EXPLANATION AND DIAGNOSIS SERVICES FOR UNSATISFIABILITY AND INCONSISTENCY IN DESCRIPTION LOGICS

Xi Deng

A thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfillment of the Requirements

For the Degree of Doctor of Philosophy

Concordia University

Montréal, Québec, Canada

September 2010

Your file Votre référence
ISBN: 978-0-494-71156-9
Our file Notre référence
ISBN: 978-0-494-71156-9

Abstract

Explanation and Diagnosis Services for Unsatisfiability and Inconsistency in Description Logics

Xi Deng, Ph.D.

Concordia University, 2010

Description Logics (DLs) are a family of knowledge representation formalisms with formal semantics and well understood computational complexities. In recent years, they have found applications in many domains, including domain modeling, software engineering, configuration, and the Semantic Web. DLs have deeply influenced the design and standardization of the Web Ontology Language OWL. The acceptance of OWL as a web standard has reciprocally resulted in the widespread use of DL ontologies on the web. As more applications emerge with increasing complexity, non-standard reasoning services, such as explanation and diagnosis, have become important capabilities that a DL reasoner should provide. For example, unsatisfiability and inconsistency may arise in an ontology due to unintentional design defects or changes in the ontology evolution process. Without explanations, searching for the cause is like looking for a needle in a haystack. It is, therefore, surprising

that most of the existing DL reasoners do not provide explanation services; they provide "Yes/No" answers to satisfiability or consistency queries without giving any reasons.

This thesis presents our solution for providing explanation and diagnosis services for DL reasoners. We firstly propose a framework based on resolution to explain inconsistency and unsatisfiability in Description Logic. A sound and complete algorithm is developed to generate explanations for the DL language $\mathcal{ALCHI}$ based on the unsatisfiability and inconsistency patterns in $\mathcal{ALCHI}$.

We also develop a technique based on Shapley values to measure inconsistencies in ontologies for diagnosis purposes. This measure is used to identify which axioms in an input ontology or which parts of these axioms need to be repaired in order to make the input consistent. We also investigate optimization techniques to compute the inconsistency measures based on particular properties of DLs.

Based on the above theoretical foundations, a running prototype system is implemented to evaluate the practicability of the proposed services. Our preliminary empirical results show that the resolution based explanation framework and the diagnosis procedure based on inconsistency measures can be applied in the real world applications.

Acknowledgments

This thesis is the outcome of my journey in obtaining the Ph.D. degree in Computer Science and fortunately I have not been a lonely traveler.

First of all, it is difficult to overstate my gratitude to my supervisors: Dr Volker Haarslev and Dr Nematollaah Shiri. I have received continuing inspiration and encouragement from them throughout my study. The insightful discussions with them have greatly shaped the ideas and development in this research in various ways. I could not imagine finishing this thesis without their guidance.

I am also indebted to my colleagues for providing a stimulating and relaxed working environment. I especially value the friendship of my officemates: Hsueh-Ieng Pai and Ali Kiani. In addition I am grateful to Yu Ding, Ming Zuo, Nima Mohajerin, Mina Aslani, Jocelyne Faddoul, Amineh Fadhil, and Qiong Huang.

I also want to thank all my friends for their emotional support. Cuiming Chen, Jiaoyue Wang, Yue Wang, Yingying She, Xin Tong and Chao Jin deserve special mention.

Lastly, and most importantly, I want to thank my parents for their endless

love and support in every step of my life. To them I dedicate this thesis.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

Description Logics (DLs) are a family of knowledge representation formalisms that use concepts and roles to model application domains in a structured and formal way. The term comes from the fact that complex *descriptions* built with concepts and roles are used to represent knowledge, equipped with *logic*-based semantics.

A DL knowledge base is composed of two parts: a TBox and an ABox. A TBox describes concepts (unary predicates representing sets of individuals) and roles (binary predicates representing relations between individuals). An ABox describes the membership of individuals and pairs of individuals in roles. Examples of basic reasoning services in Description Logics include *subsumption*, which determines whether one concept subsumes another, *instance checking*, which checks if an individual is an instance of a certain concept, *satisfiability*, which determines if there is a model for a certain concept, *consistency*, which checks the satisfiability of the knowledge

base. All other reasoning problems can be reduced to satisfiability checking, making it a pivotal reasoning service in applications.

The first DL based system KL-ONE [BS85] was proposed as an extension to semantic networks [Qui67] and frame-based knowledge base systems [Min74], which were not equipped with a formal representation language and logical reasoning. The semantics of KL-ONE is model-theoretic and as with most DL languages, it can be regarded as a fragment of first-order logic. By restricting the expressive power and taking advantage of the structured representation, more efficient reasoning techniques can be devised in KL-ONE other than the traditional decision procedures for first-order logic. In addition, reasoning in different fragments of first-order logic leads to different computational complexity problems. As a consequence, a major body of DL research has been devoted to investigating the trade-off between expressiveness and computational complexity, which has been thoroughly mapped out (a survey can be found in [HPSMW07]) and forms the theoretical foundation of the implementation line of research in DL. In earlier systems such as CLASSIC [BMPSR90] and LOOM [Mac91], reasoning algorithms were developed based on structural subsumption. Such algorithms transform concept descriptions to normal forms and then decide concept subsumption by comparing the structures of these normal forms. However, there does not exist a sound and complete structural subsumption algorithm for expressive DL languages.

As a remedy to these deficiencies, tableau algorithms were proposed in [SSS91].

A tableau algorithm proves satisfiability of a knowledge base by trying to construct a model. If such a model can be built, the knowledge base is satisfiable, otherwise it is unsatisfiable. Sound and complete tableau algorithms have been introduced for very expressive DL languages, and optimization techniques such as absorption and caching have also been proposed [Hor07]. These algorithms were successfully implemented in most of state-of-the-art reasoners, such as Racer [HM01], FaCT++ [TH06], Pellet [SP04] and HermiT [MSH09].

1.1 Motivation

In recent years, Description Logics have found their way into many application domains, including domain modeling, software engineering, configuration, and the Semantic Web [BN07]. For example, Description Logics have deeply influenced the design and standardization of the Web Ontology Language OWL [w3c]. The acceptance of OWL as a web standard has also resulted in the widespread use of DL based ontologies on the Internet. As more and more applications emerge with increasing size and complexity, unsatisfiability and inconsistency problems are more than usual to encounter. An unsatisfiable concept (or an inconsistent knowledge base) is a concept (or a knowledge base) that cannot instantiate any individuals. Unsatisfiability and inconsistency may arise due to unintentional design defects or changes during the ontology evolution process. For instance, the DICE (Diagnoses

for Intensive Care Evaluation) terminology [HS05] contains more than 2,400 concepts, out of which about 750 concepts are unsatisfiable due to migration from other terminological systems. Although these problems need immediate attention, most existing DL reasoners do not provide explanation services; they merely provide a "Yes/No" answer to a satisfiability or consistency query without explaining why the unsatisfiability or inconsistency occurs or how it can be resolved. It has turned out that providing an explanation of the origin of the unsatisfiability/inconsistency is important for users. Hence, in addition to the primitive answers, it is imperative that DL reasoners provide explanations for these answers and identify their sources. Moreover, offering suggestions to resolve the unsatisfiability/inconsistency can be helpful to users too. For example, it is possible that removing a certain source of inconsistency results in resolving other inconsistencies in the knowledge base as well, which suggests this source is more problematic, and users might prefer to first resolve the most problematic sources. Therefore, in order to further assist knowledge engineers and ontology developers to improve the quality of the knowledge base, it is also crucial to provide a diagnosis service as a useful facility for DL reasoners.

1.2 Contributions

In the first part of this thesis, we propose a resolution proof based framework to provide explanations for unsatisfiability and inconsistency in the Description Logic language $\mathcal{ALCHI}$. In general, there are two approaches to build an explanation

system. One is to construct a system specially designed to provide explanations, as suggested in [WT92]. Several proposals to support explanations in theorem proving systems follow this idea [Hua94, Mei00]. Another solution is to extend an existing reasoner to include an explanation module, which traces the internal reasoning procedure to produce explanations. Most research on explanations in logic programming and deductive databases follow this approach [Byr80, ST90]. In this thesis, we propose a framework of constructing explanations for unsatisfiability and inconsistency problems in the DL language $\mathcal{ALCHI}$ using resolution proofs, which follows the basic ideas of the first approach, although necessary interactions with the DL reasoner is also considered. $\mathcal{ALCHI}$ is a reasonably expressive DL language which includes constructors such as conjunctions, disjunctions, existential and universal restrictions. It also allows constructors for role hierarchies and inverse roles. We investigate unsatisfiability and inconsistency patterns in $\mathcal{ALCHI}$ to further improve the efficiency of generating explanations. Based on this framework, we propose an algorithm and establish its soundness and completeness.

There are three main advantages of the proposed framework. First, the resolution technique can generate explanations at a fine-grained and logical level in contrast to merely debugging the knowledge base. Besides, compared to the previous approaches of using natural deduction proofs to explain reasoning, we use the resolution technique which is more focused, since all the literals involved in a proof contribute directly to the solution. Table 1 is a variant of the example in [Hua96]

illustrating a natural deduction proof style explanation. The number shown in the second column at each row indicates the hypothesis that on which the derivation depends. The conclusion of the inference is shown in the third column. The last column indicates the inference rule and/or the hypotheses used in a row. Considering that this problem can be solved in two steps using resolution as shown in Table 2, it shows that the natural deduction proof is indeed long and tedious.

No	Hyp.	Conclusion	Reason
1.	1	A	(Hypothesis)
2.	2	$A \sqsubseteq B$	(Hypothesis)
3.	3	$\neg B$	(Hypothesis)
4.	2	$\neg A \vee B$	(Tautology 2)
5.	5	$\neg A$	(Hypothesis)
6.	1, 5	$\bot$	(1, 5)
7.	7	B	(Hypothesis)
8.	3, 7	$\bot$	(3, 7)
9.	1, 2, 3	$\bot$	(Case analysis 4, 6, 8)
10.	1, 2	B	(Indirect 9)

Table 1: A Natural Deduction Proof Style Explanation

No	Hyp.	Clause	Reason
1.	1	$\{A\}$	(Hypothesis)
2.	2	$\{\neg A, B\}$	(Hypothesis)
3.	3	$\{\neg B\}$	(Hypothesis)
4.	1, 2	$\{B\}$	(Resolution 1, 2)
5.	3, 4	$\bot$	(Resolution 3, 4)

Table 2: A Resolution Proof Style Explanation

A second advantage of our resolution based approach is that it is independent of any specific DL reasoners being used. Most implemented DL reasoners use a tableau algorithm as their decision procedure, which is known to be decidable. The development of highly efficient optimization techniques has also demonstrated that

acceptable performance can be achieved. However, tableau algorithms are designed to render results faster but not necessarily easier for users to comprehend. For example, some DL optimization techniques are adopted to make reasoning more efficient; ease of understanding of reasoning procedures are not often the concerns of such techniques. Therefore, if the internal reasoning procedures are traced in order to generate explanations for general users, they must be tailored with performance penalties. In our approach, explanations are constructed based on resolution proofs, hence no modification of the internals of DL reasoners is required. This makes our proposed solution applicable to arbitrary DL reasoners.

The third advantage of our proposed solution is that it can benefit from many features of resolution based theorem provers to provide better explanations. For example, there might be more than one reason for the unsatisfiability or inconsistency in a knowledge base, and it is not sufficient for the explanation service to stop whenever the first source of the contradiction is detected. Since many resolution based theorem provers, such as Otter [WW97], can be configured to provide all the resolution proofs they can find, we can rely on the provers to provide alternative proofs in our approach.

In the second part of the thesis, we present a diagnosis framework based on inconsistency measures. As shown in real life applications, all inconsistencies are not equally "bad." It is possible for an ontology to contain two or more sources of inconsistencies and they may have different impacts on the inconsistencies. They

may not necessarily contain the same contradiction and the same information, and may have overlapping content. The following are just two possible scenarios.

- Non-overlapping: there are more than one set of axioms that contribute to an inconsistency in an ontology and they are independent of one another.

  Suppose K' and K'' are two inconsistent subsets of an ontology $\mathcal{O}$. When we say $\mathcal{O}$ has non-overlapping sources of inconsistencies, it means that $K' \cap K'' = \emptyset$. Note that an inconsistency might occur at different levels: at the level of a single axiom, and at the level of sets of axioms.

- Overlapping: there are more than one set of axioms that contribute to an inconsistency in an ontology and they are interweaved with one another.

  An ontology $\mathcal{O}$ with overlapping sources of inconsistencies means that $K' \cap K'' \neq \emptyset$. When two sets of inconsistent axioms are overlapping, it indicates that certain axioms contribute more to the inconsistencies and these axioms are possibly more problematic than others. It is most likely the case that removing one of these axioms from $\mathcal{O}$ will result in resolving other inconsistencies as well.

In this thesis we propose a quantitative measure of inconsistencies in ontologies based on Shapley values. This measure gives users guidelines on priorities of the axioms to be removed and their consequences. It can be applied to clauses instead of axioms. Since clauses are more fine-grained than axioms, it allows us to take a deeper look inside the axioms and find out which proportion of the axiom

contributes to the inconsistency. The computational complexity of calculating the Shapley value is at least Exp-time, which shows that it does not scale well in general [CS04]. Therefore we propose to optimize the calculation based on structural relevance of the axioms and properties of the defined inconsistency measure. Our main contribution in this direction is twofold: we combine previously known game theory strategies into ontology reasoning and present a measure to systematically evaluate the inconsistencies in ontologies. To the best of our knowledge, this is the first work in Description Logics towards providing a quantitative measure of inconsistencies. This approach is independent of a particular species of ontology languages or a particular reasoning system used.

1.3 Outline of Thesis

The rest of this thesis is organized as follows:

- In Chapter 2 we discuss the necessary theoretical background of this work. We briefly introduce the syntax and semantics of the DL language $\mathcal{ALCHI}$, which is the main focus of this thesis. We also provide an overview of the fundamental reasoning services and decision procedures in DL, followed by the syntax, semantics and decision procedures in resolution based first-order logic theorem proving.

- In Chapter 3, we review the related work in expert systems, deductive databases, automated theorem provers and DL based knowledge bases. We also compare our approach with the discussed related work in debugging, explaining and repairing knowledge bases. The definitions and properties of explanations in the context of DL are presented as well.

- Chapter 4 presents our framework for providing explanations of DL $\mathcal{ALCHI}$ knowledge bases based on resolution proofs. The framework translates the DL knowledge base into first-order logic formulae and then invokes an automated theorem prover to obtain a resolution proof and its corresponding refutation graph. By traversing the refutation graph based on unsatisfiability and inconsistency patterns, an explanation is generated and presented to users.

- Chapter 5 provides implementation details of a prototype system based on the framework introduced in Chapter 4. We also evaluate the performance of the explanation procedure and compare it with other approaches.

- Chapter 6 discusses a technique based on Shapley values to measure inconsistencies in DL knowledge bases. This measure can be used to identify which parts in an inconsistent knowledge base need to be removed or modified in order to make it consistent. We also propose optimization techniques to improve the efficiency of computing Shapley values.

- Chapter 7 presents empirical results of performance and usability evaluations

of the inconsistency measures introduced in Chapter 6.

- Chapter 8 presents the open issues in our approach and outlines future research directions.

Various results of our research have been published in conference/workshop proceedings and journals. The resolution proof framework to generate explanations for $\mathcal{ALC}$ was published in [DHS05a, DHS05b]. A sound and complete algorithm is presented in [DHS06] and the underlying Description Logic language is later extended to $\mathcal{ALCHI}$ based on unsatisfiability/inconsistency patterns in [DHS07b]. The ontology inconsistency measure was published in [DHS07a].

Chapter 2

Preliminaries

In this chapter, we will first introduce Description Logics, a family of knowledge representation languages that can be used to model an application domain in a formal way. We will introduce the syntax, semantics and inference problems in the DL language $\mathcal{ALCHI}$. We will then review resolution techniques from first-order logic. For more materials on Description Logics and resolution, the interested reader is referred to [BN07] and [RV01].

2.1 Description Logics

Description Logics (DLs) are a family of concept-based knowledge representation formalisms. The name is derived from the fact that, on the one hand, the notions in the domain of interest are described by concept *descriptions*, i.e., expressions that are built from atomic concepts (unary predicates) and atomic roles (binary predicates)

using the constructors provided by the particular Description Logic language. On the other hand, Description Logics differ from their predecessors, such as semantic networks and frames, in that they are equipped with a formal, *logic*-based semantics [BHS07].

2.1.1 Introduction

Description Logics represent the knowledge of a domain by first defining the relevant concepts of the domain. These concepts are then used to specify properties of the objects and individuals in the domain. Typically a DL knowledge base has two parts: *terminology* (TBox) and *assertions* (ABox). The TBox specifies intensional knowledge in the form of axioms. The ABox contains the extensional knowledge that is specific to elements in the domain, called individuals.

We begin by reviewing the language $\mathcal{AL}$ (for *Attribute Language*), which has been introduced as a minimal language of interest in Description Logics. Other languages of this family (including $\mathcal{ALCHI}$, which is the focus of this thesis) are various extensions of $\mathcal{AL}$. The informal description is given in this section while the formal description of its syntax, semantics and inference services is given in Section 2.1.2. In $\mathcal{AL}$, basic descriptions are *atomic concepts*, designated by unary predicates to specify the objects, and *atomic roles*, designated by binary predicates to express relationships between individuals. The constructors allowed in $\mathcal{AL}$ are atomic negation, intersection, value restriction and limited existential quantification.

Arbitrary *concept descriptions* such as C and D are defined recursively from atomic concepts and roles using the DL constructors according to the following syntax rules:

$$
\begin{aligned}
C, D \rightarrow A| & \qquad \text{(atomic concept)} \\
\top| & \qquad \text{(universal concept)} \\
\bot| & \qquad \text{(bottom concept)} \\
\neg A| & \qquad \text{(atomic negation)} \\
C \sqcap D| & \qquad \text{(intersection)} \\
\forall R.C| & \qquad \text{(value restriction)} \\
\exists R.\top & \qquad \text{(limited existential quantification)}
\end{aligned}
$$

We will illustrate some typical constructors by examples. Formal definitions will be given later in Section 2.1.2. Suppose that Person and Female are atomic concepts. Then Person $\sqcap$ Female and Person $\sqcap \neg$ Female are concepts describing people who are female and people who are not female. In addition, assuming that hasSpouse is an atomic role, then the concept description Person $\sqcap \exists$hasSpouse.$\top$ denotes people who have a spouse.

Concept descriptions can be used to build axioms, which express how concepts and roles are related to each other. Generally, an axiom is a statement of the form $C \sqsubseteq D$, read as "concept C is subsumed by concept D", or $C \equiv D$, indicating that $C \sqsubseteq D$ and $D \sqsubseteq C$ hold, where C and D are concept descriptions. For instance, by the axiom:

$$\text{Married} \sqsubseteq \text{Person} \sqcap \exists \text{hasSpouse}.\top$$

we define the concept Married as people who have spouses.

A TBox is a set of axioms through which we can introduce atomic concepts and concept descriptions such as Married. In addition, we can also introduce individuals and assert properties of these individuals in an ABox. For example, suppose JANE and JACK are individual names, then Married(JANE) means that Jane is married, and hasSpouse(JANE, JACK) asserts that JANE is the spouse of JACK.

A more expressive language can be obtained if more constructors are added to $\mathcal{AL}$. For example, $\mathcal{ALCHI}$ is the extension of $\mathcal{AL}$ by allowing negation of arbitrary concepts (indicated by $\mathcal{C}$ for *Complement*), role hierarchy (indicated by $\mathcal{H}$) and inverse role (indicated by $\mathcal{I}$). For instance, using role hierarchy, we can assert that if people have spouses then they have relatives: hasSpouse $\sqsubseteq$ hasRelative. We can also define that the inverse of the role hasChild yields the role hasParent.

Modern Description Logic systems provide their users with reasoning services that can automatically deduce implicit knowledge from the explicitly represented knowledge, and always yield a correct answer in finite time. The subsumption algorithm determines subconcept-superconcept relationships: C is subsumed by D if all instances of C are necessarily instances of D, i.e., the first concept description is always interpreted as a subset of the second concept description. For example, given the definition of Married mentioned earlier and the axiom Person $\sqsubseteq$ Creature, which says that humans are creatures, Married is subsumed by $\exists$hasSpouse.Creature since instances of Married are married to some instances of Person, and all instances

of Person are also instances of Creature. The instance checking algorithm determines concept membership: an individual a is an instance of the concept description C if a is always interpreted as an element of the interpretation of C. For example, given the aforementioned assertions and the definition of Married, we conclude that JANE is an instance of Person (because JANE is an instance of Married, so she is also a person according to the definition of Married). The consistency algorithm determines whether a knowledge base (consisting of a set of assertions and a set of terminological axioms) is consistent. In a typical application, one would start building the TBox, making use of the reasoning services provided to ensure that all concepts in it are satisfiable, i.e., are not subsumed by the bottom concept, which is always interpreted as the empty set. Moreover, one would use the subsumption algorithm to compute the subsumption hierarchy, i.e., to check, for each pair of concept names, whether one is subsumed by the other. Besides, given an ABox, one would first check for its consistency with the TBox and then, for example, compute the most specific concept(s) that each individual is an instance of (this is often called realizing the ABox). We could also use a concept description as a query, i.e., we could ask the DL system to identify all those individuals that are instances of the given, possibly complex, concept description.

Obviously, all the knowledge we have described in our examples can easily be represented by formulae of first-order predicate logic (see also Section 2.2.1). The variable-free syntax of Description Logics makes TBox axioms easier to read than

the corresponding first-order formulae. And the object-oriented way of modeling the application domain is more intuitive from a user's perspective. However, the main reason for using DLs rather than predicate logic is that DLs are carefully tailored for practical purposes which provides a trade-off between expressiveness and decidability of the important reasoning problems (see Section 2.1.2).

2.1.2 The DL language $\mathcal{ALCHI}$

In following, we first review the syntax and semantics of the DL language $\mathcal{ALCHI}$. We will also define the important inference problems w.r.t. a knowledge base consisting of a TBox and an ABox.

Definition 2.1.1 ($\mathcal{ALCHI}$ syntax) *Let N_C be a set of concept names and N_R be a set of role names. The set of $\mathcal{ALCHI}$-roles is $N_R \cup \{R^- \mid R \in N_R\}$. Let $\mathtt{Inv}(R) = R^-$ and $\mathtt{Inv}(R^-) = R$ for $R \in N_R$. The set of $\mathcal{ALCHI}$-concept descriptions is the smallest set such that the following properties hold:*

1. *$\top$, $\bot$, and every concept name $A \in N_C$ is an $\mathcal{ALCHI}$-concept,*
2. *if C and D are $\mathcal{ALCHI}$-concepts and R is an $\mathcal{ALCHI}$-role, then $C \sqcap D, C \sqcup D, \neg C, \forall R.C$, and $\exists R.C$ are $\mathcal{ALCHI}$-concepts.*

In this thesis, for the sake of simplicity, we will often use "$\mathcal{ALCHI}$-concept" instead of "$\mathcal{ALCHI}$-concept description". The semantics of $\mathcal{ALCHI}$ is given in terms of *interpretations*, defined as follows.

Definition 2.1.2 ($\mathcal{ALCHI}$ semantics) *An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the domain, and the interpretation function $\cdot^{\mathcal{I}}$, which maps every $\mathcal{ALCHI}$-concept C to a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, and maps every $\mathcal{ALCHI}$ role R to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. In addition, $\mathcal{I}$ maps each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$.*

For all $\mathcal{ALCHI}$-concepts C, D and all role names R, $\mathcal{I}$ assigns meanings as follows:

$$
\begin{array}{lcl}
\top & = & \Delta^{\mathcal{I}} \\
\bot & = & \emptyset \\
\neg C & = & \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}} \\
C \sqcap D & = & C^{\mathcal{I}} \cap D^{\mathcal{I}} \\
C \sqcup D & = & C^{\mathcal{I}} \cup D^{\mathcal{I}} \\
\forall R.C & = & \{a \in \Delta^{\mathcal{I}} \mid \forall b \ (a,b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\} \\
\exists R.C & = & \{a \in \Delta^{\mathcal{I}} \mid \exists b \ (a,b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\} \\
R^{-} & = & \{(a,b) \in \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \mid (b,a) \in R^{\mathcal{I}}\}
\end{array}
$$

We say that $C^{\mathcal{I}}$ (or $R^{\mathcal{I}}$) is the extension of the concept C (or role name R) in the interpretation $\mathcal{I}$. If $x^{\mathcal{I}} \in C^{\mathcal{I}}$, then we say that x is an instance of C in $\mathcal{I}$.

As mentioned earlier, a DL knowledge base (KB) is made up of two parts, a terminological part (called the TBox) and an assertional part (called the ABox). Each part consists of a set of axioms.

Definition 2.1.3 *A knowledge base (KB) is a pair $K = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox*

and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ is a model of K if $\mathcal{I}$ is a model of $\mathcal{T}$ and $\mathcal{I}$ is a model of $\mathcal{A}$. A model of $\mathcal{T}$ (or $\mathcal{A}$) is defined to be an interpretation $\mathcal{I}$ that satisfies the axioms of $\mathcal{T}$ (or $\mathcal{A}$).

Definition 2.1.4 *A TBox $\mathcal{T}$ is called unfoldable if it satisfies the following conditions.*

1. *All axioms in $\mathcal{T}$ are definitional. Definitional axioms are the axioms of the form $A \equiv C$ or $A \sqsubseteq C$ for some concept name A.*

2. *Axioms in $\mathcal{T}$ are unique. That is, for each concept name A, $\mathcal{T}$ contains at most one axiom of the form $A \equiv C$, and if it contains an axiom of the form $A \equiv C$, then it does not contain any axiom of the form $A \sqsubseteq C$.*

3. *$\mathcal{T}$ is acyclic. That is, there is no axiom $A_i \equiv C_i \in \mathcal{T}$ such that A_i occurs in C_i. A concept name A occurs in a concept expression C if either A occurs syntactically in C, or there is a concept name A' such that A' occurs syntactically in C, and there is an axiom $A'_i \equiv C'_i \in \mathcal{T}$ such that A occurs in C'.*

The most general form of TBox axioms are so-called general concept inclusions.

Definition 2.1.5 *A TBox in $\mathcal{ALCHI}$ is a set of concept inclusion axioms and role inclusion axioms. A general concept inclusion (GCI) is of the form $C \sqsubseteq D$, where C and D are $\mathcal{ALCHI}$-concepts. A finite set of GCIs is called a TBox. An interpretation $\mathcal{I}$ is a model of a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; $\mathcal{I}$ is a model of a general*

TBox $\mathcal{T}$ if it is a model of every GCI in $\mathcal{T}$. We use $C \equiv D$ when $C \sqsubseteq D$ and $D \sqsubseteq C$. A role inclusion axiom is of the form $R \sqsubseteq S$ such that $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$, where R and S are $\mathcal{ALCHI}$-roles. For a set of role inclusion axioms $\mathcal{R}$, we define a role hierarchy as $\mathcal{R}^{+} = (\mathcal{R} \cup \{\mathtt{Inv}(R) \sqsubseteq \mathtt{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}, \overset{*}{\sqsubseteq})$, where $\overset{*}{\sqsubseteq}$ is the transitive-reflexive closure of $\sqsubseteq$ over $\mathcal{R} \cup \{\mathtt{Inv}(R) \sqsubseteq \mathtt{Inv}(S) \mid R \sqsubseteq S \in \mathcal{R}\}$.

The ABox can contain two kinds of axioms, one for asserting that an individual is an instance of a given concept, and the other for asserting that a pair of individuals is an instance of a given role.

Definition 2.1.6 *An assertional axiom is of the form $a : C$ or $(a, b) : R$, where C is an $\mathcal{ALCHI}$-concept, R is an $\mathcal{ALCHI}$-role, and a, b are individual names. A finite set of assertional axioms is called an ABox. The interpretation $\mathcal{I}$ is a model of an assertional axiom $a : C$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ or $(a, b) : R$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$.*

We define inference problems w.r.t. a KB consisting of a TBox and an ABox.

Definition 2.1.7 (Inference Services) *Given a KB $K = (\mathcal{T}, \mathcal{A})$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox, K is said to be consistent if it has a model. The basic inference services in TBoxes include satisfiability, subsumption, and equivalence.*

- ***Satisfiability*** *A concept C is satisfiable with respect to K if there is a model $\mathcal{I}$ of K with $C^{\mathcal{I}} \neq \emptyset$. Such an interpretation is called a model of C w.r.t. K.*
- ***Subsumption*** *The concept D subsumes the concept C w.r.t. K (denoted as $K \models C \sqsubseteq D$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of K.*

- ***Equivalence*** *Two concepts C, D are equivalent w.r.t. K, denoted as $K \models C \equiv D$ if $K \models C \sqsubseteq D$ and $K \models D \sqsubseteq C$ w.r.t. K.*

The latter two inference services can be reduced to (un)satisfiability.

The basic reasoning tasks in ABoxes include instance checking, realization, and retrieval.

- ***Instance check*** *verifies if a given individual a is an instance of a specified concept C with respect to K (denoted as $K \models a : C$), i.e., if $a^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ holds for all models $\mathcal{I}$ of K.*

- ***Realization*** *finds the most specific concept that an individual is an instance of.*

- ***Retrieval*** *finds the individuals in the knowledge base that are instances of a given concept.*

Similar to the inference services in TBoxes, these three inference services in ABoxes can also be reduced to the consistency problem of ABoxes.

The most widely used technique to solve the above reasoning problems is the tableau based approach.

Given a knowledge base K = $(\mathcal{T}, \mathcal{A})$, we can assume without loss of generality that the concepts mentioned in $\mathcal{T}$ and $\mathcal{A}$ are in negation normal form (NNF), i.e., the negation is applied only to concept names. An arbitrary $\mathcal{ALCHI}$ concept can be

transformed to an equivalent concept in NNF by pushing negations inwards using a combination of de Morgan's laws. For example, the concept $\neg(\exists r.A \sqcap \forall s.B)$, where A, B are concept names, can be transformed to an equivalent NNF concept $(\forall r.\neg A) \sqcup (\exists s.\neg B)$.

The idea behind the algorithm of the tableau based approach is that it tries to prove the consistency of a knowledge base $K = (\mathcal{T}, \mathcal{A})$ by constructing (a representation of) a model of K. It does this by starting from the concrete situation described in $\mathcal{A}$, and explicating additional constraints on the model that are implied by the concepts and axioms in $\mathcal{A}$ and $\mathcal{T}$.

In order to construct such a finite representation, the algorithm works on a data structure called a completion forest. This consists of a labeled directed graph, in which each node is the root of a completion tree. Each node x in the completion forest (which is either a root node or a node in a completion tree) is labeled with a set of concepts $\mathcal{L}(x)$, and each edge $\langle x, y\rangle$ (which is either between root nodes or inside a completion tree) is labeled with a set of role names $\mathcal{L}(\langle x, y\rangle)$. If $\langle x, y\rangle$ is an edge in the completion forest, then we say that x is a predecessor of y (and that y is a successor of x); in case $\langle x, y\rangle$ is labeled with a set containing the role name R, then we say that x is an R-predecessor of y (or that y is an R-successor of x). A node y is called an R-neighbor of a node x if either y is a successor of x and $S \in \mathcal{L}(\langle x, y\rangle)$ or y is a predecessor of x and $\mathtt{Inv}(S) \in \mathcal{L}(\langle y, x\rangle)$ for some S with $S \stackrel{*}{\sqsubseteq} R$. Given a knowledge base K $= (\mathcal{T}, \mathcal{A})$, the completion forest $F_{\mathcal{A}}$ is initialized by including a

root node x_a, with $\mathcal{L}(x_a) = \{C|\ a : C \in \mathcal{A}\}$, for each individual name a occurring in $\mathcal{A}$, and an edge $\langle x_a, x_b \rangle$, with $\mathcal{L}(\langle x_a, x_b \rangle) = \{R|\ (a,b) : R \in \mathcal{A}\}$, for each pair (a,b) of individual names for which the set $\{R|\ (a,b) : R \in \mathcal{A}\}$ is non-empty.

The algorithm then applies the so-called expansion rules (see Table 3), which syntactically decompose the concepts into node labels, either inferring new constraints for a given node, or extending the tree according to these constraints. For example, if $C_1 \sqcap C_2 \in \mathcal{L}(x)$, and either $C_1 \notin \mathcal{L}(x)$ or $C_2 \notin \mathcal{L}(x)$, then the $\sqcap$ -rule adds both C_1 and C_2 to $\mathcal{L}(x)$; if $\exists R.C \in \mathcal{L}(x)$, and x does not yet have an R-successor with C in its label, then the $\exists$ -rule generates a new R-successor node y of x with $\mathcal{L}(y) = C$. Note that the $\sqcup$ -rule is different from the other rules in that it is non-deterministic: if $C_1 \sqcup C_2 \in \mathcal{L}(x)$ and neither $C_1 \in \mathcal{L}(x)$ nor $C_2 \in \mathcal{L}(x)$, then it adds either C_1 or C_2 to $\mathcal{L}(x)$. In practice this is the main source of increased complexity in tableau algorithms, because it may be necessary to explore all possible choices of rule applications.

The algorithm stops if it encounters a clash: that is a completion forest in which $\{A, \neg A\} \subseteq \mathcal{L}(x)$ for some node x and some concept name A. In this case, the completion forest contains an obvious inconsistency, and thus does not yield a model. If the algorithm stops without having encountered a clash, then the obtained completion forest yields a finite representation of a forest model, and the algorithm answers "$(\mathcal{T}, \mathcal{A})$ is consistent". If all possible choices of the non-deterministic $\sqcup$ -rule fail to yield such a representation of a forest model, i.e., all of them lead to a

$\sqcap$ -rule:	if 1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, and 2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ then set $\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$ -rule:	if 1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and 2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$; then set $\mathcal{L}(x) = \mathcal{L}(x) \cup C$ for some $C \in \{C_1, C_2\}$
$\exists$ -rule:	if 1. $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and 2. x has no R-neighbor y with $C \in \mathcal{L}(y)$, then create a new node y with $\mathcal{L}(\langle x, y \rangle) = R$ and $\mathcal{L}(y) = \{C\}$
$\forall$ -rule:	if 1. $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and 2. there is an R-neighbor y of x with $C \notin \mathcal{L}(y)$ then set $\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$

Table 3: The Tableau Expansion Rules for $\mathcal{ALCHI}$

clash, then the algorithm answers "$\mathcal{T}, \mathcal{A}$ is inconsistent".

In order to guarantee termination of the expansion process, the algorithm uses a technique called *blocking*. For logics without inverse roles, the general procedure is to check the label of each new node y, and if it is a subset of the label of an existing node x, then no further expansion of y is performed: x is said to block y. The resulting tree corresponds to a cyclical model in which y is identified with x. The validity of the cyclical model simply follows from the fact that the concepts which y must satisfy must also be satisfied by x, because x's label is a superset of y's. Blocking is, however, more problematic when inverse roles are added to the logic, and a key feature of the algorithms is the introduction of an equality blocking strategy. For further details of the blocking technique, please refer to [BS01].

The computational complexity of the reasoning problems in $\mathcal{ALCHI}$ is as follows [Don07].

Theorem 2.1.1 *Checking satisfiability and subsumption of concepts in $\mathcal{ALCHI}$ and consistency of $\mathcal{ALCHI}$ ABoxes are PSpace-complete. Adding TBoxes with GCIs results in ExpTime-hardness.*

In spite of the discouraging theoretical complexities, implemented DL systems have demonstrated that acceptable performance can be achieved through the use of optimization techniques, a wide variety of which have been studied in [Hor07].

2.2 Resolution Based Theorem Proving

In this section, we review the resolution technique that our work is based upon. We assume that the reader is familiar with standard definitions of first-order logic (FOL) and clausal theorem proving in the standard logic. For details, please refer to [Fit96] and [BG01].

2.2.1 First-Order Logic (FOL)

In the following, we review the syntax and semantics of first-order logic.

Definition 2.2.1 (first-order logic syntax) *A first-order language is built over a signature $\sum = (\mathcal{F}, \mathcal{P})$, where $\mathcal{F}$ and $\mathcal{P}$ are non-empty, disjoint sets of function and predicate symbols, respectively. Every function or predicate symbol has a fixed*

arity. In addition to these sets that are specific for a first-order language, we assume an infinite set $\mathcal{X}$ of variable symbols disjoint from the symbols in $\sum$. Then the set of terms $\mathcal{T}(\mathcal{F}, \mathcal{X})$ is recursively defined as follows:

1. *every constant symbol $c \in \mathcal{F}$ with arity zero is a term,*
2. *every variable $x \in \mathcal{X}$ is a term,*
3. *whenever $t_1, \ldots, t_n$ are terms and $f \in \mathcal{F}$ is a function symbol with arity n, then $f(t_1, \ldots, t_n)$ is a term.*

If $t_1, \ldots, t_n$ are terms and $P \in \mathcal{P}$ is a predicate symbol with arity n, then $P(t_1, \ldots, t_n)$ is an atom. We recursively build well-formed formulae (wff) over atoms, the logical constants $\top$ (true), $\bot$ (false) and the logical connectives $\supset$ (implication), $\equiv$ (equivalence), $\wedge$ (conjunction), $\vee$ (disjunction), $\neg$ (negation) and the quantifiers $\forall$ (universal), $\exists$ (existential) as usual:

1. *every atom is a wff,*
2. *$\top$ and $\bot$ are wffs,*
3. *if ϕ_1 and ϕ_2 are wffs, so are $\phi_1 \supset \phi_2, \phi_1 \equiv \phi_2, \phi_1 \vee \phi_2, \phi_1 \wedge \phi_2$ and $\neg\phi_1$,*
4. *if ϕ is a wff and $x \in \mathcal{X}$, then $\forall x\phi$ and $\exists x\phi$ are wffs.*

An atom or the negation of an atom is called a *literal.* For convenience, we often write $\forall x_1, \ldots, x_n\phi$ instead of $\forall x_1 \ldots \forall x_n\phi$ and analogously for the existential quantifier.

A *clause* is a disjunction of literals, where all variables are implicitly universally quantified. A first-order logic formula can always be converted into an equisatisfiable clausal normal form (a set of clauses interpreted as a conjunction).

Definition 2.2.2 (first-order logic semantics) *An interpretation is a triple* $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, v \rangle$, *where* $\mathcal{D}$ *is a non-empty set, called the domain of discourse,* $\mathcal{I}$ *is a function that associates n-ary predicates symbols from* $\mathcal{P}$ *and function symbols from* $\mathcal{F}$ *with n-place relations and functions respectively. The variable assignment function* $v : \mathcal{X} \mapsto \mathcal{D}$ *associates with each variable symbol a concrete value taken from* $\mathcal{D}$. *We use* $v[x/a]$ *to denote the variable assignment function that is exactly like* v *except that it maps the variable* x *to the domain value* a. *We sometimes use* $\mathcal{M}[x/a]$ *as a short form of* $\langle \mathcal{D}, \mathcal{I}, v[x/a] \rangle$ *where* $\mathcal{M} = \langle \mathcal{D}, \mathcal{I}, v \rangle$. *Given an interpretation* $\mathcal{M}$ *and a term* t *the interpretation of* t *with respect to* $\mathcal{M}$, *denoted by* $\mathcal{M}(t)$, *is defined recursively as follows:*

1. $\mathcal{M}(t) = v(x)$, *if* t *is a variable* x,
2. $\mathcal{M}(f(t_1, \ldots, t_n)) = \mathcal{I}(f)(\mathcal{M}(t_1), \ldots, \mathcal{M}(t_n))$.

An interpretation $\mathcal{M}$ *satisfies a formula* ϕ, *denoted by* $\mathcal{M} \models \phi$, *which is defined recursively as follows.*

$\mathcal{M} \models \top$

$\mathcal{M} \not\models \bot$

$\mathcal{M} \models P(t_1, \ldots, t_n)$ *iff* $(\mathcal{M}(t_1), \ldots, \mathcal{M}(t_n)) \in \mathcal{I}(P)$

$\mathcal{M} \models \neg\phi$ *iff* $\mathcal{M} \not\models \phi$

$\mathcal{M} \models \phi \wedge \psi$ *iff* $\mathcal{M} \models \phi$ *and* $\mathcal{M} \models \psi$

$\mathcal{M} \models \forall x\ \phi$ *iff* $\mathcal{M}[x/a] \models \phi$ *for every* $a \in \mathcal{D}$

$\mathcal{M} \models \exists x\ \phi$ *iff* $\mathcal{M}[x/a] \models \phi$ *for some* $a \in \mathcal{D}$

As usual $\phi_1 \vee \phi_2$ *is interpreted as* $\neg(\neg\phi_1 \wedge \neg\phi_2)$, $\phi_1 \supset \phi_2$ *is interpreted as* $\neg\phi_1 \vee \phi_2$ *and* $\phi_1 \equiv \phi_2$ *is interpreted as* $(\phi_1 \supset \phi_2) \wedge (\phi_2 \supset \phi_1)$.

In case there exists an interpretation $\mathcal{M}$ *that satisfies* ϕ, *we say that* ϕ *is satisfiable and that* $\mathcal{M}$ *is a model of* ϕ. *If every interpretation satisfies a formula* ϕ *then* ϕ *is called valid and we write* $\models \phi$. *If a formula* ϕ *is transformed into a formula* ψ *by some operation (rule), we say that such a transformation preserves satisfiability, i.e.,* ϕ *is satisfiable iff* ψ *is satisfiable.*

A *substitution* σ is a mapping from the set of variables to the set of terms such that $x\sigma \neq x$ for only finitely many $x \in \mathcal{X}$. We define the *domain* of σ to be $dom(\sigma) = \{x \mid x\sigma \neq x\}$. Hence, we can denote a substitution σ by the finite set $x_1 \mapsto t_1, \ldots, x_n \mapsto t_n$ where $x_i\sigma = t_i$ and $dom(\sigma) = \{x_1, \ldots, x_n\}$. The application of substitutions to terms is given by $f(t_1, \ldots, t_n)\sigma = f(t_1\sigma, \ldots, t_n\sigma)$ for all $f \in \mathcal{F}$ with arity n. We can extend substitutions to formulae as follows: $P(t_1, \ldots, t_n)\sigma = P(t_1\sigma, \ldots, t_n\sigma), (\neg\phi)\sigma = \neg(\phi\sigma), (\phi_1 \circ \phi_2)\sigma = \phi_1\sigma \circ \phi_2\sigma$, where

$\circ \in \{\supset, \equiv, \wedge, \vee\}, (\forall x\ \phi)\sigma = \forall x\sigma\ \phi\sigma$ if $x\sigma \in \mathcal{X}$ and $(\forall x\ \phi)\sigma = \forall x\ \phi$ if otherwise. Similarly $(\exists x\ \phi)\sigma = \exists x\sigma\ \phi\sigma$ if $x\sigma \in \mathcal{X}$ and $(\exists x\ \phi)\sigma = \exists x\ \phi$ if otherwise.

A substitution σ is called a *unifier* for a set of formulae S if $S\sigma$ is a singleton. A unifier σ for S is called a *most general unifier* (mgu) for S if, for each unifier θ of S, there exists a substitution γ such that $\theta = \sigma\gamma$.

2.2.2 Resolution Based Refutation Technique

Theorem provers are procedures that can be used to check whether a given formula F (the "goal") is a logical consequence of a set of formulae N (the "theory"). Refutational theorem provers deal with the equivalent problem of showing that the set $N \cup \{\neg F\}$ is inconsistent. The inconsistency of a theory can be established either by a semantic analysis or by providing a formal proof (also called derivation) of $\perp$ (the empty clause) from the theory, where proofs are traces of inferences defined by a collection of inference rules.

Definition 2.2.3 *An inference rule is an $(n+1)$-ary relation on clauses. The elements of such a relation are usually written as:*

$$\frac{C_1 \ldots C_n}{C}$$

and called inferences. The clauses $C_1, \ldots, C_n$ are called the premises of the inference, and C is called its conclusion. An inference system Γ is a collection of inference rules.

If $\mathbf{I}$ is an inference or a set of inferences we denote as $\mathbf{C}(\mathbf{I})$ its conclusion or the set of their conclusions. We also speak of an inference *from* a set of clauses N if all premises are elements of N. In this case, we use $\Gamma(N)$ to denote the set of all inferences by Γ from N.

Definition 2.2.4 *An inference is said to be sound if its conclusion is a logical consequence of its premises, i.e.,* $C_1 \ldots C_n \models C$, *where the clauses* $C_1, \ldots, C_n$ *are the premises of the inference, and* C *is its conclusion*

Soundness is often a minimal requirement expected of an inference system, but in refutational theorem proving it is sufficient that inferences preserve consistency. We call an inference system Γ *consistency-preserving* if for all sets of clauses N and their conclusions of all inferences by an inference system (denoted by $\mathbf{C}(\Gamma(N))$), the set $N \cup \mathbf{C}(\Gamma(N))$ is consistent whenever N is consistent. A sound inference system is consistency-preserving, but the reverse is not true in general.

A *proof* of a clause C from a set of clauses N with respect to an inference system Γ is a clause set $C_1, \ldots, C_m$, such that $C = C_m$ and each clause C_i is either an element of N or the conclusion of an inference by Γ from $N \cup \{C_1, ..., C_{i-1}\}$. The clauses in N are also called *assumptions*. We write $N \vdash_\Gamma C$ if there exists a proof of C from N by Γ. If C is a contradiction we speak of a *refutation* of N.

Definition 2.2.5 *An inference system* Γ *is said to be refutationally complete if there is a refutation by* Γ *from any unsatisfiable set of clauses* N. *A set of clauses* N *is*

called saturated with respect to Γ *if the conclusion of any inference by* Γ *from* N *is an element of* N.

If an inference system Γ is refutationally complete and a set N is saturated with respect to Γ, then N is either satisfiable or contains a contradiction.

2.2.3 Resolution

Resolution is a widely used refutationally complete theorem proving method in first-order logic, which means the empty clause can be derived from any unsatisfiable set of clauses. The search for a contradiction proceeds by saturating the given clause set, i.e., systematically and exhaustively applying all inference rules.

Many versions of resolution for standard clauses have been proposed in the literature. In this chapter we only introduce the most basic variant for ground clauses: binary resolution with factoring. Interested readers are referred to [BG01] and [Llo87]. We denote the resolution calculus, consisting of the following inference rules, where the clauses $C \vee A \vee B$ and $D_1 \vee \ldots \vee D_n \vee \neg B$ are called the main premises, $C_1 \vee \ldots \vee C_n \vee A$ is called the side premise, and $C\sigma \vee A\sigma$ and $C_1\sigma \vee \ldots C_n\sigma \vee D_1\sigma \vee \ldots D_n\sigma$ are called conclusions. This calculus is used in Table 4 to deduce a contradiction from the given input clauses.

Positive factoring:

$$\frac{C \vee A \vee B}{C\sigma \vee A\sigma}$$

(1)	$A \vee B$	(input)
(2)	$\neg A \vee B$	(input)
(3)	$\neg B$	(input)
(4)	B	(Resolving on A in (1) and (2))
(5)	$\bot$	((3) and (4))

Table 4: An Example of Resolution

where σ is a most general unifier (mgu) of A and B, denoted as $\text{mgu}(A, B)$.

Resolution:

$$\frac{C_1 \vee \ldots \vee C_n \vee A \quad D_1 \vee \ldots \vee D_n \vee \neg B}{C_1\sigma \vee \ldots C_n\sigma \vee D_1\sigma \vee \ldots D_n\sigma}$$

where σ is a most general unifier (mgu) of A and B, denoted as $\text{mgu}(A, B)$.

Resolution is sound and refutationally complete: if a set of clauses is saturated up to redundancy by the inference rules, then it is satisfiable if and only if it does not contain the empty clause.

2.3 Conclusion

In this chapter, we introduced the theoretical background on which this thesis is based. We first studied the syntax and semantics of the Description Logic language $\mathcal{ALCHI}$. This language provides a set of constructors, such as conjunctions, disjunctions, negations, as well as role hierarchies and inverse roles, to represent concepts and roles in the application domain. A typical DL knowledge base consists of a set of axioms (TBoxes) and assertions (ABoxes). Some specially designed reasoning procedures called tableau algorithms are used to derive implicit knowledge.

In the second part, we presented the syntax and semantics of first-order Logic. We also introduced resolution theorem proving based on binary resolution with factoring.

Chapter 3

Related Work

Explanations and diagnosis facilities are important for Knowledge Base systems. Many deductive databases, expert systems and automated theorem provers [MD99, Byr80, LD04, FM87] are equipped with explanation or diagnosis facilities. They can help users to understand the query results and also help knowledge engineers to design and debug the system. However, explanations and diagnosis services in Description Logics are still under research and development. The aim of this chapter is to investigate what has been done so far in explanations and diagnosis in both Description Logics and other related areas.

We start by describing the notions of explanations in DL. We then review the use of explanations and debugging in DL systems, followed by the review of explanation methods in other Knowledge Base systems. We also review the research work on diagnosis in DL systems.

3.1 Definition and Properties of Explanations

Although explanations have been acknowledged to be important in Description Logic systems, the definition and characteristics of explanations have never been formally presented in the related literature. In this section we summarize the previous work in Description Logic systems and present the definition and properties of explanations in the DL context.

3.1.1 Definition of Explanations

Many researchers in Knowledge Base systems, in particular expert systems, have proposed different ways to interpret the meaning of explanations. For example, the first rule-based expert system MYCIN [Cla81] is able to show the users why a piece of information is requested and how a conclusion is obtained. In case-based reasoning systems [DDC03], a good explanation is a complete case of the highest degree of similarity with the current problem case. In deductive databases such as Coral [ARR+93], or automated theorem provers such as PROVERB [Hua94], proof trees are presented as explanations.

Intuitively, explanations are the information that can help users achieve a better understanding of a knowledge base. More specifically, explanations can enhance the users' comprehension and confidence of a system when the result is correct as expected and help them diagnose the defects when an unexpected (perhaps wrong) result has been encountered.

3.1.2 Properties of Explanations

After reviewing the literature on explanations in Description Logics (a detailed discussion can be found in Section 3.2), we summarize the following characteristics of existing explanation methods.

Purpose: There are two kinds of objectives in generating explanations, thus leading to two different categories of approaches:

Comprehension: The goal of this kind of explanations is to understand the provenance information and the conclusions generated by the system. For instance, explanations show the source of the information, the facts used by the reasoning, as well as the reasoning procedure of the system.

Debugging: This approach of explanations can be used to locate unintentional defects in the system. For example, explanations help identify the errors by pinpointing the problematic snippets of information in the system.

Presentation: This is related to how explanations are presented to users. Users can choose the level of details of the explanations disclosed. The explanations can be displayed as text in natural language style or as graphics, for instances.

Methodology: There are two kinds of methodologies when it comes to building an explanation system:

Reconstruction: In this case, a stand-alone explanation system is specially designed to provide explanations.

Extension: Another approach is to extend an existing system to include an

explanation module, which traces the internal reasoning procedure to provide explanations.

3.2 Explanation in Description Logics Systems

In this section, we review efforts that have been undertaken to provide explanations in the context of DL systems. We analyze the properties of the explanations they provide based on the discussion in Section 3.1. We also discuss the differences between these methods and our approach.

3.2.1 Explaining Structural Subsumption

The earliest work on explanations in Description Logics reasoning include [McG96, MB95]. It provides an explanation facility for subsumption and non-subsumption reasoning in CLASSIC [BMPSR90]. CLASSIC is a family of knowledge representation systems based on Description Logics which allow universal quantification, conjunction, restricted number restrictions and path (in)equations. This approach proposes to explain subsumption in a proof-theoretic manner based on structural subsumption, using inference rules to perform structural subsumption comparisons. Lengthy explanations are decomposed into smaller steps and a single step explanation is followed by more detailed explanations. We will illustrate this approach by showing a simple example from [McG96].

Consider a TBox $\mathcal{T}$ as:

$$A \equiv (\textbf{and}\ (\textbf{prim}\ \text{grape})\ (\textbf{prim}\ \text{Good Wine}))$$

where **and** denotes conjunction, and **prim** denotes concept primitive set (atomic concept). For example, (**prim** Good Wine) is a set of atomic concepts {Good, Wine}.

Consider the explanation of the subsumption:

$$A \sqsubseteq (\textbf{prim}\ \text{Wine})$$

The related defined inference rule[1] is as follows:

Prim

$$\frac{\alpha \subset \theta}{\vdash (\textbf{prim}\ \theta) \Rightarrow (\textbf{prim}\ \alpha)}$$

This rule says that if a concept's primitive set θ contains another set of primitives α, then this concept is subsumed by the smaller set of primitives.

To deal with large and complex proofs, they propose a Normalize-Compare algorithm to decompose the subsuming concept into parts that can be presented independently. In other words, it will break the concept into its component conjuncts, which are called atomic descriptions, and then proceed by separating smaller proofs of each part. In the above example, the subsuming concept can be decomposed into (**prim** grape) and (**prim** Wine).

After the subsuming concept is broken into its atomic descriptions, atomic justifications are used to explain the subsumption relationship. It has the form: A

[1]The upper part of the rule is called the antecedence, and the lower part is called the consequence.

$\Rightarrow B$ because *ruleID* (<argument list>), where B is an atomic description, *ruleId* is the name of an inference rule, and argument list shows bindings for variables in the inference rule. Now using atomic description and atomic justification, the explanation of the above example is as follows:

A is subsumed by (**prim** Wine)

because **Prim** (θ = {Good, Wine}, α = {Wine})

A subsumption test in CLASSIC is implemented as a normalization process followed by a structural comparison process. The explanation first presents the last rule used in the deductive process and then answers automatically generated or user defined follow-up questions. The follow-up questions can be generated from the form of the inference rule used. In this example, the following question can be asked: why {Wine} is a subset of {Wine, Good} in the **Prim** rule? It refers to well-known partial orders in mathematics such as subset, thus no further explanation is needed.

The procedure to explain concept subsumption is also extended to explain instance check, using extra inference rules concerning propagations, closed world reasoning, rule firing and role closure. For example, if an individual a is discovered to be an instance of an atomic concept A and A is subsumed by another concept B, then a is asserted to be an instance of B.

The system explains contradictions with the aid of intermediate objects under the assumption that after detecting an initial contradiction, CLASSIC terminates all additional deductions and reverts to a previous stable state. So the explanation

facility provides a function to take the intermediate state of an object, identify which error inference was used and then ask the appropriate follow-up questions.

Furthermore, [McG96] suggests explaining non-subsumption by giving a counter example, since the system either determines that a concept subsumes another, or presents a model that contradicts the subsumption. For example, if $A \not\sqsubseteq B$, then there must exist an individual which is an instance of A but not B. They claim that it would be a convincing explanation to show this counter-example to users. It is also mentioned that this explanation is provided by tracing the internal reasoning procedure of the system.

As illustrated in the above example, the purpose of this approach is to *comprehend* the result of the system by *internally tracing* the reasoning procedure. The explanations are presented as *text* and users have a choice of selecting the level of details.

Being one of the first DL systems to obtain an explanation facility, [McG96] proposes to explain in a proof-theoretic framework which is also adopted in this thesis. However, since the expressive power of CLASSIC is limited, e.g., disjunction and full negation are not allowed, structural subsumption comparison can be exploited in the explanation procedure. For example, the subsuming concept can be decomposed into conjunct parts and compared with the subsumed concept. Once the DL language is extended to $\mathcal{ALC}$, this approach is no longer applicable.

$(X \sqcap \bot) \sqsubseteq Y$ $\qquad$ $(X \sqcap a \sqcap \neg a) \sqsubseteq Y$

$X \sqsubseteq (\top \sqcup Y)$ $\qquad$ $X \sqsubseteq (a \sqcup \neg a \sqcup Y)$

$(X \sqcap a) \sqsubseteq (a \sqcap Y)$

Figure 1: Terminal Conditions of the Sequent Rules

$\frac{X \sqcap a \sqsubseteq Y}{X \sqcap \neg\neg a \sqsubseteq Y}$ $(l\neg\neg)$ $\qquad$ $\frac{X \sqsubseteq a \sqcup Y}{X \sqsubseteq \neg\neg a \sqcup Y}$ $(r\neg\neg)$

$\frac{X \sqcap \neg a \sqcap \neg b \sqsubseteq Y}{X \sqcap \neg(a \sqcup b) \sqsubseteq Y}$ $(l\neg\vee)$ $\qquad$ $\frac{X \sqsubseteq \neg a \sqcup \neg b \sqcup}{X \sqsubseteq \neg(a \sqcap b) \sqcup Y}$ $(r\neg\wedge)$

$\frac{X' \sqcap b \sqsubseteq Y'}{X \sqcap \exists r.b \sqsubseteq Y}$ $(l\Diamond)$ $\qquad$ $\frac{X' \sqsubseteq b \sqcup Y'}{X \sqsubseteq \forall r.b \sqcup Y}$ $(r\Box)$

$\frac{X' \sqcap \neg b \sqsubseteq Y'}{X \sqcap \neg\forall r.b \sqsubseteq Y}$ $(l\neg\Box)$ $\qquad$ $\frac{X' \sqsubseteq \neg b \sqcup Y'}{X \sqsubseteq \neg\exists r.b \sqcup Y}$ $(r\neg\Diamond)$

where $X' = \{a | \forall r.a \in X\} \cup \{\neg a | \neg\exists r.a \in X\}$, and $Y' = \{a | \exists r.a \in Y\} \cup \{\neg a | \neg\forall r.a \in Y\}$

Figure 2: Modified Sequent Rules

3.2.2 Explaining Subsumption Using Sequent Rules

In order to overcome some of the above drawbacks, in [BFH+99], the authors propose to explain subsumption using a modified sequent calculus and the corresponding DL language is extended to $\mathcal{ALC}$. This approach was later extended to definitorial $\mathcal{ALEHFR}+$ TBoxes (with global domain/range restrictions) in [LH05] and implemented in the ontology editor OntoTrack [LN05]. The sequent rules are modified to imitate the behavior of a tableau calculus. For example, in order to avoid confusion, sequent rules are modified in such a way that formulae are never shifted from antecedents to consequents or vice versa.

The conditions to terminate an explanation and the modified inference rules used in this approach are indicated in Figures 1 and 2.

Using these inference rules, we can explain:

$$\exists hasFriend.\top \sqcap \forall hasFriend.\neg(\exists hasChild.\neg Doctor \sqcup \exists hasChild.Lawyer)$$

$$\sqsubseteq \exists hasFriend.\forall hasChild.(Doctor \sqcup Rich)$$

The first step of the sequent proof is the application of $(l\Diamond)$ rule, which leads to the following condition:

$$\top \sqcap \neg(\exists hasChild.\neg Doctor \sqcup \exists hasChild.Lawyer) \sqsubseteq \forall hasChild.(Doctor \sqcup Rich)$$

This step can be explained as follows: In order to check that the combination of a $\exists R.A$ concept and a $\forall R.B$ concept is subsumed by a $\exists R.C$ concept, we can check whether the conjunction of A and B is subsumed by C. Then, by applying the $(l\neg\vee)$ rule, we obtain the following:

$$\top \sqcap \neg(\exists hasChild.\neg Doctor) \sqcap \neg(\exists hasChild.Lawyer) \sqsubseteq \forall hasChild.(Doctor \sqcup Rich)$$

That is, by applying de Morgan's laws, we propagate negation inward.

Next, by applying the $(r\Box)$ rule, we obtain the following:

$$\neg\neg Doctor \sqcap \neg Lawyer \sqsubseteq Rich \sqcup Doctor$$

This step can be explained as follows: First apply de Morgan's laws to the two existential quantifiers at the left which yields $\forall Child.\neg\neg Doctor$ and $\forall Child.\neg Lawyer$. In order to check the combination of a $\forall R.A$ concept and a $\forall R.B$ concept is subsumed by a $\forall R.C$ concept, we can check whether the conjunction of A and B is subsumed by C. Next, by applying the $(l\neg\neg)$ rule, we obtain:

$$Doctor \sqcap \neg Lawyer \sqsubseteq Doctor$$

This step can be explained as follows: Apply the double negation rule.

And so we obtain the termination axiom:

$$Doctor \sqsubseteq Doctor$$

which can be explained as follows: Obviously, a concept subsumes itself.

The purpose of this approach is to *comprehend* the result (especially the subsumption result) of the system by *internally tracing* the reasoning procedure. The explanations are presented in *natural language style.*

One of the interesting perspectives of this approach is that there is a correspondence between tableau algorithms and the modified sequent rules, thus making the explanation closely related to the problem description. The only modification that should be made is to tag the subsumer, since the tableau works by negating the subsumer to prove its unsatisfiability. By tagging the subsumer, it is possible to determine whether a concept involved in a particular tableau extension corresponds to a part of the subsumer.

While this explanation technique is tied to the tableau algorithm, its main disadvantage is that most of the common tableau optimizations ([Hor07]) cannot be applied as they modify the structure of the asserted axioms, which the explanation technique is very sensitive to. Hence, the performance penalty on the explanation generation is huge. Another drawback we see with this approach is that although from a theoretical point of view, procedures to explain subsumption can be applied

to explain unsatisfiability too, because unsatisfiability can be reduced to subsumption, e.g., "C is unsatisfiable" is equivalent to "C is subsumed by $\bot$", practically it is not suitable to use this approach to detect and explain unsatisfiability or inconsistency. Besides, in this approach extra explanations are needed for the modified rules as many of them are apparently difficult for users to understand without further clarifications. Moreover, the resulting sequent rules are usually long and may contain many irrelevant axioms, thus making the explanation hard to understand.

3.2.3 Pinpointing and Debugging

In contrast to the earlier work, non-standard reasoning algorithms based on minimization of axioms are proposed in [SC03]. This approach detects the inconsistency in TBoxes by pinpointing unsatisfiable concepts and axioms. Definitions such as *minimal unsatisfiability-preserving sub-TBoxes* (MUPS) and *minimal incoherence-preserving sub-TBoxes* (MIPS) are introduced in this approach. MUPS-TBoxes are the smallest subsets of axioms of an inconsistent terminology preserving unsatisfiability of a particular concept. Similarly, MIPS-TBoxes are the smallest subsets of axioms of an inconsistent terminology preserving unsatisfiability of at least one unsatisfiable concept. *Generalized incoherence-preserving terminology* (GITS) are TBoxes where the defining concepts of the axioms are maximally generalized without losing inconsistency. More concretely, this approach first excludes axioms which are irrelevant to the inconsistency and then provides simplified definitions which

highlight the exact position of the contradiction. The DL language considered in [SC03] is $\mathcal{ALC}$, and the algorithm for computing all the minimal subsets is a modification of the tableau-based satisfiability algorithm for $\mathcal{ALC}$, with Labels added in the algorithm to keep track of axioms responsible for an assertion to be generated during tableau rule applications. The term "axiom pinpointing" was coined for computing these minimal subsets. As observed in real world applications, many unsatisfiable concepts are dependent on each other, this approach tries to find the "core" reason of the inconsistency. The *core* represents sets of axioms occurring in the most number of incoherent TBoxes.

To summarize, this approach aims to *debug* the unsatisfiability in the TBoxes by *tracing* the reasoning procedures. However, the proposed algorithms do not give any explanations for the inconsistency.

An analog to the exclusion of irrelevant axioms in our work is automatically done by the resolution proof technique, because all the literals in a resolution proof are contributing directly to the contradiction.

This work is later extended in [PSK05] to debug OWL ontologies. The purpose of this approach is *debugging* of unsatisfiability and inconsistency in an ontology by *tracing* the tableau reasoning procedures. Similar to its predecessor, this approach does not provide explanations of any kind. It distinguishes the unsatisfiable concepts from ontologies in the context of debugging. After the defects are detected, it tries to find the reasons for the unsatisfiability. Their approaches are divided

into two categories: glass box and black box. Glass box relies on information from internals of the reasoners. It traces clashes to give the core for inconsistency. In many cases, however, showing clashes alone is not specific enough to point out the source of unsatisfiability. Glass box needs to extend or to alter the reasoner, e.g., to keep track of the source axioms, thus efficiency is affected. Black box approach uses reasoners as oracles. It relies on users to perform navigational search to show unsatisfiability dependencies. Compared to this approach, our approach does not have to alter the reasoning procedure of reasoners. Besides, an unsatisfying explanation of unsatisfiability is that all sources are tried out but have all failed. Even if this was extended to include a trace of what was tried and why it failed, these traces may include many deduction paths and quickly become unwieldy.

The algorithms and proofs are given for $\mathcal{ALC}$ only in [SC03]. In order to investigate to which subsets of DL languages and tableau algorithms the approach can be applied without major changes, [BP07, BS08] develop a general approach for extending the tableau algorithm to a pinpointing algorithm and apply the algorithm to $\mathcal{EL}+$ ontologies, which allow for conjunction, existential restriction and complex role inclusion axioms.

In [Sch04], interpolations with particular syntactic and semantic properties are used to explain subsumption in $\mathcal{ALC}$. These interpolations act as intermediate inference steps between inference rules and help explain how one subsumption follows from another. This work is later extended in [HPS08, HPS09]. In the latter work, a

justification for an entailment in an OWL ontology is defined as a minimal subset of the ontology that is sufficient for that entailment to hold. Their approach focuses on taking justifications that are difficult to understand, and choosing subsets of these justifications that can be replaced with simpler summarizing entailments, such that the result is an easier to understand justification.

We also note that the Inference Web (IW) Infrastructure [MdS04] can be used to exchange and compare the explanations across multiple reasoners. It comprises of a web-based registry for information sources, reasoners, assumptions and learned information, a portable proof specification language (PML [dSMF06]) for exchanging explanations, and a browser to view and interact with proof explanations in different formats.

Table 5 summarizes the existing approaches for explanations of reasoning in DL, including our approach.

Approaches	Involved Reasoning	Underlying DL language
CLASSIC [McG96]	Subsumption and inconsistency	$\mathcal{ALQ}$ with TBoxes and ABoxes
Bordiga et al. [BFH+99]	Subsumption	$\mathcal{ALC}$ with TBoxes
Schlobach & Cornet [SC03]	Unsatisfiability	$\mathcal{ALC}$ with unfoldable TBoxes
Parsia et al. [PSK05]	Subsumption and unsatisfiability	$\mathcal{SHOIN}$ with TBoxes and ABoxes
Horridge et al. [HPS09]	Subsumption and unsatisfiability	$\mathcal{SROIQ}$ with TBoxes and ABoxes
Our approach [DHS07b]	Inconsistency and unsatisfiability	$\mathcal{ALCHI}$ with TBoxes and ABoxes

Table 5: Comparison of Approaches of Explanations for DL Reasoning

As we will show in the following chapters in this thesis, the main purpose of our approach to explanation is *comprehension and debugging* of the unsatisfiability and inconsistency in knowledge bases by *reconstructing* an explanation system. The explanations are presented in a *textual* style.

3.3 Explanation in Automated Theorem Proving

Explanation capabilities in Automated Theorem Proving (ATP) have been inadequate which impede their usability. Therefore, researchers in ATP have made efforts to transform machine-generated proofs into human understandable proofs.

3.3.1 $\mathcal{X}$-proof system

$\mathcal{X}$-proof [FM87] system uses the sequential variants of natural deduction to present proofs, which are then further transformed into natural language explanations. Natural deduction proof trees are constructed by placing the theorem to be proved at the root and choosing an inference rule that can be used to prove the theorem and making that a child of the root. The process is repeated based on the premises of the inference rule. Once a successful tree is produced, the system provides a means for presenting the tree in an English-like form. For example, consider the following formula:

$$\forall x \forall y \forall z (R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \wedge \forall x \forall y (R(x,y) \rightarrow R(y,x))$$

$$\rightarrow \forall x (\exists y R(x,y) \rightarrow R(x,x)).$$

The tree for this proof is:

```
implies (and_l(forall_r(implies_r(exists_l(forall_l(forall_l(forall_l
        (forall_l(forall_l(positive(and_r(positive(axiom(R(a,b)),
                                                  thin(axiom(R(b,a)))),
                                        thin(axiom(R(a,b)))),
                          thin(axiom(R(a,a))))))))))))))
```

This can be presented as follows:

Assume $\forall x \forall y \forall z(R(x,y) \wedge R(y,z) \rightarrow R(x,z)) \wedge \forall x \forall y(R(x,y) \rightarrow R(y,x))$. Also assume $\exists y R(a,y)$. Choose b such that $R(a,b)$ is true. By modus ponens, we have that $R(b,a)$. Hence, $R(a,b) \wedge R(b,a)$. By modus ponens, we have that $R(a,a)$. Since a was arbitrary, we conclude that $\forall x(\exists y R(x,y) \rightarrow R(x,x))$.

This work is first of its kind to build proof trees as explanation and provides a way to present proof trees in some natural language form. This idea of converting the proof trees to some natural language style text is adopted in our work as well.

3.3.2 PROVERB and TRAMP

PROVERB [Hua94, HF96] is an automated theorem prover used for mathematical problems. It uses a reconstructive approach to generate proofs logically equivalent to the original natural deduction proofs, but at a higher level of abstraction. In PROVERB, an inference step is often described in terms of the application of a

definition, an axiom, a lemma or a theorem, which can be called an assertion. Based on this, abstractions of natural deduction (ND) proofs have been defined, where a proof step may be justified either by a ND inference rule or by the application of an assertion.

In PROVERB justifications are provided at three levels:

- Logic level: verbalization of the ND inference rules, e.g., the rule of Modus Ponens.
- Assertion level: the application of an axiom, a definition, or a theorem.
- Proof level: reuse of a previous proof segment allows atomic justification at a higher level of abstraction.

The TRAMP system [Mei00] extends PROVERB by using refutation graphs to represent assertion applications. They claim that proofs in many other refutation based formalisms can be transformed easily into refutation graphs (e.g., in [Eis91] a transformation based algorithm for resolution proofs is described). They present an algorithm to translate steps at the assertion level from the refutation graphs into ND proofs.

The abstraction of ND proof to a higher level is based on the fact that there are lots of theorems, lemmas and axioms in the mathematical domain, which apparently is not the case in DL. Although refutation graphs are used in the transformation procedure, they are used as a form of presentation [Mei00]. In our approach, due to

the characteristics of DL, we exploit graph features for explanation and inconsistency detection.

3.4 Explanation in Other Systems

In deductive databases, explanations are often generated in three steps. The reasoning trace is firstly extracted from a source program or its execution, then it is filtered to be abstracted and finally presented to the users, often with a visualization tool [MD99]. The first explanation system of this kind was developed in [Wie90] for Dedex [MWW89]. This approach redesigned an independent inference system which generates a trace of proof tree constructions. Similar to this approach, the *Explain* subsystem was developed for CORAL [RSS92]. The system extracts the proof trees from the evaluation of the query. A visualization tool allows users to navigate among the trees. However, the source program is transformed by the magic set transformation [RU93] due to optimization purposes, and the proof trees are based on the transformed program.

There are two main groups of explanation methods for expert systems: the methods that cannot adapt their behavior to different user requirements and those that can. MYCIN [SCDS84, BS84], NEOMYCIN [Cla81, Cla94], Causal MYCIN [WS84] and XPLAIN [Swa83] fall into the first category. These systems cannot update the knowledge about the users during their execution and they focus on answering the why (is the system requiring this information) and how (did the

system derive the conclusion) questions posed by the users. On the other hand, [Moo94] and [Fie01] allow users to get involved in the explanation process. Users can ask the system questions to seek clarification. The system can also update its knowledge about the users, identify the objectives and refine the explanation strategies.

In the area of model checking, a model of a system is tested to automatically verify its correctness according to the system specification [CGP99]. One approach that tries to address this problem is counterexample guided abstraction-refinement (CEGAR) [CGJ+00]. It begins with a high level abstraction of the system, and refines it until either an abstraction proves the correctness of the system or a valid counterexample is generated. During this process, if a counter-example is found, the tool analyzes its validity. If it is valid, it is reported to the user. If it is not, the proof is used to refine the abstraction and checking begins again.

3.5 Diagnosis in Description Logics Systems

Following Reiter's approach for model-based diagnosis [Rei87], [Sch05] uses minimal subsets together with the Hitting Sets to compute maximal subsets of a given knowledge base that do not have a given (unwanted) consequence. While minimal subsets help users to comprehend why a certain consequence holds, maximal subsets that do not have the unwanted consequence suggest how to change the knowledge base in a minimal way to get rid of a certain unwanted consequence. [KPSG06, Kal06]

extend this approach by defining metrics for ranking axioms that contribute to inconsistencies and generate repair plans based on axiom ranks.

3.6 Conclusion

In this chapter, we reviewed works related to explanation and diagnosis in knowledge base systems. We started with describing the notion of explanation in the context of Description Logics as enhancing the comprehension of the knowledge base and help diagnose the design defects when an unexpected result has been encountered. We defined different aspects of an explanation based on purpose, presentation and methodology. Using these aspects, we then reviewed the previous related works in the DL community. We also explained the differences between these previous works and our approach.

On the other hand, we investigated the explanation approaches in other knowledge base systems, especially in automated theorem proving, and discussed how the insights achieved in these areas shed light on our work.

Finally, we reviewed the preliminary efforts that have been undertaken regarding diagnosing inconsistencies in DL knowledge bases.

Part II

An Explanation Framework Based on Resolution

This part includes two chapters, Chapter 4 proposes a resolution based framework to explain unsatisfiability and inconsistency for the Description Logic language $\mathcal{ALCHI}$. Implementation details of a prototype system based on this framework are discussed in Chapter 5.

Chapter 4

An Explanation Procedure

In this chapter, we present the resolution based framework to provide explanations for unsatisfiability and inconsistency queries in $\mathcal{ALCHI}$. We also present our explanation algorithm based on inconsistency patterns and establish its soundness and completeness.

4.1 Resolution Based Framework

The proposed explanation framework consists of three components, described as follows.

1. Preprocessing: The explanation system communicates with a DL reasoner, e.g., Racer, which provides the answer for a query, normally in the form of "Yes" or "No" for a concept satisfiability or an ABox consistency query, or a list of unsatisfiable concepts for a TBox consistency query. If the answer is

"No", which means a concept is unsatisfiable or a TBox/ABox is inconsistent, then the original TBox/ABox will be translated into FOL formulae or clauses by the translation component.

2. Rendering resolution proofs: A resolution based automated theorem prover is applied to the FOL formulae or clauses resulted from step 1 in order to generate resolution proofs. It is important to note that since a DL reasoner is used first to retrieve the result of the query, the unsatisfiability of the concept or the inconsistency of the TBox/ABox is already known. The complexity of the resolution based decision procedure of the guarded fragment of FOL is double exponential in the worst case [GdN99]. Since $\mathcal{ALCHI}$ can be embedded into the guarded fragment, its upper bound of the computational complexity is double exponential too.

3. Explaining: The resolution proof is used by the explanation kernel to construct explanations for better human understanding. Our approach transforms the proof into its corresponding refutation graph [Eis91], which is a more abstract representation for the refutation proof. A refutation graph is a graph whose nodes are literals (grouped together in clauses) and its edges connect complementary literals (an atom and its negation). For each resolution proof, there is a minimal refutation graph, but one refutation graph can represent multiple different sequences of resolution steps. Hence we can dynamically search it to find one of the sequences as explanations. Finally, the clauses involved in each

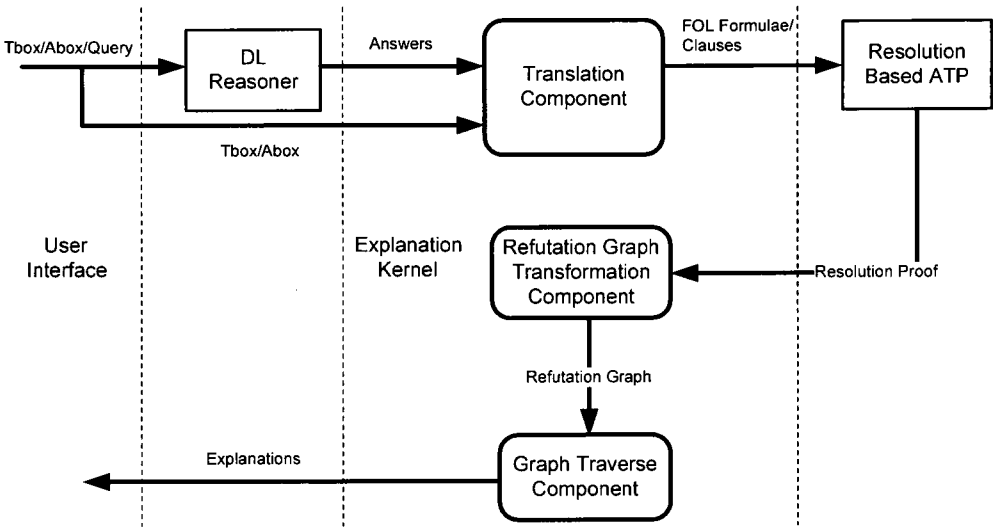

Figure 3: The Explanation Framework Architecture

resolution step are traced back to the contributing DL axioms/assertions and possibly transformed further into natural language explanations.

The architecture of the explanation process is shown in Figure 3. In the subsequent sections, we will discuss each of the components in the figure in detail.

4.1.1 Preprocessing

In the preprocessing phase, DL axioms and assertions are transformed into FOL formulae or clauses and a structural transformation is adopted.

To illustrate this procedure, we consider the following example of a TBox. Suppose we know from the DL reasoner that the concept $A1$ is unsatisfiable.

Example 4.1.1

1. $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$
2. $A2 \sqsubseteq A \sqcap A4$

Translation from DL to FOL

The translation from DL to FOL is based on the semantics of DL. For $\mathcal{ALCHI}$, concepts can be translated into $\mathcal{L}^2$ [HPSMW07], which is a first order predicate logic over unary and binary predicates with two variables, say x, y. Tables 6, 7 and 8 show this translation from $\mathcal{ALCHI}$ into $\mathcal{L}^2$. An atomic concept A is translated into a predicate logic formula by a translation mapping π with one free variable x such that for every interpretation $\mathcal{I}$, the set of elements of $\Delta^{\mathcal{I}}$ satisfying $\pi_x(A)$ is exactly $A^{\mathcal{I}}$. Similarly, a role name R is translated into a binary predicate $R(x, y)$. An individual name a is translated into the constant a.

$\pi_x(A) =$	$A(x)$
$\pi_y(A) =$	$A(y)$
$\pi_x(\neg C) =$	$\neg\pi_x(C)$
$\pi_x(C \sqcap D) =$	$\pi_x(C) \wedge \pi_x(D)$
$\pi_x(C \sqcup D) =$	$\pi_x(C) \vee \pi_x(D)$
$\pi_x(\exists R.C) =$	$\exists y(R(x, y) \wedge \pi_y(C))$
$\pi_x(\forall R.C) =$	$\forall y(R(x, y) \rightarrow \pi_y(C))$

Table 6: Translation from $\mathcal{ALCHI}$ Concepts into $\mathcal{L}^2$

$\pi(C \sqsubseteq D) =$	$\forall x(\pi_x(C) \rightarrow \pi_x(D))$
$\pi(C \equiv D) =$	$\forall x(\pi_x(C) \leftrightarrow \pi_x(D))$
$\pi(R \sqsubseteq S) =$	$\forall x, y(R(x, y) \rightarrow S(x, y))$
$\pi(C(a)) =$	$\pi_x(C)[x/a]$
$\pi(R(a, b)) =$	$R(a, b)$

Table 7: Translation from $\mathcal{ALCHI}$ Axioms into $\mathcal{L}^2$

$\pi(R) =$	$\forall x, y(R(x,y) \leftrightarrow R^{-}(y,x))$
$\pi(\mathcal{T}) =$	$\bigwedge_{\alpha \in \mathcal{T}} \pi(\alpha)$
$\pi(\mathcal{A}) =$	$\bigwedge_{\alpha \in \mathcal{A}} \pi(\alpha)$

Table 8: Translation from $\mathcal{ALCHI}$ Knowledge Base into $\mathcal{L}^2$

These translations preserve the semantics: after the translation, the DL knowledge base is equisatisfiable with its corresponding FOL knowledge base.

Definition 4.1.1 *Let $\mathcal{T}$ be a TBox and $\mathcal{A}$ an Abox in $\mathcal{ALCHI}$, C, D be concepts and a an individual.*

- *$(\mathcal{T}, \mathcal{A})$ is consistent iff $\pi(\mathcal{T}) \wedge \pi(\mathcal{A})$ is consistent*
- *$(\mathcal{T}, \mathcal{A}) \models C \sqsubseteq D$ iff $(\pi(\mathcal{T}) \wedge \pi(\mathcal{A})) \rightarrow (\pi(\{C \sqsubseteq D\}))$ is valid*
- *$(\mathcal{T}, \mathcal{A}) \models a : C$ iff $(\pi(\mathcal{T}) \wedge \pi(\mathcal{A})) \rightarrow (\pi(\{C(a)\}))$ is valid*

According to the translation rules, the resulting clauses of Example 4.1.1 are:

$\mathbf{C_{11}} = \{\neg A1(x), A2(x)\}$
$\mathbf{C_{12}} = \{\neg A1(x), \neg A(x)\}$ $\quad$ $\mathbf{C_{13}} = \{\neg A1(x), A3(x)\}$
$\mathbf{C_{21}} = \{\neg A2(x), A(x)\}$ $\quad$ $\mathbf{C_{22}} = \{\neg A2(x), A4(x)\}$

In the case of TBox inconsistency queries, there are two possible scenarios: a set of unsatisfiable concepts or an inconsistent TBox, i.e., the top concept is unsatisfiable thus causing all the concepts in the TBox to be unsatisfiable. In the case of ABoxes inconsistency queries, there are also two possible scenarios: an individual

is asserted as an instance of an unsatisfiable concept or it is asserted to belong to an unsatisfiable concept description. Consequently, these cases are distinguished in the translation procedure.

Definition 4.1.2 *Let $\mathcal{T}$ be a TBox, and C be an unsatisfiable concept in $\mathcal{T}$. An unsatisfiability translation function Γ w.r.t. C translates $\mathcal{T} \cup \{C\}$ into a set of FOL clauses.*

For example, suppose $\mathcal{T} = \{\mathsf{Person} \sqsubseteq \mathsf{Man} \sqcap \neg\mathsf{Man}\}$. The concept Person is unsatisfiable, so Person and the axiom in $\mathcal{T}$ form the input set of Γ.

Definition 4.1.3 *Let $\mathcal{T}$ be a TBox and $\mathcal{A}$ be an ABox ($\mathcal{T}$ and/or $\mathcal{A}$ can be empty). An inconsistency translation function Λ translates $\mathcal{T} \cup \mathcal{A}$ into a set of FOL clauses.*

For example, suppose the TBox $\mathcal{T}$ is $\{\mathsf{Person} \sqsubseteq \bot, \top \sqsubseteq \mathsf{Person}\}$. Here, $\mathcal{T}$ is inconsistent and it includes two axioms as the input set of Λ.

Since $A1$ is unsatisfiable in Example 4.1.1, according to the unsatisfiability translation function, $A1(c)$ is added to the input set of Γ, with c being a fresh constant.

$\mathbf{C_0}$=$\{A1(c)\}$
$\mathbf{C_{11}}$= $\{\neg A1(x), A2(x)\}$
$\mathbf{C_{12}}$= $\{\neg A1(x), \neg A(x)\}$ $\quad$ $\mathbf{C_{13}}$ = $\{\neg A1(x), A3(x)\}$
$\mathbf{C_{21}}$= $\{\neg A2(x), A(x)\}$ $\quad$ $\mathbf{C_{22}}$ = $\{\neg A2(x), A4(x)\}$

Structural Transformation

A straightforward translation as shown in Tables 6, 7 and 8 followed by skolemization (a method for removing existential quantifiers from FOL formulae [BG01]) and transformation into conjunctive normal form would easily result in the exponential blow up of the number of the clauses. Consider the axiom $E \sqsubseteq F$, where E and F are complex concept descriptions. If n and m are the numbers of clauses generated by E and F respectively, then the above formula generates $n \times m$ clauses. The reason for the exponential explosion is duplication of subformulae obtained by the exhaustive application of the distributive law. In order to avoid such blow-ups, we adopt the standard translation augmented by *structural transformation* as in [HMS05]. Roughly speaking, the structural transformation is a kind of conjunctive normal form (CNF) transformation of first-order predicate logic formulae by replacing the subformulae with some new predicates and adding suitable definitions for these predicates. In the previous example, if we replace F by a fresh concept, say C, then the above axiom transforms into two: $E \sqsubseteq C$ and $C \sqsubseteq F$. The number of clauses generated by these two axioms is $n + m$. Besides, the structural transformation also helps preserve original structures of DL axioms after their corresponding first-order logic formulae are transformed into conjunctive normal forms. For instance, consider the axiom $\forall R.A \sqsubseteq \exists S.B$. Without the transformation, the subsumee and subsumer of this axiom are distributed into four clauses, $\{R(x, f_1(x)), S(x, f_2(x))\}$,

$\{R(x, f_1(x)), B(f_2(x))\}$, $\{S(x, f_2(x)), \neg A(f_1(x))\}$, and $\{\neg A(f_1(x)), B(f_2(x))\}$, making it difficult to generate comprehensible explanations.

The structural transformation can be formally described as follows. More details can be found in [NRW98]. The basic idea is to replace non-atomic concepts, which are subformulae of the axioms, with new concept names. The notion of formula positions are then used to define the transformation. For example, let φ be a formula, $\phi = \varphi|_\pi$ is a subformula of φ at position π that we want to replace. The transformation will replace ϕ with a predicate new to φ, say R.

Definition 4.1.4 ([NRW98]) *A position is a word over the natural numbers. The set $pos(\varphi)$ of positions of a given formula φ is defined as follows:*

- *the empty word $\varepsilon \in pos(\varphi)$,*
- *for $1 \leq i \leq n$, $i.p \in pos(\varphi)$ if $\varphi = \varphi_1 \circ \ldots \circ \varphi_n$ and $p \in pos(\varphi_i)$, where $\circ$ is a first-order operator. If $p \in pos(\varphi)$, then $\varphi|_{i.p} = \varphi_i|_p$, where $\varphi = \varphi_1 \circ \ldots \circ \varphi_n$. We write $\varphi[\phi]_p$ for $\varphi|_p = \phi$. With $\varphi[p/\phi]$ where $p \in pos(\varphi)$ we denote the formula obtained by replacing $\varphi|_p$ with ϕ at position p in φ. The polarity of a formula φ at position π is denoted by $Pol(\varphi, \pi)$ and defined as: $Pol(\varphi, \varepsilon) = 1$; $Pol(\varphi, \pi.i) = Pol(\varphi, \pi)$ if $\varphi|_\pi$ is a conjunction, disjunction, or formula starting with a quantifier, or an implication with $i = 2$; $Pol(\varphi, \pi.i) = -Pol(\varphi, \pi)$ if $\varphi|_\pi$ is a formula starting with a negation symbol or an implication with $i = 1$, and $Pol(\varphi, \pi.i) = 0$ if $\varphi|_\pi$ is an equivalence.*

Definition 4.1.5 ([NRW98]) *Let φ be a formula and $\phi = \varphi|_\pi$ be a subformula of φ at position π. Let $x_1, \ldots, x_n$ be the free variables in φ and let Q be a new predicate. Then the formula*

$$\varphi[\pi/Q(x_1, \ldots, x_n)] \wedge Def_\pi^\varphi$$

is a structural transformation of φ at position π. The formula Def_π^φ is a polarity dependent definition of the new predicate Q:

$$Def_\pi^\varphi = \begin{cases} \forall x_1, \ldots, x_n[Q(x_1, \ldots, x_n) \rightarrow \phi] & if Pol(\varphi, \pi) = 1 \\ \forall x_1, \ldots, x_n[\phi \rightarrow Q(x_1, \ldots, x_n)] & if Pol(\varphi, \pi) = -1 \\ \forall x_1, \ldots, x_n[\phi \leftrightarrow Q(x_1, \ldots, x_n)] & if Pol(\varphi, \pi) = 0 \end{cases}$$

There are six types of clauses in $\mathcal{ALCHI}$ after normalization:[1]

1. $\bigvee X_i$
2. $\bigvee X_i \vee R(x, f(x))$
3. $\bigvee X_i \vee Y$
4. $\bigvee X_i \vee \neg R(x, y) \vee Z$
5. $\neg R(x, y) \vee (R^-)(y, x)$
6. $\neg R(x, y) \vee S(x, y)$

where $X_i \in \{C_i(x), \neg C_i(x)\}$, $Y \in \{D(f(x)), \neg D(f(x))\}$, and $Z \in \{D(y), \neg D(y)\}$.

Specifically, the clause type (1) is transformed from axioms $C_i \sqsubseteq C_j$, with both C_i and C_j being complex concepts. Types (2) and (3) are transformed from axioms $C \sqsubseteq \exists R.D$ or $\forall R.C \sqsubseteq D$. Type (4) is transformed from axioms $C \sqsubseteq \forall R.D$ or $\exists R.C \sqsubseteq D$. Type (5) is the result of transforming the definition of inverse roles. Type (6) is transformed from role axioms of the form $R \sqsubseteq S$.

[1] We only consider TBoxes here. ABoxes cases have similar syntactic properties.

We thus have the following theorem which is the basis of the correctness of the translation.

Theorem 4.1.1 *Let $\mathcal{T}$ be a consistent TBox in $\mathcal{ALCHI}$ and C be a named concept in $\mathcal{T}$. Then C is unsatisfiable if and only if the empty clause is derived using resolution given the unsatisfiability translation function Γ w.r.t. C.*

Proof. As proved in [HMS05], the structural transformation does not affect satisfiability. Let $\Theta(\mathcal{T})$ and $\Theta(C(a))$ be the resulting set of FOL formulae of $\mathcal{T}$ and $C(a)$ after the translation, where a is a newly introduced individual. As $\mathcal{T}$ is given as consistent, $\Theta(\mathcal{T})$ is also consistent. Since C is unsatisfiable, C does not admit any instance, i.e., $C(a)$ is inconsistent. Hence $\Theta(\mathcal{T}) \cup \Theta(C(a))$ is inconsistent. Due to completeness of refutation resolution, the empty clause can be derived.

On the other hand, if the empty clause is derived from $\Theta(\mathcal{T}) \cup \Theta(C(a))$, then $\Theta(\mathcal{T}) \cup \Theta(C(a))$ is inconsistent. Since $\mathcal{T}$ is given as consistent, C must be unsatisfiable. ■

The following result can also be obtained due to the consistency preserving property of the translation.

Theorem 4.1.2 *Let $\mathcal{T}$ be a TBox and $\mathcal{A}$ be an ABox ($\mathcal{T}$ and/or $\mathcal{A}$ can be empty). Then $\mathcal{T} \cup \mathcal{A}$ is inconsistent if and only if the empty clause is derived by resolution, given the inconsistency translation function $\Lambda(\mathcal{T} \cup A)$.*

Proof. As proved in [HMS05], the structural transformation does not affect satisfiability. If $\mathcal{T} \cup \mathcal{A}$ is inconsistent, then $\Lambda(\mathcal{T} \cup A)$ is inconsistent, which means that the empty clause can be derived. On the other hand, if the empty clause is derived from $\Lambda(\mathcal{T} \cup A)$, then it is inconsistent. Consequently, $\mathcal{T} \cup \mathcal{A}$ is inconsistent. ■

4.1.2 Obtaining Resolution Proofs

After the translation step, a resolution based theorem prover is used to generate resolution proofs.

Table 9 shows one of the possible resolution proofs. The numbers shown in the second column of each row indicate the hypotheses it depends on. The inference rule that justifies a row is given after the conclusion formula, followed by the premise rows.

No	Hyp.	Clause	Reason
1.	1	$\{A1(c)\}$	(Hypothesis)
2.	2	$\{\neg A1(x), A2(x)\}$	(Hypothesis)
3.	3	$\{\neg A1(x), \neg A(x)\}$	(Hypothesis)
4.	4	$\{\neg A2(x), A(x)\}$	(Hypothesis)
5	1, 2	$\{A2(c)\}$	(Resolution)
6.	1, 3	$\{\neg A(c)\}$	(Resolution)
7.	5, 6, 4	$\bot$	(Resolution)

Table 9: A Resolution Proof for the Working Example

Since the resolution technique operates on clauses, which are on a finer-grained level than the original DL axioms, it can determine not only which axioms are

relevant but also which parts of the asserted axioms are relevant for the particular unsatisfiability or inconsistency problem. In this example, the conjuncts $A3$ in Axiom 1 and $A4$ in Axiom 2 are irrelevant for the unsatisfiability of A, hence their corresponding clauses do not appear in the resolution proof. In order to reflect this characteristics in the explanation, we define the minimal DL axiom counterparts of clauses as the *clausal axioms*, shown as follows.

Definition 4.1.6 *A clausal axiom is inductively defined as follows, where C_i, C_j and C_k are complex concepts, D is an atomic concept, $X_i \in \{C_i(x), \neg C_i(x)\}$, and $\otimes$ stands for $\sqcap$ or $\sqcup$.*

- *if the original DL axiom of a clause $\bigvee X_i \vee D$ is $C_i \sqsubseteq D \sqcap C_j$, then its clausal axiom is $C_i \sqsubseteq D$;*
- *if the original DL axiom of a clause $\bigvee X_i \vee \neg D$ is $D \sqcup C_i \sqsubseteq C_j$, then its clausal axiom is $D \sqsubseteq C_j$;*
- *if the original DL axiom of a clause $\bigvee X_i \vee R(x, f(x))$ is $C_i \sqsubseteq \exists R.C_j \otimes C_k$ (or $\forall R.C_i \otimes C_k \sqsubseteq C_j$) , then its clausal axiom is $C_i \sqsubseteq \exists R.\top \otimes C_k$ (or $\forall R.\top \otimes C_k \sqsubseteq C_j$).*

For example, the clausal axiom of the clause $\{\neg A1(x), A2(X)\}$ is $A1 \sqsubseteq A2$ in Example 4.1.1. The original DL axiom $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$ is split into smaller parts after tracing back to the clausal axioms of its clauses. The clausal axioms are

more fine-grained than the original DL axioms and they will be used in generating the explanations.

4.1.3 Generating Explanations

No	Hyp.	Clause	Reason
1.	1	$\{A1(c)\}$	(Hypothesis)
2.	2	$\{\neg A1(x), A2(x)\}$	(Hypothesis)
3.	3	$\{\neg A1(x), \neg A(x)\}$	(Hypothesis)
4.	4	$\{\neg A2(x), A(x)\}$	(Hypothesis)
5	3, 4	$\{\neg A1(x), \neg A2(x)\}$	(Resolution)
6.	2, 5	$\{\neg A1(x)\}$	(Resolution, factoring)
7.	1, 6	$\bot$	(Resolution)

Table 10: Another Resolution Proof for the Working Example

One of the potential obstacles of understanding resolution proofs is that several different resolution proofs can be obtained due to different application orders of the inference rules. Table 10 shows such a possible alternative proof for our working example. Note that compared to Table 9, clauses 3 and 4 resolve first before clauses 1 and 2, resulting in a different resolution proof. Since some resolution proofs are easier to understand than the others, the resolving order plays an important role in generating good explanations. For instances, in Table 9, the resolving of clauses 1 and 2 can be easily explained by their corresponding DL axioms $A1 \sqsubseteq A2$. However, the resolving of clauses 3 and 4 in Table 10 is less clear for general users. In order to solve this problem, we propose to further transform the generated resolution proof

into its refutation graph representation to construct explanations. A refutation graph is based on a set of *literal nodes*, which are nodes labeled with literals. These literal nodes are grouped together to *clause nodes*, which represent multisets of literals (different literal nodes may be labeled with the same literal), i.e., clauses. In the refutation graph, literal nodes are represented as small boxes labeled with their literals. Adjacent boxes denote a clause, i.e., the disjunction of the literals in the boxes. The motivation of using a refutation graph is that there is one refutation graph presentation for several resolution proofs. By traversing the graph, a good way to read the proof can be found.

Refutation Graphs

We will use the following definitions of refutation graphs in this thesis. Please see [Eis91] for more details.

Definition 4.1.7 ([Eis91]) *A refutation graph is a quadruple* $\mathcal{G} = (\mathcal{L}, \mathcal{C}, \mathcal{M}_{\mathcal{L}}, \mathcal{K})$, *where* $\mathcal{L}$ *is a finite set of literal nodes in* $\mathcal{G}$, *and* $\mathcal{C}$ *is a partition of the set of literal nodes whose members are clause nodes in* $\mathcal{G}$. $\mathcal{M}_{\mathcal{L}}$ *is a mapping from* $\mathcal{L}$ *to a set of literals, which associates with every literal node a literal. The set of links* $\mathcal{K}$ *is a partition of a subset of* $\mathcal{L}$. *All the literal nodes in one link are labeled with literals which are unifiable. There is no pure literal node in a refutation graph, i.e., every literal node belongs to some link in* $\mathcal{K}$.

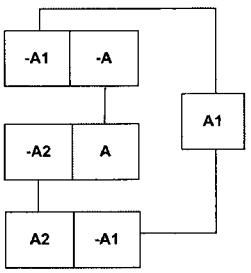

Figure 4: A Example Refutation Graph

The refutation graph of the example in Section 4.1.2 is shown in Figure 4. We extend the above definition of refutation graph to provide explanations for DL reasoning.

Definition 4.1.8 *A labeled refutation graph is a quintuple* $\mathcal{G}' = (\mathcal{L}, \mathcal{C}, \mathcal{M}_{\mathcal{L}}, \mathcal{K}, \mathcal{M}_{\mathcal{D}})$, *where* $\mathcal{L}, \mathcal{C}, \mathcal{M}_{\mathcal{L}}$ *and* $\mathcal{K}$ *are defined as in the refutation graph* $\mathcal{G}$ *above and* $\mathcal{M}_{\mathcal{D}}$ *is a mapping from* $\mathcal{L}$ *to a first-order formula, which labels the literal nodes with their originating first-order formulae.*

In our example, one of the mappings in $\mathcal{M}_{\mathcal{D}}$ is as follows, which links a literal node $\neg A1(x)$ to its first-order formula $\forall x\ A1(x) \rightarrow A2(x) \wedge \neg A(x) \wedge A3(x)$:

$$\{\neg A1(x)\} \mapsto \{\forall x\ A1(x) \rightarrow A2(x) \wedge \neg A(x) \wedge A3(x)\}$$

Definition 4.1.9 *A bridge in a refutation graph connects two sets of complementary literal nodes, which are involved in a resolution step. A traversal ordering is a partial ordering* $\leq$ *over the bridges. A factoring link (f-link for short) connects the literals that participate in a factoring step.*

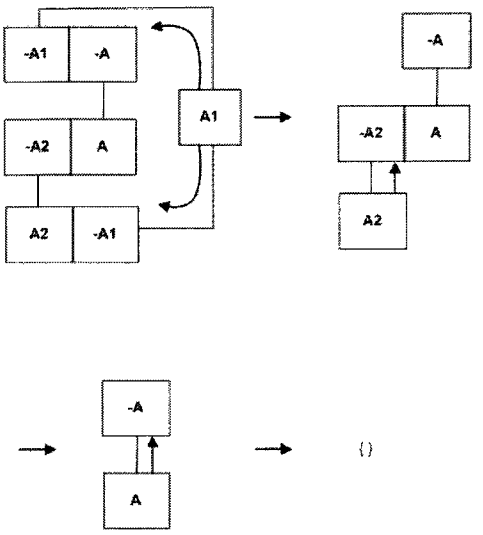

Figure 5: Derivation Represented by the Ordering in a Refutation Graph

Intuitively a refutation graph represents a way to derive the empty clause node. The derivation can be obtained by successively removing a bridge together with its literal nodes and uniting the resolvent of the two incident clause nodes to a single clause node. Since the choice of the bridges is arbitrary, the graph represents a whole class of resolution proofs that may differ only in reorderings of the inference steps. Figure 5 shows one of the derivations of the example as in Table 9. The arrows in the figure show the traversal ordering over the bridges.

Inconsistency Patterns

We believe that the quality of explanations largely depends on how the refutation graph is traversed, i.e., on how $\leq$ is defined. And the ordering can be decided according to the unsatisfiability and inconsistency patterns. Generally speaking, an

inconsistency pattern (i-pattern for short) is a resolving step over the concept names and the role names. An i-pattern helps decide the *traversal ordering* $\leq$ among the bridges.

Definition 4.1.10 *The set of* i-***patterns*** *over* $\mathcal{ALCHI}$ *is defined as one of the following cases:*

Pattern 1. $\exists R.\top$, $\forall R.C_i$, *and* $\bigcap C_i = \bot$, *where* $i = 1, \ldots, n$.

The simplified version presented in the form of a refutation graph is shown in Figure 6. The dotted line shows possible intermediate resolving steps between the two literals in bold boxes. In this case, all the bridges connecting $R(x, f(x))$ *and* $\neg R(x, y)$ *have the same traversal order, which is higher than the order of the bridges connecting the literal nodes* $C_i(y)$.

Pattern 2. $\exists R.D$, $\forall R.C_i$, *and* $D \sqcap \bigcap C_i = \bot$, *where* $i = 1, \ldots, n$.

The simplified refutation graph is shown in Figure 7. Similarly, the bridge connecting $R(x, f(x))$ *and* $\neg R(x, y)$ *has a higher order than bridges connecting* $C_i(y)$ *and* $D(f(x))$.

Pattern 3. $\bigcap C_i = \bot$, *where* $i = 1, \ldots, n$.

In this case, every C_i *belongs to different clause nodes. All the bridges have the same traversal order. The simplified refutation graph is shown in Figure 8.*

Pattern 4. $\bigcup C_i$, *where each* C_i *is an* i*-pattern, for* $i = 1, \ldots, n$.

All the bridges adjacent to C_i have the same traversal order. The simplified refutation graph is shown in Figure 9.

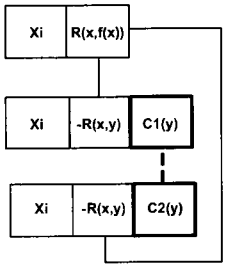

Figure 6: Inconsistency Pattern 1

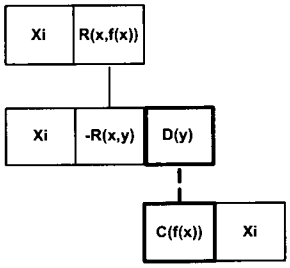

Figure 7: Inconsistency Pattern 2

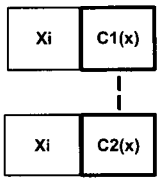

Figure 8: Inconsistency Pattern 3

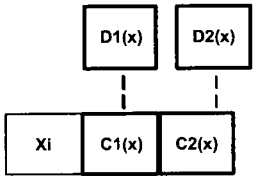

Figure 9: Inconsistency Pattern 4

The ordering of the bridges in the refutation graph is based on the i-patterns. A positive integer is set to be the initial value of each of the bridges in the graph. Then the graph is inspected to match the i-patterns. If a subgraph is matched with pattern 1 or 2, the involved bridges increase the ordering value by 1. If a subgraph is matched with pattern 3 or 4, the involved bridges' ordering values remain unchanged.

4.2 The Algorithm

After the traversal ordering of the bridges in the refutation graph is decided, the *traverse algorithm* is applied to generate explanations. Our explanation algorithm

is based on the refutation graph. It starts from a literal node N_0 and traverses the graph. Among all the bridges adjacent to N_0, it chooses the one(s) that are not lower than any others w.r.t. $\leq$. Then it chooses all the literal nodes N_i on the other side of the chosen bridges from N_0. Among each of the clause nodes that N_i resides in, it chooses the yet-to-be-chosen literal nodes. The traversal process ends when all the nodes in the graph have been chosen. After the traversal is completed, the clause nodes involved in each step are translated into an entry in an explanation list (EL) consisting of their clausal axioms in DL. After some clean-up process, such as deleting duplicate lines, the explanation list can be further transformed into natural language style explanations. This algorithm is formally stated in Figure 10. It uses a stack, called List of Traversal (LOT, for short), which includes the literal nodes which are yet to be traversed.

The following theorem establishes the soundness and completeness of the traversal algorithm. The completeness in our context means that at the end of the algorithm, no literal node is left untraversed.

Theorem 4.2.1 *The traversal algorithm is sound, complete and will terminate with an explanation.*

Proof. Termination: In each step of the traversal, the number of literal nodes that remain untraversed decreases, since once a literal node is traversed, it will not be traversed again. As the number of literal nodes in a refutation graph is finite,

Algorithm **TRAVERSE**
Input: a refutation graph $\mathcal{G}$, a query $\mathcal{Q}$
Output: explanation list EL

Set EL to be empty
if the query $\mathcal{Q}$ is an unsatisfiable query
  start from the unsatisfiable concept $\mathcal{C}$ in $\mathcal{G}$
else start from an arbitrary concept $\mathcal{C}$ in $\mathcal{G}$
add the associated literal node of $\mathcal{C}$ to LOT
for all the literal nodes L_i in LOT
  put the corresponding clausal axiom of L_i into the explanation list EL;
  for all the unvisited bridges which are adjacent to L_i, with the highest traversal order
    for all the literal nodes L_k that are in the same clause node as the other side of the bridge
      add L_k to LOT
    tag the bridge as visited
  remove L_i from LOT
return the explanation list EL

Figure 10: Traversal algorithm

the traversal algorithm will terminate. More precisely, since the traversal step is applied once for each literal node, i.e., the number of the traversal steps is linear in the number of the literal nodes.

Soundness: Since every literal node is visited exactly once, and all the literal nodes in the refutation graph are translated from the axioms that contribute to the unsatisfiability and inconsistency, the traversal algorithm will present these culprits in the explanations.

Completeness: The fact that we cannot reach a blocked situation follows upon the fact that every literal node in the refutation graph has a complementary literal node connected by a link, i.e., every literal node is reachable through other nodes.

■

4.3 Illustrating Example

In this section, we illustrate the explanation algorithm, using the following input example KB.

1. $Physician \sqsubseteq \exists hasDegree.BS$
2. $HappyPerson \sqsubseteq Doctor \sqcup \exists hasChild.(PhD \sqcap \neg Poor)$
3. $\exists hasParent.HappyPerson \sqsubseteq Married$
4. $MD \sqsubseteq \neg BS$
5. $Married \sqsubseteq Person \sqcap \exists hasSpouse.Person$
6. $Doctor \sqsubseteq \forall hasDegree.MD \sqcap Physician$
7. $PhD \sqcap Married \sqsubseteq Poor$
8. $hasParent \equiv hasChild^{-}$

Table 11: An Example DL Knowledge Base

Assume that we know from a DL reasoner that $HappyPerson$ is unsatisfiable. Consequently, the KB is augmented with $HappyPerson(a)$ where a is a fresh individual. We call the resulting knowledge base KB', shown bellow. The goal now is to show that KB' is inconsistent.

1. $Physician \sqsubseteq \exists hasDegree.BS$
2. $HappyPerson \sqsubseteq Doctor \sqcup \exists hasChild.(PhD \sqcap \neg Poor)$
3. $\exists hasParent.HappyPerson \sqsubseteq Married$
4. $MD \sqsubseteq \neg BS$
5. $Married \sqsubseteq Person \sqcap \exists hasSpouse.Person$
6. $Doctor \sqsubseteq \forall hasDegree.MD \sqcap Physician$
7. $PhD \sqcap Married \sqsubseteq Poor$
8. $hasParent \equiv hasChild^{-}$
9. $HappyPerson(a)$

Axioms 1, 2, 3, 5 and 6 contain non-literal subconcepts. But as the structural transformation does not decrease the number of the clauses nor does it simplify the explanations for 1, 3, 5 and 6, we only show how axiom 2 is converted to

FOL formulae based on structural transformation. We introduce a new name Q for this subconcept and obtain $HappyPerson \sqsubseteq Doctor \sqcup \exists hasChild.Q$ and $Q \sqsubseteq PhD \sqcap \neg Poor$ to replace $HappyPerson \sqsubseteq Doctor \sqcup \exists hasChild.(PhD \sqcap \neg Poor)$.

Hence, the knowledge base KB' after the structural transformation is as follows.

---

1. $Physician \sqsubseteq \exists hasDegree.BS$
2.1. $HappyPerson \sqsubseteq Doctor \sqcup \exists hasChild.Q$
2.2. $Q \sqsubseteq PhD \sqcap \neg Poor$
3. $\exists hasParent.HappyPerson \sqsubseteq Married$
4. $MD \sqsubseteq \neg BS$
5. $Married \sqsubseteq Person \sqcap \exists hasSpouse.Person$
6. $Doctor \sqsubseteq \forall hasDegree.MD \sqcap Physician$
7. $PhD \sqcap Married \sqsubseteq Poor$
8. $hasParent \equiv hasChild^{-}$
9. $HappyPerson(a)$

The set of clauses in KB' after normalization is shown as follows.

1.1. $\neg Physician(x) \vee hasDegree(x, f_1(x))$
1.2. $\neg Physician(x) \vee BS(f_1(x))$
2.1. $\neg HappyPerson(x) \vee Doctor(x) \vee hasChild(x, f_2(x))$
2.2. $\neg HappyPerson(x) \vee Doctor(x) \vee Q(f_2(x))$
2.3. $\neg Q(x) \vee PhD(x)$
2.4. $\neg Q(x) \vee \neg Poor(x)$
3.1. $\neg hasParent(x, y) \vee \neg HappyPerson(y) \vee Married(x)$
4.1. $\neg MD(x) \vee \neg BS(x)$
5.1. $\neg Married(x) \vee Person(x)$
5.2. $\neg Married(x) \vee hasSpouse(x, f_3(x))$
5.3. $\neg Married(x) \vee Person(f_3(x))$
6.1. $\neg Doctor(x) \vee \neg hasDegree(x, y) \vee MD(y)$
6.2. $\neg Doctor(x) \vee Physician(x)$
7.1. $\neg PhD(x) \vee \neg Married(x) \vee Poor(x)$
8.1. $\neg hasChild(x, y) \vee hasParent(y, x)$
8.2. $\neg hasParent(y, x) \vee hasChild(x, y)$
9. $HappyPerson(a)$

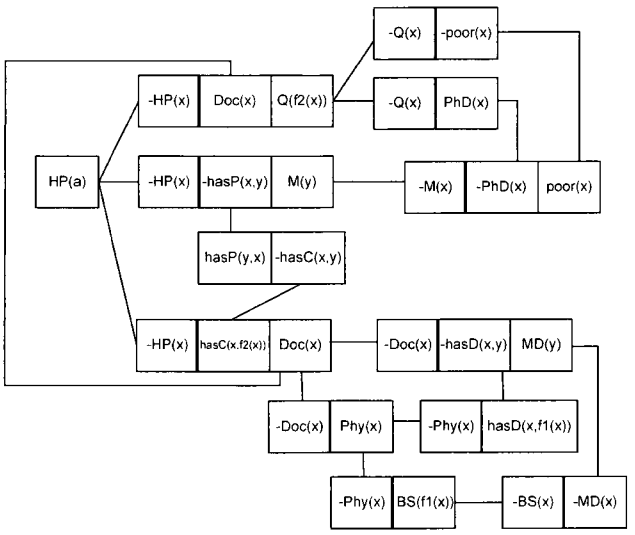

Figure 11: The Refutation Graph for the Example Knowledge Base

By applying resolution to this set of clauses and converting the resolution proof to its refutation graph, we obtain the graph shown in Figure 11. Abbreviated names are used in the graph in order to save space. For example, HP is used for $HappyPerson$, Phy is for $Physician$, M is for $Married$, Doc is for $Doctor$, $hasP$ is used for $hasParent$, $hasC$ is for $hasChild$, and $hasD$ for $hasDegree$. From the resolution proof as well the refutation graph, we know that axiom 5 does not contribute to the unsatisfiability of $HappyPerson$ since none of its clauses is in the resolution proof. As shown in Table 12, clauses that correspond to Axiom 5 are excluded from the explanation procedure.

By applying the algorithm of explaining unsatisfiable concepts, the traversal algorithm produces the traversal order in Table 13. Note that only Pattern 3 and Pattern 4 are detected in the example, so for a certain literal node, there is no

1.1.	$\neg Physician(x) \vee hasDegree(x, f_1(x))$
1.2.	$\neg Physician(x) \vee BS(f_1(x))$
2.1.	$\neg HappyPerson(x) \vee Doctor(x) \vee hasChild(x, f_2(x))$
2.2.	$\neg HappyPerson(x) \vee Doctor(x) \vee Q(f_2(x))$
2.3.	$\neg Q(x) \vee PhD(x)$
2.4.	$\neg Q(x) \vee \neg Poor(x)$
3.1.	$\neg hasParent(x, y) \vee \neg HappyPerson(y) \vee Married(x)$
4.1.	$\neg MD(x) \vee \neg BS(x)$
~~**5.1.**~~	~~$\neg Married(x) \vee Person(x)$~~
~~**5.2.**~~	~~$\neg Married(x) \vee hasSpouse(x, f_3(x))$~~
~~**5.3.**~~	~~$\neg Married(x) \vee Person(f_3(x))$~~
6.1.	$\neg Doctor(x) \vee \neg hasDegree(x, y) \vee MD(y)$
6.2.	$\neg Doctor(x) \vee Physician(x)$
7.1.	$\neg PhD(x) \vee \neg Married(x) \vee Poor(x)$
8.1.	$\neg hasChild(x, y) \vee hasParent(y, x)$
8.2.	$\neg hasParent(y, x) \vee hasChild(x, y)$
9.	$HappyPerson(a)$

Table 12: The Knowledge Base after Filtering Based on the Resolution Proof

difference between the traversal ordering values of its bridges. In this traversal order, the traversed literal node is followed by its clausal axiom and the number at the beginning of each line is the line number in Table 12.

By extracting the original axioms and assertions of each traversal step, we get the following explanations.

HappyPerson(a)
↪ **HappyPerson** $\sqsubseteq$ **Doctor** $\sqcup \exists$**hasChild**.$\top$
  ↪ **Doctor** $\sqsubseteq$ **Physician**
    ↪ **Physician** $\sqsubseteq \exists$**hasDegree**.**BS**
  ↪ **Doctor** $\sqsubseteq \forall$**hasDegree**.**MD**
    ↪ **MD** $\sqsubseteq \neg$**BS**
↪ **HappyPerson** $\sqsubseteq$ **Doctor** $\sqcup \exists$**hasChild**.**Q**
    ↪ **Q** $\sqsubseteq$ **PhD** $\sqcap \neg$**Poor**
↪ $\exists$**hasParent**.**HappyPerson** $\sqsubseteq$ **Married**
  ↪ **hasParent** $\equiv$ **hasChild**$^-$
    ↪ **PhD** $\sqcap$ **Married** $\sqsubseteq$ **Poor**

9. $HappyPerson(a)$ [**HappyPerson(a)**]
↪ **2.1.** $\neg HappyPerson(x) \vee Doctor(x) \vee hasChild(x, f_2(x))$
[**HappyPerson** $\sqsubseteq$ **Doctor** $\sqcup \exists$**hasChild.**$\top$]
↪**6.2.**$\neg Doctor(x) \vee Physician(x)$
[**Doctor** $\sqsubseteq$ **Physician**]
↪ **1.2.** $\neg Physician(x) \vee BS(f_1(x))$ [**Physician** $\sqsubseteq \exists$**hasDegree.BS**]
↪ **4.1.** $\neg MD(x) \vee \neg BS(x)$ [**MD** $\sqsubseteq \neg$**BS**]
1.1. $\neg Physician(x) \vee hasDegree(x, f_1(x))$
[**Physician** $\sqsubseteq \exists$**hasDegree.**$\top$]
↪**6.1.** $\neg Doctor(x) \vee \neg hasDegree(x, y) \vee MD(y)$
[**Doctor** $\sqsubseteq \forall$**hasDegree.MD**]
2.2. $\neg HappyPerson(x) \vee Doctor(x) \vee Q(f_2(x))$
[**HappyPerson** $\sqsubseteq$ **Doctor** $\sqcup \exists$**hasChild.Q**]
↪ **2.3.** $\neg Q(x) \vee PhD(x)$ [**Q** $\sqsubseteq$ **PhD** $\sqcap \neg$**Poor**]
2.4. $\neg Q(x) \vee \neg Poor(x)$ [**Q** $\sqsubseteq$ **PhD** $\sqcap \neg$**Poor**]
3.1.$\neg hasParent(x, y) \vee \neg HappyPerson(y) \vee Married(x)$
[$\exists$**hasParent.HappyPerson** $\sqsubseteq$ **Married**]
↪ **8.1.** $\neg hasChild(x, y) \vee hasParent(y, x)$
[**hasParent** $\equiv$ **hasChild**$^-$]
↪ **7.1.** $\neg PhD(x) \vee \neg Married(x) \vee Poor(x)$
[**PhD** $\sqcap$ **Married** $\sqsubseteq$ **Poor**]

Table 13: Explanation for the Example Knowledge Base

Note that each indented arrow shows a traversal step from one clause node to another in the refutation graph. The generated explanation reads as follows:

If a is a HappyPerson, then it can either be a Doctor or have a child which is Q (PhD and not Poor). First, if it is a Doctor, then all its degrees are MD and it is a Physician. Every Physician has a BS degree, however, BS is disjoint with MD. So there is a conflict within the branch of Doctor. Secondly, if a has a child which is a PhD and not poor, since every child of a HappyPerson is Married and every married PhD is poor, then a's child must be poor, which is a contradiction. So a cannot be a HappyPerson.

4.4 Conclusion

In this chapter, we presented our resolution based framework to provide explanations for unsatisfiability and inconsistency queries in $\mathcal{ALCHI}$. This framework firstly translates the DL knowledge base into clausal forms based on the DL semantics. In order to the preserve the original structure of the axioms, a structural transformation is adopted. The transformation method in the context of unsatisfiability and inconsistency queries is satisfiability preserving. The clauses are subsequently fed into a resolution based theorem prover to obtain the proof. During this process, the refutation graph of the proof is utilized to generate the explanation.

Since the DL language has some specific syntactic structures, we also exploit inconsistency patterns based on these features. We proposed an explanation algorithm based on inconsistency patterns and refutation graph transversal and established its soundness and completeness.

Chapter 5

Implementation and Performance

In this chapter, we discuss the implementation issues for the explanation service presented in Chapter 4. We present our system prototype and illustrate an example application.

5.1 A Prototype System for Explanation

The prototype system consists of four main components: Description Logic Reasoner, Translator, Resolution Proof Generator, and Refutation Graph Generator. The implementation is developed in Java (JDK 1.5.0).

In the following sections, we will describe each of these components in detail.

5.1.1 Description Logic Reasoner

The prototype system uses RacerPro 2.0 [HM01] as its external Description Logic reasoner. RacerPro is a knowledge representation system that implements a highly optimized tableau calculus for a very expressive description logic, $\mathcal{ALCQHIR}+$ also known as $\mathcal{SHIQ}$. This is the basic DL logic $\mathcal{ALC}$ augmented with qualified number restrictions, role hierarchies, inverse roles, and transitive roles. The input knowledge base is passed to RacerPro to test its consistency/satisifiability. The user can use OWL or Description Logic syntax script as input.

5.1.2 Translator Component

This component translates the Description Logic axioms and assertions into First Order Logic formulas based on the discussion in Section 4.1.1.

5.1.3 Resolution Proof Generator

The Resolution Proof Generator generates the corresponding resolution proof for the input First Order Logic formulas by calling an external First Order Logic reasoner. In our prototype system, we use Otter Version 3.3 [Kal01] as the first order logic reasoner. The automated deduction system Otter is designed to prove theorems stated in first-order logic with equality. Otter's inference rules are based on resolution and it includes strategies for directing and restricting searches for proofs.

5.1.4 Refutation Graph Generator

The Refutation Graph Generator transforms a resolution proof to its refutation graph and generates an explanation by traversing the graph based on the inconsistency patterns.

5.2 Illustrating Example

In this section, we demonstrate how the prototype system works by adopting the motivating example described in [HPS09]. The example is indicated below in Description Logic syntax.

---

1. $Person \sqsubseteq \neg Movie$
2. $RRated \sqsubseteq CatMovie$
3. $CatMovie \sqsubseteq Movie$
4. $RRated \equiv \forall hasViolenceLevel.High$
5. $\exists.hasViolenceLevel.\top \sqsubseteq Movie$

This example is derived from a real world ontology and it was observed that many users, including users with rich experience in OWL, failed to realize why *Person* is unsatisfiable. The users may have difficulty in understanding the source of the unsatisfiability of *Person* even if they could narrow down the problematic axioms to axioms 4 and 5. This also shows that pinpointing the axioms are not enough to help users remedy unsatisfiable concepts.

To generate an explanation for this example, the Description Logic Reasoner component in the prototype system first feeds the input axioms into Racer and

adds an assertion $Person(c)$ for the unsatisfiable concept $Person$, thus making the knowledge base inconsistent.

Then the Processor translates the knowledge base into clausal forms. After generating the corresponding resolution proof for the knowledge base, the Refutation Graph Generator produces the refutation graph. By traversing the refutation graph, the running system generates the following explanation.

Person(c)
↪ **Person** $\sqsubseteq$ $\neg$**Movie**
  ↪ $\exists$**hasViolenceLevel**.$\top$ $\sqsubseteq$ **Movie**
    ↪ $\forall$**hasViolenceLevel**.**High** $\sqsubseteq$ **RRated**
  ↪ **CatMovie** $\sqsubseteq$ **Movie**
    ↪ **RRated** $\sqsubseteq$ **CatMovie**

It shows that if c is a $Person$, then it cannot be a $Movie$. On the one hand, every $CatMovie$ is a $Movie$ and every $RRated$ is a $CatMovie$, so every $RRated$ is a $Movie$. On the other hand, the domain of $hasViolenceLevel$ is $Movie$, and $\forall hasViolenceLevel.High$ is the subset of $RRated$ (hence $Movie$), so $Movie$ always holds and c is a $Movie$, which leads to a contradiction. So c cannot be a $Person$.

It should be noted that even for some experienced users, it might not be intuitive to see that $\exists hasViolenceLevel.\top \sqsubseteq Movie$ follows from the fact that the domain of $hasViolenceLevel$ is $Movie$, and $\top \sqsubseteq Movie$ can be entailed from the knowledge base. Instead of giving a set of axioms for the unsatisfiability problem, our prototype system generates a fine-grained and detailed explanation for this example. This explanation can help experienced users to better understand the problem.

5.3 Performance Evaluation

To evaluate the usability and scalability of our proposed explanation procedure, we conducted experiments on a Windows XP system with a 2.66GHz Intel Core2 Duo processor, and 4GB memory.

5.4 Test Ontologies

Table 14 shows the test ontologies used in our experiments. The ontologies were obtained from the website of SWOOP[1]. Modifications were made in order to convert them to $\mathcal{ALCHI}$.

Ontology	Axioms	Concepts/Unsatisfiable Concepts
Chemical	87	41/33
Koala	11	6/3
University	18	20/4
Economy	1655	275/43
Transport	1004	313/41
Tambis	1846	401/61

Table 14: Test Ontologies Used in the Evaluation

Table 15 shows the experimental results using the prototype system. The first column shows the ontology used in the experiment, the second column shows the average time to generate an explanation for an arbitrary unsatisfiable concept in the ontology and the last column shows the overall time to generate explanations for all the unsatisfiable concepts. The time is measured in seconds.

[1]http://www.mindswap.org/2005/debugging/ontologies/

Ontology	Generating explanations for a concept	Generating explanations for all unsatisfiable concepts
Koala	1.013	7.05
University	2.045	23.35
Chemical	3.213	78.67
Economy	2.985	101.02
Transport	3.218	123.15
Tambis	4.523	189.79

Table 15: Performance of the Prototype System Using Otter

From the results, we can observe that the larger the knowledge base and the more unsatisfiable concepts there are in the ontology, the more time it takes to generate explanations for all the unsatisfiable concepts. However, if an explanation for only one of the unsatisfiable concepts is needed, even for a large knowledge base such as Tambis, the explanation can be generated in reasonable time.

5.4.1 Experiments with other FOL reasoners

We have also tested the explanation procedure with another FOL reasoner Vampire 7.0[RV02]. Like most efficient state-of-the-art automated theorem provers (ATPs), such as Otter, Vampire implements saturation with resolution and paramodulation. Since 1999, Vampire has won 17 division titles in the theorem proving competition[cas], especially in the first-order formulae division and formulas in conjunctive normal form division.

We conducted a smaller evaluation on a few selected ontologies including Chemical, Economy, Transport, and University on generating explanation for an

arbitrary unsatisfiable concept. The results show that Vampire consistently performs roughly twice as fast as Otter, which is expected, noting that Vampire has been arguably the best general-purpose prover in recent years. Besides, Vampire is highly tunable and its performance could be improved through adjusting its setup parameters.

5.5 Conclusion

In this chapter, we presented the main components of our implemented prototype system. The system first gets the input knowledge base and configurations from the user. It then feeds the input to the Description Logic Reasoner to test if the knowledge base is inconsistent or if there is any unsatisfiable concept. If it is the case, it calls the Translator to translate the knowledge base into First Order Logic formulas. It then feeds the formulas to the Resolution Proof Generator to generate resolution proofs. Finally, an explanation is generated by Refutation Graph Generator and presented to the user.

The implemented prototype system uses RacerPro 2.0 and Otter 3.3 as its Description Logic and First Order Logic reasoners. However, other reasoners could also be easily incorporated within the prototype system, since one of the advantages of our approach is that it is independent of the particular reasoner used.

We also demonstrated its practical value by showing how it generates explanations for a difficult problem observed in real world applications.

Part III

A Framework for Measuring Inconsistencies

In the previous part, we presented a resolution framework to provide explanations for unsatisfiability and inconsistency problems in $\mathcal{ALCHI}$. It can help users to identify the origin and causes of inconsistencies in a DL knowledge base or an ontology. In real world applications, it is often the case that there are multiple sources for an inconsistency, and some axioms may contribute more to the inconsistency than the others. So explaining the inconsistencies is part of the problem and a diagnosis mechanism is needed to make suggestions of which axioms are more problematic and need to be dealt with first. In this part, we propose an inconsistency measure technique as the foundation of a diagnosis procedure in Chapter 6. We then demonstrate its applicability to ontologies in Chapter 7.

Chapter 6

A Diagnosis Procedure

Ontologies play a key role in the infrastructure of the Semantic Web for sharing precisely defined terms which can be made accessible to automated agents. For ontologies with complex knowledge to represent and reason with, errors due to inconsistencies become quite common, and these inconsistencies can be intrinsically different. While there are Description Logic reasoners that can detect inconsistencies in input ontologies, they do not help classify and/or summarize the nature of the inconsistencies that are present.

In this chapter, we propose a technique based on Shapley values to measure inconsistencies in ontologies. This measure can be used to identify which axioms in an input ontology or which parts of these axioms need to be removed or modified in order to make the input consistent, hence it can be used as the backbone of a diagnosis and repair framework. We also propose optimization techniques to

improve the efficiency of computing Shapley values. The proposed technique and the optimization proposed are independent of the particular DL language of the ontology and are independent of a particular reasoning system used. Applications of this method can improve the quality of ontology diagnosis and repair, in general.

6.1 Inconsistency Measures

As the size of ontologies grows and applications developed become more complex, inconsistencies become hard to avoid in the design and development of ontologies. According to the classical *ex contradictione quodlibet* (ECQ) principle, anything that follows from an inconsistent ontology is useless. In order to help users to resolve the inconsistencies in ontologies, several approaches to identify and explain the cause of these inconsistencies have been proposed [DHS05a, PSK05, SC03]. An assumption often made in these approaches is that all inconsistencies are equally "bad". However, it is possible for an ontology to contain two or more sources of inconsistencies and they may have a different impact on the inconsistencies. They may not necessarily contain the same contradiction and the same information, and may have overlapping content.

In this section, we introduce a method of measuring inconsistencies in DL knowledge bases. This approach is the first of its kind to quantitatively measure the inconsistencies. First an inconsistency value is defined, and then it is used as the characteristic function to compute the Shapley value. Our approach borrows

some ideas from [HK06], which proposed to use the Shapley value to obtain an inconsistency measure for propositional logic. We take into account the proportion of the axioms that contribute to the inconsistency by considering both clauses and axioms. Since clauses are more fine-grained than axioms, it allows us to take a deeper look inside the axioms. We also discuss the relationship between the inconsistency measure and the minimal inconsistent subsets of ontologies. The computational complexity of calculating the Shapley value is at least Exp-time, which shows that it does not scale well in general [CS04]. Therefore we propose to optimize the calculation based on the structural relevance of the axioms and properties of the defined inconsistency measure.

6.1.1 Ontologies in the Semantic Web

In 2004, the Web Ontology Language (OWL) was recommended as the standard web ontology language by the World Wide Web Consortium (W3C) [w3c]. It is a machine-readable language for sharing and reasoning information using ontologies on the Internet. OWL is a vocabulary extension of the Resource Description Framework (RDF) and is a revision of the DAML+OIL Web Ontology Language.

OWL represents the domain by defining hierarchies of classes and properties. An OWL ontology consists of axioms and facts. Axioms build relationships between classes and properties. Facts describe information about individuals. OWL has three flavors: OWL Lite, OWL DL, and OWL Full. These flavors incorporate

different features, and OWL Full contains OWL DL, which in turn contains OWL Lite. OWL DL and OWL Lite corresponds semantically with certain Description Logic languages. Reasoning tasks are undecidable in OWL Full and currently there is no reasoner that supports reasoning of every feature of OWL Full.

OWL was extended to OWL2 in 2009 by adding several new features, such as property chains. Interested reader can refer to [owl] for details. There are three sublanguages in OWL2. OWL 2 EL is a fragment that has polynomial time reasoning complexity. OWL 2 QL is designed to enable easier access and query to data stored in databases. OWL 2 RL is a rule subset of OWL 2.

Roughly speaking, a concept in DL is referred to as a class in OWL. A role in DL is a property in OWL. The terms axioms and individuals have the same meaning in DL and OWL. OWL DL is based in part on the DL $\mathcal{SHOIN}(\mathcal{D})$, which includes special constructors such as oneOf, transitive properties, inverse properties and datatype properties, and its subset OWL Lite which is based on the less expressive DL $\mathcal{SHIF}(\mathcal{D})$, is $\mathcal{SHOIN}(\mathcal{D})$ without the oneOf constructor and with the number restriction constructors limited to 0 and 1. Due to the close connection between OWL and DLs, we will make no distinction between ontologies and knowledge bases in DL, and examples are given mainly in DL syntax. The DL languages that we work on are those for which consistency checks are decidable.

6.1.2 Motivating Example

The following example is adapted from [SC03], which we modified to be inconsistent. In the example, A, B, C, D, $A1$ to $A6$ denote concepts, and a and b are individuals. We assign a number to each axiom/assertion in the example.

1. $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$	**2.** $A2 \sqsubseteq A \sqcap A4$
3. $A3 \sqsubseteq A5 \sqcap A4$	**4.** $A4 \sqsubseteq C \sqcap \forall S.B$
5. $A5 \sqsubseteq \exists S.\neg B$	**6.** $D \sqcup \neg D \sqsubseteq D \sqcap \neg D$
7. $A1(a)$	**8.** $A3(b)$
9. $A6 \equiv D$	

A complete DL reasoner, such as FaCT++ [TH06], RACER [HM01], Pellet [SP04], or HermiT [MSH09], reports this knowledge base to be inconsistent. However, they can not provide crucial information, e.g., that there are four inconsistent subsets ({3, 4, 5, 8}, {1, 2, 7}, {1, 3, 4, 5, 7} and {6}) in this knowledge base, and that one axiom (axiom 6) is inherently inconsistent. We will use this as our running example throughout this chapter to show how the hidden information can be unraveled using our method.

6.1.3 Background

In this section we review some definitions of the Shapley value in game theory [HK06], which we have adapted for use in DL in this work.

Definition 6.1.1 *Given a set of axioms and assertions in a knowledge base K, a characteristic function $v : 2^K \rightarrow \mathbb{R}$ assigns a value to each subset K' of K ($K' \subseteq K$).*

We call a subset of axioms and assertions in K a coalition of K, so K' is a coalition of K.

An example of the characteristic function is the *drastic inconsistency value*, which assigns 1 to a set of axioms if it is inconsistent, and 0 to the set if it is consistent.

Definition 6.1.2 *For a set of axioms and assertions $K' \subseteq K$, the drastic inconsistency value of K' is defined as:*

$$I_d(K') = \begin{cases} 0 & \text{if } K' \text{ is consistent or } K' = \phi \\ 1 & \text{otherwise} \end{cases} \quad (1)$$

Example 6.1.1 *Some drastic inconsistency values of the running example are as follows, where we only show some examples with the inconsistency value 1, as well as some consistent ones (with this value being 0).*[1]

$I_d(\{1\}) = 0$ $\quad$ $I_d(\{2\}) = 0$ $\quad$ $I_d(\{3\}) = 0$

$I_d(\{1, 2\}) = 0$ $\quad$ $I_d(\{3, 4, 5\}) = 0$ $\quad$ $I_d(\{1, 2, 6\}) = 1$

$I_d(\{3, 4, 5, 8\}) = 1$ $\quad$ $I_d(\{3, 4, 5, 6\}) = 1$

$I_d(\{1, 3, 4, 5, 7\}) = 1$

[1] *For the sake of simplicity, we refer to the axioms and assertions by their numbers.*

As shown above, the coalition $\{1, 2, 6\}$ has the drastic inconsistency value 1. Axiom 6 is often of a great value for a coalition it joins. For example, it can bring inconsistency value 1 to $\{2, 6\}$ for making the coalition $\{1, 2, 6\}$. And it can also bring 1 to the coalition $\{3, 4, 5, 6\}$.

Similarly, we can define another characteristic function which assigns 0 to a set of axioms if a concept A is satisfiable and 1 otherwise.

Definition 6.1.3 *The concept-related inconsistency value of a set of axioms $K' \subseteq K$ w.r.t. a concept A (A occurs in K') is defined as:*

$$I_A(K') = \begin{cases} 0 & \text{if } A \text{ is satisfiable w.r.t. } K' \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

Example 6.1.2 *Some of the concept-related inconsistency values of the working example are:*

$I_{A1}(\{1, 2\}) = 1$ $\qquad$ $I_{A1}(\{1, 3, 4, 5\}) = 1$

$I_{A3}(\{3, 4, 5\}) = 1$ $\qquad$ $I_{A3}(\{3, 4, 9\}) = 0$

An inconsistent measure is to evaluate the contribution of each axiom or assertion in the knowledge base to the overall inconsistencies. The higher the measure is, the more weight an axiom carries in contributing to the inconsistencies. Shapley [Sha53] proposed such a measure, known as Shapley values, in the context of game

theory in 1953, which describes a fair allocation of gains obtained by the cooperation among several agents.

The Shapley value is defined for a game that has n agents. In the game, the agents can form coalitions, which is a subset of the n agents. Each coalition has a gain when all its members work together as a team. A problem which may arise here is "which agent contributes the most to different coalitions?" A solution to this problem can help determine which agent contributes more to the game than the others. The Shapley value is proposed to tackle this problem. The basic idea is as follows. Suppose the agents join a coalition according to a certain order, and the payoff of an agent in this coalition is its marginal contribution to the gain of the coalition. The Shapley value takes all the possible orders of the coalition formation into account and averages the agent's marginal contribution over them.

Inconsistency checking can be deemed as a game, with each axiom (or assertion) in the knowledge base deemed as an agent. Analogously, the contribution of each axiom (or assertion) to the inconsistencies can be measured using the Shapley value.

Let K be a knowledge base, σ_K be the set of all permutations on K, and $n = |K|$ be the cardinality of K. Given an order $\sigma \in \sigma_K$, we use p_σ^α to denote the set of all the axioms and assertions in σ that appear strictly before an axiom (or an assertion) α.

Definition 6.1.4 *The Shapley value for an axiom (or an assertion) α in a knowledge base K is defined as:*

$$S_\alpha(K) = \frac{1}{n!} \sum_{\sigma \in \sigma_K} [v(p_\sigma^\alpha \cup \{\alpha\}) - v(p_\sigma^\alpha)]$$

The Shapley value can be directly computed from the possible coalitions without considering the permutations as follows:

$$S_\alpha(K) = \sum_{C \subseteq K} \frac{(c-1)!(n-c)!}{n!} (v(C) - v(C \backslash \{\alpha\}))$$

where C is any coalition of the axioms and assertions K, $n = |K|$, and $c = |C|$.

6.1.4 Inconsistency Measure Based on the Shapley Value

We can take the drastic inconsistency measure defined by Definition 6.1.2 (the same computation can be applied to the concept-related measure in Definition 6.1.3) as the characteristic function, and then use the Shapley value to compute the extent to which an axiom or an assertion is concerned with the inconsistency.

For example, suppose K is a knowledge base and α is an axiom (or an assertion) in K. Then the Shapley value of α based on the drastic inconsistency value I_d is defined as:

$$S_\alpha(K) = \sum_{C \subseteq K} \frac{(c-1)!(n-c)!}{n!} (I_d(C) - I_d(C \backslash \{\alpha\})) \tag{3}$$

where $n = |K|$ and $c = |C|$.

The Shapley value of a knowledge base K is a vector of values, one for the Shapley value of each axiom (or assertion) in K.

Example 6.1.3 *The Shapley value of the knowledge base* $K = \{C \sqcup \neg C \sqsubseteq C \sqcap \neg C, \top \sqsubseteq \exists R.B, \top \sqsubseteq \forall R.\neg B, A \sqsubseteq D\}$ *is the vector* $(\frac{4}{6}, \frac{1}{6}, \frac{1}{6}, 0)$.

This shows that the axiom $\{C \sqcup \neg C \sqsubseteq C \sqcap \neg C\}$ contributes the most to the inconsistency of K.

Example 6.1.4 *The Shapley value of the working example is* $(\frac{268}{7!}, \frac{250}{7!}, \frac{120}{7!}, \frac{120}{7!}, \frac{120}{7!}, \frac{3774}{7!}, \frac{268}{7!}, \frac{120}{7!}, 0)$.

This shows that axiom 6 is the one that causes the most problems. Axioms 3, 4, 5 and 8 are equally responsible for the inconsistencies, so are 1 and 7. Axiom 9 has a value of 0, which means it does not contribute to the inconsistency. Axiom 2 contributes more to the inconsistency than 3, 4, 5 or 8, and the inconsistency is more equally distributed among 3, 4, 5 and 8.

6.1.5 Properties of the Inconsistency Measures

We make a few observations and remarks regarding the inconsistency measures.

Definition 6.1.5 *The characteristic function v is monotonic if* $v(X) \leq v(Y)$, *whenever* $X \subseteq Y$.

An monotonic function indicates that adding more agents to the coalition will never decrease the value. In the worst case, they contribute nothing to the coalition. Due to the monotonic nature of DL reasoning, we have the following result.

Proposition 6.1.1 *The drastic (concept-related) inconsistency value (see Definitions 6.1.2 and 6.1.3) is monotonic.*

Proof. For a set of axioms and assertions $K' \subseteq K$, there are three possible scenarios:

1. K is consistent and K' is consistent, in this case, $I_d(K) = I_d(K') = 0$
2. K is inconsistent and K' is inconsistent, in this case, $I_d(K) = I_d(K') = 1$
3. K is inconsistent and K' is consistent, in this case, $I_d(K) = 1, I_d(K') = 0$

Thus $I_d(K') \leq I_d(K)$, and the drastic inconsistency value is monotonic. Similarly, we can prove that the concept-related inconsistency value is monotonic.

■

A set of axioms and assertions is called *convergent* if its inconsistency value is convergent, i.e., it is the same as the inconsistency values of its supersets, and adding any other axioms or assertions does not change its inconsistency value.

Definition 6.1.6 *The convergent subset K' of a knowledge base K is defined as a set of axioms and assertions that satisfies the following two properties:*

1. $I_d(K') = 1$ *(or* $I_A(K') = 1$*), and*

2. $I_d(K'') = 0$ *(or* $I_A(K'') = 0$*), for all* $K'' \subset K'$.

Intuitively, K' is a subset of K, in which the inconsistency value flips from 0 to 1. There is a direct relation between the convergent subsets and minimal inconsistent subsets of a knowledge base, defined as follows.

Definition 6.1.7 (MIS and MUS) *We say that* $\mathcal{K}'$ *is a minimal inconsistent subset (MIS) of a knowledge base* $\mathcal{K}$ *(or a minimal unsatisfiable subset (MUS) w.r.t. a concept* C*) if the following two conditions hold:*

1. $\mathcal{K}'$ *is inconsistent (or* C *is unsatisfiable in* $\mathcal{K}'$*), and*

2. $\mathcal{K}''$ *is consistent (or* C *is satisfiable in* $\mathcal{K}''$*), for every* $\mathcal{K}'' \subset \mathcal{K}'$.

From the definition of the drastic inconsistency value and the concept-related inconsistency value, we can easily prove the following preposition.

Proposition 6.1.2 K' *defined in Definition 6.1.6 is a MIS of a knowledge base* $\mathcal{K}$ *(or a MUS w.r.t. a concept* C *).*

Proof. If K' is a convergent subset of a knowledge base K, then $I_d(K') = 1$ (or $I_A(K') = 1$), and $I_d(K'') = 0$ (or $I_A(K'') = 0$), for all $K'' \subset K'$. According to Definition 6.1.2 (or Definition 6.1.3), $\mathcal{K}'$ is inconsistent (or C is unsatisfiable in $\mathcal{K}'$), and $\mathcal{K}''$ is consistent (or C is satisfiable in $\mathcal{K}''$), for every $\mathcal{K}'' \subset \mathcal{K}'$. ■

Another inconsistency measure of a coalition K' that can be defined is based on the number of minimal inconsistent subsets that would be removed if we remove K' from the knowledge base. In other words, this measures the impact of K' on the knowledge base, formalized as follows.

Definition 6.1.8 *The impact inconsistency measure of a subset K' in a knowledge base K can be defined as follows:*

$$I_i(K') = |MIS(K)| - |MIS(K - K')|$$

where $|MIS(K')|$ denotes the number of minimal inconsistent subsets of K'.

Example 6.1.5 *There are four convergent subsets, i.e., four MISs in the working example.*

$MIS_1 = \{1, 2, 7\}$, $MIS_2 = \{3, 4, 5, 8\}$, $MIS_3 = \{1, 3, 4, 5, 7\}$, $MIS_4 = \{6\}$

$I_i(\{1\}) = I_i(\{7\}) = I_i(\{3\}) = I_i(\{4\}) = I_i(\{5\}) = 2$

$I_i(\{2\}) = I_i(\{6\}) = I_i(\{8\}) = 1$

$I_i(\{9\}) = 0$

Obviously, the removal of 1, 7, 3, 4, or 5 will remove the most number of inconsistencies. The removal of axiom 9 will not affect the inconsistencies at all.

6.1.6 Apply the Inconsistency Measures to Clauses

The inconsistency measures discussed in this chapter so far are applied to axioms in ontologies. It excludes the possibility of a more fine-grained inspection of the content of the axioms. In particular, if the inconsistency is at the level of a single axiom, then there could only be two values: consistent or inconsistent. In Chapter 4, we have developed a resolution based technique to explain inconsistency in ontologies. We found that clauses work on a more fine-grained level than DL axioms, so an approach based on clauses can identify specific parts of axioms that are responsible for an inconsistency. Consequently, the inconsistency measures can also be applied to clauses and this allows us to look inside the axiom and identify which portion of the axiom is contributing to the inconsistency.

6.2 Computational Complexity and Optimization Issues

The major source of inefficiency in calculating the Shapley value is the difficulty to determine the inconsistency value of an axiom. It is directly dependent on the complexity of consistency checking in DL reasonings. One possible way to improve this is to reduce the number of consistency checks. Besides, the complexity is also related to the computation process itself. The computation of the Shapley value involves considering all the subsets of the axioms/assertions in the knowledge base,

and hence its best-case complexity is Exp-time. However, we do not really need to consider all the subsets. In the following sections we elaborate on this and discuss some optimizations.

6.2.1 Partition Based on Structural Relevance

In DLs, axioms[2] can be related to each other through structural relevance. For example, the axiom $A \sqsubseteq B$ is structurally related to $\neg B \sqsubseteq \top$ but not to $\neg C \sqsubseteq D$. It is clear that adding a structurally unrelated axiom to a coalition will not change the relative inconsistency value. Structural relevance is an equivalence relation, hence it can be exploited to induce a partitioning of the axioms, which as shown below, can be used as an optimization to speed up the computation of the Shapley inconsistency value.

Definition 6.2.1 *We say an axiom is directly structurally related to another axiom if the intersection of their signature (the set of all (negated) concept names and role names occurring in the axiom) is not empty. The structural relevance is the transitive closure of direct structural relevance.*

Example 6.2.1 *In the motivating example,* $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$ *is directly structurally related to* $A2 \sqsubseteq A \sqcap A4$. *It is structurally related to* $A4 \sqsubseteq C \sqcap \forall S.B$ *(because* $A3 \sqsubseteq A5 \sqcap A4$*), but it is not related to* $D \sqcup \neg D \sqsubseteq D \sqcap \neg D$.

[2]For the sake of simplicity, we only refer to axioms, assertions can be considered in the same way.

Example 6.2.2 *There are two partitions in the motivating example:*

1. $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$	**2.** $A2 \sqsubseteq A \sqcap A4$
3. $A3 \sqsubseteq A5 \sqcap A4$	**4.** $A4 \sqsubseteq C \sqcap \forall S.B$
5. $A5 \sqsubseteq \exists S.\neg B$	**7.** $A1(a)$
8. $A3(b)$	

Figure 12: Partition 1 of the Motivating Example.

6. $D \sqcup \neg D \sqsubseteq D \sqcap \neg D$	**9.** $A6 \equiv D$

Figure 13: Partition 2 of the Motivating Example.

The following result suggests that partitioning according to structural relevance can be used to reduce the computational complexity of the Shapley value in our context.

Lemma 6.2.1 *If* $K = \sum_{i=1}^{T} K_i$ *is a partitioning of a knowledge base (*T *is the number of partitions after partitioning), and all* K_i *have the same inconsistency value function* I*, then for any axiom (or assertion)* $\alpha \in K_i$*,*

$$S_\alpha(K_i) = \sum_{C \subseteq K_i} \frac{(c-1)!(n-c)!}{n!}(I(C) - I(C \backslash \{\alpha\}))$$

where $n = |K_i|$ *and* $c = |C|$*.*

After the partitioning, the Shapley value of an axiom (or an assertion) can be computed in $O(\sum_{i=1}^{T} 2^{|K_i|})$*.*

The partitioning based on structural relevance preserves the total ordering on the Shapley values of the axioms (and assertions) inside the same partition. In other

words, if an axiom has a higher Shapley value than another axiom in the partition, then it will also have a higher Shapley value in the knowledge base.

Theorem 6.2.1 *If* $K = \sum_{i=1}^{T} K_i$ *is a partitioning of a knowledge base, and all* K_i *have the same inconsistency value function* I*, then for any* $\alpha, \beta \in K_i$*, if* $S_\alpha(K_i) > S_\beta(K_i)$*, then* $S_\alpha(K) > S_\beta(K)$.

Proof. For each partition $K_i \subseteq K$ and axioms $\alpha, \beta \in K_i$, if $S_\alpha(K_i) > S_\beta(K_i)$, then $\sum_{C \subseteq K_i} \frac{(c-1)!(n-c)!}{n!}(I(C) - I(C \backslash \{\alpha\})) > \sum_{C \subseteq K_i} \frac{(c-1)!(n-c)!}{n!}(I(C) - I(C \backslash \{\beta\}))$. We show $S_\alpha(K) > S_\beta(K)$ by considering the following possible cases.

1. For any $C_i \subseteq K$, if $C_i \subseteq C$, as previously indicated, $\sum_{C_i \subseteq K} \frac{(c-1)!(n-c)!}{n!}(I(C_i) - I(C_i \backslash \{\alpha\})) > \sum_{C_i \subseteq K} \frac{(c-1)!(n-c)!}{n!}(I(C_i) - I(C_i \backslash \{\beta\}))$,

2. If $C_i \nsubseteq C$ and $I(C_i - C) = 1$, then $I(C_i) - I(C_i \backslash \{\alpha\}) = I(C_i) - I(C_i \backslash \{\beta\})$, so $\sum_{C_i \subseteq K} \frac{(c-1)!(n-c)!}{n!}(I(C_i) - I(C_i \backslash \{\alpha\})) = \sum_{C_i \subseteq K} \frac{(c-1)!(n-c)!}{n!}(I(C_i) - I(C_i \backslash \{\beta\}))$,

3. If $C_i \nsubseteq C$ and $I(C_i - C) = 0$, then $I(C_i) - I(C_i \backslash \{\alpha\}) > I(C_i) - I(C_i \backslash \{\beta\})$. So $\sum_{C_i \subseteq K} \frac{(c-1)!(n-c)!}{n!}(I(C_i) - I(C_i \backslash \{\alpha\})) > \sum_{C_i \subseteq K} \frac{(c-1)!(n-c)!}{n!}(I(C_i) - I(C_i \backslash \{\beta\}))$.

Summing up all cases, $\sum_{C \subseteq K} \frac{(c-1)!(n-c)!}{n!}(I(C) - I(C \backslash \{\alpha\})) > \sum_{C \subseteq K} \frac{(c-1)!(n-c)!}{n!}(I(C) - I(C \backslash \{\beta\}))$. Therefore $S_\alpha(K) > S_\beta(K)$. ■

6.2.2 Optimization Based on Properties of the Inconsistency Measure

Partitioning the knowledge base according to structural relevance can work very well when the sizes $|K_i|$ are small, especially when they are bounded by a constant. This, however, is largely dependent on the particular knowledge base considered. If the knowledge base is large and cannot be partitioned into smaller parts, the optimization based on partitioning won't improve the performance of the calculation.

In this section, we propose an algorithm to calculate the Shapley value based on the properties of the inconsistency measure, especially the convergent property. This method aims to reduce the number of consistency checks.

The basic idea of the algorithm is quite simple: according to Definition 6.1.6 in Section 6.1.5, a convergent subset is a minimal inconsistent subset of a knowledge base. Any superset of a convergent knowledge base can have its inconsistency value derived to be 1 due to the monotonicity of DLs. Hence once a subset is convergent, there is no necessity to compute its supersets. The detailed algorithm is shown in Figure 14. We can either compute the convergent sub-terminologies on the fly during the computation of the Shapley value, or use the algorithms for MIS proposed in [Sch05] to compute them in advance.

To see how this algorithm works, we consider one of the partitions in the motivating example.

Algorithm **INCONSISTENCY MEASURE**
Input: an axiom (or assertion) α in a partition K
Output: the Shapley value of α

for all the unvisited $K' \subseteq K$, sorted by the cardinality of K'
  tag K' as visited
  if $I(K' \cup \alpha)! = 0$
    for all supersets K'' of K'
      $I(K'') = 1$
      tag K'' as visited
    add K' to the computation of the Shapley value of α
return the Shapley value of α

Figure 14: Computing Shapley Values

1. $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$	**2.** $A2 \sqsubseteq A \sqcap A4$
3. $A3 \sqsubseteq A5 \sqcap A4$	**4.** $A4 \sqsubseteq C \sqcap \forall S.B$
5. $A5 \sqsubseteq \exists S.\neg B$	**7.** $A1(a)$
8. $A3(b)$	

Example 6.2.3 *The algorithm will first compute the inconsistency value of* $A1(a)$*, and then the coalition of* $A1(a)$ *and any one of the other axioms, and then the coalition of* $A1(a)$ *and any two of the other axioms, since the following coalition has an inconsistency value of 1, we will skip computing the inconsistency value of its supersets.*

1. $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$	**2.** $A2 \sqsubseteq A \sqcap A4$
7. $A1(a)$	

It is the same case with the following coalition.

1. $A1 \sqsubseteq A2 \sqcap \neg A \sqcap A3$	
3. $A3 \sqsubseteq A5 \sqcap A4$	**4.** $A4 \sqsubseteq C \sqcap \forall S.B$
5. $A5 \sqsubseteq \exists S.\neg B$	**7.** $A1(a)$

6.3 Conclusion

With the development of more expressive ontologies in the Semantic Web community, inconsistency has become an increasing problem that seriously hampers the design and application of web ontologies. In this chapter, we presented a technique for measuring inconsistencies which uses the Shapley value proposed originally in the context of game theory. Since the Shapley value aims to distribute the gains from cooperation in a fair manner, it can be used to impute the inconsistency to each of the axioms in the problematic ontology. The idea is to first define an inconsistency value, and then take it as the characteristic function, using the Shapley value to compute the contribution of each axiom or assertion to the inconsistencies in the ontology. The measure associated with an axiom shows the degree of its responsibility for the inconsistencies, which in turn can provide guidelines for diagnosing and repairing the ontologies.

Chapter 7

Performance Evaluation

In this chapter, we present results demonstrating the practical value of the diagnosis procedure proposed in Chapter 6. We first describe the ontologies used in our experiments, followed by an illustration of the inconsistency measure applied to these ontologies. Finally we study the performance of the proposed optimizations for calculating Shapley Value based inconsistency measures.

7.1 Test Ontologies

To evaluate our proposed diagnosis algorithm, we have implemented it as well as the suggested optimizations in Chapter 6. Tests were performed on a Windows XP system with a 2.66GHz Intel Core2 Duo processor, and 4GB memory. The implementation is developed in Java (JDK 1.5.0).

All the experiments were run on existing OWL ontologies that are used in

Chapter 5. The original ontologies are consistent and they are provided as sample ontologies to test the repair service of the OWL ontology editor SWOOP 2.3 beta 3 [KPS+05], which uses Pellet as the default DL reasoner. In our experiments, we asserted fresh individuals for these unsatisfiable concepts in order to make the ontologies inconsistent. Note that after the modification, SWOOP can no longer generate repair plans for these ontologies, as it does not provide diagnosis services for inconsistent ontologies.

7.2 Resolving Inconsistencies

The effectiveness of the inconsistency measure based diagnosis framework can be illustrated using the Koala Ontology. In the Koala ontology as shown in Table **??**, there were three sources of inconsistencies.

Let us consider an adapted part of the Koala ontology as follows:

1.$Koala \sqsubseteq Marsupials$
2.$Quokka \sqsubseteq Marsupials$
3.$Person \sqsubseteq \neg Marsupials$
4.$Koala \sqsubseteq HardWorker$
5.$KoalaWithPhD \sqsubseteq \exists hasDegree.\top \sqcap Koala$
6.$Quokka \sqsubseteq HardWorker$
7.$HardWorker \sqsubseteq Person$
8.$\exists hasDegree.\top \sqsubseteq Person$
9.$Koala(Suzy)$
10.$KoalaWithPhD(Lizzy)$
11.$Quokka(Pan)$

There are three sources of inconsistencies in this ontology. A koala named

Suzy is forced to be a member of the disjoint classes marsupial and person. She is a marsupial because koala is a subclass of marsupial and she is a person because she is a hard worker and every hard worker is a person. A koala with a PhD named Lizzy is also forced to be a member of the disjoint classes marsupial and person. She is a person because person is the domain of hasDegree and every koala with a PhD must have at least one hasDegree property. Similarly, a Quokka named Pan is also causing inconsistency problems.

After the computation of the Shapley value for each axiom, axiom 3 gets the highest Shapley value, followed by axioms 7 and 1. For the naive user of these ontologies, this information can be very helpful. It shows that although intuitively marsupial and person are disjoint classes, they make the ontology inconsistent. Removing Axiom 3 will render the ontology consistent as follows, and therefore enables most of the reasoning services that users have expected from their ontology development tools.

7.3 Performance Evaluation

In order to test the performance of the optimization techniques proposed in Chapter 6, the test ontologies were run with one or more of the optimization techniques. We implemented an approach which is based on the algorithm for finding minimal inconsistent subsets [Sch05]. It employs a variation of Reiter's Hitting Set Tree algorithm [Rei87] to reduce the number of subsets that are involved in the computation

of inconsistency measures. In the following sections, we present the experimental results for optimization techniques including partitioning based on structural relevance, and properties of the inconsistency measure. At the end of this section, we also present the overall effect of all these optimization techniques.

7.3.1 Evaluation of Partition Based on Structural Relevance

In Section 6.2.1, we proposed an optimization technique that partitions the knowledge base according to its Description Logics structural relevance. To test the performance improvement for partitioning, we measured the running time (in seconds) of calculating the inconsistency measures when the knowledge base is partitioned (T_P) and when it is not partitioned (T_{noP}). The "Time Out" threshold is set to be 1000 seconds.

Figure 15 and Table 16 show the performance improvement for the test ontologies. We measure the running time improvement by using T_{noP}/T_P. The x-axis in the figure indicates the ontology being tested, and the y-axis shows the running time in seconds using logarithmic-scale. The same notations for figures will be used throughout this chapter.

There are some interesting observations that can be made about these results:

1. The larger the knowledge base and the less coupled the knowledge base, the more significant the performance improvement we can achieve by partitioning the knowledge base based on structural relevance. If a knowledge base is

Ontology	# of Partitions	With Partitioning (T_P)	Without Partitioning (T_{noP})	T_{noP}/T_P
Koala	1	12.01	12.01	1
University	3	20.45	35.12	1.72
Chemical	17	434.55	Time Out	N/A
Economy	9	Time Out	Time Out	N/A
Transport	29	Time out	Time Out	N/A

Table 16: Optimization Using Partitioning

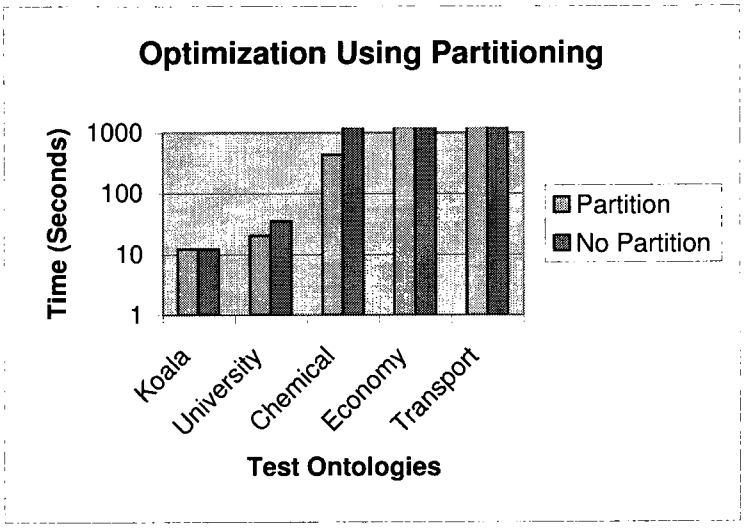

Figure 15: Evaluation of Partition Based on Structural Relevance

loosely coupled in terms of structural relevance, it can be easily partitioned into smaller knowledge bases. For example, for the Chemical ontology, if the knowledge base is not partitioned into smaller groups, the number of axioms in the ontology is too large for the inconsistency measure calculation program within the time out limit. If the same ontology is partitioned according to structural relevance, it can be computed in reasonable time. This shows the

importance of partitioning the knowledge base based on structural relevance in order to keep the axiom set as small as possible.

2. Partitioning the knowledge base into smaller groups does not always help, especially when the knowledge base is small or tightly coupled in terms of structural relevance. For example, there are only 24 axioms in the University ontology, and 20 of them are structurally related. In this case, the running time is only improved from 35.12s to 20.45s. This is not surprising since when the knowledge base is simple and small, it does not take long to calculate the inconsistency measure even without partitioning.

7.3.2 Evaluation of Properties of the Inconsistency Measure

In Section 6.2.2, we proposed an algorithm to calculate the Shapley value based on the properties of the inconsistency measure, especially the convergent property. To test the performance improvement, we measured the running time (in seconds) of calculating the inconsistency measures when this technique is adopted(T_P) and when it is not (T_{noP}). The "Time Out" threshold is also set to be 1000 seconds.

Figure 16 and Table 17 show the performance improvement for the test ontologies. We measure the running time improvement by using T_{noP}/T_P.

There are also some interesting observations that can be made about these results:

Ontology	With Property(T_P)	Without Property(T_{noP})	T_{noP}/T_P
Koala	3.21	12.01	3.74
University	8.86	35.12	3.96
Chemical	375.23	Time Out	N/A
Economy	355.12	Time Out	N/A
Transport	901.23	Time Out	N/A

Table 17: Optimization Using Measure Properties

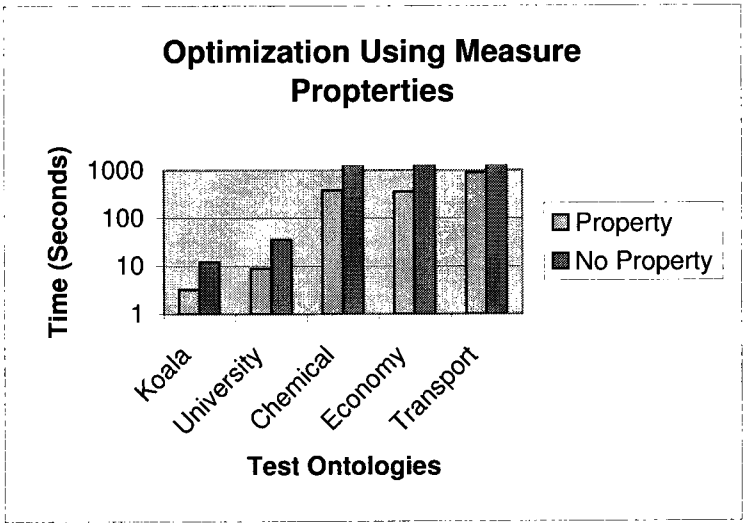

Figure 16: Evaluation of Optimization Based on Measure Properties

1. The experimental results confirm that the optimization based on the convergent property of inconsistency measure improves the running time for calculating the inconsistency measures. Overall, the optimization based on the property of inconsistency measure outperforms partitioning based on structural relevance. This is due to the fact that in large knowledge bases in the previous test on partitioning, e.g., Economy and Transport ontologies, the number of axioms in each partition is still too large as they are tightly coupled.

2. Although the Economy ontology is larger than the Chemical ontology in terms of the number of concepts and axioms, it took less time to measure its inconsistency. This shows that there are more sources of inconsistencies in the Economy ontology, and the number of axioms that has to be computed is significantly reduced after optimization based on the convergent property of the inconsistency measures.

7.3.3 Evaluation of Both Optimization Techniques

The experimental results shown in the previous subsections were evaluated by testing one optimization at a time. We now present the result of evaluating of both optimization techniques by first partitioning the knowledge base based on structural relevance, then for each partition, optimizing based on the convergent property of the inconsistent measure.

Figure 17 and Table 18 compare the performance with and without the two optimization techniques.

Ontology	With Optimization(T_P)	Without Optimization(T_{noP})	T_{noP}/T_P
Koala	1.77	12.01	6.79
University	2.60	35.12	13.50
Chemical	325.12	Time Out	N/A
Economy	294.28	Time Out	N/A
Transport	600.89	Time Out	N/A

Table 18: Optimization Using Both Techniques

It is interesting to note that when combined together, the partition based on

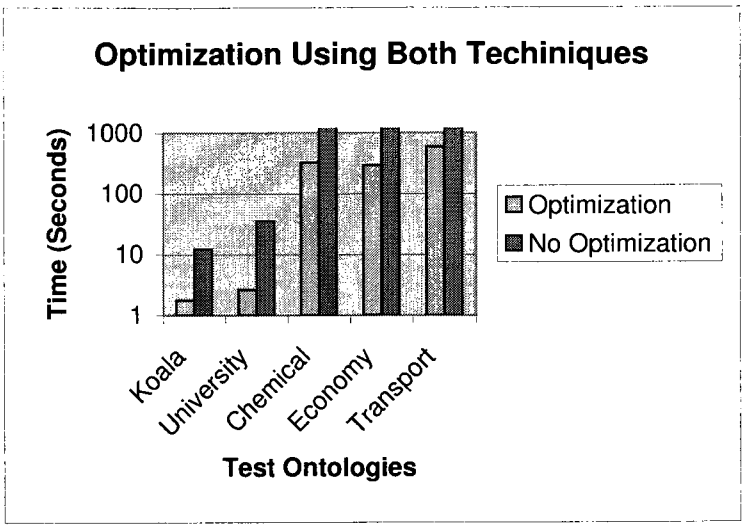

Figure 17: Evaluation of Both Optimization Techniques

structural relevance and the optimization based on the property of inconsistency measure can greatly enhance the performance. The evaluation results are much better than the previous results when testing one of the techniques alone. This is due to the fact that by first partitioning the knowledge base, the number of axioms is reduced for each partition. So we only have to consider computing the inconsistency measure for each partition, based on its convergent property.

7.4 Conclusion

In this chapter, we studied the performance of the inconsistency measure based diagnosis procedure and the proposed optimization techniques discussed in Chapter 6. Our experimental results showed that both optimization techniques can help

improve the performance. We observed from the experimental results that partitioning based on structural relevance works best when the knowledge base is loosely coupled in terms of structural relevance. The optimization based on the convergent property of the inconsistency measure was also found to be very effective in dealing with large knowledge bases, especially when the knowledge base has multiple sources of inconsistencies. We also noted that the overall effect of the combined optimization techniques is significant.

Part IV

Conclusion and Future Work

Chapter 8

Conclusion and Future Work

This thesis set out to tackle the problem of providing non-standard reasoning services, such as explanation and diagnosis, for DL reasoners. In this chapter we conclude the major contributions followed by a discussion of future research directions.

8.1 Summary

The following summaries the main contributions of this thesis.

1. As observed in many DL applications, explanation service plays an very important role in assisting users to find out the cause of an unsatisfiability or inconsistency problem. We proposed a framework to provide this service in DL

$\mathcal{ALCHI}$ knowledge bases based on resolution proofs. The framework translates the DL knowledge base into first-order logic formulae and uses an automated theorem prover to obtain the resolution proof and its corresponding refutation graph. By traversing the refutation graph, an explanation is generated and presented to users. This novelty of this approach is that it exploits the advantages of resolution based first-order logic theorem proving and provides explanations on a fine-grained level. Besides, the resolution technique is more focused, since all the literals involved in a proof contribute directly to the solution. This approach is also independent of any specific DL reasoners.

2. Based on this framework, we presented a sound and complete algorithm to generate explanations for the Description Logic language $\mathcal{ALCHI}$, by investigating unsatisfiability and inconsistency patterns in $\mathcal{ALCHI}$. This algorithm uses refutation graph transformation to provide explanations. The advantage of using refutation graph is that there is one refutation graph presentation for several resolution proofs. By traversing the graph, a good way to present the proof can be found. We demonstrated its practical value by implementing a prototype system and testing it with real world ontology applications.

3. As ontologies grow larger and increasingly complicated, inconsistencies become quite common in real world applications. Not all inconsistencies are equally bad, and some may be more problematic than the others. In order to repair an inconsistent ontology, we proposed a technique based on Shapley values to

measure inconsistencies in ontologies. Users of such an ontology can refer to the inconsistency measure to decide which axioms contribute the most to the inconsistency and need to be repaired. To the best of our knowledge, this is the first work in DL to provide a systematical and quantitative measure for diagnosis purposes.

4. We also investigated two optimization techniques to compute the inconsistency measure. The first is to partition the ontology based on structural relevance. The second technique is based on the convergent properties of the inconsistency measure. Our experimental results showed that both optimization techniques can help improve the performance.

8.2 Future Research

Our work in this thesis can be further extended in the following aspects:

1. The underlying DL language of the explanation framework is restricted to $\mathcal{ALCHI}$ and extensions to more expressive DL language are needed. A challenging direction is to extend the algorithms to handle number restrictions as well as nominals. Since most resolution based ATPs use the equality predicate to express number restrictions, this may pose new problems for explanations. Because the number of literals grows exponentially with the actual numbers, FOL formulae or clauses will become quite long and the performance might be

affected. One possible solution is to shift to DL reasoners to handle number restrictions.

2. The explanation is restricted to unsatisfiability and inconsistency queries. Although we believe explanations for this kind of queries are more useful for general users to debug their terminologies, providing explanations for subsumption, satisfiability and non-subsumption queries is also necessary. As pointed out in [MB95], an explanation can be generated by returning a model or a counter-example but more work needs to be done to identify the most suitable ones.

3. The proposed inconsistency measure is flexible in the sense that although we focused on inconsistency problems in Chapter 6, by choosing different inconsistency value functions, it can also address unsatisfiability problems. It is interesting to investigate what problems it can solve by adopting other inconsistency values.

Bibliography

[ARR+93] Tarun Arora, Raghu Ramakrishnan, William G. Roth, Praveen Seshadri, and Divesh Srivastava. Explaining Program Execution in Deductive Systems. In Deductive and Object-Oriented Databases, pages 101–119, 1993.

[BFH+99] Alexander Borgida, Enrico Franconi, Ian Horrocks, Deborah L. McGuinness, and Peter F. Patel-Schneider. Explaining $\mathcal{ALC}$ subsumption. In Proceedings of 1999 International Workshop on Description Logics (DL 1999), pages 37–40, 1999.

[BG01] Leo Bachmair and Harald Ganzinger. Resolution Theorem Proving. In John Alan Robinson and Andrei Voronkov, editors, Handbook of Automated Reasoning (in 2 volumes), pages 19–99. Elsevier and MIT Press, 2001.

[BHS07] Franz Baader, Ian Horrocks, and Ulrike Sattler. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors,

Handbook of Knowledge Representation. Elsevier, 2007.

[BMPSR90] Ronald J. Brachman, Deborah L. McGuiness, Peter F. Patel-Schneider, and Lori A. Resnick. Living with CLASSIC: when and how to use a KL-ONE-like language. In John Sowa, editor, Principles of semantic networks, pages 401–456. Morgan Kaufmann, San Mateo, US, 1990.

[BN07] F. Baader and W. Nutt. Basic Description Logic. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition), pages 47–104. Cambridge University Press, 2007.

[BP07] Franz Baader and Rafael Penaloza. Axiom Pinpointing in General Tableaux. In Proceedings of the 16th International Conference on Automated Reasoning with Analytic Tableaux and Related Methods TABLEAUX 2007, LNAI, Aix-en-Provence, France, 2007. Springer.

[BS84] Sharon Wraith Bennett and A. Carlisle Scott. Specialized Explanations for Dosage Selection. In Bruce G. Buchanan and Edward H. Shortliffe, editors, Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, pages 363–370. Reading, MA: Addison-Wesley, 1984.

[BS85] Ronald J. Brachman and James G. Schmolze. An overview of the KL-ONE knowledge representation system. Cognitive Science, 9(2):171–216, 1985.

[BS01] Franz Baader and Ulrike Sattler. An Overview of Tableau Algorithms for Description Logics. Studia Logica, 69(1):5–40, 2001.

[BS08] Franz Baader and Boontawee Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In Proceedings of the 3rd Knowledge Representation in Medicine (KR-MED'08): Representing and Sharing Knowledge Using SNOMED, volume 410 of CEUR-WS, 2008.

[Byr80] Lawrence Byrd. Understanding the control flow of Prolog programs. In Proceedings of the 1980 Logic Programming Workshop, pages 127–138, Debrecen, Hungary, 1980.

[cas] CaSC ATP Competition. `http://www.cs.miami.edu/~tptp/CASC/`. Accessed November 12, 2009.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In Proceedings of the 12th International Conference on Computer Aided Verification, pages 154–169, London, UK, 2000. Springer-Verlag.

[CGP99] Edmund M. Clarke, Jr., Orna Grumberg, and Doron A. Peled. Model checking. MIT Press, Cambridge, MA, USA, 1999.

[Cla81] William J. Clancey. The epistemology of a rule-based expert system: a framework for explanation. Technical report, Stanford University, Stanford, CA, USA, 1981.

[Cla94] William J. Clancey. Notes on "heuristic classification". pages 191–196, 1994.

[CS04] Vincent Conitzer and Tuomas Sandholm. Computing Shapley Values, Manipulating Value Division Schemes, and Checking Core Membership in Multi-Issue Domains. In Deborah L. McGuinness and George Ferguson, editors, Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence (AAAI 2004), pages 219–225, San Jose, California, USA, 2004. AAAI Press/The MIT Press.

[DDC03] Alexey Tsymbal Dónal Doyle and Pádraig Cunningham. A Review of Explanation and Explanation in Case-Based Reasoning. Technical Report TCD-CS-2003-41, Department of Computer Science, Trinity College, Dublin, 2003.

[DHS05a] Xi Deng, Volker Haarslev, and Nematollaah Shiri. A Framework for Explaining Reasoning in Description Logics. In Proceedings of the AAAI Fall Symposium on Explanation-aware Computing, pages 189–204, Washington, DC, USA, 2005. AAAI Press.

[DHS05b] Xi Deng, Volker Haarslev, and Nematollaah Shiri. A Resolution Based Framework to Explain Reasoning in Description Logics. In Proceedings of 2005 International Workshop on Description Logics (DL 2005), Edinburgh, UK, 2005.

[DHS06] Xi Deng, Volker Haarslev, and Nematollaah Shiri. Resolution Based Explanations for Reasoning in the Description Logic $\mathcal{ALC}$. In Proceedings of the Canadian Semantic Web Working Symposium, pages 55–61, Quebec City, Canada, 2006. Springer.

[DHS07a] Xi Deng, Volker Haarslev, and Nematollaah Shiri. Measuring Inconsistencies in Ontologies. In Proceedings of the 4th European Semantic Web Conference(ESWC 2007), pages 55–61, Innsbruck, Austria, 2007. Springer.

[DHS07b] Xi Deng, Volker Haarslev, and Nematollaah Shiri. Using Patterns to Explain Inferences in $\mathcal{ALCHI}$. Journal of Computational Intelligence, 23(3):373–392, 2007.

[Don07] F. M. Donini. Complexity of Reasoning. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition), pages 105–146. Cambridge University Press, 2007.

[dSMF06] Paulo Pinheiro da Silva, Deborah L. McGuinness, and Richard Fikes. A proof markup language for semantic web services. Information Systems, 31(4):381–395, 2006.

[Eis91] Norbert Eisinger. Completeness, Confluence, and Related Properties of Clause Graph Resolution. Morgan Kaufmann Publishers Inc., 1991.

[Fie01] Armin Fiedler. Dialog-driven Adaptation of Explanations of Proofs. In Bernhard Nebel, editor, Proceedings of the 17th International Joint Conference on Artificial Intelligence (IJCAI), pages 1295–1300, Seattle, WA, 2001. Morgan Kaufmann.

[Fit96] Melvin Fitting. First-order logic and automated theorem proving (2nd ed.). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.

[FM87] Amy Felty and Dale Miller. Proof Explanation and Revision. Technical Report MS-CIS-88-17, University of Pennsylvania, 1987.

[GdN99] H. Ganzinger and H. de Nivelle. A superposition decision procedure for the guarded fragment with equality. In Proceedings of the 14th

Annual IEEE Symposium on Logic in Computer Science (LICS '99), page 295, Washington, DC, USA, 1999. IEEE Computer Society.

[HF96] Xiaorong Huang and Armin Fiedler. Presenting Machine-Found Proofs. In Michael A. McRobbie and John K. Slaney, editors, Proceedings of the 13th International Conference on Automated Deduction (CADE-13), volume 1104 of Lecture Notes in Computer Science, pages 221–225, New Brunswick, NJ, USA, 1996. Springer.

[HK06] Anthony Hunter and Sóbastien Konieczny. Shapley inconsistency values. In Proceedings of the International Conference on Knowledge Representation (KR'06), pages 249–259, Windermere, UK, 2006. AAAI Press.

[HM01] Volker Haarslev and Ralf Möller. RACER System Description. In T. Nipkow R. Gori, A. Leitsch, editor, Proceedings of International Joint Conference on Automated Reasoning (IJCAR 2001), pages 701–705, Siena, Italy, 2001. Springer-Verlag.

[HMS05] U. Hustadt, B. Motik, and U. Sattler. Data Complexity of Reasoning in Very Expressive Description Logics. In Proceedings of Nineteenth International Joint Conference on Artificial Intelligence (IJCAI 2005), pages 466–471, Edinburgh, UK, AUG 2005.

[Hor07] I. Horrocks. Implementation and Optimization Techniques. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition), pages 329–374. Cambridge University Press, 2007.

[HPS08] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and Precise Justifications in OWL. In Proceedings of the 7th International Conference on The Semantic Web, pages 323–338, Berlin, Heidelberg, 2008. Springer-Verlag.

[HPS09] Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Lemmas for justifications in OWL. In Proceedings of the 2009 International Workshop on Description Logics (DL2009), Oxford, United Kingdom, 2009.

[HPSMW07] Ian Horrocks, Peter F. Patel-Schneider, Deborah L. McGuinness, and Christopher A. Welty. OWL: a Description Logic Based Ontology Language for the Semantic Web. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, The Description Logic Handbook: Theory, Implementation, and Applications (2nd Edition), pages 458–486. Cambridge University Press, 2007.

[HS05] Peter Haase and Ljiljana Stojanovic. Consistent Evolution of OWL Ontologies. In Asuncin Gmez-Prez and Jrme Euzenat, editors, Proceedings of the Second European Semantic Web Conference, volume 3532 of Lecture Notes in Computer Science, pages 182–197, Heraklion, Greece, MAY 2005. Springer.

[Hua94] Xiaorong Huang. Reconstructing Proofs at the Assertion Level. In Alan Bundy, editor, Proceedings of the 12th International Conference on Automated Deduction (CADE-12), pages 738–752. Springer-Verlag, 1994.

[Hua96] Xiaorong Huang. Translating Machine-Generated Resolution Proofs into ND-Proofs at the Assertion Level. In Norman Y. Foo and Randy Goebel, editors, Proceedings of the 4th Pacific Rim International Conference on Artificial Intelligence: Topics in Artificial Intelligence, volume 1114 of Lecture Notes in Computer Science, pages 399–410. Springer, 1996.

[Kal01] John Arnold Kalman. Automated Reasoning with Otter. Rinton Press, 2001.

[Kal06] Aditya Kalyanpur. Debugging and repair of OWL ontologies. PhD thesis, University of Maryland at College Park, 2006.

[KPS+05] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A Web Ontology Editing Browser. Journal of Web Semantics, 4:2005, 2005.

[KPSG06] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in OWL ontologies. In York Sure and John Domingue, editors, Proceedings of the 3rd European Semantic Web Conference (ESWC 2006), volume 4011 of Lecture Notes in Computer Science, pages 170–184, Budva, Montenegro, 2006. Springer.

[LD04] Carmen Lacave and Francisco J. Diez. A review of explanation methods for heuristic expert systems. The Knowledge Engineering Review, 19(2):133–146, 2004.

[LH05] Thorsten Liebig and Michael Halfmann. Explaining Subsumption in $\mathcal{ALEHF}_{R^+}$ TBoxes. In Ian Horrocks, Ulricke Sattler, and Frank Wolter, editors, Proceedings of the 2005 International Workshop on Description Logics(DL2005), pages 144–151, Edinburgh, Scotland, July 2005.

[Llo87] John Wylie Lloyd. Foundations of Logic Programming (2nd Edition). Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1987.

[LN05] Thorsten Liebig and Olaf Noppens. OntoTrack: A semantic approach for ontology authoring. Journal of Web Semantics, 3(2-3):116–131, October 2005.

[Mac91] Robert M. MacGregor. Inside the LOOM description classifier. SIGART Bulletin, 2(3):88–92, 1991.

[MB95] Deborah L. McGuinness and Alexander Borgida. Explaining subsumption in description logics. In Proceedings of the tenth International Joint Conference on Artificial Intelligence (IJCAI'95), pages 816–821, Montreal, Canada, 1995.

[McG96] Deborah McGuinness. Explaining Reasoning in Description Logics. PhD thesis, Rutgers University, New Brunswick, New Jersey, 1996.

[MD99] Sarah Mallet and Mireille Ducasse. Generating Deductive Database Explanations. In International Conference on Logic Programming, pages 154–168, 1999.

[MdS04] Deborah L. McGuinness and Paulo Pinheiro da Silva. Explaining answers from the semantic web: the inference web approach. Journal of Web Semantics, 1(4):397–413, 2004.

[Mei00] Andreas Meier. TRAMP: Transformation of Machine-Found Proofs into Natural Deduction Proofs at the Assertion Level. In

D. McAllester, editor, Proceedings of the 17th Conference on Automated Deduction (CADE–17), volume 1831 of Lecture Notes in Artificial Intelligence, pages 460–464, Pittsburgh, USA, 2000. Springer Verlag, Berlin, Germany.

[Min74] Marvin Minsky. A framework for representing knowledge. Technical report, Massachusetts Institute of Technology, Cambridge, MA, USA, 1974.

[Moo94] Johanna D. Moore. Participating in explanatory dialogues: interpreting and responding to questions in context. MIT Press, Cambridge, MA, USA, 1994.

[MSH09] Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. Journal of Artificial Intelligence Research, 36:165–228, 2009.

[MWW89] R. Marti, C. Wieland, and Beat Wüthrich. Adding Inferencing to a Relational Database Management System. In Theo Härder, editor, Datenbanksysteme in Büro, Technik und Wissenschaft, pages 266–270, 1989.

[NRW98] A. Nonnengart, G. Rock, and C. Weidenbach. On Generating Small Clause Normal Forms. In C. Kirchner and H. Kirchner, editors, Proceedings of the 15th International Conference on Automated Deduction,

number 1421 in Lecture Notes in Artificial Intelligence, pages 397–411. Springer-Verlag, 1998.

[owl] OWL2 Web Ontology Language Document Overview. `http://www.w3.org/TR/owl2-overview/`. Accessed February 26, 2010.

[PSK05] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging OWL ontologies. In Proceedings of the 14th International World Wide Web Conference (WWW 2005), pages 633–640, Chiba, Japan, May 2005. ACM Press.

[Qui67] M. R. Quillian. Word concepts: A theory and simulation of some basic capabilities. Behavioral Science, 5(12):410–430, 1967.

[Rei87] R Reiter. A theory of diagnosis from first principles. Artificial Intelligence, 32(1):57–95, 1987.

[RSS92] Raghu Ramakrishnan, Divesh Srivastava, and S. Sudarshan. CORAL - Control, Relations and Logic. In Proceedings of the 18th International Conference on Very Large Data Bases, pages 238–250, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[RU93] Raghu Ramakrishnan and Jeffrey D. Ullman. A survey of research on deductive database systems. Journal of Logic Programming, 23(2):125–149, 1993.

[RV01] John Alan Robinson and Andrei Voronkov, editors. Handbook of Automated Reasoning (in 2 volumes). Elsevier and MIT Press, 2001.

[RV02] Alexandre Riazanov and Andrei Voronkov. The design and implementation of VAMPIRE. AI Communications, 15(2,3):91–110, 2002.

[SC03] S. Schlobach and R. Cornet. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In Proceedings of the eighteenth International Joint Conference on Artificial Intelligence (IJCAI'03), pages 355–362, Acapulco, Mexico, 2003. Morgan Kaufmann.

[SCDS84] A. Carlisle Scott, William J. Clancey, Randall Davis, and Edward H. Shortliffe. Methods for generating explanations. In Bruce G. Buchanan and Edward H. Shortliffe, editors, Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, pages 338–362. Reading, MA: Addison-Wesley, 1984.

[Sch04] Stefan Schlobach. Explaining Subsumption by Optimal Interpolation. In José Júlio Alferes and João Alexandre Leite, editors, Proceedings of the 9th European Conference on Logics in Artificial Intelligence (JELIA '04), pages 413–425. Springer, 2004.

[Sch05] S. Schlobach. Diagnosing Terminologies. In Proceedings, The Twentieth National Conference on Artificial Intelligence and

the Seventeenth Innovative Applications of Artificial Intelligence Conference (AAAI 2005), pages 670–675, Pittsburgh, Pennsylvania, USA, 2005. AAAI Press.

[Sha53] Lloyd Shapley. A value for n-person games. In H. Kuhn and A. Tucker, editors, Contributions to the Theory of Games, volume 2, pages 307–317. Princeton University Press, 1953.

[SP04] Evren Sirin and Bijan Parsia. Pellet: An OWL DL Reasoner. In Proceedings of the 2004 International Workshop on Description Logics (DL2004), Whistler, British Columbia, Canada, 2004.

[SSS91] Manfred Schmidt-Schaubßand Gert Smolka. Attributive concept descriptions with complements. Artificial Intelligence, 48(1):1–26, 1991.

[ST90] O. Shmueli and S. Tsur. Logical Diagnosis of LDL Programs. In D. H. D. Warren and P. Szeredi, editors, Logic Programming: Proceedings of the Seventh International Conference, pages 112–129. MIT Press, Cambridge, MA, 1990.

[Swa83] William R. Swartout. XPLAIN: A System for Creating and Explaining Expert Consulting Programs. Artificial Intelligence, 21(3):285–325, 1983.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ description logic reasoner: System description. In Proceedings of the International Joint

Conference on Automated Reasoning (IJCAR 2006), volume 4130 of Lecture Notes in Artificial Intelligence, pages 292–297, Seattle, Washington, USA, 2006. Springer.

[w3c] OWL Web Ontology Language Overview. `http://www.w3.org/TR/owl-features/`. Accessed August 4, 2008.

[Wie90] C. Wieland. Two Explanation Facilities for the Deductive Database Management System DeDEx. In Hannu Kangassalo, editor, Proceedings of the 9th International Conference on Entity-Relationship Approach (ER'90), pages 189–203. ER Institute, 1990.

[WS84] Jerold W. Wallis and Edward H. Shortliffe. Customized explanations using causal knowledge. In Bruce G. Buchanan and Edward H. Shortliffe, editors, Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project, pages 371–388. Reading, MA: Addison-Wesley, 1984.

[WT92] Michael R. Wick and William B. Thompson. Reconstructive expert system explanation. Artificial Intelligence, 54(1-2):33–70, 1992.

[WW97] McCune WosMcCune and Wos.L. Otter - the CADE-13 competition incarnations. Journal of Automated Reasoning, 18(2):211–220, 1997.

Appendix A

Glossary

ABox : Assertional Box

$\mathcal{AL}$: A minimal language of interest in Description Logic which includes atomic concept, atomic role, atomic negation, conjunction, value restriction and limited existential quantification

$\mathcal{ALC}$: $\mathcal{AL}$ extended with concept negation

$\mathcal{ALCHI}$: $\mathcal{ALC}$ extended with role hierarchy and inverse role

$\mathcal{ALEHFR}+$: $\mathcal{AL}$ extended with full existential quantification, role hierarchy, functional role and transitive role

ATP : Automated Theorem Prover

$\mathcal{EL}+$: a Description Logic language which includes conjunction, existential quantification and complex role inclusion

CNF : Conjunctive Normal Form

DL : Description Logic

FOL : First-Order Logic

GCI : General Concept Inclusion

GITS : Generalized Incoherence-Preserving Terminology

MIPS : Minimal Incoherence-Preserving Sub-TBoxes

MIS : Minimal Inconsistent Subset

MUPS : Minimal Unsatisfiability-Preserving Sub-TBoxes

MUS : Minimal Unsatisfiable Subset

OWL : Web Ontology Language

OWL2 : Web Ontology Language 2

RDF : Resource Description Framework

$\mathcal{SHIF}(\mathcal{D})$: $\mathcal{ALC}$ extended with transitive role, role hierarchy, inverse role, data type and functional role

$\mathcal{SHIQ}$: $\mathcal{ALC}$ extended with transitive role, role hierarchy, inverse role, and qualified number restriction

$\mathcal{SHOIN}(\mathcal{D})$: $\mathcal{ALC}$ extended with transitive role, role hierarchy, nominal, inverse role, data type and unqualified number restriction

TBox : Terminological Box

W3C : World Wide Web Consortium