20th Int. Workshop on Description Logics (DL 2007)*, pages 419–426, Brixen/Bressanone, Italy, June 8–10 2007. Bozen/Bolzano University Press.

[Suntisrivaraporn(2009)] Boontawee Suntisrivaraporn. *Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies*. PhD thesis, Dresden University of Technology, 2009.

[Tsarkov and Horrocks(2006)] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[Vu(2008)] Quoc Huy Vu. Subsumption in the description logic $\mathcal{ELHI}fR^+$ w.r.t. general TBoxes. Master's thesis, Dresden University of Technology, 2008.