Dresden University of Technology
Department of Computer Science
Institute for Theoretical Computer Science

Analysis of Pinpointing Algorithms for the Description Logic $\mathcal{ALC}$

Muhammad Tayyab

Master Thesis

Supervisor: Rafael Penaloza
Supervising University Professor: Prof. Franz Baader

Contents

Abstract

Most commonly used Description Logic reasoning systems provide only limited support for debugging logically inconsistent knowledge bases. Attempts have been made to come up with algorithms that solve the debugging problem by suggesting possible resolutions. These algorithms use the method of pinpointing of the axioms in the knowledge bases that are the possible source of inconsistency.

In this thesis, we analyze two of such pinpointing algorithms devised for the Description Logic $\mathcal{ALC}$. These algorithms are extensions of the standard tableau-based reasoning algorithms for the consistency of the knowledge bases represented in $\mathcal{ALC}$. These pinpointing algorithms calculate maximal satisfiable subsets of the original knowledge base by pinpointing minimal sets of the problematic axioms.

We have shown in the thesis that although these algorithms appear to be working quite differently from each other but a close analysis reveals that they use the same idea of axiom pinpointing and execute it in a similar fashion. Both contain similar rules to expand the knowledge and the rule applications have similar results in both of the algorithms. But the two algorithms keep the information about the axioms, that generate a particular knowledge portion, in different ways.

Declaration

I hereby confirm that I have prepared this thesis of my own. No further references or auxiliary means were used but the ones mentioned in this work.

Signature:_____
Muhammad Tayyab

Chapter 1

Introduction

In today's modern world, information and knowledge grow at astonishing steps and its structure and inference problems are getting more complex as we advance to devise more sophisticated ways of knowledge representation. For the last two decades, a quest for viable and better solutions to solve the problems of knowledge representation continues. Initially, we devised relatively simple formalisms each with its own limitations and applicable only in particular scenarios. Hence, the search for more general and efficient formalisms continued.

The most efficient way to deal with knowledge and solve the related problems is logic based. Description Logics (or DLs for short) are a highly successful class of knowledge representation languages with which to represent and reason with ontologies. As the scientific community is focusing more on the description logic OWL being a W3C Recommendation for the Semantic Web, it is expected that the number of ontologies represented in DL-based ontology languages will increase in the future.

The DLs solve problems related to the knowledge bases using logic-based methods. A *knowledge base* contains individual pieces of information about an application domain each of which is called an *axiom*. These axioms express explicit knowledge about the application domain but if taken together, they contain a whole lot of implicit knowledge. The reasoning methods may encounter inconsistencies while trying to find models for a given concept in an application domain using its axioms. Developing a coherent knowledge base is a time consuming and error prone task.

One of the great advantages of a logic-based knowledge representation language is that logical errors in the knowledge bases can be detected automatically. Such errors can have a number of causes. Among the possible sources of these errors can be lack of realization on the part of designers of the knowledge bases that separate pieces of knowledge that seem logically correct when considered individually but are inconsistent when taken together. For example in a simple case, in the knowledge base of a university it can be mentioned that there is a professor aged 30 but there is also a constraint that a professor cannot be less

than 40. These kind of errors may be the result of much complex and many portions of knowledge when held together. The errors can also be introduced deliberately in order to resolve ambiguities [4]. It is helpful while transforming a less formal knowledge representations, such as frame-based systems, to a DL-based representation.

For DL-based ontology languages, optimized DL reasoners like RACER [5], FaCT [7] and FaCT++ [18] are able to detect logical errors, but they offer a very little support and need a great deal of human expertise to rectify these errors.

In most of the work on dealing with such problems, the *concept-satisfiability* problem has attracted much of the interest. The concept-satisfiability problem states that, given a knowledge base for an application domain and a concept, is there a non-empty model of the knowledge base and the concept in the application domain. To explain, we give an example here. We will explain the syntax of the following knowledge base later, here we just need to understand the concept-satisfiability problem of the knowledge bases. Consider the following part of a knowledge base:

$$\begin{aligned} brain &\doteq CentralNervousSystem \\ brain &\doteq BodyPart \\ CentralNervousSystem &\doteq NervousSystem \sqcap \neg BodyPart \end{aligned}$$

According to this, a brain is a body part as well as a central nervous system, while the later is a type of the nervous system but necessarily is not a body part. Although not logically inconsistent, this specification implies that there can be no instances of the concept *brain*. Because it implies that brain is a type of the central nervous system and also of the body part but then it mentions that central nervous system and body part are exclusively different. This indicates a modeling error.

The solutions presented to solve the problem are of two types. First, there are certain algorithms, for example, the work of [8], [9] and [16], which *pinpoint* the sources of the problem by identifying the axioms that make a concept unsatisfiable with regards to a knowledge base. Then modelers are required to rectify the errors in whatever way they like to do that. Thus there is a great need of debugging tools which will facilitate the construction of high-quality knowledge bases and rectify the logical errors by themselves.

The second way of solving the problem also uses pinpointing but it is more proactive. It suggests possible resolutions to the problem by excluding minimal possible knowledge from the knowledge base. Hence, weakening the knowledge to ensure the satisfiability of concepts. Examples of this type of approach include the work of [15], [14] and [11].

In this work, we will analyze a couple of works in this direction which fall into the second category. The first solution is presented in [10] and the second

one in [3]. In the first method, the authors propose a tableau-like algorithm for identifying the maximally satisfiable sub-terminologies of an unfoldable terminology represented in the description logic $\mathcal{ALC}$. Their work is closely related to that of [15] and the algorithm presented in [16] on which it is based. The information about the axioms of the terminology that are used to produce a fact are kept in the form of sets where each element represents an axiom. The tableau applies rules for Boolean constructs and concept definitions and makes the implicit knowledge explicit. While doing so if some conflict arises in the knowledge, they have a special *expansion rule* which tries to solve the conflict by removing sentences that can be a possible cause of the conflict.

The second solution, that we will compare with the first one, is again a tableau-like algorithm. It also works on the $\mathcal{ALC}$-knowledge base. This method is an extension of the tableau-based consistency algorithms for ABoxes described in [10, 7]. It starts with a set of ABox axioms and adds new assertional facts with the help of certain rules until the rules are exhausted. Every ABox axiom is labeled with a unique propositional letter. And the new facts produced during the algorithm are labeled with Boolean formulas from the propositional letters which mention that using which axioms the fact was generated. They look for the maximally satisfiable subsets of the given set of axioms with the help of a Boolean *clash formula* which is generated from the labels of the facts once the rules are exhausted.

The rest of the work is distributed as follows. In chapter 2, the basic notions of the description logic $\mathcal{ALC}$ and, the related terminology and concepts are defined. These concepts are necessary for the work presented in the following chapters. Then in chapter 3, those algorithms are presented which solve the problem of concept-unsatisfiability by calculating maximal satisfiable subsets of the knowledge base. After that, Chapter 4 analyzes how the two tableaux are working to keep track of the axioms related to the generation of a fact in a knowledge base. As the algorithms keep the information about the axioms related to a fact by assigning labels to every fact, so there we talk about how the algorithms work with the labels.

In Chapter 5, we talk about the development of a Boolean *clash formula* in each of the algorithms which is the collective information about the axioms that produce conflicting facts about individual objects in an application domain. We also show that for a given input if the algorithms follow the same *steps* while expanding the knowledge base, they present same information in their respective developing clash formulas at every stage of the tableaux. Which means that they share the same basic idea of solving the problem.

Afterwords, the last chapter holds some conclusions for this work.

Chapter 2

Preliminaries

This chapter lays down the theoretical background for the main objective of the thesis by introducing basic concepts that are used in the later chapters. The formalisms of Description Logics are introduced and the syntax and the semantics of one of the Description Logics called $\mathcal{ALC}$ and its related terminology and concepts are defined. The system debugging techniques for $\mathcal{ALC}$-knowledge base, that we will be analyzing in the later chapters, use this logic to represent knowledge.

2.1 Description Logics

Description Logics [1] are a family of knowledge representation languages which are used to work much efficiently with domains where the implicit knowledge is hidden in the explicitly presented knowledge about a model. They are important formalisms giving logical basis to the already being used well known traditions such as *semantic networks* [13] and *frames* [12] which lacked formal semantics. The basic purpose of these formalisms is to represent knowledge in a much simpler way and to make reasoning more efficient because of their hierarchical configuration.

The main idea of Description Logics is to *describe*, as apparent from the name, the world in terms of properties or constraints that specific individuals have to satisfy. This is basically done by describing relationship between concepts, which are defined portions of knowledge about an application domain, and relationship between concepts and objects of the domain.

The Description Logics are embodied in several knowledge-based systems and are used to develop various real-life applications. They have been used for building a variety of applications including conceptual modeling, information integration, query mechanisms, view maintenance, software management systems, planning systems, configuration systems, and natural language understanding.

There are different types of description logics depending on the definition of concepts and the description of relationships. Each description logic defines a

number of language constructs (such as intersection, union, role quantification, etc.) that can be used to define new concepts and roles. This work focuses on only one of these logics called $\mathcal{ALC}$. Hence, we will define this logic in this section. $\mathcal{ALC}$ may be seen as an instance to understand other Description Logics.

$\mathcal{ALC}$ is the abbreviation for *attribute language with complement* and extends the language $\mathcal{AL}$ as defined in [17]. In this logic, the concepts define formally the notions of the application domain using three basic constructs and a set of binary relations called *roles*. Now we define the syntax, semantics and the related terminology that will be used in the subsequent chapters.

Definition 2.1 (Syntax) *Let N_C and N_R be disjoint sets of* concept names *and* role names*, respectively. The set of $\mathcal{ALC}$-*concept terms *is defined inductively as follows:*

1. *Each concept name $A \in N_C$ is an $\mathcal{ALC}$-concept term.*
2. *$\top$ and $\bot$ are $\mathcal{ALC}$-concept terms.*
3. *if C and D are $\mathcal{ALC}$-concept terms, and $r \in N_R$, then the following are also $\mathcal{ALC}$-concept terms:*
   - $C \sqcup D$, $C \sqcap D$, $\neg C$
   - $\exists r.C$, $\forall r.C$

Example 2.2 *Let* `Person` *and* `Female` *be concept names with the intended meaning of human being and woman, respectively, and* `has-child` *be a role name with the intended meaning of the second element in the binary relation being an immediate descendant of the first one. Then,*

- `Person`$\sqcap\neg$`Female` *describes the notion of "man".*
- $\exists$`has-child.Female` *describes the notion of "parent of a daughter".*

The semantics of $\mathcal{ALC}$ is given by interpretations, which are mappings from concept terms to a specific domain.

Definition 2.3 *An* interpretation $\mathcal{I}$ *consists of a non-empty* interpretation domain $\Delta^{\mathcal{I}}$ *and an interpretation function $\cdot^{\mathcal{I}}$ that*

- *assigns to each $A \in N_C$ a subset $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$,*
- *assigns to each $r \in N_R$ a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$.*

The interpretation function is then inductively extended to the rest of $\mathcal{ALC}$ concept terms as follows:

- $\top^{\mathcal{I}} = \Delta^{\mathcal{I}}$, $\bot^{\mathcal{I}} = \emptyset$

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$
- $(\forall r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} | \forall e \in \Delta^{\mathcal{I}} : (d, e) \in r^{\mathcal{I}} \Rightarrow e \in C^{\mathcal{I}}\}$
- $(\exists r.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} | \exists e \in \Delta^{\mathcal{I}} : (d, e) \in r^{\mathcal{I}} \wedge e \in C^{\mathcal{I}}\}$

In order to improve the readability, it is useful to define new concept terms that represent more complex concept terms in a model. In other words, new concept names are *defined* in terms of relatively complex concept terms. Along with other benefits, it also makes the description of a model compact. The definitions of the concept names are represented in a knowledge base by the so called *terminology box* (TBox).

Definition 2.4 *If $A \in N_C$ and C is a concept term, then $A \doteq C$ is a* concept definition. *A finite set $\mathcal{T}$ of concept definitions is called* acyclic TBox *if following conditions hold:*

1. *There exists no $A \in N_C$ and distinct concept terms C, D such that $\{A \doteq C, A \doteq D\} \subseteq \mathcal{T}$,*
2. *There are no $n \geq 1$ and concept definitions $A_1 \doteq D_1, \cdots, A_n \doteq D_n \in \mathcal{T}$ such that*
   - *D_i contains A_{i+1}, $1 \leq i \leq n$,*
   - *$D_n \doteq A_1$*

The two conditions make sure that in any concept definition of $\mathcal{T}$, right-hand side contains no direct or indirect reference to left-hand side; i.e. there are no cyclic definitions.

Intuitively, an acyclic TBox assigns unique labels called concept names to the concept terms. A concept term can be a description of a very complex concept and can be referred by its concept name in the description of other concepts. Hence, improving the readability. The concept names are required to be unique for distinct concept terms to avoid ambiguity. In the definition of a concept name A_i with a concept term C_i; i.e. $A_i \doteq C_i$, it is made sure that C_i does not contain any concept name A_j whose definition depends on A_i directly or indirectly. In rest of the text, we will use only TBox while referring to an acyclic TBox.

Example 2.5 *A possible TBox $\mathcal{T}$ for the concept names and role name in example 2.2 is given by:*

$$\begin{aligned} \mathcal{T} = \{ \texttt{Man} &\doteq \texttt{Person} \sqcap \neg \texttt{Female}, \\ \texttt{Parent} &\doteq \texttt{Person} \sqcap \exists \texttt{has-child.Person} \} \end{aligned}$$

The concept 'Man' represents someone in an application domain, who is a human and is not a woman. And the concept 'Parent' represents someone who is human and has at least one human child. More complex concepts can be defined using already defined concepts such as Parent and Man. For example, a concept of 'Grand Father' is defined as someone who is a man and has a child who is a parent, as follows:

$$\texttt{GrandFather} \doteq \texttt{Man} \sqcap \exists \texttt{has-child}.\texttt{Parent}$$

The semantics of TBox statements is given as follows:

Definition 2.6 *An interpretation $\mathcal{I}$ is a* model *of a TBox $\mathcal{T}$, if for every concept definition $A \doteq C \in \mathcal{T}$, it holds that $A^{\mathcal{I}} = C^{\mathcal{I}}$.*

$\mathcal{ALC}$-concepts and TBoxes are used to formally describe the knowledge about an application domain. But the knowledge is further expanded by applying reasoning methods to make implicit knowledge explicit. One such reasoning method is the *satisfiability problem.*

Definition 2.7 *Given a TBox $\mathcal{T}$ and a $\mathcal{ALC}$-concept name C, C is* $\mathcal{T}$-satisfiable *if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}} \neq \emptyset$.*
$\mathcal{T}$ is satisfiable iff it is C-satisfiable for every concept name C occurring in $\mathcal{T}$. Any subset $\mathcal{T}'$ of $\mathcal{T}$ is maximally satisfiable subset *if every $\mathcal{T}''$ such that $\mathcal{T}' \subset \mathcal{T}'' \subseteq \mathcal{T}$ is concept-unsatisfiable and $\mathcal{T}'$ is satisfiable.*

The satisfiability problem of an $\mathcal{ALC}$-concept term C with respect to a TBox $\mathcal{T}$ consists of deciding if there is an instance of C that satisfies an interpretation of $\mathcal{T}$.

In addition to the TBoxes, a Description Logic knowledge-base may also contain an *assertion Box (ABox)* which contains facts about individual objects in the application domain. An object is referred by an *individual name.* We can state that an object a belongs to a concept name C by writing $a : C$, where a is an individual name. Every element of an ABox represents a fact about an individual object or a relationship among different objects. Hence, elements of the ABoxes are called *assertional facts* or simply *assertions.* We formally define an ABox as follows:

Definition 2.8 *Let C be a concept term, r be a role name, and a, b be individual names, then $a : C$ is a* concept assertion *and $r(a, b)$ is a* role assertion. *A finite set $\mathcal{A}$ of assertions is called an* ABox.

Example 2.9 *A possible ABox $\mathcal{A}$ for the individual names* `ana`, `bob` *and,* `cole` *and for concept names and role name in example 2.2 is given by:*

$$\mathcal{A} = \{\texttt{ana}:\neg\texttt{Mother},$$

`bob:Man`,
`has-child(bob,cole)`}.

The ABox above mentions the facts that an individual named Bob is a man, a woman named Ana has no children and that Bob is the father of a person named Cole.

The semantics of ABox statements is given as follows:

Definition 2.10 *An interpretation $\mathcal{I}$ is a* model *of an ABox $\mathcal{A}$, if for every concept assertion $a : C \in \mathcal{A}$ it holds that $a^{\mathcal{I}} \in C^{\mathcal{I}}$. And for every role assertion $r(a, b)$ it holds that$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$; where $a^{\mathcal{I}}, b^{\mathcal{I}}$ are mappings of individual names a, b to the application domain under the interpretation $\mathcal{I}$.*

The satisfiability problem for ABoxes is also a field of inquiry. Which states that for any given ABox, is there any interpretation that is the model for all assertional facts of the ABox.

Definition 2.11 *A given ABox $\mathcal{A}$ is* satisfiable *iff there is an interpretation I which is model of $\mathcal{A}$.*

In $\mathcal{ALC}$-concept terms, if the negation appears outside of a complex concept terms, it can be *pushed* inside such that it only appears next to the concept names.

Definition 2.12 *A concept term C is in* negation normal form *(NNF) if for every concept $\neg A$ occurring in C, it holds that $A \in N_C$*

For a given $\mathcal{ALC}$-concept term, there is an equivalent concept term in negation normal form that can be constructed in linear time with respect to the original one.

For simplicity, we will assume that the concept terms mentioned in this text are in negation normal form. Negation normal form of a term can be obtained by following the De Morgen laws and the fact that the following pairs of concept terms are equal: $\neg(\exists r.C)$ and $\forall r.\neg C$ and, $\neg(\forall r.C)$ and $\exists r.\neg C$.

When a group of the axioms is there in a knowledge base every one describing a piece of knowledge about an application domain, it is quite possible that two or more axioms describe or imply conflicting knowledge. Which means that when these axioms are taken together, it is impossible to find a model in the application domain.

One way out of this situation is to get rid of the minimal number of axioms such that rest of them are consistent by finding maximal satisfiable subsets of these axioms. The next chapter presents two tableaux-based algorithms to find maximal subsets of an $\mathcal{ALC}$-knowledge base. Both of the algorithms are extensions from the tableaux-based consistency algorithms for $\mathcal{ALC}$.

Chapter 3

Algorithms

We will introduce and explain, in this chapter, two algorithms that perform the debugging task for $\mathcal{ALC}$ knowledge bases by pinpointing the axioms in the knowledge bases that are potential source of inconsistency. These algorithms pinpoint problematic axioms from an $\mathcal{ALC}$ knowledge base such that rest of the knowledge base is satisfiable. But to make a knowledge base satisfiable, the minimum possible number of axioms are separated from it. In other words, these algorithms calculate maximal satisfiable subsets of the input $\mathcal{ALC}$ knowledge base.

So this chapter is concerned with the following algorithmic problem: *Given a set of axioms $\mathcal{T}$ represented in $\mathcal{ALC}$, find all maximally satisfiable subsets $\mathcal{T}'$ of $\mathcal{T}$.*

Each of the two algorithms is an extension of the tableaux-based consistency algorithms for $\mathcal{ALC}$ knowledge base. Every tableau-based consistency algorithm has a set of expansion rules and on a given knowledge base it applies rules until the rules are exhausted. Application of the rules adds new knowledge to the knowledge base which was implicitly there. Hence, making it explicit. When no rule is applicable, the algorithm decides whether the knowledge base is consistent or not by looking for *obvious contradictions* in the knowledge. An occurrence of two assertional facts of the form $a : A$ and $a : \neg A$ in a knowledge base is called an obvious contradiction. It is a contradiction because an individual name belongs to a concept term and its negation simultaneously. We will also refer to it as a *clash*.

Before we explain the two algorithms that calculate maximal satisfiable subsets of an $\mathcal{ALC}$ knowledge base by extending the tableaux-based $\mathcal{ALC}$ consistency algorithms, it seems a good idea to introduce one of the tableaux-based consistency algorithms as described in [10, 7]. And show how its rules are applied on a knowledge base to decide its satisfiability. The following rules of the tableaux-based consistency algorithm for $\mathcal{ALC}$ ABoxes are applied in any order. In the tableaux-based $\mathcal{ALC}$ consistency algorithms, an assertional fact

is denoted by $A(a)$ which means that the individual a belongs to the concept name A.

Let $\mathcal{M}$ be a finite set of ABoxes, and let $\mathcal{A}_0$ be an element of $\mathcal{M}$. The following rules replace $\mathcal{A}_0$ by an ABox $\mathcal{A}_1$ or by two ABoxes $\mathcal{A}_1$ and $\mathcal{A}_2$.

- *Conjunction rule.* Assume that $(C \sqcap D)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain both assertions $C(a)$ and $D(a)$. The ABox $\mathcal{A}_1$ is obtained by extending $\mathcal{A}_0$ by adding $C(a)$ and $D(a)$.

- *Disjunction rule.* Assume that $(C \sqcup D)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain $C(a)$ or $D(a)$. The ABox $\mathcal{A}_1$ is obtained from $\mathcal{A}_0$ by adding $C(a)$ and the ABox $\mathcal{A}_2$ is obtained from $\mathcal{A}_0$ by adding $D(a)$.

- *Exists-restriction rule.* Assume that $(\exists R.C)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain assertions $R(a, c)$ and $C(c)$ for some individual c. One generates a new individual name b and obtains $\mathcal{A}_1$ from $\mathcal{A}_0$ by adding $R(a, c)$ and $C(b)$.

- *Value-restriction rule.* Assume that $(\forall R.C)(a), R(a, b) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain assertion $C(b)$. The ABox $\mathcal{A}_1$ is obtained from $\mathcal{A}_0$ by adding $C(b)$.

In order to decide if an input ABox $\mathcal{A}$ is consistent, the tableau-based consistency algorithm tries to generate a finite model of $\mathcal{A}$. It starts applying the above mentioned rules on $\mathcal{A}$ and adds new assertional facts until the rules are exhausted and it gives a set of *complete* ABoxes. An ABox where no expansion rule is applicable is called a complete ABox. The algorithm stops with a set of ABoxes because of the presence of the disjunction in our language. The input ABox $\mathcal{A}$ is consistent iff one of the ABoxes produced by the algorithm is consistent. The algorithm decides that an ABox is consistent if it does not contain any clash, that is, no obvious contradiction, e.g. $A(a)$ and $\neg A(a)$ for an individual a and a concept name A. The following facts make clear that the rules of the tableaux-based consistency algorithm provides us with the decision procedure for the consistency of the $\mathcal{ALC}$ ABoxes (for proof see [10, 7]).

Proposition 3.1 1. *If $\mathcal{A}_1$ is obtained from $\mathcal{A}_0$ by application of the conjunction, exists-restriction, or value-restriction rule, then $\mathcal{A}_0$ is consistent iff $\mathcal{A}_1$ is consistent.*
2. *If $\mathcal{A}_1$ and $\mathcal{A}_2$ are obtained from $\mathcal{A}_0$ by application of the disjunction rule, then $\mathcal{A}_0$ is consistent iff $\mathcal{A}_1$ or $\mathcal{A}_2$ is consistent.*
3. *A complete ABox is consistent iff it does not contain any obvious contradiction.*
4. *The rule application process always terminates.*

One of our two algorithms to calculate maximal satisfiable subsets of an $\mathcal{ALC}$ knowledge base extends the above mentioned tableau-based consistency

algorithm for $\mathcal{ALC}$ ABoxes [3]. Because it tries to remove the potential problematic axioms after the rules are exhausted, we call it Post Rule Solution (PRS) algorithm. We explain this algorithm next.

3.1 Post Rules Solution Algorithm

PRS extends the tableaux-based $\mathcal{ALC}$ consistency algorithm by introducing labels to the assertional facts in ABoxes. Every assertional fact in the input ABox $\mathcal{A}$ is labeled with a unique propositional variable. And every new assertional fact added to the knowledge base is labeled by a *monotonic Boolean formula*, that is, propositional formula built from the variables by using conjunction and disjunction only, obtained from the labels of the assertional facts that caused its addition. The label corresponding to an assertion $A(a)$ is denoted by $ind(A(a))$. The labeled expansion rules of the algorithm (given below) are applied in any order until they are exhausted. The following rules of PRS are extension of the rules for $\mathcal{ALC}$ consistency algorithm with labels:

- *Conjunction rule.* Assume that $(C \sqcap D)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain $C(a)$ and $D(a)$ whose indices are both implied by $\text{ind}((C \sqcap D)(a))$. The ABox $\mathcal{A}_1$ is obtained by extending $\mathcal{A}_0$ by $C(a)$ with index $\text{ind}((C \sqcap D)(a))$ and by $D(a)$ with index $\text{ind}((C \sqcap D)(a))$.

- *Disjunction rule.* Assume that $(C \sqcup D)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain $C(a)$ or $D(a)$ whose index is implied by $\text{ind}((C \sqcup D)(a))$. The ABox $\mathcal{A}_1$ is obtained by extending $\mathcal{A}_0$ by $C(a)$ with index $\text{ind}((C \sqcup D)(a))$ and the ABox $\mathcal{A}_2$ is obtained by extending $\mathcal{A}_0$ by $D(a)$ with index $\text{ind}((C \sqcup D)(a))$.

- *Exists-restriction rule.* Assume that $(\exists R.C)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain assertions $R(a, c)$ and $C(c)$ whose indices are both implied by $\text{ind}((\exists R.C)(a))$. One generates a new individual name b and obtains $\mathcal{A}_1$ by extending $\mathcal{A}_0$ by adding $R(a, c)$ and $C(b)$ both with index $\text{ind}((\exists R.C)(a))$.

- *Value-restriction rule.* Assume that $(\forall R.C)(a), R(a, b) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain assertion $C(b)$ whose index is implied by $\text{ind}((\forall R.C)(a)) \wedge \text{ind}(R(a, b))$. The ABox $\mathcal{A}_1$ is obtained by extending $\mathcal{A}_0$ by $C(b)$ with index $\text{ind}((\forall R.C)(a)) \wedge \text{ind}(R(a, b))$.

The algorithm starts applying the rules on the input ABox $\mathcal{A}$. If during the consistency test, n assertions with labels $\psi_1, \cdots, \psi_n$ give rise to a new fact, this fact is labeled with $\psi_1 \wedge \cdots \wedge \psi_n$. An assertional fact may arise in more than one way in an ABox and the distinct labels for the same assertion are joined by disjunctions. For example, there can be an assertion $A(b)$ in an ABox with label ψ_1 and it is also created with label ψ_2, ψ_1 and ψ_2 being distinct, then it is written once with label $\psi_1 \vee \psi_2$.

If no ABox among the complete ABoxes $\mathcal{A}_1, \cdots, \mathcal{A}_m$ is consistent, hence $\mathcal{A}$ is inconsistent, then it is of interest to know which minimum number of facts of

$\mathcal{A}$ can be dispensed of with, making rest of the $\mathcal{A}$ consistent. After the application of the rules, every ABox from the set of ABoxes $\mathcal{A}_1, \cdots, \mathcal{A}_m$ is searched for obvious contradictions if any. The labels occurring in these ABoxes can be used to describe which of the original facts are responsible for the obvious contradictions. The information about the contradictions in the ABoxes is gathered in the form of a so called *clash formula*.

Definition 3.2. *Let $\mathcal{A}_1, \cdots, \mathcal{A}_n$ be the complete ABoxes obtained by applying the labeled consistency algorithm to $\mathcal{A}$. A particular clash $A(a), \neg A(a) \in \mathcal{A}_i$ is expressed by the propositional formula $ind(A(a)) \wedge ind(\neg A(a))$. Now let $\phi_{i,1}, \cdots, \phi_{i,k_i}$ be the formulas expressing all the clashes in $\mathcal{A}_i$. The* clash formula *associated with $\mathcal{A}$ is*

$$\bigwedge_{i=1}^{n} \bigvee_{j=1}^{k_i} \phi_{i,j} \, .$$

Conjunction is used for a single contradiction because both assertional facts are required for the contradiction. Any one contradiction is sufficient to make an ABox inconsistent, thats why disjunction is used to combine the formulas expressing clashes of a single ABox. Now every complete ABox should be inconsistent to make $\mathcal{A}$ inconsistent, thats why formulas corresponding to different ABoxes are combined by conjunction. The next proposition proves that the formula obtained in this way is actually a clash formula for $\mathcal{A}$ (for proof see [3]).

Proposition 3.3 *Let ψ be the clash formula associated with $\mathcal{A}$, $\mathcal{Q} \subseteq \mathcal{A}$, and w be the valuation which replaces the propositional variables corresponding to the $\mathcal{Q}$ by 'true' and the others by 'false'. Then $\mathcal{A}$ is inconsistent iff ψ evaluates to 'true' under w.*

Once the clash formula is built, it helps to calculate the maximal satisfiable subsets of $\mathcal{A}$. Such maximal satisfiable subsets correspond to the *maximal valuations* making the clash formula 'false'. These valuations are maximal with respect to set inclusion, i.e. $\sqsubseteq$. The decision problem of finding the maximal valuations is NP-complete. Moreover, the rules of PRS have an unpleasant property that checking the application condition is NP-hard problem [3]. The size of the clash formula associated with $\mathcal{A}$ can be exponential in the size of $\mathcal{A}$. This means that PRS, in general, needs exponential space as compared to unlabeled consistency algorithm can be realized as a PSPACE-algorithm.

The second algorithm for finding maximal satisfiable subsets of an $\mathcal{ALC}$ knowledge base is presented in [10]. A prominent characteristic of this algorithm is that it tries to exclude the potential problematic axioms during the rule application phase wherever it finds an obvious contradiction. Due to this reason, we call it Run Time Solution (RTS) algorithm. The algorithm is explained next.

3.2 Run Time Solution Algorithm

RTS is a tableaux-based algorithm to find the maximal satisfiable subsets for a given TBox $\mathcal{T}$ such that a given concept term C is satisfiable with respect to the subsets. Basically, the algorithm generates a tableau tree for $\mathcal{T}$ and C much in the same way as tableau-based $\mathcal{ALC}$ consistency algorithms, and with similar expansion rules, but with few changes. Along with every assertional fact, the information about the elements of $\mathcal{T}$ used to generate it is stored. Every axiom of $\mathcal{T}$ is labeled with a unique letter. But it is not done in the similar fashion as in PRS, by a labeling Boolean formula, rather by *index-sets*. Intuitively, an index-set mentions what group of axioms of the TBox were used to generate a particular fact at a node. Another difference is that RTS has an additional non-deterministic clash-breaking rule. When the algorithm finds an obvious contradiction (a clash) at a leaf node, it applies this rule which breaks the clash by excluding axioms involved in the generation of the clashing assertions at the node. Yet another difference is that the rules for treating facts with existential and universal role restriction are combined into one. Hence, a single rule treats an assertional fact with existential role restriction and all the facts with universal role restriction at a particular node of the tableau tree. Due to the merger of these two rules, it is required that this 'combined-rule' is applied only when certain other rules have been already applied.

The algorithm starts applying rules for an input TBox $\mathcal{T}$ and a concept term C for which satisfiable subsets of $\mathcal{T}$ are to be found. The application of rules to a root node r labeled with $(a : C, \emptyset)$, for some individual a and empty set of index-sets, starts developing a tableau tree. An arbitrary node x of the tableau tree is labeled with a set of concept assertions, denoted by $L(x)$. Additionally, every such concept assertion is associated with a set of index-sets $\mathcal{I}$ each containing integers in the range $1, \cdots, n$. More than one index-sets associated to a fact means more than one ways to generate the fact on the node. Every index-set $I_i \in \mathcal{I}$ represents the responsible axioms for the assertion being at the node. So instead of looking like $a : A$ as in a $\mathcal{ALC}$ tableaux-based consistency algorithm, assertions in RTS look like $(a : A, \mathcal{I})$ where a is an individual, A a concept name and $\mathcal{I}$ denotes the sets of axioms involved in the generation of the assertional fact there. Moreover, a node x is also associated with an *exclusion-set*, denoted by $E(x)$, containing the indices of the axioms that were excluded when applying the clash-breaking expansion rule.

The algorithm has rules regarding the concept definitions in the TBox. These rules are applied to replace a concept name by its definition in the TBox. Once a concept name in an assertion is replaced by its definition in the TBox, the assertion is *tagged* so that the rule is not applied for it more than once. Replacing a concept name by its definition is called *lazy unfolding*. Notice that lazy unfolding works only with acyclic TBoxes as for them a situation is attainable where all untagged assertions contain only undefined concepts. It is assumed that all the axioms in the acyclic TBox are of the form $A_i \doteq C_i$ for $i = 1, \cdots, n$; where n is the number of axioms in the TBox.

Definition 3.4 *Let $\mathcal{K}$ be a set of index-sets. A* hitting set *of $\mathcal{K}$ is an index-set $H \subseteq \cup\mathcal{K}$ such that , for every $K \in \mathcal{K}$, $|H \cap K| = 1$.*

The hitting sets are used in the clash breaking rule of RTS. The above definition for a hitting set is not a conventional one. It places an extra restriction of every hitting set containing exactly one element from each index-set of $\mathcal{K}$. This is to suit the clash breaking rule of RTS Where it is required to remove exactly one element from an index-set labeling an assertion. In the following, $MH(\mathcal{K})$ denotes the *minimal hitting set* with respect to set-inclusion, of the set of index-sets $\mathcal{K}$.

The algorithm consists of the following six rules:

- D^+-rule: If $(a : A_i, \mathcal{I})$ is in $L(x)$ and has not been tagged, then Tag $(a : A_i, \mathcal{I})$ and let $L(x) := L(x) \cup \{(a : C_i, \{I \cup \{i\} | I \in \mathcal{I})\}$
- D^- rule: If $(a : \neg A_i, I)$ is in $L(x)$ and has not been tagged, then Tag $(a : \neg A_i, I)$ and let $L(x) := L(x) \cup \{(a : NNF(\neg C_i), \{I \cup \{i\} | I \in \mathcal{I})\}$
- $\sqcap$-rule: If $(a : C \sqcap D, \mathcal{I}) \in L(x)$ then $L(x) := L(x) \backslash \{(a : C \sqcap D, I)\} \cup \{(a : C, \mathcal{I}), (a : D, \mathcal{I})\}$
- $\sqcup$-rule: If $(a : C \sqcup D, \mathcal{I}) \in L(x)$ then create two children y and z of x; $L(y) := L(x) \backslash \{(a : C \sqcup D, \mathcal{I})\} \cup \{(a : C, \mathcal{I})\}$; $E(y) := E(x)$; $L(z) := L(x) \backslash \{(a : C \sqcup D, \mathcal{I})\} \cup \{(a : D, \mathcal{I})\}$; $E(z) := E(x)$
- $\exists$-rule: If $(a : \exists R.C, \mathcal{I}) \in L(x)$ and rules 1-4 can't be applied then $X := \{(b : C, \mathcal{I})\} \cup \{(b : D, \mathcal{K}_D) | (a : \forall R.D, \mathcal{J}) \in L(x)\}$ (b be a new unique individual name) $L(x) := (L(x) \backslash \{(a : \exists R.C, \mathcal{I})\}) \cup X$; with $\mathcal{K}_D = \{I \cup J | I \in \mathcal{I}, J \in \mathcal{J}\}$
- $\bot$-rule: If $(a : A, \mathcal{I}) \in L(x)$ and $(a : \neg A, \mathcal{J}) \in L(x)$ then
For every $K \in (MH(\mathcal{I}) \cup MH(\mathcal{J}))$ do:
Create a new child y of x;
$L(y) := \{(b : D, \mathcal{I}_K) | (b : D, \mathcal{I}') \in L(x)\}$; $E(y) := E(x) \cup K$
EndFor

Note that $\mathcal{I}_K$ is defined as $\{I \in \mathcal{I}' | K \cap I = \emptyset\}$. It denotes all the index-sets of $\mathcal{I}$ that contain no element from the minimal hitting set K. Intuitively, in the process of removing an obvious contradiction at a node by applying the $\bot$-rule, all those index-sets of a fact at the node are to be removed that contain any of the elements from K. This is, because every element of K represents an axiom of $\mathcal{T}$ that is a potential source of the contradiction.

The first two rules perform a version of lazy unfolding. They tag the assertion they are applied to and replace the concept names by their definitions in the TBox adding the respective index in every index-set associated with the assertion. The defined concept names in both of the rules, A_i are retained to

ensure that only maximal satisfiable subsets for A_i are calculated (soundness). The next two rules are very similar to the standard $\sqcap$- and $\sqcup$-rules for $\mathcal{ALC}$. The difference is that a rule is still applicable if the constituents (disjuncts or conjuncts) of the assertion are already there at the node. For example, the $\sqcap$-rule will be applied to $(a : C \sqcap D, \mathcal{I})$, even if $(a : C, \mathcal{J})$ and $(a : D, \mathcal{K})$ are already present at the node. It is necessary to ensure that all the maximal satisfiable subsets of $\mathcal{T}$ for C are calculated (completeness). And $(a : C \sqcap D, \mathcal{I})$ is removed so that the rule is not applied more than once for this assertion. But in case of $(a : C \sqcup D, \mathcal{I})$, the assertion is not required to be removed from the node x because x is no more a leaf node and rules are only applied for assertion at the leaf nodes.

The $\exists$-rule is applied to an assertion of the form $(a : \exists r.C, \mathcal{I})$ and it also treats all the assertions with universal-role restriction at the node. Hence, the $\exists$-rule in RTS is a combination of $\exists$- and $\forall$-rules in a standard $\mathcal{ALC}$ algorithm. Here also the assertion $(a : \exists r.C, \mathcal{I})$, to which the rule is applied, is removed to make sure that the rule is not applied more than once for it. A new individual name b is generated by every application of $\exists$-rule. For every assertion of the form $(a : \forall r.D, \mathcal{J})$ at x, a new assertion $(b : D, \mathcal{K}_D)$ is generated and it is labeled with the set $\mathcal{K}_D$ obtained from the union of every index-set in $\mathcal{I}$ with every index-set in $\mathcal{J}$. This is necessary to ensure that all maximal satisfiable subsets of $\mathcal{T}$ for C are calculated. Note that the rule can only be applied if none of the first four rules is applicable. This is to make sure that when the $\exists$-rule is applied, all assertions of the form $(a : \forall r.D, \mathcal{J})$ are already present and are treated with it and not come after the $\exists$-rule has been applied. For example, consider a situation of a node with assertions $((a : \forall r.D) \sqcap E, \mathcal{J})$ and $(a : \exists r.C, \mathcal{I})$. If the $\exists$-rule is applied before the $\sqcap$-rule, the set X calculated as a part of the rule will not treat $(a : \forall r.D, \mathcal{J})$, as it should. Note also that the restriction for applying a rule is for the $\exists$-rule alone.

The $\bot$-rule is added to break clashes. Whenever a clash is detected at a node x, the $\bot$-rule becomes applicable. The idea is, for a clash $(a : A, \mathcal{I}), (a : \neg A, \mathcal{J})$, to branch by first excluding $(a : A)$ and then excluding $(a : \neg A)$, thereby resolving the clash. To ensure the removal of all the sources of an assertion $a : A$, it is necessary to remove, simultaneously, one of the axioms from every index-set occurring in $\mathcal{I}$. In other words, a hitting set is to be excluded. But as the interest here is to exclude as few axioms as possible, the minimal hitting sets of $\mathcal{I}$ are excluded. The same argument goes for $(a : \neg A, \mathcal{J})$ as well. There is a branch from x for every K occurring in $MH(\mathcal{I})$ or $MH(\mathcal{J})$. So the $\bot$-rule creates a new child y of x for every minimal hitting set K and y is labeled only with concept assertion labeling x but retains only those index-sets which do not contain any axiom from K. If there is any element in $L(y)$ such that all of its index-sets are excluded; e.g. $(a : A, \emptyset)$, then these elements are to be removed from $L(y)$. The exclusion set $E(y)$ is obtained from $E(x)$ by adding K to it.

When no rule is applicable anymore, the tableau stops. From every leaf node n of the tableau, the satisfiable subset of $\mathcal{T}$ is obtained by removing axioms in $E(n)$ from the axioms of $\mathcal{T}$. The maximal sets of all these subsets are maximal

C-satisfiable subsets of the TBox.

From maximal C-satisfiable subsets for every concept C, the maximally satisfiable subsets can be calculated in the following way. Denote by M_j the set of maximal C_j-satisfiable subsets, for $j = 1, \cdots, n$. Then the set of maximally satisfiable subsets are the maximal elements of the set $\{\bigcap_{i=1}^{i\leq n} X_i | X_i \in M_i$ for $i = 1, \cdots, n\}$. The alternative way is, instead of first finding the maximal C-satisfiable subsets for every concept term C, maximally satisfiable subsets can be obtained directly by starting the algorithm with the root node labeled with $\{(a_i : C_j) | i = 1, \cdots, n\}$ where a_is are all distinct individual names and applying the rules as before. The maximal sets of the axioms not excluded in the end are maximally satisfiable subsets of the TBox $\mathcal{T}$. The latter way of the algorithm uses the fact that concept-satisfiability is equivalent to the DL knowledge base $K = (\mathcal{T}, \{(a_i : C_i) | i = 1, \cdots, n\})$ being satisfiable.

Once the RTS tableau stops, it calculates maximal satisfiable subsets of $\mathcal{T}$ in polynomial time on the size of the output of the tableau. But like the rules of PRS, its $\bot$-rule has an unpleasant property, i.e. calculating minimal hitting sets is an NP-hard problem.

While explaining the two algorithms, we saw that PRS finds maximal satisfiable subsets of an ABox and RTS of a TBox. If we have to make a comparative analysis of the algorithms then we have to compromise on the form of axioms in one of the algorithms. Hence, in order to compare the two, we must either translate RTS to an equivalent pinpointing method for ABoxes or PRS should be translated into an equivalent pinpointing technique working for TBoxes. Next section deals with this issue.

3.3 PRS for TBoxes

Given the rules of both the algorithms, a major difference between them is that RTS treats assertions with the universal role restriction at a node while applying the $\exists$-rule for an assertional fact with existential role restriction. But PRS has two different rules each treating one kind of the assertional facts. If we choose to translate RTS into an approach that can work for ABoxes then there may be a problem in a situation where there is no assertional fact with existential role restriction but one with universal role restriction. RTS cannot apply any rule in this case while PRS has value-restriction rule to deal with the situation.

For example, consider a situation where an ABox contains a set of assertional facts $\{r(a, b)$,A(b), $\forall r.\neg A(a)\}$ and there is no existential restriction on the role r, then in PRS value-restriction rule is applicable and the ABox will evolve accordingly leading to a clash. But if a node in RTS tableau contains an equivalent set of assertional facts then the assertional fact with universal role restriction will not get treated as there is no $\exists$-rule applicable. Hence, RTS does not has the ability to deal with all ABox axioms. On the other hand, we can make PRS for TBoxes without disturbing the main idea of the algorithm.

Hence, PRS can treat TBox axioms quite nicely. Thats why in this section we translate PRS for the TBox axioms.

The statements of ABoxes contain Boolean constructs $(\sqcap, \sqcup)$ and role restrictions $(\exists, \forall)$ which are present in the statements of TBoxes as well. But TBoxes consist of definitions of concept names. So PRS has only rules concerning Boolean constructs and role restrictions. Hence, when we translate this approach for TBoxes, we need to add the rules dealing with concept definitions to the previously four rules of PRS.

Now we give the rules of PRS that can be applied to an input consisting of a TBox $\mathcal{T}$ and a concept term C whose satisfiability problem is under consideration with regards to $\mathcal{T}$. The algorithm starts with an ABox $\mathcal{A}$ containing only one assertional fact $C(a)$ for an individual a and labeled with 'true'. The rules of PRS for finding maximal satisfiable subsets of TBoxes are as follows:

- *Axiom$^+$ rule.* Assume that $C(a) \in \mathcal{A}_0$, $C \doteq D \in \mathcal{T}$ and $\mathcal{A}_0$ does not contain $D(a)$ whose index is implied by $\text{ind}(C(a)) \wedge \text{ind}(C \doteq D)$, then we get $\mathcal{A}_1$ from $\mathcal{A}_0$ by adding $D(a)$ labeled with $\text{ind}(C(a)) \wedge \text{ind}(C \doteq D)$.

- *Axiom$^-$ rule.* Assume that $\neg C(a) \in \mathcal{A}_0$, $C \doteq D \in \mathcal{T}$ and $\mathcal{A}_0$ does not contain $\neg D(a)$ whose index is implied by $\text{ind}(\neg C(a)) \wedge \text{ind}(C \doteq D)$, then we get $\mathcal{A}_1$ from $\mathcal{A}_0$ by adding $\neg D(a)$NNF labeled with $\text{ind}(\neg C(a)) \wedge \text{ind}(C \doteq D)$.

- *Conjunction rule.* Assume that $(C \sqcap D)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain $C(a)$ and $D(a)$ whose indices are both implied by $\text{ind}((C \sqcap D)(a))$. The ABox $\mathcal{A}_1$ is obtained by extending $\mathcal{A}_0$ by $C(a)$ with index $\text{ind}((C \sqcap D)(a))$ and by $D(a)$ with index $\text{ind}((C \sqcap D)(a))$.

- *Disjunction rule.* Assume that $(C \sqcup D)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain $C(a)$ or $D(a)$ whose index is implied by $\text{ind}((C \sqcup D)(a))$. The ABox $\mathcal{A}_1$ is obtained by extending $\mathcal{A}_0$ by $C(a)$ with index $\text{ind}((C \sqcup D)(a))$ and the ABox $\mathcal{A}_2$ is obtained by extending $\mathcal{A}_0$ by $D(a)$ with index $\text{ind}((C \sqcup D)(a))$.

- *Exists-restriction rule.* Assume that $(\exists R.C)(a) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain assertions $R(a, c)$ and $C(c)$ whose indices are both implied by $\text{ind}((\exists R.C)(a))$. One generates new individual name b and obtains $\mathcal{A}_1$ by extending $\mathcal{A}_0$ by adding $R(a, c)$ and $C(b)$ both with index $\text{ind}((\exists R.C)(a))$.

- *Value-restriction rule.* Assume that $(\forall R.C)(a), R(a, b) \in \mathcal{A}_0$, and that $\mathcal{A}_0$ does not contain assertion $C(b)$ whose index is implied by $\text{ind}((\forall R.C)(a)) \wedge \text{ind}(R(a, b))$. The ABox $\mathcal{A}_1$ is obtained by extending $\mathcal{A}_0$ by $C(a)$ with index $\text{ind}((\forall R.C)(a)) \wedge \text{ind}(R(a, b))$.

Note that the last four rules in PRS for TBoxes are exactly same as in the original PRS. It is because they deal with the language constructs occurring in the construction of a complex concept term. Hence, no different than the original PRS, they perform the expansion of an $\mathcal{ALC}$-concept term by adding

new assertional facts that are constituents (in case of conjunction and disjunction) or implied by (in case of universal and existential-role restrictions) the fact to which a rule is applied. The first two rules are the ones that make this algorithm for TBoxes. They perform a version of lazy unfolding by adding facts where a defined concept name is replaced by its definition in the TBox. And label the new assertional fact by the label obtained from the conjunction of the label of the original assertion and the propositional variable labeling the concept definition in the TBox that made the rule application possible. Unlike RTS, the original assertional facts are not required to be tagged because the application condition of the rules makes sure that the rule is applied only once for an assertional fact.

After the translation of PRS for TBoxes, we have two algorithms that pinpoint the problematic axioms which make TBoxes unsatisfiable. In order to remove the bad axioms from the original knowledge base, both the tableau algorithms keep information about the axioms that cause the presence of a certain assertional fact anywhere in the tableau. So that if an assertional fact makes a clash with any other fact, the axioms involved in the generation of the conflicting facts are known. But the two algorithms handle the information about the axioms in different ways. The next chapter focuses on the issue of labels and shows which algorithm keeps the information in more efficient way.

Chapter 4

Labels in RTS and PRS

In this chapter, we analyze the ways the two algorithms keep their labels. We know that the tableaux while trying to exclude bad axioms, keep the information about the axioms that were responsible to generate a certain assertion as labels of the assertions. In RTS tableau, indices of the TBox axioms associated with an assertion are kept as a set of index-sets. It will be shown that this way of keeping labels is expansive in terms of space. On the other hand, PRS tableau develops monotonic Boolean formulas to maintain the information about axioms. The labels in RTS may grow exponentially larger as compared to those in PRS.

The idea behind the index-sets associated with an assertional fact in RTS is to group together all the axioms in a set which when appear together in an ontology will generate the assertional fact in the tableau for the concept term C and the TBox $\mathcal{T}$. So if there are more than one index-sets associated with an assertional fact as its label, it means there are more than one ways to get the assertional fact at that specific point in the tableau. For example, at a node x, $(a : A, \{\{p, q\}, \{p, r\}\})$ indicates that the assertion $a : A$ can be generated at x by the axioms represented by p and q together or alternatively by the axioms represented by p and r. Note that the axiom p is mentioned separately with q and r because it will be the cause of generating $a : A$ at x independently with q or r. This can result in quite large labels of assertional facts in RTS. PRS, on the other hand, can keep the labels in a shorter form. In the above example, PRS *can*, if the rules are applied in an appropriate order, mention the label of the assertional fact $a : A$ with Boolean formula $p \wedge (q \vee r)$, i.e. mentioning the axiom p once rather twice.

Hence, in RTS labels are larger than those in PRS. In fact, their size can blow up exponentially with regards to the size of the labels in PRS. The following section deals with the expansion of the labels in RTS as compared to those in PRS.

4.1 Blow up of Labels in RTS

Although the labels of each assertional fact in RTS are in the form of set (of sets), they can still be considered as monotonic Boolean formulas. For example, if an assertion has a labels set $\{\{p, q, r\}, \{r, s\}\}$ then it can be seen as a monotonic Boolean formula $((p \wedge q \wedge r) \vee (r \wedge s))$. As in RTS, the labels are represented by index-sets and all the axioms in a set are required (conjunction of all these axioms) to get the corresponding assertion and a slightly different way, in terms of the axioms, to get the same assertion needs the whole new index-set to be written. Hence, at least one of these sets of axioms (disjunction of the sets) is necessary for the assertion to be there at a certain node. It also points to the fact that RTS keeps the labels in *disjunctive normal form* (DNF), i.e. in the form of a disjunction of conjunctions. In PRS, labels are extended by disjunction if an assertional fact is produced by more than one different ways but the labels are not kept in DNF necessarily.

Keeping the labels in DNF can cause an exponential blow up of the labeling sets of assertional facts in RTS as compared to the labels in PRS for the same input.

To illustrate, we look into a general form of an input such that for any given natural number n, there is an input TBox and a concept term such that while trying to find maximal satisfiable subsets of the TBox with respect to the concept term, PRS produces an assertional fact whose labeling Boolean formula is polynomially large on n. For the same input, RTS produces the same assertional fact whose labeling index-set is exponentially large to n and it is DNF of the formula for the assertion in PRS.

Consider an input TBox of the form $\mathcal{T} := \{(A_1 \doteq \forall r.C)^{p_1}, \cdots, (A_n \doteq \forall r.C)^{p_n}, (B_1 \doteq \exists r.D)^{q_1}, \cdots, (B_n \doteq \exists r.D)^{q_n}\}$, and a concept term $A_1 \sqcap \cdots \sqcap A_n \sqcap B_1 \sqcap \cdots \sqcap B_n$ where p_i and q_i are propositional variables, with $i := 1, \cdots, n$, labeling the axioms. Both tableaux start with applying a conjunction rule once or at most $2n - 1$ times as the order of rule application may vary.

No matter in which order we apply the rules, every potential assertional fact in the input of the tableau will be treated finally and same set of the maximal satisfiable subsets will be calculated in both of the algorithms. But different order will produce different labels of assertional facts. Which implies that the clash formula in PRS will be syntactically different for different orders but every potential clash is treated and mentioned in the clash formula regardless of the order of the rules.

Due to the greater freedom of choosing an assertional fact for a rule application and the way labels are extended as a result of these rule applications, in PRS one can find an order that keeps labels of the assertional facts and clash formula succinct to such a degree that even a best possible order in RTS results into a significantly bigger labels. This fact is explained by applying rules for both the algorithms for the above mentioned input as follows.

In PRS, after applying the conjunction rule $2n - 1$ times there are $2n$ assertions of the form $A_i(a)$ with empty label. Then for every concept-definition rule

application on $A_i(a)$, new assertion $\exists r.D(a)^{q_i}$ or $\forall r.C(a)^{p_i}$ is generated depending upon the definition of A_i in $\mathcal{T}$. During this process when a new assertion has to be generated but its already there then the label of already existing assertion is extended by disjunction. Same order is possible for RTS as well. After exhausting conjunction and concept definition rules in PRS, there are two assertions $\forall r.C(a)^{p_1 \vee \cdots \vee p_n}$, we denote this assertion by $a_\forall$, and $\exists r.D(a)^{q_1 \vee \cdots \vee q_n}$, we denote this assertion by $a_\exists$, on which rules can be applied. For RTS, $a_\forall$ is $([a : \forall r.C, \{\{p_1\}, \cdots, \{p_n\}\}])$ and $a_\exists$ is $([a : \exists r.D, \{\{q_1\}, \cdots, \{q_n\}\}])$.

Now PRS applies exists-restriction rule on $a_\exists$ and generates a new assertion with new individual name b, $D(b)$ and a role assertion $r(a, b)$ each with same label as that of $\exists r.D(a)$. Then value-restriction rule becomes applicable on $a_\forall$ and its application generates assertion $C(b)$, we call it the *common assertion*, with the label obtained from the conjunction of the labels of $\forall r.C(a)$ and $r(a, b)$ which is $(p_1 \vee \cdots \vee p_n) \wedge (q_1 \vee \cdots \vee q_n)$, we call it the *compact-label*. In RTS, the $\exists$-rule treats both the assertions, $a_\exists$ and $a_\forall$, and generates assertion $b : C$, the common-assertion, with the set of index-sets obtained from those of the two assertions $a_\exists$ and $a_\forall$ which is $\{\{p_1, q_1\}, \{p_2, q_1\}, \cdots, \{p_n, q_n\}\}$, lets name it the *expanded-label*. We take the number of the propositional variables as the size of a Boolean formula (and of a set for that matter). It is clear that the compact-label is polynomial to n with $2n$ propositional variables while the expanded-label is exponential to n containing $2n^2$ propositional variables.

Proposition 4.1 *The compact-label of the common-assertion in PRS is semantically equivalent to the expanded-label of the common-assertion in RTS.*

Proof: As we have already argued, a representation of axioms in the form of a set of sets can be considered as a monotonic Boolean formula. The propositional variables relating to the axioms in an index-set are connected by conjunction as all these axioms are responsible for an assertion to be present at a certain node of RTS tableau. These conjunctions of the propositional variables relating to each individual index-set in a set of index-sets for an assertion $a : C$ are then connected by disjunction as at least any one of these set of axioms is necessary for $a : C$ to be there at a certain node.

Hence, the expanded-label $\{\{p_1, q_1\}, \{p_2, q_1\}, \cdots, \{p_n, q_n\}\}$ can be seen as a monotonic Boolean formula $((p_1 \wedge q_1) \vee (p_2 \wedge q_1) \vee \cdots \vee (p_n \wedge q_n))$. This Boolean formula is the DNF of the compact-label $((p_1 \vee \cdots \vee p_n) \wedge (q_1 \vee \cdots \vee q_n))$ labeling the common-assertion in PRS. Thus the two labels for the common-assertions $b : D$ in RTS and $D(b)$ in PRS are equivalent. □

Example 4.2 For $n \equiv 2$,
$\Gamma \equiv \{(A_1 \doteq \forall r.C)^{p_1}, (A_2 \doteq \forall r.C)^{p_1}, (B_1 \doteq \exists r.D)^{q_1}, (B_2 \doteq \exists r.D)^{q_2}\}$ and,
$\mathcal{A}_0 \equiv \{a : A_1 \sqcap A_2 \sqcap B_1 \sqcap B_2\}$

PRS proceeds on this input as follows:

$$\mathcal{A}_0 \equiv \{A_1 \sqcap A_2 \sqcap B_1 \sqcap B_2(a)\}$$

$$\downarrow \text{conjunction rule three times}$$

$$\mathcal{A}_1 \equiv \mathcal{A}_0 \cup \{A_1(a), A_2(a), B_1(a), B_2(a)\}$$

$$\downarrow axiom^{+}\text{- rule four times}$$

$$\mathcal{A}_2 \equiv \mathcal{A}_1 \cup \{\forall r.C(a)^{p_1 \vee p_2}, \exists r.D(a)^{q_1 \vee q_2}\}$$

$$\downarrow \text{exists-restriction rule}$$

$$\mathcal{A}_3 \equiv \mathcal{A}_2 \cup \{r(a,b)^{q_1 \vee q_2}, D(b)^{q_1 \vee q_2}\}$$

$$\downarrow \text{value-restriction rule}$$

$$\mathcal{A}_4 \equiv \mathcal{A}_3 \cup \{C(b)^{(p_1 \vee p_2) \wedge (q_1 \vee q_2)}\}$$

Note that any other order of rules will produce the same complete ABox $\mathcal{A}_4$ with possibly different labels.

RTS will work on the same input as follows:

$$L(x) \equiv \{(a : A_1 \sqcap A_2 \sqcap B_1 \sqcap B_2, \{\})\}$$

$$\downarrow \sqcap\text{-rule three times}$$

$$L(x) \equiv \{(a : A_1, \{\}), (a : A_2, \{\}), (a : B_1, \{\}), (a : B_2, \{\})\}$$

$$\downarrow D^{+}\text{-rule four times}$$

$$L(x) \equiv \{(a : A_1, \{\})^{tag}, (a : A_2, \{\})^{tag}, (a : B_1, \{\})^{tag}, (a : B_2, \{\})^{tag}, (a : \forall r.C, \{\{p_1\}, \{p_2\}\}), (a : \exists r.D, \{\{q_1\}, \{q_2\}\})\}$$

$$\downarrow \exists\text{-rule}$$

$$L(x) \equiv \{(a : A_1, \{\})^{tag}, (a : A_2, \{\})^{tag}, (a : B_1, \{\})^{tag}, (a : B_2, \{\})^{tag}, (a : \forall r.C, \{\{p_1\}, \{p_2\}\}), (b : D, \{\{q_1\}, \{q_2\}\}), (b : C, \{\{p_1, q_1\}, \{p_2, q_1\}, \{p_1, q_2\}, \{p_2, q_2\}\})\}$$

The 'tag' symbol indicates that the assertion is tagged. As we can see that both the tableaux have produced the common-assertion $b : C/C(b)$ and the labeling Boolean formula of $C(b)$ in PRS is polynomial to the given n but the length of the index-set on $b : C$ in RTS is exponential to n.

Proposition 4.3: *There is no order of rule application in PRS such that the compact-label is greater in length than the length of the expanded-label in RTS produced by any order of rule application.*

Proof[sketch]: As proposition 4.1 suggests that the compact-label in PRS and the expanded-label in RTS are semantically equivalent. Furthermore, the argument given above shows that the expanded-label is always DNF of the compact-label. No matter how naively rules are applied in PRS, the length of the compact-label cannot exceed the length of its DNF in worse case of the order of rule application. Hence, the expanded-label cannot be smaller than the compact-label. Moreover, RTS cannot apply rules in a more efficient way, to keep the labels compact, than the one we have shown in the example. It is because of the rule restriction for the application of the $\exists$-rule. □

The above argument even works for all kinds of inputs for the two tableaux and orderings of rule application. In all cases, the labels of assertional facts in PRS are succinct or in worse case equal in length to those in RTS. And its not the other way around.

Despite the apparent differences of the rules and the labeling methods of the two tableaux, there are some amazing similarities between them. The next chapter talks about these similarities of the two tableaux.

Chapter 5

Development of the Clash Formula

In this chapter, we argue about the development of clash formulas in the two tableaux during the execution of the algorithms; i.e. on-the-fly development. While explaining their algorithm, Baader and Hollunder do not talk about looking for any clash formula on-the-fly. They only look for the clash formula when the tableau stops and no more rules are applicable. But obviously this clash formula is under making during the rule application phase rather they just get it in the end. On the contrary, in RTS it is not of the interest to calculate a clash formula for the tableau. But as the clash formula has the information about clashes in the tableau, it is possible to develop one on-the-fly that contains the information about all clashes in RTS tableau those were resolved by the clash breaking rule. It will help in the comparison of the two tableaux by showing that although apparently they look different but internally they work in a similar fashion.

It will be shown that by adding some appropriate apparatus in each of the two algorithms, we can develop a clash formula along the rule application phase. This introduction of the new apparatus does not interfere with the outline of the original algorithms rather only collects the necessary information about the clashes and develops a clash formula for the tableau. For this purpose, we need to give a general notion of a clash formula.

Definition 5.1. *Let $\mathcal{T}$ be a TBox and C a concept term. Given $\mathcal{T}' \subseteq \mathcal{T}$, $w_{\mathcal{T}'}$ is the valuation that maps $t \in \mathcal{T}'$ to 'true' and others to 'false'.*
A monotonic Boolean formula ψ is a general clash formula *for $\mathcal{T}$ and C if for every $\mathcal{T}' \subseteq \mathcal{T}$ it holds that $w_{\mathcal{T}'}$ satisfies ψ iff C is $\mathcal{T}'$-unsatisfiable.*

5.1 Development of Clash Formula On-The-Fly

In this section, we try to show that how we can understand the execution of the algorithms as one producing the clash formula. For PRS, we can look for the clash formula that is being built during the rule application phase of the algorithm. We do this by adding a rule into the algorithm that searches for the clashes on-the-fly. This rule does not undermine the original central idea of the algorithm.

In PRS, the clash formula is obtained from the labels of the assertions and all the rules of the algorithm manipulate these labels. So in order to get a rule for on-the-fly clash formula, we introduce the bottom symbol, $\perp$, for every ABox in the tableau and during the algorithm we develop a clash formula related to an ABox as the label of $\perp$. The bottom rule is the following:

- *Bottom rule*: Assume that $C(a) \in \mathcal{A}_0$, $\neg C(a) \in \mathcal{A}_0$ and $\text{ind}(\perp)$ is not implied by $\text{ind}(C(a)) \wedge \text{ind}(\neg C(a))$. Then ABox $\mathcal{A}_1$ is obtained from $\mathcal{A}_0$ by extending $\text{ind}(\perp)$ to $\text{ind}(\perp) \vee (\text{ind}(C(a)) \wedge \text{ind}(\neg C(a)))$.

We refer to PRS with the bottom rule as PRSB. Note that every ABox has, if it contains any obvious contradiction, its $\perp$ symbol which collects the information about the clashes in that particular ABox. By adding this rule, the algorithm will generate everything that was generated before and nothing else except for the $\perp$ symbol for every ABox in the tableau. The bottom rule is applied upon finding an obvious contradiction between two assertional facts $A(a)^{\varphi_1}$ and $\neg A(a)^{\varphi_2}$ in an ABox $\mathcal{A}_i$. The propositional formula obtained from the conjunction of φ_1 and φ_2 is added to the label of the $\perp$ symbol of $\mathcal{A}_i$ as both of the assertional facts are necessary to produce the clash. The propositional formulas of individual clashes in $\mathcal{A}_i$ are combined with disjunction. At any point, the developing clash formula of the tableau during rule application can be obtained from the conjunction of $\text{ind}(\perp)$ for every ABox.

For a given input, if we apply the same order of rules in PRSB as in PRS and the additional bottom-rule in PRSB where applicable then both the tableaux will generate exactly the same number of ABoxes and for every ABox in PRS there is an ABox in PRSB containing exactly same assertional facts and a particular assertional fact is labeled by the same Boolean formula in both the tableaux. This implies that the clash formula for PRS generated at the end should be same as the one generated from the labels of the $\perp$ symbols in PRSB.

Proposition 5.2. *Let ψ be the Boolean formula generated by the conjunction of all the ind($\perp$) of the complete ABoxes of PRSB tableau for a TBox $\mathcal{T}$ and a concept term C. Then ψ is the clash formula associated with $\mathcal{T}$ and C.*

Proof: Suppose we start with PRS and PRSB for the same input and follow the same order of the rule application with additional bottom-rule in PRSB when there is an obvious contradiction in any of the ABoxes. And let

ψ_1 be the clash formula generated by PRS at the end of the tableau and ψ_2 be the clash formula generated by PRSB from the labels of the $\bot$ symbols.At the end of the PRS tableau, for every clash $A(a)^{\varphi_1}$, $\neg A(a)^{\varphi_2}$ in an ABox $\mathcal{A}_i$, the Boolean formula $\varphi_1 \wedge \varphi_2$ is added by disjunction with Boolean formulas from other clashes in $\mathcal{A}_i$. And ψ_1 is obtained from the conjunction of the Boolean formulas for all ABoxes of the tableau. PRSB also contains $\mathcal{A}_i$ with same assertional facts and labels as in PRS. In PRSB, with every application of the bottom-rule for a clash $A(a)^{\varphi_1}$, $\neg A(a)^{\varphi_2} \in \mathcal{A}_i$, $ind(\bot)$ is extended by $\varphi_1 \wedge \varphi_2$ disjunctively. Hence, at the end of the PRSB tableau, $ind(\bot)$ is the disjunction of all formulas for individual clashes in $\mathcal{A}_i$. And ψ_2 is obtained from the conjunction of $ind(\bot)$ for every ABox in the tableau. This is exactly what we get in PRS. Hence, ψ_1 is same as ψ_2. □

The proposition above illustrates that both, PRS and PRSB, tableaux generate exactly similar ABoxes with similar assertions for the same order of rules. Hence, every clash generated in PRS is also generated in PRSB and as clashes are generated in the two tableaux, the same clash formula is developing in both the cases. It shows the fact for which we introduced the bottom rule in PRS, that is, a clash formula is developing on the fly in PRS tableau.

Now we see if there is a clash formula building up during the application of rules in RTS. We need to notice here that there are two types of branching in the RTS tableau. First, when $\sqcup$-rule is applied and two branches are generated from the node where the rule is applied. Other type of branching is when the $\bot$-rule is applied. Each branch removes one of the the two conflicting assertions by removing the minimal hitting set of its index-sets.

To calculate the clash formula on-the-fly for RTS, we attach a propositional formula $\mathcal{C}_x$ for every leaf node x of the tableau that represents the developing clash history along the branch of the tableau where x is the leaf node. An individual clash at x, say between the assertions $(a : A, \mathcal{I})$ and $(a : \neg A, \mathcal{J})$, is represented in $\mathcal{C}_x$ by the subformula $\varphi_{K_1} \vee \cdots \vee \varphi_{K_n}$ where every φ_{K_i} is a propositional formula obtained from the conjunction of the propositional variables in $K_i \in \{I \cup J | I \in \mathcal{I}, J \in \mathcal{J}\}$. For example, a clash between the assertions $(a : A, \{\{p, q\}, \{q, r\}\})$ and $(a : \neg A, \{p_1, q_1\})$ is represented by the formula $((p \wedge q \wedge p_1 \wedge q_1) \vee (q \wedge r \wedge p_1 \wedge q_1))$. $\mathcal{C}_x$ is the disjunction of all the formulas representing individual clashes in x and its ancestral nodes (preceding x in the branch up to the root). The overall developing clash formula of the RTS tableau for a TBox $\mathcal{T}$ and a concept term C is obtained from the conjunction of $\mathcal{C}_n$s for every leaf node n. We use this idea to give the definition for the development of the clash formula in RTS next.

Definition 5.3 *Let $\mathcal{C}_x$ be the propositional formula representing all the clashes up till the leaf node x in a branch of the RTS tableau. Applying the $\bot$-rule for a clash between $(a : A, \mathcal{I})$ and $(a : \neg A, \mathcal{J})$ at x creates $\mathcal{C}_y := \mathcal{C}_x \vee \varphi_{K_1} \vee \cdots \vee \varphi_{K_n}$ for every child node y of x. Where φ_{K_i} is the conjunction of all propositional variables in $K_i \in \{I \cup J | I \in \mathcal{I}, J \in \mathcal{J}\}$. The conjunction of all $\mathcal{C}_n$s, n being a*

leaf node, gives the developing clash formula *for the RTS tableau.*

Note that a φ_{K_i} is a conjunction of all the propositional variables pertaining to the axioms responsible for producing the clashing assertions. Individual potential causes of the clashes, the sets of TBox axioms, in a branch of the RTS tableau are combined with disjunctions. Because a single clash in the leaf node of a branch makes it inconsistent. And the conjunction of all the clash formulas for the leaf nodes gives the clash formula for the tableau.

It is required to modify the $\sqcup$- and $\bot$-rules of RTS to do steps needed to calculate the clash formula as follows:

- $\sqcup$-rule: If $(a : C \sqcup D, \mathcal{I}) \in L(x)$ then create two children y and z of x;
  $L(y)$:=$L(x)\backslash\{(a : C \sqcup D, \mathcal{I})\} \cup \{(a : C, \mathcal{I})\}$; $E(y)$:=$E(x)$; $\mathcal{C}_y$:=$\mathcal{C}_x$;
  $L(z)$:=$L(x)\backslash\{(a : C \sqcup D, \mathcal{I})\} \cup \{(a : D, \mathcal{I})\}$; $E(z)$:=$E(x)$; $\mathcal{C}_z$:=$\mathcal{C}_x$;

- $\bot$-rule: If $(a : A, \mathcal{I}) \in L(x)$ and $(a : \neg A, \mathcal{J}) \in L(x)$ then
  For every $K \in (MH(\mathcal{I}) \cup MH(\mathcal{J}))$ do:
  Create a new child y of x;
  $L(y) := \{(b : D, \mathcal{I}_K) | (b : D, \mathcal{I}') \in L(x)\}$; $E(y) := E(x) \cup K$;
  $\mathcal{C}_y$:=$\mathcal{C}_x \vee \varphi_{K_1} \vee \cdots \vee \varphi_{K_n}$, φ_{K_i} being the conjunction of the propositional variables in $K_i \in \{I \cup J | I \in \mathcal{I}, J \in \mathcal{J}\}$
  EndFor

Note that modification of these rules does not affect the algorithm in any way but only helps in explicitly showing the development of the clash formula on-the fly.

As a result of an application of the $\sqcup$- or $\bot$-rule, x ceases to be a leaf node. In case of the $\sqcup$-rule, the child nodes of x; i.e. y and z, simply inherit $\mathcal{C}_x$ without any change. Hence, $\mathcal{C}_x = \mathcal{C}_y = \mathcal{C}_z$. In case of the $\bot$-rule, the information about a clash at x between the assertions $(a : A, \mathcal{I})$ and $(a : \neg A, \mathcal{J})$ for which the rule is applied, must be added to $\mathcal{C}_x$ to get $\mathcal{C}_y$ for every child node y of x. Therefore, the information about this clash at x is added in $\mathcal{C}_x$ to get $\mathcal{C}_y$.

The responsibility of a clash between two assertional facts $(a : A, \mathcal{I})$ and $(a : \neg A, \mathcal{J})$ at a node x in the RTS tableau lies upon the axioms in $\mathcal{I}$ and $\mathcal{J}$. But every $I_i \in \mathcal{I}$ and $J_i \in \mathcal{J}$ is independently responsible for the presence of $a : A$ and $a : \neg A$ at x respectively. This implies that, for $\mathcal{I} = \{I_1, \cdots, I_n\}$ and $\mathcal{J} = \{J_1, \cdots, J_m\}$, every element of $\mathcal{K} := \{I_i \cup J_k | 1 \leq i \leq n, 1 \leq k \leq m\}$ is a potential cause of the clash. In other words, occurrence of all the axioms represented in a $K_i \in \mathcal{K}$ produces the clash.

Proposition 5.4 *Let ψ be a Boolean formula generated by the conjunction of $\mathcal{C}_n$s for every leaf node n of a fully expanded tableau tree of RTS for a TBox $\mathcal{T}$ and a concept term C. Then ψ is a clash formula associated with $\mathcal{T}$ and C.*

Proof: Let $\mathcal{T}'$ be a subset of $\mathcal{T}$. In the following, the valuation $w_{\mathcal{T}'}$ is assumed to be such that it replaces the variables corresponding to the elements of $\mathcal{T}'$ by 'true' and rest by 'false'.

Consider the case where ψ is evaluated to 'false' by $w_{\mathcal{T}'}$. Then there is a leaf node n whose associated $\mathcal{C}_n$ is evaluated to 'false' by $w_{\mathcal{T}'}$. Which implies that there is no φ_{K_j} in $\mathcal{C}_n$ evaluated to 'true' by $w_{\mathcal{T}'}$. This means that $\mathcal{C}_n$ is free of clashes. Hence, C is $\mathcal{T}'$-satisfiable.

Now conversely, consider the case where ψ is evaluated to 'true' by $w_{\mathcal{T}'}$. Then every $\mathcal{C}_n$ in ψ associated with a leaf node n is evaluated to 'true' by $w_{\mathcal{T}'}$. Which means that every $\mathcal{C}_n$ of ψ contains a φ_{K_j} that is evaluated to 'true' by $w_{\mathcal{T}'}$. This implies that every leaf node n of the fully expanded tableau tree contains clashes. Hence, C is $\mathcal{T}'$-unsatisfiable. □

It is obvious now that along the RTS tableau as the $\perp$-rule is applied, a clash formula is developing which once developed completely, obtained from the conjunction of $\mathcal{C}_n$s of a fully expanded tableau tree, contains the information about all the clashes making the given concept term unsatisfiable for a given TBox. Similarly, PRSB also generates a clash formula while on the run through the application of the bottom rule. After rules are exhausted, the clash formula from the conjunction of every $ind(\perp)$ in the tableau contains information about every clash in the tableau. The interesting question should be if we apply same order of the rules in both the tableaux, do we get equivalent Boolean formulas at all stages during the tableaux.

In the previous chapter, we saw that beginning with the same input, similar assertions in the two tableaux, PRS and RTS, caused by the same axioms of the TBox have equivalent labels. It seems quite plausible that if PRSB and RTS start with the same input and apply the same order of rules including the bottom rules, their respective clash formulas should be equivalent at all stages in the tableaux. Its mainly so because similar rules in the two tableaux generate similar new assertions in the knowledge base and the assertions generated by same set of axioms in the two tableaux have equivalent labels. Obviously, the clash formulas in both the tableaux are generated from these labels of the conflicting assertional facts. Which implies that the clash formulas should be equivalent at all stages provided the order of rule application is same in the two tableaux.

Now we show that if the two tableaux start applying rules in the same order for a given TBox and a concept term then although apparently they seem working quite differently but under the surface they both are doing the same things.

5.2 Parallel Development of Clash Formula in PRSB and RTS

In this section, we will show that if the two algorithms apply same rules in the same order for a given input then they get equivalent clash formulas and contain equivalent knowledge at all stages during the tableaux. Let us remind ourselves that RTS and PRS are algorithms whose main goal is to calculate maximally satisfiable subsets of a given $\mathcal{ALC}$-knowledge base. Given the same

knowledge base, the completeness of the algorithms imply that both must find all maximally satisfiable subsets of the knowledge base. This implies that both algorithms discover similar clashes between the assertions and pinpoint same sets of axioms in the knowledge base that when taken together make the knowledge base inconsistent. Added to the fact that the rules for same constructs and role-restrictions in the two algorithms add similar knowledge and equivalently effect the labels of the assertional facts, leads to conclude that if same order of the rules is followed in the two algorithms then at all stages in the tableaux they must contain same knowledge and have equivalent developing clash formulas.

We need to define some terms that we will use while comparing the development of the clash formulas in the tableaux.

For every assertional fact $A(a)^{\varphi}$ in PRSB, the function $f : PRSB \rightarrow RTS$ maps it to an equivalent RTS assertional fact $(a : A, \mathcal{I})$ where $\mathcal{I}$ is the set of index-sets such that the Boolean formula obtained from $\mathcal{I}$, as argued in the section 4.1, is DNF of the formula φ. We refer to an assertional fact a in PRSB and $f(a)$ in RTS as *equivalent assertions* and their labels as *equivalent labels*. Moreover, for a given rule R in RTS, R' is its *related-rule* in PRSB if it treats the same construct/concept-definition/role-restriction as R treats in RTS.

Notice that for an $\exists$-rule for an assertional fact $f(a)$ in RTS, its equivalent rule in PRSB includes exists-restriction rule for a and value-restriction rule for every assertional fact with universal-role restriction as its main construct at the node.

Definition 5.5 *Suppose $x_1, \cdots, x_m$ are all the leaf nodes (ABoxes) for RTS (PRSB) with assertional facts and their index-sets (labels). The collection of these nodes (ABoxes) is called a* state *for the tableau. The root-node (input-ABox) is called the initial-state.*

In order to compare a given state in the RTS tableau with a state in the PRSB tableau and the effects of applying equivalent rules, we define a relationship between the states of the two tableaux.

Definition 5.6 *Let s_k be a state for the RTS tableau with nodes $x_1, \cdots, x_m$ and the tableau clash formula ψ. Then the state s'_k for the PRSB tableau is called the* related-state *to s_k if the followings hold:*

- *for every assertion $a : B$ in a node in s_k there is an ABox in s'_k containing equivalent assertion $B(a)$ with label equivalent to the index-set of $a : B$;*
- *The PRSB (developing) tableau clash formula ψ' is equivalent to ψ;*
- *For any rule applicable in s_k for assertion $a : B$, its related-rule is applicable in s'_k for assertion $B(a)$ with equivalent label as index-set of $a : B$.*

Note also that all the assertional facts in a state of the RTS tableau are present in the related state of the PRSB tableau but reverse is not necessarily true. It is because RTS tableau gets rid of some assertional facts after applying

rules for them but PRSB keeps all the assertional facts in the successive state. Hence, a state in RTS tableau is a subset of its related state in PRSB tableau.

Now we claim that if RTS tableau is in state s_n and PRSB tableau is in state s'_n, and these two states are related then applying a rule on an assertion $f(a_i)$ in s_n and applying the related-rule on the equivalent assertion a_i in s'_n will move the two tableaux to related-states. In other words, applying a rule in the RTS tableau has the same effect in terms of generating new assertions and clash formula as that of applying the related-rule in the PRSB tableau.

Proposition 5.7 *Let s_n be the state of RTS tableau and applying a rule R takes the tableau to a state s_{n+1}. And let s'_n be the related-state to s'_n of PRSB tableau and applying a rule R' takes the tableau to a state s'_{s+1}. If R and R' are related-rules then s_{n+1} and s'_{n+1} are related states.*

Proof: Let s_n be a state of the RTS tableau with assertional facts $\{a_1, \cdots, a_m\}$ and the developing tableau clash formula ψ, and s'_n be a related stated of s_n in the PRSB tableau with assertions $\{a_1, \cdots, a_n\} \supseteq \{f(a_1), \cdots, f(a_m)\}$ and with the developing tableau clash formula ψ' equivalent to ψ.

Suppose a rule R is applicable on an assertion a_i at a leaf node x in the RTS tableau then there is a related-rule R' applicable to the equivalent assertion $a'_i = f(a_i)$ in an ABox in the PRSB tableau. Let applying R rule for a_i in the RTS tableau moves it to a state s_{n+1} and application of R' on a'_i in the PRSB tableau moves it to a state s'_{n+1}. To show if s_{n+1} and s'_{n+1} are related, we proceed a follows.

For any rule R, other then the $\bot$-rule, applied in the state s_n of the RTS tableau for the assertion a_i, the new state s_{n+1} is obtained by adding some new knowledge (assertional facts) possibly removing a_i. The related-rule R' would be applicable in the related-state s'_n and it will add same new knowledge to the successive state s'_{n+1} in PRSB tableau. For example, applying $\sqcap$-rule to $(a : C \sqcap D)$ adds new assertional facts $(a : C)$ and $(a : D)$. The related conjunction rule of PRSB applied to the equivalent assertional fact $C \sqcap D(a)$ adds the assertional facts $C(a)$ and $D(a)$ that correspond to $(a : C)$ and $(a : D)$ respectively. Note that in both the cases new assertional facts will get the labels of the original fact.

After the application of the R and R' rules, if a rule R_1 is applicable in RTS tableau then there are two possibilities. The first possibility is that its precondition was there before the application of the rule R. Then the precondition for the application of the related rule R'_1 was also there in PRSB tableau as well because both the algorithms were in related states. So the new related rules R_1 and R'_1 are applicable in there respective tableau. The second possibility is that the rule R_1 became applicable in RTS as the result of the new knowledge added to the state which made the precondition of the rule application. Then the same knowledge is added in the new state of PRSB so R'_1 should be applicable in PRSB as well.

Suppose R and R' were the bottom rules in their respective tableaux for the clashing assertions a_i and a_j, then the PRSB tableau simply extends $ind(\perp)$ to $\text{ind}(\perp)\vee(\text{ind}(a_i)\wedge\text{ind}(a_j))$ and does not add or remove anything else in s'_{n+1}.

The RTS tableau branches for every minimal hitting set of $\mathcal{I}$ for $(a : A, \mathcal{I})$ and of $\mathcal{J}$ for $(a : \neg A, \mathcal{J})$ at the node x. Each branch removing one of the clashing assertional facts and the assertional facts which were the cause of this assertional fact. The overall effective knowledge remains the same in the state s_{n+1} as in s_n because what is removed in one branch is present in other one. The developing clash formula $\mathcal{C}_x$ for every new child node y of x in ψ is extended to $\mathcal{C}_x \vee (\varphi_{K_1}) \vee \cdots \vee \varphi_{K_n})$ with φ_{K_i} is the conjunction of the propositional variables in $K_i \in \{I_i \cup J_k | 1 \leq n, 1 \leq m\}$. On the basis of the argument used for proposition 4.1, it can be stated that the disjunctive extension of ψ' in PRSB by $(\text{ind}(a_i)\wedge\text{ind}(a_j))$ is equivalent to the extension of ψ in RTS by $(\varphi_{K_1}) \vee \cdots \vee (\varphi_{K_n})$ after applying the $\perp$-rule. □

It is obvious from the rules of the two algorithms that the application conditions of the related rules for the two tableaux are similar, hence each rule applicable in RTS is also applicable in PRS and both working on the same assertional fact generate similar new knowledge in the knowledge base. We have shown that the application of the same order has same effect on the output of the tableaux.

The order of the rules applied in PRS tableau can be matched with those applied in RTS but the reverse can not be true as PRS gives a complete freedom of choosing an assertional fact for applying the rule which is not the case for RTS. But if the same order of rules is applied in both the algorithms then they express the same knowledge at the same time. Hence, internally they work in the same fashion while from the outside they may appear different.

Chapter 6

Conclusions

In this work, a couple of algorithms from the literature to debug $\mathcal{ALC}$ knowledge bases were analyzed and compared. Both the algorithms use axiom pinpointing to calculate maximal satisfiable subsets of the knowledge bases. And finally it was shown that although apparently looking different, actually they do the same job of removing bad axioms from a knowledge base in a quite similar fashion using tableaux-like algorithms.

To be more explicit, we introduced RTS and PRS algorithms to find maximal satisfiable subsets of knowledge bases. PRS which is originally devised to pinpoint ABox axioms, was translated to pinpoint TBox axioms to facilitate the comparison between the two. It was done by adding the rules that replace defined concept names in the knowledge by their definitions in the input TBox.

It was shown that while both the approaches keep the axioms along with an assertional fact that cause the generation of it but PRS is much more efficient than RTS and keeps the information about the axioms in a very compact form. For a given TBox and a concept term, RTS tries to keep every set of axioms which generate an assertion independently and it leads to the DNF of the labels. PRS has a better way to keep the labels succinct in the form of Boolean formulas. This difference gives rise to the expansion of the labels in RTS by exponential factor as compared to the ones in PRS.

Finally, we proved that for the same input, same order of the rules in the two tableaux have same effect on the development of the clash formulas in the respective tableau and generation of the new knowledge. Both the algorithms produce equivalent clash formulas on-the-fly if equip with necessary apparatus to calculate the clash formula while applying the rules.

We provided this apparatus to both the algorithms and showed that both of them produce equivalent clash formulas for the same input and same order of the rule application. PRS can offers much smarter order of rules application, thanks to its freedom of applying rule for any assertional fact available, which allows to keep the tableau smaller and clash formula shorter.

The two algorithms we discussed do the debugging job for terminologies expressed in $\mathcal{ALC}$. It should be of great interest to adopt these algorithms for more expressive languages and for general TBoxes. It should be also of particular interest to develop applications focusing on the Semantic Web as these application are focused upon debugging the existing terminologies and logical contradictions caused by the transformation or merging of the terminologies.

Bibliography

[1] Franz Baader, Diego Calvanese, Deborah McGuinness, Deniele Nardi, and Peter Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[2] Franz Baader and P. Hanschke. A scheme for integrating concrete domains into concept languages. Research Report RR-91-10, DFKI Kaiserslautern, 1991.

[3] Franz Baader and Bernhard Hollunder. Embedding defaults into terminological knowledge representation formalisms. *Journal of Automated Reasoning* 14:149-180, 1995.

[4] R. Cornet and A. Abu-Hanna. Evaluation of a frame-based ontology, a formalization oriented approach. In *Proceedings of MIE2002, Studies in health Technology and Information,* volume 90, pages 488-493, 2002.

[5] Volker Haarslev and Ralf Mller. Racer system description. In R. Gore, A. Leitsch, and T. Nipkow, editors, *IJCAR 2001,* volume LNAI 2100, 2001.

[6] Bernhard Hollunder. Hybrid inferences in KL-ONE-based knowledge representation systems. In *14th German Workshop on Artificial Intelligence, Ebingerfled, Germany*, volume 251 of *Informatik-Fachberichte*, Springer, pages 38-47, 1990.

[7] I. Horrocks. The FaCT system. In H de Swart, editor, *Tebleaux '98*, volume LNAI 1397, pages 307-312, 1998.

[8] A. Kalyanpur, B. Parsia, and E. Sirin. Black box techniques for debugging unsatisfiable concepts. In *International workshop on Description Logics*, 2005.

[9] A. Kalyanpur, B. Parsia, J. Hendler and E. Sirin. Debugging unsatisfiable classes in OWL ontologies. In *Journal of Web Semantics - Special Issue of the Semantic Web Track of WWW2005*, 2005.

[10] Thomas Meyer, Kevin Lee, Richard Booth and Jeff Z. Pan. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$. In *Proceedings of AAAI06*, 2006.

[11] Thomas Meyer, Kevin Lee, and Richard Booth. Knowledge integration for description logics. In *Proceedings of AAAI05*, pages 645-650, 2005.

[12] Marvin Minsky. A framework for representing knowledge. Technical report, Cambridge, MA, USA, 1974.

[13] M. Quillian. Semantic memory. In M. Minsky, editor, *Semantic Information Processing*, pages 227-270. MIT Press, 1968.

[14] Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32:57-92, 1987.

[15] Stefan Schlobach. Diagnosing terminologies. In *Proceedings of AAAI05*, pages 670-675, 2005.

[16] Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for debugging of description logic terminologies. In *Proceedings of IJCAI 2003*, pages 355-360. Morgan Kaufmann, 2003.

[17] Manfred Schmidt-Schau, and Gert Smolka. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1-26, 1991.

[18] Dmitry Tsarkov and Ian Horrocks. Efficient reasoning with range and domain constraints. In *Proceedings of the Description Logics Workshop (DL2004)*, pages 41-50, 2004.