



FREIE UNIVERSITÄT BOZEN
LIBERA UNIVERSITÀ DI BOLZANO
FREE UNIVERSITY OF BOZEN - BOLZANO

FREE UNIVERSITY OF BOZEN - BOLZANO

Diagrammatic Representation of OWL Ontologies

by

Mohammad Aminur Rashid

Supervised By

Professor Sergio Tessaris

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Science

in the
Faculty Of Computer Science

March 2009

*“Real transformation requires that you have a beginner’s mind, that you are willing to say,
‘Whatever I think I know may be wrong’”*

’Layne and Paul Cutright’

FREE UNIVERSITY OF BOZEN-BOLZANO

Abstract

Faculty Of Computer Science

Master in Computer Science

by [Mohammad Aminur Rashid](#)

The diagrammatic representation in knowledge formalism is widely used in database in the form of ER, UML diagram. The use of diagram help user understand the system easily than logical view. Advanced Knowledge representation formalism like OWL-DL W3C recommendation are logic based which hide well known logical pattern into logical implications (so called axioms). Inspired from the Intelligent Conceptual Modelling Tool (ICOM) [1] which is a CASE tool to design multiple EER diagram with constraints. The tool reason on the model by encoding them in single description logic. The aim of my thesis is to study and use semantic preserving operations to manipulate OWL ontologies in order to isolate the implicit information which can be represented diagrammatically (e.g. ISA, domain/range restrictions, mandatory participations constraints etc).

Acknowledgements

I would like to express my sincere gratitude to all those who gave me the possibility to complete this thesis. I want to thank the Faculty of Computer Science of Free University of Bozen-Bolzano for giving me permission to commence this thesis. I am deeply indebted to my supervisor Prof. Dr. Sergio Tessaris whose help, guidance, stimulating suggestions and encouragement helped me in all the time for working and writing of this thesis. I also want to thank some of my teachers whose lectures helped me a lot to understand my work. Professor Diego Calvanese, Professor Alessandro Artale and Professor Enrico Franconi are among them for their contribution. I want to thank those people who help me with hints and suggestions in the Computer Science Faculty.

I owe my loving thanks to my parents, my wife and my son. They have lost a lot due to my research. Without their encouragement and understanding it would have been impossible for me to finish this work.

Contents

Abstract	ii
Acknowledgements	iii
Table of Contents	iv
List of Figures	vi
List of Tables	vii
1 Introduction	1
1.1 Preliminaries	2
1.1.1 OWL: Semantic Web Language	2
1.1.2 Knowledge Base and Description Logic	2
1.1.3 Description Language	3
1.1.4 Diagrammatic View of the Knowledge Base	4
1.1.5 Thesis Outline	5
2 Description Logics	6
2.1 Description Logics	6
2.1.1 Semantics	7
2.1.2 DL Terminology	8
2.1.3 Reasoning on T-Box	9
2.1.4 T-Box Equivalence	10
2.2 Theoretical and Implemented DLs	12
2.2.1 DLR_{ifd} : DL with functional dependency	12
2.2.1.1 Semantics of a KB in DLR_{ifd}	13
2.2.2 \mathcal{ALCQI}	13
2.2.3 \mathcal{ALC}	14
3 Logic in UML Class Diagram	16
3.1 Different elements of UML corresponding to FOL	16
3.1.1 \mathcal{ALC} and UML class diagram	23
3.2 Proposed Representation	24
4 Normal Form & Absorption	26
4.1 Normal Form	26
4.1.1 \mathcal{ALN} Normal Form	27
4.1.2 \mathcal{ALC} Normal Form	28
4.1.3 Suitable Normal Form	30
4.2 Absorption	31
4.2.1 Unfolding	31

4.2.2	Lazy Unfolding	32
4.2.3	GCI Absorption	32
4.2.4	Dealing with Domain and Range Restriction:	34
4.2.5	The Absorption Algorithm	36
5	Working with OWL Ontologies	38
5.1	Racer Pro as Description Logic System	38
5.1.1	Ontology Collection	40
5.1.2	The Result by RACER	41
5.1.3	Search Target	42
5.1.4	Bug in RacerPro	42
6	Axiom Transformation for Diagrammatic View	44
6.1	Processing Axioms	44
6.1.1	Other form of Axioms	47
6.2	Systemize the procedure	48
6.2.1	Preprocessing	48
6.2.1.1	Simplification	48
6.2.1.2	Normal Form and Absorption	49
6.2.2	Transformation Rules	51
6.2.2.1	Transformation Rules	51
6.2.3	Representation of Axioms in Logic Programs	51
6.3	Implementation	53
7	Conclusion	55
	Bibliography	56

List of Figures

1.1	An example of Network-based representation	3
1.2	An example of Class	4
1.3	An example of Relation	5
2.1	T-Box Example	9
2.2	T-Box Unfolding Example	9
2.3	Equivalence of axioms	10
4.1	GCI Absorption	33
6.1	Axiom Absorption	44
6.2	Absorption of Negated axiom	50
6.3	Domain Absorption through Inverse Role	50
6.4	Architecture of axiom processing with Transformation rules	53
6.5	Diagrammatic view of Transformed Knowledge Base	54

List of Tables

1.1	Syntax of Attributive Language	4
2.1	Syntax and Semantics of Concept Expression	7
2.2	Abbreviation of \mathcal{DLR} notation	12
2.3	\mathcal{ALCQI} abbreviation	14
2.4	\mathcal{ALC} Syntax and Semantics	14
3.1	FOL notation corresponding to UML diagram	18
3.2	FOL corresponding to properties of diagram	19
3.3	UML and corresponding $\mathcal{DLR}_{\{\uparrow\}}$	20
3.4	\mathcal{ALCQI} notation corresponding to UML diagram	21
3.5	Reasoning Task in \mathcal{ALCQI} for UML diagram	22
3.6	\mathcal{ALC} KB and corresponding UML diagram	23
3.7	Proposed Covering by Graphviz	24
3.8	Proposed diagram for DL elements	25
4.1	Handling Inverse Role	35
5.1	Racer Pro notation corresponding to DL notation	39
6.1	Transformation Rules for T-Box axioms	51
6.2	DL transformation and corresponding Prolog notation	52
6.3	Example of RacerPro Absorbed T-Box	54
6.4	Transformed T-Box	54

Chapter 1

Introduction

The growing interest in the Semantic Web and Web Ontology Language (OWL)[2] reveals the power of Description Logics in different projects and in industry. Description Logics (DLs) are a form of formalism in Knowledge Representation which uses frames and semantic networks for large number of axioms. Therefore for performance reasons, optimization algorithms and different techniques are employed for reasoning on Knowledge Base. Several works on reasoning on the diagram, based on Description Logic Knowledge Base [3] opens a big scope for the system designer. Because of Diagrammatic view of a system has more appeal than textual description to the customer, so it makes more sense to work on diagram and have the reasoning power as well. It is more comprehensible than Logical notation. Description Logic based knowledge representation systems have proved useful in a range of application domain and also system design, but still there are some limitations on expressivity and design view.

This thesis investigates the issue of providing a suitable representation of Knowledge Base for an expressive Description Logic so that it can be represented in the form of diagram. While most of the tools like RacerPro¹, FaCT++² and Pellet³ display the hierarchical structure of Knowledge Base, our intention is to provide a suitable method to transform the Knowledge Base (OWL Ontology) to some suitable form so that it can be represented in the form of diagram similar to UML⁴ Class diagram.

Before we discuss our proposed method in detail, we first discuss some basic terminology and other preliminaries of our work.

¹<http://www.racer-systems.com/>

²<http://owl.man.ac.uk/factplusplus/>

³<http://clarkparsia.com/pellet/>

⁴<http://www.uml.org/>

1.1 Preliminaries

1.1.1 OWL: Semantic Web Language

The Semantic Web ⁵ aims to make resources on the web available for automated processing by the system. The semantics of information and services on the web is defined, making it possible for the web to understand and satisfy the requests of people and machines to use the web content [4]. The main standards for the Semantic Web proposed by W3C is the Web Ontology Language (OWL). This is based on other two standards: Resource Description Framework (RDF) [5] and the corresponding schema language RDF Schema (RDFS) [6]. OWL is mainly used for Ontology representation language. State-of-the-art description logic for ontology representation in the form of T-Box and A-Box is represented in the form of OWL for the Web. There are other standard language such as KRSS ⁶, XML, Manchester OWL ⁷ for the ontology which are used by different research groups, but for the web the W3C standard language is the OWL. The rich semantics of OWL provides powerful reasoning capabilities that help build, maintain and query domain models for many purposes. In short OWL is the result of combining an expressive Description Logic (DL) with techniques and standards of the Web.

1.1.2 Knowledge Base and Description Logic

Description Logic as Knowledge Representation System support the logical description of concepts and roles and their combination, constructed by different operators, to have a more complex description. In DL terminology a *concept* denotes a set of individuals that belongs to that particular domain while *role* denotes relationships between concepts. As for example, for a concept *woman* with existing role *haschild* form another new concept *mother*. It is also used to represent facts about individuals based on concept and role. As for example, it can be asserted that Mary is a *woman* (concept assertion) and Mary *haschild* Monika (role assertion). This type of Knowledge Representation was formalised in semantic network and frame based representation based on first-order logic (FOL). Figure 1.1 gives an example of network-based representation [7].

In network-based representation the basic elements are nodes and links. Concepts are represented by nodes and Relationships between the concepts are represented by links. In DL based KB, a concept represents a set of individuals while links as properties of each individual that has either some other concept or attribute.

Another important characteristic of DL based system is the reasoning capability. For example, **Woman** can be defined as concept of **Female Person** from the example, then we can infer that a *mother* is also a *woman* because **Mother** is both a **Female** and a **Person**. Therefore the most important characteristics of the system is that it has the ability to find the implicit knowledge from its explicit represented knowledge.

⁵The Semantic Web is an evolving extension of the World Wide Web

⁶<http://www.bell-labs.com/user/pfps/papers/krss-spec.ps>

⁷http://www.co-ode.org/resources/reference/manchester_syntax/

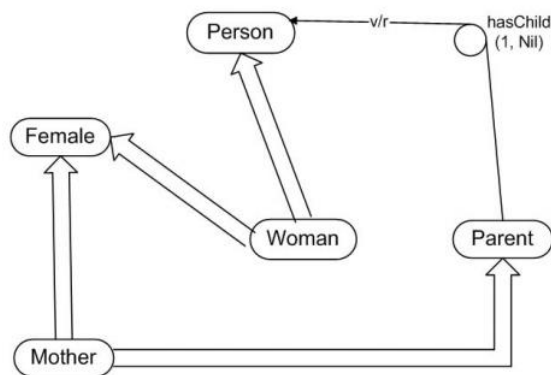


FIGURE 1.1: An example of Network-based representation

From a practical point of view, network-based knowledge representation is not fully satisfactory because of the precise semantics [7]. It may happen that the similar looking components might behave differently from system to system. Therefore the need of semantics for representation structure is necessary for both simple representation and efficient reasoning. The DL based Knowledge Representation can play a role in defining the elements of the structure and therefore a language is must for this kind of work.

1.1.3 Description Language

The basic elements to construct the language is to define atomic concept and atomic role by use of disjoint set of symbols and then build complex concept from *atomic concept* and *atomic roles* by use of set of operators on them. Atomic concepts are unary predicate symbols and atomic roles are binary predicates. Besides, the atomic concept represents a set of individuals and atomic role as relation between these concept names. We use *concept name* for atomic concept for our thesis. For example, from figure 1.1, **Woman**, **Person**, **Mother**, **Parent**, **Female** are *atomic concepts*, and **haschild** is an *atomic role*.

Complex descriptions are built inductively from *atomic concepts* and *atomic roles* through the use of constructors. In our example, Person who is not a **Woman** can be defined as **Person** $\sqcap \neg$ **Woman** and therefore this complex concept is named **Male**. We used the *intersection* and *negation* operator to construct the concept Male. *Union*, *complement* are other construct operators used in DL Language. Another kind of constructs of complex concept is based on role restriction which is used as existential restriction or value restriction. This kind of restriction gives us to express concept of the form *Persons having child* as \exists *haschild*. **Person** and all individuals all of whose children are Persons as \forall *haschild*. **Person**.

In abstract notation, we use the letters A, B for atomic concepts and letter R, S for atomic roles, and letter C, D for complex concept. The complex roles is not considered here. The most common description language is \mathcal{AL} (attributive language)[8]. Other description languages can be seen as extensions of the \mathcal{AL} language.

Concept descriptions in \mathcal{AL} are formed according to the following syntax rule:

A	(atomic concept)
\top	(universal/top concept)
\perp	(bottom concept)
$\neg A$	(atomic negation)
$C \sqcap D$	(intersection)
$\exists R. \top$	(exists restriction)
$\forall R. C$	(value restriction)

TABLE 1.1: Syntax of Attributive Language

In \mathcal{AL} , negation can only be applied to atomic concepts, and the only concept allowed in the role filler of existential quantifications is the top concept \top . Also $A \sqcap B$ is an \mathcal{AL} -concept, but $A \sqcup B$ is not. Again, $\exists R. \top$ is an \mathcal{AL} -concept, but $\exists R. A$ is not.

1.1.4 Diagrammatic View of the Knowledge Base

Knowledge Representation and Reasoning Community on Description Logics (DLs) addressed the issues of representing knowledge for conceptual representation [7]. These logics are specifically designed so that it can be represented in terms of classes and relationships between classes that admit decidable reasoning. UML Class diagram which is one of the most important component of UML are used to model the information on the domain of interest in terms of classes and relationships between them. Several other tools like ArgoUML⁸, Poseidon, Rational Rose⁹ are used for system design for large industrial environment. These tools have features of editing, designing and maintaining multiple UML class diagrams. But these diagrams lack the reasoning power so that for a complicated system, the tool cannot identify inconsistencies and redundancies in the system design.

The first contribution made by [3] showed that such UML class diagrams can be equipped with reasoning services. They showed that DLs are very promising technology for implementing core reasoning service for next generation CASE (Computer Aided Software Engineering) tools. Also their work showed that reasoning on UML class diagram is EXPTIME-hard.

UML class diagrams are the mainstay of object-oriented analysis and design. Class diagrams show the classes of the system, their interrelationships. For the time being, basic notions of class diagram related to Logical formalism are depicted here. Detail will be discussed later.

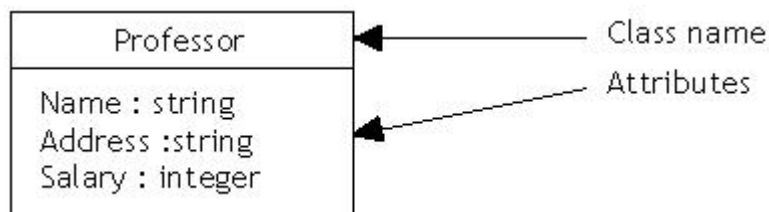


FIGURE 1.2: An example of Class

Classes: A class can be thought of as set of individuals with common features. In our example, **Professor** is a class which shares some common properties. The first part denotes the name of

⁸<http://argouml.tigris.org/>

⁹<http://www-01.ibm.com/software/rational/announce/rose/>

the class which is unique to the whole diagram. The attribute part of a class are optional. In our example it means that each instance of class **Professor** associates to instances of *Name*, *Address* and *Salary*. Hence without loss of generality, a class in UML can be defined as a Concept in Description Logic language.

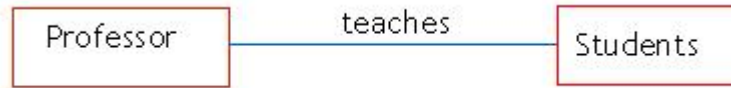


FIGURE 1.3: An example of Relation

Relation or Association: The association relationship is the most common relationship in a class diagram. The association shows the relationship between instances of classes. For example, the class **Professor** is associated with the class **Students** by relation *teaches*. Hence the relation in UML is viewed as Relation in DL Knowledge Base.

1.1.5 Thesis Outline

In the following chapters we describe the work behind our thesis and our proposed method. The remainder of the thesis is organised as follows:

Chapter 2 Presents the standard DL syntax and Semantics, the T-Box view and reasoning capabilities on concept descriptions and subsumption relationships. Different DL Language are also discussed.

Chapter 3 Explains the basic elements of UML and DLs corresponding to UML Class diagram, the semantics of UML Class diagram and our proposed representation for different DL construct.

Chapter 4 Introduces the Normal Form, describes the necessity, and how Normal Form can help the representation of axioms for our work. Absorption, an optimization technique, is discussed along with Absorption algorithm and techniques applied to absorb different axioms in Knowledge Base.

Chapter 5 States the work on OWL ontologies in the RacerPro system. How we transform our OWL knowledge Base to KRSS like syntax are also discussed.

Chapter 6 Explains preprocessing steps and our proposed transformation that are applied on existing Knowledge Base. Also the implementation of our transformation method with example are given.

Chapter 7 Concludes our work with future target for diagrammatic view of our Knowledge Base.

Chapter 2

Description Logics

In this Chapter, we present a formal definition of the syntax and the semantics of Description logics. We introduce the basic description logic \mathcal{ALCN} in section 2.1.

2.1 Description Logics

Description Logics (DLs) are logics serving primarily for formal description of concepts and roles (relations). Up until now we have only the informal DL, but the semantics of symbols used in the language were not discussed. Therefore it is necessary to present the semantics for concept description language. Semantically this logics are found on predicate logic, but their language is formed so that it would be enough for practical modeling purposes and so that the logic would have good computational properties such as decidability. The focus of our thesis in DLs is how the various DL constructs are usable for diagrammatic representation.

The knowledge bases in Description Logics consists of a T-Box \mathcal{T} and an A-Box \mathcal{A} . It is a pair $(\mathcal{T}, \mathcal{A})$ to denote a Knowledge Base. Knowledge is also represented in other way in DL system. Rule axioms ,i.e. axioms of the form $(A \Rightarrow C)$, are sometime used instead of general axioms in T-Box. Here A is an atomic concept and C is an arbitrary concept name. The meaning of this rule axioms is “if an individual is an instance of concept A, then that individual is also an instance of concept C ”. This holds one way, not the other way. Normally in DL KB system the axioms comes in the format of $(C \sqsubseteq D)$ or $(C \equiv D)$. It is noticable that axioms of the format $C \sqsubseteq D$ is not same as rule axiom $(C \Rightarrow D)$. The axiom $C \sqsubseteq D$ means “if there is some individual that is proven to be an instance of C, then it is also an instance of D and if an individual is not proven to be an instance of D, then it is not an instance of C”. In logical notation it is $(C \Rightarrow D)$ and $(\neg D \Rightarrow \neg C)$. So $(C \sqsubseteq D)$ represents two way reasoning. This particular axiom is of main interest for our work since it appears frequently in terminological knowledge base and we have to find a way to manage these types of axioms so that it has suitable form for our representation.

2.1.1 Semantics

The semantics of a concept in Description Logic is interpreted as set of individuals and role as set of pair of individuals. The semantics of DL language is defined by interpretation \mathcal{I} which consists of a non empty set $\Delta^{\mathcal{I}}$ called the domain of interpretation and interpretation function $\langle \cdot^{\mathcal{I}} \rangle$, which assign every atomic concept A to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and every atomic role R to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.

Definition (Interpretation): An interpretation is a pair $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$, where $\Delta^{\mathcal{I}}$ is a non-empty set, called the domain of \mathcal{I} and $\cdot^{\mathcal{I}}$ is a mapping function that maps NC ¹ to $2^{\Delta^{\mathcal{I}}}$ and NR ² to $2^{\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}}$. DLs support wide variety of operations on concept to build new concept expressions.

Table 2.1 presents the semantics of the concept expression.

Name	Syntax	Semantics
top concept	\top	$\Delta^{\mathcal{I}}$
bottom concept	\perp	\emptyset
concept	C	$C^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$
concept negation	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
concept conjunction	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
concept disjunction	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
existential restriction	$\exists R.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} ((x, y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})\}$
universal 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	$\geq nR$	$\{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x) \geq n\}$
number restriction	$\leq nR$	$\{x \in \Delta^{\mathcal{I}} \mid R^{\mathcal{I}}(x) \leq n\}$
qualified number restriction	$\geq nR.C$	$\{x \in \Delta^{\mathcal{I}} \mid R(x) \cap C^{\mathcal{I}} \geq n\}$

TABLE 2.1: Syntax and Semantics of Concept Expression

In DL, a distinction is drawn between T-Box (terminological box) and A-Box (assertional box). In T-Box the knowledge is in the form of axiom which describes relation between concept names and A-Box has sentence which describes the individual belongs to some class. As for example,

Every Parent haschild Person.

This is an example of Terminological axiom which describes that *Parent* class has relation *haschild* with class *Person*. Example of A-Box is

John haschild Peter

which states that *John* an individual of class *Parent* has relation *haschild* with individual *Peter* who is *Person* (i.e Peter is son of John). This holds for Peter and John. For other individual the assertion might be different. Next we introduce different forms of axioms in Terminological Box and their semantics.

¹Atomic Concept Names in T-Box

²Atomic Role Names in T-Box

2.1.2 DL Terminology

The *terminological box* or T-Box is a set of axioms which are some logic expression to represent the schema of Knowledge Base system for a particular domain. The axioms are sentences in description logic that tells the hierarchical properties among different set of individuals or pair of individuals. T-Box or Terminological Box consists of a finite set of axioms with hierarchical or definitional expression. It also introduce new concept names or role names, assert subsumption relations. DL Language \mathcal{ALC} is of particular interest in our case. The axioms in DL T-Box is defined as follows.

$$C \sqsubseteq D \mid C \doteq D$$

where C and D are complex concept expression. This kind of axiom is the basis of \mathcal{ALC} language for defining T-Box.

The semantics of terminological axioms can be defined by Interpretation as before. An interpretation which satisfies every axiom in T-Box T, denoted $I \models T$ is called admissible. The non-empty subset of all admissible interpretation that satisfy all axioms in T-Box is called a model of T. For a given I and for two arbitrary TBoxes T and T_1 , if $I \models T$, we also have $I \models T_1$ and we say T entails T_1 . It is denoted as $T \models T_1$.

By the definition of Interpretation we can draw that T is a model *iff* for each axiom $C \sqsubseteq D \in T$, $C^I \subseteq D^I$ holds, and for $C \doteq D \in T$, $C^I = D^I$ holds. A concept D subsumes a concept C with respect to T *iff* for all admissible interpretation I with $I \models T$, $C^I \subseteq D^I$ holds.

Let T be an arbitrary T-Box of DL and I is an admissible interpretation. Then I is a model of T *iff*, for each axiom $C_1 \sqsubseteq C_2 \in T$, $C_1^I \subseteq C_2^I$ holds and for each $C_1 \doteq C_2 \in T$, $C_1^I = C_2^I$ holds. A concept D subsumes a concept C with respect to T *iff* for all I that are admissible with $I \models T$, $C^I \subseteq D^I$ holds.

The reasoning task is mainly based on T-Box. The assertion in T-Box are known as intensional knowledge that builds the structure of the Knowledge base. The main reasoning tasks are -

Satisfiability: Concept satisfiability means whether a concept C is satisfiable or not with respect to T-Box.

Subsumption: A concept C is subsumed by a concept D with respect to T-Box if concept C is more specific than concept D.

Equivalence: Two concepts C and D are equivalent with respect to T-Box if they are subset of each other for every model I of T-Box.

Disjointness: Two concepts C and D are disjoint with respect to T-Box if set of individuals in C does not belong to set of individuals in D.

2.1.3 Reasoning on T-Box

Every KB system is built to do reasoning to answer the above questions of *Satisfiability*, *Summption*, *Equivalence* and *Disjointness*. Axioms in DL T-Box of the format $(C \sqsubseteq D)$ is easily reducible to concept satisfiability problem. The idea behind this is to make the reasoning procedure easier. Also reasoning based on a concept without a T-Box, i.e., reasoning w.r.t an empty T-Box, is easier as well. The idea is to rewrite the axioms in original T-Box so that the newly built T-Box is semantically equivalent to original one.

There are ways to make equivalent T-Box from the original one. One way is by *unfolding* [9]. We will consider only axioms of the format

$$C \equiv D \text{ or } C \sqsubseteq D$$

Let us consider the following T-Box as an example:

The axioms:

Mother \equiv Woman $\sqcap \exists hasChild.Person$
 Woman \equiv Person \sqcap Female

FIGURE 2.1: T-Box Example

introduces the axiom in which the left hand side is an atomic concept called the *defined concept*. The right-hand side is called *definition*. The concept which occurs only on the right hand side are called *base concepts*. The concepts which occur on the left hand side are *name-concepts*. Here *Person* is *base concept* while *Mother* and *Woman* are *name-concepts*. Axioms with atomic concept A on left of \sqsubseteq (inclusion symbol) which represents only necessary conditions, is called *Primitive Concept*. For example,

$$\text{Human} \sqsubseteq \text{animal} \sqcap \text{biped}$$

It means *Human* is necessarily both an *animal* and a *biped* but it is not sufficient that $\text{animal}(x)$ and $\text{biped}(x)$ infer $\text{Human}(x)$

T-Box in figure 2.1 by replacing the defined concepts occurring on the right-hand side by their definitions recursively. We call the T-Box generated in this way an unfolded T-Box. In this example, we refer to the unfolded T-Box of T as T' as shown in figure 2.2.

Mother \equiv Person \sqcap Female $\sqcap \exists hasChild.Person$
 Woman \equiv Person \sqcap Female

FIGURE 2.2: T-Box Unfolding Example

The defined concept *Woman* in first axiom is unfolded and have the new T-Box T' .

It was proved that the two T-Boxes T and T' are equivalent semantically as they have the same *base-concept* and *name-concept*.

It has been found from above discussion that unfolding of defined concepts in axioms make the reasoning easier and have an semantically equivalent T-Box of original one. The two T-Box s are equivalent since the concept is replaced by its definitional concept. But it is assumed that the axioms are all definition axiom and each atomic concept has only one definition.

Generalization is another technique for reducing concept reasoning w.r.t a T-Box to concept reasoning w.r.t an empty T-Box. As already mentioned that T-Box mainly has two kinds of axioms ($C \equiv D$) or ($C \sqsubseteq D$). By using the following equivalence T-Box can be reduced to axioms of the form $\top \sqsubseteq C$.

$$\begin{aligned} C \sqsubseteq D &\iff \top \sqsubseteq \neg C \sqcup D \\ C \equiv D &\iff \top \sqsubseteq (C \sqcup \neg D) \sqcap (\neg C \sqcup D) \end{aligned}$$

FIGURE 2.3: Equivalence of axioms

C is called the *reduction concept* of \mathcal{T}

2.1.4 T-Box Equivalence

The main target of our thesis is to find a T-Box which is suitable for diagrammatic representation. It comes with T-Box rewriting to have a new T-Box for the diagram. Therefore it is also necessary to find whether the newly built T-Box is semantically equivalent to original one. We have to check whether the two T-Boxes are equivalent or not. Therefore, the T-Box equality checking is one of the topics of our interest.

From the definition of unfolding we know the two T-Boxes are equivalent as we replace the concept name with its definition. From the definition of *generalization*, we know that each T-Box can be reduced to a concept. The resulting concept is called the reduction concept of \mathcal{T} . Then if it can be proved that if two reduction concepts are equivalent, then the corresponding two T-Boxes are also equivalent as well.

Therefore we can say T-Box T_1 and T_2 are equivalent *iff* the corresponding reduction concepts C_1 and C_2 are equivalent. However to deal with axioms of the format $C \sqsubseteq D$, sometimes it is necessary to introduce new concept name to simplify the subsumption relation. It is because of diagrammatic representation of the axioms. But does the new names change the semantics of the original T-Box? That will be discussed next.

Let us assume that we have a T-Box T_1 which has axiom $A \sqsubseteq B_1 \sqcup B_2$. This axiom can be rewritten as two axioms, $A \sqsubseteq B$ and $B \sqsubseteq B_1 \sqcup B_2$. We name the newly built T-Box T_{diag} , means T-Box for diagram. We will check whether the two T-Boxes are equivalent or not by using the reduction method-

Applying the reduction method on axiom $A \sqsubseteq B_1 \sqcup B_2$ of T_1 we get $\top \sqsubseteq \neg A \sqcup (B_1 \sqcup B_2)$ and we name the reduction concept C_1 . Again while applying the reduction method on axioms $A \sqsubseteq B$ and $B \sqsubseteq B_1 \sqcup B_2$ of T_{diag} we get $\top \sqsubseteq \neg A \sqcup B$ and $\top \sqsubseteq \neg B \sqcup (B_1 \sqcup B_2)$. We can rewrite these axioms into one axiom $\top \sqsubseteq (\neg A \sqcup B) \sqcap (\neg B \sqcup (B_1 \sqcup B_2))$. We name this reduction concept C_{diag} . Then we have to prove that whether C_1 and C_{diag} are equivalent or not.

By normalizing and simplyfying (which will be discussed shortly) the concept expression C_{diag} within the axiom in T_{diag} in the following way we get

$$\begin{aligned}
& (\neg A \sqcup B) \sqcap (\neg B \sqcup (B_1 \sqcup B_2)) \\
& \equiv \neg A \sqcup B \sqcap \neg B \sqcup B_1 \sqcup B_2 \\
& \equiv \neg A \sqcup \perp \sqcup B_1 \sqcup B_2 \text{ ; By simplifying } (B \sqcap \neg B) \equiv \perp \\
& \equiv \neg A \sqcup B_1 \sqcup B_2 \\
& \equiv \neg A \sqcup (B_1 \sqcup B_2) \\
& \equiv C_1
\end{aligned}$$

Therefore the two T-Boxes are equivalent. It can be proved by using the interpretation function as well. However, to test the equivalence of two concepts by reduction concepts, is time consuming and complicated. Other indirect techniques are often applied. To introduce other techniques, let us first introduce the definition of direct consequence which is widely used in many logic related fields.

As we discussed before that two Tboxes are equivalent iff $T_1 \models T_2$ and $T_2 \models T_1$, i.e. $T_1 \equiv T_2$. Actually this method is more used than other techniques and it is efficient. But we need to add something more to this technique. It is called *direct consequence*[10].

If $A \Rightarrow B$, then B is called a *direct consequence* of A. This means that if A is true for some interpretation then B is also true in that interpretation. Therefore for two TBoxes T_1 and T_2 , if every axioms in T_2 is direct consequence of T_1 , then $T_1 \models T_2$. As a proof of this proposition lets say I is a given interpretation of T_1 , then $I \models T_1$, means I satisfies all axioms of T_1 . Since T_2 is direct consequence of T_1 , then $I \models T_2$ as well, i.e I satisfies all axioms of T_2 . Thus from the definiton we can say $T_1 \models T_2$ hold. To prove the equivalence of two T-Boxes, we have to prove the other way also, i.e. $T_2 \models T_1$.

Let us take the previous example and prove by using this indirect technique- Axioms in T_1 is $A \sqsubseteq B_1 \sqcup B_2$ and axioms in T_{diag} are $A \sqsubseteq B$ and $B \sqsubseteq B_1 \sqcup B_1$. To prove that $T_1 \models T_{diag}$, we have to show that axioms in T_{diag} are direct consequences of T_1 , i.e.

$$(A \sqsubseteq B_1 \sqcup B_2) \Rightarrow ((A \sqsubseteq B) \sqcap (B \sqsubseteq B_1 \sqcup B_1))$$

Proof: Suppose axiom $A \sqsubseteq B_1 \sqcup B_2$ is satisfiable. Then there is an interpretation I which satisfies the axiom in T_1 , i.e. $A^I \sqsubseteq (B_1 \cup B_2)^I$ holds for some individual x. It means that $x \in (A^I \sqsubseteq (B_1 \cup B_2)^I)$

means if $x \in A^I$, then $x \in (B_1 \cup B_2)^I$

means if $x \in A^I$, then $x \in B_1$ or $x \in B_2$

Now we have to prove that $I \models T_{diag}$, i.e. $x \in ((A \sqsubseteq B)^I \sqcap (B \sqsubseteq B_1 \sqcup B_1)^I)$. It means that

$x \in (A \sqsubseteq B)^I$ and $x \in (B \sqsubseteq B_1 \sqcup B_1)^I$

means if $x \in A^I$, then $x \in B^I$; and if $x \in B^I$, then $x \in (B_1 \cup B_2)^I$

means if $x \in A^I$, then $x \in (B_1 \cup B_2)^I$

since $A^I \sqsubseteq B^I$ and $B^I \sqsubseteq (B_1 \cup B_2)^I$; for sure $A^I \sqsubseteq (B_1 \cup B_2)^I$

Therefore we can say x also satisfies the axioms in T_{diag} and $I \models T_{diag}$. So $T_1 \models T_{diag}$. To prove the direct consequence the other way we have to show-

$$((A \sqsubseteq B) \sqcap (B \sqsubseteq B_1 \sqcup B_2)) \Rightarrow (A \sqsubseteq B_1 \sqcup B_2)$$

The proof is as before.

2.2 Theoretical and Implemented DLs

This section introduces some of the DLs which is related to our work and are necessary for future work on this. A particular DL is characterised by the kinds of concept and role expressions allowed in its description language and the kinds of axiom allowed in its terminologies.

2.2.1 DLR_{ifd} : DL with functional dependency

DL (Description Logic) with support of n-ary relation is called \mathcal{DLR} . A fragment of \mathcal{DLR} with **identification constraints** on concepts and **functional dependency** of n-ary relation is called DLR_{ifd} [11].

Syntax of \mathcal{DLR} :

$$C ::= \top_1 | A | \neg C | C_1 \sqcap C_2 | (\leq k[i]R)$$

$$R ::= \top_n | P | i/n : C | \neg R | R_1 \sqcap R_2$$

Here A, P are atomic concept and atomic role respectively; i denotes a component of a relation of arity n. $i \in 1..n$, k is an non-negative integer.

The following abbreviation are made from above

$C_1 \sqcup C_2$	for	$\neg(\neg C_1 \sqcap \neg C_2)$
$C_1 \Rightarrow C_2$	for	$\neg C_1 \sqcup C_2$
$(\geq k[i]R)$	for	$\neg(\leq k-1[i]R)$
$\exists[i]R$	for	$\geq 1[i]R$
$\forall[i]R$	for	$\neg\exists[i]\neg R$
$R_1 \sqcup R_2$	for	$\neg(\neg R_1 \sqcap \neg R_2)$
$(i/n : C)$	for	$(i : C)$

TABLE 2.2: Abbreviation of \mathcal{DLR} notation

\mathcal{DLR} KB constitutes a finite set of inclusion assertions $R_1 \sqsubseteq R_2$ and $C_1 \sqsubseteq C_2$ where R_1 and R_2 are arbitrary role names with same arity and C_1 and C_2 are arbitrary concept names.

DLR_{ifd} also allows identification constraints defined as

$$(\mathbf{id} \ C [i_1]R_1, \dots, [i_h]R_h)$$

It tells that if for any two instances a, b of class C with tuple t_j and s_j of R_j respectively such that a is the i_j th component of tuple t_j and b is the i_j th component of tuple s_j . Tuple t_j and s_j agrees on all components different from i_j , then a and b are the same object.

DLR_{ifd} also allows functional dependency on roles defined as

$$(\mathbf{fd} \ R \ i_1, \dots, i_h \rightarrow j)$$

where $h \geq 2$. i_1, \dots, i_h, j denotes components of R.

2.2.1.1 Semantics of a KB in DLR_{ifd}

We first define interpretation I that satisfies as assertion-

I satisfies inclusion assertion $R_1 \sqsubseteq R_2 (C_1 \sqsubseteq C_2)$ if $R_1^I \subseteq R_2^I$ (resp, $C_1^I \subseteq C_2^I$)

I satisfies the assertion $(\mathbf{id} \ C[i_1]R_1, \dots, [i_h]R_h)$ for all $a, b \in C^I$ for all $t_1, s_1 \in R_1^I, \dots, t_h, s_h \in R_h^I$

$$\left. \begin{array}{l} a = t_1[i_1] = \dots = t_h[i_h] \\ b = s_1[i_1] = \dots = s_h[i_h] \\ t_j[i] = s_j[i] \text{ for } j \in \{1, \dots, h\} \text{ and for } i \neq i_j \end{array} \right\} \rightarrow a = b$$

An interpretation I satisfies the assertion $(\mathbf{fd} \ R_{i_1}, \dots, i_h \rightarrow j)$ if for all $t, s \in R^I$, t and s are the tuples; we have that $t[i_1] = s[i_1], \dots, t[i_h] = s[i_h]$ implies $t[j] = s[j]$

An interpretation that satisfies all the assertions in a KB K is called a model of K. C is satisfiable w.r.t KB K if there is a model I of K such that $C^I \neq \emptyset$.

2.2.2 $ALCQI$

$ALCQI$ [12] is another fragment of description logic which rises from DLR_{ifd} with restricted binary relation, with no inclusion assertion on relation, no identification constraints and no functional dependency. This limitations are due to not having arity three.

The simplest form of concept and relation for $ALCQI$ are as follows:

$$C ::= A | \neg C | C_1 \sqcap C_2 | (\leq kR.C)$$

$$R ::= P | P^-$$

where A and P are atomic concept and atomic role respectively.

The following abbreviation are made from above

$ALCQI$ KB has only inclusion assertion of the form $C_1 \sqsubseteq C_2$; No $R_1 \sqsubseteq R_2$. Semantics are same as for DLR

The semantics of Inverse Atomic Role

\perp	for	$A \sqcap \neg A$
\top	for	$\neg \perp$
$C_1 \sqcup C_2$	for	$\neg(\neg C_1 \sqcap \neg C_2)$
$C_1 \Rightarrow C_2$	for	$\neg C_1 \sqcup C_2$
$(\geq kR.C)$	for	$\neg(\leq k-1R.C)$
$\exists R.C$	for	$(\geq 1R.C)$
$\forall R.C$	for	$\neg \exists R.\neg C$

TABLE 2.3: \mathcal{ALCQI} abbreviation

$$(P^-)^I = \{(a, a') \in \Delta^I \times \Delta^I \mid (a', a) \in P^I\}$$

The semantics for Qualified number restriction in \mathcal{ALCQI} is as follows-

$$(\leq kR.C)^I = \{a \in \Delta^I \mid \#\{a' \in \Delta^I \mid (a, a') \in R^I \wedge a' \in C^I\} \leq k\}$$

Since we are not considering Inverse role at the moment, we restrict ourself to the Language \mathcal{ALCN} , a fragment of DL where \mathcal{ALC} is augmented with number restriction (\mathcal{N}). Next we have basic DL, i.e \mathcal{ALC} language and it's syntax and semantics.

2.2.3 \mathcal{ALC}

The Description Logic \mathcal{ALC} [7] is the minimum language to support most of the ontological construct for semantic web. Table 2.4 gives the \mathcal{ALC} constructs and corresponding meaning-

Name	Syntax	Semantics
top concept	\top	Δ^I
bottom concept	\perp	\emptyset
concept	C	$C^I \subset \Delta^I$
concept negation	$\neg C$	$\Delta^I \setminus C^I$
concept conjunction	$C_1 \sqcap C_2$	$C_1^I \cap C_2^I$
concept disjunction	$C_1 \sqcup C_2$	$C_1^I \cup C_2^I$
existential restriction	$\exists R.C$	$\{x \in \Delta^I \mid \exists y \in \Delta^I ((x, y) \in R^I \wedge y \in C^I)\}$
universal restriction	$\forall R.C$	$\{x \in \Delta^I \mid \forall y \in \Delta^I ((x, y) \in R^I \rightarrow y \in C^I)\}$

TABLE 2.4: \mathcal{ALC} Syntax and Semantics

Inspired by the result in [13], it was found in [3] that the DL \mathcal{ALC}^- is obtained from \mathcal{ALC} by dropping intersection and building complex concept constructs using \mathcal{ALC} , i.e.,

$$C ::= A \mid \neg A \mid A_1 \sqcup A_2 \mid \exists P.A \mid \forall P.A$$

where A is the atomic concept and P is the atomic role.

It was proved in [3], *Concept satisfiability w.r.t an \mathcal{ALC} KB can be linearly reduced to Atomic concept satisfiability w.r.t a primitive \mathcal{ALC} KB.*

An atomic concept A_0 is satisfied w.r.t a primitive \mathcal{ALC} KB if and only if A_0 is satisfiable w.r.t the (primitive) \mathcal{ALC}^- KB K' obtained as above.

The \mathcal{ALC}^- KB allows primitive assertion of the form $A \sqsubseteq C$, where C is the \mathcal{ALC}^- concept.

Till now we have worked on Syntax and Semantics of different DL Language. In the next chapter we will see how these constructs are represented in the form of diagram.

Chapter 3

Logic in UML Class Diagram

Reasoning on UML Class diagram in terms of particular logic named Description logic was introduced in [3]. Their work showed that DLs are very promising technology for implementing core reasoning engines for next generation CASE tools. We are interested in their work for the purpose of working with axioms in DL terminology and their corresponding notation in UML Class diagram.

UML: Unified Modelling Language is the standardized visual specification language for object modeling in the field of Software engineering. It is a graph based language to create abstract model of a system. A class diagram is a type of UML that shows static structure through classes, their properties and the relationship between classes. Reasoning on UML diagram for a large scale system is always required by software designers. We are especially interested in reasoning on UML Class diagram. Before we discuss the technique to follow for reasoning on UML Class Diagram, first look at the construction of UML Class diagram and elements in terms of a particular formal logic belonging to Description logic which is subset of First Order Logic that are used for representing objects of the system.

3.1 Different elements of UML corresponding to FOL

Class: Class in UML represents a collection of objects with common features. Each class has its name which is unique in the domain of interest. Attribute part of the class gives properties of the class and the operations involved in classes. An attribute within a class associates each instance of the class C a set of instances of type T . An attribute a of type T for class C corresponds to binary predicate and the FOL notation is $\forall x, y. (C(x) \wedge a(x, y)) \supset T(y)$

Association: Association in UML is the relation between two or more classes. An association between two or more classes in UML correspond to relation between instances of two or more classes in logical perspective.

The FOL notation corresponding to an UML class diagram and the semantics of each construct in UML class diagrams in terms of first order logic (FOL) is given in table 3.1

Again the First Order logic (FOL) formalization corresponding to properties of UML class diagram is given in table 3.2.

FOL theorem prover shows undecidability. A fragment of FOL that is decidable and has reasoning power is considered as Description Logic. It has mainly three parts

1. DL Language
2. Knowledge Base
3. Automated Reasoning Task





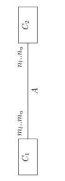
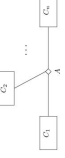


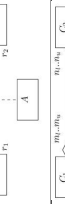

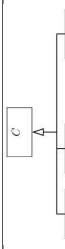

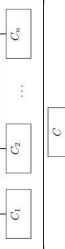
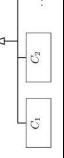
UML	FOL	Comments
	$\forall x, y (C(x) \wedge a(x, y) \supset T(y))$	a is treated like a binary relation which has an element of Class C and the element of class T:Type
	$\forall x, C(x) \supset (i \leq \#\{y a(x, y)\} \leq j)$	Multiplicity within attribute; the number of times a associates to each instance of C ; here it associates at least i and atmost j times.
	$\forall x, p_1, p_2, \dots, p_m, r, f(x, p_1, \dots, p_m, r) \supset \bigwedge_{i=1}^m P_i(p_i)$ $\forall x, p_1, p_2, \dots, p_m, r, r_1, f(x, p_1, \dots, p_m, r) \wedge f(x, p_1, \dots, p_m, r_1) \supset r = r_1$ $\forall x, p_1, p_2, \dots, p_m, r, C(x) \wedge f(x, p_1, \dots, p_m, r) \supset R(r)$	Correct typing of parameter and it depends on the name of the operation; P_i is the name of atomic roles The return value is unique for the given operation on a given object with parameters Correct typing of the result depending on class (and parameters) to which the operation is applied
	$\forall x_1, x_2, A(x_1, x_2) \supset C_1(x_1) \wedge C_2(x_2)$	The association is unique and A is the name of the association.
	$\forall x, C_1(x) \supset (m_1 \leq \#\{y A(x, y)\} \leq n_u)$ $\forall y, C_2(y) \supset (m_2 \leq \#\{x A(x, y)\} \leq m_u)$	Instance of C_1 can participate at least n_l times and at most n_u times. Instance of C_2 can participate at least m_l times and at most m_u times to relation A
	$\forall x_1, \dots, x_n, A(x_1, \dots, x_n) \supset C_1(x_1) \wedge C_2(x_2) \dots C_n(x_n)$	Association between multiple classes.
	$\forall x, y, A(x) \wedge r_i(x, y) \supset C_i(y)$ where $i = 1, \dots, n$ $\forall x, A(x) \supset \exists y, r_i(x, y); i = 1, \dots, n$ $\forall x, y, y_1, A(x) \wedge r_i(x, y) \wedge r_i(x, y_1) \supset y = y_1$ $(\forall y_1, \dots, y_n, x, x_1, A(x) \wedge A(x_1) \wedge \bigwedge_{i=1}^n (r_i(x, y_i) \wedge r_i(x_1, y_i))) \supset x = x_1$	Types the association A, r_i is the role for each class in the association. For each component of A, there is at least one relation r_i At most one element of r_i that is related to component of A. No two instances of A that represent same tuple.
	$\forall y_1, C_1(y_1) \supset n_l \leq \#\{x A(x) \wedge r_1(x, y_1)\} \leq n_u$ $\forall y_2, C_2(y_2) \supset m_l \leq \#\{x A(x) \wedge r_2(x, y_2)\} \leq m_u$	r_1 and r_2 are role names of C_1 and C_2 respectively
	$\forall x, y, G(x, y) \supset C_1(x) \wedge C_2(y)$	Each instance of C_1 contains a set of instances of C_2
	$\forall x, A(x) \supset B(x)$	for all instances x of A, x is also instance of B
	$\forall x, C_i(x) \supset C(x)$ where $i = 1, \dots, n$	Several generalization group together to form Hierarchy
	$\forall x, C_i(x) \supset \bigwedge_{j=i+1}^n \neg C_j(x); i = 1, \dots, n - 1$	This is kind of negative information
	$\forall x, C(x) \supset \bigvee_{i=1}^n C_i(x)$	Each instance of C is an instance of at least one of C_1, \dots, C_n
	$\forall x, (C_1(x) \wedge C_2(x)) \supset C_{12}(x)$	instance of class C_{12} belongs to both C_1 and C_2 .

TABLE 3.1: FOL notation corresponding to UML diagram

Properties of class diagram	FOL formalization
Consistency of UML diagram	Checking FOL asrtrion admits a model in which at least one class has an non-empty extension.
Class Subsumption	In FOL, in every model the extension of C_1 is a superset of extension of C_2 also it can be said that the properties of C_1 also holds for C_2 .
Class Equivalence	Same set of instances of two classes.
Refinement of Properties	Exploiting the implicit multiplicities or typing between classes and associations to enhance readability of the diagram
Implicit Consequence	The property is logically implied by FOL assertions. The property holds in every model of the assertions.

TABLE 3.2: FOL corresponding to properties of diagram

The following part shows the encoding of UML class diagrams in DLR_{ifd} . We discuss construct in UML and the corresponding DLR_{ifd} notation in table 3.3

UML	DLR_{ifd} notation
	$C \sqsubseteq \forall[1](a \Rightarrow (2 : T))$
	$C \sqsubseteq (\geq i[1]a) \sqcap (\leq j[1]a)$
	<p>f_{p_1, \dots, p_m} of arity $1+m+1$; first argument is Concept name, m are the number of parameters for the function and last one is the result.</p> <ul style="list-style-type: none"> • Correct Type of parameter : $f_{p_1, \dots, p_m} \sqsubseteq (2 : P_1) \sqcap \dots \sqcap (m+1 : P_m)$ • Invocation of Operation on given object on given parameters gives unique result : (fd $f_{p_1, \dots, p_m} 1, \dots, m+1 \rightarrow m+2$) • Assertion for correct type of the result: $C \sqsubseteq \forall[1](f_{p_1, \dots, p_m} \Rightarrow (m+2 : R))$
	$A \sqsubseteq (1 : C_1) \sqcap (2 : C_2) \sqcap \dots \sqcap (n : C_n)$
	$A \sqsubseteq \exists[1]r_1 \sqcap (\leq [1]r_1) \sqcap (\forall[1](r_1 \Rightarrow 2 : C_1)) \sqcap$ \vdots $\exists[1]r_n \sqcap (\leq [1]r_n) \sqcap (\forall[1](r_n \Rightarrow 2 : C_n))$ (id $A[1]r_1, \dots, [1]r_n$ which specifies that each instance of A represents a distinct tuple in $C_1 \times C_2 \dots \times C_n$
	$C_1 \sqsubseteq (\geq n_l[1]A) \sqcap (\leq n_u[1]A)$ $C_2 \sqsubseteq (\geq m_l[2]A) \sqcap (\leq m_u[2]A)$
	$C_1 \sqsubseteq (\geq n_l[2](r_1 \sqcap (1 : A))) \sqcap (\leq n_u[2](r_1 \sqcap (1 : A)))$ $C_2 \sqsubseteq (\geq m_l[1](r_2 \sqcap (1 : A))) \sqcap (\leq m_u[1](r_2 \sqcap (1 : A)))$
	$G \sqsubseteq (1 : C_1) \sqcap (2 : C_2)$ $C_1 \sqsubseteq (\geq n_l[1]G) \sqcap (\leq n_u[1]G)$ $C_2 \sqsubseteq (\geq m_l[2]G) \sqcap (\leq m_u[2]G)$
	<p>Generalization : $C_i \sqsubseteq C$ for $i \in \{1, \dots, n\}$</p> <p>Disjointness : $C_i \sqsubseteq_{j=i+1}^n \neg C_j$</p> <p>Covering : $C \sqsubseteq \bigsqcup_{j=1}^n C_j$</p>

TABLE 3.3: UML and corresponding $\mathcal{DLR}_{\{f\}}$

Next we discuss the encoding of UML class diagram directly in \mathcal{ALCQI} . Although it does not preserve models, but it is sound and complete with respect to the main reasoning tasks on UML class diagrams. In table 3.5, it is shown.

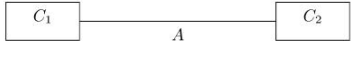
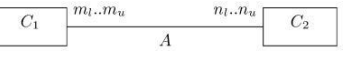
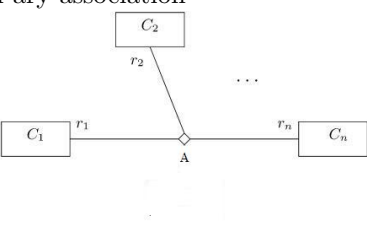
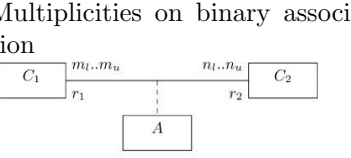
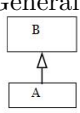
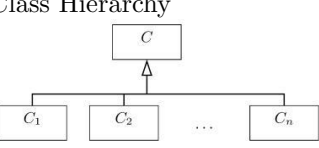
UML digram	\mathcal{ALCQI} encoding
Class C, Classes are atomic, Class A has attribute a of type T	$C \sqsubseteq \forall a.T$
Multiplicity [i..j] of attribute a	$C \sqsubseteq (\geq i a.T) \sqcap (\leq j a.T)$
Multiplicity [1..1] of attribute a	$C \sqsubseteq (\exists a.T) \sqcap (\leq 1 a.T)$
Operation $f() : R$	$C \sqsubseteq \forall R_{f()}.R \wedge (\leq R_{f()}.T) ; [R_{f()} \text{ is binary}]$
Operation $f(p_1, \dots, p_m) : R$	<ul style="list-style-type: none"> • $C_{f(p_1, \dots, p_m)} \sqsubseteq \exists r_1.T \sqcap (\leq 1 r_1.T) \sqcap$ <li style="text-align: center;">• \vdots <li style="text-align: center;">• $\sqsubseteq \exists r_{m+1}.T \sqcap (\leq 1 r_{m+1}.T)$ • $C_{f(p_1, \dots, p_m)} \sqsubseteq \forall r_2.P_1 \sqcap \dots \sqcap \forall r_{m+1}.P_m$ • $C \sqsubseteq \forall r_1^-.C_{f(p_1, \dots, p_m)} \Rightarrow \forall r_{m+2}.R$
	Atomic role A for association class $T \sqsubseteq \forall A.C_2 \sqcap \forall A^-.C_1$
	$C_1 \sqsubseteq (\geq n_l A.T) \sqcap (\leq n_u A.T)$ $C_2 \sqsubseteq (\geq m_l A^-.T) \sqcap (\leq m_u A^-.T)$
n-ary association 	Reification is used $A \sqsubseteq \exists r_1.C_1 \sqcap \dots \sqcap \exists r_n.C_n \sqcap (\leq 1 r_1) \sqcap \dots \sqcap (\leq 1 r_n)$
Multiplicities on binary association 	$C_1 \sqsubseteq (\geq n_l r_1^-.A) \sqcap (\leq n_u r_1^-.A)$ $C_2 \sqsubseteq (\geq m_l r_2^-.A) \sqcap (\leq m_u r_2^-.A)$
Generalization 	$A \sqsubseteq B$
Class Hierarchy 	Class Hierarchy: $C_1 \sqsubseteq C, \dots, C_n \sqsubseteq C$ Disjointness : $C_i \sqsubseteq \prod_{j=i+1}^n \neg C_j$ Covering : $C \sqsubseteq \bigsqcup_{i=1}^n C_i$

TABLE 3.4: \mathcal{ALCQI} notation corresponding to UML diagram

The following describes the action needed in \mathcal{ALCQI} language in accordance to UML diagrams

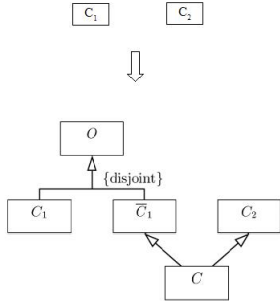
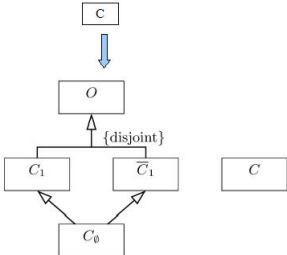
UML	Reasoning Task	Reduction to
	Find $C_2 \sqsubseteq C_1$	checking whether C is consistent
	Find if C is consistent	check $C \sqsubseteq C_0$

TABLE 3.5: Reasoning Task in *ALCQI* for UML diagram

3.1.1 \mathcal{ALC} and UML class diagram

We now resort to less expressive DL, namely \mathcal{ALC} for which tools for reasoning on UML class diagram exists. This is a simpler DL, obtained from \mathcal{ALCQI} by dropping inverse roles and restricting qualified number restrictions to existential restrictions only.

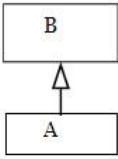
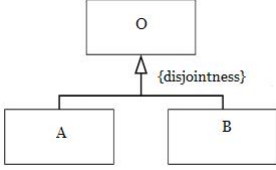
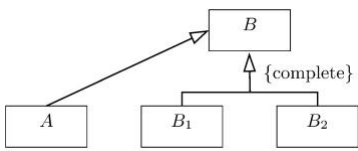
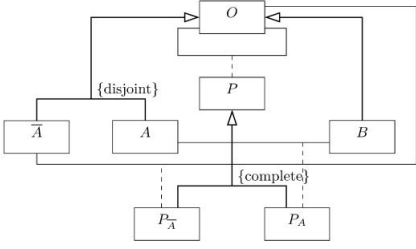
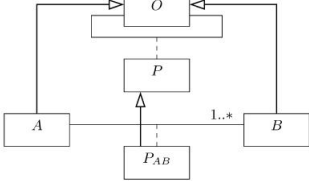
\mathcal{ALC} KB K'	UML encoding
atomic concept A	a class A in D
atomic role P	association P between two classes.
$A \sqsubseteq B$	
$A \sqsubseteq \neg B$	
$A \sqsubseteq B_1 \sqcup B_2$ $\rightarrow A \sqsubseteq B; B \sqsubseteq B_1 \sqcup B_2$	
$A \sqsubseteq \forall P.B$	
$A \sqsubseteq \exists P.B$	

TABLE 3.6: \mathcal{ALC} KB and corresponding UML diagram

The FOL captures UML, but due to intrinsic undecidability of FOL, it is not suitable for automated reasoning. Then comes Description Logic. The DL with n-ary relation, named \mathcal{DLR} captures association with arity n. It is extended to \mathcal{DLR}_{ifd} which captures identification constraints and functional dependency, add more reasoning power. But current state-of the art DL based reasoning does not support all constructs of \mathcal{DLR}_{ifd} . So we limit ourself to \mathcal{ALCQI} which is sufficient to capture major elements of class diagram with nice resoning power and complexity EXPTIME upper bound. Next we have different representation of axioms in DL Knowledge Base motivated by the work [3].

3.2 Proposed Representation

The diagrammatic representation of different construct in [3] and our proposed one are similar, except in representing Covering constraint, Number Restriction and Disjointness. We used a concrete representation in UML for Covering Constraint. We used a big dot for intermediate notation of Covering and from which it points to the concept so it makes more sense. We used it similar to one representation in ORM [Object Role Modeling](#). The graphviz and ORM representation is shown below:

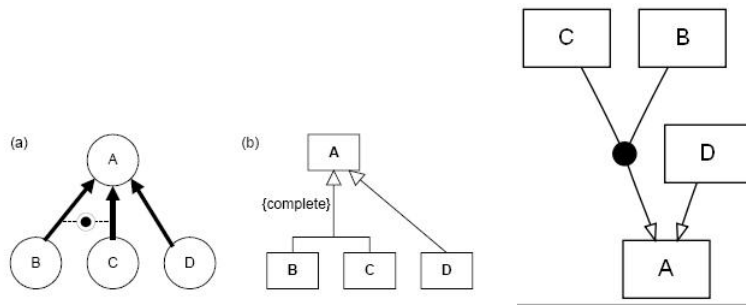


TABLE 3.7: Proposed Covering by Graphviz

Now we give the DL constructs and their corresponding Diagram in the following table-

DL Constructs	Diagram	Comments
Concept Name		
Property Name		
Hierarchy		
Covering (Complete)		


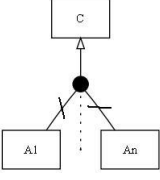
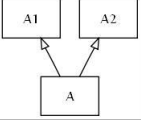
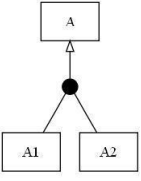

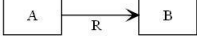
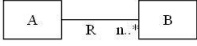

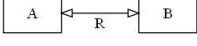
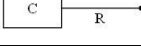
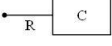
Disjoint of Concepts		
Disjoint and Complete		
Conjunction		
Disjunction		
Existential Restriction		It is part of Qualified Number Restriction which defines at least one value the given property has to have
Universal Restriction		Universal restriction restricts a range of a class to the extension of certain class or datatype
At-least Restriction		
At-most Restriction		
Equivalence		
Domain Constraint		
Range Constraint		

TABLE 3.8: Proposed diagram for DL elements

We have presented the elements of DL variant of OWL. Our model directly corresponds to the language primitives available in \mathcal{ALC} .

Chapter 4

Normal Form & Absorption

Concept definition in DL KB is expressed either as *primitive* or *defined*. A *primitive* definition is of the form $A \sqsubseteq C$ which lacks sufficient defining characteristics. On the other hand *defined* one have sufficient defining characteristics to logically define all subtype concepts. The axiom with this form $A \doteq C$ belong to this category. It indicates a fully defined, non-primitive concept definition. There are several reasons to have a normal form. One of them is to have a standard distribution so that it is easily undersatndable to user to know the content to expect. Secondly by giving a name for each format, it makes clear and unambiguous sense to everyone while communicating each other.

4.1 Normal Form

Normal form $nf(C)$ of a concept C is the set of atoms appearing in a definition. The \mathcal{ALCN} language can be used to describe knowledge about individuals and relationships. Before that we need some basic notion of Atomic Concept and Concept Description.

$A = C$ and $A \sqsubseteq C$ are terminological axioms. The form $A = C$ is called complete definition while $A \sqsubseteq C$ is called primitive definition. Here A is atomic concept or concept name and C is Concept Description.

The concepts (complex) have to be rephrased to the following \mathcal{ALCN} disjunctive normal form [14]. In order to describe the disjuncts obtained by \mathcal{ALCN} normal form, some notations is needed to access the different parts of a concept description C .

- a. $prim(C)$ denotes the set of all (negated) concept names and the bottom concept occuring on the top level conjunction of C .
- b. $val_r(C) := C'$, if there exists a value restriction of the form $\forall r : C'$ on the top-level conjunction of C , otherwise, $val_r(C) := \top$,

- c. $ex_r(C) := \{\exists r.C' \mid \text{there exists } \exists r.C' \text{ on the top-level conjunction of } C\}$.
- d. $min_r(C) := max_k \mid C \sqsubseteq (\geq k r)$ (Note that $min_r(C)$ is always finite.).
- e. $max_r(C) := min_k \mid C \sqsubseteq (\leq k r)$; if there exists no k with $C \sqsubseteq (\leq k r)$, then $max_r(C) := \infty$.

Now \mathcal{ALCN} concept description C is in \mathcal{ALCN} normal form iff $C = \top$, $C = \perp$ or $C = C_1 \sqcup \dots \sqcup C_n$ and each $C_i (i = 1, \dots, n)$ is of the form:

$C_i =$

$$\prod_{A \in prim(C_i)} A \sqcap \prod_{r \in N_R} \left(\prod_{C' \in ex_r(C_i)} \exists r.C' \sqcap \forall r.val_r(C_i) \sqcap (\geq min_r(C_i)r) \sqcap (\leq max_r(C_i)r) \right) \quad (4.1)$$

for all $i = 1, \dots, n$, where the concept descriptions $val_r(C_i)$ and C' again are in \mathcal{ALCN} -normal form and C_i is removed from the disjunction in case $C_i \equiv \perp$.

4.1.1 \mathcal{ALN} Normal Form

Language: \mathcal{ALN} :

Syntax of \mathcal{ALN} Concept: $(C \sqcap D, \forall R.C, \top, \perp, \neg A, \exists R.\top, \geq nR, \leq nR)$

Normal form for \mathcal{ALN} [14]

$$C \equiv L_1 \sqcap \dots \sqcap L_m \sqcap \forall R_1.C_1 \sqcap \dots \sqcap \forall R_n.C_n \text{ or } C \equiv \perp,$$

L_i atomic concepts, their negation, or $\geq nR, \leq nR$

C_i in normal form, R_i, A_i distinct.

Computing Normal Form for \mathcal{ALN} :

More expressive language can be obtained by adding more constructs to \mathcal{AL} . When Number restrictions such as $(\leq nR, \geq nR)$ are added to \mathcal{AL} , the \mathcal{AL} extension language is called \mathcal{ALN} . Here n is non negative integer. $(\leq nR)$ is called at-most restriction while $(\geq nR)$ is called at-least restriction. \mathcal{ALN} concept is of the form: $(C \sqcap D, \forall R.C, \top, \perp, \neg A, \exists R.\top, \geq nR, \leq nR)$ Normal form for \mathcal{ALN} concept can be computed if C has the form $C \equiv L_1 \sqcap \dots \sqcap L_m \sqcap \forall R_1.C_1 \sqcap \dots \sqcap \forall R_n.C_n$ or $C \equiv \perp$,

Algorithm for \mathcal{ALN} Normal form:

Look at outermost connective

1. $\neg, \top, \perp, \geq nR, \leq nR, \exists R.\top$: return concept
2. $\forall R.C$: $C' =$ recurse on C ; return $\forall R.C'$.
3. $C \sqcap D$: recurse on C, D , generating C', D' .

4. If top level of $C' \sqcap D'$ includes conflict $(A, \neg A; \perp; \geq nR, \leq mR (n < m); \geq nR, \forall R. \perp)$, return \perp .
5. Return $C' \sqcap D'$.

4.1.2 \mathcal{ALC} Normal Form

Syntax for \mathcal{ALC} language:

Concepts expressions in \mathcal{ALC} are built up from a set C of atomic concepts and a set R of atomic roles according to the following recursive definition:

$$C ::= \top | \perp | A | \neg C | C \sqcap C | C \sqcup C | \forall R. C | \exists R. C$$

where $A \in C$ and $R \in \mathcal{R}$.

Normal form for \mathcal{ALC} :

Let C be an \mathcal{ALC} concept description. The disjunction and full negation of concept description are dealt in \mathcal{ALC} . Non atomic concepts are avoided by Negation Normal Form. Additionally, the bottom concept have to be represented uniquely and every disjunction on every role level have to be in disjunctive normal form. C is in \mathcal{ALC} -normal form [15], iff $C = \perp$, $C = \top$, or C is of the form $C = C_1 \sqcup \dots \sqcup C_n$ where each $C_i =$

$$\prod_{A \in \text{prim}(C_i)} A \sqcap \prod_{C' \in \text{ex}_r(C_i)} \exists r. C' \sqcap \forall r. \mathbf{val}_r(C_i) \quad (4.2)$$

$\forall i$ where (1) $C_i \not\equiv \perp$ for all i and (2) $\mathbf{val}_r(C_i)$ and every concept in $\text{ex}_r(C_i)$ again are in \mathcal{ALC} -normal form.

Let's follow a simple example from [15]. Let A , B and C be atomic concept names. Let $C := \neg \forall r. (A \sqcap B) \sqcap (B \sqcup \exists r. \neg A)$. The negation normal form of C yields $\exists r. (\neg A \sqcup \neg B) \sqcap (B \sqcup \exists r. \neg A)$. By distributing conjuncts over the disjunctions we obtain $(\exists r. (\neg A \sqcup \neg B) \sqcap B) \sqcup (\exists r. (\neg A \sqcup \neg B) \sqcap \exists r. \neg A)$.

Algorithm to find \mathcal{ALC} Normal Form:

The Algorithm to find \mathcal{ALC} Normal form is similar to the one for \mathcal{ALN} except the Number restriction is not considered. Additionally Disjunction and negation of full concept are considered. Negation of full concept description are handled with Negation Normal form which means that the negation only appears in front of Concept Name in the expression. We will not explain the algorithm for converting Negation Normal Form. Our main concern is to get the Normal form for \mathcal{ALC} concept description and here is the Algorithm:

Let C be an \mathcal{ALC} concept expression. For every \mathcal{ALC} concept expression there is an equivalent Normal form. So find it in recursive manner-

1. If $C \equiv \top$, return \top . If $C \equiv \perp$: return \perp .

2. If $\neg C$, then $C' =$ recurse on C. Find NNF of C' , i.e. negation appears only in front of concept name.
3. If any of C_i in C contains \top return \top .
4. for each disjunct C_i in C which is in conjunctive form the rules are followed:

Look at the outermost connective

- a. \top, \perp : return concept
- a. If C_i is a single Concept description and not an atomic concept then C' : recurse on C_i ; return C'
- b. If C_i has conjuncts of Concept description then $C'_i =$ recurse on each conjuncts of C_i to find the normal form.
 - i. if there is any conflicts $(A, \neg A; \perp, \forall R.\perp)$ return \perp
 - ii. return C'_i
- c. $\forall R.C_{ic}$: $C' =$ recurse on C_{ic} , return $\forall R.C'$. The same holds for $\exists R.C_{ic}$

recurse function replace the concept on the right hand side of definition by the concept it stands for. The recursive process of dependency-elimination substitutions (known as *unfolding*) is done until to cycle in the definition exists although it increases the size exponentially compared to the original size.

Here is an example of having a normal form of concept expression of DL terminological axiom. This is taken from [7].

$$Male \sqsubseteq \neg Female$$

$$Woman \sqsubseteq Human \sqcap Female$$

$$Man \sqsubseteq Human \sqcap Male$$

The above axioms can be replaced by

$$Male = \neg Female \sqcap Male^*$$

$$Male^* \sqsubseteq \top$$

$$Woman = Human \sqcap Female \sqcap Woman^*$$

$$Woman^* \sqsubseteq \top$$

$$Man = Human \sqcap Male \sqcap Man^*$$

$$Man^* \sqsubseteq \top$$

by introducing new concepts $Male^*$, Man^* , $Woman^*$ respectively.

All primitive concepts are replaced by complete definition and only primitive concept of the form $A \sqsubseteq \top$ are left. Since such an axiom does not restrict the extension of A in any way, primitive definition need no longer be taken into account. To get rid of complete definition defined concept are iteratively replaced by their defining description until all names occurring on the right hand side of a complete definition are primitive.

In the above example the definition for Man is $Man = Human \sqcap Male \sqcap Man^*$.

Here the concepts on the right hand side are not primitive. So Male is replaced by the complete

definition of Male and the following definition we get

$$Man = Human \sqcap \neg Female \sqcap Male^* \sqcap Man^*$$

Most DL are fragment of first order logic. The major inference problem for DL is subsumption. For that the algorithm transforms each concept description to a certain normal form; the structure of the normal forms are then compared to decide concept subsumption -

$$C \sqsubseteq D \text{ iff } nf(C) \subseteq nf(D)$$

where nf is the normal form.

Early DL uses structural subsumption algorithm. We can restrict our attention to descriptions containing conjunction and value restrictions

$$A_1 \sqcap \dots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \dots \sqcap \forall R_n.C_n$$

A_1, \dots, A_m are atomic concept and $\forall R_i.C_i$ are distinct concept description.

Assume C is in normal form $A_1 \sqcap \dots \sqcap A_m \sqcap \forall R_1.C_1 \sqcap \dots \sqcap \forall R_n.C_n$

and D is in normal form $B_1 \sqcap \dots \sqcap B_k \sqcap \forall S_1.D_1 \sqcap \dots \sqcap \forall S_l.D_l$

Then $C \sqsubseteq D$ iff

1. for all i , $1 \leq i \leq k$ there exists j , $1 \leq j \leq m$ such that $B_i = A_j$
2. for all i , $1 \leq i \leq l$ there exists j , $1 \leq j \leq n$ such that $S_i = R_j$ and C_j is subsumed by D_i

The rules are used to decompose the problem until it is reduced to axiomatic primitive subsumption relationships in the terminology T.

4.1.3 Suitable Normal Form

From the above discussion it can be concluded that to have \mathcal{ALCN} Normal form for our task is enough as it has \mathcal{ALC} + Qualified Number restriction. But we have to make sure that this Normal form should be in Conjunctive Normal form for concept expression on right side of inclusion axiom. Also the right side of the axiom would be in Disjunctive Normal form or in case it can be a single Atomic concept Name, which one appears first.

T-Box with primitive definition can be transformed into one where all primitive definitions have \top on the right side of the definition. For conjunctive concept, by extracting and caching the primitive components of all concepts, it is possible to check subsumption by comparing the primitive components. C can be subsumed by D if the set of primitive components of C is a subset of the primitive components of D.

4.2 Absorption

Complexity of DL based KB system reasoning is usually due to the reasoning with respect to a T-Box. The performance degrades when non-determinism happens during the expansion in Tableau algorithm. Some optimization techniques are followed to avoid this non-determinism. Before we proceed to more details, we would like to introduce few terms that might be necessary for the discussion.

Introduction axiom: The axioms in DL terminology often restricted to terms that introduce new concept names, roles or attribute names and associate them with expression either through an equality or a subsumption relationship. These kinds of axioms are called **introduction axioms**.

Primitive: Names which are associated with an expression via a subsumption (\sqsubseteq) are known as primitive definition. Example- The axiom $human \sqsubseteq animal \sqcap biped$ tells that the primitive concept human is necessarily both an animal and biped. So an instance of human(x) refers that x is animal and biped, but the conjunction of animal(x) and biped(x) is not sufficient to infer human(x).

Non-Primitive: Names which are associated with an expression via an equality relation ($=$) are known as non-primitive definition. It fully defines the characteristics. It provides both necessary and sufficient condition. As for example, consider axiom $woman \doteq human \sqcap female$. Here non primitive concept says that a woman is necessarily both a human and a female and also the conjunction of $human(x)$ and $female(x)$ is sufficient to infer $woman(x)$. It is also called definition.

Atomic Primitive: The Primitive expression of DL KB which has atomic concept, atomic role name on right side and nothing is known about those atomic concept or role name other than the fact that their interpretation are subsets of Δ^I for concept and $\Delta^I \times \Delta^I$ for role names. The axiom $vegan \sqsubseteq \forall eats.plant$ is an atomic primitives since concept plant and role eats are atomic primitives. Nothing can be said about them.

4.2.1 Unfolding

Simplifying and restricting a terminology so that it is unfoldable. *Unfolding* a concept means that it only contains primitive concept names. Subsumption between unfolded concept expressions then are evaluated independently of the terminology. There are certain conditions by which a terminology is unfoldable [9]:

1. All concept axioms are introduction axiom of the form

$$CN \sqsubseteq C \mid CN = C$$

No GCIs or concept equation in T.

2. All concept introduction axioms are unique. For each concept name CN there is at most one axiom in T of the form $CN \sqsubseteq C \mid CN = C$

3. All introduction axiom are acyclic

- An introduction axiom $CN \sqsubseteq C$ or $CN \doteq C$ directly uses a concept name CN_1 if CN_1 appears in C ($CN = \exists R.CN_1$ directly uses CN_1)
- an Introduction axiom uses a concept name CN_1 if it directly uses CN_1 or it directly uses a concept name CN_2 and CN_2 uses CN_1 ; ($CN = \exists R.CN_1$; or $CN = \exists R.CN_2, CN_2 = \exists R.CN_1$)
- An introduction axiom $CN \sqsubseteq C$ or $CN = C$ is acyclic unless it uses CN (.e.g., C does not contain CN)

4.2.2 Lazy Unfolding

Tableaux algorithm generally assumes the concept expression to be tested is fully unfolded. But in practice it is unfolded when required. So it waits until the time comes for unfolding for certain concept expression [16]. This technique is known as *Lazy Unfolding*. For example testing the satisfiability of $\exists R.CN$, CN is a concept name. Unfolding of CN is delayed until \exists rule creates an R successor y with $L(y) = \{CN\}$

Example: Testing satisfiability of $\exists R.CN \sqcap \forall R.\neg CN$. As soon as it creates R successor y $\{CN, \neg CN\} \subseteq L(y)$, then the contradiction is detected. This could save a lot of time if the concept expression is large and complex one.

Elimination of Primitive Concept Introduction axiom: Primitive Concept Introduction axioms can be eliminated from T by replacing each axioms $CN \sqsubseteq C \in T$ with an equivalent non-primitive introduction axiom $CN \doteq CN' \sqcap C$; CN' is a unique, new atomic primitive concept. $CN^I = (CN' \sqcap C)^I = CN'^I \sqcap C^I \subseteq C^I$

Example (Unfolding): Given the terminology T containing the introduction axioms of the form

$$T = \left\{ \begin{array}{l} human \sqsubseteq animal \sqcap biped, \\ woman = human \sqcap female. \end{array} \right\}$$

The axiom $human \sqsubseteq animal \sqcap biped$ is replaced with $human = animal \sqcap biped \sqcap human'$

The *woman* concept then can be unfolded with respect to T by substituting *woman* with its definition $human \sqcap female$ and then substituting *human* with $human' \sqcap animal \sqcap biped$.

4.2.3 GCI Absorption

General Concept Inclusion (GCI) axioms are too costly to reason with due to high degree of non determinism that they introduce. So the reasoning process is optimized by eliminating GCI from Knowledge Base whenever possible.

One of the most efficient technique for T-Box reasoning is Absorption. The technique reduces the non determinism during tableau expansion by removing GCI (General Concept Inclusion) axioms from T-Box T . Instead of using GCI during reasoning, the rule axioms are introduced to

solve the problem. But not all axioms are absorbable, therefore there are two parts for a T-Box, T_u and T_g , one is unfoldable part that contains rule axioms with concept names on left hand side, while T_g contains the rest of T. $T = T_u \cup T_g$.

For diagrammatic representation of T-Box axioms we follow the general absorption mechanism. As it was already proved that pre-absorption and post-absorption T-Box are semantically equivalent, we take the notation for the diagram which is suitable. A discussion about the absorption mechanism and the corresponding representation for the diagram is main concern now.

Case 1: T_{ab} is an absorption of T-Boxes T :

$$T = T_u \cup T_g \text{ and } T_u = \emptyset ; T_g = \{A \sqsubseteq C; \neg A \sqsubseteq D\};$$

$$T_{ab} = T'_u \cup T'_g \text{ and } T'_u = \{A \Rightarrow C; \neg A \Rightarrow D\}; T'_g = \emptyset$$

Here A is an atomic concept and C,D are arbitrary concepts and T_{ab} is valid absorption.

Case 2: T_{ab} is an absorption of TBoxes T :

$$T = T_u \cup T_g \text{ be a TBox, and } T_u = \emptyset ; T_g = \{A \equiv D\}, A \in NC;$$

$$T_{ab} = T'_u \cup T'_g \text{ and } T'_u = \{A \Rightarrow D; \neg A \Rightarrow \neg D\}; T'_g = \emptyset$$

T_{ag} is valid absorption of T.

Case 3: Let T be a TBox containing axioms entirely in the form of $A \sqsubseteq D$ where $A \in NC$. If A has already a rule definition in T_u , say $A \Rightarrow C$, and $\neg A$ has no rule definition in T_u , then $\{(T_u \setminus A \Rightarrow C) \cup (A \Rightarrow (D \sqcap C)), T_g\}$ is a valid absorption of $T \cup T'$.

Case 4: Let T be a TBox containing axioms entirely in the form of $A \sqsubseteq C$ where $A \in NC$. If $\neg A$ has already a rule definition in T_u , say $\neg A \Rightarrow D$, and A has no rule definition in T_u , then $\{(T_u \cup \{A \Rightarrow C\}), T_g \cup \{C \sqcup D\}\}$ is a valid absorption of $T \cup T'$.

Then the notation for the absorbed concepts

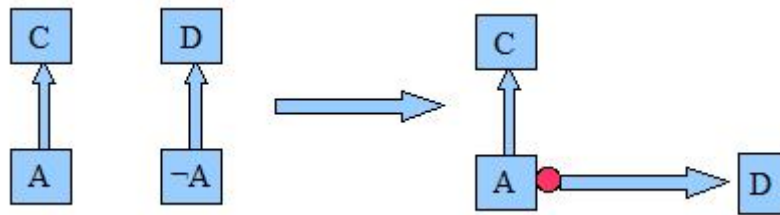


FIGURE 4.1: GCI Absorption

Case 5: Let (T_u, T_g) be a valid absorption of a TBox T, if T' is a TBox that consists of axiom of the format $A \sqsubseteq E$, where $A \in NC$. If A already has a rule definition in T_u , suppose $A \Rightarrow C$, and $\neg A$ also has a rule definition in T_u , say $\neg A \Rightarrow D$, then $\{(T_u \setminus \{A \Rightarrow C\}) \cup \{A \Rightarrow (C \sqcap E)\}, (T_g \setminus \{A \sqsubseteq E\}) \cup \{T \sqsubseteq D \sqcup E\}\}$ is a valid absorption of $T \cup T'$. There are some other cases to be considered.

Example: Here is an example of absorbing GCI taken from [9].

geometric-figure $\sqcap \exists$ **angles.three** $\sqsubseteq \exists$ **sides.three**

It can be absorbed into the primitive concept introduction axiom:

$$\mathbf{geometric-figure} \sqsubseteq \mathbf{shape}$$

transform the GCI using using identity $CN \sqcap C \sqsubseteq D = CN \sqsubseteq D \sqcup \neg C$

$$\mathbf{geometric-figure} \sqsubseteq \exists \mathbf{sides.three} \sqcup \neg \exists \mathbf{angles.three}$$

Absorb this into the introduction axiom

$$\mathbf{geometric-figure} \sqsubseteq \mathbf{shape} \sqcap (\exists \mathbf{sides.three} \sqcup \neg \exists \mathbf{angles.three})$$

The idea of our thesis is to have a representation of the axiom which may be unfolded so that either the left side is a primitive concept name or conjunctive concept expression one of whose terms is a primitive concept name.

When the antecedent of GCI is primitive concept name, the GCI simply states additional condition for the primitive concept and can be absorbed into primitive introduction axiom-

$$CN \sqsubseteq C \text{ and } CN \sqsubseteq D = CN \sqsubseteq C \sqcap D$$

The validity of which is obvious from semantics $CN^I \subseteq C^I \wedge CN^I \subseteq D^I \iff CN^I \subseteq C^I \cap D^I$
When antecedent of GCI is a conjunctive concept expression one of whose term is a primitive concept, it can be transformed into GCI with a primitive concept name as its antecedent using identity.

$$CN \sqcap C \sqsubseteq D \text{ absorbs into } CN \sqsubseteq D \sqcup \neg C$$

The validity of which is obvious from semantics

$$CN^I \cap C^I \subseteq D^I \iff CN^I \subseteq D^I \cup (\neg C)^I$$

Consequent is a negated primitive concept name or a disjunction expression-one of whose elements is a negated primitive concept name.

$$C \sqsubseteq \neg CN \sqcup D = CN \sqsubseteq D \sqcup \neg C$$

4.2.4 Dealing with Domain and Range Restriction:

Domain: *Domain* is the set of all values that an independent variable of a function can have. In Description logic it means the set of individuals, called concept or class that can have values from. *Domain* and *Range* restriction can be applied on OWL ontology. This ontology language supports domain and range constraint on roles.

Restricting the **Domain** of a role R to C is equivalent to GCI $\exists R.\top \sqsubseteq C$ which means the concepts whose instances are all related to some other individuals by role R is subsumed by C .

Range: Restricting the **Range** of a role R to concept D is equivalent to saying $\top \sqsubseteq \forall R.D$. In practice this kind of GCIs are not amenable for absorption. So a different optimisation technique has been followed in [17] which shows this kind of GCIs can be rewritten so that they can be easily handled. The extended version is called *role absorption* [18] that transforms GCI of the above format or more complex format can be absorbed into domain and range constraint.

An axiom of the form $\exists R.C \sqsubseteq D$

Rewritten as $\exists R.\top \sqsubseteq D \sqcup \neg \exists R.C$.

This is absorbed into domain constraint $Domain(R, D \sqcup \neg \exists R.C)$.

Other type of axiom like $D \sqsubseteq \forall R.C$ can be absorbed into

Domain Constraint $Domain(R, \neg D \sqcup \neg \exists R.\neg C)$.

In RacerPro¹, it is possible to transform a domain restriction of the form $\exists_{\geq 1}R \sqsubseteq C$ into an equivalent inclusion axiom of the form $\neg C \sqsubseteq \exists_{\leq 0}R$. If there are no axiom of the form $C \sqsubseteq \dots$ exists, then the axioms of the form $\neg C \sqsubseteq \dots$ can be absorbed [17]. Racer supports absorption of GCIs of the form $\neg A \sqsubseteq C_1$ (but only if no concept introduction axiom of the form $A \sqsubseteq C_2$ and no concept definition axiom of the form $A \doteq C_2$ for A exists).

The value restriction $\forall R.C$ is interpreted as the set of all individuals in the domain whose R -successors (if any) all belong to the interpretation of C . The limited existential restriction $\exists R.\top$ is interpreted as the set of all individuals in the domain that have at least one R -successor.

Axioms of the form $\exists R.C \sqsubseteq D$ can be represented in the above form which is part of absorption technique for handling this type of axiom.

Alternative Technique:

In[19], axiom of the form $\exists R.C_1 \sqsubseteq C_2$ is equivalent to $C_1 \sqsubseteq \forall R^-.A$, $A \sqsubseteq C_2$ where A is a new introduced concept name. Also axiom of the form $\exists R^-.C_1 \sqsubseteq C_2$ is equivalent to $C_1 \sqsubseteq \forall R.A$, $A \sqsubseteq C_2$.

The corresponding UML would be

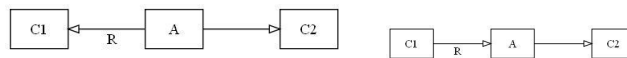


TABLE 4.1: Handling Inverse Role

The UML does not capture the actual meaning of the axiom $\exists R.C_1 \sqsubseteq C_2$ and $\exists R^-.C_1 \sqsubseteq C_2$. So the transformed one captures the actual meaning in UML which is given in above diagram.

¹<http://www.racer-systems.com/>

Axiom of the form $\top \sqsubseteq C \sqcup \forall R.D$ is handled in [19] and similar technique presented in [20] shows that the axiom is equivalent to $\top \sqsubseteq D \sqcup \forall R^-.C$ where R^- is an inverse relation of R . The direct absorption of this axiom is $\neg D \sqsubseteq \forall R^-.C$ that is followed in [21] instead of removing inverse role. For UML representation we didn't consider the inverse of roles. In [19], the inverse role is handled by transforming inverse role and introducing a new concept name. So $\top \sqsubseteq \forall R^-.C$ is transformed to $\top \sqsubseteq \forall R^-.C \sqcup A'$ from which we get $\neg C \sqsubseteq \forall R.A'$. $\neg A'$ is added to the set.

Role Absorption

The new algorithm transform GCIs into range and domain axioms. Role absorption is important as there are large number of domain and range constraints in the form of GCIs. Protege and OilEd uses range and domain constraint, while FaCT does not use these constraints. They introduce two kinds of role absorption : *basic* and *extended* role absorptions.

Basic role absorption

The simple form of role absorption, which deals with axioms of the form $\exists R.\top \sqsubseteq C$ and $\top \sqsubseteq \forall R.C$ and formalised as

1. An interpretation \mathcal{I} satisfies \mathcal{R} and $\exists R.\top \sqsubseteq C$ iff \mathcal{I} satisfies $R \cup \{Domain(R, C)\}$.
2. An interpretation \mathcal{I} satisfies \mathcal{R} and $\top \sqsubseteq \forall R.C$ iff \mathcal{I} satisfies $R \cup \{Range(R, C)\}$.

Extended Role Absorption

Rewriting techniques similar to those used in concept absorption can be used to extend the basic role absorption technique to deal with a wider range of axioms. An axiom of the form $\exists R.C \sqsubseteq D$ can be absorbed into a domain constraint $Domain(R, D \sqcup \neg \exists R.C)$ by rewriting it as $\exists R.\top \sqsubseteq D \sqcup \neg \exists R.C$. Similarly, an axiom of the form $D \sqsubseteq \forall R.C$ can be absorbed into a domain constraint $Domain(R, \neg D \sqcup \neg \exists R.\neg C)$

The extended tableaux algorithm and role absorption optimisation have been implemented in FaCT++ DL reasoner.

4.2.5 The Absorption Algorithm

Given a TBox T , the absorption algorithm constructs a triple of TBoxes $\langle \mathcal{T}_{def}, \mathcal{T}_{sub}, \mathcal{T}_g \rangle$

- \mathcal{T}_{def} is a set of axioms of the form $A \equiv C$ (equivalent to a pair of axioms $\{A \sqsubseteq C, C \sqsubseteq A\} \subseteq T$), where $A \in C$ (i.e., A is a concept name) and there is most one such axiom for each $A \in C$. Such an axiom is often called a *definition* (of A).
- \mathcal{T}_{sub} is a set of axioms of the form $A \sqsubseteq D$, where $A \in C$ and there is no axiom $A \equiv C$ in \mathcal{T}_{def} .
- \mathcal{T}_g contains all the remaining axioms from T .

A GCI $C \sqsubseteq D$ can be absorbed into a primitive concept introduction axiom, whenever possible.

1. $G = \{D, \neg C\}$ - G contains consequent and negated antecedent.
2. For some negated concept $\neg A \in G$ such that $A \in C$ and there is no axiom of the form $A \equiv C$ in \mathcal{T}_{def} , then add $A \sqsubseteq \sqcup(G \setminus \{\neg A\})$ to \mathcal{T}_{sub} and exit.
3. If there is a concept $\exists R.C \in G$, then add $Domain(R, \sqcup(G))$ to \mathcal{R} and exit.
4. For every $C \in G$ such that C is of the form $(C_1 \sqcup \dots \sqcup C_n)$ change $G = G \cup \{C_1, \dots, C_n\} \setminus \{C\}$
5. For every $A \in G$ (resp. $\neg A \in G$), if there is an axiom $A = C$ in \mathcal{T}_{def} , then substitute $A \in G$ (resp. $\neg A \in G$) with C (resp. $\sim C$).
6. If the above two rules are applied then start from begin again.
7. If there is some $C \in G$ such that C is of the form $(C_1 \sqcap \dots \sqcap C_n)$, then for each C_i try to absorb (recursively) $G \cup \{C_i\} \setminus \{C\}$, and exit. Otherwise, absorption of G has failed; leave G in \mathcal{T}_g , and exit.

Chapter 5

Working with OWL Ontologies

This chapter is divided into 2 parts. Working with OWL ontologies in RacerPro, checking the axioms after transformation (absorption, normalization etc). This is particularly useful for our task as the axioms generated by racer pro after the transformation are of the format $A \sqsubseteq C$, where A is atomic concept name. In the following section we mark some features that we use from RacerPro and the process in detail.

5.1 Racer Pro as Description Logic System

Racer Pro is the system for representing Knowledge Base based on highly descriptive Description Logic. It reasons on multiple T-Box and A-Box and implements DL Language $\mathcal{ALCQHI}_{\mathcal{R}+}$. RacerPro provides some important services like-

- Concept Consistency
- Concept Subsumption
- Finding inconsistent Concept name
- Determine parent and child of concept name w.r.t a T-Box
- Query answering w.r.t A-Box and T-Box of a KB
- Instance checking, Instance retrieval, checking entailment

RacerPro implements most of the functions specified in the older **Knowledge Representation System Specification (KRSS)**. But it mainly implements HTTP based quasi standard DIG¹ for interconnecting DL systems with interfaces and applications using an XML based protocol. Graph based representation of T-Box (parent child relation) between concepts of the ontology in another feature of the RacerPro.

¹DL implementation Group

RacerPro Edition: RacerPro comes with RacerPro Server, Racer Porter, Racer Plus, Racer Master edition. The most common one is RacerPro Server. It is a server for Description Logic or OWL interface services. To validate the system for an organization, the RacerPro server can be used to maximize the utility.

Racer Porter is a graphical user client for Racer Pro. It uses TCP/IP network interface to connect to one or more RacerPro Servers. Several key features of this interface are visualization of T-Box, A-Box, load multiple KBs, switch between different taxonomies etc. Racer Plus is for minimizing the load while network-based communication is involved and RacerMaster is for developing user chosen complex application for Semantic Web.

The RacerPro Knowledge Base: Before going into detail of our work, the syntax and the corresponding semantics in Racer system need to be stated. The racer language follows the previous KRSS like syntax. It covers the syntax of very expressive Description Logic. The following table 5.1 shows DL notations and the corresponding RacerPro syntax.

Name	DL Notation	RacerPro Notation
Negation	$\neg C$	(not C)
Conjunction	$C_1 \sqcap C_2 \sqcap \dots \sqcap C_n$	(and $C_1, C_2, \dots C_n$)
Disjunction	$C_1 \sqcup C_2 \sqcup \dots \sqcup C_n$	(or $C_1, C_2, \dots C_n$)
Exists Restriction	$\exists R.C$	(some R C)
Value Restriction	$\forall R.C$	(all R C)
At-most restriction	$\leq nR$	(at-most n R)
At-least restriction	$\geq nR$	(at-least n R)
Exactly restriction	$= nR$	(exactly n R)
Qualified at-most restriction	$\leq nR.C$	(at-most n R C)
Qualified at-least restriction	$\geq nR.C$	(at-least n R C)
Qualified exactly restriction	$= nR.C$	(exactly n R C)
General Concept Inclusion axiom	$C_1 \sqsubseteq C_2$	(implies $C_1 C_2$)
Concept Equations axiom	$C_1 \doteq C_2$	(equivalent $C_1 C_2$)
Concept disjointness axiom	$C_1 \sqsubseteq \neg(C_2 \sqcup \dots \sqcup C_n)$ $C_2 \sqsubseteq \neg(C_3 \sqcup \dots \sqcup C_n)$ \vdots $C_{n-1} \sqsubseteq \neg C_n$	(disjoint $C_1, \dots C_n$)
Primitive concept axiom	$(CN \sqsubseteq C)$	(define-primitive-concept CN C)
Concept Definition	$(CN \doteq C)$	(define-concept CN C)

TABLE 5.1: Racer Pro notation corresponding to DL notation

RacerPro environment

RacerPro can be used with ontology of different formats. KRSS, XML, RDF are supported. To work with other ontology rather than KRSS and racer format, it does not ensure safeness. For OWL files, the KB processing in RacerPro is unsafe. Some statements such as *save-kb*, *save-tbox*, *save-abox* may cause security problem to RacerPro server. Therefore in order to support these features to work with OWL or other types of KB, the unsafe mode was activated. RacerPro was started with option **-u** (for unsafe mode). The command to start RacerPro in unsafe mode is

```
$ RacerPro -u filename
```

This unsafe mode in Racer Editor allow to display and edit OWL ontologies and evaluate OWL files. The three commands *save-kb*, *save-tbox* and *save-abox* mentioned above are used to save

the KB, T-Box and A-Box in KRSS like syntax or xml or daml format. As Racer support KRSS like syntax which is called Racer syntax, is also possible to generate. In order for the command to work on KB, it is first loaded in RacerPorter and then in the shell prompt the following command is executed:

```
(save-tbox test.racer :syntax :racer :transformed nil)
```

:syntax indicate the syntax of the KB to be gathered. In our case we used *:racer* to generate racer like syntax. *:transformed* indicates the optimization techniques are applied on KB, nil means the transformation is not applied. Instead if we use **(save-tbox test.racer :syntax :racer :transformed t)**, the T-Box generated in this way is the transformed one in which **absorption** is applied as part of optimization.

Lets look at a simple example which is reply from RacerPro technical service in request of an email of how to save KB after absorption. There is no straight forward way to find out which axioms in T-Box are absorbed. A simple solution is to check the axioms in original KB and the transformed one to find out the absorbed axioms.

Here is the example

File name: test.racer (original version)

```
(in-tbox default :size 2 :role-size 4)
(define-primitive-role r)
(implies (and c d) e)
(implies c (some r f))
```

File name: test2.racer (absorbed version)

```
(in-tbox default :size 4 :role-size 4)
(define-primitive-concept e)
(implies d (or e (not c)))
(define-primitive-concept f)
(implies c (some r f))
```

So the axiom $c \sqcap d \sqsubseteq e$ is absorbed into $d \sqsubseteq (e \sqcup \neg c)$

5.1.1 Ontology Collection

Collection of OWL ontologies was the second step along with testing in RacerPro. Most of the OWL ontologies are from Protege Library. Most of the ontologies are well known and built in Protege. The task of working with real life OWL Ontologies need collecting ontologies from different sources. It needs active investigation. The protege OWL library was the source of some real life ontologies which were mainly built in Protege. Google search engine was also used in collecting OWL files from the web. More than 15 ontologies were collected from web initially.

The second task was quite time consuming. Working in Racer Pro environment is easy but need to learn to get the full functionality of the tool. The tool is perfect for reasoning ontology and running query. The conversion of OWL ontology to RacerPro syntax was necessary as to have the absorbed version of the T-Box. After loading the file in Racer Porter, save the file in RacerPro format with extension **.racer**. The next task was to extract T-Box. This will save the T-Box without optimization. During the transformation from OWL to racer syntax, the concept name is prefixed with XML namespace. The name

`http://www.owl-ontologies.com/unnamed.owl#person`

is called qualified name and

`http://www.owl-ontologies.com/unnamed.owl#`

is called XML namespace prefix. This not only applies to concept names but also to individuals and roles.

5.1.2 The Result by RACER

The output by RacerPro is the Absorbed T-Box with axioms. Most of the files with T-Box is smaller in size than the original KB. Some of them are bigger but because of importing other ontology in T-Box. The axioms are picked up randomly for drawing the diagram. The axioms are in the format of Concept definition or inclusion axiom. The left side of axiom contains only concept name (CN). The diagram is drawn in paper as some of the T-Box contains concepts of more than 500. The axioms in thses ontology files are mainly of type:

define-datatype-property
 define-primitive-role
 define-primitive-attribute
 define-concept
 implies
 role-equivalent

The axioms in transformed ontologies are either *definition* of *primitive*-axioms or data-type or IS-A (implies) relation with concept name on left side of the axiom. The construction for right side of most of the axioms comes with **and** operator. It comes in the following form:

$$C \sqsubseteq E_1 \sqcap E_2 \sqcap \dots E_n$$

where each E_1, E_2, \dots, E_n are either in **and**, **or** relation with negation, exists, all, number restriction operator. Most of the axioms are in simple IS-A (implies) relation between the concept names: $C_1 \sqsubseteq C_2$.

5.1.3 Search Target

For concept name there can be axiom either in definiton form (\equiv) or General Concept Inclusion (GCI) axiom form (\sqsubseteq). Also the definiton of a concept name does not have to be unique; several definitons may be given for the same concept. To define a concept in RacerPro, the following is the command- (define-concept Grandmother (and Mother (some hasChild Parent)))

RacerPro supports preprocessing optimization (Normalization+Simplification+ Absorption+.....). The idea is to modify KB so that classification and subsumption testing becomes easier. Optimised DL performs a static analysis of a given terminological axiom. Processing becomes faster. GCIs are transformed into concept definition. Not all GCIs can be absorbed. The (get-meta-constraint) function return the non-absorbed GCI as meta constraint in the form of concept. To get the concept definition from optimized T-Box axioms,(get-concept-definition CN) is used where CN is concept name. It returns the definition compiled by RACER during the absorpton phase. The absorption process in RacerPro is heuristic, so it can transform GCI in other different possible way.

As for example GCI of the form

(implies top (or a b c))

can be absorbed into

(implies (not b) (or a c)) OR

(implies (not a) (or b c)) OR

(implies (not c) (or a b))

The concept name on left side can be *primitive concept* or *defined Concept*. Defined concept represents the equality of concept name and concept term. As for example in IPDFull.racer T-Box (collected OWL from [http://protegewiki.stanford.edu/index.php/ Protege_Ontology_Library](http://protegewiki.stanford.edu/index.php/Protege_Ontology_Library)), total number of concepts 873. Out of that 9 are defined concepts while *equivalent* states the equivalence between two concept terms. Therefore for the diagrammatic view this kind of axiom with single concept name on left side is appropriate.

5.1.4 Bug in RacerPro

The example given in [7], page 327, It is given that for given axioms ($A \sqsubseteq D_1 \in T_u$ and $(A \sqcup \exists R.C \sqsubseteq D_2) \in T_g$), general axiom would be partly absorbed into the definition axiom to give $(A \sqsubseteq (D_1 \sqcap D_2)) \in T_u$, leaving a smaller general axiom $\neg \sqcap (\neg D_2, \exists R.C) \in T_g$. But in practice using RacerPro the following things happen-

(implies a d1)

OKAY

(implies (or a (some r c)) d2)

```

OKAY
(get-meta-constraint)
top
(save-tbox "test.racer" :transformed nil)
OKAY
(save-tbox "test1.racer" :transformed t)
OKAY

```

The contents of two files are-

```

test.racer (original one without absorption)
(in-tbox default :size 2 :role-size 4)
(define-primitive-role r)
(implies a d1)
(implies (or a (some r c)) d2)

```

```

test1.racer (absorbed version)
(in-tbox default :size 4 :role-size 4)
(define-primitive-concept d2)
(define-primitive-concept c)
(define-primitive-concept d1)
(implies a d1)

```

The problem is that the non absorbed axioms can't be found and also d_2 is not partially absorbed.

Later the bug was fixed in the next version of RacerPro and it delivers this T-Box as the result of

```

save-tbox :transformed t:

(in-tbox default :size 4 :role-size 4)
(define-primitive-concept d1)
(implies a d1)
(define-primitive-concept c)
(implies (not d2) (and (all r (not c)) (not a)))

```

We conclude our chapter here and next we discuss our procedure to get our transformed KB after several preprocessing steps and transformation.

Chapter 6

Axiom Transformation for Diagrammatic View

Till now we have worked with different representation and method for processing of axioms in DL Knowledge Base. This Chapter will describe the preprocessing steps and transformation needed for different construct for axioms of T-Box.

6.1 Processing Axioms

In this section, we discuss different types of axioms that might appear in T-Box of a KB. Consider $c \sqcap d \sqsubseteq e$. It can be absorbed into $d \sqsubseteq (e \sqcup \neg c)$. It is not possible to represent the first axiom in UML diagram unless we introduce some new concept which is equivalent to $c \sqcap d$. On the other hand the absorbed version, the next one, can be represented easily in UML model. What we can do in the second case, the axiom is processed to

$$d \sqsubseteq d' \sqcup d'' \sqsubseteq (e \sqcup \neg c)$$

$$d' \sqsubseteq e$$

$$d'' \sqsubseteq \neg c$$

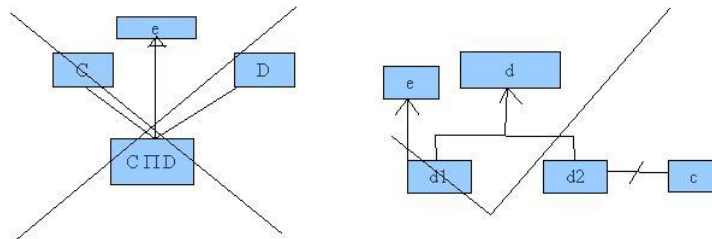


FIGURE 6.1: Axiom Absorption

Lets look other type of axiom where right hand side of the axiom is in Conjunctive form-

$$C \sqsubseteq E_1 \sqcap E_2 \sqcap \dots \sqcap E_n$$

This can be processed as

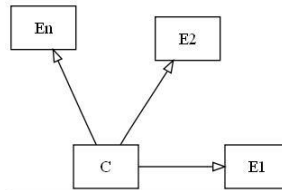
$$C \sqsubseteq E_1$$

$$C \sqsubseteq E_2$$

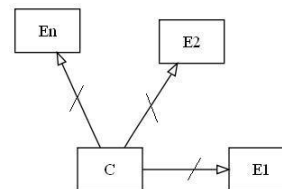
⋮

$$C \sqsubseteq E_n$$

The diagram for that -



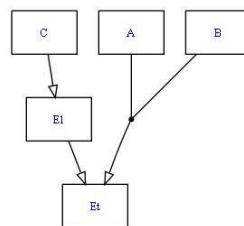
If $E_1, E_2 \dots$ in negated form and they are primitive, then



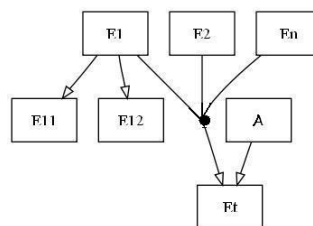
If E_1 has primitive definition of the form $E_1 \sqsubseteq (A \sqcup B)$

In this case $C \sqsubseteq E_1$ and $E_1 \sqsubseteq E_t$ and $E_t = (A \sqcup B)$ where E_t is covering of A and B

We get the following graph



Again let us assume that the axioms are in Negation Normal Form and right side is in Disjunctive Normal Form. If each E_i on the right side is in Disjunctive Form, $A \sqsubseteq E_1, E_2, \dots$, here $E_1 \dots E_n$ are in Conjunctive form. We get the following diagram-

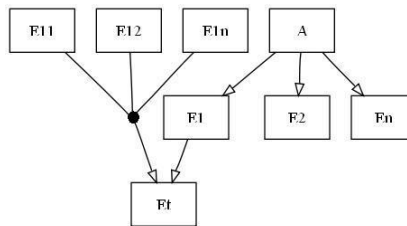


Here E_t is a new concept name which works as a covering of Concept $E_1, E_2 \dots E_n$

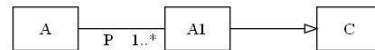
If the right side is in Conjunctive Normal Form, $A \sqsubseteq E_1, E_2, \dots$, here $E_1 \dots E_n$ are in Disjunctive form. We get -

$$\begin{aligned} A &\sqsubseteq E_1 \\ A &\sqsubseteq E_2 \\ &\vdots \\ A &\sqsubseteq E_n \end{aligned}$$

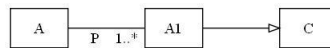
We get the following diagram-



The next thing is about Existential and Universal Restriction. For existential restriction of the form $A \sqsubseteq \exists P.C$ can be replaced by $A \sqsubseteq \exists P.A_1$ and $A_1 \sqsubseteq C$, where C is a complex concept. The diagram is as follows-



The same holds for universal restriction. For universal restriction of the form $A \sqsubseteq \forall P.C$ can be replaced by $A \sqsubseteq \forall P.A_1$ and $A_1 \sqsubseteq C$, where C is a complex concept and A_1 is a new atomic concept name. The diagram is as follows-



Transforming Concept terms on the right hand side of inclusion axiom with only positive and negative literals the concept term need to be unfolded until there are only positive or negative literals remain. The mechanism is straightforward. After unfolding the concept term into literals reveals the most specific construct for the concept on the left side. Complex concept on left side either in conjunctive form or disjunctive form needs introduction of new concept name for the diagram. So we have to transform this type of axiom into absorbed one where a single concept name appears on the left side of inclusion axiom. Example from [3], axiom $A \sqsubseteq C_1 \sqcup C_2$, where C_1 and C_2 are concept terms. This axiom can be represented as $A \sqsubseteq A_1 \sqcup A_2$ and $A_1 \sqsubseteq C_1$, $A_2 \sqsubseteq C_2$, where A_1 and A_2 are new concept names. Even when the right side of inclusion axioms is disjunction of atomic concept names, the UML encoding of axiom of the form $A \sqsubseteq B_1 \sqcup B_2$, where B_1 and B_2 are atomic concept names introduces new concept name B for covering. So

the new axiom would be $A \sqsubseteq B, B \sqsubseteq B_1 \sqcup B_2$.

6.1.1 Other form of Axioms

Axioms of the form $C \sqsubseteq D$ where C is in DNF ($C_{11} \sqcup C_{12} \sqcup \dots \sqcup C_{1n}$) and D is in CNF ($D_{11} \sqcap D_{12} \sqcap \dots \sqcap D_{1n}$). Then from the structural subsumption relationship

$$C \sqsubseteq D \text{ implies } (C_{11} \sqcup C_{12} \sqcup \dots \sqcup C_{1n}) \sqsubseteq (D_{11} \sqcap D_{12} \sqcap \dots \sqcap D_{1n})$$

This implies that

$$C_{11} \sqsubseteq (D_{11} \sqcap D_{12} \sqcap \dots \sqcap D_{1n}) \text{ and}$$

$$C_{12} \sqsubseteq (D_{11} \sqcap D_{12} \sqcap \dots \sqcap D_{1n}) \text{ and}$$

⋮

$$C_{1n} \sqsubseteq (D_{11} \sqcap D_{12} \sqcap \dots \sqcap D_{1n}).$$

Then for each C_{1i} which is in conjunctive form can be absorbed into one of its concept name. If there is a definition for the concept name, the definition part would be added together. Here is a sample example (run on RacerPro).

```
(in-tbox default :size 2 :role-size 4)
(define-concept c1 (and c11 c12 c13)) ⇒ C1 ≡ C11 ∩ C12 ∩ C13
(implies (or (and c11 c12 c13) c2) (and d1 d2)) ⇒ (C11 ∩ C12 ∩ C13) ∪ C2 ⊆ (d1 ∩ d2)
```

then the absorbed one would be

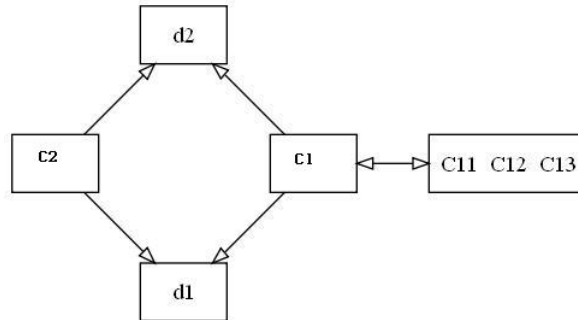
```
(in-tbox default :size 7 :role-size 4)
(define-primitive-concept d2)
(define-primitive-concept d1)
(implies c2 (and d2 d1)) ⇒ C2 ⊆ d2 ∩ d1
(implies c13 (or (and d2 d1) (not c12) (not c11))) ⇒ C13 ⊆ (d2 ∩ d1) ∪ ¬C12 ∪ ¬C11
(define-primitive-concept c12)
(define-primitive-concept c11)
(define-concept c1 (and c13 c12 c11)) ⇒ C1 ⊆ C13 ∩ C12 ∩ C11
```

Here the concept definition \equiv is represented as two way inclusion.

If we consider the definition of $(C_{11} \cap C_{12} \cap C_{13})$, which is C_1 and replace the part with C_1 we get new axiom of the form

$$C_1 \sqcup C_2 \sqsubseteq (d_1 \sqcap d_2)$$

the following diagram is simpler one then previous absorbed one



6.2 Systemize the procedure

It is important now to handle every axiom in ontology efficiently. Specially the axioms in GCI form need to be dealt with care and in a systemetic way so that they can be represented in a optimized way (suitable for the diagram). Till now we have dealt with GCI axioms of different types. But there was no specific rules that were followed . We will try to give some guidance on transformation of certain axioms in KB. We consider only axioms in GCI format because they are difficult to handle but bear importance in hierarchical relation in an ontology. For every axiom we follow 3 steps-

1. Apply Simplification and find the Conjunctive Normal form on left side of axiom.
2. Use Absorption so that we have a primitive definition for every concept name.
3. Apply the transformation (comes shortly) so that we have a representation of our axioms suitable for diagram.

6.2.1 Preprocessing

These are few steps for processing the axioms before our Transformation rules are applied on the axioms found in the KB.

6.2.1.1 Simplification

In the pre-processing step we add another technique named simplification during TBox rewriting so that the contradiction and tautologies are detected. These simplification rules from (a) to (k) are from [10] and the last two (l) amd (m) are added to have more simplified Knowledge Base. The rules for the simplification are as follows:

(a) $(A \sqcap \neg A) \equiv \perp$

- (b) $(A \sqcup \neg A) \equiv \top$
- (c) $A \sqcap (A \sqcup B) \equiv A$
- (d) $A \sqcup (A \sqcap B) \equiv A$
- (e) $A \sqcap (\neg A \sqcup B) \equiv A \sqcap B$
- (f) $A \sqcup (\neg A \sqcap B) \equiv A \sqcup B$
- (g) $(\forall R. A \sqcap \forall R. B) \equiv \forall R. (A \sqcap B)$
- (h) $(\exists R. A \sqcup \exists R. B) \equiv \exists R. (A \sqcup B)$
- (i) $\exists R. \perp \equiv \perp$
- (j) $\forall R. \top \equiv \top$
- (k) $(A \Rightarrow C \text{ and } B \Rightarrow C) \Leftrightarrow ((A \sqcup B) \Rightarrow C)$
- (l) $(A \sqsubseteq \forall R. B \text{ and } A \sqsubseteq \forall R. B') \Rightarrow A \sqsubseteq \forall R. (B \sqcap B')$ [Additional]
- (m) $(A \sqsubseteq \exists R. B \text{ and } A \sqsubseteq \exists R. B') \Rightarrow A \sqsubseteq \exists R. (B \sqcup B')$ [Additional]
- (n) $\exists R. C \sqcap \forall R. D \Rightarrow \exists R. (C \sqcap D) \sqcap \forall R. D$ [15]

Some other simplification rules may be in the form

- i. $\top \sqcup C \equiv C$
- ii. $\top \sqcap C \equiv \top$
- iii. $\perp \sqcup C \equiv C$
- iv. $\perp \sqcap C \equiv \perp$

Some additional rules comes trivially based on *simplification* allows that the unnecessary disjuncts does not appear in concept expression (which is in disjunctive form)

- aa. $A \sqcup (A \sqcup C) \equiv (A \sqcup C)$
- ab. $A \sqcup (A \sqcap C) \equiv A$ [rule (d) of simplification]

6.2.1.2 Normal Form and Absorption

Next we try to manage Normal form of concept expression in the axioms of T-Box. For this, we choose Disjunctive Normal form on left side of the inclusion axiom and Conjunctive normal form on the right side. We already had discussed these issues in previous chapter. We apply suitable algorithm to have the Normal form for our expression.

After successful computation of Normal form, Absorption is applied. We follow standard absorption[22], but other extension of the work are taken into account for processing of axioms. The primary

idea of Absorption is to move axioms from T_g to T_u while keeping the semantics of new T-Box identical. Here we consider some other axioms for which standard absorption might not work.

Negated Absorption

The T-Box in DL might contain axiom of the form $\neg A \sqsubseteq C$. The semantics can be expressed with r to an interpretation. For any interpretation I , it says that $(\neg A)^I \subseteq C^I$. This is equivalent to saying that for some individual x which does not belong to A , belongs to C only when $(\neg C)^I \subseteq A^I$ also holds.

The diagrammatic representation for this axiom can be formulated by introducing a new concept name A' which is in (\equiv) relation with $\neg A$, i.e. $A' \equiv \neg A$. As A and $\neg A$ are disjoint, so are A' and A . Therefore the following diagram can be used for representing $\neg A \sqsubseteq C$.

$$A' \equiv \neg A$$

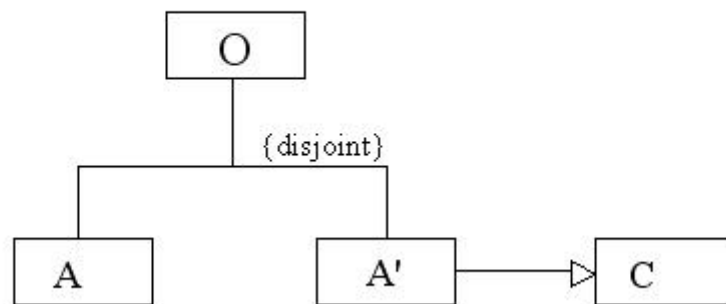


FIGURE 6.2: Absorption of Negated axiom

Domain Absorption

The domain of a concept is represented as $\exists R.T \sqsubseteq C$ in T-Box. The limited existential restriction is interpreted as the set of all individuals in the domain that have at least one R successor and those set of individuals are member of C . To represent a relation between two concept names with the axiom $\forall P.D \sqsubseteq C$ is straightforward. Our target is to handle this kind of axiom similar way that is followed in [19] so that it has primitive definition. The following transformation can be applied:

Main axiom $\forall P.D \sqsubseteq C$

transformed $\top \sqsubseteq C \sqcup \exists P.\neg D$

transformed $\top \sqsubseteq \exists P^-.C \sqcup \neg D$

transformed $D \sqsubseteq \exists P^-.C$

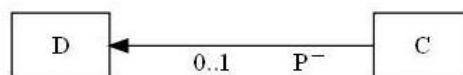


FIGURE 6.3: Domain Absorption through Inverse Role

6.2.2 Transformation Rules

After the preprocessing of axioms, we have the suitable Knowledge Base with primitive definition of each concept name. We now introduce some transformation rules that will be applied on each axiom in DL Knowledge Base.

6.2.2.1 Transformation Rules

The following table 6.1 describes the transformation rules that are used to get the modified KB for our diagram. The constructs are in NNF and only negation symbols appear in front of concept name. The T-Box consists of set of axioms of the form $C \sqsubseteq D$, where C and D are concept terms. C, D are built from concepts using different constructs. Here we consider concept name on the left side of inclusion axiom to have a better understanding. Once we have a successful transformation we can have complex construct on left side. The main objective of our transformation is to get the definition for each concept name in the form of primitive axiom, such as $A \sqsubseteq C$, where A is concept name.

and:	if	$A \sqsubseteq C_1 \sqcap C_2$ and left side has single concept name
	then	$A \sqsubseteq C_1, A \sqsubseteq C_2$
or:	if	$A \sqsubseteq C_1 \sqcup C_2$ and left side has single concept name
	then	$A \sqsubseteq A_1 \sqcup A_2, A_1 \sqsubseteq C_1, A_2 \sqsubseteq C_2$
some:	if	$A \sqsubseteq \exists P.C$ and left side has single concept name
	then	$A \sqsubseteq \exists P.A'$ and $A' \sqsubseteq C$
all :	if	$A \sqsubseteq \forall P.C$ and left side has single concept name
	then	$A \sqsubseteq \forall P.A'$ and $A' \sqsubseteq C$
negation :	if	$A \sqsubseteq \neg C$ and left side has single concept name
	then	$A \sqsubseteq \neg A'$ and $A' \sqsubseteq C$

TABLE 6.1: Transformation Rules for T-Box axioms

6.2.3 Representation of Axioms in Logic Programs

Logic programs is the KR system whose semantics underlies in a large part of rule systems. Prolog is among them for defining the rule system. A set of rules in Prolog having the form

$$H \leftarrow B_1 \wedge \cdots \wedge B_m$$

H is the head of the rule and $B_1 \wedge \cdots \wedge B_m$ is the body. The \leftarrow is to be read as 'if', if m=0, the body is empty.

Our transformation rules for \mathcal{ALC} language can be represented in the form of Prolog rules. In that case we have to be careful that the left side of rule expression has only single atom. The table below shows the transformation rules and corresponding Prolog notation.

TABLE 6.2: DL transformation and corresponding Prolog notation

Name	\mathcal{ALC} Transformation	Prolog Construct
and	$A \sqsubseteq C_1 \sqcap C_2 \Rightarrow A \sqsubseteq C_1, A \sqsubseteq C_2$	$subclassof(A, C_i) \leftarrow subclassofAND(A, and(C_i))$ [$C_i = C_1, C_2 \dots$]
or	$A \sqsubseteq C_1 \sqcup C_2 \Rightarrow A \sqsubseteq A_1 \sqcup A_2,$ $A_1 \sqsubseteq C_1, A_2 \sqsubseteq C_2$	$subclassof(A, or(A_1, A_2)) \leftarrow subclassofOR(A, or(C_1, C_2))$ $subclassof(A_i, C_i) \leftarrow subclassofOR(A, or(C_i))$ [$C_i = C_1, C_2 \dots$] [$A_i = A_1, A_2, \dots$]
exists	$A \sqsubseteq \exists P.C \Rightarrow A \sqsubseteq \exists P.A', A' \sqsubseteq C$	$subclassof(A, exists(P, A')) \leftarrow subclassofEx(A, exists(P, C))$ $subclassof(A', C) \leftarrow subclassofEx(A, exists(P, C))$
all	$A \sqsubseteq \forall P.C \Rightarrow A \sqsubseteq \forall P.A', A' \sqsubseteq C$	$subclassof(A, forall(P, A')) \leftarrow subclassofAll(A, forall(P, C))$ $subclassof(A', C) \leftarrow subclassofAll(A, forall(P, C))$
negation	$A \sqsubseteq \neg C \Rightarrow A \sqsubseteq \neg A', A' \sqsubseteq C$	$subclassof(A, not(A')) \leftarrow subclassofNeg(A, not(C))$ $subclassof(A', C) \leftarrow subclassofNeg(A, not(C))$

6.3 Implementation

This chapter illustrates by example how the system works to process the T-Box axioms to find out suitable form for the diagram. The formal architecture for our system is pictured in the diagram-

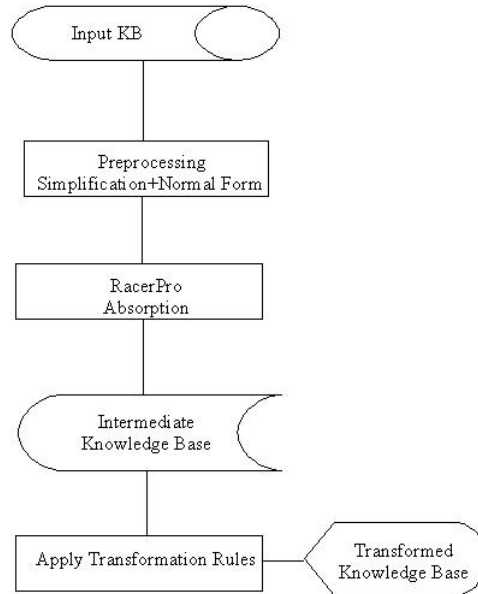


FIGURE 6.4: Architecture of axiom processing with Transformation rules

To check the effectiveness of our procedure let us look at some test KB with standard RACER syntax as well as DL syntax. Test Case 1: We have the sample Knowledge Base with the following T-Box

Racer Syntax	DL Syntax
(implies (and A1 A2) C1)	$A_1 \sqcap A_2 \sqsubseteq C_1$
(implies A3 (some R1 C2))	$A_3 \sqsubseteq \exists R_1.C_2$
(implies (or A4 A5) (or C3 C4))	$A_4 \sqcup A_5 \sqsubseteq C_3 \sqcup C_4$
(equivalent C3 (and A1 A2))	$C_3 \equiv A_1 \sqcap A_2$
(equivalent C4 (all R4 A6))	$C_4 \equiv \forall R_4.A_6$

After the Absorption process applied in RacerPro we got the KB with absorbed axioms as in the table 6.3.

Since RacerPro provides all the preprocessing, we skip the part and apply our transformation rules on the final Absorbed KB which is table 6.4. We unfold all the definition of the concept if there is any.

Here is the sample diagram of the above KB after the transformation.

In the diagram Bx is for B'' and B for B' .

Racer Syntax	DL Syntax
(implies A5 (and C4 C3))	$A_5 \sqsubseteq C_4 \sqcap C_3$
(implies A4 (and C4 C3))	$A_4 \sqsubseteq C_4 \sqcap C_3$
(define-primitive-concept C1)	
(define-primitive-concept A6)	
(implies A2 (or C1 (not A1)))	$A_2 \sqsubseteq C_1 \sqcup \neg A_1$
(define-primitive-concept A1)	
(define-primitive-concept C2)	
(implies A3 (some R1 C2))	$A_3 \sqsubseteq \exists R_1.C_2$
(define-concept C3 (and A2 A1))	$C_3 \equiv A_2 \sqcap A_1$
(define-concept C4 (all R4 A6))	$C_4 \equiv \forall R_4.A_6$

TABLE 6.3: Example of RacerPro Absorbed T-Box

Axioms	Transformation Applied
$A_5 \sqsubseteq C_4 \sqcap C_3$	Unfolding C_4 and C_3 $A_5 \sqsubseteq \forall R_4.A_6 \sqcap A_2 \sqcap A_1$ $\Rightarrow_t A_5 \sqsubseteq \forall R_4.A_6$ $A_5 \sqsubseteq A_2, A_5 \sqsubseteq A_1$
$A_4 \sqsubseteq C_4 \sqcap C_3$	Unfolding C_4 and C_3 $A_4 \sqsubseteq \forall R_4.A_6 \sqcap A_2 \sqcap A_1$ $\Rightarrow_t A_4 \sqsubseteq \forall R_4.A_6$ $A_4 \sqsubseteq A_2, A_4 \sqsubseteq A_1$
primitive-concept C_1	No Action
primitive-concept A_6	No Action
$A_2 \sqsubseteq C_1 \sqcup \neg A_1$	$\Rightarrow_t A_2 \sqsubseteq B_1,$ $B_1 \sqsubseteq C_1 \sqcup B', B' \sqsubseteq \neg A_1$
primitive-concept A_1	No Action
primitive-concept C_2	No Action
$A_3 \sqsubseteq \exists R_1.C_2$	$\Rightarrow_t A_3 \sqsubseteq \exists R_1.B''$ $B'' \sqsubseteq C_2$
$C_3 \equiv A_2 \sqcap A_1$	No Action
$C_4 \equiv \forall R_4.A_6$	No Action

TABLE 6.4: Transformed T-Box

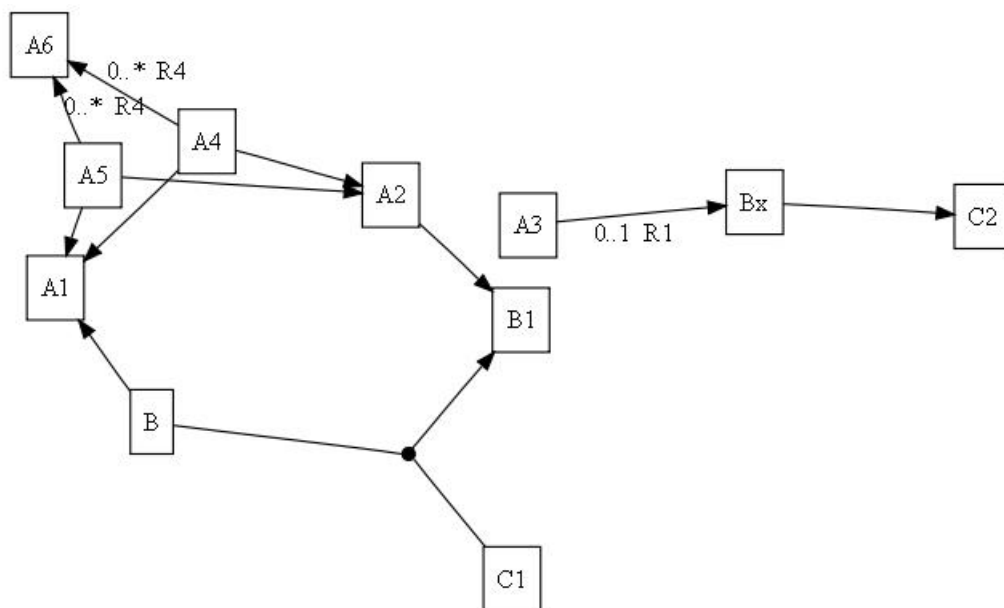


FIGURE 6.5: Diagrammatic view of Transformed Knowledge Base

Chapter 7

Conclusion

Motivated primarily by the task on Reasoning on UML Class Diagram, prospects of reusing the task and generate the UML model from an existing Ontology is dominant in the field of Ontology editing and reasoning. Our method generates an ontology from existing OWL ontology. The transformation rules are meant to be generating a standard form of axioms which might be suitable for UML like diagram. Axioms which may not be converted to the form, suitable for Transformation rules, need further investigation.

Bibliography

- [1] Enrico Franconi and Gary Ng. The icom tool for intelligent conceptual modelling. In *In Proc. of the 7 th International Workshop on Knowledge Representation meets Databases (KRDB'2000)*, pages 45–53, 2000.
- [2] Deborah L. McGuinness and Frank van Harmelen. OWL web ontology language overview. W3C recommendation, W3C, February 2004.
- [3] Daniela Berardi, Diego Calvanese, and Giuseppe De Giacomo. Reasoning on uml class diagrams. *Artif. Intell.*, 168(1):70–118, 2005. ISSN 0004-3702. doi: <http://dx.doi.org/10.1016/j.artint.2005.05.003>. URL <http://fparreiras/papers/ReasoningUMLClassDiagrams2.pdf>.
- [4] Tim B. Lee. *Weaving the Web*. Texere Publishing Ltd., November 1999. URL <http://www.amazon.co.uk/exec/obidos/ASIN/0752820907/citeulike-21>.
- [5] Dan Brickley and R. V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Recommendation. World Wide Web Consortium, February 2004. URL <http://www.w3.org/TR/rdf-schema/>.
- [6] Ora Lassila and Ralph Swick. Resource description framework (rdf) model and syntax specification. Technical report, 1998.
- [7] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The description logic handbook: theory, implementation, and applications*. Cambridge University Press, New York, NY, USA, 2003. ISBN 0521781760. URL <http://portal.acm.org/citation.cfm?id=885746>.
- [8] Manfred Schmidt-Schauß and Gert Smolka. Attributive concept descriptions with unions and complements. *IWBS Report*, 68, 1989.
- [9] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997. URL <download/1997/phd.pdf>.
- [10] Ming Zuo. High performance absorption algorithms for terminological reasoning in description logics, 2006. URL <download/2006/masters-2sss.ps.gz>.
- [11] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Identification constraints and functional dependencies in description logics. In Bernhard Nebel, editor, *IJCAI*, pages 155–160. Morgan Kaufmann, 2001. ISBN 1-55860-777-3.

- [12] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in expressive description logics. In Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, volume II, pages 1581–1634. Elsevier Science Publishers, 2001.
- [13] M. Buchheit, F. M. Donini, and A. Schaerf. Decidable reasoning in terminological knowledge representation systems. *JOURNAL OF ARTIFICIAL INTELLIGENCE RESEARCH*, 1: 109, 1993. URL <http://www.citebase.org/abstract?id=oai:arXiv.org:cs/9312101>.
- [14] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximating \mathcal{ALCN} -concept descriptions. In *Proceedings of the 2002 International Workshop on Description Logics*, 2002.
- [15] S. Brandt, R. Küsters, and A.-Y. Turhan. Approximation and difference in description logics. Number 01-06, Germany, 2001.
- [16] Franz Baader, Bernhard Hollunder, Bernhard Nebel, Hans jurgen Pro Tlich, and Enrico Franconi. An empirical analysis of optimization techniques for terminological representation systems or: Making kris get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:270–281, 1994.
- [17] V. Haarslev and R. Möller. High performance reasoning with very large knowledge bases: A practical case study. In B. Nebel, editor, *Proceedings of Seventeenth International Joint Conference on Artificial Intelligence, IJCAI-01*, pages 161–166, 2001.
- [18] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *In Proc. of the 2004 Description Logic Workshop (DL 2004)*, pages 41–50, 2004.
- [19] Alexander K. Hudek and Grant E. Weddell. Binary absorption in tableaux-based reasoning for description logics. In *Description Logics*, 2006.
- [20] Yu Ding, Volker Haarslev, and Jiewen Wu. A new mapping from alci to alc. In *Description Logics*, 2007.
- [21] Jiewen Wu and Volker Haarslev. Planning of axiom absorption. In Franz Baader, Carsten Lutz, and Boris Motik, editors, *Description Logics*, volume 353 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008. URL <http://dblp.uni-trier.de/db/conf/dlog/dlog2008.html#WuH08>.
- [22] Ian Horrocks and Stephan Tobies. Reasoning with axioms: Theory and practice. In *In Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296. Morgan Kaufmann, 2000.