Module Extraction for Inexpressive Description Logics

by

Riku Nortjé

submitted in accordance with the requirements
for the degree of

Master of Science

in the subject

Computer Science

at the

UNIVERSITY OF SOUTH AFRICA

Supervisor: Dr. Katarina Britz
Co-supervisor: Prof. Thomas Meyer

August 2011

Declaration

I declare that this dissertation is my own work and that all the sources that I have used or quoted have been indicated and acknowledged by means of complete references.

Riku Nortjé

Abstract

Module extraction is an important reasoning task, aiding in the design, reuse and maintenance of ontologies. Reasoning services such as subsumption testing and MinA extraction have been shown to benefit from module extraction methods. Though various syntactic traversal-based module extraction algorithms exist for extracting modules, many only consider the subsumee of a subsumption statement as a selection criterion for reducing the axioms in the module.

In this dissertation we extend the bottom-up reachability-based module extraction heuristic for the inexpressive Description Logic $\mathcal{EL}$, by introducing a top-down version of the heuristic which utilises the subsumer of a subsumption statement as a selection criterion to minimize the number of axioms in a module. Then a combined bidirectional heuristic is introduced which uses both operands of a subsumption statement in order to extract very small modules. We then investigate the relationship between MinA extraction and bidirectional reachability-based module extraction. We provide empirical evidence that bidirectional reachability-based module extraction for subsumption entailments in $\mathcal{EL}$ provides a significant reduction in the size of modules for almost no additional costs in the running time of the original algorithms.

Acknowledgments

I wish to express my sincere gratitude to the CSIR-Meraka Institute and UNISA for their financial assistance. Thank you Dr. K. Britz and Prof. T. Meyer for your faith in me, your support and patience and for all the times you went far beyond the call of duty as supervisors in order to give me the opportunity to do my research effectively. I wish to thank Ms. T. Horne for her continued support and assistance, both morally and administratively, without whom I would not have been able to complete this research work. Thanks to Kody Moodley for his introduction and sample source code for a Protégé plug-in, which assisted me to quickly and effortlessly adapt the algorithms presented and integrate them into a plug-in for the Protégé 4.1 environment. My thanks also go out to the members of the education office, especially Mr. Maseko, Ms. April and Mr. Hlatswhayo. I would also like to thank Chad Berry and Gabriel Matsimbe for being patient friends and sounding boards and their invaluable help with proof reading my work. Lastly I would like to thank all those authors whose work this dissertation directly or indirectly uses or extends. Without your work this dissertation would not ever have been possible.

Contents

Chapter 1

Introduction

1.1 From Hieroglyphs to Knowledge-Based Systems

From the beginning of human history mankind understood the need to capture, reproduce and disseminate domain knowledge. Not only did mankind's survival depend on accurately transferring knowledge and skills from generation to generation, but knowledge reuse has been the crux of technological progress throughout history.

The advent of the Information Age has seen an exponential growth in the quantity of information produced and stored. The sheer volume of information produced daily has made computers the preferred tool to store, share and process information while networking and communication technology further allows users to share these resources and collaborate on projects from opposite sides of the globe.

However, as volumes of information increase, the mere storing of information has become insufficient and man thus endeavours to create tools that are more and more 'intelligent', where intelligent, in this context, refers to the computer's ability to accept core descriptions and relationships about a domain *(explicit expert knowledge)* and then reason over this knowledge and produce implicit consequences inherent in it. Such systems are considered to be 'Knowledge-Based' systems (Baader et al. 2007) and lead to what today is known as KR (Knowledge Representation) systems that aim to capture expert domain knowledge and allow computers to reason over this knowledge.

Earlier KR systems can be categorized into two main categories: Logic-based systems and Network-based system. Logic-based systems usually have their basis in first-order logic and reasoning amounts to verifying and determining logical consequences. Network-based systems[1] such as *Semantic Networks* (Quillian 1967) and *Frames* (Minsky 1975), on the

[1]Semantic networks and frames differ greatly, but can be seen as networks, where nodes represent sets of individuals and arcs the relationships between them.

other hand, had a more intuitive approach based on research into human cognitive abilities. These systems used *ad hoc* data structures to represent data with similarly *ad hoc* reasoning algorithms for reasoning over these structures.

The problem with network-based systems was that they lacked precise semantics, the result being that every system behaved differently from the others, even with identical relationship names and virtually identical looking components. Researchers therefore began looking into ways to add precise semantics to network-based systems. Hayes (1979) showed that both semantic networks and frames could be given semantics by relying on First-Order Logic (FOL). Though logic could be used as a basis for clearly specifying semantics in these networks, they do not require the full power of FOL, but rather different fragments of it (Brachman & Levesque 1985). Thus specialized reasoning techniques could be used for reasoning over these representational languages, which are subsets of FOL, without the need for using full first-order logic theorem provers. The complexity of reasoning would thus be proportional to the expressivity of the language.

This work lead directly to the introduction of the ancestor of all Description Logics (DL) systems, KL-One (Brachman & Schmolze 1985), which signalled the transition from semantic networks to more well-founded terminological (description) logics. The influence of KL-One was profound and it is considered the root of the entire family of description languages (Woods & Schmolze 1992).

One of the strengths of DL systems lies in the fact that the reasoning complexity of a DL is directly proportional to the complexity of the constructors available to it and thus proportional to the complexity of the knowledge that may be modelled by it. This allows developers to select appropriate DLs on a per domain basis, thereby not only allowing for the efficient representation of knowledge, but also for using the most efficient reasoning algorithms suitable to the complexity of the knowledge being modelled.

1.2 Motivation and Dissertation Outline

For even the most inexpressive DLs, as the size of ontologies increase to hundreds of thousands or even millions of axioms, serious difficulties arise (Noy & Musen 2004, Stuckenschmidt & Klein 2004). Amongst others these include:

- the running time of reasoning services increases proportionally to the size of the ontology;

- reusing and sharing whole ontologies when only small specific subsets are needed at any one time is inefficient;
- large ontologies introduce a cognitive challenge, necessitating large teams of experts to compile, maintain and evolve the ontology, and
- finding axioms responsible for inconsistencies in large ontologies and consequently debugging these ontologies becomes extremely difficult and time consuming.

In order to address these issues the notion of modularization has been introduced in the literature. Intuitively, modules are small self contained sub-ontologies preserving a specific subtopic or statement of interest. Various formal specifications of modules have been introduced in the literature, each specification dependant upon the exact problem modularization is to address.

In this dissertation we investigate modularization for the inexpressive $\mathcal{EL}$ family of Description Logics. In particular, we investigate modularization as applied to the problem of optimizing reasoning tasks. We focus on two reasoning services:

- subsumption testing between individual concept names. That is, given an ontology $\mathcal{O}$ and the subsumption statement $A \sqsubseteq B$, where A and B are individual concept names, we test whether $\mathcal{O} \models A \sqsubseteq B$, and
- the debugging of ontologies, specifically the extraction of justifications for inconsistent entailments. That is, given that there exists an incorrect subsumption entailment $\mathcal{O} \models A \sqsubseteq B$, we find all the minimal sets of statements that causes this entailment to hold.

The reachability-based module extraction method (Suntisrivaraporn 2009) has been criticised by Du et al. (2009) for considering only the sub-concept in a subsumption based entailment, thereby providing little help in finding justification for entailments. We aim to address this shortcoming by introducing a top-down reachability-based module extraction method that considers only the super-concept in an entailment. We then introduce a novel bidirectional reachability-based algorithm that considers both the sub- and super-concepts in a subsumption statement, thus reducing the size of standard reachability-based modules by removing irrelevant axioms.

Next we investigate the application of minimal bidirectional reachability-based sub-modules to the problem of finding justifications for entailments. We introduce an indexed bidirectional

reachability-based module extraction algorithm derived from the Earley algorithm for parsing with Context Free Grammars, and show how the resulting modules may be used to extract all justifications (Minimal Axioms Sets) for entailments.

The dissertation is structured as follows: Chapter 2 introduces the basic background to Description Logics and the concepts, definitions and terminology used throughout this dissertation. The chapter also introduces the $\mathcal{EL}$ family of Description Logics, with a brief overview of both tractable as well as intractable extensions to it.

Chapter 3 introduces the notion of module extraction. We discuss some of the common modularization methods including an overview of reachability-based module extraction. We then introduce the top-down reachability-based modularization method which extracts modules based on the super-concept of a subsumption statement. Section 3.4 combines reachability-based modules with the top-down version to introduce the notion of bidirectional reachability-based modules.

Chapter 4 investigates the debugging of ontologies, more specifically finding justifications (MinAs) for entailments. We investigate the properties of MinAs for the Horn Logic equivalent DL $\mathcal{HL}$, and derive an indexed based bidirectional module extraction algorithm from the Earley parser for Context Free Grammars. We show that every minimal bidirectional reachability-based sub-module for $\mathcal{HL}$ corresponds to a MinA and provide an algorithm to extract all such MinAs. We then investigate the properties of $\mathcal{EL}$ MinAs and show that the algorithms for $\mathcal{HL}$ may also be used to extract all MinAs for $\mathcal{EL}$ TBoxes consisting of only primitive concept definitions. Lastly we investigate MinA extraction using bidirectional reachability-based sub-modules for general $\mathcal{EL}$ TBoxes.

Chapter 5 deals with the practical implementation and empirical results of the algorithms introduced in the previous chapters. We compare bidirectional reachability-based modules with standard reachability-based modules for SNOMED (Cote et al. 1993), NCI and GO ontologies[2]. Lastly we investigate the performance of the MinA extraction algorithms introduced, and summarize our findings.

[2]We used versions obtained from http://lat.inf.tu-dresden.de/systems/cel/

Chapter 2

Background

In the scope of this dissertation we define a Description Logic system (DL) as a Knowledge Representation (KR) system based on Description Logics. We define a DL system to consist of the following components:

- Description Language - Set of all possible concept descriptions formed from:
  - Set of logical constructors
  - RN - Set of role names
  - CN - Set of concept names
  - IND - Set of all individual names
- Ontology
  - Terminology - A set of axioms constructed by combining concept descriptions with Ontological or Terminological constructors (TBox)
  - Assertions - A set of assertions about individuals (ABox)
- Reasoning Services

A DL system is comprised of all these components combined, whereas the description logic is composed of only the Description language and the Ontology components. In this chapter we define the above components of a DL system as used throughout this dissertation.

2.1 Description Language

DL systems represent knowledge as concepts and relationships between concepts. We denote the set of all *concept names* in an ontology $\mathcal{O}$ by $CN(\mathcal{O})$, the set of all *role names* (binary

relationships) by $RN(\mathcal{O})$, the set of all individual names by $IND(\mathcal{O})$, $CN(\mathcal{O})^{\top} = CN(\mathcal{O}) \cup \{\top\}$ and $CN(\mathcal{O})^{\bot} = CN(\mathcal{O}) \cup \{\bot\}$. We employ the standard convention and use the capital symbols A and B to denote atomic concepts, C and D to denote concept descriptions and r, s and t to denote role names. Each of these may include subscripts when needed.

A *Description Language* consists of all the legal statements that may be composed of the sets CN and RN using elements from a set of logical constructors. Each of these legal statements is referred to as a *concept description*.

Some common logical constructors, their syntax and respective semantics are listed in Table 2.1. This list is by no means exhaustive and only includes a subset of constructors that are of interest in this dissertation. The last column in the table shows the standard short-hand symbol used later to indicate that this constructor is present within a language.

Name	**Syntax**	**Semantics**	**S**
Concept Names			
top	$\top$	$\Delta^{\mathcal{I}}$	
bottom	$\bot$	$\emptyset$	
Conjunction Operator - $\sqcap$			
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	
Disjunction Operator - $\sqcap$			
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	$\mathcal{U}$
Negation Operator - $\neg$			
atomic negation	$\neg A$	$\Delta^{\mathcal{I}} \setminus A^{\mathcal{I}}$	
full negation (complements)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	$\mathcal{C}$
Existential Quantification Operator - $\exists$			
unqualified existential quantification	$\exists r.\top$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}}\}$	
full existential quantification	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	$\mathcal{E}$
Universal Quantification Operator - $\forall$			
value restriction	$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$	
Number Restriction Operators - $\leq$ and $\geq$			
at-most restriction	$(\leq rn)$, $n \in \mathcal{N}$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$	$\mathcal{N}$
at-least restriction	$(\geq rn)$, $n \in \mathcal{N}$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$	$\mathcal{N}$

Table 2.1: Syntax and semantics of common Description Language constructors

In general, all concept names $A \in CN(\mathcal{O})$, $\bot$ and $\top$ are concept descriptions, using the legal syntax of the relevant logical constructors, more complex concept descriptions are built up from these as shown in the following example:

Example 1 *Let the set {Person, Man} represent concept names, and let {hasChild} be a set containing a single role name. Using the constructors $\{\sqcap, \exists\}$ from Table 2.1, we may construct a concept description to represent the notion of being a parent as follows:*

$$Person \sqcap \exists hasChild.Person$$

Which can be read as "a person that has at least one child who is a person". We may now use it to define the notion of grandfather as:

$$Person \sqcap Man \sqcap \exists hasChild.(Person \sqcap \exists hasChild.Person)$$

Read as "a male person with at least one child who is both a person and has a child that is a person".

Definition 1 (Semantics of concept descriptions) *An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}; .^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$ called the interpretation domain and an interpretation function $.^{\mathcal{I}}$, which assigns to each concept name $A \in CN$ a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and to each role name $r \in RN$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation function is extended to concept descriptions by the inductive semantic definitions given in Table 2.1.*

In general there are two classes of description languages of interest, those languages that form part of the $\mathcal{AL}$ (attributive language) family as well as the sub-Boolean set of languages such as the $\mathcal{EL}$ family. The language $\mathcal{AL}$ is defined as:

Definition 2 (The description language $\mathcal{AL}$) *Let CN represent the set of all concept names, RN the set of all role names then $\mathcal{AL}$ is inductively defined as all legal concept descriptions C such that:*

$$C := A \mid \top \mid \bot \mid \neg A \mid C \sqcap C \mid \exists r.\top \mid \forall r.C$$

Any arbitrary description language $\mathcal{L}$ may be extended by any subset of the additional constructors to yield the appropriate member of the $\mathcal{L}$-language family. Using the constructors from Table 2.1, each $\mathcal{L}$-language is named by a string of the form

$$\mathcal{L}[\mathcal{C}][\mathcal{E}][\mathcal{N}][\mathcal{U}]$$

where a letter in the name stands for the presence of the corresponding constructor. For instance, $\mathcal{LEN}$ is the extension of $\mathcal{L}$ by both full existential quantification and number restrictions.

In terms of the semantics of the constructors, not all of these languages are distinct. The semantics enforces the logical equivalences between $C \sqcup D$ and $\neg(\neg C \sqcap \neg D)$ as well as between $\exists r.C$ and $\neg\forall r.\neg C$ (Baader et al. 2007). Hence, by using full negation we may express disjunction and full existential quantification in terms of conjunction and value restrictions respectively, and vice versa. Therefore, disjunction and full existential quantification are implicitly available in every language that contains *full* negation, conjunction and universal quantification. It follows that all $\mathcal{L}$-languages, containing these constructors, can be written using the letters $\mathcal{U}$, $\mathcal{E}$ and $\mathcal{N}$ only. We may therefore interchange the letters $\mathcal{UE}$ and $\mathcal{C}$ without loss of generality in all these languages.

In Chapter 4 we investigate the DL $\mathcal{HL}$ formally defined as:

Definition 3 (The description language $\mathcal{HL}$) *Let CN represent the set of all concept names, RN the set of all role names. $\mathcal{HL}$ is inductively defined as all legal concept descriptions C such that:*

$$C := A \mid \top \mid C \sqcap C$$

Strictly speaking a description language must contain at least one quantifier (Suntisrivaraporn 2009). Any language that contains no quantifiers is a propositional language and does not strictly form part of the set of description languages. However, in this paper we relax this restriction and investigate the *Horn Logic* language $\mathcal{HL}$, which allows for only the conjunction operator. Though not strictly speaking a description language, we use the term loosely in the case of $\mathcal{HL}$. Note that $\mathcal{HL}$ abbreviates Horn Logic, and that this deviates from the convention that $\mathcal{H}$ refers to role hierarchies (Table 2.2).

2.2 The Ontology

In this section we investigate the notion of an ontology. In general, given a description language $\mathcal{L}$, an ontology $\mathcal{O}$ in $\mathcal{L}$ consists of the following components:

- a set of concept names from $\mathcal{L}$ denoted by $CN(\mathcal{O})$,

- a set of binary relation names (role names) from $\mathcal{L}$ denoted by $RN(\mathcal{O})$,
- a set of individuals $IND(\mathcal{O})$ from $\mathcal{L}$,
- a set of terminological axioms called a TBox denoted by $\mathcal{T}_{\mathcal{O}}$, and
- a set of assertions about individuals called an ABox denoted by $\mathcal{A}_{\mathcal{O}}$.

Using the description language of a DL we may create arbitrary concept descriptions. However, when defining terminologies we require a method to define and reuse concept descriptions or state facts about the domain we are modelling. For instance, in Example 1 we compiled concept descriptions to represent the notions *Parent* and *GrandFather* respectively. Reusing these descriptions whenever we intend to imply the respective concepts can become cumbersome and error prone. A better solution would be to define the concept *Parent* as a concept in its own right. We may do this by using the symbol '$\sqsubseteq$' which is read as *subsumes* meaning *contains* or *includes*. *Parent* can thus be defined as follows:

$$Parent \sqsubseteq Person \sqcap \exists hasChild.Person$$

Thus we may use the new term *Parent* when defining the term *GrandFather* as:

$$GrandFather \sqsubseteq Man \sqcap Parent \sqcap \exists hasChild.Parent$$

Besides defining terms, we may also wish to state that certain concepts are *disjoint*, some roles represent transitive relations or even define role hierarchies. These new statements or *axioms* are formed by utilising a set of ontological/terminological constructors. A set of the most common ontological constructors together with their respective syntax and semantics is listed in Table 2.2.

When axioms are constructed using the ontological constructors where the left hand side of an axiom is an atomic concept we differentiate between *primitive concept definitions* and *concept definitions*:

Definition 4 (Primitive concept definition) *Let A be a concept name and C a concept description. Then, $A \sqsubseteq C$ is a primitive concept definition and A is said to be primitively defined.*

Whereas a concept definition is defined as:

Definition 5 (Concept definition) *Let A be a concept name and C a concept description. Then, $A \equiv C$ is a concept definition and A is said to be fully defined.*

Name	Syntax	Semantics
Concept Definition	$A \equiv C$	$A^{\mathcal{I}} = C^{\mathcal{I}}$
Concept Inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
Concept Disjointness	$C \sqcap D \sqsubseteq \bot$	$C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$
Role Hierarchy	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
Role Inclusion	$r_1 \circ \ldots \circ r_k \sqsubseteq r$	$r_1^{\mathcal{I}} \circ \ldots \circ r_k^{\mathcal{I}} \sqsubseteq r^{\mathcal{I}}$
Transitive Roles	transitive(r)	$\forall x, y, z \in \Delta^{\mathcal{I}} : (x, y), (y, z) \in r^{\mathcal{I}} \rightarrow (x, z) \in r^{\mathcal{I}}$
Reflexive Roles	reflexive(r)	$\forall x \in \Delta^{\mathcal{I}} : (x, x) \subseteq r^{\mathcal{I}}$
Range Restrictions	range(r)$\sqsubseteq C$	$\{y \in \Delta^{\mathcal{I}} \mid \exists x : (x, y) \in r^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}$
Domain Restrictions	domain(r)$\sqsubseteq C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x, y) \in r^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}$
Concept Assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
Role Assertion	$r(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

Table 2.2: Syntax and semantics of ontological constructors

A concept name that is neither fully nor primitively defined is called *undefined*[1]. The set of all axioms in a domain constructed from the set of concept names and role names, using a specified description language, is referred to as a terminology or a TBox. There are various forms of TBoxes, these include *unfoldable*, *cyclic* and *general* TBoxes. In order to define the various forms of TBoxes we need to define the *Uniqueness Condition* and *Cyclicity*:

Definition 6 (Uniqueness condition (Baader et al. 2007)) *Let $\mathcal{S}$ be a finite set of axioms of the form $A \sqsubseteq C$ or $A \equiv C$. Then $\mathcal{S}$ satisfies the uniqueness condition if, and only if, for each concept name A, there is at most one axiom $A \sqsubseteq C \in \mathcal{S}$ or $A \equiv C \in \mathcal{S}$ for any concept description C.*

Definition 7 (Cyclicity (Baader et al. 2007)) *Let A and B be atomic concepts occurring in a finite set of axioms $\mathcal{S}$. A* directly uses *B in $\mathcal{S}$ if B appears on the right-hand side of the definition of A. The term* uses *refers to the transitive closure of the relation* directly uses*. The set $\mathcal{S}$ contains a cycle, and is called cyclic, if there exists an atomic concept A in $\mathcal{S}$ that uses itself. If $\mathcal{S}$ does not contain a cycle it is called* acyclic*.*

The definition of cyclicity can be extended to axioms as follows: given two axioms $\alpha_1 : \alpha_{L1} \sqsubseteq \alpha_{R1}$ and $\alpha_2 : \alpha_{L2} \sqsubseteq \alpha_{R2}$, we say that α_1 uses α_2 if α_{L2} and α_{R1} contain at least one common symbol.

Example 2 *Given the set $S_1 = \{\alpha_1 : A \sqsubseteq B, \alpha_2 : B \sqsubseteq A\}$ then in axiom α_1 A directly uses B, S_1 is cyclic since A uses itself in S_1, and α_1 uses α_2 because of the common symbol*

[1]Some papers refer to both fully and primitively defined concepts as simply defined concepts, and undefined concepts are referred to as atomic concepts.

B. Given the set $S_2 = \{\alpha_3 : A \sqsubseteq B, \alpha_4 : B \sqsubseteq C\}$ A directly uses B, S_2 is acyclic since no concept uses itself. Lastly α_3 uses α_4 because of the common symbol B.

We now formally define a TBox or terminology as follows:

Definition 8 (TBox) *A TBox $\mathcal{T}$ is a finite set of (possibly primitive) concept definitions such that the uniqueness condition holds. When the TBox contains no cycles it is called an* acyclic *or an* unfoldable *TBox.*

The terms unfoldable and acyclic are usually used synonymously as reasoning problems with respect to these TBoxes may be systematically transformed, by a process called unfolding, into equivalent reasoning problems with respect to an empty TBox. Therefore, reasoning with these TBoxes is equivalent to reasoning with concept descriptions and is often used during the development stages of reasoning algorithms for testing purposes (Baader et al. 2007).

Definition 9 (General TBox) *Let C and D be concept descriptions. Then $C \sqsubseteq D$ is a (general) concept inclusion (GCI) axiom. A general TBox is a finite set of GCIs.*

General TBoxes may contain axioms of the form $C \equiv D$, where C and D are concept names or concept descriptions, as shorthand notation for the set of axioms $\{C \sqsubseteq D, D \sqsubseteq C\}$. A TBox containing no GCIs may be transformed to a general TBox by noting that primitive concept definitions are special forms of GCIs where the left hand side is a single concept name instead of a complex concept description.

Definition 10 (ABox) *Let IND be a set of individuals disjoint from CN and RN. Then, expressions of the forms $C(a)$ and $r(a; b)$ are called concept assertions and role assertions, respectively, where $a, b \in IND$, $r \in RN$, and C is a concept description. An ABox is a finite set of concept and role assertions.*

An ontology is defined as containing both a (general) TBox and an ABox. However, many terminologies such as SNOMED, GO and NCI used during the course of this dissertation consist of only a terminological component. Many of the reasoning services dealt with later on are performed over TBoxes, and all of the work we introduce in later chapters deals with only the TBox component of an ontology. We therefore use a less formal definition of an ontology, and use the term Ontology and TBox synonymously throughout this text; where the inclusion of an ABox is required it will be noted explicitly. Further in this dissertation,

whenever we refer to arbitrary axioms within a TBox we use the Greek symbols α, σ, β. Where we want to refer explicitly to the sub-concept and super-concept in an axiom, we may also write $\alpha_L \sqsubseteq \alpha_R$.

Following the definition of semantics for a concept description, we may now use Table 2.2 to extend the interpretation function to cover individual axioms and thus the Ontology $\mathcal{O}$.

Definition 11 (Semantics of an Ontology) *Let $\mathcal{O}$ be an ontology, α an axiom and $\mathcal{I}$ an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ as defined in Definition 1. Then we say that $\mathcal{I}$* satisfies *α, written as $\mathcal{I} \models \alpha$, if and only if, the semantic conditions for α from Table 2.2 is fulfilled under $\mathcal{I}$. If $\mathcal{I}$ satisfies all axioms $\alpha \in \mathcal{O}$ we say that $\mathcal{I}$ is a* model *of $\mathcal{O}$.*

During the course of this work, we often refer to certain properties of ontologies. These include the number of axioms in an ontology $\mathcal{O}$, denoted by $\mid \mathcal{O} \mid$, as well as the set of symbol and role names occurring in an axiom or ontology, defined as:

Definition 12 (Signature of an ontology) *Let $\mathcal{O}$ be an ontology, let $CN(\mathcal{O})$ represent the set of all concept names in $\mathcal{O}$ and $RN(\mathcal{O})$ the set of all role names in $\mathcal{O}$. We define the signature of $\mathcal{O}$, denoted as $Sig(\mathcal{O})$, as the union of all concept and role names occurring in $\mathcal{O}$ i.e., $Sig(\mathcal{O}) = CN(\mathcal{O}) \cup RN(\mathcal{O})$. Similarly, for* any *DL statement σ, $Sig(\sigma)$ is the union of all concept and role names occurring in σ. Here a statement σ is defined as any concept definition or GCI.*

2.3 Reasoning Services for Description Logics

Reasoning services for DLs may be divided into two main categories:

- Standard Reasoning Services - Common reasoning services most DLs are expected to provide, and
- Supplementary Reasoning Services - Reasoning Services that aid in the design and maintenance of DLs.

Since the focus of this dissertation is on modularization, a supplementary reasoning service, Section 2.3.1 gives an overview of standard reasoning services for $\mathcal{EL}$. We show how these may be implemented using implication sets and investigate both those extensions to $\mathcal{EL}$ for which these reasoning services remain tractable as well as those extensions for which reasoning degrades to exponential complexity.

2.3.1 Standard Reasoning Services

In this section we investigate the standard reasoning services all DL systems are expected to provide. Just as DLs vary in expressivity, so do different DL systems vary in the reasoning services they provide. There are however reasoning services that are considered standard and thus mandatory for any DL system. Each of these addresses some logical inference problem and provides a method for making implicit knowledge, inherent in the ontology, explicit. These reasoning services include the following (Baader et al. 2007):

- *Concept satisfiability:* Let $\mathcal{O}$ be an ontology and C, D concept descriptions. C is satisfiable with respect to $\mathcal{O}$ if there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that $C^{\mathcal{I}} \neq \emptyset$. Otherwise, C is unsatisfiable with respect to $\mathcal{O}$. Further, two concepts C and D are said to be disjoint with respect to $\mathcal{O}$ if their conjunction $C \sqcap D$ is unsatisfiable with respect to $\mathcal{O}$.

- *Concept subsumption:* Let $\mathcal{O}$ be an ontology and C, D concept descriptions. D subsumes C with respect to $\mathcal{O}$, written as $\mathcal{O} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{O}$. If $\mathcal{O} \models C \sqsubseteq D$ but $\mathcal{O} \not\models D \sqsubseteq C$, then we say that D *strictly subsumes* C, written as $\mathcal{O} \models C \sqsubset D$. Here C is usually referred to as the *subsumee* or *sub-concept* whilst D is referred to as the *subsumer* or *super-concept*.

- *Instance checking:* Let $\mathcal{O}$ be an ontology, C a concept description, r a role name and a, b individuals. The individual a is an instance of C w.r.t $\mathcal{O}$, written as $\mathcal{O} \models C(a)$, if and only if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{O}$. In terms of roles, the pair of individuals (a, b) is an instance of r w.r.t $\mathcal{O}$, written as $\mathcal{O} \models r(a, b)$, if and only if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ for all models $\mathcal{I}$ of $\mathcal{O}$.

- *Consistency checking:* Let $\mathcal{O}$ be an ontology. $\mathcal{O}$ is consistent if and only if it has a model; otherwise it is inconsistent. The ABox $\mathcal{A}$ is consistent with respect to the TBox $\mathcal{T}$ if and only if $\mathcal{A}$ and $\mathcal{T}$ have a common model $\mathcal{I}$.

Instead of providing algorithms for each of these reasoning services, Baader et al. (2005) shows that all of these reasoning services are mutually inter-reducible to each other except for sub-Boolean logics such as the $\mathcal{EL}$ family. Since all these reasoning services are therefore reducible to subsumption testing, and that subsumption testing is the reasoning problem of choice in this dissertation, we will, for the rest of this chapter, focus on reasoning algorithms for the subsumption problem only. Strictly speaking, the algorithms presented later compute not only a single subsumption relation, but subsumption relationships between all pairs of

concept names occurring in the ontology. Algorithms of this type are called *classification* algorithms.

2.3.2 Reasoning Complexity

When implementing reasoning services it is important to be able to measure the performance of the algorithms in some universal way. A universal method is required since algorithms may be implemented in different programming languages, run on different hardware platforms, even including small mobile devices, and be implemented using totally different runtime strategies.

In general there are two methods used when evaluating the performance of algorithms. The first, and most prevalent, is the running time of the algorithm whereas the second measures the runtime space requirements of the algorithm. All problems considered in this dissertation are *decidable*, that is, there exists in principle an algorithm that may be used to solve it irrespective of its complexity, in contrast to *undecidable* problems for which no possible algorithm exists, nor may there ever be one constructed, irrespective of the advances in technology.

We measure the complexity of an algorithm as the number of *elementary* or *unit* operations required to successfully complete a run of the algorithm. This measurement is expressed as a function of the *size of the input* of the algorithm, where the *input* is restricted to being a sequence of objects and the *size* of the input to being the number of objects in this sequence.

Though reasoning problems can be divided into many different classes such as: polynomial(P), non-deterministic polynomial(NP), complement of a non-deterministic polynomial (co-NP), exponential(EXPTime) and polynomial space (PSpace) problems, for the purposes of this dissertation we do not provide an in depth discussion on these but refer the interested reader to Papadimitriou & Steiglitz (1989) for an in-depth discussion on algorithmic complexity.

Loosely speaking, the class P consists of problems for which algorithms exists whose runtime complexity can be expressed as a polynomial function in the size of the input. For example, given that n is the size of the input, then

$$f(n) = n, \quad f(n) = n^2, \quad f(n) = 2 \times n^3 + 3 \times n$$

are all polynomial functions. Other algorithms for problems in P with runtime complexity not strictly polynomial but bounded by a polynomial are considered to be solutions to problems

in P. These include functions such as:

$$f(n) = \log n, \quad f(n) = n \log n, \quad f(n) = n^{2.1}$$

Runtime complexity of algorithms for problems in the class EXPTime can be expressed as an exponential function in the size of the input. Given that n is the size of the input, then

$$f(n) = 2^n, \quad f(n) = n^{\log n}, \quad f(n) = n!$$

are all exponential functions.

Problems in P are usually referred to as *tractable* problems whereas those outside of P are referred to as *intractable*. Unless P = NP, the runtime complexity of all algorithms for problems not in P, such as those in NP, coNP and EXPTime, can usually be expressed as an exponential function in the size of the input (Papadimitriou & Steiglitz 1989).

Definition 13 (Polynomial reduction (Papadimitriou & Steiglitz 1989)) *Let P_1 and P_2 be two reasoning problems, and let I_1 be the input for a successful solution to P_1. We say that P_1* polynomially reduces *to P_2 if and only if I_1 can be transformed in polynomial time to I_2, such that I_2 is the input for a successful solution to P_2.*

Let λ be a complexity class, then we say that a reasoning problem R is λ-*hard* if we can show that all other problems in λ can be polynomially reduced to R. When we can also show that $R \in \lambda$ then we say that R is λ-*complete*.

The class PSpace can be defined as all problems for which algorithms exists that require runtime space bounded by a polynomial function in the size of the input. Since an algorithm running in polynomial time cannot require more than polynomial space, we know that problems in P are also in PSpace. Papadimitriou & Steiglitz (1989) showed that NP is in PSpace, by showing that only one non-deterministic polynomial solution needs to occupy runtime space at any given time. In contrast, the class EXPSpace is defined as all problems for which algorithms exists that require runtime space bounded by an exponential function in the size of the input. Figure 2.1 illustrates the relationships between the classes P, PSpace, EXPTime and EXPSpace.

2.3.3 Viability of Inexpressive DLs

When clarifying the status of property arcs in Semantic Networks and slots in Frames, the decision was taken that they should be represented by universal quantifications ($\forall$) instead of

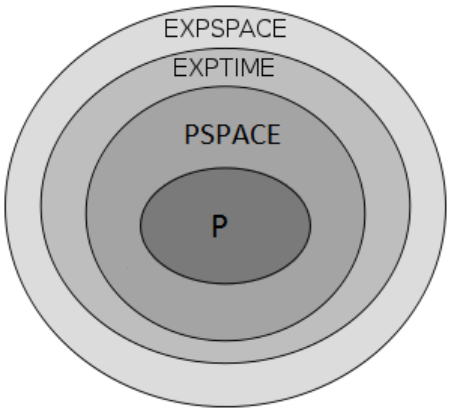

Figure 2.1: Complexity classes

existential quantifications ($\exists$). Most DLs are therefore based on a subset of FOL that includes universal quantification, the smallest of which is the sub-Boolean DL $\mathcal{FL}_0$, which allows only for the conjunction of concepts ($C \sqcap D$) and value restrictions ($\forall r.C$)[2]. The complexity of reasoning in $\mathcal{FL}_0$ is thus very important, since the worst case complexity in this simple logic, sets the lower bound for the reasoning complexity of all logics that extend it.

Terminology	Complexity
concept descriptions (empty TBox)	Polynomial (Brachman & Levesque 1984)
acyclic TBoxes (unfoldable TBox)	coNP-complete (Nebel 1990)
cyclic TBoxes	PSpace-complete (Baader 1996, Kazakov & Nivelle 2003)
TBoxes with GCI's (general TBox)	EXPTime-complete (Baader et al. 2005)

Table 2.3: Complexity of the subsumption problem in $\mathcal{FL}_0$

From Table 2.3.3, we see that as soon as reasoning is to be performed over terminologies more complex than simple concept descriptions, the complexity of reasoning becomes hard. Further, there is a direct correlation between the complexity of axioms allowed in a terminology and hardness of reasoning over it. Since reasoning over acyclic TBoxes is hard, all DLs that extend $\mathcal{FL}_0$ also have the same minimum hardness constraints. Researchers therefore focused on more complex DLs, most including *General Concept Inclusions (GCI's)* with as many constructors as possible, and finding practical reasoning algorithms (easy to implement and optimize) for them. This research lead to the introduction of highly optimized *tableau* algorithms for standard reasoning services (Horrocks 1998, Haarslev & Möller 2001). Though these algorithms have theoretical exponential or worse, worst case behaviour, they perform well in practice (Horrocks et al. 2000, Horrocks & Sattler 2004).

[2]The term value restriction is synonymous with universal quantification.

Reasoning with General Concept Inclusions

From the previous section we see that reasoning with terminologies containing General Concept Inclusions (GCI's) is hard for even the most inexpressive of DLs based on the universal quantification operator. Recall from Definition 9 that GCI's are axioms that contain arbitrary concept descriptions on both sides of the terminological operator '$\sqsubseteq$'

Reasoning with general TBoxes is EXPTime-Complete for the DL $\mathcal{FL}_0$ and any logic that extends it. This complexity result would suggest that during the design of an ontology the use of GCI's should as far as possible be eliminated. However, from an application point of view, the utility of GCI's has long been observed. For example, in the *GALEN* (Rector 2007) medical ontology GCI's are used for two main purposes (Brandt 2004*a*):

- indicating the status of objects: Instead of introducing different concepts for the same concept in different states, for example: normal insulin secretion, abnormal but harmless insulin secretion, and pathological insulin secretion, only insulin secretion is defined while the status, i.e., normal, abnormal but harmless, and pathological is implied by GCIs of the form $\ldots \sqsubseteq \exists has\_status.pathological$.

- to bridge levels of granularity and add implied meaning to concepts. A classical example (Horrocks et al. 1996) is to use a GCI like

$$ulcer \sqcap \exists has\_loc.stomach \sqsubseteq ulcer \sqcap \exists has\_loc.(lining \sqcap \exists is\_part\_of.stomach)$$

  to render the description of 'ulcer of stomach' more precisely to 'ulcer of lining of stomach' if it is known that 'ulcer of stomach' is specific of the lining of the stomach.

Research on reasoning with respect to general TBoxes has mainly focused on very expressive DLs, such as $\mathcal{ALCNR}$ and $\mathcal{SHIQ}$, in which deciding subsumption of concepts with respect to general TBoxes is EXPTime-hard. Givan et al. (2002) shows the problem to remain EXPTime-complete for a DL providing only conjunction, value restriction and existential restriction. The same holds for the small DL $\mathcal{AL}$ which allows for conjunction, value and unqualified existential restriction, and primitive negation (Donini 2007). From Table 2.3.3, we see that for the simple DL $\mathcal{FL}_0$, which only allows for conjunction and value restriction, subsumption with respect to cyclic TBoxes with descriptive semantics is PSpace-hard (Kazakov & Nivelle 2003), and is EXPTime-complete for general TBoxes (Baader et al. 2005).

The DL $\mathcal{EL}$

It was shown that for the DL $\mathcal{EL}$, a small sub-boolean DL allowing only existential restrictions and concept conjunction, the subsumption problem with respect to cyclic terminologies as well as general TBoxes can be decided in polynomial time (Baader 2003, Brandt 2004*a*). The relatively low complexity of reasoning with $\mathcal{EL}$ as well as the fact that it has practical applications, especially in medical ontologies, serves as a strong motivation for considering the DL as an alternative where value restrictions are not an explicit requirement.

For example, in the medical field the widely used medical terminology *SNOMED* (Cote et al. 1993) corresponds to an $\mathcal{EL}$ TBox (Spackman 2001) and the representation language underlying the medical terminology *GALEN* (Rector 2007) in which GCIs are used extensively, similarly can be represented by a general $\mathcal{EL}$ TBox with added role inclusion axioms.

Terminology	Complexity
concept descriptions (empty TBox)	polynomial (Baader et al. 1998)
acyclic TBoxes (unfold-able TBox)	polynomial (Baader 2003)
cyclic TBoxes	polynomial (Baader 2003)
TBoxes with GCI's (general TBox)	polynomial (Brandt 2004*a*)

Table 2.4: Complexity of the subsumption problem in $\mathcal{EL}$

In Table 2.3.3 we clearly see that reasoning in $\mathcal{EL}$ remains tractable even for the most general kind of TBoxes. $\mathcal{EL}$ is therefore not only tractable but also lends itself to practical applications. We therefore make a distinction between *expressive* and *inexpressive* languages, where inexpressive languages refers to those languages not based on $\mathcal{FL}_0$ for which polynomial-time worst-case reasoning algorithms exist.

The polynomial complexity of reasoning tasks for $\mathcal{EL}$, taken together with its practical utility in especially the medical field, where value restrictions are not an explicit requirement, has renewed interest in this inexpressive sub-boolean language. In the rest of this dissertation we will mainly focus our attention on the $\mathcal{EL}$-family of DLs, with the base language $\mathcal{EL}$ defined as:

Definition 14 (Syntax of $\mathcal{EL}$) *Let CN and RN be two disjoint sets of concept and role names, respectively. Then, $\mathcal{EL}$ concept descriptions or concepts are defined inductively as follows:*

- *each concept name $A \in CN$ is an $\mathcal{EL}$ concept description, and*
- *if C and D are $\mathcal{EL}$ concept descriptions and $r \in RN$ is a role name, then the top*

concept $\top$, concept conjunction $C \sqcap D$, and existential quantification $\exists r.C$ are also $\mathcal{EL}$ concept descriptions.

An $\mathcal{EL}$ concept description is atomic *if it is the top concept or a concept name from CN. Otherwise, it is said to be complex.*

As with any description language, extending $\mathcal{EL}$ with additional constructions yields different languages in the $\mathcal{EL}$ family. Some of the members of the $\mathcal{EL}$ family may be defined as follows :

- $\mathcal{EL}^{\neg}$ is defined as $\mathcal{EL}$ extended with negation of concept descriptions,
- $\mathcal{EL}^{(\neg)}$ is defined as $\mathcal{EL}$ extended with atomic negation,
- $\mathcal{ELU}$ is defined as $\mathcal{EL}$ extended with the disjunction of concept descriptions ($C \sqcup D$),
- $\mathcal{ELN}$ is defined as $\mathcal{EL}$ extended with both at-most number restrictions ($\leq\ r\, n$) and at-least number restrictions($\geq\ r\, n$).

Here $\mathcal{EL}^{\neg}$ represents $\mathcal{EL}$ extended with full negation, and may, in conformance with the above notational convention, be represented by $\mathcal{ELC}$. However, using the semantic equivalences between $\neg(C \sqcap D)$ and $\neg C \sqcup \neg D$ as well as that between $\exists r.\neg C$ and $\neg\forall r.C$, we see that as soon as full negation is added to $\mathcal{EL}$ it becomes a notational variant of the language $\mathcal{ALC}$. As a result the tractable nature of $\mathcal{EL}$ is lost and the resultant DL will inherit the minimum complexity results from Table 2.3.3. Therefore, when extending a language with additional constructors whilst maintaining the complexity of the original language, care should be taken so that the combination of constructors used preserves the core complexity of the language, and not extend it to a language with completely different complexity results.

Baader (2003) has shown that subsumption in $\mathcal{EL}$ with respect to GCI's can be decided in polynomial time. The question arises whether we may simply use existing tableaux based algorithms designed for more expressive DLs or if new algorithms are required.

2.3.4 Subsumption Testing in $\mathcal{ELH}$

Given an $\mathcal{EL}$ TBox $\mathcal{T}$ and the subsumption entailment $\mathcal{T} \models C \sqsubseteq D$, in order to decide if the subsumption entailment holds, an intuitive decision procedure to choose would be the $\mathcal{ALC}$ tableaux algorithm deciding consistency of $\mathcal{ALC}$-concepts with respect to $\mathcal{ALC}$ terminologies (Baader et al. 2003). The DL $\mathcal{ALC}$ extends $\mathcal{EL}$ by value restrictions ($\forall$), disjunction ($\sqcup$) and

negation($\neg$). The entailment $\mathcal{T} \models C \sqsubseteq D$ can thus be decided by deciding the satisfiability of $C \sqcap \neg D$. However, Brandt (2004*b*) shows that using tableaux based algorithms for more expressive DLs like $\mathcal{ALC}$ in order to reason over less expressive DLs like $\mathcal{EL}$ does in fact not reduce the exponential worst case running time of these algorithms and that specialized algorithms are needed to utilize the reduced complexity of $\mathcal{EL}$.

In this section we illustrate the standard approach used to decide the subsumption problem for the tractable subset of the $\mathcal{EL}$ family, listed in Table 2.5[3], by outlining the algorithm as applied to $\mathcal{ELH}$.

Name	$\mathcal{EL}$	$\mathcal{ELH}$	$\mathcal{ELH}^{\perp}$	$\mathcal{EL}^{+}$
top	×	×	×	×
bottom			×	×
conjunction	×	×	×	×
existential restriction	×	×	×	×
GCI	×	×	×	×
RH (role hierarchy)		×	×	×
RI (role inclusion)				×
transitive roles				×
range restrictions				×
domain restrictions				×
reflexive roles				

Table 2.5: Tractable subset of the $\mathcal{EL}$ family of DLs

In order to limit the use of complex concept descriptions to the most basic cases, the notion of a normalized TBox is introduced. In the case of $\mathcal{ELH}$ a normalized TBox is defined as:

Definition 15 (Normalized $\mathcal{ELH}$ TBox (Brandt 2004*a*)) *Let $\mathcal{T}$ be an $\mathcal{ELH}$-TBox over CN and RN. $\mathcal{T}$ is normalized if:*

1. *$\mathcal{T}$ contains only GCIs and role hierarchies $r \sqsubseteq s$ and,*
2. *all of the GCIs have one of the following forms:*

$$\begin{aligned} A &\sqsubseteq B \\ A_1 \sqcap A_2 &\sqsubseteq B \\ A &\sqsubseteq \exists r.B \\ \exists r.A &\sqsubseteq B \end{aligned}$$

where A, A_1, A_2, B represent concept names from CN, and r, s roles names from RN.

[3]We note that, for the purposes of this dissertation, our definition of $\mathcal{EL}^{+}$ in Table 2.5 excludes reflexive roles.

The normal form can be computed in linear time and does not increase the size of the TBox more than linearly (Brandt 2004*b*). To achieve the normalization the following normalization rules can be applied to an arbitrary $\mathcal{ELH}$ TBox.

Definition 16 (Normalization rules for an $\mathcal{ELH}$ TBox (Brandt 2004*a*)) *Let $\mathcal{T}$ be an $\mathcal{ELH}$-TBox over CN and RN. For every $\mathcal{ELH}$-concept description C, D, E over $CN(\mathcal{T})^\top$ and $RN(\mathcal{T})$, the $\mathcal{ELH}$-normalization rules are defined modulo commutativity of conjunction ($\sqcap$) as follows:*

$$
\begin{array}{lrcl}
NF1 & C \equiv D & \rightarrow & \{C \sqsubseteq D, D \sqsubseteq C\} \\
NF2 & \hat{C} \sqcap D \sqsubseteq E & \rightarrow & \{A \sqsubseteq \hat{C}, A \sqcap D \sqsubseteq E\} \\
NF3 & \exists r.\hat{C} \sqsubseteq D & \rightarrow & \{A \sqsubseteq \hat{C}, \exists r.A \sqsubseteq D\} \\
NF4 & C \sqsubseteq \exists r.\hat{D} & \rightarrow & \{C \sqsubseteq \exists r.A, A \sqsubseteq \hat{D}, \} \\
NF5 & C \sqsubseteq D \sqcap E & \rightarrow & \{C \sqsubseteq D, C \sqsubseteq E\}
\end{array}
$$

Here C, D, E represent concept names. $\hat{C}, \hat{D}$ denote complex concept descriptions, that is, concept descriptions that do not consist of single concept names only. A denotes a new concept name not from CN.

Applying a rule $R := G \rightarrow S$ to $\mathcal{T}$ changes $\mathcal{T}$ to $(\mathcal{T} \setminus G) \cup S$. The normalized TBox norm($\mathcal{T}$) is defined by exhaustively applying Rules NF1 to NF3 and, after that, exhaustively applying Rules NF4 and NF5.

The size of $\mathcal{T}$ is increased only linearly by exhaustive application of Rule NF1. Since this rule never becomes applicable as a consequence of Rules NF2 to NF5, we may restrict our attention to Rules NF2 to NF5. A single application of one of the Rules NF2 to NF3 increases the size of $\mathcal{T}$ only by a constant, introducing a new concept name and splitting one GCI into two. Exhaustive application therefore produces an ontology of linear size in the size of $\mathcal{T}$. After exhaustive application of Rules NF1 to NF3, the left-hand side of every GCI is of constant size. Hence, applying Rules NF4 and NF5 exhaustively similarly yields an ontology of linear size in $\mathcal{T}$. Consequently, the following lemma holds.

Lemma 1 *(Brandt 2004*a*) The normalized TBox norm($\mathcal{T}$) can be computed in linear time in the size of $\mathcal{T}$. The resulting ontology is linear in the size of $\mathcal{T}$.*

Once the TBox has been normalized, the next part of the classification algorithm is to compute for every concept $A \in CN^{\top}$ a set of concepts $S_{\mathcal{T}}(A) \subseteq CN^{\top}$ with the following property: for all models $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ of $\mathcal{T}$ and all individuals $a \in \Delta^{\mathcal{I}}$, if $a \in A^{\mathcal{I}}$ then for every concept name $B \in S_{\mathcal{T}}(A)$ we have that $a \in B^{\mathcal{I}}$. The simple structure of an $\mathcal{ELH}$-TBox in normal form allows us to define an implication set as follows.

Definition 17 (Implication sets for concept names (Brandt 2004*a*)) *Let $\mathcal{T}$ denote a normalized $\mathcal{ELH}$-TBox $\mathcal{T}$ over CN and RN. For every $A \in CN(\mathcal{T})^{\top}$, the* implication set *$S_{\mathcal{T}}(A)$ is defined by $\bigcup_{n\geq 0} S_n(A)$ where the sets S_n are inductively defined on n such that $S_0(A) := \{A, \top\}$. If $S_n(B)$ is already defined for all concept names $B \in CN^{\top}$, then $S_{n+1}(A)$ is the result of an exhaustive application of the extension rules in Fig 2.2.*

Definition 18 (Implication sets for role names (Brandt 2004*a*)) *Let $\mathcal{T}$ denote a normalized $\mathcal{ELH}$-TBox $\mathcal{T}$ over CN and RN. For every $r \in RN(\mathcal{T})$, the* implication set *$S_{\mathcal{T}}(r)$ is defined by $\bigcup_{n\geq 0} S_n(r)$ where the sets S_n are inductively defined on n such that $S_0(r) := \{r\}$. If $S_n(r)$ is already defined for all role names $s \in RN$, then $S_{n+1}(r)$ is the result of an exhaustive application of the extension rules in Fig 2.2.*

ISR	If $s \in S_i(r)$ and $s \sqsubseteq t \in \mathcal{T}$ and $t \notin S_{i+1}(r)$ then $S_{i+1}(r) := S_{i+1}(r) \cup t$
IS1	If $A_1 \in S_i(A)$ and $A_1 \sqsubseteq B \in \mathcal{T}$ and $B \notin S_{i+1}(A)$ then $S_{i+1}(A) := S_{i+1}(A) \cup B$
IS2	If $A_1, A_2 \in S_i(A)$ and $A_1 \sqcap A_2 \sqsubseteq B \in \mathcal{T}$ and $B \notin S_{i+1}(A)$ then $S_{i+1}(A) := S_{i+1}(A) \cup B$
IS3	If $A_1 \in S_i(A)$ and $A_1 \sqsubseteq \exists r.B \in \mathcal{T}$ and $B_1 \in S_i(B)$ and $s \in S_i(r)$ and $\exists s.B_1 \sqsubseteq C \in \mathcal{T}$ and $C \notin S_{i+1}(A)$ then $S_{i+1}(A) := S_{i+1}(A) \cup C$

Figure 2.2: $\mathcal{ELH}$ Implication set rules

Note that the successor $S_{i+1}(A)$ of some $S_i(A)$ is generally not the result of only a single rule application. $S_{i+1}(A)$ is complete only if no more rules are applicable to any $S_i(B)$ or $S_i(r)$. Implication sets induce a reflexive and transitive but not symmetric relation on $CN(\mathcal{T})$ and $RN(\mathcal{T})$, since $B \in S_n(A)$ does not imply $A \in S_n(B)$. The following theorems from Brandt (2004*a*) show that implication sets characterize subsumption in $\mathcal{ELH}$, as well as the fact that they can be computed in polynomial time.

Theorem 1 *(Brandt 2004*a*) For every normalized $\mathcal{ELH}$-TBox $\mathcal{T}$ over CN and RN*

1. *for every $r, s \in RN(\mathcal{T}), s \in S_{\mathcal{T}}(r)$ implies $\mathcal{T} \models r \sqsubseteq s$, and*

2. *for every* $A, B \in CN(\mathcal{T})^{\top}$ *it holds that* $B \in S_{\mathcal{T}}(A)$ *iff* $\mathcal{T} \models A \sqsubseteq B$

It is therefore clear that any algorithm for computing implication sets also computes all possible subsumption relationships between concept names and between role names. Since the algorithm computes all possible subsumption relationships it is in reality a *classification* algorithm. What remains to be shown is that this can be achieved in polynomial time.

Theorem 2 *(Brandt 2004*b*) Subsumption in* $\mathcal{EL}\mathcal{H}$ *with respect to GCIs can be decided in polynomial time.*

In this section we considered subsumption testing for the DL $\mathcal{EL}\mathcal{H}$ obtained by extending $\mathcal{EL}$ with role hierarchies. Since $\mathcal{EL}$ is a strict subset of $\mathcal{EL}\mathcal{H}$, the same algorithm will work equally well on it. Suntisrivaraporn (2005) provides an algorithm based on implication sets and the satisfiability algorithm for propositional Horn formulae (Dowling & Gallier 1984). The algorithm computes subsumption between $\mathcal{EL}$ concepts in time cubic in the size of the TBox $\mathcal{T}$, i.e. $|\mathcal{T}|^3$.

Though the algorithm introduced in this section is designed specifically for classifying an $\mathcal{EL}\mathcal{H}$ TBox, similar polynomial worst case algorithms, based on computing implication sets, exist for other tractable members of the $\mathcal{EL}$-family as summarized in Table 2.5.

2.3.5 Intractable Members of the $\mathcal{EL}$-family

Having introduced the $\mathcal{EL}$-family of DLs, as well as the basis for polynomial worst case time subsumption testing algorithms, it is of practical interest to look at those extensions to $\mathcal{EL}$ for which reasoning becomes intractable. For convenience Table 2.6 lists a more comprehensive set of both description language constructors and ontological constructors.

The first intractable extension we discuss is $\mathcal{EL}^{\neg}$ (Table 2.7). It is defined as $\mathcal{EL}$ extended with negation of concept descriptions, and $\mathcal{EL}^{(\neg)}$ as $\mathcal{EL}$ extended with atomic negation, i.e. only negation of concept names are allowed. If C is a complex concept description, we see that by introducing a new concept name A and the two GCIs $A \sqsubseteq C, C \sqsubseteq A$, the negation of C, namely $\neg C$ can be represented as atomic negation $\neg A$. The two DLs are therefore equivalent and thus share the same complexity results. $\mathcal{EL}^{\neg}$ is a notational variant of $\mathcal{ALC}$ and shares the complexity results for subsumption in $\mathcal{ALC}$ w.r.t TBoxes with GCIs (Baader et al. 2005, Lutz & Satler 2000).

Description Language Constructors

Name	**Syntax**	**Semantics**
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
value restriction	$\forall r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
disjunction	$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$
inverse roles	$\exists r^{-}.C$	$\{\ x \mid \exists y \in \Delta^{\mathcal{I}} : (y,x) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\ \}$
role negation	$\exists \neg r.C$	$\{\ x \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \notin r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\ \}$
role union	$\exists r \cup s.C$	$\{\ x \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in r^{\mathcal{I}} \cup s^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\ \}$
transitive closure	$\exists r^{*}.C$	$\{\ x \mid \exists y \in \Delta^{\mathcal{I}} : (x,y) \in (r^{\mathcal{I}})^{+} \wedge y \in C^{\mathcal{I}}\ \}$

Ontologicial Constructors

Name	**Syntax**	**Semantics**
concept inclusion	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
concept definition	$C \equiv D$	$C^{\mathcal{I}} = D^{\mathcal{I}}$
role hierarchy	$r \sqsubseteq s$	$r^{\mathcal{I}} \subseteq s^{\mathcal{I}}$
role composition	$r_1 \circ \ldots \circ r_k \sqsubseteq r$	$r_1^{\mathcal{I}} \circ \ldots \circ r_k^{\mathcal{I}} \sqsubseteq r^{\mathcal{I}}$
transitive roles	transitive(r)	$\forall x,y,z \in \Delta^{\mathcal{I}} : (x,y),(y,z) \in r^{\mathcal{I}} \rightarrow (x,z) \in r^{\mathcal{I}}$
reflexive roles	reflexive(r)	$\forall x \in \Delta^{\mathcal{I}} : (x,x) \subseteq r^{\mathcal{I}}$
range restrictions	range(r)$\sqsubseteq C$	$\{y \in \Delta^{\mathcal{I}} \mid \exists x : (x,y) \in r^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}$
domain restrictions	domain(r)$\sqsubseteq C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y : (x,y) \in r^{\mathcal{I}}\} \subseteq C^{\mathcal{I}}$
at-most restriction	$(\leq rn)$, $n \in \mathcal{N}$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \leq n\}$
at-least restriction	$(\geq rn)$, $n \in \mathcal{N}$	$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \mid (x,y) \in r^{\mathcal{I}}\} \geq n\}$

Table 2.6: Syntax and semantics of DL constructors

Theorem 3 *(Baader et al. 2005) In $\mathcal{EL}^{(\neg)}$ subsumption and satisfiability with respect to general TBoxes is EXPTime-complete.*

$\mathcal{ELU}$ is defined as $\mathcal{EL}$ with the disjunction of concept descriptions ($C \sqcup D$). Baader et al. (2005) show that satisfiability in $\mathcal{EL}^{(\neg)}$ can be reduced to subsumption in $\mathcal{ELU}$, by appropriate substitutions. Since by Theorem 3 satisfiability in $\mathcal{EL}^{(\neg)}$ is EXPTime-complete, the same result holds for $\mathcal{ELU}$.

Theorem 4 *(Baader et al. 2005) In $\mathcal{ELU}$ subsumption with respect to general TBoxes is EXPTime-complete.*

$\mathcal{ELN}$ is defined as $\mathcal{EL}$ extended with both at-most number restrictions ($\leq rn$) and at-least number restrictions($\geq rn$). Baader et al. (2005) show that subsumption in $\mathcal{ELU}$ w.r.t general TBoxes can be reduced to subsumption in $\mathcal{EL}$ extended with at-least restrictions. Therefore the EXPTime hardness from $\mathcal{ELU}$ carries over to $\mathcal{ELN}$. The same result holds for $\mathcal{EL}$ extended with at-most number restrictions.

Name	$\mathcal{EL}^{\neg}$	$\mathcal{ELU}$	$\mathcal{ELN}$	$\mathcal{EL}^{R\cup}$	$\mathcal{EL}^{*}$	$\mathcal{ELI}$	$\mathcal{EL}^{R\neg}$
top	×	×	×	×	×	×	×
bottom	×						
conjunction	×	×	×	×	×	×	×
existential restriction	×	×	×	×	×	×	×
concept inclusion	×	×	×	×	×	×	×
negation	×						
disjunction	×	×					
value restriction	×						
at-most restriction			×				
at-least restriction			×				
inverse roles						×	
role negation							×
role union				×			
transitive closure					×		

Table 2.7: Intractable subset of the $\mathcal{EL}$ family of DLs

Theorem 5 *(Baader et al. 2005) In $\mathcal{ELN}$ subsumption with respect to general TBoxes is EXPTime-complete.*

Subsumption in $\mathcal{ELU}$ can be reduced, with appropriate substitutions, to subsumption in $\mathcal{EL}^{R\cup}$, $\mathcal{EL}^{*}$ and $\mathcal{EL}^{R\neg}$ respectively (Baader et al. 2005). Therefore we have from Theorem 4 that subsumption with respect to general TBoxes in any of these is EXPTime-complete.

Theorem 6 *(Baader et al. 2005) In $\mathcal{EL}^{R\cup}$, $\mathcal{EL}^{*}$ and $\mathcal{EL}^{R\neg}$ subsumption with respect to general TBoxes is EXPTime-complete.*

$\mathcal{ELI}$ is defined as $\mathcal{EL}$ extended with inverse roles. Satisfiability of $\mathcal{ALE}$ with respect to primitive TBoxes can be reduced to subsumption in $\mathcal{ELI}$ w.r.t general TBoxes (Baader et al. 2005). The problem has been shown by Calvanese (1996) to be PSPACE-Complete.

Theorem 7 *(Calvanese 1996) In $\mathcal{ELI}$ subsumption with respect to general TBoxes is PSPACE-hard.*

2.3.6 Supplemental Reasoning Services for the $\mathcal{EL}$-family

The standard reasoning services from the previous section address very specific reasoning tasks. Though these form some of the most basic reasoning tasks required by DL systems, there is a second set of reasoning tasks gaining more and more relevance in terms of aiding in the design and maintenance of ontologies. As the size of ontologies increases from thousands

to hundreds of thousands of axioms, ontology engineers find it more and more difficult to cope with the design and maintenance aspect of these ontologies. Not only is it difficult to understand the expert knowledge contained in these hundreds of thousands of axioms, but understanding the relationships inherent in this knowledge is a task that requires teams of experts and close collaboration when changes are made to the ontology.

With the growth of the Internet and the emergence of the semantic web comes the emergence of highly distributed ontologies. Not only is there the need for reusing ontologies in order to compose larger ontologies in a distributed fashion, it is also of paramount importance that only small relevant subsets of ontologies be extracted from existing ontologies (Schlobach & Cornet 2003, Cuenca Grau et al. 2007, 2008, Suntisrivaraporn 2009).

To this end modularization as a supplementary reasoning service has gained importance. Intuitively, a module is a small self contained subset of an ontology preserving a specific statement or set of statements[4] of interest. We formally define a module as follows:

Definition 19 (Module for the arbitrary DL $\mathcal{L}$) *Let $\mathcal{L}$ be an arbitrary description language, $\mathcal{O}$ an $\mathcal{L}$ ontology, and σ a statement formulated in $\mathcal{L}$. Then, $\mathcal{O}' \subseteq \mathcal{O}$ is a* module for σ in $\mathcal{O}$*(a* σ-module in $\mathcal{O}$*) whenever: $\mathcal{O} \models \sigma$ if and only if $\mathcal{O}' \models \sigma$. We say that $\mathcal{O}'$ is a module for a signature $\boldsymbol{S}$ in $\mathcal{O}$ (an* **S**-module in $\mathcal{O}$*) if, for every $\mathcal{L}$ statement σ with Sig(σ) $\subseteq \boldsymbol{S}$, $\mathcal{O}'$ is a σ-module in $\mathcal{O}$.*

Definition 19 is sufficiently general so that any subset of an ontology preserving a statement of interest is considered a module, the entire ontology is therefore a module in itself. An important property of modules in terms of the modular reuse of ontologies is *safety* (Cuenca Grau et al. 2007, 2008). Intuitively, a module conforms to a safety condition whenever an ontology $\mathcal{T}$ reuses concepts from an ontology $\mathcal{T}'$ in such a way so that it does not change the meaning of any of the concepts in $\mathcal{T}'$. We may then say that $\mathcal{T}$ uses $\mathcal{T}'$ safely. This safety condition holds exactly whenever a module is closed under domain expansions:

Definition 20 (Domain expansion (Cuenca Grau et al. 2008)) *Let $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ and $\mathcal{J} = (\Delta^{\mathcal{J}}, .^{\mathcal{J}})$ be two interpretations such that:*

1. *$\Delta^{\mathcal{J}} = \Delta^{\mathcal{I}} \cup \Phi$ with Φ a non empty set disjoint with $\Delta^{\mathcal{I}}$,*

[4]Here the statement of interest usually refers to an axiom composed using the same rules as axioms contained within the terminology. In order to differentiate between axioms within a terminology and statements over which we wish to perform reasoning we utilise the convention from Suntisrivaraporn (2009) and refer to the latter as statements of interest.

2. $A^{\mathcal{J}} = A^{\mathcal{I}}$ *for each concept name* A,

3. $R^{\mathcal{J}} = R^{\mathcal{I}}$ *for each role name* R.

We say that $\mathcal{J}$ *is the expansion of* $\mathcal{I}$ *with* Φ.

Intuitively, the interpretation $\mathcal{J}$ is identical to $\mathcal{I}$ except for the fact that it contains some additional elements in the interpretation domain. These elements do not participate in the interpretation of concept or roles.

A different but semantically equivalent definition of safety depends on the notion of conservative extensions (Cuenca Grau et al. 2007, 2008):

Definition 21 (Conservative extension (Cuenca Grau et al. 2008)) *Let* $\mathcal{T}$ *and* $\mathcal{T}_1$ *be two ontologies such that* $\mathcal{T}_1 \subseteq \mathcal{T}$*, and let* $\mathcal{S}$ *be a signature. Then*

1. $\mathcal{T}$ *is an* $\mathcal{S}$*-Conservative Extension of* $\mathcal{T}_1$ *if, for every* α *with Sig(*α*)* $\subseteq \mathcal{S}$*, we have* $\mathcal{T} \models \alpha$ *iff* $\mathcal{T}_1 \models \alpha$

2. $\mathcal{T}$ *is a Conservative Extension of* $\mathcal{T}_1$ *if* $\mathcal{T}$ *is an* $\mathcal{S}$*-conservative extension of* $\mathcal{T}_1$ *for* $\mathcal{S}$ = *Sig(*$\mathcal{T}_1$*)*

The safety for a module may now be formally defined as follows:

Definition 22 (Safety for a module (Cuenca Grau et al. 2008)) *An ontology* $\mathcal{T}$ *is* safe *for* $\mathcal{T}'$ *if* $\mathcal{T} \cup \mathcal{T}'$ *is a conservative extension of* $\mathcal{T}'$.

As is the case for standard reasoning services, when designing supplemental reasoning services we wish to normalise an ontology so that complex concept descriptions are reduced to a few base cases. In terms of modularization there are two normal forms of interest. The first is the normal form for $\mathcal{ELH}$ from Definition 15 extended to include all the constructors for the DL $\mathcal{EL}^+$ (Table 2.5).

We require that an $\mathcal{EL}^+$ ontology $\mathcal{O}$ be in normal form. We use the same form as Brandt (2004*a*) and Suntisrivaraporn (2009). Any $\mathcal{EL}^+$ ontology $\mathcal{O}$ can be converted to an ontology $\mathcal{O}'$ in normal form in linear time, with at most a linear increase in the size of the ontology (Suntisrivaraporn 2009).

Definition 23 (Normal Form for $\mathcal{EL}^+$) *An* $\mathcal{EL}^+$ *ontology* $\mathcal{O}$ *is in* normal form *if the following conditions are satisfied:*

1. *all concept inclusions in $\mathcal{O}$ have one of the following forms:*
$$\begin{aligned} A_1 \sqcap \ldots \sqcap A_n &\sqsubseteq B, \\ A_1 &\sqsubseteq \exists r.A_2, \\ \exists r.A_1 &\sqsubseteq B \end{aligned}$$
where $A_i \in CN^{\top}(\mathcal{O})$ and $B \in CN^{\bot}(\mathcal{O})$;

2. *all role inclusions in $\mathcal{O}$ have one of the following forms:*
$$\begin{aligned} \epsilon &\sqsubseteq r, \\ r &\sqsubseteq s, \\ r \circ s &\sqsubseteq t, \end{aligned}$$
where $r, s, t \in RN(\mathcal{O})$ and ϵ is the identity element;

3. *there are no reflexivity statements, transitivity statements or domain restrictions, and all range restrictions are of the form $range(r) \sqsubseteq A$ with A a concept name.*

The second, but less restrictive, normal form will be used in this dissertation exclusively during the extraction of reachability-based modules in Section 3.2 and may be defined as:

Definition 24 (Module normal form (Suntisrivaraporn 2009)) *An $\mathcal{EL}^+$ ontology $\mathcal{O}$ is in module normal form if all of its axioms are either:*

- *concept inclusions $C \sqsubseteq D$, where $\bot$ does not occur in C, or*
- *role inclusions $r_1 \circ \cdots \circ r_n \sqsubseteq s$, or*
- *range restrictions $range(r) \sqsubseteq D$.*

We note that every $\mathcal{EL}^+$ module in normal form is by default also in module normal form, the converse however does not necessarily hold. In this paper when we refer to the term 'normal form' we refer to the normal form of Definition 23, where we need to refer to the less restrictive form of Definition 24 we will explicitly use the term 'module normal form'.

The second supplementary reasoning service we investigate is designed to aid in the debugging of ontologies. More specifically, as inconsistencies and modelling errors are introduced during the design or augmentation of ontologies, standard reasoning services may reveal unwanted consequences or inconsistencies. Pinpointing, by hand, the exact reasons for these modelling errors amongst hundreds of thousands of axioms is at best an impossible task. In order to assist in finding such errors the notion of Axiom Pinpointing (Schlobach & Cornet

2003) has been introduced as a supplementary reasoning service. It is worthwhile to note that the technique for pinpointing was first introduced by Baader & Hollunder (1995).

In essence Axiom Pinpointing refers to the process of extracting Minimal Axiom Sets or MinAs for a specified statement of interest. A MinA is formally defined as:

Definition 25 (Minimal axiom set (Suntisrivaraporn 2009)) *Let $\mathcal{O}$ be an ontology, and σ a subsumption statement such that $\mathcal{O} \models \sigma$. A subset $S \subseteq \mathcal{O}$ is a minimal axiom set (MinA) for σ with respect to $\mathcal{O}$, also written as 'S is a MinA for $\mathcal{O} \models \sigma$', if and only if*

1. *$S \models \sigma$, and*
2. *for every $S' \subset S$, $S' \not\models \sigma$.*

From the definition it is clear that a MinA is a minimal module which preserves the given entailment, the set of all MinAs is then the set of all minimal modules. Though it is the case that every MinA is a minimal module for an entailment, many different MinAs may exists for the same entailment. In the literature MinAs are synonymously referred as to MUPS (Schlobach & Cornet 2003) or as justifications (Kalyanpur et al. 2007), where a single MinA is a justification for the entailment to hold in the ontology.

Chapter 3

Module Extraction

In this chapter we investigate module extraction methods and extend the notion of reachability-based modules (Suntisrivaraporn 2009) by introducing a top-down and bidirectional version of a reachability heuristic. Preliminary versions of these new heuristics were first introduced at AOW'09 (Nortjé et al. 2009).

One of the original motivations behind ontology research was the belief that ontologies can help with reuse in knowledge representation (Gruber 1993). Designing ontologies specifically aimed at reuse is considered by Barnaras et al. (1996) as an engineering good practice principle for building ontologies. However, reusing ontologies poses certain difficulties (Noy & Musen 2004, Stuckenschmidt & Klein 2004):

- General ontologies are very large, containing from a few thousand to a few hundred thousand concepts, from this the developer only requires a small self-contained portion of the ontology.
- Importing a whole ontology includes a lot of unnecessary concepts, which will never be of use to the developer.
- From the view of the developer, there is the cognitive challenge of understanding a large ontology, with teams of experts often working in conjunction in order to maintain it.
- Large ontologies introduce a computational penalty when the user needs to perform reasoning over the whole ontology.

Modularization aims to resolve these difficulties by extracting small subsets of ontologies, referred to as modules. These modules may then:

- be reused by importing them into other ontologies,
- serve as self contained sub-ontologies,

- assist the developer in understanding specific smaller subsets of an ontology,
- simplify query answering,
- improve the efficiency of reasoning performance, and
- aid in the debugging of ontologies.

Though modules are of great practical use, their utility comes at a price. The various functional roles modules are to perform necessitate different formal specifications of their exact nature. Each of these has its own strengths and weaknesses.

In Section 3.1 we look at some of the more common approaches to module extraction and their uses. Then in Section 3.2 we focus on reachability-based modules as introduced by Suntisrivaraporn (2009). Section 3.3 introduces a new module extraction heuristic based on top-down reachability. Lastly Section 3.4 combines the work from Sections 3.2 and 3.3 to introduce a bidirectional reachability-based heuristic specifically aimed at optimizing subsumption testing and debugging. Hence Sections 3.3 and 3.4 constitute original contributions made by this dissertation.

3.1 Modularization Methods

In this section we investigate different approaches to module extraction, the criteria these modules must adhere to and their uses. This section by no means aims to be comprehensive in its treatment of all module extraction methods, but rather aims to introduce some of the key concepts behind different modularization methods.

Partitioning Methods: Small self contained modules aid in the understanding, verification, debugging, reuse and collaborative development of larger ontologies. For large web ontologies, where collaboration is uncoordinated, there is a requirement that small self-contained modules may be extracted that are usable outside the context of the original ontology, whilst yet preserving its meaning in the context of the original ontology.

Cuenca Grau et al. (2005) define a module as a self contained unit where no subsumption relationships exist between concepts defined inside the module and concepts defined outside the module, enforcing a logical separation between a module and its context. The authors show that only those ontologies meeting a *safety* requirement may be modularized in this way, where an ontology is considered safe if all its models are closed under domain expansions.

Unsafe axioms are, for example, those axioms that establish concepts as equivalent to the universal concept ($\top \sqsubseteq Fish$).

Once the classical consistency and safety of an ontology have been verified, a partitioning graph is created. The partitioning graph contains partitions that describe intuitively disjoint subject matter corresponding to well defined application domains. From these partitions, modules meeting the safety criteria above may then be extracted. A criticism raised against this method is that not all ontologies meet the *safety* requirement, thus limiting the availability of reusable ontologies (Doran et al. 2007).

Stuckenschmidt & Klein (2004) argue that, for a module to be self contained, concept names inside a module should not have strong dependencies to concept names outside of the module. They calculate dependencies between concept names in terms of the structure of the ontology, which includes the concept hierarchy, constraints on properties, domains and ranges. This is achieved by creating a dependency graph between concepts and then forming a proportional strength network by assigning weights to these dependencies using formulae from social networking theory. Using this framework, concepts with a low number of dependencies with other concepts are more strongly dependant on each other than highly connected concepts. The proportional strength network is then used to iteratively break up the ontology into dynamically sized modules.

Syntactic Traversal Methods: A close analogy can be drawn between query answering in ontologies and query answering in databases, as obtaining self contained portions of a resource has long been an area of research in databases. A database view provides that a user may specify, by means of a query, the exact portion of a database she is interested in. Creating a specific view of the data in the database, where all instances of data satisfying a specific query, constitutes a view. An ontology view is analogously defined as a portion of an ontology that results from a query specified in some ontology-query language.

A criticism against ontology views is that they do not allow users to specify a portion of an ontology that results from a particular traversal of ontology links (Noy & Musen 2004). As an example, a user may specify, in a medical ontology, that she requires everything that is directly related to the concept 'heart' as well as definitions of all organs and organ parts that surround the heart. The user may also ask to include all organs or organ parts that are 'twice removed' from the heart. Traditional ontology views do not allow the user to restrict the results in this way.

Noy & Musen (2004) suggest a complementary way of defining ontology views: i.e. a *traversal specification*. In such a specification, a starter concept or concepts is defined, these are the 'central' or 'focus' concepts in the results. Next, the details of the relationship to traverse, starting at these concepts, are specified. The result of such a query is called a traversal view.

Intuitively, a traversal view is specified by specifying a starter concept (e.g. Heart), a list of relationships (property names) that should be traversed, and the maximum distance to traverse along each of the relationships. Since ontology views are not extracted, nor treated as separate entities, the user may shrink or expand the view as required, thereby targeting specific parts of the ontology as needed.

Modular reuse of ontologies is not limited to extracting modules from existing ontologies but also composing larger ontologies from smaller self-contained ones, each containing a set of internal and external symbols. External symbols are those symbols for which axioms will be imported from another ontology. These have the property that they are used in the definition of internal concept names but their interpretations are not constrained by the internal concept names in any way.

For example, when a TBox $\mathcal{T}$ reuses concepts from another TBox $\mathcal{T}'$, the integration process must be carried out in such a way that the consequences of $\mathcal{T}'$ are not changed. This property may be formally expressed by the notion of *safety* for a module as defined in Definition 22. However, since computing *safety* is undecidable for complex DLs, EXPTime complete for $\mathcal{EL}$ and 2EXPTime complete for $\mathcal{ALC}$, Cuenca Grau et al. (2008) introduce *locality* as sufficient conditions for safety. Locality of a module can be expressed in terms of either *semantic locality*, which is computationally hard, or its approximation *syntactic locality*.

Based on these locality conditions, an ontology will be called a *local* ontology if, for a given signature $\mathcal{S}$, all its axioms are satisfied by a specified class of interpretations. Two classes of interpretations are of interest, namely bottom($\bot$)-locality, used for refining the symbols from a signature, and top($\top$)-locality, used for generalising the symbols from the signature (Cuenca Grau et al. 2008). Locality provides sufficient conditions for safely extracting and integrating modules, as well as a framework for developers to use during the collaborative design stage of modules. These methods have been implemented in the ProSE (Jiménez-Ruiz et al. 2008) plugin for the Protégé ontology development environment.

3.2 Reachability-based Module Extraction

In this section we look at reachability-based modules as introduced by Suntisrivaraporn (2009). Though modules in this section may be usable in the ontology reuse scenario, the main focus of modules as introduced by Suntisrivaraporn (2009) is for the optimization of reasoning tasks. Therefore, given an ontology $\mathcal{O}$ and an entailment $\mathcal{O} \models \sigma$, with σ a statement of interest, extracting a module in this section aims to obtain a small subset $\mathcal{O}'$ of $\mathcal{O}$, such that entailment of σ is preserved. For the purposes of this section σ is always a subsumption statement, the DL is restricted to $\mathcal{EL}^+$ and every ontology is in at least 'module normal form' as defined in Definition 24.

Extracting modules aims to preserve both subsumption and non-subsumption relationships in a subset of an ontology. This can be understood as the reachability problem in a directed graph, considering concept names as nodes and explicit subsumption relationships as edges in the graph.

Definition 26 (Tight reachability (Suntisrivaraporn 2009)) *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology, and A, B concept names in $\mathcal{O}$. The tight reachability graph $G_t(\mathcal{O})$ for $\mathcal{O}$ is a tuple (V_t,E_t) with $V_t = CN(\mathcal{O})$ and E_t the smallest set containing all edges $\langle A, B\rangle$ if $A \sqsubseteq D \in \mathcal{O}$ s.t. B is a conjunct in D. We say that B is tightly reachable from A in $\mathcal{O}$ if there is a path from A to B in $G_t(\mathcal{O})$.*

Definition 27 (Loose reachability (Suntisrivaraporn 2009)) *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology, and A, B concept names in $\mathcal{O}$. The loose reachability graph $G_l(\mathcal{O})$ for $\mathcal{O}$ is a tuple (V_l,E_l) with $V_l = CN(\mathcal{O})$ and E_l the smallest set containing all edges $\langle A, B\rangle$ if $C \sqsubseteq D \in \mathcal{O}$ s.t. A occurs in C and B occurs in D. We say that B is loosely reachable from A in $\mathcal{O}$ if there is a path from A to B in $G_l(\mathcal{O})$.*

According to the definition of loose reachability, given the axiom $C_1 \sqcap \ldots \sqcap C_n \sqsubseteq B$ and $A \sqsubseteq C_i$ for some i s.t. $1 \leq i \leq n$, there exists a possible subsumption relationship $A \sqsubseteq B$. However, in terms of the semantics of conjunction, we know that for $A \sqsubseteq B$ to hold, we require that $A \sqsubseteq C_i$ for all $1 \leq i \leq n$, hence this definition of reachability is too weak. This definition is therefore extended so that an ontology is viewed in terms of a directed hypergraph (Ausiello et al. 2001) where each inclusion axiom $\alpha_L \sqsubseteq \alpha_R \in \mathcal{O}$ essentially specifies a collection of hyperedges from the connected node $\mathrm{Sig}(\alpha_L)$ to each of the symbols in $\mathrm{Sig}(\alpha_R)$. Formally:

Definition 28 (Bottom-up reachability-based modules (Suntisrivaraporn 2009)) [1] *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology and $S \subseteq Sig(\mathcal{O})$ a signature. The set of S-reachable names in $\mathcal{O}$ is defined inductively as:*

- *x is S-reachable in $\mathcal{O}$, for every $x \in S$;*
- *for all inclusion axioms $\alpha_L \sqsubseteq \alpha_R$, if x is S-reachable in $\mathcal{O}$ for every $x \in Sig(\alpha_L)$, then y is S-reachable in $\mathcal{O}$ for every $y \in Sig(\alpha_R)$.*

We call an axiom $\alpha_L \sqsubseteq \alpha_R$ S-reachable in $\mathcal{O}$ if every element of $Sig(\alpha_L)$ is S-reachable in $\mathcal{O}$. The reachability-based module for S in $\mathcal{O}$, denoted by $\mathcal{O}_S^{reach}$, consists of all S-reachable axioms in $\mathcal{O}$.

When S is the single concept A, we write A-reachable and $\mathcal{O}_A^{reach}$. An interesting result of reachability is that it can be used to test negative subsumption. That is, if B is not A-reachable in $\mathcal{O}$, then $\mathcal{O} \not\models A \sqsubseteq B$, unless A is unsatisfiable w.r.t $\mathcal{O}$ (Suntisrivaraporn 2009). We observe that, for the DL $\mathcal{EL}^+$, there are axioms $\alpha_L \sqsubseteq \alpha_R$ for which $\text{Sig}(\alpha_L) = \emptyset$. These axioms have the form $\top \sqsubseteq \alpha_R$, $\epsilon \sqsubseteq r$ are trivially reachable from any concept, thus they will form part of every reachability-based module extracted.

We note that the set $\mathcal{O}_S^{reach}$ is unique in terms of the ontology $\mathcal{O}$ and the signature S. This is apparent from the inductive definition of $\mathcal{O}_S^{reach}$ and that every S-reachable axiom in $\mathcal{O}$ is always part of $\mathcal{O}_S^{reach}$ by definition.

An important result of reachability-based modules are that they not only preserve all subsumption relationships in terms of the subsumee of an entailment, but that they correspond to strong subsumption modules and thus contain all possible MinAs, as defined in Definition 25 for any entailment with the given subsumee. Strong subsumption modules are defined as follows:

Definition 29 (Subsumption module (Suntisrivaraporn 2009)) *Let $\mathcal{O}$ be an ontology, and A a concept name occurring in $\mathcal{O}$. Then, $\mathcal{O}' \subseteq \mathcal{O}$ is a subsumption module for A in $\mathcal{O}$ whenever: $\mathcal{O} \models A \sqsubseteq B$ if and only if $\mathcal{O}' \models A \sqsubseteq B$ holds for every concept name B occurring in $\mathcal{O}$.*

A subsumption module $\mathcal{O}'$ for A in $\mathcal{O}$ is called strong *if the following holds for every concept name B occurring in $\mathcal{O}$: if $\mathcal{O} \models A \sqsubseteq B$, then every MinA for $\mathcal{O} \models A \sqsubseteq B$ is a subset of $\mathcal{O}'$.*

[1] *The original definition by Suntisrivaraporn does not have the qualifier 'bottom-up', but because we introduce 'top-down' reachability-based modules later on in Definition 31, the qualifier is used to avoid confusion.*

Theorem 8 *(Suntisrivaraporn 2009). The module $\mathcal{O}_A^{reach}$ is a strong subsumption module for A in $\mathcal{O}$.*

Suntisrivaraporn (2009) shows that reachability-based modules are equivalent to minimal locality-based modules (Cuenca Grau et al. 2008), with the interesting consequence that any algorithm for extracting reachability-based modules is also an algorithm for extracting minimal locality-based modules. Syntactic locality based modules for $\mathcal{EL}^+$ are defined as:

Definition 30 (Locality-based modules (Cuenca Grau et al. 2007)) *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology, and S a signature. The following grammar recursively defines the set $\mathbf{Con}^\bot(\mathbf{S})$ for a signature S:*

$$\mathbf{Con}^\bot(\mathbf{S}) := A^\bot | (C^\bot \sqcap C) | (C \sqcap C^\bot) | (\exists r.C^\bot) | (\exists r^\bot.C)$$

with r a role name, C a concept description, $A^\bot \notin S$ a concept name, $r^\bot \notin S$ a role name, and $C^\bot \in \mathbf{Con}^\bot(\mathbf{S})$. An $\mathcal{EL}^+$ axiom α is syntactically local with respect to S if it has one of the following forms:

1. *$R^\bot \sqsubseteq s$ where $R^\bot$ is either a role name $r^\bot \notin S$ or a role composition $r_1 \circ \cdots \circ r_n$ with $r_i \notin S$ for some $i \leq n$, or*
2. *$C^\bot \sqsubseteq C$ where $C^\bot \in \mathbf{Con}^\bot(\mathbf{S})$.*

We write $\mathbf{local}(\mathbf{S})$ to denote the collection of all $\mathcal{EL}^+$ axioms that are syntactically local with respect to S. If $\mathcal{O}$ can be partitioned into $\mathcal{O}_1$ and $\mathcal{O}_2$ such that every axiom in $\mathcal{O}_2$ is syntactically local with respect to $S \cup Sig(\mathcal{O}_1)$, then $\mathcal{O}_1$ is a locality-based module for S in $\mathcal{O}$. We say that $\mathcal{O}_1$ is minimal if there is no $\mathcal{O}' \subset \mathcal{O}_1$ that is a locality-based module for S in $\mathcal{O}$.

Given an ontology $\mathcal{O}$ and a signature $S \subseteq \text{Sig}(\mathcal{O})$, Algorithm 1 correctly extracts a reachability-based module $\mathcal{O}_S^{reach}$ in $O(n^2)$ time where n is the number of axioms in the ontology. Using elements from the algorithms in Dowling & Gallier (1984) it is possible to extract reachability-based modules in linear time, i.e. $O(n)$, where n is the size of the ontology. It must be noted that, though the algorithm given runs in quadratic time, the size of the input is the number of axioms in the ontology, whereas for a linear algorithm the size of the input is the number of concepts in the ontology. Here active-axioms(x) comprises all and only those axioms $\alpha_L \sqsubseteq \alpha_R \in \mathcal{O}$ such that $x \in \text{Sig}(\alpha_L)$.

Theorem 9 (Suntisrivaraporn 2009) *Algorithm 1 produces $\mathcal{O}_S^{reach}$.*

Algorithm 1 (Extract bottom-up reachability-based module (Suntisrivaraporn 2009))

Procedure extract-module($\mathcal{O}$, S)
Input 1: $\mathcal{O}$ - $\mathcal{EL}^+$ ontology in at least module normal form*;*
Input 2: S - signature
Output: $\mathcal{O}_S$ - reachability-based module for S in $\mathcal{O}$

```
1:  O_S := ∅
2:  queue := active-axioms(S)
3:  while not empty(queue) do
4:    (α_L ⊑ α_R) := fetch(queue)
5:    if Sig(α_L) ⊆ S∪Sig(O_S)
6:      O_S := O_S ∪ {α_L ⊑ α_R}
7:      queue := queue ∪(active-axioms(Sig(α_R))\O_S)
8:    end if
9:  end while
10: return O_S
```

3.3 Top-Down Reachability-based Modules for $\mathcal{EL}^+$

The reachability heuristic from Section 3.2 can be viewed as a bottom-up searching procedure, that is, starting with a concept A we expand the search to find all super-concepts B such that $A \sqsubseteq B$. One of the criticisms that may be raised against these bottom-up reachability-based modules, in terms of finding justifications, is that they contain many irrelevant axioms, and in some cases do not reduce the size of the ontology at all (Du et al. 2009). This stems from the fact that $\mathcal{O}_A^{reach}$ considers only the sub-concept A in $\mathcal{O} \models A \sqsubseteq B$; the super-concept B is never used to eliminate unwanted axioms. The following example illustrates this:

Example 3 *Given the small ontology $\mathcal{O}$ below, as well as $\mathcal{O} \models A \sqsubseteq B$, $\mathcal{O}_A^{reach}$ will consist of axioms 3.1, 3.2 and 3.4. Axiom 3.4 is irrelevant in terms of finding justifications for $\mathcal{O} \models A \sqsubseteq B$, yet it is included in $\mathcal{O}_A^{reach}$.*

$$A \sqsubseteq \exists r.D \tag{3.1}$$

$$\exists r.D \sqsubseteq B \tag{3.2}$$

$$E \sqsubseteq B \tag{3.3}$$

$$A \sqsubseteq F \tag{3.4}$$

It is clear from this example that, for large ontologies, we may include many such irrelevant axioms in a reachability-based module. As a first step towards extracting modules that consider both the sub-concept as well as the super-concept of an entailment, we show

in this section that reachability may also be applied in a top-down manner, that is, starting with the super-concept B we expand the search to find all of its subsumees A such that $A \sqsubseteq B$.

By definition, a module must contain all axioms that are required to preserve the entailment of a specific statement of interest. In terms of top-down reachability our aim is to obtain a module in terms of the super-concept of a subsumption statement. Therefore, the inclusion of axioms in a top-down reachability-based module must be dependent on the symbols of the super-concepts appearing on the right hand side of axioms. Given an axiom $\alpha_L \sqsubseteq \alpha_R$ in normal form, as defined in Definition 23, α_R may have one of the following forms:

1. $\alpha_R \in CN(\mathcal{O})$, or
2. $\alpha_R \in RN(\mathcal{O})$, or
3. $\alpha_R = \bot$, or
4. $\alpha_R = \exists r.C$ with $C \in CN(\mathcal{O})$.

In the discussion that follows we assume that the statement of interest is $A \sqsubseteq B$. Considering forms 1 and 2 we see that $\text{Sig}(\alpha_R)$ contains only a single symbol each. It is clear that axioms in one of these forms must be included in a top-down reachability-based module whenever these symbols are top-down reachable from the super-concept symbol B.

For form 3 however $\text{Sig}(\alpha_R)=\emptyset$, and in terms of the semantics of $\bot$, we know that $\bot \sqsubseteq x_i$ for all $x_i \in CN(\mathcal{O})$. Therefore, in order to preserve all subsumption relationships, these axioms have to form part of every top-down reachability-based module.

The only remaining axioms to consider are axioms of form 4. At a first glance it may seem that the requirement for the inclusion of the axiom $\alpha_L \sqsubseteq \exists r.C$ containing an existential restriction on the right hand side in a top-down reachability-based module is that both the symbols r and C be reachable from B. The following example shows that this assumption is incorrect:

Example 4 *Given the small ontology $\mathcal{O}$ below, and $A \sqsubseteq B$ as the statement of interest, if we require that all symbols of an existential restriction be top-down reachable from B, only axiom 3.6 will be chosen. Since Sig($\top$) = $\emptyset$, C in axiom 3.5 will never be reachable from B. This is undesirable since $\mathcal{O} \models A \sqsubseteq B$ and the module we extract must preserve all entailments in*

terms of the super-concept.

$$A \sqsubseteq \exists r.C \tag{3.5}$$

$$\exists r.\top \sqsubseteq B \tag{3.6}$$

From this example it is clear that requiring both symbols of an existential restriction to be reachable from the top will result in modules that do not preserve all entailments. We therefore require that axioms of this type always be included in a module whenever any one of the symbols becomes reachable from the top. Applying this method to the previous example we see that this approach results in a module being extracted that preserves the given entailment. We therefore define top-down reachability as follows:

Definition 31 (Top-down reachability-based module) *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology and $S \subseteq Sig(\mathcal{O})$ a signature. The set of $\overleftarrow{\text{S}}$-reachable names in $\mathcal{O}$ is defined inductively as:*

- *x is $\overleftarrow{\text{S}}$-reachable in $\mathcal{O}$, for every $x \in S$;*
- *for all inclusion axioms $\alpha_L \sqsubseteq \alpha_R$, if x is $\overleftarrow{\text{S}}$-reachable in $\mathcal{O}$ for some $x \in Sig(\alpha_R)$, or if $\alpha_R = \bot$, then y is $\overleftarrow{\text{S}}$-reachable in $\mathcal{O}$ for every $y \in Sig(\alpha_L)$.*

We call an axiom $\alpha_L \sqsubseteq \alpha_R$ $\overleftarrow{\text{S}}$-reachable in $\mathcal{O}$ if some element of $Sig(\alpha_R)$ is $\overleftarrow{\text{S}}$-reachable or if $\alpha_R = \bot$. The top-down reachability-based module for S in $\mathcal{O}$, denoted by $\mathcal{O}^{reach}_{\overleftarrow{\text{S}}}$, consists of all $\overleftarrow{\text{S}}$-reachable axioms from $\mathcal{O}$.

Because of its inductive nature the set $\mathcal{O}^{reach}_{\overleftarrow{\text{S}}}$ is by definition unique in terms of the ontology $\mathcal{O}$ and the signature S. We note that the reachability graph for top-down reachability differs from that in Definition 27 in that nodes in the top-down graph includes both concept and roles names. Whereas in the reachability graph from Definition 27 nodes consists of only concept names.

From this definition it is clear that besides the direction of application, there are some differences between bottom-up and top-down reachability-based modules. That is,

1. When extracting $\mathcal{O}^{reach}_{A}$, the axiom $\alpha_L \sqsubseteq \alpha_R$ becomes A-reachable only when all $x_i \in \text{Sig}(\alpha_L)$ are A-reachable, whereas
2. when extracting $\mathcal{O}^{reach}_{\overleftarrow{\text{B}}}$, the axiom $\alpha_L \sqsubseteq \alpha_R$ is $\overleftarrow{\text{B}}$-reachable whenever any $x_i \in \text{Sig}(\alpha_R)$ is $\overleftarrow{\text{B}}$-reachable.

We now proceed to show that top-down reachability modules preserves all subsumption relationships in terms of super-concepts. In order to do so we define top-down subsumption modules analogous to reachability-based subsumption modules. We then proceed to prove that they indeed preserve all subsumptions and furthermore, that they also contain all MinAs for an entailment $\mathcal{O} \models A \sqsubseteq B$.

Definition 32 (Top-down subsumption module) *Let $\mathcal{O}$ be an ontology, and B a concept name occurring in $\mathcal{O}$. Then, $\mathcal{O}' \subseteq \mathcal{O}$ is a top-down subsumption module for B in $\mathcal{O}$ whenever: $\mathcal{O} \models A \sqsubseteq B$ if and only if $\mathcal{O}' \models A \sqsubseteq B$ holds for every concept name A occurring in $\mathcal{O}$.*

A top-down subsumption module $\mathcal{O}'$ for B in $\mathcal{O}$ is called strong *if the following holds for every concept name A occurring in $\mathcal{O}$: if $\mathcal{O} \models A \sqsubseteq B$, then every MinA for $\mathcal{O} \models A \sqsubseteq B$ is a subset of $\mathcal{O}'$.*

Lemma 2 *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology, as defined in Table 2.5, and $S \subseteq Sig(\mathcal{O})$ a signature. Then, $\mathcal{O} \models C \sqsubseteq D$ if and only if $\mathcal{O}^{reach}_{\overleftarrow{S}} \models C \sqsubseteq D$ for arbitrary $\mathcal{EL}^+$ concept descriptions C and D such that $Sig(D) \subseteq S$.*

Proof: We have to prove two parts. First: If $\mathcal{O}^{reach}_{\overleftarrow{S}} \models C \sqsubseteq D$ then $\mathcal{O} \models C \sqsubseteq D$. This follows directly from the fact that $\mathcal{O}^{reach}_{\overleftarrow{S}} \subseteq \mathcal{O}$ and that $\mathcal{EL}^+$ is monotonic.

Conversely, we show that, if $\mathcal{O} \models C \sqsubseteq D$ then $\mathcal{O}^{reach}_{\overleftarrow{S}} \models C \sqsubseteq D$: Assume the contrary, that is, assume $\mathcal{O} \models C \sqsubseteq D$ but that $\mathcal{O}^{reach}_{\overleftarrow{S}} \not\models C \sqsubseteq D$. Then there must exist an interpretation $\mathcal{I}$ and an individual $w \in \Delta^{\mathcal{I}}$ such that $\mathcal{I}$ is a model of $\mathcal{O}^{reach}_{\overleftarrow{S}}$ and $w \in C^{\mathcal{I}} \setminus D^{\mathcal{I}}$. Modify $\mathcal{I}$ to $\mathcal{I}'$ by setting $x^{\mathcal{I}'} := \Delta^{\mathcal{I}}$ for all concept names $x \in \text{Sig}(\mathcal{O}) \setminus (S \cup \text{Sig}(\mathcal{O}^{reach}_{\overleftarrow{S}}))$, and $r^{\mathcal{I}'} := \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for all roles names $r \in \text{Sig}(\mathcal{O}) \setminus (S \cup \text{Sig}(\mathcal{O}^{reach}_{\overleftarrow{S}}))$ and leaving everything else unchanged. $\mathcal{I}'$ is a model of $\mathcal{O}^{reach}_{\overleftarrow{S}}$ since it does not change the interpretation of any symbol in its signature. Every $\alpha = (\alpha_L \sqsubseteq \alpha_R) \in \mathcal{O} \setminus \mathcal{O}^{reach}_{\overleftarrow{S}}$ is not $\overleftarrow{S}$-reachable and thus when:

1. α_R is a concept name, we have that $\alpha_R^{\mathcal{I}'} = \Delta^{\mathcal{I}}$, or
2. α_R is a role name, we have that $\alpha_R^{\mathcal{I}'} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, or
3. α_R is an existential restriction $\exists r.A_2$, we have that $r^{\mathcal{I}'} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ and $A_2^{\mathcal{I}'} = \Delta^{\mathcal{I}}$ so that $(\exists r.A_2)^{\mathcal{I}'} = \Delta^{\mathcal{I}}$.

Since for all cases $\alpha_L^{\mathcal{I}'} \subseteq \alpha_R^{\mathcal{I}'}$ we conclude that $\mathcal{I}'$ is a model for $\mathcal{O}$. But $\mathcal{I}$ and $\mathcal{I}'$ correspond on all symbols $y \in \text{Sig}(D) \subseteq S$ and therefore $D^{\mathcal{I}'} = D^{\mathcal{I}}$. Now since $C^{\mathcal{I}} \subseteq C^{\mathcal{I}'}$ and $w \in C^{\mathcal{I}}$

we have that $w \in C^{\mathcal{I}'} \setminus D^{\mathcal{I}'}$ and hence $\mathcal{O} \not\models C \sqsubseteq D$, contradicting the assumption. □

In order to show that $\mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$ contains all MinAs for the entailment $\mathcal{O} \models A \sqsubseteq B$, we show that $\mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$ is a strong top-down subsumption module:

Theorem 10 *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology and B a concept name occurring in $\mathcal{O}$. Then $\mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$ is a strong top-down subsumption module for B in $\mathcal{O}$.*

Proof: That $\mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$ is a top-down subsumption module follows directly from Lemma 2 above. To show that it is strong, assume that $\mathcal{O} \models A \sqsubseteq B$ for some concept name A, but there is a MinA M for $\mathcal{O} \models A \sqsubseteq B$ that is not contained in $\mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$. Thus, there must be an axiom $\alpha \in M \setminus \mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$. Define $M_1 := M \cap \mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$. M_1 is a strict subset of M since $\alpha \notin M_1$. We claim that $M_1 \models A \sqsubseteq B$, which contradicts the fact that M is a MinA for $\mathcal{O} \models A \sqsubseteq B$.

We use proof by contraposition to show this. Assume that $M_1 \not\models A \sqsubseteq B$ i.e., there is a model $\mathcal{I}_1$ of M_1 such that $A^{\mathcal{I}_1} \not\subseteq B^{\mathcal{I}_1}$. We modify $\mathcal{I}_1$ to $\mathcal{I}$ by setting $y^{\mathcal{I}} := \Delta^{\mathcal{I}_1}$ for all concept names y that are not $\overleftarrow{\mathrm{B}}$-reachable, and $r^{\mathcal{I}} := \Delta^{\mathcal{I}_1} \times \Delta^{\mathcal{I}_1}$ for all roles names r that are not $\overleftarrow{\mathrm{B}}$-reachable. We have $B^{\mathcal{I}} = B^{\mathcal{I}_1}$ since B is $\overleftarrow{\mathrm{B}}$-reachable, and $A^{\mathcal{I}} = A^{\mathcal{I}_1}$ if A is $\overleftarrow{\mathrm{B}}$-reachable, or $A^{\mathcal{I}} = \Delta^{\mathcal{I}_1}$ otherwise. For both cases it follows that $A^{\mathcal{I}} \not\subseteq B^{\mathcal{I}}$. It remains to be shown that $\mathcal{I}$ is indeed a model of M, and therefore satisfies all axioms $\beta = (\beta_L \sqsubseteq \beta_R)$ in M, including $A \sqsubseteq B$. There are two possibilities:

- $\beta \in M_1$. Since $M_1 \subseteq \mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$, all symbols in $\mathrm{Sig}(\beta_L)$ and one or more symbols in $\mathrm{Sig}(\beta_R)$ are $\overleftarrow{\mathrm{B}}$-reachable. Consequently, $\mathcal{I}_1$ and $\mathcal{I}$ coincide on the names occurring in β_L and since $\mathcal{I}_1$ is a model of M_1, we have that $(\beta_L)^{\mathcal{I}} = (\beta_L)^{\mathcal{I}_1}$ and $(\beta_R)^{\mathcal{I}_1} \subseteq (\beta_R)^{\mathcal{I}}$. Therefore $(\beta_L)^{\mathcal{I}} \subseteq (\beta_R)^{\mathcal{I}}$.

- $\beta \notin M_1$. Since $B \in M$, we have that $B \notin \mathcal{O}^{reach}_{\overleftarrow{\mathrm{B}}}$, and hence β is not $\overleftarrow{\mathrm{B}}$-reachable. Thus no $x \in \mathrm{Sig}(\beta_R)$ is $\overleftarrow{\mathrm{B}}$-reachable. By the definition of $\mathcal{I}$, $(\beta_R)^{\mathcal{I}} = \Delta^{\mathcal{I}_1}$. Hence $(\beta_L)^{\mathcal{I}} \subseteq (\beta_R)^{\mathcal{I}}$.

Therefore $\mathcal{I}$ is a model for M. But since $A^{\mathcal{I}} \not\subseteq B^{\mathcal{I}}$ we have that $M \not\models A \sqsubseteq B$ proving the contrapositive. □

Algorithm 2 shows one method to extract a top-down reachability based module given an $\mathcal{EL}^+$ $\mathcal{O}$ and a signature S as input. In the context of the algorithm `active-axioms`(x) are all those, and only those axioms $(\alpha_L \sqsubseteq \alpha_R) \in \mathcal{O}$ such that $x \in \mathrm{Sig}(\alpha_R)$, thus every such

Algorithm 2 (Extract top-down reachability-based module)

Procedure extract-top-down-module($\mathcal{O}$, S)
Input: $\mathcal{O}$ - $\mathcal{EL}^+$ ontology; S - signature
Output: $\mathcal{O}_S$: top-down reachability-based module for S in $\mathcal{O}$

```
1: O_S := ∅
2: queue := active-axioms(S)
3: while not empty(queue) do
4:   (α_L ⊑ α_R) := fetch(queue)
5:   O_S := O_S ∪ {α_L ⊑ α_R}
6:   queue := queue ∪ (active-axioms(Sig(α_L)) \O_S)
7: end while
8: return O_S
```

axiom is also by definition top-down reachable. For a signature S we define `active-axioms` $(S) := \bigcup_{x \in S}$`active-axioms`(x).

Theorem 11 (Algorithm 2 produces $\mathcal{O}^{reach}_{\overline{S}}$) *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology, n the number of axioms in $\mathcal{O}$, and $S \subseteq Sig(\mathcal{O})$ a signature. Algorithm 2 terminates after $O(n)$ steps and returns the top-down reachability-based module for S in $\mathcal{O}$.*

Proof: The proof consists of three parts: There are at most n axioms that may be added to $\mathcal{O}_S$, once they are added they are never removed. After each addition, the queue is augmented by `active-axioms(.)`, but every axiom in `active-axioms(.)` is by definition reachable from the S, therefore every axiom will only ever be added to the queue once and once added, never considered again. Assuming that each operation runs in constant time, the algorithm's runtime is $O(n)$.

Let $\mathcal{O}^i_S$ be the value of $\mathcal{O}_S$ the algorithm has computed at the ith iteration, and $\mathcal{O}^\infty_S$ be the output after no more active axioms are to be processed. We prove by induction on i that all axioms in $\mathcal{O}^i_S$ are $\overleftarrow{\overline{S}}$-reachable. The base case is trivial since $\mathcal{O}^0_S$ is empty. For the induction step, assume that $\alpha_L \sqsubseteq \alpha_R \in \mathcal{O}^{i+1}_S$ is the new axiom added at iteration $i+1$. This is possible only when some $x \in \text{Sig}(\alpha_R)$ is $\overleftarrow{\overline{S}}$-reachable. By the induction hypothesis, all axioms in $\mathcal{O}^i_S$ are $\overleftarrow{\overline{S}}$-reachable, and every axiom in `active-axioms(.)` is $\overleftarrow{\overline{S}}$-reachable, thus α_R is $\overleftarrow{\overline{S}}$-reachable. Therefore, $\alpha_L \sqsubseteq \alpha_R \in \mathcal{O}^i_S$ is $\overleftarrow{\overline{S}}$-reachable as required.

It remains to be shown that the algorithm extracts all $\overleftarrow{\overline{S}}$-reachable axioms. An axiom $\alpha = (\alpha_L \sqsubseteq \alpha_R)$ is $\overleftarrow{\overline{S}}$-reachable if some symbol in $\text{Sig}(\alpha_R)$ is $\overleftarrow{\overline{S}}$-reachable. All potential $\overleftarrow{\overline{S}}$-reachable symbols according to Definition 31 are considered through the way the algorithm initializes and maintains `queue`. In fact, it starts with the set of all `active-axioms(S)`

which corresponds to the base case in Definition 31. Then, it recursively extends `queue` with all `active-axioms`(x) for $x \in (\text{Sig}(\alpha_L)$, since all `active-axioms`$(\alpha_L)$ are $\overleftarrow{\text{S}}$-reachable, this corresponds to the induction case in Definition 31. □

Though top-down reachability-based modules as defined above are correct, the following example shows that they contain many irrelevant axioms in themselves and the decision procedure regarding the inclusion of axioms containing existential restrictions on the right hand side of an axiom may possibly be further improved, leading to possible better top-down reachability heuristics. Investigating these are however beyond the scope of this dissertation.

Example 5 *Given the small ontology $\mathcal{O}$ below, as well as $A \sqsubseteq B$ as the statement of interest, if we require that any one of the symbols of an existential restriction be top-down reachable from B, both axioms 3.7 and 3.8 will be chosen. Since $\mathcal{O} \not\models A \sqsubseteq B$, axioms like these will result in modules that include many unnecessary axioms.*

$$A \sqsubseteq \exists r.C \tag{3.7}$$

$$C \sqsubseteq B \tag{3.8}$$

3.4 Bidirectional Reachability-based Modules

One of the motivations behind module extraction methods in Description Logics is to increase the performance of reasoning tasks as well as reduce the complexity of debugging ontologies. Lutz et al. (2007) have shown that the problem of extracting minimal modules (*conservative extensions*) for $\mathcal{ALC}$ is undecidable, and various approximation methods, such as locality-based modules, have been introduced in the literature. These methods aim to extract small modules in polynomial time that are not necessarily minimal.

In this section we focus on extracting modules for specific entailments, with the aim of improving reasoning performance. We have seen that one criticism that may be raised against bottom-up reachability module extraction methods is that they only extract axioms based on the subsumee of an entailment, yet never consider the subsumer of the entailment to further reduce the number of axioms in the module. The same criticism may be raised against the top-down reachability based extraction method from Section 3.3, in that the top-down reachability heuristic only considers the subsumer of an entailment and never the subsumee in the pruning process.

Given an ontology $\mathcal{O}$ and an entailment $\mathcal{O} \models A \sqsubseteq B$ we have that bottom-up reachability $\mathcal{O}_A^{reach}$ preserves all entailments in terms of the sub-concept A and $\mathcal{O}_{\overleftarrow{\mathrm{B}}}^{reach}$ preserve all entailments in terms of the super-concept B. In order to introduce modules that consider both the subsumee and subsumer of an entailment we note the following:

Theorem 12 *Given the axiom $\alpha = \alpha_L \sqsubseteq \alpha_R$:*

- *$\mathcal{O}_A^{reach}$ will contain α if, and only if, α_L is A-reachable, and*
- *$\mathcal{O}_{\overleftarrow{\mathrm{B}}}^{reach}$ will contain α if, and only if, α_R is $\overleftarrow{\mathrm{B}}$-reachable*

Proof: The proof follows trivially from the definitions of $\mathcal{O}_A^{reach}$ and $\mathcal{O}_{\overleftarrow{\mathrm{B}}}^{reach}$. □

The definitions of bottom-up reachability and top-down reachability are both inductive, and the inclusion of any axiom in each of these sets, is dependent on the inclusion of all axioms that it requires to conform to its respective definition. Therefore, the set $\mathcal{O}_A^{reach} \cap \mathcal{O}_{\overleftarrow{\mathrm{B}}}^{reach}$ contains all axioms that are simultaneously both A-reachable and $\overleftarrow{\mathrm{B}}$-reachable.

It is easy to show that top-down reachability-based modules are equivalent to a subset of $\top$-locality modules (Cuenca Grau et al. 2008, Sattler et al. 2009). These modules can be criticized in a similar manner to bottom-up reachability-based modules, in that they include many irrelevant axioms. Combining $\bot$-locality modules with $\top$-locality based modules allows us to extract so called nested locality modules denoted by $\top\bot$ or $\bot\top$ (Sattler et al. 2009). We introduce a slightly different form of module called *bidirectional reachability-based modules*, aimed towards finding small modules preserving subsumption relationships between single concept names. We thus define the *bidirectional reachability-based module* denoted by $\mathcal{O}_{A \leftrightarrow B}^{reach}$ as follows:

Definition 33 (Bidirectional reachability-based module) *The bidirectional reachability-based module, denoted $\mathcal{O}_{A \leftrightarrow B}^{reach}$, for the statement $A \sqsubseteq B$ in terms of $\mathcal{O}$, is defined as the set of all axioms $\alpha_L \sqsubseteq \alpha_R \in \mathcal{O}$ such that:*

- *for every $x_i \in Sig(\alpha_L)$, x_i is A-reachable in terms of $\mathcal{O}$, and*
- *α_R is $\overleftarrow{\mathrm{B}}$-reachable in terms of $\mathcal{O}$*

Any non-empty subset $\mathcal{S} \subseteq \mathcal{O}_{A \leftrightarrow B}^{reach}$ such that $\mathcal{S}_{A \leftrightarrow B}^{reach} = \mathcal{S}$ is called a bidirectional reachability-based sub-module of $\mathcal{O}$ in terms of $\mathcal{S}$ for the statement $A \sqsubseteq B$. $\mathcal{S}_{A \leftrightarrow B}^{reach}$ is minimal if there exists no non-empty $\mathcal{S}_1 \subset \mathcal{S}$ such that $\mathcal{S}_1{}_{A \leftrightarrow B}^{reach} = \mathcal{S}_1$.

Figure 3.1: Sample ontology

α_1	A	$\sqsubseteq$	C_1
α_2	A	$\sqsubseteq$	D
α_3	D	$\sqsubseteq$	C_3
α_4	C_1	$\sqsubseteq$	$\exists R.C_2$
α_5	C_2	$\sqsubseteq$	C_3
α_6	$C_3 \sqcap C_4$	$\sqsubseteq$	B
α_7	$\exists r.C_3$	$\sqsubseteq$	C_4
α_8	C_3	$\sqsubseteq$	B
α_9	C_2	$\sqsubseteq$	E
α_{10}	E	$\sqsubseteq$	F

These modules differ from nested locality modules as follows: Given a subsumption statement $A \sqsubseteq B$, and the $\bot\top$ module $\mathcal{O}'$, then $\mathcal{O}'$ will contain all axioms for the signature $S = \{A, B\}$, thus it will include all the axioms for the entailments $\mathcal{O}' \models A \sqsubseteq B$ and $\mathcal{O}' \models B \sqsubseteq A$. A bidirectional reachability-based module $\mathcal{O}''$, however, only contains axioms for the entailment $\mathcal{O}'' \models A \sqsubseteq B$. Using the notation that $\bot\{A, B\}$ represents the $\bot$ locality module for the signature $\{A, B\}$ the relationship between these modules can be illustrated as follows:

$$\mathcal{O}^{reach}_{A \leftrightarrow B} \subseteq \bot\{A\}\top\{B\} \subseteq \bot\{A\}\top\{B\} \cup \bot\{B\}\top\{A\} \subseteq \bot\top\{A, B\} \subseteq \bot\{A, B\}$$

Nested locality based modules can thus be considered as bidirectional reachability-based modules for general signatures.

The following example shows the relationship between a bidirectional reachability-based module, bidirectional reachability-based sub-modules and minimal bidirectional reachability-based modules.

Example 6 *Given the ontology $\mathcal{O}$ in Figure 3.1, the entailment $\mathcal{O} \models A \sqsubseteq B$, we have that:*

- $\mathcal{O}^{reach}_{A} = \mathcal{O}$,
- $(\mathcal{O}^{reach}_{A})^{reach}_{\overleftarrow{\mathrm{B}}} = \mathcal{O}^{reach}_{A \leftrightarrow B}$ *consist of axioms:*

$$\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7, \alpha_8$$

- *Given that the sets S_0, S_1, S_2 and S_3 are defined as follows:*

$$S_0 = \{\alpha_1, \alpha_4, \alpha_5, \alpha_6, \alpha_7\}$$

$$S_1 = \{\alpha_2, \alpha_3, \alpha_8\}$$

$$S_2 = \{\alpha_1, \alpha_4, \alpha_5, \alpha_8\}$$

$$S_3 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5, \alpha_6, \alpha_7\}$$

- *S_0, S_1, S_2 and S_3 are bidirectional reachability-based sub-modules $\mathcal{S}_i \subseteq \mathcal{O}^{reach}_{A\leftrightarrow B}$, that is, $\mathcal{S}_i = \mathcal{S}_i{}^{reach}_{A\leftrightarrow B}$.*
- *S_1 and S_2 are both minimal bidirectional reachability-based sub-modules with S_1 being the only one of these sets that is both a minimal bidirectional reachability-based sub-module and a MinA for the statement $A \sqsubseteq B$.*
- *S_3 is a MinA for the statement $A \sqsubseteq B$ but S_3 is not a minimal bidirectional reachability-based module.*

Since $\mathcal{O}^{reach}_{A\leftrightarrow B}$ consist of all axioms in $\mathcal{O}^{reach}_{A} \cap \mathcal{O}^{reach}_{\overline{\text{B}}}$, we may now extract $\mathcal{O}^{reach}_{A\leftrightarrow B}$ by finding $(\mathcal{O}^{reach}_{A})^{reach}_{\overleftarrow{\text{B}}}$ or $(\mathcal{O}^{reach}_{\overleftarrow{\text{B}}})^{reach}_{A}$. This can be achieved by applying algorithms 1 and 2 in any order and in sequence. Since $\mathcal{O}^{reach}_{\overline{\text{B}}}$ is in general very large, our algorithm extracts $(\mathcal{O}^{reach}_{A})^{reach}_{\overleftarrow{\text{B}}}$.

Algorithm 3 (Extract bidirectional reachability-based module)

Procedure extract-bidirectional-module($\mathcal{O}$, $A \sqsubseteq B$)
Input: $\mathcal{O}$ - $\mathcal{EL}^+$ ontology; $A \sqsubseteq B$ - entailment
Output: $\mathcal{O}^{reach}_{A\leftrightarrow B}$

```
1:  O_t := O_A^reach
2:  return O_t^reach_B←
```

Theorem 13 (Algorithm 3 produces $\mathcal{O}^{reach}_{A\leftrightarrow B}$) *Let $\mathcal{O}$ be an $\mathcal{EL}^+$ ontology, n the number of axioms in $\mathcal{O}$, and $\mathcal{O} \models A \sqsubseteq B$ an entailment. Algorithm 3 terminates after $O(n^2)$ steps in the worst case and returns the bidirectional reachability based module for $\mathcal{O} \models A \sqsubseteq B$.*

Proof: The proof consists of three parts: There are at most n axioms in $\mathcal{O}$, once they are removed they are never added again. At Line 1, $\mathcal{O}^{reach}_{A}$ terminates after at most n^2 operations in the worst case. After this no axioms are added to $\mathcal{O}_t$. $\mathcal{O}_t{}^{reach}_{\overleftarrow{\text{B}}}$ at Line 2 terminates after at most n operations in the worst case. In total the algorithm terminates after performing at most $n + n^2 = \text{O}(n^2)$ operations in the worst case

By Theorem 9, after extracting $\mathcal{O}^{reach}_{A}$ at Line 1, $\mathcal{O}_t$ contains only those axioms that are A-reachable in terms of $\mathcal{O}$. By Theorem 11, extracting $\mathcal{O}_t{}^{reach}_{\overleftarrow{\text{B}}}$ at Line 2 will remove all axioms in $\mathcal{O}_t$ that are not $\overleftarrow{\text{B}}$-reachable in terms of $\mathcal{O}_t$. But $\mathcal{O}_t \subseteq \mathcal{O}$, thus every $\overleftarrow{\text{B}}$-reachable

axiom in $\mathcal{O}_t$ is by definition also $\overleftarrow{\text{B}}$-reachable in terms of $\mathcal{O}$. Since the resultant module contains all those axioms that are A-reachable and simultaneously $\overleftarrow{\text{B}}$-reachable, it contains only those axioms that are both A-reachable and $\overleftarrow{\text{B}}$-reachable. □

Algorithm 3 extracts the bidirectional reachability-based module for the statement $A \sqsubseteq B$ in terms of the ontology $\mathcal{O}$ by using iterative applications of bottom-up and top-down reachability-based module extraction methods. By Theorem 9, extracting $\mathcal{O}_A^{reach}$ results in a module that contain all A-reachable axioms. This module is unique in terms of the ontology $\mathcal{O}$ and the signature $\{A\}$, since only one such module exists for A in $\mathcal{O}$. By Theorem 11 the same uniqueness principle holds for $\mathcal{O}_{\overleftarrow{\text{B}}}^{reach}$, that is $\mathcal{O}_{\overleftarrow{\text{B}}}^{reach}$ is unique in terms of the symbol B and the ontology $\mathcal{O}$. It follows that after each iteration of any of these algorithms, that the remaining set of axioms is unique in terms of the ontology and the statement of interest, thus upon termination of the algorithm, the set $\mathcal{O}_{A\leftrightarrow B}^{reach}$ is unique in terms of the ontology $\mathcal{O}$ and the statement $A \sqsubseteq B$.

From Theorem 8 we have that bottom-up reachability-based modules are strong subsumption modules and from Theorem 10 we have that top-down reachability based modules are strong top-down subsumption modules, we therefore have that an iterative application of these extraction methods preserve all MinAs at each iteration. Thus upon completion of Algorithm 3 $\mathcal{O}_{A\leftrightarrow B}^{reach} \models A \sqsubseteq B$ only if $\mathcal{O} \models A \sqsubseteq B$. $\mathcal{O}_{A\leftrightarrow B}^{reach}$ is thus a strong bidirectional subsumption module for the statement $A \sqsubseteq B$ in terms of $\mathcal{O}$.

Theorem 14 *$\mathcal{O}_{A\leftrightarrow B}^{reach}$ preserves all MinAs for $\mathcal{O} \models A \sqsubseteq B$. ($\mathcal{O}_{A\leftrightarrow B}^{reach}$ is a strong subsumption module for $A \sqsubseteq B$ in $\mathcal{O}$)*

3.5 Conclusion

In this chapter we investigated module extraction methods with the focus on syntactic traversal-based algorithms. We showed that a top-down version of reachability-based module extraction preserves all MinAs in terms of the super-concept of an entailment. The top-down reachability and bottom-up reachability were then combined to form a bidirectional version of reachability, which considers both the subsummee and the subsumer of a subsumption entailment in order to extract smaller modules. Bidirectional reachability-based modules are strong subsumption modules and thus preserve all MinAs for a subsumption statement. In the next chapter we investigate the exact nature of the relationship between MinAs and

bidirectional reachability-based sub-modules more closely.

Chapter 4

MinA Extraction

In this chapter we investigate the relationship between MinAs and bidirectional reachability-based sub-modules and how MinA extraction may benefit from bidirectional module extraction methods. By definition every MinA is a minimal module, and since bidirectional reachability-based modules are strong subsumption modules they preserve all MinAs for the statement $A \sqsubseteq B$, thus we have that every MinA is by definition a bidirectional reachability-based sub-module (Theorem 15). What is unclear is the exact nature of the bidirectional reachability-based sub-module corresponding to a MinA.

We start this chapter by investigating a subset of the common approaches to MinA extraction methods as categorized into black-box and glass-box approaches. In Section 4.2 we investigate the relationship between MinAs and minimal bidirectional based sub-modules for the very inexpressive DL $\mathcal{HL}$. We show that the problem of finding MinAs for $\mathcal{HL}$ syntactically resembles that of parsing sentences using Context Free Grammars. As a result of this syntactic resemblance we show how the popular Earley algorithm (Earley 1970) for parsing with Context Free Grammars may be used to extract an indexed bidirectional reachability-based module from which every $\mathcal{HL}$ MinA for an entailment may be obtained without performing a single subsumption test. In Section 4.3 we investigate how this method may be extended to members of the $\mathcal{EL}$ family of DLs. We show that for $\mathcal{EL}$ TBoxes consisting of only primitive concept definitions, the algorithms for $\mathcal{EL}$ may be used directly to extract all MinAs. However, when general $\mathcal{EL}$ TBoxes are introduced, there is an exponential explosion in the number of bidirectional reachability-based sub-modules for a single MinA. With the exception of Sections 4.1 and 4.2.2, all the work presented in this chapter constitute original contributions made by this dissertation.

4.1 Introduction to MinA Extraction

The term *axiom pinpointing*, coined by Schlobach & Cornet (2003), refers to the process of obtaining justifications (explanations) for entailments and plays an important role in the debugging of ontologies. As ontologies grow in size, from a few hundred to hundreds of thousands of axioms, finding a minimal set of axioms responsible for a wrongful entailment becomes extremely difficult for the ontology developer. Axiom Pinpointing algorithms aim to simplify this process by extracting Minimal Axioms Sets (MinAs) for an entailment.

Suntisrivaraporn (2009) shows with the following example that there exists an $\mathcal{HL}$ ontology $\mathcal{O}$ that contains exponentially many MinAs for an entailment.

Example 7 (Exponentially many MinAs) *For all $n \geq 1$, define $\mathcal{T}_n$ to be an acyclic $\mathcal{HL}$ TBox consisting of the following concept inclusions:*

$$\begin{aligned}
A &\sqsubseteq P_1 \sqcap Q_1 \\
P_i &\sqsubseteq P_{i+1} \sqcap Q_{i+1} \quad \textit{for } 1 \leq i \leq n \\
Q_i &\sqsubseteq P_{i+1} \sqcap Q_{i+1} \quad \textit{for } 1 \leq i \leq n \\
P_n &\sqsubseteq B \\
Q_n &\sqsubseteq B
\end{aligned}$$

The size of $\mathcal{T}_n$ is linear in n, and we have the consequence $\mathcal{T}_n \models A \sqsubseteq B$. There are 2^n MinAs for $\mathcal{T}_n \models A \sqsubseteq B$ since, for each i such that $1 \leq i \leq n$, it suffices to have either P_i's or Q_i's definition. That is, either $P_i \sqsubseteq P_{i+1} \sqcap Q_{i+1}$ or $Q_i \sqsubseteq P_{i+1} \sqcap Q_{i+1}$ in the case that $i \leq n$, and either $P_n \sqsubseteq B$ or $Q_n \sqsubseteq B$. For the case where $n = 2$ we have that $\mathcal{T}_n$ consists of the following axioms:

$$\begin{aligned}
&\alpha_1 : A \sqsubseteq P_1 \sqcap Q_1 \qquad \alpha_4 : P_2 \sqsubseteq B \\
&\alpha_2 : P_1 \sqsubseteq P_2 \sqcap Q_2 \qquad \alpha_5 : Q_2 \sqsubseteq B \\
&\alpha_3 : Q_1 \sqsubseteq P_2 \sqcap Q_2
\end{aligned}$$

With the final MinAs($2^2 = 4$) being:

$$\begin{aligned}
MinA_1 :&\quad \alpha_4 \quad \alpha_2 \quad \alpha_1 \\
MinA_2 :&\quad \alpha_4 \quad \alpha_3 \quad \alpha_1 \\
MinA_3 :&\quad \alpha_5 \quad \alpha_2 \quad \alpha_1 \\
MinA_4 :&\quad \alpha_5 \quad \alpha_3 \quad \alpha_1
\end{aligned}$$

The exponential behaviour of extracting all MinAs for an $\mathcal{HL}$ entailment, even with an unfoldable TBox, naturally extends to all possible DLs. Since the worst case behaviour of extracting all MinAs for an entailment is unavoidable, researchers have endeavoured to find practical algorithms that minimize the cost of finding MinAs as much as possible. There are two main approaches to axiom pinpointing namely *black-box* and *glass-box* methods.

4.1.1 Black-box Algorithms

'Black-box' algorithms utilize an existing DL reasoner's standard inference engines, such as subsumption, without the need to internally modify the algorithm. These black-box algorithms systematically eliminate unwanted axioms from consideration by using an elimination function. After each iteration of this elimination function, a standard subsumption test is run in order to test whether the entailment still holds in the resultant set of axioms. This iterative procedure continues until no more axioms may be removed without invalidating the entailment. Since subsumption testing is expensive even for the inexpressive DL $\mathcal{EL}^+$, which runs in $O(n^4)$ time in the number of axioms, black-box algorithms aim to reduce as many axioms as possible during every iteration of the elimination procedure, thus reducing the total number of subsumption tests needed to extract each MinA. Various algorithms have been proposed in the literature, including:

- **Sliding Window**: This technique is introduced in Kalyanpur et al. (2007) as part of their fast pruning algorithm which tests unsatisfiability of a concept. The idea here is to use a window of n axioms which *slides* across all axioms in a possible MinA. Removing axioms from this window allows us to test if the concept remains unsatisfiable or not w.r.t the removed axioms. If the concept remains unsatisfiable, then the whole window of n axioms can be eliminated. Thus the algorithm can remove chunks of n axioms at once, reducing the size of the search space by n axioms during each iteration. The window is initially set to one tenth of the number of axioms, this size is then systematically reduced in the case where the set contains a relevant axiom. The pruning process is repeated until the window contains only those axioms that may not be removed.

- **Logarithmic algorithm (Suntisrivaraporn 2009)**: Instead of going through the axioms one by one and pruning one axiom at a time, the algorithm partitions the ontology into two halves. It then employs a DL reasoner to check whether one of the halves preserves the entailment. If the answer is 'yes,' it immediately recurses on that

half, throwing away half of the axioms in one step. Intuitively, this means that the essential axioms in a MinA are all in one of the partitions. Otherwise, i.e., if the answer is 'no,' essential axioms are spread over both partitions of the ontology. In this case, the algorithm recurses on each half, while using the other half as the "support set".

- **Relevance based selection**: Qi et al. (2008) introduced a relevance based selection function in order to obtain a MinA. Two axioms α, β are directly relevant if they share a common symbol name. The selection function is recursive, in that at each iteration it adds new axioms that become relevant to the axioms added at the previous iteration. The axioms introduced at each iteration form an ordering relation, where axioms introduced later are *further* away from the initial concept, i.e. axioms introduced in the final iterations are less relevant than axioms introduced in the first iterations. They show that in order to extract a justification that is more relevant to a concept it suffices to extract the more relevant axioms from the ontology.

Once an optimized algorithm has been designed to extract a single MinA, most algorithms employ some variation of Reiter's *Hitting Set Tree (HST)* algorithm in order to extract all MinAs (Reiter 1987). Hitting sets are defined as follows:

Definition 34 (Hitting set) *Given a universal set U, and a set $S = \{s_1, \cdots, s_n\}$ of subsets of U that are conflict sets, i.e. subsets of the system components responsible for the error. A hitting set T for S is a subset of U such that $s_i \cap T \neq \emptyset$ for all $1 \leq i \leq n$. A minimal hitting set T for S is a hitting set such that no $T' \subset T$ is a hitting set for S. A hitting set T is cardinality-minimal if there is no other hitting set T' such that $|T'| \leq |T|$.*

Reiter's algorithm is used to calculate minimal hitting sets for a collection $S = \{s_1, \cdots, s_n\}$ of sets by constructing a labeled tree, called a *Hitting Set Tree (HST)*. In a HST, each node is labeled with a set $s_i \in S$, and each edge is labeled with an element in $\cup_{s_i \in S} s_i$. For each node n in a HST, let $H(n)$ be the set of edge labels on the path from the root of the HST to n. Then the label for n is any set $s \in S$ such that $s \cap H(n) = \emptyset$, if such a set exists. Suppose s is the label of a node n, then for each $\sigma \in s$, n has a successor n_σ connected to n by an edge with σ in its label. If the label of n is the empty set, then we have that $H(n)$ is a hitting set of S.

In the case of finding justifications for an entailment, the universal set describing the system corresponds to the total set of axioms in the ontology, and a justification corresponds

to a single conflict set. The algorithm correctly and dynamically finds all conflict sets *(justifications)*.

Suntisrivaraporn et al. (2008) found that by first extracting the standard bottom-up reachability-based module, and then utilizing Reiter's HST algorithm with a relevance based selection function, the performance of MinA extraction is increased by an order of magnitude. Module extraction therefore plays an important role in reducing the search space for Black-box algorithms.

4.1.2 Glass-box Algorithms

The term 'glass box' refers to the visibility of the internal workings of a DL reasoner's algorithms. The glass-box approach to MinA extraction takes an existing DL algorithm for testing logical entailments and modifies it to generate all MinAs for the entailment in question. All previous work on glass-box methods essentially extends a descision procedure, such as a tableaux based subsumption testing algorithm, with a labelling function that keeps information about axioms during its computation cycle. It then uses this information to generate a monotone Boolean formula that represents all possible sets of axioms that form part of MinAs (Suntisrivaraporn 2009).

Since the internal workings of specific reasoning algorithms needs to be modified, a disadvantage of glass-box approaches is that they are usually tailored for very specific DLs (Schlobach & Cornet 2003, Kalyanpur et al. 2007, Baader & Hollunder 1995, Meyer et al. 2006, Suntisrivaraporn 2009). However, Baader & Penaloza (2007) considered a glass-box approach for arbitrary DLs. Another disadvantage of glass-box based algorithms is that they alter the complexity of the decision algorithm when extended with a labelling functionality. As an example, in Suntisrivaraporn (2009) the quartic $O(n^4)$ classification algorithm deteriorates to a exponential worst case classification algorithm when augmented with labelling functionality.

4.2 Computing and Extracting all MinAs in $\mathcal{HL}$

In this section we investigate MinA extraction in the DL $\mathcal{HL}$ and the relationship between a single MinA and a minimal bidirectional reachability-based sub-module. First we look at some properties of MinAs in $\mathcal{HL}$ that are of interest in designing an algorithm based on minimal bidirectional reachability-based sub-modules. Then we introduce the Earley

algorithm for parsing with Context Free Grammars and modify the algorithm to compute an indexed bidirectional reachability-based module from which all possible sub-modules may be extracted. Each of these sub-modules therefore corresponds to a bidirectional reachability-based sub-module, some of which are minimal bidirectional reachability-based sub-modules. Lastly, we introduce an algorithm to extract all MinAs from the module returned by the modified Earley algorithm.

4.2.1 Relationship Between MinAs and Bidirectional Reachability-based sub-modules in $\mathcal{HL}$

In this section we investigate the relationship between $\mathcal{HL}$ MinAs and bidirectional reachability-based sub-modules. Recall from Definition 3 that the sublanguage $\mathcal{HL}$ does not allow for roles and existential restrictions, with the name motivated by the fact that GCIs involving $\mathcal{HL}$ concepts are essentially propositional Horn clauses (Suntisrivaraporn 2009).

From Theorem 14 we know that the set $\mathcal{O}^{reach}_{A\leftrightarrow B}$ contains all MinAs for the statement $A \sqsubseteq B$. Thus if M_1 is a MinA for the statement $A \sqsubseteq B$, then it follows that M_1 is a bidirectional reachability-based sub-module for the statement $A \sqsubseteq B$ in terms of itself. Formally:

Theorem 15 *Given an $\mathcal{HL}$ TBox $\mathcal{T}$ and the statement $A \sqsubseteq B$ such that $\mathcal{T} \models A \sqsubseteq B$. Let M_1 be a MinA for $A \sqsubseteq B$, then $M_1{}^{reach}_{A\leftrightarrow B} = M_1$.*

Proof: By Definitions 25 and 33 and Theorem 14 we know that $M_1 \subseteq M_1{}^{reach}_{A\leftrightarrow B}$. But $M_1{}^{reach}_{A\leftrightarrow B}$ is a bidirectional reachability-based sub-module for the ontology M_1 and thus by definition $M_1{}^{reach}_{A\leftrightarrow B} \subseteq M_1$. Since $M_1 \subseteq M_1{}^{reach}_{A\leftrightarrow B}$ and $M_1{}^{reach}_{A\leftrightarrow B} \subseteq M_1$, we conclude that $M_1 = M_1{}^{reach}_{A\leftrightarrow B}$. Thus every MinA M for $A \sqsubseteq B$ is of the form $M^{reach}_{A\leftrightarrow B}$. □

The absence of role names and existential restrictions in $\mathcal{HL}$ has the following effect:

Lemma 3 *Given an general $\mathcal{HL}$ TBox $\mathcal{T}$ and the statement $A \sqsubseteq B$ such that $\mathcal{T} \models A \sqsubseteq B$, then for every axiom $\alpha_L \sqsubseteq \alpha_R \in \mathcal{T}^{reach}_A$ we have that $\mathcal{T}^{reach}_A \models A \sqsubseteq \alpha_R$.*

Proof: Since the definition of $\mathcal{T}^{reach}_A$ is inductive and Algorithm 1 extracts $\mathcal{T}^{reach}_A$ we prove this lemma by using Algorithm 1 and induction. The initialization at Step 2 adds all *active* axioms to the queue. These axioms are all axioms of the form $A \sqsubseteq \alpha_R$ or $\top \sqsubseteq \alpha_R$. Each of these axioms will be added to $\mathcal{O}_S$ by Lines 5 and 6. Thus for these axioms it follows trivially that $\mathcal{O}_S \models A \sqsubseteq \alpha_R$. Assume that after n iterations of the algorithm, each axiom $\alpha_L \sqsubseteq \alpha_R$ in

$\mathcal{O}_S$ has the property $\mathcal{O}_S \models A \sqsubseteq \alpha_R$. Now let $\sigma_L \sqsubseteq \sigma_R$ be the axiom added to $\mathcal{O}_S$ at iteration $n+1$. The test at Line 5 ensures that for all $x \in \text{Sig}(\sigma_L)$ it is the case that $x \in \text{Sig}(\mathcal{O}_S)$. But x may only be an element in $\text{Sig}(\mathcal{O}_S)$ if $x = A, x = \top$ or x appears on the right hand side of at least one of the axioms in $\mathcal{O}_S$. For all these cases it follows that $\mathcal{O}_S \models A \sqsubseteq \sigma_L$ and thus that $\mathcal{O}_S \models A \sqsubseteq \sigma_R$. Therefore, by induction it follows that for every axiom $\alpha_L \sqsubseteq \alpha_R$ in $\mathcal{T}_A^{reach}$ we have that $\mathcal{T}_A^{reach} \models A \sqsubseteq \alpha_R$. □

An interesting result that directly follows from this lemma is that any two axioms, $\alpha_{L_1} \sqsubseteq x$ and $\alpha_{L_2} \sqsubseteq x \in \mathcal{T}_A^{reach}$ that share the same symbol on the right hand side, have the property that $\mathcal{T}_A^{reach} \models A \sqsubseteq x$. Since the TBox is acyclic and both these axioms have the same concept on the right hand side, none of these axioms may *use* (Definition 7) each other. Thus they are independent of each other and may not both form part of the same MinA.

Theorem 16 *Given an acyclic general $\mathcal{HL}$ TBox $\mathcal{T}$ in normal form, the statement $A \sqsubseteq B$ and a MinA M_1 for $A \sqsubseteq B$, then for every $x \in Sig(M_1)$ there is at most one axiom $\alpha_L \sqsubseteq x \in M_1$.*

Proof: We give a proof by contradiction. Let M_1 be a MinA for $A \sqsubseteq B$. Suppose that for some $x \in Sig(M_1)$ there are two axioms $\alpha_{L_1} \sqsubseteq x$ and $\alpha_{L_2} \sqsubseteq x$ in M_1. Then by Lemma 3 we have that for each of these axioms $M_1 \models A \sqsubseteq x$. Now let $M_2 = M_1 \setminus (\alpha_{L_1} \sqsubseteq x)$. Since $\mathcal{T}$ is acyclic we know that neither one of these axioms *uses* the other and therefore the inclusion of either of these axioms in M_1 is not dependent at all on the inclusion of the other. Now extract $(M_2)_A^{reach}$, since the inclusion of $\alpha_{L_2} \sqsubseteq x$ is not dependent on the symbol x as introduced by $\alpha_{L_1} \sqsubseteq x$, we have that $\alpha_{L_2} \sqsubseteq x \in (M_2)_A^{reach}$. But both $\alpha_{L_1} \sqsubseteq x$ and $\alpha_{L_2} \sqsubseteq x$ introduce the same symbol x, hence all axioms $\alpha_L \sqsubseteq \alpha_R \in M_1$ that require the symbol x to be active will still be included in M_2, and for each of these we have that $M_2 \models A \sqsubseteq \alpha_R$ by Lemma 3. Thus $M_2 \models A \sqsubseteq B$ and since $M_2 \subset M_1$ we conclude that M_1 is not a MinA. □

The above proof is applicable only to acyclic $\mathcal{HL}$ TBoxes in normal form. A similar result holds for cyclic $\mathcal{HL}$ TBoxes in normal form. For these ontologies whenever the said two axioms do not *use* each other, the proof is the same as above. However, when one of these axioms *uses* the other, the choice of which axiom is to be removed may not be arbitrarily decided upon but instead the axiom being *used* must be removed. The proof then proceeds directly as stated above.

Given that by Theorem 15 every MinA is a bidirectional reachability-based sub-module, we now prove that every MinA in $\mathcal{HL}$ is a *minimal* bidirectional reachability-based sub-module in terms of itself. This is quite a subtle point, because MinAs are already minimal. Note, however, that MinAs are minimal with respect to the property of entailing a given statement of interest, whereas bidirectional reachability-based sub-modules are minimal with respect to the syntactic requirement for both bottom-up reachability and top-down reachability.

Theorem 17 *Given an acyclic general $\mathcal{HL}$ TBox $\mathcal{T}$ in normal form, the statement $A \sqsubseteq B$ and a MinA M_1 for $A \sqsubseteq B$, then M_1 is a minimal bidirectional reachability-based sub-module $M_1{}^{reach}_{A \leftrightarrow B}$ in terms of M_1.*

Proof: From Theorem 15 we have that every MinA M_i for the statement $A \sqsubseteq B$ is a bidirectional reachability-based sub-module in terms of M_i .i.e $M_i = M_i{}^{reach}_{A \leftrightarrow B}$, and thus so is M_1. Suppose M_1 is not minimal, then M_1 must contain an axiom $\alpha_L \sqsubseteq x_i$ such that $M_2 = M_1 \setminus (\alpha_L \sqsubseteq x_i)$ contains some bidirectional reachability-based sub-module in terms of itself, say $M_3 \subseteq M_2$. Since $M_3 \subset M_1$ we have that M_3 is not a MinA and thus $M_3 \not\models A \sqsubseteq B$. Now for M_3 to be a bidirectional reachability-based sub-module in terms of itself, for every axiom $(\sigma_{L_1} \sqsubseteq \sigma_{R_1}) \in M_3$ and every $y_i \in \text{Sig}(\sigma_{L_1})$ we have that there must be some other axiom $(\sigma_{L_2} \sqsubseteq y_1) \in M_3$ in order for bidirectional reachability to be preserved. Since M_3 is a bidirectional reachability-based sub-module there is such an axiom for the symbol x_i namely $(\sigma_L \sqsubseteq x_i) \in M_3$. But by Theorem 16 there is at most one axiom in every MinA for each symbol x_i on the right hand side of an axiom. Thus there can be no axiom $(\sigma_L \sqsubseteq x_i) \in M_3$. We conclude that M_3 is not a bidirectional reachability-based sub-module in terms of itself and that M_1 is a minimal bidirectional reachability-based sub-module in terms of itself. □

Next we show that every minimal bidirectional reachability-based sub-module in $\mathcal{HL}$ for a statement $A \sqsubseteq B$ corresponds to a MinA.

Theorem 18 *Given a acyclic $\mathcal{HL}$ TBox $\mathcal{T}$ in normal form, the statement $A \sqsubseteq B$ such that $\mathcal{T} \models A \sqsubseteq B$ and a minimal bidirectional reachability-based sub-module $M_1{}^{reach}_{A \leftrightarrow B}$ with $M_1 \subseteq \mathcal{T}$, then M_1 is a MinA for $A \sqsubseteq B$.*

Proof: We prove by contradiction. Suppose M_1 is a minimal bidirectional reachability-based sub-module $M_1{}^{reach}_{A \leftrightarrow B}$ for the statement $A \sqsubseteq B$ but that $M_1 \not\models A \sqsubseteq B$. By definition of bidirectional reachability every axiom in $M_1{}^{reach}_{A \leftrightarrow B}$ is A-reachable and thus by Lemma 3 we have

for every axiom $(\alpha_L \sqsubseteq x_i) \in {M_1}^{reach}_{A \leftrightarrow B}$ that $M_1 \models A \sqsubseteq x_i$, including the case where $x_i = B$. This contradicts the assumption that $M_1 \not\models A \sqsubseteq B$. Since M_1 is minimal there exist only one axiom $(\alpha_L \sqsubseteq x_i) \in {M_1}^{reach}_{A \leftrightarrow B}$ for each x_i, thus $M_1 \models A \sqsubseteq x_i$ only once for each x_i. Hence by Theorem 16 M_1 is a MinA. □

Theorems 17 and 18 allows us to conclude that there is a one-to-one correspondence between minimal bidirectional reachability-based sub-modules and MinAs in $\mathcal{HL}$.

Corollary 1 *There is a one-to-one correspondence between MinAs and minimal bidirectional reachability-based sub-modules in $\mathcal{HL}$.*

One of the problems experienced with mapping normal form axioms in a MinA back to axioms in the original ontology, is that the resulting axiom set may no longer be minimal (Suntisrivaraporn 2009). Another interesting consequence of Theorem 16 is that in order to minimize the resultant set after mapping a MinA back to its original axioms, one does not have to decide arbitrarily which axioms to remove in order to minimize it. The theorem provides a good heuristic for this process: if a set is no longer minimal there must be two or more axioms where the same symbol appears on the right hand side. Starting the minimizing process at these axioms may provide a good pruning heuristic which may improve the performance of a minimizing algorithm.

4.2.2 The Earley Algorithm for Parsing Sentences with CFG's

In the previous section we saw that every MinA M_1 in an acyclic general $\mathcal{HL}$ TBox corresponds to a minimal bidirectional reachability-based sub-module ${\mathcal{M}_1}^{reach}_{A \leftrightarrow B}$. As there are an exponential number of possible MinAs for the DL $\mathcal{HL}$, it follows that there are an exponential number of minimal bidirectional reachability-based sub-modules. For any set P with n elements, there are a possible 2^n unique subsets, thus given $\mathcal{O}^{reach}_{A \leftrightarrow B}$ there exists an exponential number of bidirectional reachability-based sub-modules $\mathcal{S}^{reach}_{A \leftrightarrow B} \subseteq \mathcal{O}^{reach}_{A \leftrightarrow B}$ only some of which are minimal. Hence when extracting all possible minimal bidirectional reachability-based sub-modules and thus all MinAs in $\mathcal{HL}$ we not only wish to extract all bidirectional reachability-based sub-modules, but only those that are minimal.

In the rest of this chapter, whenever we refer to the term *module* we refer to the bidirectional reachability-based module for a statement of interest, every bidirectional reachability-based sub-module is then called a sub-module and a MinA is a minimal module. When

reference needs to be made to a different class of module it will be explicitly mentioned.

In this section we investigate a dynamic programming algorithm designed to compute all parse trees given a Context Free Grammar and a sentence to parse. The Earley algorithm for parsing with CFGs computes a representation of an exponential number of parse trees in $O(n^3)$ worst-case time, where n is the length of the input string. Each of the individual parse trees may then be extracted in linear time, without any additional computation. In the rest of this section we introduce the Earley algorithm as well as give a definition of Context Free Grammars. We then show the syntactic resemblance between CFG production rules and that of $\mathcal{HL}$ axioms in normal form. We show that the algorithm computes a representation of all sub-modules in the form of an indexed bidirectional reachability-based module, from which all minimal modules may be extracted.

Earley's parsing algorithm is a well known algorithm for parsing sentences given some Context Free Grammar (CFG) (Earley 1970). It computes all possible parse trees in parallel with a worst case running time of $O(n^3)$, where n is the length of the input string. Though the relationship between parsing sentences and extracting all sub-modules and hence also minimal modules is not apparent at first, the Earley algorithm is of interest due to the following:

- The algorithm uses a dynamic programming approach which reduces a potential exponential problem to one of polynomial complexity. This is achieved by employing a parallel left to right top-down depth-first searching method with bottom-up filtering to reduce the search space (Jurafsky & Martin 2008).

- Top-down searching with bottom-up filtering can be seen as abstractly resembling bidirectional reachability-based module extraction methods, where top-down searching can be viewed as extracting a top-down reachability-based module with bottom-up filtering limiting the axioms extracted to only those that form part of a bottom-up reachability-based module.

- When viewed in terms of top-down reachability, CFG production rules have a syntactic resemblance to $\mathcal{HL}$ GCI axioms in normal form. CFG production rules have the form $X \rightarrow Y_1 \dots Y_n$, whereas $\mathcal{HL}$ GCI axioms in normal form have the form $C_1 \sqcap \dots \sqcap C_n \sqsubseteq D$. Viewing these axioms from the right hand side and removing the conjunction operators we have $D \sqsupseteq C_1 \dots C_2$.

- The algorithm has been well studied and optimized (Chiang & Fu 1984), with existing hardware implementations available (Pavlatos et al. 2003), with a potential gain for utilizing these hardware implementations in order to extract MinAs.

Context free grammars (CFGs) provide a well-known method for modeling the structure of English and other natural languages (Jurafsky & Martin 2008). A grammar consists of a set of productions or rules, each of which expresses the ways the symbols (strings) in a language can be grouped together, as well as a lexicon of words or symbols.

Definition 35 (CFG production rules) *Let X represent a single non-terminal, the symbol 'a' represents a single terminal and α represent a mixed string of terminals and non-terminals, including the* null *string. CFG production rules have the form:*

$$X \rightarrow \alpha \tag{4.1}$$

$$X \rightarrow a \tag{4.2}$$

Definition 36 (Context Free Grammar) (Jurafsky & Martin 2008) *A Context Free Grammar (CFG) is defined as a 4-tuple (N, Σ, P, S) where:*

- *N is a set of non-terminal symbols, and*
- *σ is a set of terminal symbols disjoint from N, and*
- *a set P of productions, each of the form as defined in Definition 35, and*
- *a designated start symbol.*

A language is defined via derivation, i.e. where one string can be rewritten as a second one by a series of rule applications. Formally

Definition 37 (Language generated by a CFG) (Jurafsky & Martin 2008) *If $A \rightarrow \beta$ is a production of P and α and γ are any strings in the set $(\Sigma \cup N)^*$, then we say that $\alpha A \gamma$* directly derives *$\alpha\beta\gamma$, or $\alpha A \gamma \Rightarrow \alpha\beta\gamma$. Derivation is then a generalization of direct derivation. Let $\alpha_1, \alpha_2, \ldots, \alpha_m$ be strings in $(\Sigma \cup N)^*$, $m \geq 1$, such that*

$$\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{m-1} \Rightarrow \alpha_m$$

then we say that α_1 derives *α_m or $\alpha_1 \Rightarrow^* \alpha_m$. The language $\mathcal{L}_G$ generated by a grammar G is defined as the set of strings composed of terminal symbols which can be derived from the designated start symbol S : $\mathcal{L}_G = W \mid w$ is in Σ^* and $S \Rightarrow^* w$.*

Parsing refers to the process of mapping a string of words to its parse tree, assigning syntactic structure to it. The Earley parsing algorithm (Earley 1970) uses a dynamic programming approach, applying a single left-to-right, top-down, depth-first parallel search strategy to compute a chart that contains all possible parses for a given input. It accomplishes this in polynomial worst case time (n^3), where n is the size of the input string.

Example 8 *Consider the sample CFG for a subset of English grammar below. The set of symbols* $\{that, book, flight\}$ *represent terminal symbols and all other symbols represent non-terminals.*

$$\begin{aligned} S &\rightarrow VP \\ S &\rightarrow NP \quad VP \\ VP &\rightarrow VP \quad NP \\ NP &\rightarrow Det \quad Noun \\ VP &\rightarrow Verb \\ Det &\rightarrow that \\ Verb &\rightarrow book \\ Noun &\rightarrow flight \end{aligned}$$

During execution the Earley algorithm generates a state entry for each production rule it operates on. The purpose of the state is to record the progress made during the parsing process.

Definition 38 (Parse states) *Let X and Y represent single non-terminal symbols, let a represent a single terminal symbol, and let α, β and σ represent mixed strings of terminal and non-terminal symbols, including the* null *string. Then for each token (word) in the input string, the Earley algorithm creates a set of states, called a chart. A chart at position k of the input is represented by $\mathcal{C}_k$. Each state consists of a tuple ($X \rightarrow \alpha \bullet \beta$, i) where*

1. *$X \rightarrow \alpha\beta$ is the current production rule,*
2. *$\bullet$ indicates the* dot rule *which represents the current parsing position in the state, with $\alpha \bullet \beta$ indicating that α has previously been parsed and β is expected next, and*

3. i indicates the starting index of the substring where parsing of this production began.

Algorithm 4 (Earley parser (Jurafsky & Martin 2008)) *The parser consists of three sub-parts, the* predictor, scanner *and* completer. *Initially Chart C_0 contains all production rules for the start symbol S. For each state in chart $\mathcal{C}_i$, the tuple $(X \rightarrow \alpha \bullet \beta,\ j)$ is evaluated and the appropriate sub-part executed:*

1. ***Predictor:*** *If state $= (X \rightarrow \alpha \bullet Y\beta,\ j)$, then for every production $Y \rightarrow \sigma$, if $(Y \rightarrow \bullet\sigma) \notin \mathcal{C}_i$ then $\mathcal{C}_i := \mathcal{C}_i + (Y \rightarrow \bullet\sigma,\ i)$,*

2. ***Scanner:*** *If state $= (X \rightarrow \alpha \bullet a\beta,\ j)$, with a the next symbol in the input stream, and if $(X \rightarrow \alpha a \bullet \beta,\ j) \notin \mathcal{C}_{i+1}$ then $\mathcal{C}_{i+1} := \mathcal{C}_{i+1} + (X \rightarrow \alpha a \bullet \beta,\ j)$,*

3. ***Completer:*** *If state $= (X \rightarrow \gamma\bullet,\ j)$, then for every $(Y \rightarrow \alpha \bullet X\beta,\ k) \in \mathcal{C}_j$, $\mathcal{C}_i := \mathcal{C}_i + (Y \rightarrow \alpha X \bullet \beta,\ k)$.*

The algorithm executes all states iteratively in a top-down manner until no new states are available for processing, and no state may appear more than once in a given chart. The output of the Earley algorithm is defined as a set of $n+1$ charts, where n is the length of the input string. Each chart contains a set of parse states as defined in Definition 38. A successful parse is indicated by the state $S \rightarrow \alpha\bullet$ in chart $n+1$.

Example 9 *The following table shows the output of the Earley algorithm given the input string* 'book that flight' *and the CFG in Example 8. Each state entry consists of 4 components.*

- *the number of the entry in the set of all charts,*
- *production rule $T \rightarrow NT \ldots NT$,*
- *index j of the chart this entry was first introduced by the predictor, and*
- *the procedure and line number that placed this rule in current state.*

The state $(20.\ S \rightarrow VP\bullet)$ *in chart 3 represents a successful parse of the string "Book that flight".*

Chart 0:	• *book that flight*
1. $S \rightarrow \bullet\ NP\ VP$	$j = 0$ *: Inital State*
2. $S \rightarrow \bullet\ VP$	$j = 0$ *: Inital State*
3. $NP \rightarrow \bullet\ Det\ Noun$	$j = 0$ *: Predictor 1*
4. $VP \rightarrow \bullet\ VP\ NP$	$j = 0$ *: Predictor 2*
5. $VP \rightarrow \bullet\ Verb$	$j = 0$ *: Predictor 2*
6. $Det \rightarrow \bullet\ that$	$j = 0$ *: Predictor 3*
7. $Verb \rightarrow \bullet\ book$	$j = 0$ *: Predictor 5*

Chart 1:	*book* • *that flight*
8. $Verb \rightarrow book \bullet$	$j = 0$ *: Scanner 7*
9. $VP \rightarrow Verb \bullet$	$j = 0$ *: Completer 5, 8*
10. $VP \rightarrow VP \bullet NP$	$j = 0$ *: Completer 4, 9*
11. $NP \rightarrow \bullet\ Det\ Noun$	$j = 1$ *: Predictor 10*
12. $Det \rightarrow \bullet\ that$	$j = 1$ *: Predictor 11*
Chart 2:	*book that* • *flight*
13. $Det \rightarrow that \bullet$	$j = 1$ *: Scanner 12*
14. $NP \rightarrow Det \bullet Noun$	$j = 1$ *: Completer 11, 13*
15. $Noun \rightarrow \bullet\ flight$	$j = 2$ *: Predictor 14*
Chart 3:	*book that flight* •
16. $Noun \rightarrow flight \bullet$	$j = 2$ *: Scanner 15*
17. $NP \rightarrow Det\ Noun \bullet$	$j = 1$ *: Completer 14, 16*
18. $VP \rightarrow VP\ NP \bullet$	$j = 0$ *: Completer 10, 17*
19. $VP \rightarrow VP \bullet NP$	$j = 0$ *: Completer 4, 18*
20. $S \rightarrow VP \bullet$	$j = 0$ *: Completer 2, 18*

4.2.3 Converting an $\mathcal{HL}$ ontology in normal form to a CFG

Before the Earley algorithm may be applied to the problem of extracting a representation of all sub-modules and thus MinAs, where those sub-modules are minimal, we need to transform the given TBox to an equivalent CFG. In this subsection we demonstrate how this may be done given an $\mathcal{HL}$ ontology $\mathcal{O}$. In the discussion that follows, we assume that we have an $\mathcal{HL}$ TBox $\mathcal{T}$ in normal form and a statement $A \sqsubseteq B$ such that $\mathcal{T} \models A \sqsubseteq B$, where A and B are single concept names.

In order to be able to compute all sub-modules, every production rule introduced during the conversion process must preserve bidirectional reachability. Given $\mathcal{T}^{reach}_{A \leftrightarrow B}$, every axiom $\alpha_L \sqsubseteq \alpha_R \in \mathcal{T}^{reach}_{A \leftrightarrow B}$ has the following two properties:

1. every $x_i \in \text{Sig}(\alpha_L)$ is A-reachable, and
2. some $y_i \in \text{Sig}(\alpha_R)$ is $\overleftarrow{\text{B}}$-reachable.

In terms of A-reachability in $\mathcal{HL}$ we know that there exists special cases in which A-reachability implicitly holds for an axiom. For $\mathcal{HL}$, implicit A-reachable axioms have the form:

$$\top \sqsubseteq \alpha_R$$

In the steps that follow, all production rules we introduce have the form $y_i \rightarrow \sigma$, read as: the rule $y_i \rightarrow \sigma$ is bidirectionally reachable if the symbol y_i is $\overleftarrow{\text{B}}$-reachable and all symbols $x_i \in \sigma$ are A-reachable, which clearly conforms to the definition of bidirectional reachability. We further note that the symbols on the right hand side of CFG production rules have a fixed order, whereas the conjunction of $\mathcal{HL}$ concepts and roles are commutative, i.e. $A \sqcap B = B \sqcap A$, and thus order is unimportant. We therefore place no restrictions on the order of the symbols on the right hand side of production rules and thus consider production rules differing only in the order of symbols on the right hand side as identical.

The conversion process below proceeds in a step by step manner until all axioms in $\mathcal{T}$ have been processed.

Step 1: All axioms $\alpha_L \sqsubseteq \alpha_R \in \mathcal{T}$ such that $\text{Sig}(\alpha_L) = \emptyset$ are (implicitly) A-reachable. By definition of bottom-up reachability all such axioms are always A-reachable. For each such axiom the implicit A-reachability of α_L is made explicit by introducing the production rule:

$$y_i \rightarrow A$$

This rule is read as: $y_i \rightarrow A$ is bidirectionally reachable if the symbol y_i is $\overleftarrow{\text{B}}$-reachable and A is A-reachable.

Step 2: For each axiom $\alpha_L \sqsubseteq y_i \in \mathcal{T}$ introduce the production rule:

$$y_i \rightarrow Sig(\alpha_L)$$

Axioms of this kind do not have any implicit reachability concerns like those in Step 1 above. This rule is read as: $y_i \rightarrow Sig(\alpha_L)$ is bidirectionally reachable if the symbol y_i is $\overleftarrow{\text{B}}$-reachable and all symbols $x_i \in Sig(\alpha_R)$ are A-reachable.

Every step in the conversion process simply converts an axiom to an equivalent production rule. Since there is a one-to-one correspondence between $\mathcal{HL}$ axioms in normal form and production rules introduced by the above conversion process, it follows that a bidirectional reachability based module extracted on either the CFG or the TBox will result in the same set of axioms. Therefore bidirectional reachability is in no way affected by the conversion process and the correctness of the conversion process is implicit in the individual steps. We may thus formally define a reachability preserving CFG as:

Definition 39 ***Reachability preserving CFG for a $\mathcal{HL}$ TBox.***
Let $\mathcal{T}$ be an $\mathcal{HL}$ TBox in normal form and $A \sqsubseteq B$ a statement such that $\mathcal{T} \models A \sqsubseteq B$, then the reachability preserving CFG, denoted $\mathcal{CFG}^A_{\mathcal{T}}$, is a minimal set of CFG production rules such that for each axiom $\alpha_L \sqsubseteq \alpha_R \in \mathcal{T}$:

- *if $Sig(\alpha_L) = \emptyset$ the rule $x_i \rightarrow A \in \mathcal{CFG}^A_{\mathcal{T}}$ for each $x_i \in Sig(\alpha_R)$;*
- *for all other axioms the rule $x_i \rightarrow Sig(\alpha_L) \in \mathcal{CFG}^A_{\mathcal{T}}$;*

where the symbol A represents the only terminal symbol and the set $Sig(\mathcal{T})\backslash A$ represents the set of non-terminals.

Given a TBox $\mathcal{T}$ and a statement $A \sqsubseteq B$, we note that during the transformation process every axiom in the TBox $\mathcal{T}$ will be transformed into an equivalent production rule. Since the only terminal symbol is A, all production rules not containing A will consist of only non-terminal symbols. Therefore, during parsing, if some production rules are never both A and B reachable, they will not form part of any parse tree and thus not part of any MinA.

The conversion process may be illustrated by the following example:

Example 10 *Given the acyclic $\mathcal{HL}$ TBox $\mathcal{T}$ and the statement $A \sqsubseteq D$, with $\mathcal{T}'$ as $\mathcal{T}$ in normal form:*

TBox $\mathcal{T}$	*Normal form TBox* $\mathcal{T}'$
$A \sqsubseteq B_1 \sqcap B_2$	$\alpha_1 : A \sqsubseteq B_1$
$B_1 \sqsubseteq C_1 \sqcap D$	$\alpha_2 : A \sqsubseteq B_2$
$B_2 \sqcap C_1 \sqsubseteq D$	$\alpha_3 : B_1 \sqsubseteq C_1$
$\top \sqsubseteq B_2$	$\alpha_4 : B_1 \sqsubseteq D$
	$\alpha_5 : B_2 \sqcap C_1 \sqsubseteq D$
	$\alpha_6 : \top \sqsubseteq B_2$

Then $\mathcal{CFG}_{\mathcal{T}}^{A}$ *is given by:*

$B_1 \rightarrow A$ *(Step 2 applied to* α_1*)*

$B_2 \rightarrow A$ *(Step 2 applied to* α_2 *or Step 1 applied to* α_6*)*

$C_1 \rightarrow B_1$ *(Step 2 applied to* α_3*)*

$D \rightarrow B_1$ *(Step 2 applied to* α_4*)*

$D \rightarrow B_2 C_1$ *(Step 2 applied to* α_5*)*

4.2.4 Computing the Set of all Sub-modules with the Earley Algorithm

From the previous section we know that an $\mathcal{HL}$ TBox may be transformed to an equivalent CFG such that bidirectional reachability is preserved. In order to apply the Earley algorithm on the resulting CFG so that we may compute a representation of all sub-modules, we note that besides a CFG, the Earley algorithm requires the following additional information:

- a set of terminal symbols, and
- an input string to parse, and
- a start symbol.

Terminal symbols: Applying the Earley algorithm to the problem of computing MinAs is effectively a top-down depth first search on the TBox, corresponding to a top-down reachability searching procedure. When the search reaches the sub-concept in the entailment we initiate a bottom-up filtering process, corresponding to bottom-up reachability filtering. Thus the only symbol corresponding to a terminal symbol is the sub-concept of the entailment.

Input string: The input string serves two main purposes during the parsing process:

- Firstly, each symbol in the input string is a terminal symbol, thus search trees ending in the correct terminal symbol corresponds to a possible correct parse. This guides the search process and reduces the search space. When computing MinAs there is no explicit input string and the only terminal symbol is the sub-concept of the entailment. Thus the input string may be implicitly defined as a finite string consisting of a sufficient number of repetitions of the sub-concept.

- Secondly, the complexity of the algorithm is bounded by the length of the input string. In terms of MinA computation there exists no explicit input sentence and thus the complexity of the algorithm is harder to define.

Start symbol: The start symbol is chosen as any symbol name S such that $S \notin CN(\mathcal{T}) \cup RN(\mathcal{T})$.

Applying the standard Earley algorithm to the sub-module extraction problem may be illustrated by the following example. Given the $\mathcal{HL}$ subset of the Nci ontology, find the set of all MinAs for the entailment $\mathcal{T}_{Nci} \models \texttt{12p13-2} \sqsubseteq \texttt{Cell-Struct}$. In this example the set of terminal symbols consists of the symbol {`12p13-2`}, the input is a finite string consisting of a sufficient number of repetitions of this symbol and the start symbols is `S`.

	Axioms	Production Rules
α_1	`Subcell-Struct` $\sqsubseteq$ `Cell-Struct`	`Cell-Struct` $\rightarrow$ `Subcell-Struct`
α_2	`Nuclear-Struct` $\sqsubseteq$ `Subcell-Struct`	`Subcell-Struct` $\rightarrow$ `Nuclear-Struct`
α_3	`Organelle` $\sqsubseteq$ `Subcell-Struct`	`Subcell-Struct` $\rightarrow$ `Organelle`
α_4	`Chromosome` $\sqsubseteq$ `Nuclear-Struct`	`Nuclear-Struct` $\rightarrow$ `Chromosome`
α_5	`Chromosome-Struct` $\sqsubseteq$ `Nuclear-Struct`	`Nuclear-Struct` $\rightarrow$ `Chromosome-Struct`
α_6	`Nucleus` $\sqsubseteq$ `Organelle`	`Organelle` $\rightarrow$ `Nucleus`
α_7	`Human-Chromosome` $\sqsubseteq$ `Chromosome`	`Chromosome` $\rightarrow$ `Human-Chromosome`
α_8	`Chromosome-Arm` $\sqsubseteq$ `Chromosome-Struct`	`Chromosome-Struct` $\rightarrow$ `Chromosome-Arm`
α_9	`Chromosome-Band` $\sqsubseteq$ `Chromosome-Struct`	`Chromosome-Struct` $\rightarrow$ `Chromosome-Band`
α_{10}	`Chromosome-12` $\sqsubseteq$ `Human-Chromosome`	`Human-Chromosome` $\rightarrow$ `Chromosome-12`
α_{11}	`12p` $\sqsubseteq$ `Chromosome-Arm`	`Chromosome-Arm` $\rightarrow$ `12p`
α_{12}	`12p-13-2` $\sqsubseteq$ `Chromosome-Band`	`Chromosome-Band` $\rightarrow$ `12p-13-2`

Figure 4.1: Sample from Nci ontology

Example 11 *Compute the set of all sub-modules for the entailment* $(\mathcal{T}_{Nci})^{reach}_{12p13-2} \models \texttt{12p13-2} \sqsubseteq \texttt{Cell-Struct}$ *from the* $\mathcal{HL}$ *subset of the Nci ontology in Figure 4.1.*

Adding the initial state S → •Cell-Struct *to Chart 0, the charts produced by the Earley algorithm are as follows:*

	Chart 0			
1.	S → •Cell-Struct	j=0 :	Initial	
2.	Cell-Struct → •Subcell-Struct	j=0 :	Predictor:	*1*
3.	Subcell-Struct → •Nuclear-Struct	j=0 :	Predictor:	*2*
4.	Subcell-Struct → •Organelle	j=0 :	Predictor:	*2*
5.	Nuclear-Struct → •Chromosome	j=0 :	Predictor:	*3*
6.	Nuclear-Struct → •Chromosome-Struct	j=0 :	Predictor:	*3*
7.	Organelle → •Nucleus	j=0 :	Predictor:	*4*
8.	Chromosome → •Human-Chromosome	j=0 :	Predictor:	*5*
9.	Chromosome-Struct → •Chromosome-Arm	j=0 :	Predictor:	*6*
10.	Chromosome-Struct → •Chromosome-Band	j=0 :	Predictor:	*6*
11.	Human-Chromosome → •Chromosome-12	j=0 :	Predictor:	*8*
12.	Chromosome-Arm → •12p	j=0 :	Predictor:	*9*
13.	Chromosome-Band → •12p13-2	j=0 :	Predictor:	*10*
	Chart 1			
14.	Chromosome-Band → 12p13-2•	j=0 :	Scanner:	*13*
15.	Chromosome-Struct → Chromosome-Band•	j=0 :	Completer:	*14*
16.	Nuclear-Struct → Chromosome-Struct•	j=0 :	Completer:	*15*
17.	Subcell-Struct → Nuclear-Struct•	j=0 :	Completer:	*16*
18.	Cell-Struct → Subcell-Struct•	j=0 :	Completer:	*17*
19.	S → Cell-Struct•	j=0 :	Completer:	*18*

Starting with the final correct parse at Line 19, and working backwards we see that there is only a single parse tree to extract and it consists of entries 14, 15, 16, 17, 18 and 19. Mapping these back to the axioms in Figure 4.1, we obtain the set of axioms $\alpha_1, \alpha_2, \alpha_5, \alpha_7$ *and* α_{12}*. This set corresponds exactly to a single MinA for the entailment* $(\mathcal{T}_{Nci})^{reach}_{12p13-2} \models$ 12p13-2 $\sqsubseteq$ Cell-Struct*.*

The Earley algorithm extracts all parse trees given a CFG and an input string to parse. It terminates in $O(n^3)$ worst case time, where n is the length of the input string (Earley 1970). The running time of the Earley algorithm is bounded by the length of the input

string. However, in terms of sub-module computation there is no explicit input string and the number of charts created by the algorithm is dependent on the TBox in question. The following example shows that there exists an acyclic $\mathcal{HL}$ TBox such that the running time of the Earley algorithm is exponential in the number of production rules.

Example 12 (Exponential running time of the Earley algorithm.) *Define $\mathcal{T}_n$ to be the acyclic $\mathcal{HL}$ TBox consisting of the following GCI's:*

GCI's	Production Rules
$A \sqsubseteq C_1$	$C_1 \rightarrow A$
$C_1 \sqcap \ldots \sqcap C_{i-1} \sqsubseteq C_i$, *for* $1 < i \leq n$	$C_i \rightarrow C_{i-1} \ldots C_1$, *for* $1 < i \leq n$
$C_1 \sqcap \ldots \sqcap C_n \sqsubseteq B$	$B \rightarrow C_n \ldots C_1$

It follows that $\mathcal{T}_n \models A \sqsubseteq B$. The only production rule that terminates is $C_1 \rightarrow A$, every time it is present in a chart, the scanner procedure will be called, resulting in a new chart being created and the state entry $C_1 \rightarrow A\bullet$ added to it. Given the rule $C_2 \rightarrow \bullet C_1$, the predictor will add $C_1 \rightarrow \bullet A$ to the current chart, which in turn will cause the scanner to be called once again. Every rule $C_i \rightarrow C_{i-1} \ldots C_1$ encountered, where $2 < i \leq n$, has both C_1 and C_2 as part of its right hand side. Each of these rules will thus cause the scanner to be called at least twice. Rule $C_3 \rightarrow \ldots$ contains only C_1 and C_2 and as a result will call the scanner twice. Rule C_4 contains C_1 and C_2 as well as rule C_3, thus the scanner will be called a total of 2^2 times. For rule C_5 a total of 2^3 times, and rule C_i a total of 2^{i-2} times. Finally, rule $B \rightarrow C_n \ldots C_1$ will call the scanner a total of 2^{n-1} times, resulting in a total of 2^{n-1} new charts being created. In order to compute all sub-modules in this example, the Earley algorithm will run in exponential time.

4.2.5 Adapting and Improving the Algorithm

From the previous section we see that the Earley algorithm applied to sub-module computation results in an algorithm that runs in exponential worst-case time. The deterioration of the algorithm's performance is mainly due to the absence of an input string. In an attempt to improve the performance of the algorithm we investigate the effect of the input string more closely as well as how the difference between sub-module computation and sentence parsing may influence the performance of the algorithm. Consider the following production rules from Example 12 where $n = 4$:

$$\begin{aligned} B &\rightarrow C_4C_3C_2C_1 \\ C_4 &\rightarrow C_3C_2C_1 \\ C_3 &\rightarrow C_2C_1 \\ C_2 &\rightarrow C_1 \\ C_1 &\rightarrow C_A \end{aligned}$$

When parsing an input string, whenever the scanner procedure is called for a production rule, there is a terminal symbol on the right hand side of the dot that matches the word at the current position of the input string. In our example, after the completer has completed the rule $C_4 \rightarrow C_3C_2C_1$, the predictor will be called for the rule $B \rightarrow C_4 \bullet C_3C_2C_1$ in order to add all rules for the non-terminal C_3. Though all these production rules have been expanded previously, the repeated searching of sub-trees is neccesary, since at every position of the input string, the word at that position is potentially different from all words in other parts of the string, resulting in different sub-trees being completed.

However, in terms of sub-module computation, our input string is a finite string of the same symbol. Therefore re-searching the subtrees associated with a specific non-terminal symbol will result in exactly the same sub-trees being searched and completed repeatedly at every occurrence of this symbol. By excluding such repetitive searching of sub-trees we may potentially improve our sub-module computation algorithm. The first change to the algorithm is to the predictor procedure, which in turn cascades to changes in the completer procedure and the data structures used. The following changes to the algorithm were first proposed by Nortjé et al. (2009).

Changes to the Predictor: The predictor procedure normally expands all relevant production rules for a symbol and adds new states for these rules to the current chart. Duplicate states may never occur in the same chart, in a different chart however, the same state may be added again. We adapt the procedure so that every production rule may only be introduced once during the entire run of the algorithm. In this way, once the search of a sub-tree has been started, we never initiate a search down the same sub-tree again.

Changes to the Completer: There are two changes needed to the completer procedure in order to facilitate the above optimization to the predictor. Firstly, when the completer is called for the production rule $X \rightarrow ABC\bullet$, it searches for all other production rules, in the chart that initiated this rule, for all rules such that X immediately appears on the right hand side of the dot. For each such state found it advances the dot over this symbol and

adds the modified state to the current state for further processing. However, when a different rule $Y \rightarrow \bullet X \ldots$ needs to be processed by the predictor whenever the symbol X have been completed, the above changes to the predictor will cause it not to add any rules for X again. The algorithm will thus fail to advance the dot over X and processing for this rule will not continue.

Considering that we never search X rules again once it has been completed, we may modify the completer procedure so that once it completes the non-terminal X, it changes the symbol from a non-terminal to a terminal symbol. In this way, when the above rule $Y \rightarrow \bullet X \ldots$ is encountered the symbol X is no longer considered a non-terminal but a terminal symbol. In this case the scanner instead of the predictor will be called to further process the rule. We therefore require a dynamic terminal list, which initially has only one terminal symbol, and expand it with every symbol the completer completes.

Let the rules $Y \rightarrow \bullet X \ldots$ and $X \rightarrow \bullet \ldots$ appear in chart i, and the rule $Z \rightarrow \bullet X \ldots$ in chart $i+1$. Now let the rule $X \rightarrow \ldots \bullet$ be completed in chart k with $k > i+1$. Since the rule $X \rightarrow \ldots$ was initiated in chart i, the above modification to the completer will select the rule $Y \rightarrow \bullet X \ldots$ in chart i, and advance the dot over the symbol X and add X to the list of terminal symbols. However since we are processing chart k, chart $i+1$ will never be processed again and as a consequence the dot in rule $Z \rightarrow \bullet X \ldots$ will never be advanced over the X, resulting in and searching prematurely terminating for this rule. The completer must therefore not only search for previous rules, with the symbol X to the right of the dot, in the chart that originated the X rule, but rather it must search for any rule in any prior chart that meets this condition.

Modifying the completer to scan all previous charts instead of only the one that initiated the rule of interest could result in excessive backwards scanning. However, considering that the non-terminal being completed will in the future be considered a terminal symbol, whenever another rule completing the same non-terminal needs to be completed, the Completer does not have to scan all previous charts again. Let the rule $X \rightarrow \ldots \bullet$ be the first rule to be completed containing the symbol X on the left hand side. Then for all rules $Z_i \rightarrow \ldots \bullet X \ldots$ in any chart, the completer will add the rule $Z_i \rightarrow \ldots X \bullet \ldots$ to the current chart. Thus there is no non-completed rule in any chart prior to the current in which the bullet appear before the X. And all those that do contain an X to the right of the bullet in the current chart, X will be considered a terminal and thus handled by the scanner procedure.

Summarizing the changes to the Completer procedure we have that: for every rule $X \rightarrow$

$\ldots \bullet$ handled by the completer, if X is not a terminal symbol, then it is the first time any rule with X on the left hand side is completed. We therefore,

- mark X as a terminal symbol, and
- scan all previous charts for rules $Z_i \rightarrow \ldots \bullet X \ldots$, advance the dot over to symbol X, and add the new rule to the current chart.

Changes to the data structures: After processing, valid parses will leave the state $S \rightarrow \alpha\bullet$ in the chart. The current implementation of the Earley algorithm presented has no way of retrieving the tree structures associated with these states. Jurafsky & Martin (2008) address this issue by making a change to the completer procedure. Recall that the completer creates new states by advancing incomplete ones when the symbols following the dot is completed. The only change that is necessary is to have the completer add a pointer to the older state, onto a list of previous states, to the new state. Retrieving a parse tree from the chart is then merely a recursive retrieval starting with the state(s) representing a complete S. Let $Y \rightarrow \bullet X \ldots$ and $X \rightarrow \bullet \ldots$ be two states, and let the completer complete the state $X \rightarrow \ldots \bullet$ in position i of the chart. The state $Y \rightarrow X \bullet \ldots$ is then inserted into the current chart but augmented with the list $[i, \ldots]$.

A dynamic terminal set, combined with the above changes to the completer procedure, complicates the method for marking parse trees. The new Completer only completes the first occurrence of the rule $X \rightarrow \ldots \bullet$, whenever another such rule is encountered by it, it does not rescan all previous charts to advance the dot. The method for marking parse trees as proposed by Jurafsky & Martin (2008) therfore does not apply to the adapted alogorithm. This issue may be addressed by creating a two dimensional table structure in which the first column corresponds a non-terminal symbol Z_i, and the second to a list of pointers to states in which rules with Z_i on the left hand side are completed. Thus, whenever the completer encounters a rule $X \rightarrow \ldots \bullet$, it must not only perform all previous operations but also, in addition, it must add a pointer to the rule it completes. Figure 4.2.5 shows the initial table for Example 12.

Algorithm 5, initially proposed in Nortjé et al. (2009), is a preliminary algorithm which implements the above changes:

Algorithm 5 (Modified Earley parser) *The parser consists of three sub-parts, the* predictor, scanner *and* completer. *For each state in chart $\mathcal{C}_i$, the tuple $(X \rightarrow \alpha \bullet \beta,\ j)$, with α*

Non-terminal	List of pointers
$[B]$	$[\ldots]$
$[C_1]$	$[\ldots]$
$[C_2]$	$[\ldots]$
$[C_3]$	$[\ldots]$
$[C_4]$	$[\ldots]$

Figure 4.2: Completion Table

and β mixed string of terminals and non-terminals, is evaluated and the appropriate sub-part executed:

1. ***Predictor:*** *If state = $(X \rightarrow \alpha \bullet Y\beta)$, then for every production $Y \rightarrow \sigma$, if $(Y \rightarrow \bullet\sigma) \notin \mathcal{C}_i$ and no production rule for Y has ever been added to any chart, then $\mathcal{C}_i := \mathcal{C}_i + (Y \rightarrow \bullet\sigma)$,*

2. ***Scanner:*** *If state = $(X \rightarrow \alpha \bullet a\beta)$, with a the next symbol in the input string, and if $(X \rightarrow \alpha a \bullet \beta) \notin \mathcal{C}_{i+1}$ then $\mathcal{C}_{i+1} := \mathcal{C}_{i+1} + (X \rightarrow \alpha a \bullet \beta)$,*

3. ***Completer:*** *If state = $(X \rightarrow \gamma\bullet)$, then*

   - *add a pointer to this state in the completion table for X,and*
   - *mark X as a terminal symbol, and*
   - *if X is a new terminal symbol, then*
     - *for every $(Y \rightarrow \alpha \bullet X\beta) \in \mathcal{C}_k$ for $0 \leq k < i$, $\mathcal{C}_i := \mathcal{C}_i + (Y \rightarrow \alpha X \bullet \beta)$, and*
     - *for every $(Y \rightarrow \alpha \bullet X\beta) \in \mathcal{C}_i$ occurring prior to this state, $\mathcal{C}_i := \mathcal{C}_i + (Y \rightarrow \alpha X \bullet \beta)$.*

The algorithm executes all states iteratively in a top-down manner until no new states are available for processing, and no state may appear more than once in a given chart.

The exponential complexity of Algorithm 4 when applied to sub-module computation has been shown to result from the absence of an input string, which caused, in the worst case, an exponential number of new charts to be created. The following proof shows that Algorithm 5 creates in the worst case a polynomial number of new charts.

Theorem 19 *Let $\mathcal{T}$ be an $\mathcal{HL}$ TBox in normal form, and let $CFG_{\mathcal{T}}$ be the reachability preserving Context Free Grammar for $\mathcal{T}$. Further let $|CFG_{\mathcal{T}}| = n$ be the number of production rules in $CFG_{\mathcal{T}}$, and m the total number of distinct symbols in $CFG_{\mathcal{T}}$. Then Algorithm 5*

creates $n \times (n+1)$ new charts in the worst case, with each chart containing a maximum of $n \times (m+1)$ state entries.

Proof: For each non-completed production rule $X \rightarrow Y_1 \dots Y_m \in \text{CFG}_{\mathcal{T}}$ and each Y_i for $1 \leq i \leq m$, the algorithm may either call the predictor or the scanner. The scanner is only called for symbols that are terminals, for all other symbols the predictor is called. But initially there is at most one terminal symbol, and each of the n rules completed may add at most one new terminal symbol, hence there are at most $n+1$ terminal symbols. Thus for each non-completed rule, the scanner may be called at most $n+1$ times. The predictor only adds a rule once throughout the entire run of the algorithm, hence there are at most n uncompleted rules that may at any point still require further processing. The scanner may be called at most $n+1$ times for each of these, and hence a total of $n \times (n+1)$ times in the worst case.

Every production rule may have at most m symbols on its right hand side, with the dot having a maximum of $m+1$ positions in each rule. Given that each of these $m+1$ possible dot positions determines a unique state for each production rule, and that each rule may appear at most once in any chart C_i, we have that every chart C_i contains at most $n \times (m+1)$ state entries. From above we have that $1 \leq i \leq n \times (n+1)$ in the worst case and thus a maximum of $n \times (n+1) \times n \times (m+1) = (m+1) \times n^3 + (m+1) \times n^2$ state entries upon termination of the algorithm. □

The above adaptations produce a significant performance benefit in that it reduces the number of charts created by Algorithm 4 from an exponential number to a polynomial number of charts created by Algorithm 5. There are however further adaptations to be made to Algorithm 5 when the commutative nature of conjunction is taken into account. In terms of string parsing, the order in which the right hand side of a production rule's symbols must be processed is fixed. However, due to the commutative nature of conjunction, the order symbols are processed during sub-module computation is irrelevant. The commutative property of conjunction not only allows us to apply the predictor to the symbols on the right hand side of a rule in any order without affecting the outcome of the algorithm, but also apply it to all non-terminals in a rule simultaneously. The following adaptations effectively changes the algorithm from a parallel depth-first searching algorithm to a parallel breadth-first algorithm:

Predictor: For every production rule $X \rightarrow Y_1 \dots Y_n$ encountered, for each Y_i, add all productions rules $Y_i \rightarrow \dots, 1 \leq i \leq n$ not in the chart to the end of the current chart.

Completer: For every rule $X \rightarrow Y_1 \dots Y_n$ encountered with all Y_i being terminal symbols, mark X as a terminal symbol. If X is marked a terminal symbol for the first time, apply the completer recursively to all production rules in the chart that, by virtue of X becoming a terminal symbol, contain only terminal symbols on the right hand side of the rule.

The scanner: The changes to the Predictor and Completer procedures ensure that all symbols on the right hand side of a rule are processed simultaneously, therefore both the dot and the scanner procedure becomes obsolete and may be removed from the algorithm.

Algorithm 6 (Sub-module computation) *The parser consists of two sub-parts, the* predictor *and* completer*. For each state in CHART, the state $(X \rightarrow \alpha \bullet \beta)$ is evaluated and the appropriate sub-part executed:*
***Input**: Reachability preserving CFG - $\mathcal{CFG}_{\mathcal{T}}^{A}$ for an $\mathcal{HL}$ TBox $\mathcal{T}$ and the statement $A \sqsubseteq B$;*
***Output**: Reference table CHART and a Completion Table (Figure 4.2.5).*

1. *Add the initial state $S \rightarrow B$ to CHART.*
2. ***Predictor:** Given the state $(X \rightarrow Y_1 \dots Y_n)$, for all Y_i such that $(Y_i \rightarrow \sigma) \notin$ CHART, add all rules $(Y_i \rightarrow \sigma)$ to CHART.*
3. ***Completer:** If state $= (X \rightarrow Z_1 \dots Z_m)$ with all Z_i terminals, then*
   - *add a pointer to this state in the completion table for X,and*
   - *if X is not a terminal symbol, then mark it a terminal symbol, and*
   - *if X is a new terminal symbol, then call the completer for each rule $(Y \rightarrow \dots X \dots) \in$CHART such that all symbols on the right hand side of the rule are terminal symbols.*

The algorithm executes all states iteratively in a top-down manner until no new states are available for processing. The absence of the Scanner procedure guarantees that there will only be one chart CHART, with no state appearing more than once in CHART. All non completed rules in CHART is removed once the algorithm terminates. Thus CHART contains only those entries that are bidirectionally reachable.

Given $\mathcal{CFG}_{\mathcal{T}}^{A}$, the output of Algorithm 6 consist of a single chart, containing only completed production states (Definition 38), as well as a completion table (Figure 4.2.5). Since every production state in CHART has been completed, there exists a parse tree for the derivation $S \Rightarrow^* A$ that includes every production rule used in CHART. Every rule application preserves bidirectional reachability, hence the production rule for each state in CHART is bidirectionally reachable from the input parameters. Hence CHART contains only rules that for part of some bidirectional reachability-based sub-module, where each sub-module corresponds to a parse tree for a derivation $S \Rightarrow^* A$.

From Theorem 17 we know that every MinA, for an acyclic $\mathcal{HL}$ TBox consisting of only GCI's, corresponds to a minimal bidirectional reachability-based sub-module. We show that Algorithm 6 computes all possible bidirectional reachability-based sub-modules and thus all possible MinAs, and that it terminates in polynomial time.

Theorem 20 *Given an acyclic $\mathcal{HL}$ TBox $\mathcal{T}$ in normal form and the statement $A \sqsubseteq B$ such that $\mathcal{T} \models A \sqsubseteq B$, with $\mathcal{CFG}_{\mathcal{T}}^{A}$ the context free grammar associated with $\mathcal{T}$. If n is the number of production rules in $\mathcal{CFG}_{\mathcal{T}}^{A}$, then Algorithm 6 computes all possible bidirectional reachability-based sub-modules in $O(n^2)$ worst case running time.*

Proof: The proof proceeds in two parts: For the first part of the proof we have that there are at most n rules that may be added by the predictor, once they are added they are never removed. For every rule $X \rightarrow Y_1 \dots Y_n$ where $X \notin$ TERMINALS and all $Y_i \in$ TERMINALS, the completer will add X to TERMINALS. It then proceeds to check all rules $Y \rightarrow Z_i \dots X \dots Z_m$ already in CHART. For each of these, if all $Z_i \in$ TERMINALS and $Y \notin$ TERMINALS, the completer will call itself recursively for this rule. Every rule may be completed only once, hence the completer may be called a maximum of n times. During the n possible iterations of the Completer it may check at most $n-1$ axioms in CHART for new rules to complete. Assuming each check takes constant time, the Completer will perform at most $n \times (n-1)$ operations in the worst case. The algorithm thus has an $\mathrm{O}(n^2)$ worst case running time.

Secondly, we show by contradiction that all bidirectional reachability-based sub-modules are computed by the algorithm. Let T_1 be a bidirectional reachability-based sub-module in $\mathrm{CFG}_{\mathcal{T}}$ that is not computed by the algorithm. Then it must be the case that for at least one rule $\alpha = (X \rightarrow Y_1 \dots Z \dots Y_n) \in T_1 \subseteq \mathcal{CFG}_{\mathcal{T}}^{A}$ such that:

- $\alpha \notin$ CHART, or

- $\alpha \in$ CHART but there is no completion reference in the completion table for the symbol X to this rule.

First assume that $\alpha \notin$ CHART. Starting with the initial rule $S \rightarrow B$, for every rule $X \rightarrow \sigma_R$ encountered in CHART and every symbol $Y \in \sigma_R$ the predictor will add all rules $Y \rightarrow \ldots \notin$CHART to CHART. The addition of these rules are independent of any completions, thus for every rule encountered in CHART, all possible additional rules will be added by the Predictor for every symbol $Y \in \sigma_R$ except for rules for the symbol A, the initial terminal symbol. Since $\mathcal{T}$ is acyclic we have that there can be no sub-module where the symbol A uses itself, thus the predictor must never expand rules for this symbol. Hence, if $\alpha \notin$ CHART, the symbol X does not appear on the right hand side of any rule in CHART, and since all rules preserve top-down reachability, we have that the symbol X is not $\overleftarrow{\text{B}}$-reachable, and hence can not form part of any bidirectional reachability-based sub-module and thus that, T_1 is not a bidirectional reachability-based sub-module, which contradicts the assumption.

Now assume that $\alpha \in$ CHART. For every rule $X \rightarrow \sigma_R$ encountered by the Completer in CHART, such that all $Y \in \sigma_R$ are terminal symbols, a reference to this rule is made in the completion table for the symbol X, the symbol X is marked as a terminal symbol, and Completer is called recursively for all rules $Z \rightarrow Z_i \ldots X \ldots Z_m$ in CHART. Upon completion of the algorithm, we have that for every symbol X completed, there is a reference to every rule $X \rightarrow \sigma_R$ in which the symbol X has been completed. If there is no reference to α in the completion table for the symbol X, then at least one symbol $Y \in \sigma_R$ has not been completed. Since all rules preserve bottom-up reachability, we have that, if a symbol Y has never been completed then Y is not A-reachable, and hence the rule $Y \in \sigma_R$ cannot form part of any bidirectional reachability-based sub-module. In conclusion, T_1 is not a bidirectional reachability-based sub-module, which contradicts the assumption. □

4.2.6 Extracting MinAs

Once Algorithm 6 terminates, the chart returned contains a representation of all possible bidirectional reachability-based sub-modules, and hence a representation of all MinAs. This set is essentially an indexed bidirectionally reachable module. Though the algorithm terminates in polynomial time, we know that the chart may contain an exponential number of sub-modules, hence any algorithm extracting the set of minimal modules (MinAs) from the

chart must run in exponential worst-case time. When designing an extraction algorithm the properties of MinAs listed in Section 4.2.1 are of paramount importance. That is:

- All MinAs are bidirectional reachability-based sub-modules, thus for every symbol Y_i on the right hand side of a production rule, we need at least one other rule $Y_i \rightarrow \ldots$, with Y_i on the left hand side of the rule, in order to preserve bidirectional reachability.
- By Theorem 16 we have that, in every MinA, at most one rule $X_i \rightarrow \ldots$ for the symbol X_i on the left hand side may be present.

Based on these two properties, we note that, given a MinA M_1 for the statement $A \sqsubseteq B$ such that $M_1 \models A \sqsubseteq B$, we have that for every symbol $x_i \in \text{Sig}(M_1)$, other than the symbol A, there is exactly one production rule $(x_i \rightarrow \ldots) \in CFG_{M_1}$. We therefore base our extraction algorithm on finding a production rule for every symbol in the set of symbols represented by SYM, where a symbol $x_i \in SYM$ if and only if it appears on the right hand side of some production rule. The algorithm we present makes use of the following data structures:

- CHART - Reference table returned by Algorithm 6,
- SYM - The set of symbols $\{x_1, \ldots, x_i, \ldots x_n\}$ for which no production rule $x_i \rightarrow \ldots$ have yet been chosen,
- IMPL - The set of symbols $\{y_i, \ldots, y_j, \ldots y_m\}$ for which production rules $y_i \rightarrow \ldots$ have already been chosen,
- LIST - The set of all current production rules being processed,
- REF - A reference table, indexing every symbol z_i on the the right hand side of every production rule chosen. Every reference for symbol z_i in REF indicates the occurrence of z_i on the right hand side of some rule in the current set of rules in LIST.

Algorithm 7 extracts all MinAs for a given statement of interest given the reference table CHART returned by Algorithm 6. Given the statement of interest $A \sqsubseteq B$, it utilises a couple of procedures in order to extract all possible MinAs in $\mathcal{HL}$. Each of these procedures must must maintain the properties of MinAs in $\mathcal{HL}$ and thus the following set of invariants:

- INVARIANT-1: IMPL $\cap$ SYM $= \emptyset$.

Algorithm 7 (Extract-All-$\mathcal{HL}$-MinAs)

Procedure Extract-All-$\mathcal{HL}$-MinAs

```
27:  LIST:=REF:=SYM:=IMPL:=ResultSet:=∅
28:  SYM:=SYM + S
29:  while SYM≠ ∅
30:     Extract-Single-HL-MinA
31:     if LIST.Size > 0 then
32:       ResultSet := ResultSet + LIST
33:     Prepare-Next-MinA
34:  return ResultSet
```

- INVARIANT-2: for every rule $y_i \rightarrow \sigma_R \in$LIST, and for every $z_i \in \text{Sig}(\sigma_R)$ besides A, we have that $z_i \in$SYM or $z_i \rightarrow \ldots \in$LIST but not both.
- INVARIANT-3: for every rule $y_i \rightarrow \sigma_R \in$LIST, and for every $z_i \in \text{Sig}(\sigma_R)$ besides A, we have that REF$[z_i]$ contains a reference to $y_i \rightarrow \sigma_R$.

The first procedure adds a production rule to the set of production rules that make up the current MinA being extracted. It adds the given rule to the MinA and adds any new symbols on the right hand side of the rule that needs to be implemented to SYM. The procedure is formally defined as follows:

Procedure ADD_RULE($x_i \rightarrow \alpha_R$)

Input: $x_i \rightarrow \alpha_R$ - Production rule to add to list

```
1 :   LIST:=LIST + (x_i → α_R)
2 :   SYM:=SYM - x_i
3 :   IMPL:=IMPL + x_i
4 :   for each y_j ∈ α_R do
5 :     REF[y_j] := REF[y_j] + (x_i → α_R)
6 :     if y_j ∉IMPL then SYM := SYM + y_j
```

Lemma 4 *ADD_RULE($x_i \rightarrow \alpha_R$) preserves invariants INVARIANT-1, INVARIANT-2 and INVARIANT-3.*

Proof: Assuming that all three invariants hold when the procedure is entered, and that $x_i \in$ SYM, then upon termination of ADD_RULE($x_i \rightarrow \alpha_R$) all invariants will be preserved:

- INVARIANT-1: Only Lines 2, 3 and 6 alter the sets SYM and IMPL, as soon as x_i is removed from SYM at Line 2 it is added to IMPL at Line 3, thus IMPL $\cap$ SYM $= \emptyset$.

Line 6 adds y_j to SYM only when $y_j \notin IMPL$ and thus IMPL $\cap$ SYM $= \emptyset$. Therefore, ADD_RULE maintains this invariant.

- INVARIANT-2: Since the invariant holds when the algorithm is entered, and $x_i \in$SYM, we have that lines 1 and 2 together ensure that $x_i \notin$ SYM and that $x_i \rightarrow \ldots \in$ LIST. Now for every $y_j \in$ Sig(α_R), lines 4 and 6 ensures that $y_j \in$ IMPL or $y_j \in$ SYM but not both. However, by Line 3, $y_j \in$ IMPL whenever $y_i \rightarrow \ldots \in$LIST. Therefore, Line 6 adds y_j to SYM whenever $y_j \rightarrow \ldots \notin$LIST. Hence, INVARIANT-2 is preserved.

- INVARIANT-3: This invariant is trivially preserved by lines 1, 4 and 5. Line 1 adds the rule $(x_i \rightarrow \alpha_R)$ to LIST and while iterating through all $y_j \in$Sig(α_R) in Line 4, Line 5 adds a reference to $(x_i \rightarrow \alpha_R)$ in REF[y_j]. □

The second procedure we implement is designed to remove a production rule from the current MinA being extracted. It removes the production rule, then after verifying that a symbol on its right hand side is not required by any other rule it removes the symbol from SYM:

Procedure REMOVE_RULE($x_i \rightarrow \alpha_R$)
Input: $x_i \rightarrow \alpha_R$ - Production rule to remove from list
7 : LIST:=LIST - $(x_i \rightarrow \alpha_R)$
8 : IMPL:=IMPL - x_i
9: for each $y_j \in \alpha_R$ do
10: REF[y_j] := REF[y_j] - $(x_i \rightarrow \alpha_R)$
11: if REF[y_j] = $\emptyset$
12: SYM := SYM - y_j
13: if REF[x_i] $\neq \emptyset$
14: SYM := SYM + x_i

Lemma 5 *REMOVE_RULE($x_i \rightarrow \alpha_R$) preserves invariants INVARIANT-1, INVARIANT-2 and INVARIANT-3.*

Proof: Assuming that all three invariants hold when the procedure is entered, and that $x_i \rightarrow \alpha_R \in$LIST, then upon termination of REMOVE_RULE($x_i \rightarrow \alpha_R$) all invariants will be preserved:

- INVARIANT-1: By the invariants, when the procedure starts, we have that $x_i \notin$ SYM, $x_i \rightarrow \alpha_R \in$ LIST and $x_i \in$IMPL. Line 8 removes x_i from IMPL and lines 13 and 14 conditionally adds x_i to SYM. Whether or not x_i is added to SYM at Line 14, we have that IMPL $\cap$ SYM $= \emptyset$ and thus INVARIANT-1 holds true when the procedure terminates.

- INVARIANT-3: Lines 7, 9 and 10 trivially maintains this invariant. Line 7 removes $x_i \rightarrow \alpha_R$ from LIST and lines 9, 10 iterates over all $y_j \in$Sig(α_R) and removes the reference to $x_i \rightarrow \alpha_R$ from REF$[y_j]$.

- INVARIANT-2: By INVARIANT-3 we have that whenever REF$[y_j] \neq \emptyset$, the symbol y_i appears on the right hand side of some rule in LIST. For the symbol x_i, Line 7 removed $x_i \rightarrow \alpha_R$ from LIST and thus lines 13 and 14 ensures that if x_i appears on the right hand side of any remaining rule in LIST that $x_i \in$SYM. Thus for the symbol x_i the invariant is preserved. For all symbols $y_j \in$Sig(α_R) only appearing on the right hand side of the rule being removed and not in any other rule in LIST, lines 11 and 12 ensures that these symbols also does not appear in SYM. Hence, upon termination of the procedure, INVARIANT-2 is preserved. □

The next procedure we introduce extracts a single MinA, for the statement $A \sqsubseteq B$, by iterating over SYM and introducing a production rule for every entry in SYM except the symbol A and possibly the symbol S. Upon termination of Algorithm 6 the reference table CHART$[A] = \emptyset$ since A is the initial terminal symbol and thus there is no production rule $A \rightarrow \alpha_R$ referenced. For all other symbols C_i besides S, occurring on the left hand side any production rule in the bidirectional reachability-based sub-module retuned by Algorithm 6, CHART$[C_i] \neq \emptyset$. CHART$[S]$=$\emptyset$ whenever no bidirectional reachability-based sub-module exists for the statement $A \sqsubseteq B$ and thus no MinA.

Procedure Extract-Single-$\mathcal{HL}$-MinA	
15:	`for each` $x_i \in$ `SYM do`
16:	`if CHART[`x_i`] ≠ ∅`
17:	`ADD_RULE(CHART[`x_i`].First)`
18:	`else`
19:	`SYM := SYM -` x_i

Lemma 6 *Extract-Single-$\mathcal{HL}$-MinA preserves invariants INVARIANT-1, INVARIANT-2 and INVARIANT-3 and upon termination LIST is either empty or it contains all production rules representing a single MinA M_1 for the statement $A \sqsubseteq B$. Further, let m be the number of symbols in Sig(M_1), then Extract-Single-$\mathcal{HL}$-MinA terminates after at most $m \times m$ operations.*

Proof: Firstly, we assume that all three invariants hold when the procedure is entered, then we show that upon termination all invariants will be preserved. Secondly, we show that given that the invariants hold upon termination of the algorithm and that SYM = $\emptyset$, then the set LIST is either empty or contains all the production rules representing a single MinA.

The first part of the proof is trivial since no modifications are made to any of the data structures except by Procedure ADD_RULE, but by Lemma 4, ADD_RULE preserves the invariants for all $C_i \in$SYM except when CHART[S]=$\emptyset$. In this last case the invariants are trivially preserved since IMPL=LIST=$\emptyset$ and Line 19 will remove S from SYM.

Now let SYM= $\emptyset$, LIST $\neq \emptyset$ and let INVARIANT-1, INVARIANT-2 and INVARIANT-3 hold when Extract-Single-$\mathcal{HL}$-MinA terminates. By INVARIANT-2 we have that for every rule $y_i \rightarrow \sigma_R \in$LIST, and for every $z_i \in$ Sig(σ_R) besides A, we have that $z_i \in$SYM or $z_i \rightarrow \ldots \in$LIST but not both, yet SYM = $\emptyset$ thus for every rule $y_i \rightarrow \sigma_R \in$LIST, and for every $z_i \in$ Sig(σ_R) besides A, we have that $z_i \rightarrow \ldots \in$LIST. Hence, LIST represents a bidirectional reachability-based sub-module for the statement $A \sqsubseteq B$. To show that it is minimal, we note that by INVARIANT-1, IMPL $\cap$ SYM = $\emptyset$ at every iteration of the for loop at Line 17. Since ADD_RULE preserves INVARIANT-1 and for every rule $C_i \rightarrow \ldots$ processed adds the symbol C_i to IMPL and removes it from SYM, it is the case that once a rule $C_i \rightarrow \ldots$ have been processed by ADD_RULE the symbol C_i will never be added to SYM again during the entire run of the procedure. Hence, every symbol C_i occurs at most once on the left hand side of any production rule in LIST. Thus LIST is a minimal bidirectional reachability-based sub-module and by Theorem 17 a MinA.

Lastly we show that Extract-Single-$\mathcal{HL}$-MinA terminates in quadratic worst case time in the number of symbols in the MinA M_1. Let m be the number of symbols occurring in the MinA M_1, i.e. let $m = |\text{Sig}(M_1)|$. Then from the second part of the proof above, we know that each symbol C_i is added to SYM at most once throughout the entire run of Extract-Single-$\mathcal{HL}$-MinA, thus at most m symbols may ever be added to SYM. Since for each of these Procedure ADD_RULE is called exactly once, we have that the for loop at Line 17 will do

at most m iterations. However, since every rule may have at most m symbols appearing on its right hand side, we see that during the execution of ADD_RULE the for loop at Line 4 may cause at most m iterations of lines 5 and 6. Hence, assuming that all other operations runs in constant time, the worst case running time of Procedure Extract-Single-$\mathcal{HL}$-MinA is $O(m^2)$. □

The previous algorithm extracts a single MinA by assigning, in a depth-first manner, for each possible symbol $C_i \in$ SYM, a single production rule $C_i \rightarrow \ldots$ to LIST. In order to extract all possible MinAs, the same depth first approach may be followed by choosing, for each rule $C_i \rightarrow \alpha_R$ occurring in LIST, an alternate rule $C_i \rightarrow \sigma$ for the same symbol C_i. For each of these CHART$[C_i]$ represents all possible rules for the symbol C_i. However, Procedure ADD_RULE only ever adds the first entry in CHART$[C_i]$ to LIST. In order to prepare LIST and SYM so that all alternate MinAs may be extracted we introduce the procedure Prepare-Next-MinA.

Intuitively, starting with the last rule added to LIST for the symbol C_i, we remove the rule from LIST and choose, if possible, an alternate rule from CHART$[C_i]$ and add it to LIST, after which Extract-Single-$\mathcal{HL}$-MinA may be called to extract the next MinA. When no more alternates are available for the symbol C_i, the previous next last rule is chosen from LIST and alternates chosen. This procedure continues until an alternate has been found for some rule in LIST or no rule in LIST has an alternate, in which case there are no more additional MinAs to extract.

Procedure Prepare-Next-MinA	
20:	`while LIST.Size > 0 do`
21:	`  Index := LIST.Size - 1`
22:	`  `$x_i \rightarrow \alpha_R$ `:=LIST[Index]`
23:	`  REMOVE_RULE(`$x_i \rightarrow \alpha_R$`)`
24:	`  if CHART[`x_i`].HasNext AND (`$x_i \in$ `SYM or` $x_i = S$`)`
25:	`    ADD_RULE(CHART[`x_i`].Next)`
26:	`    return`

Lemma 7 *Procedure Prepare-Next-MinA preserves invariants INVARIANT-1, INVARIANT-2 and INVARIANT-3. Upon termination, either LIST* $= \emptyset$ *or exactly one rule* $C_i \rightarrow \alpha_R$ *for the symbol* C_i *has been replaced by an alternative rule* $C_i \rightarrow \sigma_R$.

Proof: Assuming that all three invariants hold when the procedure is entered, we show that upon termination of Prepare-Next-MinA all invariants are trivially preserved. The sets SYM, IMPL, LIST and REF are accessed only through the procedures ADD_RULE and REMOVE_RULE. After execution of REMOVE_RULE at Line 23, by Lemma 5 all invariants will be preserved. If an alternative rule exists for the symbol x_i, we have that by INVARIANT-2, $x_i \in$ SYM only if x_i appears on the right hand side of some other rule in LIST or when $x_i = S$. The check at Line 24 ensures that the alternate rule for x_i will only be added by ADD_RULE when $x_i \in$ SYM, thus, by Lemma 4, after execution of Line 25 all invariants will be preserved. Since all changes to the data structures are done at these two points, after every iteration of the while loop at Line 24, all invariants will be preserved.

Next we show that at termination of Prepare-Next-MinA, either LIST $= \emptyset$ or exactly one alternate rule has bee chosen. At every iteration of the while loop, a single rule $x_i \rightarrow \ldots$ is removed by REMOVE_RULE at Line 23. Since all invariants are preserved at this point, $x_i \in$ SYM only if x_i appears on the right hand side of some remaining rule in LIST. At Line 24, CHART$[x_i]$ is checked for an alternative rule for x_i. If an alternative rule exists then it is added to LIST by Line 25, after which Line 26 immediately exists the procedure. Since the inclusion of an alternate rule in LIST immediately leads to termination of the procedure, if an alternate rule is chosen, exactly one such rule is chosen during the execution of Prepare-Next-MinA. If no alternative rule has been found for x_i, no rule for x_i will be in LIST, the new last rule $y_i \rightarrow \ldots \in$ LIST will be chosen and an alternative rule chosen if possible. This process continues until a alternative rule has been found for some symbol C_i or LIST $= \emptyset$. □

We may now show that given CFG$_\mathcal{O}$, for the statement $A \sqsubseteq B$, Algorithm 7 returns a set containing all MinAs, and that each MinA is extracted in O(m^2) worst case time, where m is the number of symbols occurring in Sig(CFG$_\mathcal{O}$).

Theorem 21 *Given the indexed bidirectional reachability-based $\mathcal{HL}$ CFG for the statement $A \sqsubseteq B$, CFG$_\mathcal{O}$ and the reference table CHART returned by Algorithm 6, Algorithm 7 will extract all MinAs M_i such that $M_i \models A \sqsubseteq B$. Each M_i will be extracted in O(m^2) worst case running time, where $m = |Sig(CFG_\mathcal{O})|$.*

Proof: We give a proof in three parts. **Termination**: By Lemma 6, Extract-Single-$\mathcal{HL}$-MinA terminates in O(m^2) worst case time for the MinA M_1, where m is $|\text{Sig}(M_1)| \subseteq |\text{Sig}(\text{CFG}_\mathcal{O})|$. Procedure Prepare-Next-MinA does m operations in the worst case, which

occurs whenever no alternate rules for all m symbols exists and all m rules are removed from LIST. Hence each iteration of the while loop at Line 29, will run in at most O(m^2+m)=O(m^2) worst case time. During the execution of Prepare-Next-MinA at least one rule is removed from LIST and possibly changed with an alternative rule for the same symbol. Since $\mathcal{O}$ is acyclic, and CHART contains a finite number of alternate production rules for every symbol, Procedure Prepare-Next-MinA may be called at most a finite number of times before no more substitution is possible. At this point after the call to Prepare-Next-MinA, the set LIST = $\emptyset$, at which point the algorithm will terminate. By Example 7 there can be an exponential many MinAs and hence Algorithm 7 runs and terminates in exponential worst case time.

Soundness: By Lemma 6, if INVARIANT-1, INVARIANT-2 and INVARIANT-3 hold when Procedure Extract-Single-$\mathcal{HL}$-MinA is called, LIST contains a MinA upon its termination. We therefore show that during each iteration of the while loop at Line 29, all invariants are preserved and thus that after every execution of Line 30, LIST contains a valid MinA. Initially IMPL=LIST=REF=$\emptyset$ and SYM=$\{S\}$, hence, all invariants are trivially preserved during the first execution of Line 30. By Lemma 6, after execution of Extract-Single-$\mathcal{HL}$-MinA all invariants are preserved and by Lemma 7 every call to Prepare-Next-MinA preserves all invariants. Since these are the only two procedures called during the execution of every iteration of the while loop at Line 29, we have that every time Line 30 is executed all invariants are true, and thus by Lemma 6, every time Line 32 is executed, the set LIST contains a valid MinA.

Completeness: Alternate MinAs are introduced by Prepare-Next-MinA. By Theroem 7, after termination of this procedure LIST is either empty, which indicates that no more MinAs are available, or exactly one alternate rule has been introduced in LIST. Since this rule change preserves all invariants, we know that SYM contains all symbols y_i that appear on the right hand side of rules in LIST for which no rule $y_i \rightarrow \ldots$ appear in LIST. For these symbols, at the next call to Extract-Single-$\mathcal{HL}$-MinA, at Line 30, a MinA is extracted. Since Prepare-Next-MinA introduces every possible combination of rules for each symbol in LIST in turn, and after each such rule change all invariants are preserved, Algorithm 7 will extract all possible MinAs before termination. □

4.3 Computing and Extracting all MinAs in $\mathcal{EL}$

In this section we investigate the applicability of the sub-module computation algorithm (Algorithm 6) and the $\mathcal{HL}$ MinA extraction algorithm (Algorithm 7) as MinA computation algorithms for the more expressive DL $\mathcal{EL}$. In Section 4.3.1 we show how an $\mathcal{EL}$ TBox may be transformed to an equivalent CFG such that each production rule in the CFG preserves bidirectional reachability. Then in Section 4.3.2 we show that, for an $\mathcal{EL}$ TBox consisting of only primitive concept definitions, Algorithms 6 and 7 may be applied directly in order to extract all MinAs for a subsumption statement of interest. Lastly, we show there is an exponential explosion in the number of minimal modules associated with every MinA when general $\mathcal{EL}$ TBoxes are introduced.

4.3.1 Converting an $\mathcal{EL}$ ontology in normal form to a CFG

As in the case for $\mathcal{HL}$ TBoxes we show how a general $\mathcal{EL}$ TBox may be converted to an equivalent CFG. In the discussion that follows we assume that we have an $\mathcal{EL}$ TBox $\mathcal{T}$ in normal form, and a statement $A \sqsubseteq B$ such that $\mathcal{T} \models A \sqsubseteq B$, where A and B are single concept names.

Similar to $\mathcal{HL}$ we have that every MinA in $\mathcal{EL}$ is a bidirectional reachability-based sub-module, hence every production rule introduced during the conversion process must preserve bidirectional reachability. Given $\mathcal{T}^{reach}_{A \leftrightarrow B}$, every axiom $\alpha_L \sqsubseteq \alpha_R \in \mathcal{T}^{reach}_{A \leftrightarrow B}$ has the following two properties:

1. every $x_i \in \text{Sig}(\alpha_L)$ is A-reachable, and
2. some $y_i \in \text{Sig}(\alpha_R)$ is $\overleftarrow{\text{B}}$-reachable.

We know that there also exists a special case in which A-reachability implicitly holds for an axiom. For the DL $\mathcal{EL}$, implicit A-reachable axioms have the form:

$$\top \sqsubseteq \alpha_R$$

The conversion process below proceeds exactly as in the case of $\mathcal{HL}$. Every axiom is dealt with in a step by step manner until all axioms in $\mathcal{T}$ have been processed.

Step 1: All axioms $\alpha_L \sqsubseteq \alpha_R \in \mathcal{T}$ such that $\text{Sig}(\alpha_L) = \emptyset$ are (implicitly) A-reachable. By definition of bottom-up reachability all such axioms are always A-reachable. For

each such axiom the implicit A-reachability of α_L is made explicit by introducing the production rule:

$$y_i \rightarrow A$$

This rule is read as: $y_i \rightarrow A$ is bidirectionally reachable if the symbol y_i is $\overleftarrow{\text{B}}$-reachable and A is A-reachable.

Step 2: For each axiom $\alpha_L \sqsubseteq \alpha_R$ in $\mathcal{O}$ such that |Sig(α_L)|>= 1 and |Sig(α_R)|>= 1, introduce the production rule:

$$y_i \rightarrow Sig(\alpha_L)$$

for each $y_i \in Sig(\alpha_R)$. Axioms of this kind do not have any implicit reachability concerns like those in Step 1 above. This rule is read as: $y_i \rightarrow Sig(\alpha_L)$ is bidirectionally reachable if the symbol y_i is $\overleftarrow{\text{B}}$-reachable and all symbols $x_i \in Sig(\alpha_R)$ are A-reachable.

Every step in the conversion process conserves the bidirectional reachability properties of the original axioms. During the execution of Step 2, axioms of the form $\alpha_L \sqsubseteq \exists r.C$ are broken up into two separate production rules namely:

$$r \rightarrow \alpha_L$$

and

$$C \rightarrow \alpha_L$$

By Definition 31, top-down reachability is preserved whenever any of the symbols of an existential restriction that appears on the right hand side of an axiom are $\overleftarrow{\text{B}}$-reachable, hence the introduction of two separate productions rules preserves top-down reachability. Therefore, bidirectional reachability is in no way affected by the conversion process and the correctness of the conversion process is implicit in the individual steps. We may thus formally define a reachability preserving CFG as:

Definition 40 (Reachability preserving CFG for an $\mathcal{EL}$ TBox.) *Let $\mathcal{T}$ be an $\mathcal{EL}$ TBox in normal form, and $A \sqsubseteq B$ a statement such that $\mathcal{T} \models A \sqsubseteq B$, then the reachability preserving CFG, denoted $\mathcal{CFG}^A_{\mathcal{T}}$, is a minimal set of CFG production rules such that for each axiom $\alpha_L \sqsubseteq \alpha_R \in \mathcal{T}$:*

- *if $Sig(\alpha_L) = \emptyset$ the rule $x_i \rightarrow A \in \mathcal{CFG}^A_{\mathcal{T}}$ for each $x_i \in Sig(\alpha_R)$;*
- *otherwise, the rule $x_i \rightarrow Sig(\alpha_L) \in \mathcal{CFG}^A_{\mathcal{T}}$ for each $x_i \in Sig(\alpha_R)$;*

where the symbol A represents the only terminal symbol and the set Sig($\mathcal{T}$)\A represents the set of non-terminals.

The conversion process may be illustrated by the following example:

Example 13 *Given an acyclic $\mathcal{EL}$ TBox $\mathcal{T}$, with $\mathcal{T}'$ as $\mathcal{T}$ in normal form, and $A \sqsubseteq D$ the statement of interest:*

TBox $\mathcal{T}$	*Normal form TBox $\mathcal{T}'$*
$A \sqsubseteq B_1$	$\alpha_1 : A \sqsubseteq B_1$
$B_1 \sqsubseteq C_1 \sqcap D$	$\alpha_2 : B_1 \sqsubseteq C_1$
$B_2 \sqcap C_1 \sqsubseteq D$	$\alpha_3 : B_1 \sqsubseteq D$
$\top \sqsubseteq B_2$	$\alpha_4 : B_2 \sqcap C_1 \sqsubseteq D$
$C_1 \sqsubseteq \exists r.D$	$\alpha_5 : \top \sqsubseteq B_2$
$\exists r.B_1 \sqsubseteq B_2$	$\alpha_6 : C_1 \sqsubseteq \exists r.D$
	$\alpha_7 : \exists r.B_1 \sqsubseteq B_2$

Then $\mathcal{CFG}_{\mathcal{T}}^{A}$ is given by:

$B_1 \rightarrow A$ *(Step 2 applied to α_1)*
$B_2 \rightarrow A$ *(Step 1 applied to α_5)*
$C_1 \rightarrow B_1$ *(Step 2 applied to α_2)*
$D \rightarrow B_1$ *(Step 2 applied to α_3)*
$D \rightarrow B_2 C_1$ *(Step 2 applied to α_4)*
$r \rightarrow C_1$ *(Step 2 applied to α_6)*
$D \rightarrow C_1$ *(Step 2 applied to α_6)*
$B_2 \rightarrow r B_1$ *(Step 2 applied to α_7)*

4.3.2 Extracting all $\mathcal{EL}$ MinAs

In Section 4.2 we saw that, given a CFG representation of an $\mathcal{HL}$ TBox, Algorithm 6 computes an indexed bidirectional reachability-based sub-module from which all sub-modules may be extracted. Algorithm 7 may then be used to extract all minimal modules and thus MinAs. Since the algorithm's correctness is dependent on the implicit correctness of the CFG production rules and not the specific TBox the CFG represents, the algorithm may be applied to any DL for which bidirectional reachability-based sub-modules may be extracted. We therefore state the following without proof:

Theorem 22 *Let $\mathcal{T}$ be an acyclic $\mathcal{EL}$ TBox in normal form, and $A \sqsubseteq B$ a statement such that $\mathcal{T} \models A \sqsubseteq B$, with $CFG_{\mathcal{T}}$ the context free grammar associated with $\mathcal{T}$. If n is the number of production rules in $CFG_{\mathcal{T}}$, then Algorithm 6 computes all possible bidirectional reachability-based sub-modules in $O(n^2)$ worst case running time.*

Once Algorithm 6 terminates we need to extract all possible sub-modules representing MinAs. By Corollary 1 there is a one-to-one correspondence between $\mathcal{HL}$ MinAs and minimal bidirectional reachability-based sub-modules. However, when existential restrictions are introduce, it is no longer the case that every minimal module corresponds to a MinA.

Example 14 *Let $\mathcal{T}$ be an acyclic $\mathcal{EL}$ TBox consisting of only primitive concept definitions. Further let M_1 be a minimal bidirectional reachability-based sub-module for the statement $A \sqsubseteq B$ consisting of the axioms $A \sqsubseteq \exists r.C$ and $C \sqsubseteq B$. Then $M_1^{reach}_{A \leftrightarrow B} = M_1$ and M_1 is minimal, but $M_1 \not\models A \sqsubseteq B$ unless $M_1 \models B \sqsubseteq \bot$. Hence M_1 is not a MinA for $\mathcal{T} \models A \sqsubseteq B$.*

In fact, let $\mathcal{T}$ be an acyclic $\mathcal{EL}$ TBox consisting of only primitive concept definitions, and let M_1 be a minimal bidirectional reachability-based sub-module for the statement $A \sqsubseteq B$ such that $(\alpha_L \sqsubseteq \exists r.C)$ is the only existential restriction occurring in M_1. Since $\mathcal{T}$ consists of only primitive concept definitions, the symbol r does not appear on the left hand side of any axiom in $\mathcal{T}$ and bidirectional reachability is preserved solely through the symbol C. However, since $\mathcal{EL}$ does not contain the $\bot$ concept or $\neg$ constructor and only allows for the conjunction of concept descriptions, all concepts are satisfiable including C and thus $M_1 \not\models C \sqsubseteq \bot$. Thus any axiom with $\exists$ in its right hand side cannot contribute to the entailment $\mathcal{T} \models A \sqsubseteq B$, and hence any minimal bidirectional reachability-based sub-module containing such a statement cannot be a MinA. Each minimal bidirectional reachability-based sub-module M_i not containing existential restrictions on the right hand side of any axiom in M_i corresponds to a minimal sub-module in the DL $\mathcal{HL}$.

Corollary 2 *Let $\mathcal{T}$ be a $\mathcal{EL}$ TBox consisting of only primitive concept definitions in normal form, and let $A \sqsubseteq B$ be a statement such that $\mathcal{T} \models A \sqsubseteq B$. Then for every minimal bidirectional reachability-based sub-module N_i such that $\alpha_L \sqsubseteq \exists r.C \in N_i$ we have that $N_i \not\models A \sqsubseteq B$. Further, for every minimal bidirectional reachability-based sub-module M_i such that $\alpha_L \sqsubseteq \exists r.C \notin M_i$ we have that $M_i \models A \sqsubseteq B$.*

Consequently, Algorithm 7 may be used in order to extract all minimal modules and thus MinAs for acyclic $\mathcal{EL}$ TBoxes consisting of only primitive concept definitions. When a

minimal module includes axioms containing existential restrictions this module may simply be discarded as not being a MinA. Thus Algorithm 7 is complete in that it will extract all MinAs, however since not all minimal modules extracted are MinAs it is no longer sound.

Extending the MinA extraction problem to general $\mathcal{EL}$ TBoxes we show, by the following example, that for these TBoxes there are MinAs that are not minimal bidirectional reachability-based sub-modules.

Example 15 *Let $\mathcal{T}$ be an acyclic general $\mathcal{EL}$ TBox in normal form, further let M be a bidirectional module for the statement $A \sqsubseteq D$ such that $M \models A \sqsubseteq D$, where M consists of the following axioms:*

$$A \sqsubseteq C_1, \quad A \sqsubseteq B, \quad B \sqsubseteq C_3, \quad C_1 \sqsubseteq \exists r.C_2, \quad C_2 \sqsubseteq C_3, \quad C_3 \sqcap C_4 \sqsubseteq D, \quad \exists r.C_3 \sqsubseteq C_4$$

Then there are two minimal bidirectional reachability-based sub-modules $M_1{}^{reach}_{A \leftrightarrow B}$ and $M_2{}^{reach}_{A \leftrightarrow B} \subset \mathcal{T}^{reach}_{A \leftrightarrow B}$ such that $M_1{}^{reach}_{A \leftrightarrow B} \not\models A \sqsubseteq D$ and $M_2{}^{reach}_{A \leftrightarrow B} \not\models A \sqsubseteq D$.

$$M_1 = \{A \sqsubseteq C_1, \quad A \sqsubseteq B, \quad B \sqsubseteq C_3, \quad C_1 \sqsubseteq \exists r.C_2, \quad C_3 \sqcap C_4 \sqsubseteq D, \quad \exists r.C_3 \sqsubseteq C_4\}$$

and

$$M_2 = \{A \sqsubseteq C_1, \quad C_1 \sqsubseteq \exists r.C_2, \quad C_2 \sqsubseteq C_3, \quad C_3 \sqcap C_4 \sqsubseteq D, \quad \exists r.C_3 \sqsubseteq C_4\}$$

Analysing this example we can conclude that:

1. M is the only MinA, but since it is not a minimal bidirectional reachability-based sub-module, not all MinAs are minimal bidirectional reachability-based sub-modules, and

2. M_1 and M_2 are minimal bidirectional reachability-based sub-modules, but neither M_1 not M_2 are MinAs, minimal bidirectional reachability-based sub-modules are not necessarily MinAs.

Here the loss of minimality is caused by concept C_3. In both M_1 and M_2 we have that C_3 occurs on the left hand side of axioms on two occasions. For each of these bidirectional reachability requires the occurrence of C_3 on the right hand side of only one axiom. In terms of production rules, we see that there are two rules for the symbol $(C_3 \rightarrow \ldots)$. In order to preserve minimality Algorithm 7 restricts the choice to only one of these, however in this case, whenever the symbol C_3 occurs on the right hand side of a production rule, a different choice of production rule implementing the symbol C_3 needs to be made. We note here that the

above samples are for general TBoxes and thus not MinAs do not neccesarily correspond to minimal bidirectional sub-modules, however, for the acyclic TBoxes containing only primitive concept inclusion every MinA still corresponds to a minimal bidirectional reachability-based sub-module.

Generalizing this result we have that for every occurrence of the symbol C_i on the right hand side of a production rule, a MinA extraction algorithm would have to iterate over every possible implementation of the symbol C_i. Each possible choice does not necessarily lead to subsumption being preserved, and where subsumption is preserved, the resulting set may not necessarily be minimal. Hence not only is it required to extract all possible bidirectional reachability-based sub-modules and test for subsumption preservation, it is also required to run a minimality test on each module returned. This leads to a possible exponential explosion in the number of modules being extracted and tested.

Theorem 23 *Let $\mathcal{T}$ be an acyclic general $\mathcal{EL}$ TBox in normal form and $A \sqsubseteq B$ a statement of interest. Let CHART represent the resultant reference set returned by Algorithm 6. Let M_1 be a MinA such that $M_1 \models A \sqsubseteq B$ and P_1 represent the set of production rules for M_1. Now let m_i be the number of times a symbol C_i occurs on the right hand side of all production rules in P_1 and let k_i be the number of entries in CHART[C_i]. Then for the n possible symbols in P_1 there are a total of $\prod_{i=1}^{n} \sum_{j=1}^{j=m_i \leq k_i} C(\binom{k_i}{j})$ bidirectional reachability-based sub-modules.*

Proof: Let S be a set of n elements and let $1 \leq r \leq n$, then the number of subsets of S with r elements is given by $C(\binom{n}{r})$. Now for each symbol C_i the number of possible choices is given by k_i, one choice for each entry in CHART[C_i]. Thus, if the symbol C_i appears only once on the right hand side of all production rules in P_1, there are a total of $C(\binom{k_i}{1}) = k_i$ possible variations of sub-modules introduced by this symbol. Now, when the symbols C_i appears twice on the right hand side of all production rules in P_1, we may assign the same rule from CHART[C_i] to both occurrences of the symbols which allows for $C(\binom{k_i}{1}) = k_i$ possible choices, or we may assign all possible 2-combinations of alternating rules from CHART[C_i] i.e. $C(\binom{k_i}{2})$. Combining these results we have a total of $C(\binom{k_i}{1}) + C(\binom{k_i}{2})$ possible choices of rule assignments. For the m_i occurrences of the symbol C_i on the right hand side of all production rules in P_1, we have a total of $\sum_{j=1}^{j=m_i \leq k_i} C(\binom{k_i}{j})$ possible choices of elements from CHART[C_i], where j iterates over all values between 1 and m_i or 1 and k_i whenever $m_i > k_i$. The maximum number of choices for any symbol C_i is occurs when $m_i = k_i$ and amounts to a total of $2^{k_i} - 1$ variations. However, whenever $m_i < k_i$, we have that $\sum_{j=1}^{j=m_i < k_i} C(\binom{k_i}{j}) \geq 2^{m_i} - 1$. Thus if

$|C_i|$ denotes the number of possible sub-module variations introduced by the symbol C_i, then $|C_i|$ lies in within the boundary given by $2^{m_i} \leq |C_i| + 1 \leq 2^{k_i}$. Now let $|C_k|$ denote the total number of variants of sub-modules introduced by the symbol C_k, then for both C_i and C_k together there are a total of $|C_i| \times |C_k|$ possible sub-modules, and thus for all n symbols a total of $\prod_{i=1}^{n} |C_i|$ distinct variants. □

Calculating the total number of bidirectional reachability-based sub-modules in Example 15 we have that all symbols C_i besides C_3 occur only once on the right hand side of production rules and have only one entry in CHART. For all these symbols we have that $C(\binom{1}{1}) = 1$. The symbol C_3 appears twice on the right hand side of production rules, and twice in CHART$[C_3]$, hence we have a total of $2^2 - 1 = 3$ bidirectional reachability-based sub-modules, namely M, M_1 and M_2.

From Theorem 23 we see that any algorithm iterating over all possible bidirectional reachability-based sub-modules in order to extract MinAs for general $\mathcal{EL}$ TBoxes runs in exponential worst case time for every single MinA being extracted. Every bidirectional reachability-based sub-module M_i extracted also has to be tested for subsumption i.e. $M_i \models A \sqsubseteq B$ and for minimality. Since other MinA extraction methods, such as those presented by Suntisrivaraporn et al. (2008), require in the worst case, only a logarithmic number of subsumption tests in order to extract a single MinA, iterating over an exponential number of bidirectional reachability-based sub-modules in order to find a single MinA is potentially inefficient.

4.4 Conclusion

In this chapter we investigated the relationship between bidirectional reachability-based modules and MinAs. We showed that there is a one-to-one correspondence between MinAs and minimal bidirectional reachability-based modules for a subsumption entailment in the DL $\mathcal{HL}$. Extending our investigation to $\mathcal{EL}$ we saw that for $\mathcal{EL}$ TBoxes consisting of only primitive concept definitions, axioms in normal form containing existential restrictions on the right hand side cannot contribute to a MinA and thus that, for these TBoxes, the algorithms for $\mathcal{HL}$ can be used to extract all MinAs. For general $\mathcal{EL}$ TBoxes we know that every MinA is a bidirectional reachability-based sub-module but we have shown that these need not neccesarily be minimal and thus in order to extract a single MinA, any algorithm iterating over

bidirectional reachability-based sub-modules would need, in the worst case, to iterate over all such modules in order to extract even a single MinA. Thus any MinA extraction algorithm that iterates over bidirectional reachability-based sub-modules would be extremely inefficient.

Chapter 5

Empirical Results

In this chapter we test the algorithms presented in Chapters 3 and 4 and evaluate their performance in terms of real world biomedical ontologies. The main contribution in this dissertation is the introduction of a bidirectional reachability-based modularization method, which may be used to obtain small modules that consider both the sub-concept and super-concept in a subsumption entailment. Testing is therefore limited to that of extracting modules based on subsumption statements between single concepts. Module extraction based on general signatures are not evaluated in this dissertation.

5.1 Test Suite

Three real world, generally large scale, biomedical ontologies are considered for testing purposes in this chapter, which for comparison purposes, are identical to those used by Suntisrivaraporn (2009) during the testing of the CEL reasoner[1]. Since the focus of the algorithms that were introduced in the previous chapters is on inexpressive DLs in the $\mathcal{EL}$ family we have restricted our choice of ontologies to the following:

$\mathcal{O}_{Snomed}$ - The Systematized Nomenclature of Medicine, Clinical Terms (Snomed). A comprehensive medical and clinical ontology. It is an unfoldable $\mathcal{ELH}$ TBox augmented with two complex role inclusion axioms. We utilise the January/2005 release which consists of 340 972 primitive concept definitions, 38 719 full concept definitions, 11 role hierarchy axioms, and a role inclusion axiom, and with a total of 379 691 concept names and 62 role names.

[1]These were obtained and used without alteration from http://lat.inf.tu-dresden.de/systems/cel/

$\mathcal{O}_{Nci}$ **- The Thesaurus of the US National Cancer Institute (Nci).** This a large cancer-centric ontology that is a reference terminology for covering domains such as diseases, drugs, anatomy, genes, gene products, techniques and biological processes. The ontology is designed as an unfoldable $\mathcal{EL}$ TBox augmented with domain and range restrictions. It consists of 27 652 primitive concept definitions which refers to 70 role names. In the CEL version of the ontology, as obtained from http://lat.inf.tu-dresden.de/systems/cel/, all explicit domain and range restrictions have been removed.

$\mathcal{O}_{Go}$ **- The Gene Ontology (Go).** This ontology is a collaborative effort to provide consistent descriptions of gene products over different databases. It is formulated as an unfoldable $\mathcal{EL}$ TBox with a single transitive role part-of. The release of $\mathcal{O}_{Go}$ used in Suntisrivaraporn (2009) and in our evaluations, is an acyclic primitive TBox and consists of 20 465 primitive concept definitions, each defining a single concept name.

The algorithms presented were all implemented in Java as part of a plugin for the Protégé 4.1 ontology editor. All tests were performed on a single Intel Quad Core based computer, with 6 Gig of RAM, running on Microsoft Windows 7 x64 and hosted in a 64 bit Java virtual machine. No consideration was given during the implementation of the algorithms for utilising the multi-core capabilities of the processor. All algorithms were thus implemented using a single threaded approach.

In all cases the standard classes from the OWLApi version 3.0.0 were used without alteration, no additional processing was performed to alter an ontology once loaded. Thus any internal processing, bookkeeping, normalisation and indexing as performed by the standard ontology classes were left unaltered. All axioms were explicitly left in their standard denormalised form as loaded by the ontology. Where normalisation was required by the algorithms, it was done implicitly during a separate indexing phase on both the left and right hand side of axioms, where elements of a signature were indexed based on their projected normal forms. In this way axioms could be left unaltered whilst all processing could be performed using the relevant indexes.

We did not implement nor utilise an optimized subsumption testing algorithm for inexpressive DLs. Where subsumption testing was required by the MinA extraction algorithm, the procedure adopted were as follows. Given a set of axioms S representing a possible MinA, create a new ontology consisting of these axioms. Once the ontology has been created we

call the standard HerMit[2] reasoner to perform the neccesary subsumption testing. Though this method is probably not as efficient as providing a custom subsumption testing algorithm specifically optimized for the task at hand, our goal was to test the effectiveness of our MinA extraction algorithm by using the existing standard reasoning algorithms provided by the Protégé environment.

5.2 Module Extraction

In this section we investigate the performance of Algorithm 3 in terms of extracting bidirectional reachability-based sub-modules for subsumption statements between single concept names.

Given that our aim is to extract bidirectional reachability-based sub-modules for all pairs of concept names, we observe that, given an ontology $\mathcal{O}$ and a concept name A, extracting a bidirectional reachability-based sub-module $\mathcal{O}^{reach}_{A\leftrightarrow B}$ for any concept name B not in $\text{Sig}(\mathcal{O}^{reach}_{A})$, will result in an empty module being extracted. We therefore followed the following approach: for an ontology $\mathcal{O}$, and each symbol $A \in \text{Sig}(\mathcal{O})$, we extracted the reachability-based module $\mathcal{O}^{reach}_{A}$ and limited bidirectional reachability-based sub-module extraction to only those concept names in $\text{Sig}(\mathcal{O}^{reach}_{A})$.

Thus for each concept name $A \in \text{Sig}(\mathcal{O})$ and all concept names B in $\text{Sig}(\mathcal{O}^{reach}_{A})$ such that $\mathcal{O} \models A \sqsubseteq B$ we calculated the average additional time to extract, as well as the size of, the bidirectional reachability-based module for $A \sqsubseteq B$. For each symbol $A \in \text{Sig}(\mathcal{O})$ we averaged the results over all bidirectional reachability-based modules $\mathcal{O}^{reach}_{A\leftrightarrow B}$ for each symbol $B \in \text{Sig}(\mathcal{O}^{reach}_{A})$. Median and maximum values are thus averaged values for these bidirectional reachability-based modules and therefore not integer values.

Table 5.1 shows the results[3] of all bidirectional reachability-based sub-modules extracted using Algorithm 3. The columns in the table are organised as follows:

- Ontology - The ontology for which the modules are being extracted.

[2] http://hermit-reasoner.com/

[3] We note that the number of axioms in the reachability-based modules reported here differs substantially from those reported in Suntisrivaraporn (2009). The reason for this discrepancy is not due to differences in the core reachability module extraction algorithm, but rather the fact that we do not normalize axioms within the ontology. Thus we have fewer axioms in each extracted module and no penalties with regards to de-normalisation. We further note that, we did not calculate the average values for all bottom-up reachability-based modules, but rather only for those for which bidirectional reachability-based modules for a subsumption entailment could be extracted.

- $| \mathcal{O}_A^{reach} |$ - The number of axioms in the reachability-based modules for all concepts $A \in \text{Sig}(\mathcal{O})$.

- $\text{T}(\mathcal{O}_A^{reach})$ - The average time, in seconds, required by the algorithm to extract all reachability-based modules.

- $| \mathcal{O}_{A \leftrightarrow B}^{reach} |$ - The average number of axioms for all bidirectional reachability-based sub-modules.

- $\text{T}(\mathcal{O}_{A \leftrightarrow B}^{reach})$ - The additional time, in seconds, required to extract the bidirectional reachability-based sub-modules, i.e. Total time = $\text{T}(\mathcal{O}_A^{reach}) + \text{T}(\mathcal{O}_{A \leftrightarrow B}^{reach})$.

Average Values

Ontology	$\lvert \mathcal{O}_A^{reach} \rvert$	$\text{T}(\mathcal{O}_A^{reach})$	$\lvert \mathcal{O}_{A \leftrightarrow B}^{reach} \rvert$	$\text{T}(\mathcal{O}_{A \leftrightarrow B}^{reach})$
$\mathcal{O}_{Go}$	13.16	0.000032	4.48	0.000006
$\mathcal{O}_{Nci}$	25.68	0.000048	5.59	0.000006
$\mathcal{O}_{Snomed}$	27.70	0.040725	18.40	0.000175

Maximum Values

Ontology	$\lvert \mathcal{O}_A^{reach} \rvert$	$\text{T}(\mathcal{O}_A^{reach})$	$\lvert \mathcal{O}_{A \leftrightarrow B}^{reach} \rvert$	$\text{T}(\mathcal{O}_{A \leftrightarrow B}^{reach})$
$\mathcal{O}_{Go}$	68	0.000417	20.15	0.000666
$\mathcal{O}_{Nci}$	398	0.001916	55.00	0.000569
$\mathcal{O}_{Snomed}$	254	0.217781	222.06	0.004843

Median Values

Ontology	$\lvert \mathcal{O}_A^{reach} \rvert$	$\text{T}(\mathcal{O}_A^{reach})$	$\lvert \mathcal{O}_{A \leftrightarrow B}^{reach} \rvert$	$\text{T}(\mathcal{O}_{A \leftrightarrow B}^{reach})$
$\mathcal{O}_{Go}$	10	0.000026	3.86	0.000005
$\mathcal{O}_{Nci}$	11	0.000026	4.37	0.000005
$\mathcal{O}_{Snomed}$	16	0.001800	6.66	0.000008

Table 5.1: Bidirectional reachability-based module extraction using Algorithm 3

Table 5.2 shows a summary of the average values from Table 5.1. Here we see that bidirectional reachability-based modules are between 30% and 80% smaller than standard reachability-based modules and may be extracted at the additional cost of between 0.4% and 19.0% in the running time of the algorithm. Though the initial reachability-based modules are small in themselves, the reductions can still be considered considerable when the negligible extra cost for these reductions are taken into consideration.

We note here that the average runtime increases for the GO and NCI ontologies tested seems excessively high. However, it is important to note that, the running times are measured in the low microsecond range. At these extremely small intervals the accuracy of our measuring tools is very low and the true runtime performance of the algorithms only becomes evident in relatively large ontologies. Therefore, the runtime performance of the algorithms

for the SNOMED ontology gives a more accurate measure of the true performance of the algorithms.

Ontology	Decrease in size	% Increase in Time
$\mathcal{O}_{Go}$	66.00%	18.00%
$\mathcal{O}_{Nci}$	78.00%	12.00%
$\mathcal{O}_{Snomed}$	33.00%	0.40%

Table 5.2: Average value summary

The summary of median values as shown in Table 5.2 indicates that extracting bidirectional reachability-based modules results in very stable performance across all ontologies tested, with an approximate 59% decrease in the size of all modules extracted.

Ontology	Decrease in size	% Increase in Time
$\mathcal{O}_{Go}$	61%	19.20%
$\mathcal{O}_{Nci}$	60%	18.50%
$\mathcal{O}_{Snomed}$	58%	00.46%

Table 5.3: Median value summary

The relatively bad average performance of the algorithm in the case of $\mathcal{O}_{Snomed}$ can be attributed to the extensive use of the *roleGroup* role name, which is used in about 28% of all axioms throughout the ontology and serves the purpose of grouping together multiple existential restrictions in a definition (Suntisrivaraporn 2009). In terms of top-down reachability, whenever this role is present in one axiom in the module, all axioms containing the symbol on its right hand side will be included in the module. This behaviour of the top-down reachability heuristic seems to reduce its effectiveness in cases, such as in $\mathcal{O}_{Snomed}$, where specific role name symbols are used extensively throughout an ontology.

Ontology	Decrease in size	Increase in Time	
		Algorithm 3	Algorithm 6
$\mathcal{O}_{Go}$	66%	0.000006	0.000018
$\mathcal{O}_{Nci}$	78%	0.000006	0.000020

Table 5.4: Comparative summary

The sub-module computation algorithm (Algorithm 6) extracts an indexed bidirectional reachability-based module that requires an additional iteration of the extracted module as well as runtime overhead for bookkeeping purposes. As a result, upon termination of Algorithm 6, all MinAs may be extracted from the module, whereas Algorithm 3 extracts a bidirectional reachability-based modules without any additional preparation for MinA extraction. From

the comparison of the average additional running times of Algorithms 3 and 6 in Table 5.2 we see that the running time of Algorithm 6 is slightly higher than that of Algorithm 3. However, since these values are effectively measured in the very low microsecond range the performance of the two algorithms can be seen as almost being equal and that the additional cost of the bookkeeping done by Algorithm 6 is negligible.

5.3 MinA Extraction

In this section we look at the performance of Algorithms 6 and 7 in terms of MinA extraction for the $\mathcal{O}_{Nci}$ and $\mathcal{O}_{Go}$ ontologies. Tests were not performed on $\mathcal{O}_{SNOMED}$ as it contains GCI's. The presence of GCI's causes an exponential blow-up in the number of bidirectional reachability-based sub-modules to test, where each such sub-module does not necessarily correspond to a MinA. This Algorithm 7 cannot be used to extract all MinAs. The tests were performed as follows:

Similar to the process used for testing module extraction, we did not normalise the TBoxes nor did we convert them to their equivalent CFG representations. Instead, during runtime, the form of all axioms were checked and an on-the-fly implicit conversion were done, the results of which were used to index axioms so that the algorithms could be applied as if explicit normalisation and conversion was done. Using this approach we eliminate all minimality issues usually associated with normalisation and de-normalisation as well as mapping axioms to CFG and vice versa.

For every concept name $A \in$ in Sig($\mathcal{O}$) we extracted $\mathcal{O}_A^{reach}$, then Algorithm 6 was called for each concept name $B \in$Sig($\mathcal{O}_A^{reach}$) in order to extract $\mathcal{O}_{A \leftrightarrow B}^{reach}$. For each of these indexed bidirectional reachability-based modules we then extracted all possible minimal bidirectional reachability-based sub-modules M_i, the standard HerMit reasoner was then called to test each minimal module M_i in order to see whether or not $M_i \models A \sqsubseteq B$.

Instead of doing the additional subsumption tests for each minimal module, we could have simply eliminated all axioms dependant upon existential restrictions as described in Section 4.3.2. However, we wanted to test the performance of the algorithm on all possible minimal bidirectional reachability-based sub-modules. The columns in the table are organised as follows:

- Ontology - The ontology for which the MinAs are being extracted.

- $| \mathcal{O}^{reach}_{A \leftrightarrow B} |$ - The average number of axioms for all bidirectional reachability-based sub-modules.
- $| Min(\mathcal{O}^{reach}_{A \leftrightarrow B}) |$ - The average number of minimal bidirectional reachability-based sub-modules.
- T($Min(\mathcal{O}^{reach}_{A \leftrightarrow B})$) - The additional time, in seconds, required to extract all minimal bidirectional reachability-based sub-modules i.e the running time of Algorithm 7.
- %MinAs - The percentage of minimal bidirectional reachability-based sub-modules that are MinAs.
- |MinA| - The average size of each MinA.
- T(MinA) - The additional time required to test subsumption for all minimal modules, i.e. To calculate the total time to extract all MinAs from the ontology = T($Min(\mathcal{O}^{reach}_{A \leftrightarrow B})$) + T(MinA).

Average Values

Ontology	$\| \mathcal{O}^{reach}_{A \leftrightarrow B} \|$	$\| Min(\mathcal{O}^{reach}_{A \leftrightarrow B}) \|$	T($Min(\mathcal{O}^{reach}_{A \leftrightarrow B})$)	%MinAs	|MinA|	T(MinA)
$\mathcal{O}_{Go}$	13	2.720188	0.000023	89.18%	3.298866	0.005472
$\mathcal{O}_{Nci}$	26	2.180851	0.000014	91.47%	3.721915	0.002842

Maximum Values

Ontology	$\| \mathcal{O}^{reach}_{A \leftrightarrow B} \|$	$\| Min(\mathcal{O}^{reach}_{A \leftrightarrow B}) \|$	T($Min(\mathcal{O}^{reach}_{A \leftrightarrow B})$)	%MinAs	|MinA|	T(MinA)
$\mathcal{O}_{Go}$	68	41.750000	0.000431	100.00%	9.359477	0.121236
$\mathcal{O}_{Nci}$	398	72.375000	0.001298	100.00%	10.480000	0.199400

Median Values

Ontology	$\| \mathcal{O}^{reach}_{A \leftrightarrow B} \|$	$\| Min(\mathcal{O}^{reach}_{A \leftrightarrow B}) \|$	T($Min(\mathcal{O}^{reach}_{A \leftrightarrow B})$)	%MinAs	|MinA|	T(MinA)
$\mathcal{O}_{Go}$	10	1.500000	0.000013	100.00%	3.000000	0.002327
$\mathcal{O}_{Nci}$	11	1.000000	0.000010	100.00%	3.500000	0.001452

Table 5.5: MinA extraction using Algorithms 6 and 7

From Table 5.5 we see that the results are very similar for both ontologies tested. On average there are between 2 and 3 minimal bidirectional reachability-based sub-modules for each possible subsumption statement. From these, about 90% are MinAs, each of which contains between 3 and 4 axioms on average. The total additional time required to extract all minimal bidirectional reachability-based sub-modules using Algorithm 7 is in the low hundredths of a millisecond. The most expensive costs incurred, occurred when we tested whether subsumption holds for each minimal bidirectional module, with a total running time for testing all minimal modules for a statement of interest in the low to mid millisecond range.

This testing however is unnecessary and is only included to illustrate the costs involved in testing all minimal modules for subsumption.

Overall, once a bottom-up reachability-based module has been extracted, the additional runtime costs incurred in order to extract a bidirectional reachability module together with all minimal bidirectional reachability-based sub-modules, and thus MinAs, is less that 1% of the cost of performing a subsumption test on a single MinA. This makes MinA extraction, for acyclic $\mathcal{EL}$ TBoxes consisting of only primitive concept definitions, negligible. For more expressive DLs in the $\mathcal{EL}$ family, the average reduction of 59% in the number of axioms for bidirectional reachability-based modules, over that of standard reachability-based modules, will still yield a significant improvement during MinA extraction when standard black-box algorithms are used. The projected improvement in performance of these algorithms results not only from the 59% reduction in the number of axioms to consider during a single MinA extraction cycle, but a 59% reduction in the number of all axioms over all possible MinAs for a subsumption statement. Testing the performance of these algorithms however is beyond the scope of this dissertation.

Chapter 6

Conclusion

In this dissertation we investigated syntactic module extraction methods for the inexpressive $\mathcal{EL}$ family of description logics. Though it has been shown that extracting minimal modules is undecidable for $\mathcal{ALC}$, various heuristic based extraction algorithms, such as bottom-up reachability-based module extraction, have been introduced in the literature. The various functional roles modules are to perform, necessitate different functional specifications of exactly what a module should be. However, during this dissertation, we limited our attention to the extraction of modules that would aid in reasoning tasks such as subsumption testing and MinA extraction.

Discussion and Results

Given a specific subsumption statement, bottom-up reachability-based modules have been shown to significantly reduce the running time of these reasoning services. It has however been criticised for only utilising the subsumee of the entailment to extract a module, never utilising the subsumer to further reduce the number of axioms in a module, thus in certain cases not providing the expected improvements. As a first step to addressing this issue, we introduced the notion of top-down reachability-based modules. Instead of utilising the subsumee of an entailment to extract modules, top-down reachability uses the subsumer to extract modules. However, these modules are generally very large and can be criticized in a similar fashion to bottom up reachability-based modules for considering only one of the operands in a subsumption statement. We then combined the two reachability heuristics, forming the basis for bidirectional reachability-based modules, which consider both the subsumee and the subsumer of an entailment when extracting modules.

The empirical results from Section 5.2 show that, for the bio-medical ontologies tested, bidirectional reachability-based module extraction provides, on average, an additional 59%

reduction in the size of the modules extracted over that of bottom-up reachability-based modules. Taking into consideration that this reduction, for large ontologies like SNOMED, can be obtained for less than a 1% increase in the running time of the bottom-up reachability module extraction algorithm, further increases the utility to be gained from bidirectional reachability-based modules.

Since bidirectional reachability-based modules have been shown to be strong subsumption modules, and each MinA for a subsumption statement is also a bidirectional reachability-based module for that statement, we investigated the relationship between MinAs and bidirectional reachability-based sub-modules more closely. We determined that for the DL $\mathcal{HL}$, every MinA corresponds exactly to a minimal bidirectional reachability-based sub-module. We provided an algorithm based on the Earley algorithm for extracting parse trees given a CFG, that extracts an indexed bidirectional reachability-based module, from which all bidirectional reachability-based sub-modules may be extracted, including all minimal ones. We then introduced an algorithm to extract all MinAs in $\mathcal{HL}$.

Having determined the exact relationship between MinAs and bidirectional reachability-based sub-modules for $\mathcal{HL}$, we investigated the relationship between MinAs in $\mathcal{EL}$ and bidirectional reachability-based sub-modules. We found that for acyclic $\mathcal{EL}$ TBoxes consisting of only primitive concept definitions, every MinA corresponds to a minimal bidirectional reachability-based sub-module, however, unlike for $\mathcal{HL}$, not every minimal bidirectional reachability-based module corresponds to a MinA. Thus, though the same algorithms for extracting all $\mathcal{HL}$ MinAs could be used to extract all MinAs in $\mathcal{EL}$, additional processing would be required to separate the minimal modules that correspond to MinAs and those that don't. We then showed that as soon as general $\mathcal{EL}$ TBoxes are introduced, there is an exponential number of possible bidirectional reachability-based sub-modules that must be considered in order to extract a single MinA.

Future Work

The top-down reachability heuristic performs badly in the presence of existential quantifications, especially when those are used extensively throughout the ontology. A case in point is the roleGroup role name symbol in the Snomed Ontology. Possible improvements to the top-down reachability heuristic may be obtained by investigating these more closely and determining the syntactic relationship between the role name symbols and the role filler symbols

and their effect on top-down and thus bidirectional reachability-based module extraction.

Syntactic locality based modules, an equivalent to bottom-up reachability-based modules, have been used by Suntisrivaraporn et al. (2008) to improve MinA extraction for the DL $\mathcal{SHOIN}$. The empirical benefits of bidirectional reachability-based module extraction methods, over that of bottom-up reachability-based modules for $\mathcal{EL}$, warrants further research into extending the top-down and thus bidirectional reachability heuristics to more expressive DL's such as the DL $\mathcal{SROIQ}$ (Horrocks et al. 2006) which forms the basis for the W3C standardized OWL 2 DL.

Though bidirectional reachability-based modules reduce the size of ontologies, and thus may improve the extraction of all MinAs using black-box methods, we have seen that for a single $\mathcal{EL}$ MinA, there may be an exponential number of bidirectional reachability-based modules. Black-box MinA extraction algorithms arbitrarily removes axioms from an ontology, until no more may be removed without invalidating a subsumption statement, after each removal operation a subsumption test is usually run on the remaining set of axioms. This process continues until no more axioms may be removed and thus a MinA has been found. However, we know that every MinA is also a bidirectional reachability-based module. Investigating how axiom removal may be guided in such a way that bidirectional reachability is always preserved may therefore improve these black-box algorithms and reduce the overall number of subsumption tests required during every MinA extraction cycle.

Bibliography

Ausiello, G., Franciosa, P. G. & Frigioni, D. (2001), Directed hypergraphs: Problems, algorithmic results, and a novel decremental approach, *in* 'Proceedings of the Seventh Italian Conference on Theoretical Computer Science (ICTCS)', Vol. 2202 of *Lecture Notes in Computer Science*, Springer, London, UK, UK, pp. 312–327.

Baader, F. (1996), 'Using automata theory for characterizing the semantics of terminological cycles', *Annals of Mathematics and Artificial Intelligence* **18**(2-4), 175–219.

Baader, F. (2003), Terminological cycles in a description logic with existential restrictions, *in* G. Gottlob & T. Walsh, eds, 'Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)', Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 325–330.

Baader, F., Brandt, S. & Lutz, C. (2005), Pushing the $\mathcal{EL}$ envelope, *in* 'Proceedings of the 19th International Conference on Artificial Intelligence (IJCAI-05)', Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 364–369.

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D. & Patel-Schneider, P. F., eds (2007), *The Description Logic Handbook: Theory, Implementation and Applications*, 2 edn, Cambridge University Press, Cambridge, UK.

Baader, F., Hladik, J., Lutz, C. & Wolter, F. (2003), From tableaux to automata for description logics, *in* M. Vardi & A. Voronkov, eds, 'Proceedings of the 10th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-03)', Vol. 2850 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, pp. 1–32.

Baader, F. & Hollunder, B. (1995), 'Embedding defaults into terminological knowledge representation formalisms.', *Automated Reasoning* **14**, 149–180.

Baader, F., Kusters, R. & Molitor, R. (1998), Computing least common subsumers in description logics with existential restrictions, LTCS-Report LTCS-98-06, LuFG Theoretical Computer Science, RWTH Aachen, Germany.

Baader, F. & Penaloza, R. (2007), Axiom pinpointing in general tableaux, *in* N. Olivetti, ed., 'Proceedings of the 16th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods (TABLEAUX'07)', Lecture Notes in Computer Science, Springer, Berlin Heidelberg, pp. 11–27.

Barnaras, A., Laresgoiti, I. & Corera, J. (1996), Building and reusing ontologies for electrical network applications, *in* W. Wahlster, ed., 'Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)', John Wiley and Sons, pp. 298–302.

Brachman, R. J. & Levesque, H. J. (1984), The tractability of subsumption in frame-based description languages, *in* 'Proceedings of the 9th Conference on Artificial Intelligence (AAAI-84)', AAAI Press, Menlo Park, California, pp. 34–37.

Brachman, R. J. & Levesque, H. J., eds (1985), *Readings in Knowledge Representation*, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Brachman, R. J. & Schmolze, J. G. (1985), 'An overview of the KL-ONE knowledge representation system', *Cognitive Science* **9**(2), 171–216.

Brandt, S. (2004*a*), Polynomial time reasoning in a description logic with existential restrictions, gci axioms, and - what else?, *in* R. L. de Mántaras & L. Saitta, eds, 'Proceedings of the 16th European Conference on Artifcial Intelligence (ECAI-2004)', IOS Press, Amsterdam, The Netherlands, pp. 298–302.

Brandt, S. (2004*b*), Subsumption and instance problem in $\mathcal{ELH}$ w.r.t. general tboxes, LTCS-Report LTCS-04-04, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany. See http://lat.inf.tu-dresden.de/research/reports.html.

Calvanese, D. (1996), Reasoning with inclusion axioms in description logics: Algorithms and complexity, *in* 'Proceedings of the 12th European Conference on Artificial Intelligence (ECAI-96)', IOS Press, Amsterdam, The Netherlands, pp. 303–307.

Chiang, Y. & Fu, K. (1984), 'Parallel parsing algorithms and VLSI implementation for syntactic pattern recognition', *IEEE Transactions on Pattern Analysis and Machine Intelligence* **6**(3), 302–314.

Cote, R., Rothwell, D., Palotay, J., Beckett, R. & Brochu, L. (1993), The systematized nomenclature of human and veterinary medicine, Technical report, SNOMED International.
URL: *http://www.ihtsdo.org/*

Cuenca Grau, B., Horrocks, I., Kazakov, Y. & Sattler, U. (2007), Just the right amount: extracting modules from ontologies, *in* C. Williamson & M. Zurko, eds, 'Proceedings of the 16th International Conference on World Wide Web (WWW '07)', ACM, New York, NY, USA, pp. 717–726.

Cuenca Grau, B., Horrocks, I., Kazakov, Y. & Sattler, U. (2008), 'Modular reuse of ontologies: Theory and practice.', *Journal of Artificial Intelligence Research (JAIR)* **31**, 273–318.

Cuenca Grau, B., Parsia, B., Sirin, E. & Kalyanpur, A. (2005), Modularity and web ontologies, *in* 'Proceedings of the KCAp-2005 Workshop on Ontology management', CEUR Workshop Proceedings, CEUR-WS.
URL: *http://ceur-ws.org/*

Donini, F. M. (2007), Complexity of reasoning, *in* F. Baader, D. Calvanese, D. Nardi & P. Patel-Schneider, eds, 'The Description Logic Handbook: Theory, Implementation and Application', 2 edn, Cambridge University Press, chapter 3, pp. 101–141.

Doran, P., Tamma, V. & Iannone, L. (2007), Ontology module extraction for ontology reuse: an ontology engineering perspective, *in* 'Proceedings of the 16th ACM conference on Conference on information and knowledge management (CIKM '07)', ACM, New York, NY, USA, pp. 61–70.

Dowling, W. F. & Gallier, J. (1984), 'Linear-time algorithms for testing the satisfiability of propositional Horn formulae', *Journal of Logic programming* **1**(3), 267–284.

Du, J., Qi, G. & Ji, Q. (2009), Goal-directed module extraction for explaining OWL DL entailments, *in* A. Bernstein, D. R. Karger, T. Heath, L. Feigenbaum, D. Maynard, E. Motta & K. Thirunarayan, eds, 'Proceedings of the 18th International Semantic Web Conference (ISWC'09)', Vol. 5823 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, pp. 163–179.

Earley, J. (1970), 'An efficient context-free parsing algorithm', *Communications of the Association for Computing Machinery* **13**(2), 94–102.

Givan, R., McAllester, D. A., Witty, C. & Kozen, D. (2002), 'Tarskian set constraints', *Information and Computation* **174**(2), 105–131.

Gruber, T. R. (1993), 'A translation approach to portable ontology specification', *Knowledge Acquisition* **5**(2), 199–220.

Haarslev, V. & Möller, R. (2001), Racer system description, *in* R. Goré, A. Leitsch & T. Nipkow, eds, 'Proceedings of the 1st International Joint Conference on Automated Reasoning (IJCAR'01)', Vol. 2083 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, pp. 701–705.

Hayes, P. J. (1979), The logic of frames, *in* D. Metzing, ed., 'Frame Conceptions and Text Understanding', Walter de Gruyter and Co., Berlin, Germany, pp. 46–61.

Horrocks, I. (1998), 'Using an expressive description logic: Fact or fiction?', *Proceedings of the 6th International Conference on Principles of Knowledge Representation and Reasoning (KR-98)* pp. 636–647.

Horrocks, I., Kutz, O. & Sattler, U. (2006), The even more irresistible $\mathcal{SROIQ}$, *in* P. Doherty, J. Mylopoulos & C. Welty, eds, 'Principles of Knowledge Representation and Reasoning: Proceedings of the 10th International Conference (KR-06)', AAAI Press, Menlo Park, California, pp. 57–67.
URL: *http://www.aaai.org/Press/Proceedings/kr06.php*

Horrocks, I., Rector, A. L. & Goble, C. A. (1996), A description logic based schema for the classification of medical data, Vol. 4 of *CEUR Workshop Proceedings*, CEUR-WS.
URL: *http://ceur-ws.org/*

Horrocks, I. & Sattler, U. (2004), 'Decidability of $\mathcal{SHIQ}$ with complex role inclusion axioms', *Artificial Intelligence* **160**(1), 79–104.

Horrocks, I., Sattler, U. & Tobies, S. (2000), 'Practical reasoning for very expressive description logics', *Journal of the Interest Group in Pure and Applied Logic* **8**(3), 239–264.

Jiménez-Ruiz, E., Cuenca Grau, B., Sattler, U., Schneider, T. & Berl, R. (2008), Safe and

economic re-use of ontologies: a logic-based methodology and tool support, *in* '5th European Semantic Web Conference (ESWC2008)', Vol. 5021 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, pp. 185–199.

Jurafsky, D. S. & Martin, J. H. (2008), *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*, 2 edn, Prentice Hall, Upper Saddle River, NJ.

Kalyanpur, A., Parsia, B., Horridge, M. & Sirin, E. (2007), Finding all justifications of OWL DL entailments, *in* K. A. et al., ed., 'Proceedings of the 18th International Semantic Web Conference (ISWC'07)', Vol. 4825 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, pp. 267–280.

Kazakov, Y. & Nivelle, H. D. (2003), Subsumption of concepts in $\mathcal{FL}_0$ for (cyclic) terminologies with respect to descriptive semantics is pspace-complete, *in* 'Proceedings of the 2003 International Workshop on Description Logics (DL2003)', Vol. 81 of *CEUR Workshop Proceedings*, CEUR-WS.
URL: *http://ceur-ws.org/*

Lutz, C. & Satler, U. (2000), Mary likes all cats, *in* 'Proceedings of the 2000 International Workshop on Description Logics (DL-2000)', Vol. 3 of *CEUR Workshop Proceedings*, CEUR-WS, pp. 213–226.
URL: *http://ceur-ws.org/*

Lutz, C., Walther, D. & Wolter, F. (2007), Conservative extensions in expressive description logics, *in* 'Proceedings of the Twentieth International Joint Conference on Articial Intelligence (IJCAI-07)', AAAI Press, Menlo Park, California, pp. 453–459.

Meyer, T., Lee, K., Pan, J. & Booth, R. (2006), Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$, *in* 'Proceedings of the 21^{st} National Conference on Artificial Intelligence (AAAI'06)', AAAI Press, Menlo Park, California, pp. 269–274.

Minsky, M. (1975), A framework for representing knowledge, *in* P. Winston, ed., 'The Psychology of Computer Vision', McGraw-Hill, New York, pp. 211–277.

Nebel, B. (1990), 'Terminological reasoning is inherently intractable', *Artificial Intelligence* **43**, 235–249.

Nortjé, R., Britz, K. & Meyer, T. (2009), Finding $\mathcal{EL}^+$ justifications using the Earley parsing algorithm, *in* T. Meyer & K. Taylor, eds, 'Australasian Ontology Workshop 2009 (AOW 2009)', Vol. 112 of *CRPIT*, ACS, Melbourne, Australia, pp. 27–35.
URL: *http://crpit.com/confpapers/CRPITV112Nortje.pdf*

Noy, N. F. & Musen, M. A. (2004), Specifying ontology views by traversal, *in* S. A. McIlraith, D. Plexousakis & F. van Harmelen, eds, 'Proceedings of the International Semantic Web Conference (ISWC'04)', number 3298 *in* 'Lecture Notes in Computer Science', Springer, Berlin Heidelberg, pp. 713–725.

Papadimitriou, C. H. & Steiglitz, K., eds (1989), *Combinatorial Optimization: Algorithms and Complexity*, 1 edn, Dover Publications Inc, Mineola, NY.

Pavlatos, C., Koulouris, A. & Papakonstantinou, G. (2003), Hardware implementation of syntactic pattern recognition algorithms, *in* M. Hamza, ed., 'IASTED International Conference on Signal Processing and Pattern Analysis', Acta Press, Calgary, Canada, pp. 360–365.

Qi, G., Ji, Q. & Haase, P. (2008), A relevance-based algorithm for finding justifications of DL entailments, Technical report, University of Karlsrube. http://www.aifb.uni-karlsrube.de/WBS/qgi/papers/RegAlg.pdf.

Quillian, R. M. (1967), 'Word concepts: A theory and simulation of some basic capabilities', *Behavioral Science* **12**, 410–430. Republished in Brachman & Levesque (1985).

Rector, A. (2007), Medical informatics, *in* F. Baader, D. Calvanese, D. Nardi & P. Patel-Schneider, eds, 'The Description Logic Handbook: Theory, Implementation and Application', 2 edn, Cambridge University Press, chapter 13, pp. 415–435.

Reiter, R. (1987), 'A theory of diagnoses from first principles', *Artificial Intelligence* **32**, 57–95.

Sattler, U., 0002, T. S. & Zakharyaschev, M. (2009), Which kind of module should i extract?, *in* 'Description Logics'.

Schlobach, S. & Cornet, R. (2003), Non-standard reasoning services for the debugging of description logic terminologies, *in* G. Gottlob & T. Walsh, eds, 'Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI-03)', Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 355–362.

Schmidt-Schaubβ, M. & Smolka, G. (1991), 'Attributive concept descriptions with complements.', *Artificial Intelligence* **48**, 1–26.

Spackman, K. A. (2001), 'Normal forms for description logic expressions of clinical concepts in SNOMED RT', *Journal of the American Medical Informatics Association: Proccedings of the annual AMIA Symposium* pp. 627–631.
URL: *http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2243264/*

Stuckenschmidt, H. & Klein, M. (2004), Structure-based partitioning of large concept hierarchies, *in* 'Proceedings of the 3rd International Semantic Web Conference', Hiroshima, Japan.

Suntisrivaraporn, B. (2005), Optimization and implementation of subsumption algorithms for the description logic $\mathcal{EL}$ with cyclic TBoxes and general concept inclusion axioms, Master's thesis, Technical University of Dresden.

Suntisrivaraporn, B. (2009), Polynomial-Time Reasoning Support for Design and Maintenance of Large-Scale Biomedical Ontologies, PhD thesis, Technical University of Dresden.

Suntisrivaraporn, B., Qi, G., Ji, Q. & Haase, P. (2008), A modularization-based approach to finding all justifications for OWL DL entailments, *in* J. Dominque & C. Anutariya, eds, 'Proceedings of the Asian Semantic Web Conference', Vol. 5367 of *Lecture Notes in Computer Science*, Springer, Berlin Heidelberg, pp. 1–15.

Woods, W. A. & Schmolze, J. G. (1992), 'The KL-ONE family', *Computers and Mathematics with Applications* **23**(2-5), 133–177.