UNCERTAINTY MANAGEMENT FOR DESCRIPTION LOGIC-BASED ONTOLOGIES

Hsueh-Ieng Pai

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

April 2008

Your file Votre référence
ISBN: 978-0-494-37747-5
Our file Notre référence
ISBN: 978-0-494-37747-5

Abstract

Uncertainty Management for Description Logic-Based Ontologies

Hsueh-Ieng Pai, Ph.D.

Concordia University, 2008

Description Logics (DLs) play an important role in the Semantic Web as the foundation of ontology language OWL DL. The standard DLs, based on the classical logic, are more suitable to describe concepts that are crisp and well-defined in nature. However, for many emerging applications, we also need to deal with uncertainty. In recent years, a number of proposals have been put forward for incorporating uncertainty in DL frameworks. While much progress has been made on the representation and reasoning of the uncertainty knowledge, query optimization in this context has received little attention.

In this thesis, we tackle this problem in both theoretical and practical aspects by taking a generic approach. We first propose the $\mathcal{ALC}_U$ framework which extends the standard DL $\mathcal{ALC}$ with uncertainty. This is done by extending each component of the $\mathcal{ALC}$ framework, including the description language, the knowledge base, and the reasoning procedure. In particular, the resulting semantics of the description language is captured using the certainty lattice and the combination functions. The

knowledge base is extended by associating with each axiom and assertion a certainty value and a pair of combination functions to interpret the concepts that appear in the axiom/assertion. A sound, complete, and terminating tableau reasoning procedure is developed to handle such uncertainty knowledge bases. An interesting feature of the $\mathcal{ALC}_U$ framework is that, by simply tuning the combination functions that are associated with the axioms and assertions, different notions of uncertainty can be modeled and reasoned with, using a single reasoning procedure.

Using this framework as the basis, we then investigate optimization techniques in our context. We adapt existing optimization techniques developed for standard DLs and other software systems to deal with uncertainty. New techniques are also developed to optimize the handling of uncertainty constraints generated by the reasoning procedure.

In terms of practical contribution, we developed a running prototype, URDL – an Uncertainty Reasoner for DL $\mathcal{ALC}_U$, which implements the proposed optimization techniques. Experimental results show the practical merits of the $\mathcal{ALC}_U$ framework, as well as the effectiveness of the proposed optimization techniques, especially when dealing with large knowledge bases.

Acknowledgments

First of all, I would like to thank my supervisors, Dr. Volker Haarslev and Dr. Nematollaah Shiri, for their guidance, insightful discussion, encouragement, and support throughout the development of this thesis. Dr. Haarslev introduced me to the world of the Semantic Web and Description Logics, and Dr. Shiri introduced me to the field of uncertainty management and knowledge base systems. Without them, this work would not have been possible.

I would like to give special thanks to my brother, Dr. Cheng-Yu Pai. His insightful discussion inspired my research on the optimization aspects of the uncertainty management.

This work is supported by the following agencies which I would like to acknowledge: Natural Sciences and Engineering Re-search Council (NSERC) of Canada, Genome Québec, and Faculty of Engineering and Computer Science (ENCS), Concordia University.

I would like to express my gratitude to my fellow students, Ali Kiani and Xi Deng, with whom we spent countless time together throughout the past few years. My colleagues at the Database Research Lab, especially Mihail Halachev, Qiong Huang,

Nima Mohajerin, Ahmed Alasoud, Srinidhi Kannappady, and Anand Thamildurai, also helped me with valuable comments and suggestions regarding my technical presentations.

Finally, I would like to dedicate this thesis to my family. I thank my parents and brother for their love, support, and encouragement. This work is as much theirs as it is mine.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There is a huge amount of heterogeneous information available on the Web today. In year 2000, the size the entire Web was estimated to be 1 billion pages [She00]. This number increased to over 19 billion in 2005 [Mar05]. This trend of phenomenal information growth is most likely to continue in the future. Hence, there is an increasing need for finding efficient ways to locate desired information on the Web.

The current Web, also known as the *Syntactic Web* [Wik07], consists of information that is mostly encoded using markup languages such as HyperText Markup Language (HTML) [W3C99]. Such information does not contain any special tagging to convey the meaning of the Web content. This makes it difficult for computers to understand the current Web content. Although humans are capable of understanding the content, the amount of information available on the Web is too huge for humans to handle. To overcome this problem, it is necessary to rely on the computational power of the machines to make sense of the huge amount of information on the Web.

1.1 Towards the Semantic Web

The vision of the *Semantic Web* [BHL01] was first introduced by Tim Berners-Lee as "a Web of data that can be processed directly or indirectly by machines" [BF00]. The idea is to make Web resources more machine-interpretable by giving them a well-defined meaning through semantic markups.

To move toward this goal, the *Resource Description Framework* (RDF) [W3C04d] was developed by the World Wide Web Consortium (W3C) as a language to represent information about Web resources [W3C04b]. RDF uses Extensible Markup Language (XML) [W3C06a] syntax, and describes each Web resource using a triple consisting of the *subject*, the *predicate*, and the *object*. A subject is anything that can be identified by a Universal Resource Identifier (URI) [W3C06b]. A predicate expresses the relationship between the resource and its value. Finally, an object may be atomic (such as a string, an integer, etc.) or can be another resource [Tau06].

RDF Schema (RDFS) [W3C04c], also developed by W3C, further extended RDF by giving additional semantics to particular RDF predicates and resources, such as classes, class inheritance, properties, property inheritance, and domain/range restrictions. This gives extra hints to computers as to how the terms and their relationships should be interpreted. However, RDFS is still not enough to describe resources in sufficient details. For instance, it lacks support for existential and universal constraints, and cannot represent general axioms. As a result, more expressive languages were needed to better describe the meaning of Web resources.

1.2 Description Logic-Based Ontology Languages

In recent years, a number of expressive Semantic Web *ontology languages* were proposed [DAM00, HFB+00, W3C01, W3C04a]. An *ontology* is "an explicit specification of a conceptualization" [Gru93]. Informally, an ontology consists of a set of terms in a domain, the relationships between the terms, and a set of constraints imposed on the way in which those terms can be combined. Constraints such as concept conjunction, disjunction, negation, existential quantifier, and universal quantifier can all be expressed using ontology languages. By explicitly defining the relationships and constraints among the terms, the semantics of a term is constrained by restricting the number of possible interpretations of the term [BP02]. This makes the meaning of the terms more precise and better understandable.

Among Semantic Web ontology languages [DAM00, HFB+00, W3C01, W3C04a], the OWL Web Ontology Language [W3C04a] is the most recent W3C Recommendation. One of its species, OWL DL, is named because of its correspondence with *Description Logics* (DLs) [BCM+03]. The family of DLs is mostly a subset of first-order logic (FOL) [Hod01] that is considered to be attractive because it keeps a good compromise between the expressive power and the computational tractability [BCM+03]. The well-defined semantics as well as the availability of the powerful reasoning tools make the family of DLs particularly interesting to the Semantic Web community [BHS02].

1.3 Motivation for Extending Description Logics with Uncertainty

The standard DLs, such as the one that OWL DL is based on, focus on the classical logic, which is more suitable to describe concepts that are crisp and well-defined in nature. However, in the real-world applications, *uncertainty*, which refers to a form of deficiency or imperfection in the information for which the truth of such information is not established definitely [LS01a], is everywhere. Not only because the real-world information is mostly imperfect or deficient, but also because many realistic applications need the capability to handle uncertainty – from classification of genes in bioinformatics, schema matching in information integration, to matchmaking in Web services. Modeling uncertainty and reasoning with it have been challenging issues for over two decades in database and artificial intelligence research [Bac90, LS01a, MS97, Par96], and is inevitably also a challenge for the Semantic Web/DL community.

The need to model and reason with uncertainty has been found in many different Semantic Web contexts, such as Semantic Web services [MR05], multimedia annotation [VSP05], ontology learning [HV05], and bioinformatics [SEW+07]. For example, in an online medical diagnosis system, one might want to find out to what degree a person, John, would have heart disease if the certainty that an obese person would have heart disease lies between 0.7 and 1, and John is obese with a degree between 0.8 and 1. Such knowledge cannot be expressed nor reasoned with standard DLs. Indeed, to support uncertainty, the syntax and the semantics of the standard DLs

need to be extended.

1.4 Motivation and Objectives of the Thesis

In the last few years, a number of frameworks have been proposed on extending DLs with uncertainty [Jae94b, KLP97, TM98, Str01, GL02a, PFT+04, Str05a, SSSP06, SB07]. Some of them deal with only vagueness while others deal with only probabilistic knowledge. In addition, existing frameworks focus on representation and reasoning of the uncertainty knowledge, but have paid little attention to efficient query processing and implementation. Rather than addressing these issues for individual frameworks, we are interested in studying these problems so that the results obtained and the tools developed will be useful over a spectrum of frameworks for DL with uncertainty. This aim is further developed into the following objectives:

1. To propose a generic framework that extends the DL $\mathcal{ALC}$ [SS91], a DL fragment that is of practical interest, so that uncertainty knowledge can be represented and reasoned with in a uniformed way.

2. To develop query optimization techniques for the proposed framework.

3. To develop a system to show the practical merits of the proposed framework and to serve as the test bed for the optimization techniques.

1.5 Contributions of the Thesis

To fulfill the above objectives, this thesis makes the following main contributions:

- We classify the existing frameworks for DLs with uncertainty into three approaches based on the underlying mathematical foundation and the type of uncertainty modeled (Section 2.3). In addition, we identify the common components of the DL frameworks that need to be extended in order to support uncertainty. A variation of this work will appear in [HPS09].

- The $\mathcal{ALC}_U$ framework is proposed (Chapter 3), which extends the $\mathcal{ALC}$ framework by allowing various forms of uncertainty knowledge be expressed and reasoned with. To work towards this goal, we make the following contributions.

  - We define the semantics of the description language $\mathcal{ALC}_U$ based on the underlying certainty lattice as well as the combination functions that satisfy some pre-defined properties (Section 3.1).

  - We present the syntax and semantics of $\mathcal{ALC}_U$ knowledge bases, which allow various forms of uncertainty knowledge be expressed by changing the combination functions that are associated with axioms and assertions (Section 3.2).

  - We propose a generic reasoning procedure that can deal with uncertainty knowledge expressed in $\mathcal{ALC}_U$ (Section 3.3).

The work related to $\mathcal{ALCC}_U$ description language and knowledge base was published in [HPS05]. That work was further extended in [HPS06b] with a core reasoning procedure. A reasoning procedure for dealing with acyclic uncertainty knowledge bases was presented in [HPS06a], which was then extended for general knowledge bases [HPS] as presented in this thesis.

- We establish soundness and completeness of the proposed $\mathcal{ALCC}_U$ reasoning procedure, and show that it terminates (Section 3.3.4). This work is under evaluation [HPS].

- We study applicability of the standard DL optimization techniques to the $\mathcal{ALCC}_U$ framework, and propose new optimization techniques in our context (Chapter 4). We show that some optimization techniques from standard DLs cannot be applied to the $\mathcal{ALCC}_U$ framework. Even for those that can be applied, care must be taken when uncertainty is present. Most of these techniques were presented in [HPS07].

- We developed a running prototype which shows the practical merits of the $\mathcal{ALCC}_U$ framework. It also served as a test bed for the proposed optimization techniques (Chapter 5).

- We investigate the performance of the $\mathcal{ALCC}_U$ system, and the effectiveness of the proposed optimization techniques (Chapter 6). Some preliminary experimental results were published in [HPS07].

1.6 Thesis Outline

This thesis is organized as follows. Chapter 2 gives an overview of the $\mathcal{ALC}$ framework, and presents the related work. Chapter 3 presents the $\mathcal{ALC}_U$ framework in detail, along with an illustrating example. We also compare the $\mathcal{ALC}_U$ framework with the related work. In Chapter 4, we study the optimization techniques used in the standard DLs and other software systems, and investigate how they can be adapted to the $\mathcal{ALC}_U$ framework. New techniques are also presented to optimize the uncertainty reasoning procedure. Chapter 5 presents an $\mathcal{ALC}_U$ system named URDL, an Uncertainty Reasoner for the DL $\mathcal{ALC}_U$. We study the performance of URDL and the effectiveness of the proposed optimization techniques in Chapter 6. Finally, Chapter 7 concludes the thesis and presents some possible future improvements.

Chapter 2

Background and Related Work

This chapter describes relevant background knowledge and work related to this thesis. We first give an overview of the $\mathcal{ALC}$ framework in Section 2.1, which we use as the basis to extend with uncertainty in Chapter 3. We then present related work. Our work on the $\mathcal{ALC}_U$ framework was inspired by the parametric framework for deductive database with uncertainty, which we present in Section 2.2. We also survey the existing frameworks for DLs with uncertainty in Section 2.3.

2.1 The $\mathcal{ALC}$ framework

Description logics (DLs) are a family of knowledge representation languages that can be used to represent the knowledge of an application domain using concept *descriptions*, and have formal, *logic*-based semantics [BHS07, BL06, BCM+03]. In this thesis, we focus on the DL fragment called $\mathcal{ALC}$ [SS91], which corresponds to the

propositional multi-modal logic $\mathbf{K_{(m)}}$ [Sch91].

The $\mathcal{ALC}$ framework consists of three main components – the description language, the knowledge base, and the reasoning procedure (see Figure 1). In what follows, we present each of these components in detail.

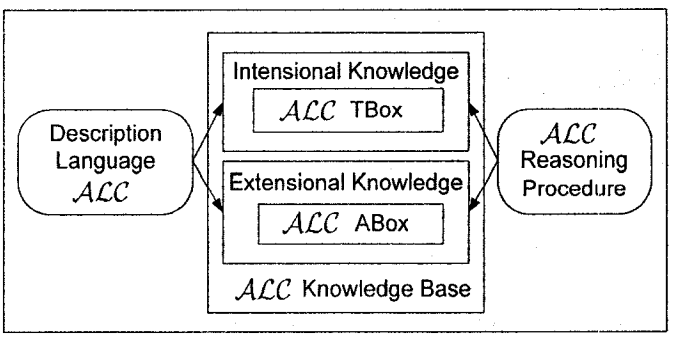

Figure 1: The $\mathcal{ALC}$ Framework

2.1.1 Description Language $\mathcal{ALC}$

The description language refers to the language used for building concepts. Description languages are distinguished by the constructors they provide. By adding more constructors to the description language, more expressive languages may be obtained.

Every description language has elementary descriptions which include atomic concepts (unary predicates) and atomic roles (binary predicates). Complex descriptions can then be built inductively from concept constructors. The description language $\mathcal{ALC}$ consists of a set of language constructors that are of practical interest. Specifically, let R be a role name, the syntax of a concept description (denoted C or D) in $\mathcal{ALC}$ is described as follows, where the name of each rule is given in parenthesis.

$C, D \rightarrow A$ (Atomic Concept) |

$\neg C$ (Concept Negation) |

$C \sqcap D$ (Concept Conjunction) |

$C \sqcup D$ (Concept Disjunction) |

$\exists R.C$ (Role Exists Restriction) |

$\forall R.C$ (Role Value Restriction)

For example, let *Person* be an atomic concept and *hasParent* be a role. Then $\forall hasParent.Person$ is a concept description. We use $\top$ as a synonym for $A \sqcup \neg A$, and $\bot$ as a synonym for $A \sqcap \neg A$.

The semantics of the description language is defined using the notion of interpretation. An interpretation $\mathcal{I}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty domain of the interpretation, and $\cdot^{\mathcal{I}}$ is an interpretation function that assigns to every atomic concept A a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and to every atomic role R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretations of concept descriptions are shown below:

$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash C^{\mathcal{I}}$

$(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$

$(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$

$(\exists R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} |\ \exists b : (a, b) \in R^{\mathcal{I}} \wedge b \in C^{\mathcal{I}}\}$

$(\forall R.C)^{\mathcal{I}} = \{a \in \Delta^{\mathcal{I}} |\ \forall b : (a, b) \in R^{\mathcal{I}} \rightarrow b \in C^{\mathcal{I}}\}$

2.1.2 $\mathcal{ALC}$ Knowledge Base

As shown in Figure 1, the knowledge base Σ consists of two components: the intensional knowledge denoted by $\mathcal{T}$ (the Terminological Box, known as the TBox) and the extensional knowledge denoted by $\mathcal{A}$ (Assertional Box, known as the ABox).

$\mathcal{ALC}$ TBox

A TBox $\mathcal{T}$ is a set of statements about how concepts in an application domain are related to each other. Let A be an atomic concept, and C and D be concept descriptions. Also, let $\sqsubseteq$ denote the subsumption (sub-class/super-class) relationship, and let $\equiv$ denote the equality relationship. The TBox is a finite, possibly empty, set of terminological axioms that could be a combination of:

- *Concept subsumptions* of the form $\langle A \sqsubseteq C \rangle$
- *Concept definitions* of the form $\langle A \equiv C \rangle$, which is equivalent to $\langle A \sqsubseteq C \rangle$ and $\langle C \sqsubseteq A \rangle$
- *General concept inclusions* (*GCIs*) of the form $\langle C \sqsubseteq D \rangle$
- *Concept equations* of the form $\langle C \equiv D \rangle$, which is equivalent to $\langle C \sqsubseteq D \rangle$ and $\langle D \sqsubseteq C \rangle$

For example, the axiom $\langle ObesePerson \equiv Person \sqcap Obese \rangle$ is a concept definition that is equivalent to the two concept subsumptions: $\langle ObesePerson \sqsubseteq Person \sqcap Obese \rangle$ and $\langle Person \sqcap Obese \sqsubseteq Obese\ \ Person \rangle$. Note also that a GCI is simply a more

general form of concept subsumption, and a concept equation is a more general form of concept definition.

An interpretation $\mathcal{I}$ satisfies $\langle C \sqsubseteq D \rangle$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$, and it satisfies $\langle C \equiv D \rangle$ if $C^{\mathcal{I}} = D^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies a TBox $\mathcal{T}$ iff $\mathcal{I}$ satisfies every axiom in $\mathcal{T}$.

$\mathcal{ALC}$ ABox

An ABox $\mathcal{A}$, or a world description, is a set of statements that describe a specific state of affairs, with respect to some individuals, of an application domain in terms of concepts and roles. Let a and b be individuals, C be a concept, R be a role, and let ":" denote "is an instance of". The ABox consists of a set of assertions that could be a combination of:

- *Concept assertions* of the form $\langle a : C \rangle$
- *Role assertions* of the form $\langle (a, b) : R \rangle$

For example, the concept assertion $\langle John : Obese \rangle$ asserts that individual *John* is an instance of concept *Obese*. Similarly, the role assertion $\langle (John, Mary) : hasParent \rangle$ asserts that *John*'s parent is *Mary*.

An interpretation $\mathcal{I}$ satisfies $\langle a : C \rangle$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$, and it satisfies $\langle (a, b) : R \rangle$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies an ABox $\mathcal{A}$ iff $\mathcal{I}$ satisfies every assertion in $\mathcal{A}$ with respect to a TBox $\mathcal{T}$.

An interpretation $\mathcal{I}$ *satisfies* (or is a *model* of) a knowledge base Σ (denoted $\mathcal{I} \models \Sigma$), if and only if it satisfies both components of Σ. The knowledge base Σ

is *consistent* if there exists an interpretation $\mathcal{I}$ that satisfies Σ. We say that Σ is *inconsistent* otherwise.

Definition 2.1.1 (Predecessor/Ancestor) Let R be a role, and a and b be individuals. An individual a is a *predecessor* of an individual b (or b is an R-successor of a) if the ABox $\mathcal{A}$ contains the assertion $\langle (a, b) : R \rangle$. An individual a is an *ancestor* of b if it is either a predecessor of b or there exists a chain of assertions $\langle (a, b_1) : R_1 \rangle$, $\langle (b_1, b_2) : R_2 \rangle$,..., $\langle (b_k, b) : R_{k+1} \rangle$ in $\mathcal{A}$.

2.1.3 $\mathcal{ALC}$ Reasoning Procedure

The purpose of the reasoning procedure is to explicate knowledge that is stored implicitly in a give knowledge base. Most DL systems use tableau-based reasoning procedures (called tableau algorithms) to provide reasoning services, which work by trying to construct a tableau/model [BCM+03].

Figure 2 gives an overview of a general tableau-based reasoning procedure. The rectangles represent some data or knowledge, the arrows show the data flow, and the gray rounded boxes show where data processing is performed. As shown in the figure, some pre-processing steps are first applied to obtain the initial extended ABox $\mathcal{A}_0^{\mathcal{E}}$. Then, a set of completion rules are applied.

In this section, we first describe the reasoning services that are commonly provided by standard DL frameworks. We then present the tableau algorithm for the description logic $\mathcal{ALC}$.

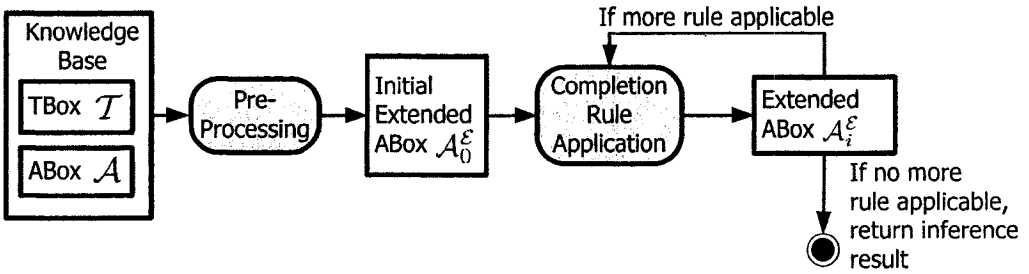

Figure 2: Tableau-based reasoning procedure for standard DL

Reasoning Services

The following are some main reasoning services supported by standard DL frameworks:

- **Consistency Problem:** The ABox $\mathcal{A}$ is consistent with respect to the TBox $\mathcal{T}$ if there is an interpretation $\mathcal{I}$ that is a model of both $\mathcal{A}$ and $\mathcal{T}$. The knowledge base is consistent if and only if its ABox is consistent with respect to its TBox [Mol01].

- **Entailment Problem:** An assertion X is entailed by a knowledge base Σ, denoted $\Sigma \models X$, if every model of Σ is a model of X. The entailment problem can be reduced to the consistency problem. Let $\langle a : C \rangle$ be an assertion. Then, $\Sigma \models \langle a : C \rangle$ if and only if $\Sigma \cup \{\langle a : \neg C \rangle\}$ is inconsistent [BCM+03].

- **Concept Satisfiability Problem:** A concept C is satisfiable with respect to a TBox $\mathcal{T}$ if there exists a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is not empty. The concept satisfiability problem can be reduced to the consistency problem: C is satisfiable if and only if $\{\langle a : C \rangle\}$ is consistent, where a is a new individual

name [BCM+03].

- **Subsumption Problem:** A concept C is subsumed by a concept D with respect to a TBox $\mathcal{T}$, denoted $\mathcal{T} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for every model $\mathcal{I}$ of $\mathcal{T}$. The subsumption problem can be reduced to concept satisfiability problem: C is subsumed by D if and only if $C \sqcap \neg D$ is unsatisfiable [Sch94, BCM+03]. The concept satisfiability problem can, in turn, be reduced to the consistency problem.

Since the above reasoning services can all be reduced to consistency problem, we focus on the consistency problem in this thesis.

Tableau Algorithm for $\mathcal{ALC}$

The tableau algorithm for checking the consistency of knowledge base works slightly differently, depending on whether or not the TBox is unfoldable, described as follows.

Definition 2.1.2 (Unfoldable TBox) A TBox is *unfoldable* [Baa90] if all of the following conditions hold.

- All the axioms in the TBox are either concept subsumptions or concept definitions. That is, there is no GCIs nor concept equations in the TBox.
- All the axioms in the TBox are unique on the left hand side. In other words, for every atomic concept A, there is at most one axiom of the form $\langle A \sqsubseteq C \rangle$ or $\langle A \equiv C \rangle$ in the TBox.

- All the axioms are acyclic. That is, there is no recursive axiom in which a concept name directly or indiretly uses itself. For example, there is no axiom such as $\langle Person \sqsubseteq \forall hasParent.\ Person \rangle$, nor is there mutually recursive axioms such as $\langle Person \sqsubseteq \forall hasParent.Father \rangle$ and $\langle Father \sqsubseteq \forall hasChild.Person \rangle$. Note that the pair of concept subsumptions $\{\langle A \sqsubseteq C \rangle, \langle C \sqsubseteq A \rangle\}$ is not considered to be cyclic since it is equivalent to the concept definition $\langle A \equiv C \rangle$.

With unfoldable TBox, we can eliminate all the defined names from the right hand side of all axioms [BCM+03]. For example, suppose a TBox consists of these two axioms:

$\langle Parent \equiv Person \sqcap \exists hasChild.Person \rangle$

$\langle Mother \equiv Parent \sqcap Female \rangle$

Then, it can be unfolded as:

$\langle Parent \equiv Person \sqcap \exists hasChild.Person \rangle$

$\langle Mother \equiv Person \sqcap \exists hasChild.Person \sqcap Female \rangle$

Similarly, if we have concept subsumptions, we can first replace an axiom of the form $\langle A \sqsubseteq C \rangle$ with the concept definition $\langle A \equiv C \sqcap A' \rangle$, where A' is a new concept name. Then, we can follow the same procedure described above to unfold the TBox.

Once a TBox is unfolded, we can unfold a concept C with respect to the TBox (see Definition 2.1.3). Since all the concepts in a knowledge base with unfoldable TBox can be unfolded, the problem of reasoning with respect to an unfoldable TBox can

be reduced to the problem of reasoning with respect to an empty TBox [BCM+03], which simplifies the reasoning procedure as we will describe shortly.

Definition 2.1.3 (Unfoldable Concept) Given an unfoldable TBox, a concept C in the knowledge base is unfoldable by recursively substituting each non-atomic concept name in C by its definition in the unfolded TBox until all the sub-concepts in C are atomic concept names.

In the following, we first present a tableau algorithm for $\mathcal{ALC}$ with unfoldable TBox. We then describe what needs to be done when the TBox is not unfoldable.

Tableau Algorithm for $\mathcal{ALC}$ with an Unfoldable TBox

Given a knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is an unfoldable TBox, the tableau algorithm starts with some pre-processing. In this phase, each concept in the ABox $\mathcal{A}$ is unfolded with respect to the TBox $\mathcal{T}$. In addition, each concept description is transformed into its negation normal form (NNF) (see the definition below). The resulting ABox is referred to as the *initial extended ABox*, $\mathcal{A}_0^{\mathcal{E}}$.

Definition 2.1.4 (Negation Normal Form) A concept C is in *negation normal form* (NNF) if the negation operator appears only in front of concept names.

Let C and D be concepts, and R be a role. Any concept description can be converted to NNF by applying the following equivalence rules:

- $\neg\neg(C) \equiv C$

- $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$
- $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$
- $\neg\exists R.C \equiv \forall R.\neg C$
- $\neg\forall R.C \equiv \exists R.\neg C$

For example, NNF of the concept $\neg\exists hasPet.(Dog \sqcup Cat)$ is $\forall hasPet.(\neg Dog \sqcap \neg Cat)$.

Next, the tableau algorithm tries to construct a model by iteratively applying a set of completion rules in arbitrary order. Each completion rule application adds one or more additional inferred assertions to the extended ABox to represent explicitly the knowledge that was previously stored implicitly. The algorithm terminates when no further completion rule is applicable. If one could arrive at a completion that contains no contradiction (also called the clash), then the knowledge base Σ is consistent. Otherwise, it is inconsistent.

The completion rules for $\mathcal{ALC}$ with unfoldable TBox are introduced in Table 1 [BCM+03]. Note that the disjunction rule is nondeterministic since it can be applied in different ways to the same ABox. Furthermore, the role exists restriction rule is the only rule that generates new individuals.

Definition 2.1.5 (Forest, Node Label) The model being constructed by the tableau algorithm can be thought as a *forest* [HST00]. A forest is a collection of trees, with nodes corresponding to individuals, edges corresponding to relationships/roles

Rule Name	Rule
Clash Trigger	$\{\langle a : A\rangle, \langle a : \neg A\rangle\} \subseteq \mathcal{A}_i^{\mathcal{E}}$
Conjunction Rule	if $\langle a : C \sqcap D\rangle \in \mathcal{A}_i^{\mathcal{E}}$ and $\{\langle a : C\rangle\langle a : D\rangle\} \nsubseteq \mathcal{A}_i^{\mathcal{E}}$ then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle a : C\rangle\langle a : D\rangle\}$
Disjunction Rule	if $\langle a : C \sqcup D\rangle \in \mathcal{A}_i^{\mathcal{E}}$ and $\{\langle a : C\rangle\langle a : D\rangle\} \cap \mathcal{A}_i^{\mathcal{E}} = \emptyset$ then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \langle a : C\rangle$, or $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \langle a : D\rangle$
Role Exists Restriction Rule	if $\langle a : \exists R.C\rangle \in \mathcal{A}_i^{\mathcal{E}}$ and $\nexists$ an individual b such that $\{\langle (a, b) : R\rangle\langle b : C\rangle\} \subseteq \mathcal{A}_i^{\mathcal{E}}$ then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle (a, b) : R\rangle\langle b : C\rangle\}$ where b is a new individual
Role Value Restriction Rule	if $\langle a : \forall R.C\rangle \in \mathcal{A}_i^{\mathcal{E}}$ and $\exists$ an individual b such that $\langle (a, b) : R\rangle \in \mathcal{A}_i^{\mathcal{E}}$ and $\langle b : C\rangle \notin \mathcal{A}_i^{\mathcal{E}}$ then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \langle b : C\rangle$

Table 1: Completion rules for $\mathcal{ALC}$ with an unfoldable TBox

between individuals, and root nodes corresponding to individuals present in the initial extended ABox $\mathcal{A}_0^{\mathcal{E}}$. Each node is labeled using the notation $\mathcal{L}(\textit{individual\_name})$ to indicate the concept assertions associated with it. These labels are referred to as the *node labels.*

Example 2.1.6

To illustrate the completion rules, assume the initial extended ABox is:

$$\mathcal{A}_0^{\mathcal{E}} = \{\langle a : D \sqcap \exists R.C\rangle,$$
$$\langle (b, c) : R\rangle,$$
$$\langle (b, d) : S\rangle,$$
$$\langle b : \forall S.E\rangle\}$$

We start by applying the conjunction rule to the first assertion, which yields the new

extended ABox:

$$\mathcal{A}_1^{\mathcal{E}} = \mathcal{A}_0^{\mathcal{E}} \cup \{\langle a : D\rangle,$$

$$\langle a : \exists R.C\rangle\}$$

We can then apply the role value restriction rule to the third and forth assertions in $\mathcal{A}_0^{\mathcal{E}}$, which gives:

$$\mathcal{A}_2^{\mathcal{E}} = \mathcal{A}_1^{\mathcal{E}} \cup \{\langle d : E\rangle\}$$

Finally, the role exists restriction rule is applied to assertion $\langle a : \exists R.C\rangle$. This yields:

$$\mathcal{A}_3^{\mathcal{E}} = \mathcal{A}_2^{\mathcal{E}} \cup \{\langle (a, ind1) : R\rangle,$$

$$\langle ind1 : C\rangle\}$$

Since there are no more rules applicable, the reasoning procedure terminates. The resulting model is shown as the forest in Figure 3.

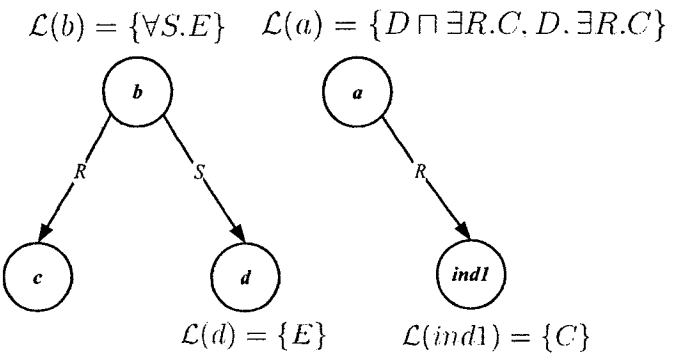

Figure 3: A model for Example 2.1.6

Tableau Algorithm for $\mathcal{ALC}$ with a General TBox

A general TBox is a TBox that may contain GCIs, concept equations, and cycles.

To check the consistency of such a knowledge base, we need to perform the following pre-processing steps before applying the completion rules.

1. Replace each concept equation of the form $\langle C \equiv D \rangle$ with two GCIs: $\langle C \sqsubseteq D \rangle$ and $\langle D \sqsubseteq C \rangle$.

2. Transform every CGI in the TBox $\mathcal{T}$ into normal form. That is, replace each GCI of the form $\langle C \sqsubseteq D \rangle$ with $\langle \top \sqsubseteq \neg C \sqcup D \rangle$.

3. Transform every concept into NNF (refer to Definition 2.1.4).

4. Augment the ABox $\mathcal{A}$ with respect to the TBox $\mathcal{T}$. That is, for each individual a in $\mathcal{A}$ and each axiom $\langle \top \sqsubseteq \neg C \sqcup D \rangle$ in $\mathcal{T}$, add $\langle a : \neg C \sqcup D \rangle$ to $\mathcal{A}$. The reason behind this augmentation is that a GCI $\langle C \sqsubseteq D \rangle$ is satisfied by a model $\mathcal{I}$ iff every individual in the model satisfies $\neg C \sqcup D$ [BDS93]. We call the resulting ABox as the *initial extended ABox* $\mathcal{A}_0^{\mathcal{E}}$.

5. Apply the clash trigger (described in Table 1) to check if the initial knowledge base is inconsistent.

With general TBoxes, generation of new individuals by the Role Exists Restriction rule may cause the completion-rule application to fail to terminate (see Example 2.1.8). To ensure termination, the notion of blocking is introduced below.

Definition 2.1.7 (Blocking) Let a and b be generated individuals in the extended ABox $\mathcal{A}_i^{\mathcal{E}}$. Also, let $\mathcal{L}(a)$ and $\mathcal{L}(b)$ be the node labels for a and b respectively. Then,

individual b is blocked by some ancestor a (or a is the blocking individual for b) if $\mathcal{L}(b) \subseteq \mathcal{L}(a)$.

With the notion of blocking, the modified Role Exists Restriction Rule can then be presented. Table 2 shows the modified rule that checks for the blocking condition before the rule is applied. In addition, each time a new individual is generated, it must be asserted to satisfy all the general axioms in the TBox.

Role Exists Restriction Rule:

if $\langle a : \exists R.C \rangle \in \mathcal{A}_i^{\mathcal{E}}$ and
  a is not blocked and
  $\nexists$ an individual b such that $\{\langle (a,b) : R \rangle \langle b : C \rangle\} \subseteq \mathcal{A}_i^{\mathcal{E}}$
then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle (a,b) : R \rangle \langle b : C \rangle\}$ where b is a new individual
    for each axiom $\langle \top \sqsubseteq \neg C \sqcup D \rangle$ in the TBox $\mathcal{T}$
    {
      $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle b : \neg C \sqcup D \rangle\}$
    }

Table 2: Role Exists Restriction Rule for $\mathcal{ALC}$ with general TBox

Example 2.1.8

To illustrate the need for blocking and the modified Role Exists Restriction Rule, consider the knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where:

$\mathcal{T} = \{\langle Person \sqsubseteq \exists hasParent.Person \rangle\}$

$\mathcal{A} = \{\langle John : Person \rangle\}$

We first perform the pre-processing steps. This is done by first translating the axiom in $\mathcal{T}$ into its normal form:

$\mathcal{T} = \{\langle \top \sqsubseteq (\neg Person \sqcup \exists hasParent.Person) \rangle\}$

Next, augment the ABox with respect to the TBox by adding the following assertion to the ABox:

$\langle John : (\neg Person \sqcup \exists hasParent.Person)\rangle$

Then, we initialize the extended ABox to:

$\mathcal{A}_0^{\mathcal{E}} = \{\langle John : Person\rangle,$

$\langle John : (\neg Person \sqcup \exists hasParent.Person)\rangle\}$

In this case, according to the clash triggers, there is no trivial contradiction in the knowledge base. Once the pre-processing steps are over, we are ready to apply the completion rules.

Since the first assertion, $\langle John : Person\rangle$, contains atomic concept, no rule is applied. For the second assertion, $\langle John : (\neg Person \sqcup \exists hasParent.Person)\rangle$, the Disjunction rule is applied, which yields:

$\mathcal{A}_1^{\mathcal{E}} = \mathcal{A}_0^{\mathcal{E}} \cup \langle John : \exists hasParent.Person\rangle.$

Next, we apply the Role Exists Restriction rule to the new assertion in $\mathcal{A}_1^{\mathcal{E}}$, and obtain:

$\mathcal{A}_2^{\mathcal{E}} = \mathcal{A}_1^{\mathcal{E}} \cup \{\langle (John, ind1) : hasParent\rangle,$

$\langle ind1 : Person\rangle,$

$\langle ind1 : (\neg Person \sqcup \exists hasParent.Person)\rangle\}$

After applying the Disjunction rule to the last assertion in $\mathcal{A}_2^{\mathcal{E}}$, we obtain:

$\mathcal{A}_3^{\mathcal{E}} = \mathcal{A}_2^{\mathcal{E}} \cup \langle ind1 : \exists hasParent.Person\rangle.$

Next, the application of the Role Exists Restriction rule to the new assertion in $\mathcal{A}_3^{\mathcal{E}}$ yields:

$$\mathcal{A}_4^{\mathcal{E}} = \mathcal{A}_3^{\mathcal{E}} \cup \{\langle (ind1, ind2) : hasParent \rangle,$$

$$\langle ind2 : Person \rangle,$$

$$\langle ind2 : (\neg Person \sqcup \exists hasParent.Person) \rangle\}$$

After applying the Disjunction rule to the last assertion, we obtain:

$$\mathcal{A}_5^{\mathcal{E}} = \mathcal{A}_4^{\mathcal{E}} \cup \langle ind2 : \exists hasParent.Person \rangle.$$

Since $ind1$ is an ancestor of $ind2$ and $\mathcal{L}(ind2) \subseteq \mathcal{L}(ind1)$, individual $ind2$ is blocked. Therefore, we will not continue applying the Role Exists Restriction rule, and the tableau algorithm terminates at this point.

Note that without blocking, the tableau algorithm would never terminate since new individual will be generated for each application of the Role Exists Restriction rule.

2.2 Deductive Database with Uncertainty

The parametric framework [LS01b] is a generic language for deductive databases with uncertainty. The authors classified existing frameworks for deductive databases with uncertainty into two approaches:

- Implication-based (IB) approach, where each rule is associated with a certainty value and is represented as:

  $A \xleftarrow{\alpha} B_1, \cdots, B_n$

  meaning that the certainty that $B_1 \wedge \cdots \wedge B_n$ implies A is α. Given the certainty of $B_i s$, the certainty of A is then computed by taking the conjunction of

the certainties of $B_i s$, and then propagating this certainty value from the rule body to its head, using α as an attenuation factor.

- Annotation-based (AB) approach, where each rule is an expression of the form:

  $A : f(\beta_1, \cdots, \beta_n) \leftarrow B_1 : \beta_1, \cdots, B_n : \beta_n$

  That is, each subgoal is associated with a certainty factor, but the implication is as in the standard case. This rule asserts that, given the certainty of B_i is at least β_i, the certainty of A is computed by the n-ary function $f(\beta_1, \cdots, \beta_n)$ associated with the head.

The parametric framework generalizes all the existing IB frameworks. This is done by first assuming that certainty values form a complete lattice $\mathcal{T}$, which allows various forms of certainty values to be modeled. Associated with each rule in the parametric framework is a triplet of parameters:

$A \overset{\alpha}{\leftarrow} B_1, \cdots, B_n \ \langle f^d, f^p, f^c \rangle$

The first parameter, f^d, is known as the disjunction function, used to combine the certainties associated with different derivations of the same ground atom. The second parameter, f^p, is known as the propagation function. It is used to combine the certainty of the rule body with the rule's certainty to derive the certainty of the head. Finally, the third parameter, f^c, is known as the conjunction function. It is used to combine certainties of the atoms in the rule body and returns the overall certainty of the rule body. To ensure that these three functions (also known as the combination functions) are meaningful, the parametric framework enforces that each

of the functions must satisfy some properties. For example, for any conjunction function f^c and any pair of certainty values α and β, $f^c(\alpha, \beta)$ cannot be more than α or β.

By setting different certainty lattice and by considering different combination functions associated with the rule, different types of uncertainty IB formalisms can be simulated by the parametric framework. For example, by setting the certainty lattice to be $\mathcal{T} = \{0, 1\}$, α to 1, f^p and f^c to the *min* function, and f^d to the *max* function, the classical logic programming framework can be simulated. Similarly, by setting the certainty lattice to $\mathcal{T} = [0, 1]$, α to some certainty value in [0, 1], f^p to the algebraic product ($prod(x, y) = x \times y$), f^c to the *min* function, and f^d to the *max* function, it becomes van Emden's framework [Van86].

The parametric framework uses a bottom-up method to evaluate the programs. For example, consider the following program:

$r1 : p(X, Y) \overset{0.9}{\leftarrow} q(X, Z), p(Z, Y)\ \langle max, prod, min \rangle$

$r2 : p(X, Y) \overset{0.7}{\leftarrow} q(X, Y)\ \langle max, min, min \rangle$

$r3 : q(a, b) \overset{0.8}{\leftarrow}\ \langle max, -, - \rangle$

$r4 : q(b, c) \overset{0.6}{\leftarrow}\ \langle max, -, - \rangle$

After deriving the facts $q(a, b) : 0.8$ and $q(b, c) : 0.6$ at iteration one, in the second iteration, the rule $r2$ infers $p(a, b) : 0.7$ and $p(b, c) : 0.6$ since the propagation function associated with $r2$ is *min*. In the third iteration, $r1$ is applied. Since the conjunction function is *min*, the certainty of $q(a, b)$ and $p(b, c)$ is 0.6. Then, the propagation function associated with $r1$, *prod*, is applied, which yields $p(a, c) : 0.54$. Since no

more rule can be fired, the evaluation terminates.

2.3 Description Logics with Uncertainty - A Survey

In this section, we first categorize the existing approaches to incorporate DL with uncertainty. We then describe how each component of the DL framework (the description language, the knowledge base, and the reasoning component) is extended in different approaches.

2.3.1 Approaches for Extending Description Logics with Uncertainty

In the last few years, a number of frameworks have been proposed for extending DLs with uncertainty [Jae94b, Hol94, KLP97, TM98, GL02a, PFT+04, Str05a, SSSP06, QPJ07, SSP+07]. Based on the underlying mathematical foundation and the type of uncertainty modeled, we can classify each of these frameworks into one of the three approaches.

The fuzzy approach is based on fuzzy set theory [Zad65]. It deals with the vagueness in the knowledge, where a proposition is true only to some degree. For example, the statement "Jason is obese with degree 0.4" indicates Jason is slightly obese. Here, the value 0.4 is the degree of membership that Jason is in concept obese.

The probabilistic approach is based on the classical probability theory. It deals

with the uncertainty due to lack of knowledge, where a proposition is either true or false, but one does not know for sure which one is the case. Hence, the certainty value refers to the probability that the proposition is true (or false). For example, one could state that: "The probability that Jason would have heart disease given that he is obese lies in the range $[0.8, 1]$."

Finally, the possibilistic approach is based on possibility theory [Zad78]. It allows both certainty (necessity measure) and possibility (possibility measure) be handled in the same formalism. For example, by knowing that "Jason's weight is above 80 kg", the proposition "Jason's weight is 80 kg" is necessarily true with certainty 1, while "Jason's weight is 90 kg" is possibly true with certainty 0.5.

2.3.2 Description Language with Uncertainty

As in standard DLs, the description languages contain a set of language constructors that serve as the building blocks for the description. However, these description language constructors are extended to take into account the presence of uncertainty. In this section, we study how description languages are extended in fuzzy, probabilistic, and possibilistic approaches.

Fuzzy Description Language

All existing fuzzy DL frameworks keep the syntax of the description languages the same as the standard case. However, the semantics of the description languages are extended by fuzzifying their interpretation using fuzzy logic.

In general, a fuzzy interpretation $\mathcal{I}$ is a pair $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain and $\cdot^{\mathcal{I}}$ is an interpretation function that maps language elements to some membership degree in the unit interval $[0, 1]$. For instance, the semantics of an atomic concepts A is defined as $A^{\mathcal{I}}(a) \in [0,1]$, for all $a \in \Delta^{\mathcal{I}}$. Intuitively, it means that if an individual a is in the domain, then the interpretation of A gives the membership degree with which a belongs to A. The semantics of concept descriptions are defined in a rather straightforward way. For instance, for any fuzzy concepts C and D, the semantics of the concept conjunction is defined as $(C \sqcap D)^{\mathcal{I}}(a) = min(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$, for all $a \in \Delta^{\mathcal{I}}$ [TM98, Str98, Str01, HKS02, ST04, Str04a, SSSP06]. In other words, when we intersect two fuzzy concepts C and D, the resulting certainty degree is computed by taking the minimum of the certainty degrees of C and D.

Among existing fuzzy description languages, the ones reported in [PFT+04, Str05a, Str05b, Str04b] differ from the rest. The first three allow the semantics of the description language constructors to be flexibly defined. For example, instead of fixing the semantics of the concept conjunction to the standard *min* function, these frameworks use a more general function, *t-norm*. Some commonly used t-norm functions are: the standard minimum function ($t(a, b) = min(a, b)$), the algebraic product ($t(a, b) = ab$), and the bounded difference ($t(a, b) = max(0, a + b - 1)$. On the other hand, [Str04b] allows the certainty values to be flexibly defined by assuming that the certainty values lie in a complete lattice. Note that the parametric framework [LS01b] also takes the lattice approach.

In addition to extending the semantics of the description language, a number of

fuzzy-based frameworks also add new language constructs to the description language. These language constructors can be categorized into two types: manipulators and fuzzy quantifiers. The *manipulators* [TM98, HKS02, Str05a, Str05b], also known as *modifiers*, are unary operators such as "mostly", "more or less", and "very" that can modify the membership functions of the concepts they are applied to. The *fuzzy quantifiers* proposed in [ST04] extend the expressiveness of the description language by allowing one to express vague quantities (such as "about 2") or quantity intervals (such as "roughly between 1 and 3").

Probabilistic Description Language

There have been two approaches proposed to extend the description language with probability.

The first approach [DS05, Dur05] keeps the syntax of the description language the same as the standard DL, and extends the corresponding semantics with probability. A probabilistic interpretation $\mathcal{I}$ consists of the domain $\Delta^{\mathcal{I}}$ and an interpretation function, where the interpretation function maps language elements to some probabilistic value in the unit interval $[0, 1]$. The semantics of concept descriptions are defined in a straightforward way. For instance, with the independence assumption, the semantics of concept conjunction is defined as $(C \sqcap D)^{\mathcal{I}}(a) = C^{\mathcal{I}}(a) \cdot D^{\mathcal{I}}(a)$, and the semantics of concept disjunction is defined as $(C \sqcup D)^{\mathcal{I}}(a) = C^{\mathcal{I}}(a) + D^{\mathcal{I}}(a) - (C \sqcap D)^{\mathcal{I}}(a)$, for all $a \in \Delta^{\mathcal{I}}$.

The second approach [Hei94, Jae94a, Jae94b, GL02a, GL02b] keeps both the syntax and semantics of the description language the same as the standard DL. However, a new language constructor, namely the conditional probability constructor, is added to handle uncertainties. The probabilistic interpretation $\mathcal{I}$ in this approach consists of the domain $\Delta^{\mathcal{I}}$ and an interpretation function that maps the language elements to some probabilistic value in the unit interval $[0, 1]$. The new language constructor $(C|D)$ can be thought as the abbreviation for the conditional probability in the classical probability theory as $(C|D) = (C \sqcap D)/D$, and its semantics is defined as $(C|D)^{\mathcal{I}}(a) \in [0, 1]$, for all $a \in \Delta^{\mathcal{I}}$.

Possibilistic Description Language

The possibilistic DL [Hol94, QPJ07] keeps the syntax of the description language the same as the standard DL, while changing their interpretation using possibility theory.

To define the semantics of the possibilistic description language, the most basic notion is the possibility distribution π defined as a mapping from the domain Ω to $[0, 1]$. From the possibility distribution, two measures can be determined. Given an event or formula C, the *possibility measure* is defined as: $\Pi(C) = max\{\pi(\omega) \mid \omega \models C\}$, and it gives the extent to which event C is possible. For example, the possibility of the universal event (or the Top concept) is 1, i.e., $\Pi(\top) = 1$, and the possibility of the null event (or the Bottom concept) is 0, i.e., $\Pi(\bot) = 0$. As another example, the possibility of the occurrence of C or D is the maximum of the possibilities of the two, i.e., $\Pi(C \sqcup D) = max\{\Pi(C), \Pi(D)\}$. The second measure is the *necessity measure*

which is simply defined as $\mathrm{N}(C) = 1 - \Pi(\neg C)$, and it characterizes the extent to which an event is necessary or certain to occur.

2.3.3 Knowledge Base with Uncertainty

A knowledge base Σ in uncertainty frameworks consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, where each axiom and assertion is extended to account for the certainty values.

An interpretation $\mathcal{I}$ *satisfies* (or is a *model* of) a knowledge base Σ (denoted as $\mathcal{I} \models \Sigma$), if and only if it satisfies each element of Σ (i.e., both the $\mathcal{T}$ and $\mathcal{A}$). In addition, the knowledge base Σ is said to be *consistent* if there exists an interpretation $\mathcal{I}$ that satisfies Σ; otherwise, Σ is said to be *inconsistent*.

TBox with Uncertainty

Fuzzy TBox There are two approaches to incorporate fuzzy notions into the TBox.

The first approach keeps the syntax to be the same as the standard axioms while extending only the semantics using fuzzy logic [Str01, HKS02, ST04, PFT+04, Str04a, Str04b, Str05a]. A fuzzy interpretation $\mathcal{I}$ satisfies a fuzzy concept inclusion $\langle C \sqsubseteq D \rangle$ if for all $a \in \Delta^{\mathcal{I}}$, $C^{\mathcal{I}}(a) \leq D^{\mathcal{I}}(a)$. That is, the certainty value of a sub-concept C (such as *veryTall*) is no more than the certainty value of the super-concept D (such as *Tall*). Similarly, an interpretation $\mathcal{I}$ satisfies a fuzzy concept definition $\langle C \equiv D \rangle$ if for all $a \in \Delta^{\mathcal{I}}$, $C^{\mathcal{I}}(a) = D^{\mathcal{I}}(a)$.

The second approach extends the TBox both syntactically and semantically. The syntax is extended by associating with each axiom a certainty value and optionally an

operator. For example, given two concepts C and D, and a certainty value $\alpha \in [0, 1]$, a fuzzy concept inclusion can be expressed as $\langle C \sqsubseteq D \geq \alpha \rangle$, indicating that the certainty that C is subsumed by D is at least α [Str98, Str05b]. In terms of the semantics, the expression $C \sqsubseteq D$ can be viewed as the first order formula: $\forall a.C(a) \rightarrow D(a)$, where $C(a) \rightarrow D(a)$ can be viewed as $(\neg C \vee D)(a)$ and $\forall$ can be viewed as a conjunction over all the elements of the domain. Therefore, an interpretation $\mathcal{I}$ satisfies a fuzzy concept inclusion $\langle C \sqsubseteq D \geq \alpha \rangle$ if $min_{a \in \Delta^{\mathcal{I}}}\{(\neg C \sqcup D)^{\mathcal{I}}(a)\} \geq \alpha$.

Probabilistic TBox The probabilistic TBox contains a set of the standard terminological axioms and a set of probabilistic terminological axioms. There are two main approaches to represent probabilistic terminological axioms. The first approach is to embed probabilistic information as part of the terminological axiom, while the second approach stores the probability information using Bayesian networks.

In the first approach, a probabilistic terminological axiom is expressed as $P(C|D) \in [l, u]$, where C and D are concept descriptions, and $0 \leq l \leq u \leq 1$ [Hei94, Jae94a, Jae94b, GL02a, GL02b]. This axiom states that, if an individual is known to belong to concept D, then the probability that this individual belongs to concept C lies in the interval $[l, u]$. Note that, although some frameworks such as [Jae94b, Jae94a] express probabilistic terminological axiom using exact probability, $P(C|D) = u$, it is simply a special case of the interval probability, since $P(C|D) = u$ can be expressed as $P(C|D) \in [u, u]$.

The second approach uses Bayesian networks to express probabilistic information

[KLP97, Yel99]. For example, the probabilistic component defined in [KLP97] consists of a set of probabilistic classes (p-classes), where each p-class represents the probabilistic information related to a certain class of individuals. Each p-class is represented using a Bayesian network N_P [Pea88]. Also, each node in N_P is associated with a conditional probability table that defines the probability of each possible value of the node, given each combination of values for the node's parents.

Possibilistic TBox The possibilistic frameworks extend the standard TBox both syntactically and semantically [Hol94, QPJ07]. To be more specific, let C and D be concepts, and $\alpha \in [0, 1]$ be a certainty value. A possibilistic axiom is expressed as $\langle C \sqsubseteq D, \alpha \rangle$. If α represents the possibility degree, the axiom means "$C \sqsubseteq D$ is possibly true at least with degree α." On the other hand, if α represents the necessity degree, the axiom asserts "$C \sqsubseteq D$ is necessarily true at least with degree α."

ABox with Uncertainty

Fuzzy ABox The fuzzy frameworks extend the standard assertions both syntactically and semantically [TM98, ST04, PFT+04, HKS02, Str01, Str04a, Str04b, Str05a, Str05b, Str98]. In general, a fuzzy assertion can be represented in the form $\langle X \text{ op } \alpha \rangle$, where X is either a concept assertion $a : C$ or a role assertion $(a, b) : R$, op is a comparison operator in $\{\geq, \leq, >, <, =\}$, and α is a certainty value. For example, the fuzzy concept assertion $\langle a : C \geq 0.5 \rangle$ means the certainty that individual a belongs to concept C is at least 0.5.

Probabilistic ABox There are a couple of approaches to extend ABox with probabilities. In the first approach [Jae94a, Jae94b, DS05], a probabilistic concept assertion is an expression of the form $P(a : C) = \alpha$, where a is an individual, C is a concept, and $\alpha \in [0, 1]$. Intuitively, this asserts that "the probability that an individual a belongs to concept C is α." Similarly, a probabilistic role assertion is an expression of the form $P((a, b) : R) = \alpha$, where a and b are individuals, R is a role, $\alpha \in [0, 1]$, and it asserts that "the probability that an individual a is related to individual b through role R is α."

In the second approach [GL02a, GL02b], the set of individuals is partitioned into classical individuals and probabilistic individuals. The probabilistic individuals are defined as individuals for which some probabilistic knowledge is explicitly stored. Let a and b be individuals, C be a concept, and R be a role. An ABox consists of a set of standard assertions, and a set of probabilistic assertions. The probabilistic concept assertions are expressed as $(C|\{a\})[l, u]$, where $l, u \in [0, 1]$. It asserts that "the probability that individual a is in concept C lies in the range $[l, u]$." Similarly, the probabilistic role assertions are expressed as $(\exists R.\{b\}|\{a\})[l, u]$, where $l, u \in [0, 1]$. It asserts that "the probability that individual a is related to individual b through role R lies in the range $[l, u]$."

Possibilistic ABox The possibilistic assertion extends the standard assertion by associating with it a certainty value [Hol94, QPJ07]. Specifically, let a and b be individuals, C be a concept, R be a role, and $\alpha \in [0, 1]$ be some certainty value.

A possibilistic concept assertion is expressed as $\langle a : C, \alpha \rangle$, and a possibilistic role assertion is expressed as $\langle (a, b) : R, \alpha \rangle$. For example, if α represents the possibility degree, then the concept assertion above means "individual a is possibly in concept C with at least degree α."

2.3.4 Reasoning with Uncertainty

The reasoning component in uncertainty frameworks provide inference services that enable implicit knowledge to become explicit, while taking into account the presence of the certainty values associated with the axioms and assertions. This section presents how the reasoning component is extended using fuzzy, probabilistic, and possibilistic approaches.

Fuzzy Reasoning

There are two main approaches for extending the reasoning component in fuzzy frameworks.

Approach 1: Transform Fuzzy DL into Standard DL In this approach, the fuzzy knowledge bases are transformed into standard knowledge bases [Str04a], which are then fed into a standard DL system to perform the actual inference.

As a simple example, consider a fuzzy knowledge base with empty TBox and the ABox: $\mathcal{A} = \{\langle a : A \geq 0.4 \rangle, \langle a : A \leq 0.7 \rangle, \langle a : B \leq 0.2 \rangle, \langle b : B \leq 0.1 \rangle\}$. To transform this fuzzy knowledge base into a standard one, new concepts are first introduced:

$A_{\geq 0.4}$, $A_{\leq 0.7}$, $B_{\leq 0.2}$, and $B_{\leq 0.1}$. Then, axioms that describe the relationship between the newly introduced concepts such as $B_{\leq 0.2} \sqsubseteq B_{\leq 0.1}$ are created. Finally, the fuzzy assertions are mapped to standard ones as $\langle a{:}A_{\geq 0.4}\rangle$, $\langle a{:}A_{\leq 0.7}\rangle$, $\langle a{:}B_{\leq 0.2}\rangle$, $\langle b{:}B_{\leq 0.1}\rangle$, $\langle b{:}B_{\leq 0.2}\rangle$.

The advantage of this approach is that one can directly use the existing reasoners developed for standard DL. Although this may save time from reinventing the wheel, the existing reasoners do not consider the certainty values in fuzzy DL as something special. Furthermore, no extra optimization techniques can be applied to inferences that involve the use of certainty values. Another problem with this approach is that, as shown in the above example, many new concepts and axioms need to be generated during the reasoning process even for small knowledge bases. This problem makes this approach less practical for real-world applications, where the knowledge base size is usually large.

Approach 2: Tailored Reasoning Procedure for Fuzzy DL Most existing fuzzy frameworks introduce new reasoning procedures that are tailored for fuzzy DL. The advantage of this approach is that the certainty values are being treated as the first class citizen. However, the drawback is that the reasoner must be built from scratch.

The inference problems that have been studied in this approach include: consistency problem, entailment problem, and subsumption problem. In what follows, we briefly look at each of these inference problems.

Consistency Problem Let Σ be a fuzzy knowledge base and $\mathcal{I}$ be a fuzzy interpretation. The consistency problem amounts to checking whether Σ has any model. To solve this problem, the tableau algorithms developed for standard DLs are extended so that they consider certainty values in fuzzy DL. There are two different approaches to extend the tableau algorithm.

The first approach deals with the certainty values within the tableau algorithm [HKS02, Str01, Str04b, Str98]. For example, suppose the fuzzy assertion $\langle a : C \sqcap D \geq 0.5\rangle$ is in the ABox, then the conjunction rule would derive assertions $\langle a : C \geq 0.5\rangle$ and $\langle a : D \geq 0.5\rangle$. The completion rules are applied until either all branches in the extended ABox contain a clash (indicating that no model can be build), or there exists a clash-free completion of ABox (meaning that a model can be built).

The second approach relies on bounded Mixed Integer Programming (bMIP) to deal with certainty values [Str05a]. In this approach, a set of completion rules is applied to the ABox to infer new assertions together with a set of inequations over [0,1]-valued variables. For example, if the ABox contains the fuzzy assertion $\langle a : C \sqcup D, \alpha\rangle$, then the disjunction rule would infer the assertions $\langle a : C, x_1\rangle$ and $\langle a : D, x_2\rangle$, and the constraints $x_1 + x_2 = \alpha, x_1 \leq y, x_2 \leq 1 - y, x_1 \in [0, 1]$, and $y \in \{0, 1\}$, where x_1 and x_2 are two new variables, and y is a new control variable. The completion rules are then applied until either the extended ABox contains a clash, or no rule can be further applied. If there is a clash, then the ABox is immediately inconsistent. Otherwise, the bMIP technique is applied to the extended ABox to solve the system of inequations generated to determine consistency of the knowledge base.

Entailment Problem There are two types of entailment problems investigated in fuzzy frameworks. The first one is similar to the standard entailment problem which, given a fuzzy assertion X and a fuzzy interpretation $\mathcal{I}$, determines whether $\mathcal{I} \models X$, for all models $\mathcal{I}$ of the knowledge base Σ. It is shown in [Str01] that such fuzzy entailment problem can be reduced to the consistency problem. That is, $\Sigma \models \langle Y \geq \alpha \rangle$ if $\Sigma \cup \{\langle Y < \alpha \rangle\}$ is inconsistent. Also, $\Sigma \models \langle Y \leq \alpha \rangle$ if $\Sigma \cup \{\langle Y > \alpha \rangle\}$ is inconsistent.

The second entailment problem is also known as the best degree bound (BDB) problem [Str04a, Str05b, SB07]. Given a fuzzy knowledge base Σ, the BDB problem determines the degree to which an assertion is entailed. Specifically, let $\langle a : C \geq x \rangle$ be a fuzzy assertion. The BDB problem can be reduced to the consistency problem by finding the degree of x such that $\Sigma \cup \{\langle a : \neg C \geq 1 - x \rangle\}$ is consistent.

Subsumption Problem Let Σ be a fuzzy knowledge base, and C and D be concepts. The fuzzy subsumption problem determines the degree to which C is subsumed by D [Str01, Str05a, SB07]. This problem can be reduced to the consistency problem by finding the value of x such that $\Sigma \cup \{\langle a : C \sqcap \neg D \geq 1 - x \rangle\}$ is consistent, where a is a new individual name.

Probabilistic Reasoning

The probabilistic reasoning procedure depends heavily on how the probabilistic information is represented in the knowledge base. If Bayesian networks are used to express the probabilistic information, the inference algorithm developed for Bayesian

networks can be directly applied as shown in [KLP97, DP04, DPP04]. On the other hand, if the probabilistic information is embedded in the knowledge base, inference procedures need to be developed. However, unlike in the fuzzy case, none of the existing probabilistic frameworks extended tableau-based reasoning procedures from standard DLs. Instead, these frameworks rely on existing algorithms developed for probabilistic reasoning. For example, the consistency checking algorithm proposed in [GL02a, Luk07] is based on the default reasoning algorithm from conditional knowledge bases [GP91].

Possibilistic Reasoning

Like in the probability case, none of the existing possibilistic frameworks directly extended the tableau-based reasoning procedures from standard DLs. In [QPJ07], it is shown that the inconsistency degree of a possibilistic DL knowledge base can be computed by using binary search. On the other hand, [Hol94] shows that the possibilistic entailment problem is reducible to the standard entailment problem.

2.4 Summary and Concluding Remarks

In this chapter, we presented relevant background knowledge and related works.

By studying the architecture of the $\mathcal{ALC}$ framework, we learned that the $\mathcal{ALC}_U$ framework to be presented in the following chapter also needs to have three components, where the description language provides a set of language constructors so as to represent concepts and roles, the knowledge base consists of a set of axioms and

assertions, and the reasoning procedure explicates knowledge that is stored implicitly in a knowledge base. In addition, we learned that attention needs to be paid when we design a tableau algorithm for general TBoxes, since blocking is needed to ensure termination of the completion-rule application.

We also reviewed the parametric framework for deductive database with uncertainty, which inspired our work on the $\mathcal{ALC}_U$ framework. Specifically, we learned that, by assuming certainty values form a certainty lattice, various forms of certainty values can be modeled. In addition, each rule and fact can be associated with some parameters, where each parameter is a combination function used to combine certainty values in the rule/assertion. By tuning these parameters, different uncertainty frameworks for deductive databases can then be simulated.

Finally, we surveyed the existing frameworks for DLs with uncertainty. We observed that, despite the fact that different frameworks may support different notions of uncertainty, they follow roughly the same pattern. That is, all the three components of the DLs framework, namely the description language, the knowledge base, and the reasoning procedure, need to be extended in order to represent and reason with uncertainty knowledge.

Chapter 3

A Framework for the Description Logic $\mathcal{ALC}_U$

This chapter presents a formal framework for the DL $\mathcal{ALC}_U$. This framework extends each component of the $\mathcal{ALC}$ framework while abstracting away the notion of uncertainty in the extension. As shown in Figure 4, the $\mathcal{ALC}_U$ framework consists of the following three components:

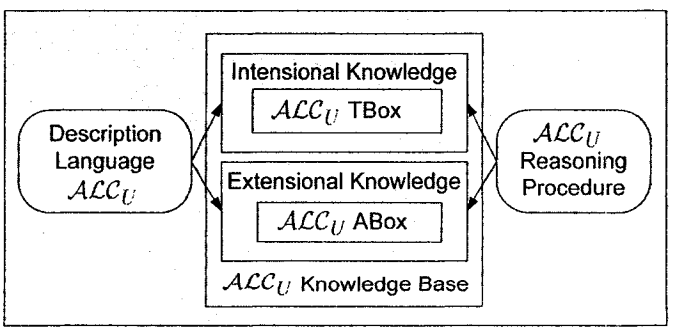

Figure 4: The $\mathcal{ALC}_U$ framework

1. *Description Language $\mathcal{ALC}_U$:* The description language $\mathcal{ALC}_U$ extends the semantics of the description language $\mathcal{ALC}$ to allow different notions of uncertainty be expressed.

2. *$\mathcal{ALC}_U$ Knowledge Base:* The $\mathcal{ALC}_U$ knowledge base extends both the syntax and the semantics of the $\mathcal{ALC}$ knowledge base to allow the modeling of uncertainty in axioms and assertions.

3. *$\mathcal{ALC}_U$ Reasoning Procedure:* The $\mathcal{ALC}_U$ framework is equipped with the reasoning procedure that takes into account the presence of uncertainties associated with axioms and assertions in the $\mathcal{ALC}_U$ knowledge base.

In the rest of this chapter, we first describe each of the above three components in Sections 3.1, 3.2, and 3.3 respectively. Then, we illustrate through an example the various extended components of the $\mathcal{ALC}_U$ framework in Section 3.4. Finally, we compare the $\mathcal{ALC}_U$ framework with related works in Section 3.5.

3.1 Description Language $\mathcal{ALC}_U$

Recall that the description language refers to the language used for building concepts. The syntax of the $\mathcal{ALC}_U$ description language is identical to that of the standard $\mathcal{ALC}$, while the corresponding semantics is extended to support different notions of uncertainty. In this section, we first describe how certainty values can be represented in a generic way. We then describe how the semantics of the description language $\mathcal{ALC}_U$ is defined.

3.1.1 Representation of Certainty Values

To represent different forms of uncertainty, we assume that certainty values form a complete lattice shown as $\mathcal{L} = \langle \mathcal{V}, \preceq \rangle$, where $\mathcal{V}$ is the certainty domain, and $\preceq$ is the partial order on $\mathcal{V}$. Other comparison operators $\prec$, $\succeq$, $\succ$, and $=$ are used with their obvious meanings. We use l to denote the bottom or the least element in $\mathcal{V}$, and use t to denote the top or the greatest element in $\mathcal{V}$. The least upper bound (join) operator in $\mathcal{L}$ is denoted by $\oplus$, its greatest lower bound (meet) operator is denoted by $\otimes$, and its negation operator is denoted by $\sim$. We assume that there is only one underlying certainty lattice for the entire knowledge base.

The advantage of using a lattice is that it can be used to model both qualitative and quantitative certainty values. An example for the former is the classical logic whose certainty values form a certainty lattice $\mathcal{L} = \langle \{0, 1\}, \leq \rangle$, where $\leq$ is the usual order on binary values $\{0, 1\}$. For the latter, an example would be a family of multi-valued logics such as fuzzy logic whose certainty values form a certainty lattice $\mathcal{L} = \langle [0, 1], \preceq \rangle$, where $\preceq$ is the usual order on real numbers.

3.1.2 Semantics of the Description Language $\mathcal{ALC}_U$

The $\mathcal{ALC}_U$ framework treats each type of uncertainty formalism as a special case. Hence, it would be restrictive to consider any specific function to describe the semantics of the description language constructors (for example, fixing the *min* function as the semantics of concept conjunction). Instead, we allow the user to flexibly define combination functions to define the semantics of the description language as long as

these functions satisfy some pre-defined properties which ensure the admissibility of the description language semantics.

Specifically, the semantics of the description language is based on the notion of an interpretation. An interpretation $\mathcal{I}$ is defined as a pair $(\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is the domain and $\cdot^{\mathcal{I}}$ is an interpretation function that maps each

- atomic concept A into a certainty function CF_C, where $CF_C : \Delta^{\mathcal{I}} \rightarrow \mathcal{V}$
- atomic role R into a certainty function CF_R, where $CF_R : \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \rightarrow \mathcal{V}$
- individual name a to an element $a \in \Delta^{\mathcal{I}}$

where $\mathcal{V}$ is the certainty domain. For example, let individual *John* be an element in the domain and let *Obese* be an atomic concept. Then, for the fuzzy approach, $Obese^{\mathcal{I}}(John^{\mathcal{I}})$ gives the degree (membership) that *John* belongs to the concept *Obese*, and for the probabilistic approach, $Obese^{\mathcal{I}}(John^{\mathcal{I}})$ gives the probability that *John* is *Obese*.

We now define the semantics of the $\mathcal{ALC}_U$ description language constructors, including the top concept, the bottom concept, concept negation, concept conjunction, concept disjunction, role exists restriction, and role value restriction.

Top Concept

The interpretation of the top concept $\top$ is the greatest element in the certainty domain $\mathcal{V}$, that is, $\top^{\mathcal{I}}(a) = t$, for all $a \in \Delta^{\mathcal{I}}$. For instance, the interpretation of $\top$ is 1 (or true) in the standard logic with $\mathcal{V} = \{0, 1\}$, as well as in other logics with $\mathcal{V} = [0, 1]$.

Note that in $\mathcal{ALC}$, it is not necessary to introduce the top concept $\top$ in the description language, since it can be represented using other description language constructors, namely $A \sqcup \neg A$. However, when uncertainty is present, $\top$ is no longer $A \sqcup \neg A$. Assume that an individual a belongs to a fuzzy concept A with a membership degree of 0.6. Then, a would belong to $\neg A$ with a membership degree of 0.4. Also, the certainty that a is in $A \sqcup \neg A$ would be the maximum of the two certainties, which is 0.6. This is not what one would expect, since the certainty that a is in $\top$ should be 1 in the fuzzy case. Therefore, it is necessary to explicitly introduce the Top ($\top$) concept in $\mathcal{ALC}_U$.

Bottom Concept

The interpretation of the bottom concept $\bot$ is the least element in the certainty domain $\mathcal{V}$, that is, $\bot^{\mathcal{I}}(a) = l$, for all $a \in \Delta^{\mathcal{I}}$. For example, this corresponds to 0 (or false) in the standard logic with $\mathcal{V} = \{0, 1\}$, as well as in other logics with $\mathcal{V} = [0, 1]$. Note that it is necessary to introduce the Bottom ($\bot$) concept in $\mathcal{ALC}_U$ for the same reason as the one for the top concept.

Concept Negation

Given a concept C, the interpretation of the concept negation $\neg C$ is defined by the negation function $\sim$, where $\sim$: $\mathcal{V} \rightarrow \mathcal{V}$ must satisfy the following properties:

- Boundary Conditions: $\sim l = t$ and $\sim t = l$.
- Double Negation: $\sim(\sim\alpha) = \alpha$, for all $\alpha \in \mathcal{V}$.

The negation operator $\sim$ in the certainty lattice is used as the default negation function. That is, $(\neg C)^{\mathcal{I}}(a) = \sim C^{\mathcal{I}}(a)$, for all $a \in \Delta^{\mathcal{I}}$. A common interpretation of $\neg C$ is $1 - C^{\mathcal{I}}(a)$. For example, if the certainty domain is $\mathcal{V} = [0, 1]$, and if the certainty that individual John is Obese is 0.8. Then, the certainty that John is not Obese is $1 - 0.8 = 0.2$.

Before introducing the semantics of the other description language constructors, let us define the combination functions, which are used to specify how one should interpret a given description language constructor. We also define two types of combination functions called the conjunction function and the disjunction function.

Definition 3.1.1 (Combination Function) A combination function f is a binary function from $\mathcal{V} \times \mathcal{V}$ to $\mathcal{V}$. This function combines a pair of certainty values into one. A combination function must satisfy some properties, as listed in Table 3 [LS01b].

ID	Property Name	Property Definition
P_1	Monotonicity	$f(\alpha_1, \alpha_2) \preceq f(\beta_1, \beta_2)$, if $\alpha_i \preceq \beta_i$, for $i = 1, 2$
P_2	Bounded Above	$f(\alpha_1, \alpha_2) \preceq \alpha_i$, for $i = 1, 2$
P_3	Bounded Below	$f(\alpha_1, \alpha_2) \succeq \alpha_i$, for $i = 1, 2$
P_4	Boundary Condition (Above)	$\forall \alpha \in \mathcal{V}$, $f(\alpha, l) = \alpha$ and $f(\alpha, t) = t$
P_5	Boundary Condition (Below)	$\forall \alpha \in \mathcal{V}$, $f(\alpha, t) = \alpha$ and $f(\alpha, l) = l$
P_6	Continuity	f is continuous w.r.t. each one of its arguments
P_7	Commutativity	$\forall \alpha, \beta \in \mathcal{V}$, $f(\alpha, \beta) = f(\beta, \alpha)$
P_8	Associativity	$\forall \alpha, \beta, \delta \in \mathcal{V}$, $f(\alpha, f(\beta, \delta)) = f(f(\alpha, \beta), \delta)$

Table 3: Combination function properties

Definition 3.1.2 (Conjunction Function) A *conjunction function* f_c is a combination function that should satisfy properties P_1, P_2, P_5, P_6, P_7, and P_8 described in Table 3. The monotonicity property asserts that increasing the certainties of the arguments in f improves the certainty that f returns. The bounded value and boundary condition properties are included so that the interpretation of the certainty values makes sense. The commutativity property allows reordering of the arguments of f, say for optimization purposes. Finally, the associativity of f ensures that different evaluation orders of concept conjunctions will not yield different results. Some common conjunction functions are the well-known minimum function, the algebraic product ($prod(x, y) = xy$) and the bounded difference ($bDiff(x, y) = max(0, x + y - 1)$).

Definition 3.1.3 (Disjunction Function) A *disjunction function* f_d is a combination function that should satisfy properties P_1, P_3, P_4, P_6, P_7, and P_8 described in Table 3. These properties are enforced for similar reasons as the conjunction case. Some common disjunction functions are the maximum function, the probability independent function ($ind(x, y) = x + y - xy$) and the bounded sum function ($bSum(x, y) = min(1, x + y)$).

In what follows, we define the semantics of the remaining $\mathcal{ALC}_U$ description language constructors, including concept conjunction, concept disjunction, role exists restriction, and role value restriction.

Concept Conjunction

Given concepts C and D, the interpretation of the concept conjunction $C \sqcap D$ is defined by the conjunction function f_c (refer to Definition 3.1.2) as:

$(C \sqcap D)^{\mathcal{I}}(a) = f_c(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$, for all $a \in \Delta^{\mathcal{I}}$

For example, if the conjunction function is min, the certainty that an individual Mary is Tall is 0.8, and the certainty that Mary is Thin is 0.6, then the certainty that Mary is both Tall and Thin is $min(0.8, 0.6) = 0.6$.

Concept Disjunction

Given concepts C and D, the interpretation of concept disjunction $C \sqcup D$ is defined by the disjunction function f_d (refer to Definition 3.1.3) as:

$(C \sqcup D)^{\mathcal{I}}(a) = f_d(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$, for all $a \in \Delta^{\mathcal{I}}$

For example, if the disjunction function is max, the certainty that an individual Mary is Tall is 0.8, and the certainty that Mary is Thin is 0.6, then the certainty that Mary is Tall or Thin is $max(0.8, 0.6) = 0.8$.

Role Exists Restriction

Given a role R and a concept C (also known as the role filler), the interpretation of the "role exists" restriction $\exists R.C$ is defined as:

$(\exists R.C)^{\mathcal{I}}(a) = \oplus_{b \in \Delta^{\mathcal{I}}}\{f_c(R^{\mathcal{I}}(a, b), C^{\mathcal{I}}(b))\}$, for all $a \in \Delta^{\mathcal{I}}$

where $\oplus$ is the join operator in the certainty lattice. The intuition here is that $\exists R.C$

is viewed as the open first order formula $\exists b.\ R(a,b) \wedge C(b)$, where $\exists$ is viewed as a disjunction over certainty values associated with $R(a,b) \wedge C(b)$. Specifically, the semantics of $R(a,b) \wedge C(b)$ is captured using the conjunction function as $f_c(R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))$, and $\exists b$ is captured using $\oplus_{b \in \Delta^{\mathcal{I}}}$.

Role Value Restriction

Given a role R and a concept C, the interpretation of the "role value" restriction $\forall R.C$ is defined as:

$$(\forall R.C)^{\mathcal{I}}(a) = \otimes_{b \in \Delta^{\mathcal{I}}}\{f_d(\sim R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\}, \text{ for all } a \in \Delta^{\mathcal{I}}$$

where $\otimes$ is the meet operator in the certainty lattice. The intuition is that $\forall R.C$ is viewed as the open first order formula $\forall b.\ R(a,b) \rightarrow C(b)$, where $R(a,b) \rightarrow C(b)$ is logically equivalent to $\neg R(a,b) \vee C(b)$, and $\forall$ is viewed as a conjunction over certainty values associated with the implication $R(a,b) \rightarrow C(b)$. To be more precise, the semantics of $R(a,b) \rightarrow C(b)$ is captured using the disjunction and negation functions as $f_d(\sim R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))$, and $\forall b$ is captured using $\otimes_{b \in \Delta^{\mathcal{I}}}$.

Additional Inter-Constructor Properties

In order to convert a concept description into its *negation normal form* (see Definition 2.1.4), we further assume that the following inter-constructor properties hold:

- De Morgan's Rule: $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$ and $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$.
- Negating Quantifiers Rule: $\neg\exists R.C \equiv \forall R.\neg C$ and $\neg\forall R.C \equiv \exists R.\neg C$.

Table 4 summarizes the syntax and the semantics of the description language $\mathcal{ALC}_U$.

Name	Syntax	Semantics ($a \in \Delta^{\mathcal{I}}$)
Top Concept	$\top$	$\top^{\mathcal{I}}(a) = t$
Bottom Concept	$\bot$	$\bot^{\mathcal{I}}(a) = l$
Concept Negation	$\neg C$	$(\neg C)^{\mathcal{I}}(a) = \sim C^{\mathcal{I}}(a)$
Concept Conjunction	$C \sqcap D$	$(C \sqcap D)^{\mathcal{I}}(a) = f_c(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$
Concept Disjunction	$C \sqcup D$	$(C \sqcup D)^{\mathcal{I}}(a) = f_d(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$
Role Exists Restriction	$\exists R.C$	$(\exists R.C)^{\mathcal{I}}(a) = \oplus_{b \in \Delta^{\mathcal{I}}}\{f_c(R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\}$
Role Value Restriction	$\forall R.C$	$(\forall R.C)^{\mathcal{I}}(a) = \otimes_{b \in \Delta^{\mathcal{I}}}\{f_d(\sim R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\}$

Table 4: Syntax and semantics of the description language $\mathcal{ALC}_U$

3.2 $\mathcal{ALC}_U$ Knowledge Base

An $\mathcal{ALC}_U$ *knowledge base* Σ is a pair $\langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is a TBox and $\mathcal{A}$ is an ABox. An interpretation $\mathcal{I}$ *satisfies* (or is a *model* of) Σ (denoted $\mathcal{I} \models \Sigma$), if and only if it satisfies both $\mathcal{T}$ and $\mathcal{A}$. The knowledge base Σ is *consistent* if there exists an interpretation $\mathcal{I}$ that satisfies Σ, and is *inconsistent* otherwise. Finally, the ABox $\mathcal{A}$ is *consistent with respect to a TBox* $\mathcal{T}$ if there is an interpretation that is a model of both $\mathcal{A}$ and $\mathcal{T}$.

In what follows, we describe how the TBox and the ABox are extended syntactically and semantically to model uncertainty in axioms and assertions.

3.2.1 $\mathcal{ALC}_U$ TBox

An $\mathcal{ALC}_U$ TBox $\mathcal{T}$ consists of a set of terminological axioms defining how concepts are related to each other. Each axiom is associated with a certainty value as well

as a conjunction function and a disjunction function which are used to interpret the concept descriptions in the axiom. Specifically, an $\mathcal{ALC}_U$ TBox consists of axioms that could include:

- *Concept subsumptions* of the form $\langle A \sqsubseteq C \mid \alpha, f_c, f_d \rangle$
- *Concept definitions* of the form $\langle A \equiv C \mid \alpha, f_c, f_d \rangle$, which is equivalent to $\langle A \sqsubseteq C \mid \alpha, f_c, f_d \rangle$ and $\langle C \sqsubseteq A \mid \alpha, f_c, f_d \rangle$
- *General concept inclusions* (*GCIs*) of the form $\langle C \sqsubseteq D \mid \alpha, f_c, f_d \rangle$
- *Concept equations* of the form $\langle C \equiv D \mid \alpha, f_c, f_d \rangle$, which is equivalent to $\langle C \sqsubseteq D \mid \alpha, f_c, f_d \rangle$ and $\langle D \sqsubseteq C \mid \alpha, f_c, f_d \rangle$

where A is an atomic concept, C and D are concept descriptions, $\alpha \in \mathcal{V}$ is the certainty that the axiom holds, f_c is the conjunction function used as the semantics of concept conjunction and part of the role exists restriction, and f_d is the disjunction function used as the semantics of concept disjunction and part of the role value restriction. In case the choice of the combination function in the current axiom is immaterial, "–" is used as a place holder. For example, the axiom

$\langle \mathit{Rich} \sqsubseteq ((\exists \mathit{owns}.\mathit{ExpensiveCar} \sqcup \exists \mathit{owns}.\mathit{Airplane}) \sqcap \mathit{Golfer}) \mid [0.8, 1], \mathit{min}, \mathit{max} \rangle$

states that the concept *Rich* is subsumed by owning expensive car or owning an airplane, and being a golfer. The certainty of this axiom is at least 0.8, with all the concept conjunctions interpreted using *min* function, and all the concept disjunctions interpreted using *max*.

Note also that a GCI is simply a general form of the concept subsumption, and a concept equation is a general form of the concept definition.

In order to transform an axiom of the form $\langle C \sqsubseteq D \mid \alpha, f_c, f_d \rangle$ into its normal form, $\langle \top \sqsubseteq \neg C \sqcup D \mid \alpha, f_c, f_d \rangle$, the semantics of the concept subsumption is restricted to be $f_d(\sim C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$, for all $a \in \Delta^{\mathcal{I}}$, where $\sim C^{\mathcal{I}}(a)$ captures the semantics of $\neg C$, and f_d captures the semantics of $\sqcup$ in $\neg C \sqcup D$. Hence, an interpretation $\mathcal{I}$ satisfies $\langle C \sqsubseteq D \mid \alpha, f_c, f_d \rangle$ if $f_d(\sim C^{\mathcal{I}}(a), D^{\mathcal{I}}(a)) = \alpha$, for all $a \in \Delta^{\mathcal{I}}$.

3.2.2 $\mathcal{ALC}_U$ ABox

An $\mathcal{ALC}_U$ ABox $\mathcal{A}$ consists of a set of assertions, each of which is associated with a certainty value and a pair of combination functions used to interpret the concept description(s) in the assertion. Specifically, an $\mathcal{ALC}_U$ ABox consists of a set of assertions that could include:

- *Concept assertions* of the form $\langle a : C \mid \alpha, f_c, f_d \rangle$
- *Role assertions* of the form $\langle (a, b) : R \mid \alpha, -, - \rangle$

where a and b are individuals, C is a concept, R is a role, $\alpha \in \mathcal{V}$, f_c is the conjunction function, f_d is the disjunction function, and $-$ denotes that the corresponding combination function is not applicable.

For instance, the assertion "Mary is either tall or thin, and smart with degree between 0.6 and 0.8" can be expressed as $\langle \mathit{Mary} : (\mathit{Tall} \sqcup \mathit{Thin}) \sqcap \mathit{Smart} \mid [0.6,$

$0.8], min, -\rangle$. Here, the concept conjunction is interpreted using the *min* function, and the disjunction function is interpreted using *max*.

In terms of the semantics of the assertions, an interpretation $\mathcal{I}$ satisfies $\langle a : C \mid \alpha, f_c, f_d \rangle$ if $C^{\mathcal{I}}(a^{\mathcal{I}}) = \alpha$, and $\mathcal{I}$ satisfies $\langle (a, b) : R \mid \alpha, -, - \rangle$ if $R^{\mathcal{I}}(a^{\mathcal{I}}, b^{\mathcal{I}}) = \alpha$.

There are two types of individuals that could be in an ABox – defined individuals and generated individuals, defined as follows. We also introduce the notion of predecessor and ancestor in Definition 3.2.2.

Definition 3.2.1 (Defined/Generated Individual) Let $\mathbf{I}$ be the set of all individuals in an ABox. We call individuals whose names explicitly appear in the ABox as *defined individuals* ($\mathbf{I_D}$), and those generated by the reasoning procedure as *generated individuals* ($\mathbf{I_G}$). Note that $\mathbf{I_D} \cap \mathbf{I_G} = \emptyset$, and $\mathbf{I_D} \cup \mathbf{I_G} = \mathbf{I}$.

Definition 3.2.2 (Predecessor/Ancestor) An individual a is a *predecessor* of an individual b (or b is an R-successor of a) if the ABox $\mathcal{A}$ contains the assertion $\langle (a, b) : R \mid \alpha, -, - \rangle$. An individual a is an *ancestor* of b if it is either a predecessor of b or there exists a chain of assertions $\langle (a, b_1) : R_1 \mid \alpha_1, -, - \rangle$, $\langle (b_1, b_2) : R_2 \mid \alpha_2, -, - \rangle$,..., $\langle (b_k, b) : R_{k+1} \mid \alpha_{k+1}, -, - \rangle$ in $\mathcal{A}$.

We now define the notion of "admissibility" for $\mathcal{ALC}_U$ axioms, assertions, and knowledge bases.

Definition 3.2.3 (Admissibility of $\mathcal{ALC}_U$ Axiom/Assertion) An $\mathcal{ALC}_U$ axiom (resp. assertion) is admissible if the combination functions associated with it and the negation operator of the certainty lattice satisfy the pre-defined properties specified in Section 3.1.2.

Definition 3.2.4 (Admissibility of $\mathcal{ALC}_U$ Knowledge Base) An $\mathcal{ALC}_U$ knowledge base is admissible if all its axioms and assertions are admissible.

3.3 $\mathcal{ALC}_U$ Reasoning Procedure

The $\mathcal{ALC}_U$ framework is equipped with a tableau-based reasoning procedure (also called the $\mathcal{ALC}_U$ tableau algorithm) that takes into account the presence of uncertainties associated with axioms and assertions in the $\mathcal{ALC}_U$ knowledge base. Figure 5 gives an overview of our reasoning procedure. The rectangles represent data or knowledge bases, the arrows show the data flow, and the gray rounded boxes show where data processing is performed.

Compared with the reasoning procedure for standard DL shown in Figure 2, the main difference between the two is that, for the uncertainty case, in addition to deriving assertions, each completion rule application also generates a set of constraints which encode the semantics of the assertion. Once there is no more rule applicable, we feed the generated set of constraints into the constraint solver to check its solvability, and return the inference result.

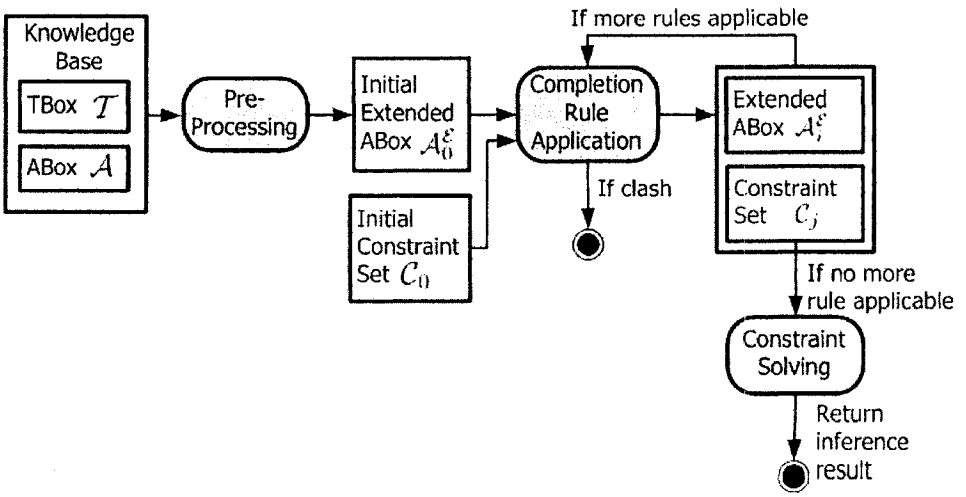

Figure 5: Overview of the $\mathcal{ALCC}_U$ reasoning procedure

In what follows, we present the $\mathcal{ALCC}_U$ tableau algorithm in detail. We start by introducing the reasoning services offered, and then present the pre-processing phase and the completion rules. We also establish correctness of our $\mathcal{ALCC}_U$ tableau algorithm.

3.3.1 $\mathcal{ALCC}_U$ Reasoning Services

The $\mathcal{ALCC}_U$ reasoning services include the consistency, the entailment, and the subsumption problems as described below.

Consistency Problem

To check if an admissible $\mathcal{ALCC}_U$ knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ is consistent, we first apply the pre-processing steps (see Section 3.3.2) to obtain the initial extended ABox, $\mathcal{A}_0^{\mathcal{E}}$. In addition, the constraints set $\mathcal{C}_0$ is initialized to the empty set $\{\}$. We then apply

the completion rules (see Section 3.3.3) to derive implicit knowledge from explicit ones. Through the application of each rule, we add any assertions that are derived to the extended ABox $\mathcal{A}_i^{\mathcal{E}}$. In addition, constraints which denote the semantics of the assertions are added to the constraints set $\mathcal{C}_j$, in the form of linear or nonlinear inequations. The completion rules are applied in arbitrary order as long possible, until either $\mathcal{A}_i^{\mathcal{E}}$ contains a clash or no further rule could be applied to $\mathcal{A}_i^{\mathcal{E}}$. If $\mathcal{A}_i^{\mathcal{E}}$ contains a clash, the knowledge base is inconsistent. Otherwise, the system of inequations in $\mathcal{C}_j$ is fed into the constraint solver to check its solvability. If the system of inequations is unsolvable, the knowledge base is inconsistent. Otherwise, the knowledge base is consistent.

Entailment Problem

Given an admissible $\mathcal{ALC}_U$ knowledge base Σ, the entailment problem determines the degree to which an assertion X is true. Like in standard DLs, the entailment problem can be reduced to the consistency problem. That is, let X be an assertion of the form $\langle a : C \mid x_{a:C}, f_c, f_d \rangle$. The degree that Σ entails X is the degree of $x_{a:C}$ such that $\Sigma \cup \{\langle a : \neg C \mid x_{a:\neg C}, f_c, f_d \rangle\}$ is consistent.

Subsumption Problem

Let $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ be an admissible $\mathcal{ALC}_U$ knowledge base, and $\langle C \sqsubseteq D \mid x_{C \sqsubseteq D}, f_c, f_d \rangle$ be the subsumption relationship to be checked. The subsumption problem determines the degree to which C is subsumed by D with respect to the TBox $\mathcal{T}$. Like in

standard DLs, this problem can be reduced to the consistency problem by finding the degree of $x_{a:\neg C \sqcup D}$ such that $\Sigma \cup \{\langle a : C \sqcap \neg D \mid x_{a:C \sqcap \neg D}, f_c, f_d \rangle\}$ is consistent, where a is a new, generated individual name.

To present the $\mathcal{ALC}_U$ tableau algorithm further, we need to define a few terms as follows.

Definition 3.3.1 (Node Label, Node Constraint, Edge Label) As in the standard DL, the model being constructed by the $\mathcal{ALC}_U$ tableau algorithm can be thought as a forest, which is a collection of trees, with nodes corresponding to individuals, edges corresponding to relationships/roles between individuals, and root nodes corresponding to individuals present in the initial extended ABox. Each node is associated with a *node label*, $\mathcal{L}(individual)$, to show the concept assertions associated with a particular individual, as well as a *node constraint*, $\mathcal{C}(individual)$, for the corresponding constraints. Unlike in the standard DL where each element in the node label is a concept, each element in our node label is a quadruple, $\langle Concept, Certainty, ConjunctionFunction, DisjunctionFunction \rangle$. Finally, unlike in the standard DL where each edge is labeled with a role name, each edge in our case is associated with an *edge label*, $\mathcal{L}(\langle individual_1, individual_2 \rangle)$ which consists of a pair of elements $\langle Role, Certainty \rangle$. In case the certainty is a variable, "–" is used as a place holder.

Definition 3.3.2 (Evaluation) Let $Var(\mathcal{C})$ be the set of certainty variables occurring in the constraints set $\mathcal{C}$, and $\mathcal{V}$ be the certainty domain. If the system of inequations in $\mathcal{C}$ is solvable, the solution to the constraints set $\pi : Var(\mathcal{C}) \rightarrow \mathcal{V}$ is called an *evaluation.*

Definition 3.3.3 (Complete) An extended ABox $\mathcal{A}_c^{\mathcal{E}}$ is complete if no more completion rule can be applied to $\mathcal{A}_c^{\mathcal{E}}$ and the set of constraints $\mathcal{C}$ obtained during the rule application is solvable.

Definition 3.3.4 (Model) Let $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$ be an admissible knowledge base, and $\mathcal{A}_c^{\mathcal{E}}$ be the extended ABox obtained by applications of the completion rules to the extended ABox $\mathcal{A}_i^{\mathcal{E}}$. Also, let $\mathcal{I}$ be an interpretation, π be an evaluation, α be a certainty value in the certainty domain, and x_X be the variable representing the certainty of assertion X. The pair $\langle \mathcal{I}, \pi \rangle$ is a model of the extended ABox $\mathcal{A}_c^{\mathcal{E}}$ if all the following hold:

- for each assertion $\langle a : C \mid \alpha, f_c, f_d \rangle \in \mathcal{A}_c^{\mathcal{E}}$, $C^{\mathcal{I}}(a) = \alpha$.
- for each assertion $\langle a : C \mid x_{a:C}, f_c, f_d \rangle \in \mathcal{A}_c^{\mathcal{E}}$, $C^{\mathcal{I}}(a) = \pi(x_{a:C})$.
- for each assertion $\langle (a, b) : R \mid \alpha, -, - \rangle \in \mathcal{A}_c^{\mathcal{E}}$, $R^{\mathcal{I}}(a, b) = \alpha$.
- for each assertion $\langle (a, b) : R \mid x_{(a,b):R}, -, - \rangle \in \mathcal{A}_c^{\mathcal{E}}$, $R^{\mathcal{I}}(a, b) = \pi(x_{(a,b):R})$.

The knowledge base Σ is consistent if there exists a model for the extended ABox $\mathcal{A}_c^{\mathcal{E}}$.

3.3.2 Pre-processing Phase

The $\mathcal{ALCC}_U$ tableau algorithm starts by applying the following pre-processing steps. Note that, since GCI is the general form of the concept subsumption, and a concept equation is the general form of the concept definition, we only use the general forms here.

1. Replace each axiom of the form $\langle C \equiv D \mid \alpha, f_c, f_d \rangle$ with the following two equivalent axioms: $\langle C \sqsubseteq D \mid \alpha, f_c, f_d \rangle$ and $\langle D \sqsubseteq C \mid \alpha, f_c, f_d \rangle$.

2. Transform every axiom in the TBox into its normal form. That is, replace each axiom of the form $\langle C \sqsubseteq D \mid \alpha, f_c, f_d \rangle$ with $\langle \top \sqsubseteq \neg C \sqcup D \mid \alpha, f_c, f_d \rangle$.

3. Transform every concept (in the TBox and the ABox) into its NNF (refer to Definition 2.1.4). Due to the Concept Negation properties and Inter-Constructor properties described in Section 3.1.2, the equivalence rules described in Definition 2.1.4 can be used to transform any $\mathcal{ALCC}_U$ concept into its NNF. That is, let C and D be concepts, and R be a role. NNF of an $\mathcal{ALCC}_U$ concept can be obtained by applying the following equivalence rules:

   - $\neg\neg(C) \equiv C$
   - $\neg(C \sqcup D) \equiv \neg C \sqcap \neg D$
   - $\neg(C \sqcap D) \equiv \neg C \sqcup \neg D$
   - $\neg\exists R.C \equiv \forall R.\neg C$
   - $\neg\forall R.C \equiv \exists R.\neg C$

4. Augment the ABox $\mathcal{A}$ with respect to the TBox $\mathcal{T}$. That is, for each individual a in $\mathcal{A}$ and each axiom $\langle \top \sqsubseteq \neg C \sqcup D \mid \alpha, f_c, f_d \rangle$ in $\mathcal{T}$, add $\langle a : \neg C \sqcup D \mid \alpha, f_c, f_d \rangle$ to $\mathcal{A}$.

We call the resulting ABox after the pre-processing phase as the *initial extended ABox*, denoted by $\mathcal{A}_0^{\mathcal{E}}$.

3.3.3 $\mathcal{ALC}_U$ Completion Rules

The $\mathcal{ALC}_U$ completion rules are a set of satisfiability preserving transformation rules, where each rule either detects inconsistencies in the knowledge base or derives one or more assertions and/or constraints that encode the semantics of the knowledge base. Since the application of the completion rules may lead to nontermination (see Section 3.4 for an example), we introduce the notion of blocking to handle this situation.

Definition 3.3.5 (Blocking) Let $a, b \in \mathbf{I_G}$ be generated individuals in the extended ABox $\mathcal{A}_i^{\mathcal{E}}$, $\mathcal{A}_i^{\mathcal{E}}(a)$ and $\mathcal{A}_i^{\mathcal{E}}(b)$ be all the concept assertions for a and b in $\mathcal{A}_i^{\mathcal{E}}$. An individual b is blocked by some ancestor a (or a is the blocking individual for b) if $\mathcal{A}_i^{\mathcal{E}}(b) \subseteq \mathcal{A}_i^{\mathcal{E}}(a)$.

Let $\mathcal{T}$ be the TBox obtained after the pre-processing phase, $\mathcal{A}_i^{\mathcal{E}}$ be an extended ABox which is initialized to be the initial extended ABox $\mathcal{A}_0^{\mathcal{E}}$, and $\mathcal{C}_0$ be the initial constraints set. Also, let α and β be certainty values, and let Γ be either a certainty value or be the variable x_X denoting the certainty of assertion X. The $\mathcal{ALC}_U$ completion rules are defined as follows.

Clash Triggers:

$\langle a : \bot \mid \alpha, -, - \rangle \in \mathcal{A}_i^{\mathcal{E}}$, with $\alpha \succ l$

$\langle a : \top \mid \alpha, -, - \rangle \in \mathcal{A}_i^{\mathcal{E}}$, with $\alpha \prec t$

$\{\langle a : A \mid \alpha, -, - \rangle, \langle a : A \mid \beta, -, - \rangle\} \subseteq \mathcal{A}_i^{\mathcal{E}}$, with $\otimes(\alpha, \beta) = l$

$\{\langle (a, b) : R \mid \alpha, -, - \rangle, \langle (a, b) : R \mid \beta, -, - \rangle\} \subseteq \mathcal{A}_i^{\mathcal{E}}$, with $\otimes(\alpha, \beta) = l$

The purpose of the clash triggers is to detect possible inconsistencies in the knowledge base. Note that the last two clash triggers detect the contradiction in terms of the certainty values specified for the same assertion. For example, suppose the certainty domain is $\mathcal{V} = \mathcal{C}[0, 1]$, i.e., the set of closed subintervals $[\alpha, \beta]$ in [0, 1] where $\alpha \preceq \beta$. If a knowledge base contains both assertions $\langle John : Tall \mid [0, 0.2], -, - \rangle$ and $\langle John : Tall \mid [0.7, 1], -, - \rangle$, then the third clash trigger will detect this as an inconsistency.

Concept Assertion Rule:

if $\langle a : A \mid \Gamma, -, - \rangle \in \mathcal{A}_i^{\mathcal{E}}$

then if $(x_{a:A} = \Gamma) \notin \mathcal{C}_j$ and Γ is not the variable $x_{a:A}$

then $\mathcal{C}_{j+1} = \mathcal{C}_j \cup \{(x_{a:A} = \Gamma)\}$

if $(x_{a:\neg A} = \sim\Gamma) \notin \mathcal{C}_j$

then $\mathcal{C}_{j+1} = \mathcal{C}_j \cup \{(x_{a:\neg A} = \sim\Gamma)\}$

This rule simply adds the certainty value of each atomic concept assertion and

its negation to the constraints set C_j. For example, suppose we have the assertion $\langle John : Tall \mid [0.6, 1], -, - \rangle$ in the extended ABox. If the certainty domain is $\mathcal{V} = C[0, 1]$ and if the negation function is $\sim(x) = t - x$, where t is the top certainty in the lattice, then we add the constraints $(x_{John:Tall} = [0.6, 1])$ and $(x_{John:\neg Tall} = [0, 0.4])$ to the constraints set C_j. On the other hand, if we have the assertion $\langle John : Tall \mid x_{John:Tall}, -, - \rangle$ in the extended ABox, we add the constraint $(x_{John:\neg Tall} = t - x_{John:Tall})$ to C_j.

Role Assertion Rule:

if $\langle (a, b) : R \mid \Gamma, -, - \rangle \in \mathcal{A}_i^{\mathcal{E}}$

then if $(x_{(a,b):R} = \Gamma) \notin C_j$ and Γ is not the variable $x_{(a,b):R}$

then $C_{j+1} = C_j \cup \{(x_{(a,b):R} = \Gamma)\}$

if $(x_{\neg(a,b):R} = \sim\Gamma) \notin C_j$

then $C_{j+1} = C_j \cup \{(x_{\neg(a,b):R} = \sim\Gamma)\}$

Similar to the Concept Assertion Rule, this rule simply adds the certainty value of each atomic role assertion and its negation to the constraints set C_j. For example, suppose we have the assertion $\langle (John, Diabetes) : hasDisease \mid 0.9, -, - \rangle$ in the extended ABox. If the certainty domain is $\mathcal{V} = [0, 1]$ and if the negation function is $\sim(x) = t - x$ where t is the top certainty in the lattice, then we add the constraints $(x_{(John,Diabetes):hasDisease} = 0.9)$ and $(x_{(John,Diabetes):\neg hasDisease} = 0.1)$ to C_j. On the other hand, if the assertion $\langle (John, Diabetes) : hasDisease \mid x_{(John,Diabetes):hasDisease},$

$-,-\rangle$ is in the ABox, the constraint $(x_{(John,Diabetes):\neg hasDisease} = t - x_{(John,Diabetes):hasDisease})$ is added to $\mathcal{C}_j$.

Negation Rule:

if $\langle a : \neg A \mid \Gamma, -, -\rangle \in \mathcal{A}_i^{\mathcal{E}}$

then if $\langle a : A \mid\sim\Gamma, -, -\rangle \notin \mathcal{A}_i^{\mathcal{E}}$

then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle a : A \mid\sim\Gamma, -, -\rangle\}$

The intuition behind the Negation Rule is that, if we know an assertion has certainty value Γ, then the certainty of its negation can be obtained by applying the negation operator in the lattice to Γ. For example, suppose the certainty domain is $\mathcal{V} = [0, 1]$, and the negation operator is defined as $\sim(x) = 1 - x$. Then, if the assertion $\langle John : \neg Tall \mid 0.8, -, -\rangle$ is in the ABox, we could infer $\langle John : Tall \mid 0.2, -, -\rangle$, which is added to the extended ABox.

Conjunction Rule:

if $\langle a : C \sqcap D \mid \Gamma, f_c, f_d\rangle \in \mathcal{A}_i^{\mathcal{E}}$

then for each $\Psi \in \{C, D\}$

if Ψ is atomic and $\langle a : \Psi \mid x_{a:\Psi}, -, -\rangle \notin \mathcal{A}_i^{\mathcal{E}}$

then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle a : \Psi \mid x_{a:\Psi}, -, -\rangle\}$

else if Ψ is not atomic and $\langle a : \Psi \mid x_{a:\Psi}, f_c, f_d\rangle \notin \mathcal{A}_i^{\mathcal{E}}$

then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle a : \Psi \mid x_{a:\Psi}, f_c, f_d\rangle\}$

if $(f_c(x_{a:C}, x_{a:D}) = \Gamma) \notin \mathcal{C}_j$,

then $\mathcal{C}_{j+1} = \mathcal{C}_j \cup \{(f_c(x_{a:C}, x_{a:D}) = \Gamma)\}$

The intuition behind this rule is that, if we know an individual is in $C \sqcap D$, then we know it is in both C and D. In addition, according to the semantics of the description language, we know that the semantics of $a : C \sqcap D$ is defined by applying the conjunction function to the interpretation of $a : C$ and the interpretation of $a : D$.

For example, if the extended ABox includes the assertion $\langle Mary : Tall \sqcap Thin \mid 0.8, min, max\rangle$, then we could infer that $\langle Mary : Tall \mid x_{Mary:Tall}, -, -\rangle$ and $\langle Mary : Thin \mid x_{Mary:Thin}, -, -\rangle$. In addition, the constraint $min(x_{Mary:Tall}, x_{Mary:Thin}) = 0.8$ must be satisfied.

Disjunction Rule:

if $\langle a : C \sqcup D \mid \Gamma, f_c, f_d\rangle \in \mathcal{A}_i^{\mathcal{E}}$

then for each $\Psi \in \{C, D\}$

if Ψ is atomic and $\langle a : \Psi \mid x_{a:\Psi}, -, -\rangle \notin \mathcal{A}_i^{\mathcal{E}}$

then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle a : \Psi \mid x_{a:\Psi}, -, -\rangle\}$

else if Ψ is not atomic and $\langle a : \Psi \mid x_{a:\Psi}, f_c, f_d\rangle \notin \mathcal{A}_i^{\mathcal{E}}$

then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle a : \Psi \mid x_{a:\Psi}, f_c, f_d\rangle\}$

if $(f_d(x_{a:C}, x_{a:D}) = \Gamma) \notin \mathcal{C}_j$,

then $\mathcal{C}_{j+1} = \mathcal{C}_j \cup \{(f_d(x_{a:C}, x_{a:D}) = \Gamma)\}$

The intuition behind this rule is that, if we know an individual is in $C \sqcup D$, then

we know it is in either C, D, or in both. In addition, according to the semantics of the description language, we know that the semantics of $a : C \sqcup D$ is defined by applying the disjunction function to the interpretation of $a : C$ and that of $a : D$.

It is interesting to note is that the disjunction rule in the standard DL is non-deterministic, since it can be applied in different ways to the same ABox. However, note that the disjunction rule in $\mathcal{ALC}_U$ is deterministic. This is because the semantics of the concept disjunction is now encoded in the disjunction function in the form of a constraint. For example, if the extended ABox includes the assertion $\langle Mary : Tall \sqcup Thin \mid 0.8, min, max \rangle$, then we infer that $\langle Mary : Tall \mid x_{Mary:Tall}, -, -\rangle$ and $\langle Mary : Thin \mid x_{Mary:Thin}, -, -\rangle$. Moreover, the constraint $max(x_{Mary:Tall}, x_{Mary:Thin}) = 0.8$ must be satisfied, which means that $x_{Mary:Tall} = 0.8$, or $x_{Mary:Thin} = 0.8$, or $x_{Mary:Tall} = x_{Mary:Thin} = 0.8$.

Role Exists Restriction Rule:

if $\langle a : \exists R.C \mid \Gamma, f_c, f_d \rangle \in \mathcal{A}_i^{\mathcal{E}}$ and a is not blocked

then if $\nexists$ individual b such that $(f_c(x_{(a,b):R}, x_{b:C}) = x_{a:\exists R.C}) \in \mathcal{C}_j$

then let b be a new individual

$\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle (a,b) : R \mid x_{(a,b):R}, -, -\rangle\}$

if C is atomic

then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle b : C \mid x_{b:C}, -, -\rangle\}$

else $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle b : C \mid x_{b:C}, f_c, f_d\rangle\}$

$\mathcal{C}_{j+1} = \mathcal{C}_j \cup \{(f_c(x_{(a,b):R}, x_{b:C}) = x_{a:\exists R.C})\}$

for each axiom $\langle \top \sqsubseteq \neg C \sqcup D \mid \alpha, f_c, f_d \rangle$ in the TBox $\mathcal{T}$

$$\mathcal{A}^{\mathcal{E}}_{i+1} = \mathcal{A}^{\mathcal{E}}_{i} \cup \{\langle b : \neg C \sqcup D \mid \alpha, f_c, f_d \rangle\}$$

if Γ is not the variable $x_{a:\exists R.C}$

then if $(x_{a:\exists R.C} = \Gamma') \in \mathcal{C}_j$

then if $\Gamma \neq \Gamma'$ and Γ is not an element in Γ'

then $\mathcal{C}_{j+1} = \mathcal{C}_j \setminus \{(x_{a:\exists R.C} = \Gamma')\} \cup \{(x_{a:\exists R.C} = \oplus(\Gamma, \Gamma'))\}$

else $\mathcal{C}_{j+1} = \mathcal{C}_j \cup \{(x_{a:\exists R.C} = \Gamma)\}$

The intuition behind this rule is that, if we know that an individual a is in $\exists R.C$, there must exist at least an individual, say b, such that a is related to b through the relationship R, and b is in the concept C. If no such individual b exists in the extended ABox, then we create such a new individual. In addition, this new individual must satisfy all the axioms in the TBox. For example, suppose the assertion $\langle Tom : \exists hasDisease.Diabetes \mid [0.4, 0.6], min, max \rangle$ is in the extended ABox and the axiom $\langle \top \sqsubseteq \neg Obese \sqcup \exists hasDisease.Diabetes \mid [0.7, 1], \times, ind \rangle$ is in the TBox. Assume also that the ABox originally does not contain any individual b such that Tom is related to b through the role $hasDisease$, and b is in the concept $Diabetes$. Then, we could infer $\langle (Tom, d1) : hasDisease \mid x_{(Tom,d1):hasDisease}, -, - \rangle$ and $\langle d1 : Diabetes \mid x_{d1:Diabetes}, -, - \rangle$, where $d1$ is a new individual. In addition, since $d1$ must satisfy the axioms in the TBox, the assertion $\langle d1 : \neg Obese \sqcup \exists hasDisease.$ $Diabetes \mid [0.7, 1], \times, ind \rangle$ is added to the extended ABox. Finally, the constraints $(min(x_{(Tom,d1):hasDisease}, x_{d1:Diabetes}) = x_{Tom:\exists hasDisease.Diabetes})$ as well as $(x_{Tom:\exists hasDisease.}$

$_{Diabetes} = [0.4, 0.6]$) must be satisfied. Now, suppose there is another assertion $\langle Tom : \exists hasDisease.Diabetes \mid [0.5, 0.9], min, max \rangle$ in the extended ABox. Then, when we apply the Role Exists Restriction Rule, we do not generate a new individual. Instead, we simply replace the constraint $(x_{Tom:\exists hasDisease.Diabetes} = [0.4, 0.6])$ in $\mathcal{C}_j$ with the constraint $(x_{Tom:\exists hasDisease.Diabetes} = sup([0.5, 0.9], [0.4, 0.6]))$, where sup is the join operator in the lattice $\oplus$. This new constraint takes into account the certainty value of the current assertion as well as that of the previous assertion.

Role Value Restriction Rule:

if $\{\langle a : \forall R.C \mid \Gamma, f_c, f_d \rangle, \langle (a, b) : R \mid \Gamma', -, - \rangle\} \subseteq \mathcal{A}_i^{\mathcal{E}}$

then if C is atomic and $\langle b : C \mid x_{b:C}, -, - \rangle \notin \mathcal{A}_i^{\mathcal{E}}$

then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle b : C \mid x_{b:C}, -, - \rangle\}$

else if C is not atomic and $\langle b : C \mid x_{b:C}, f_c, f_d \rangle \notin \mathcal{A}_i^{\mathcal{E}}$

then $\mathcal{A}_{i+1}^{\mathcal{E}} = \mathcal{A}_i^{\mathcal{E}} \cup \{\langle b : C \mid x_{b:C}, f_c, f_d \rangle\}$

if $(f_d(x_{\neg(a,b):R}, x_{b:c}) = x_{a:\forall R.C}) \notin \mathcal{C}_j$

then $\mathcal{C}_{j+1} = \mathcal{C}_j \cup \{(f_d(x_{\neg(a,b):R}, x_{b:c}) = x_{a:\forall R.C})\}$

if Γ is not the variable $x_{a:\forall R.C}$

then if $(x_{a:\forall R.C} = \Gamma'') \in \mathcal{C}_j$

then if $\Gamma \neq \Gamma''$ and Γ is not an element in Γ''

then $\mathcal{C}_{j+1} = \mathcal{C}_j \setminus \{(x_{a:\forall R.C} = \Gamma'')\} \cup \{(x_{a:\forall R.C} = \otimes(\Gamma, \Gamma''))\}$

else $\mathcal{C}_{j+1} = \mathcal{C}_j \cup \{(x_{a:\forall R.C} = \Gamma)\}$

The structure of the Role Value Restriction rule is similar to that of Role Exists Restriction rule, although they have different semantics. The intuition behind the Role Value Restriction rule is that, if we know that an individual a is in $\forall R.C$, and if there is an individual b such that a is related to b through the relationship R, then b must be in the concept C. For example, assume we have assertions $\langle Jim : \forall hasPet.Dog \mid [0.4, 0.6], min, max\rangle$ and $\langle (Jim, d1) : hasPet \mid [0.5, 0.8], -, -\rangle$ in the extended ABox. Then, we could infer $\langle d1 : Dog \mid x_{d1:Dog}, -, -\rangle$. In addition, the constraints $(max(x_{(Jim,d1):\neg hasPet}, x_{d1:Dog}) = x_{Jim:\forall hasPet.Dog})$ as well as $(x_{Jim:\forall hasPet.Dog} = [0.4, 0.6])$ must be satisfied. Now, suppose we have yet another assertion $\langle Jim : \forall hasPet.Dog \mid [0.5, 0.9], min, max\rangle$ in the extended ABox. Then, when we apply the Role Value Restriction rule, we simply replace the constraint $(x_{Jim:\forall hasPet.Dog} = [0.4, 0.6])$ in $\mathcal{C}_j$ with the constraint $(x_{Jim:\forall hasPet.Dog} = inf([0.5, 0.9], [0.4, 0.6]))$, where inf is the meet operator in the lattice $\otimes$. Note that the new constraint takes into account the certainty value of the current assertion as well as that of the previous assertion.

Note 3.3.6 Like in standard DLs, our tableau algorithm treats each axiom in $\mathcal{T}$ as a meta constraint. That is, for each individual a in $\mathcal{A}$ and each axiom $\langle C \sqsubseteq D \mid \alpha, f_c, f_d\rangle$ in $\mathcal{T}$, we add $\langle a : \neg C \sqcup D \mid \alpha, f_c, f_d\rangle$ to $\mathcal{A}$. This results in a large number of assertions with concept disjunction be added to $\mathcal{A}$. In standard DLs, this would dramatically extend the search space and is the main cause for empirical intractability, since the disjunction rule in standard DLs is non-deterministic. However, since the disjunction rule in $\mathcal{ALC}_U$ is deterministic, large number of assertions with concept disjunction

does not cause any problem in our context. Note also that the idea of unfolding does not work in $\mathcal{ALC}_U$ because each axiom is associated with a certainty value and a pair of combination functions. Consider the axioms $\langle A \equiv B \sqcap \exists R.C \mid 0.6, min, max\rangle$ and $\langle D \equiv A \sqcup E \mid 0.7, \times, ind\rangle$. In the standard DLs, we would replace A on the right hand side of the concept definition D with the definition of A. However, we cannot do so in our context, since there is a certainty value (0.6) and two combination functions $\langle min, max\rangle$ associated with the concept definition A.

3.3.4 Correctness of the $\mathcal{ALC}_U$ Tableau Algorithm

We establish the correctness of the $\mathcal{ALC}_U$ tableau algorithm by showing that it is sound, complete, and it terminates, as shown below.

Lemma 3.3.7 (Soundness) Let $\mathcal{A}^{\mathcal{E}'}$ be an extended ABox obtained from the extended ABox $\mathcal{A}^{\mathcal{E}}$ by applying the completion rule. Let $\mathcal{I}$ be an interpretation and π be an evaluation. Then, $\langle \mathcal{I}, \pi\rangle$ is a model of $\mathcal{A}^{\mathcal{E}}$ iff $\langle \mathcal{I}, \pi\rangle$ is a model of $\mathcal{A}^{\mathcal{E}'}$.

Proof.

The "if" direction: Let $\mathcal{C}$ be the constraints set associated with the extended ABox $\mathcal{A}^{\mathcal{E}}$, and $\mathcal{C}'$ be the constraints set associated with the extended ABox $\mathcal{A}^{\mathcal{E}'}$. Since $\mathcal{A}^{\mathcal{E}} \subseteq \mathcal{A}^{\mathcal{E}'}$ and $\mathcal{C} \subseteq \mathcal{C}'$, if $\langle \mathcal{I}, \pi\rangle$ is a model of $\mathcal{A}^{\mathcal{E}'}$, it is also a model of $\mathcal{A}^{\mathcal{E}}$.

The "only if" direction: We prove the claim by considering each completion rule.

Since the cases of Concept Assertion, Role Assertion, and Negation rules are straightforward, we skip them here.

Let C and D be concepts, a and b be individuals in the domain, and R be a role. Also, let $\langle \mathcal{I}, \pi \rangle$ be a model of $\mathcal{A}^{\mathcal{E}}$, and assume that the following completion rule is triggered.

Conjunction Rule: By applying the conjunction rule to $\langle a : C \sqcap D \mid \Gamma, f_c, f_d \rangle$ in $\mathcal{A}^{\mathcal{E}}$, we obtain the extended ABox $\mathcal{A}^{\mathcal{E}'} = \mathcal{A}^{\mathcal{E}} \cup \{\langle a : C \mid x_{a:C}, f_c, f_d \rangle, \langle a : D \mid x_{a:D}, f_c, f_d \rangle\}$ and the constraints set $\mathcal{C}' = \mathcal{C} \cup \{(f_c(x_{a:C}, x_{a:D}) = \Gamma)\}$. Since $\langle \mathcal{I}, \pi \rangle$ is a model of $\mathcal{A}^{\mathcal{E}}$, $\mathcal{I}$ satisfies $\langle a : C \sqcap D \mid \Gamma, f_c, f_d \rangle$, and we know that, by definition, $(C \sqcap D)^{\mathcal{I}}(a) = f_c(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a)) = \Gamma$. Therefore, the pair $(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$ is in $\{(x, y) \mid f_c(x, y) = \Gamma\}$. Hence, there exists some $\alpha_1, \alpha_2 \in \mathcal{V}$ such that $C^{\mathcal{I}}(a) = \alpha_1$ and $D^{\mathcal{I}}(a) = \alpha_2$. That is, $\mathcal{I}$ satisfies both $\langle a : C \mid \alpha_1, f_c, f_d \rangle$ and $\langle a : D \mid \alpha_2, f_c, f_d \rangle$.

Disjunction Rule: Applying the disjunction rule to $\langle a : C \sqcup D \mid \Gamma, f_c, f_d \rangle$ in $\mathcal{A}^{\mathcal{E}}$ yields the extended ABox $\mathcal{A}^{\mathcal{E}'} = \mathcal{A}^{\mathcal{E}} \cup \{\langle a : C \mid x_{a:C}, f_c, f_d \rangle, \langle a : D \mid x_{a:D}, f_c, f_d \rangle\}$ and the constraints set $\mathcal{C}' = \mathcal{C} \cup \{(f_d(x_{a:C}, x_{a:D}) = \Gamma)\}$. Since $\langle \mathcal{I}, \pi \rangle$ is a model of $\mathcal{A}^{\mathcal{E}}$, $\mathcal{I}$ satisfies $\langle a : C \sqcup D \mid \Gamma, f_c, f_d \rangle$, and we know by definition that $(C \sqcup D)^{\mathcal{I}}(a) = f_d(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a)) = \Gamma$. Therefore, the pair $(C^{\mathcal{I}}(a), D^{\mathcal{I}}(a))$ is in $\{(x, y) \mid f_d(x, y) = \Gamma\}$. Hence, there exists some $\alpha_1, \alpha_2 \in \mathcal{V}$ such that $C^{\mathcal{I}}(a) = \alpha_1$ and $D^{\mathcal{I}}(a) = \alpha_2$. That is, $\mathcal{I}$ satisfies both $\langle a : C \mid \alpha_1, f_c, f_d \rangle$ and $\langle a : D \mid \alpha_2, f_c, f_d \rangle$.

Role Exists Restriction Rule: When the role exists restriction rule is applied to $\langle a : \exists R.C \mid \Gamma, f_c, f_d \rangle$ in $\mathcal{A}^{\mathcal{E}}$, there are two possible augmentations to the extended ABox/constraints set: (i) There is already an individual b such that $\{\langle (a, b) : R \mid x_{(a,b)}$:

$_R, -, -\rangle, \langle b : C \mid x_{b:C}, f_c, f_d\rangle\} \subseteq \mathcal{A}_c$. Also, $\{(f_c(x_{(a,b):R}, x_{b:C}) = x_{a:\exists R.C}), (x_{a:\exists R.C} = \Gamma')\}$ $\subseteq \mathcal{C}$. In this case, we replace the constraint $(x_{a:\exists R.C} = \Gamma')$ with $(x_{a:\exists R.C} = \oplus(\Gamma, \Gamma'))$.

(ii) A new individual b is generated, and we have $\mathcal{A}^{\mathcal{E}'} = \mathcal{A}^{\mathcal{E}} \cup \{\langle (a,b) : R \mid x_{(a,b):R}, -,$ $-\rangle, \langle b : C \mid x_{b:C}, f_c, f_d\rangle\}$ as well as $\mathcal{C}' = \mathcal{C} \cup \{(f_c(x_{(a,b):R}, x_{b:C}) = x_{a:\exists R.C}), (x_{a:\exists R.C} = \Gamma)\}$. Since $\langle \mathcal{I}, \pi\rangle$ is a model of $\mathcal{A}^{\mathcal{E}}$, we know that $\mathcal{I}$ satisfies $\langle a : \exists R.C \mid \Gamma, f_c, f_d\rangle$, the evaluation π gives the certainty that a is in $\exists R.C$ (denoted $\pi(x_{a:\exists R.C})$), and by definition, we know that $(\exists R.C)^{\mathcal{I}}(a) = \oplus_{b\in\Delta^{\mathcal{I}}}\{f_c(R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\} = \pi(x_{a:\exists R.C})$. Hence, there are certainty values $\alpha_1, \alpha_2 \in \mathcal{V}$ such that $R^{\mathcal{I}}(a,b) = \alpha_1$ and $C^{\mathcal{I}}(b) = \alpha_2$. That is, $\mathcal{I}$ satisfies both $\langle (a,b) : R \mid \alpha_1, -, -\rangle$ and $\langle b : C \mid \alpha_2, f_c, f_d\rangle$.

Role Value Restriction Rule: Assume that the role value restriction rule is applied to $\langle a : \forall R.C \mid \Gamma, f_c, f_d\rangle$ in $\mathcal{A}^{\mathcal{E}}$. Then, for every individual b that is an R-successor of individual a, we either obtain the extended ABox $\mathcal{A}^{\mathcal{E}'} = \mathcal{A}^{\mathcal{E}} \cup \{\langle b : C \mid x_{b:C}, f_c, f_d\rangle\}$ and the constraints set $\mathcal{C}' = \mathcal{C} \cup \{(f_d(x_{\neg(a,b):R}, x_{b:c}) = x_{a:\forall R.C}),\ (x_{a:\forall R.C} = \Gamma)\}$, or if the constraint $(x_{a:\forall R.C} = \Gamma'')$ is already in $\mathcal{C}$, we replace it with $(x_{a:\forall R.C} = \otimes(\Gamma, \Gamma''))$. Since $\langle \mathcal{I}, \pi\rangle$ is a model of $\mathcal{A}^{\mathcal{E}}$, $\mathcal{I}$ satisfies $\langle a : \forall R.C \mid \Gamma, f_c, f_d\rangle$ and, for every individual b that is an R-successor of a, $\mathcal{I}$ satisfies $\langle (a,b) : R \mid \Gamma', -, -\rangle$. In addition, the evaluation π gives the certainty that a is in $\forall R.C$ (denoted $\pi(x_{a:\forall R.C})$) and the certainty that b is an R-successor of a (denoted $\pi(x_{(a,b):R})$). We know by definition that $(\forall R.C)^{\mathcal{I}}(a) = \otimes_{b\in\Delta^{\mathcal{I}}}\{f_d(\sim R^{\mathcal{I}}(a,b), C^{\mathcal{I}}(b))\} = \pi(x_{a:\forall R.C})$. Hence, for every b that is an R-successor of a, there exists some $\alpha \in \mathcal{V}$ such that $R^{\mathcal{I}}(a,b) = \pi(x_{(a,b):R})$ and $C^{\mathcal{I}}(b) = \alpha$. That is, $\mathcal{I}$ satisfies $\langle b : C \mid \alpha, f_c, f_d\rangle$.

■

Lemma 3.3.8 (Completeness) Any clash-free and complete extended ABox $\mathcal{A}_c^{\mathcal{E}}$ has a model.

Proof. Let $\mathcal{A}_c^{\mathcal{E}}$ be a clash-free and complete extended ABox, and $\mathcal{C}$ be the constraints set associated with $\mathcal{A}_c^{\mathcal{E}}$. Since $\mathcal{A}_c^{\mathcal{E}}$ is clash-free and complete, there exists an evaluation $\pi : Var(\mathcal{C}) \rightarrow \mathcal{V}$ that is a solution to the constraints set $\mathcal{C}$, where $Var(\mathcal{C})$ is the set of variables occurring in the constraints set $\mathcal{C}$, and $\mathcal{V}$ is the certainty domain.

We now define a canonical interpretation $\mathcal{I}_{\mathcal{A}}$ of $\mathcal{A}_c^{\mathcal{E}}$ as follows:

- The domain $\Delta^{\mathcal{I}_{\mathcal{A}}}$ of $\mathcal{I}_{\mathcal{A}}$ consists of all the individual names occurring in $\mathcal{A}_c^{\mathcal{E}}$.
- For every atomic concept A in $\mathcal{A}_c^{\mathcal{E}}$, we define

$$A^{\mathcal{I}_{\mathcal{A}}}(a) = \begin{cases} \pi(x_{a:A}) & \text{if } \langle a : A \mid x_{a:A}, -, - \rangle \in \mathcal{A}_c^{\mathcal{E}} \\ \alpha & \text{if } \langle a : A \mid \alpha, -, - \rangle \in \mathcal{A}_c^{\mathcal{E}} \\ l & \text{otherwise, where } l \text{ is the least value in } \mathcal{V} \end{cases}$$

- For all roles R, we define

$$R^{\mathcal{I}_{\mathcal{A}}}(a_1, a_2) = \begin{cases} \pi(x_{(a_1,a_2):R}) & \text{if } \langle (a_1, a_2) : R \mid x_{(a_1,a_2):R}, -, - \rangle \in \mathcal{A}_c^{\mathcal{E}} \\ \alpha & \text{if } \langle (a_1, a_2) : R \mid \alpha, -, - \rangle \in \mathcal{A}_c^{\mathcal{E}} \\ l & \text{otherwise} \end{cases}$$

Next, we show that the pair $\langle \mathcal{I}_{\mathcal{A}}, \pi \rangle$ is a model of $\mathcal{A}_c^{\mathcal{E}}$. That is, $\mathcal{I}_{\mathcal{A}}$ satisfies all the assertions in $\mathcal{A}_c^{\mathcal{E}}$, and π is a solution to the constraints set $\mathcal{C}$.

By definition, $\mathcal{I}_{\mathcal{A}}$ satisfies all the role assertions in $\mathcal{A}_c^{\mathcal{E}}$. We now show that $\mathcal{I}_{\mathcal{A}}$ also satisfies all the concept assertions of the form $\langle a : C \mid \Gamma, f_c, f_d \rangle$ in $\mathcal{A}_c^{\mathcal{E}}$. For this, we

use the induction technique on the structure of the concept C, where Γ is either a certainty value in the certainty domain or the variable $x_{a:C}$ denoting the certainty of the assertion.

Base Case:

If C is an atomic concept, then by definition, $\mathcal{I}_{\mathcal{A}}$ satisfies the concept assertion.

Induction Step:

If $C = C_1 \sqcap C_2$, we have $\langle a : C_1 \sqcap C_2 \mid \Gamma, f_c, f_d \rangle \in \mathcal{A}_c^{\mathcal{E}}$. Since $\mathcal{A}_c^{\mathcal{E}}$ is complete, there is no more rule applicable, and we have $\{\langle a : C_1 \mid x_{a:C_1}, f_c, f_d \rangle, \langle a : C_2 \mid x_{a:C_2}, f_c, f_d \rangle\} \subseteq \mathcal{A}_c^{\mathcal{E}}$ and $(f_c(x_{a:C_1}, x_{a:C_2}) = \Gamma) \in \mathcal{C}$. By the induction hypothesis, we know that $\mathcal{I}_{\mathcal{A}}$ satisfies $\langle a : C_1 \mid x_{a:C_1}, f_c, f_d \rangle$ and $\mathcal{I}_{\mathcal{A}}$ satisfies $\langle a : C_2 \mid x_{a:C_2}, f_c, f_d \rangle$. Also, since π is a solution to the constraints set $\mathcal{C}$, we have $f_c(\pi(x_{a:C_1}), \pi(x_{a:C_2})) = \Gamma$, where the evaluation π gives the certainties to variables $x_{a:C_1}$ and $x_{a:C_2}$. Hence, $f_c(C_1^{\mathcal{I}_{\mathcal{A}}}(a), C_2^{\mathcal{I}_{\mathcal{A}}}(a)) = \Gamma$. Since, according to Table 4, we have $f_c(C_1^{\mathcal{I}}(a), C_2^{\mathcal{I}}(a)) = (C_1 \sqcap C_2)^{\mathcal{I}}(a)$, and since an interpretation $\mathcal{I}$ satisfies $\langle a : C_1 \sqcap C_2 \mid \Gamma, f_c, f_d \rangle$ if $(C_1 \sqcap C_2)^{\mathcal{I}}(a) = \Gamma$, the canonical interpretation $\mathcal{I}_{\mathcal{A}}$ satisfies concept assertions of the form $\langle a : C_1 \sqcap C_2 \mid \Gamma, f_c, f_d \rangle$.

If $C = C_1 \sqcup C_2$, we have $\langle a : C_1 \sqcup C_2 \mid \Gamma, f_c, f_d \rangle \in \mathcal{A}_c^{\mathcal{E}}$. Since $\mathcal{A}_c^{\mathcal{E}}$ is complete, no more rule is applicable, and we have $\{\langle a : C_1 \mid x_{a:C_1}, f_c, f_d \rangle, \langle a : C_2 \mid x_{a:C_2}, f_c, f_d \rangle\} \subseteq \mathcal{A}_c^{\mathcal{E}}$ and $(f_d(x_{a:C_1}, x_{a:C_2}) = \Gamma) \in \mathcal{C}$. By the induction hypothesis, we know that $\mathcal{I}_{\mathcal{A}}$ satisfies $\langle a : C_1 \mid x_{a:C_1}, f_c, f_d \rangle$ as well as $\langle a : C_2 \mid x_{a:C_2}, f_c, f_d \rangle$. Also, since π is a solution to the constraints set $\mathcal{C}$, we have $f_d(\pi(x_{a:C_1}), \pi(x_{a:C_2})) = \Gamma$, and hence, $f_d(C_1^{\mathcal{I}_{\mathcal{A}}}(a), C_2^{\mathcal{I}_{\mathcal{A}}}(a)) = \Gamma$. Since $f_d(C_1^{\mathcal{I}}(a), C_2^{\mathcal{I}}(a)) = (C_1 \sqcup C_2)^{\mathcal{I}}(a)$ according to Table 4, and since an interpretation $\mathcal{I}$ satisfies $\langle a : C_1 \sqcup C_2 \mid \Gamma, f_c, f_d \rangle$ if $(C_1 \sqcup C_2)^{\mathcal{I}}(a) = \Gamma$,

the canonical interpretation $\mathcal{I}_{\mathcal{A}}$ satisfies concept assertions of the form $\langle a : C_1 \sqcup C_2 \mid \Gamma, f_c, f_d \rangle$. Note that the proof presented here is much simpler than that of standard $\mathcal{ALC}$. This is because the disjunction rule in $\mathcal{ALC}$ is nondeterministic, while the disjunction rule in $\mathcal{ALC}_U$ is deterministic, as explained in Section 3.3.3.

If $C = \neg A$, we have $\langle a : \neg A \mid \Gamma, -, - \rangle \in \mathcal{A}_c^{\mathcal{E}}$. Since $\mathcal{A}_c^{\mathcal{E}}$ is complete, no more rule is applicable, and we have $\langle a : A \mid\sim \Gamma, -, - \rangle \in \mathcal{A}_c^{\mathcal{E}}$ and $\{(x_{a:A} = \sim\Gamma), (x_{a:\neg A} = \Gamma)\} \subseteq \mathcal{C}$. Since π is a solution to the constraints set $\mathcal{C}$, we have $\pi(x_{a:\neg A}) = \Gamma$, where π gives the evaluation to the variable $x_{a:\neg A}$. Hence, $\sim A^{\mathcal{I}_{\mathcal{A}}}(a) = \Gamma$. Since $\sim A^{\mathcal{I}}(a) = (\neg A)^{\mathcal{I}}(a)$ according to Table 4, and since an interpretation $\mathcal{I}$ satisfies $\langle a : \neg A \mid \Gamma, -, - \rangle$ if $(\neg A)^{\mathcal{I}}(a) = \Gamma$, the canonical interpretation $\mathcal{I}_{\mathcal{A}}$ satisfies concept assertions of the form $\langle a : \neg A \mid \Gamma, -, - \rangle$.

If $C = \exists R.C_1$, we have $\langle a : \exists R.C_1 \mid \Gamma, f_c, f_d \rangle \in \mathcal{A}_c^{\mathcal{E}}$. Since $\mathcal{A}_c^{\mathcal{E}}$ is complete, no more rule is applicable. The application of Role Exists Restriction rule either: (i) generated a new individual b, added assertions $\{\langle (a,b) : R \mid x_{(a,b):R}, -, - \rangle, \langle b : C_1 \mid x_{b:C_1}, f_c, f_d \rangle\}$ to the extended ABox $\mathcal{A}_c^{\mathcal{E}}$, and added the constraint $(f_c(x_{(a,b):R}, x_{b:C_1}) = x_{a:\exists R.C_1})$ to the constraints set $\mathcal{C}$, or (ii) did not generate a new individual because there already existed an individual b such that $\{\langle (a,b) : R \mid x_{(a,b):R}, -, - \rangle, \langle b : C_1 \mid x_{b:C_1}, f_c, f_d \rangle\} \subseteq \mathcal{A}_c^{\mathcal{E}}$, and the constraint $(f_c\ (x_{(a,b):R}, x_{b:C_1}) = x_{a:\exists R.C_1})$ was already in the constraints set $\mathcal{C}$, or (iii) did not generate new individual because a is blocked by some ancestor b with $\mathcal{A}_i^{\mathcal{E}}(a) \subseteq \mathcal{A}_i^{\mathcal{E}}(b)$; in such case, we could construct the model by having $\langle (a,b) : R \mid x_{(a,b):R}, -, - \rangle$ and $\langle b : C_1 \mid x_{b:C_1}, f_c, f_d \rangle$. In all the three cases, there exists at least one individual b such that $\langle (a,b) : R \mid x_{(a,b):R}, -, - \rangle$, $\langle b : C_1 \mid x_{b:C_1}, f_c, f_d \rangle$, and

$f_c(x_{(a,b):R}, x_{b:C_1}) = x_{a:\exists R.C_1}$. By the induction hypothesis, we know that for each individual b such that (a, b) is in R and b is in C_1, $\mathcal{I}_\mathcal{A}$ satisfies $\langle (a, b) : R \mid x_{(a,b):R}, -, - \rangle$ and $\langle b : C_1 \mid x_{b:C_1}, f_c, f_d \rangle$. Also, since π is a solution to constraints set $\mathcal{C}$, we have $\oplus_{b \in \Delta^\mathcal{I}} \{ f_c(\pi(x_{(a,b):R}), \pi(x_{b:C_1})) \} = \pi(x_{a:\exists R.C_1})$. Hence, $\oplus_{b \in \Delta^{\mathcal{I}_\mathcal{A}}} \{ f_c(R^{\mathcal{I}_\mathcal{A}}(a, b), {C_1}^{\mathcal{I}_\mathcal{A}}(b)) \}$ $= (\exists R.C_1)^{\mathcal{I}_\mathcal{A}}(a)$. That is, $\mathcal{I}_\mathcal{A}$ satisfies concept assertions of the form $\langle a : \exists R.C_1 \mid \Gamma, f_c, f_d \rangle$.

If $C = \forall R.C_1$, we have $\langle a : \forall R.C_1 \mid \Gamma, f_c, f_d \rangle \in \mathcal{A}_c^\mathcal{E}$. Since $\mathcal{A}_c^\mathcal{E}$ is complete, no more rule is applicable, and for every individual b such that $\langle (a, b) : R \mid x_{(a,b):R}, -, - \rangle \in \mathcal{A}_c^\mathcal{E}$, we have $\langle b : C_1 \mid x_{b:C_1}, f_c, f_d \rangle \in \mathcal{A}_c^\mathcal{E}$ and also $(f_d(x_{(a,b):\neg R}, x_{b:C_1}) = x_{a:\forall R.C_1}) \in \mathcal{C}$. By the induction hypothesis, we know that for each individual b such that (a, b) is in R and b is in C_1, $\mathcal{I}_\mathcal{A}$ satisfies $\langle (a, b) : R \mid x_{(a,b):R}, -, - \rangle$ and $\mathcal{I}_\mathcal{A}$ satisfies $\langle b : C_1 \mid x_{b:C_1}, f_c, f_d \rangle$. Also, since π is a solution to the constraints set $\mathcal{C}$, we have $\otimes_{b \in \Delta^\mathcal{I}} \{ f_d(\pi(x_{(a,b):\neg R}), \pi(x_{b:C_1})) \} = \pi(x_{a:\forall R.C_1})$. Hence, $\otimes_{b \in \Delta^{\mathcal{I}_\mathcal{A}}} \{ f_d(\sim R^{\mathcal{I}_\mathcal{A}}(a, b), {C_1}^{\mathcal{I}_\mathcal{A}}(b)) \} = (\forall R.\ C_1)^{\mathcal{I}_\mathcal{A}}(a)$. That is, $\mathcal{I}_\mathcal{A}$ satisfies concept assertions of the form $\langle a : \forall R.C_1 \mid \Gamma, f_c, f_d \rangle$. ∎

Lemma 3.3.9 If an extended ABox $\mathcal{A}_c^\mathcal{E}$ contains a clash, or if the constraints set $\mathcal{C}$ associated with $\mathcal{A}_c^\mathcal{E}$ is unsolvable, then $\mathcal{A}_c^\mathcal{E}$ does not have a model.

Proof. If an extended ABox $\mathcal{A}_c^\mathcal{E}$ contains a clash, then no interpretation can satisfy $\mathcal{A}_c^\mathcal{E}$. Thus, $\mathcal{A}_c^\mathcal{E}$ is inconsistent and has no model. Similarly, if the constraints set $\mathcal{C}$ associated with $\mathcal{A}_c^\mathcal{E}$ is unsolvable, there does not exist an evaluation $\pi : Var(\mathcal{C}) \rightarrow \mathcal{V}$ that is a solution to the constraints set $\mathcal{C}$, where $Var(\mathcal{C})$ is the set of variables occurring

in the constraints set $\mathcal{C}$, and $\mathcal{V}$ is the certainty domain. Hence, $\mathcal{A}_c^{\mathcal{E}}$ is not satisfied and has no model. ■

Before proving the termination property, we need to introduce the term "concept subsets."

Definition 3.3.10 (Concept Subsets) Let C be a concept. The subsets of C, denoted $subset(C)$, is recursively defined as follows.

$subset(A) = \{A\}$, where A is an atomic concept

$subset(C_1 \sqcap C_2) = \{C_1 \sqcap C_2\} \cup subset(C_1) \cup subset(C_2)$

$subset(C_1 \sqcup C_2) = \{C_1 \sqcup C_2\} \cup subset(C_1) \cup subset(C_2)$

$subset(\exists R.C_1) = \{\exists R.C_1\} \cup subset(C_1)$

$subset(\forall R.C_1) = \{\forall R.C_1\} \cup subset(C_1)$

Lemma 3.3.11 (Termination) Let X be any assertion in the extended ABox $\mathcal{A}^{\mathcal{E}}$. The application of completion rules to X terminates.

Proof. Let X be of the form $\langle a : C \mid \alpha, f_c, f_d \rangle$, and $s = |subset(C)|$. As in the standard DL [HST99], termination is a result of the following properties of the completion rules:

1. The completion rules are designed to avoid duplicated rule applications.

2. The completion rules never remove any assertion from the extended ABox nor change/remove any concept in the assertion.

3. Successors are only generated by Role Exists Restriction Rule, and each application of such rule generates at most one successor. Since there cannot be more than s Role Exists Restrictions, the out-degree of the tree is bounded by s.

4. Each node label contains non-empty subsets of $subset(C)$. Hence, if there is a path of length at least 2^s, there must be two nodes along the path that have the same node label, and hence blocking occurs. Since the path cannot grow longer once a blocking takes place, the length of the path is at most 2^s.

■

3.4 Illustrative Example

To illustrate the $\mathcal{ALC}_U$ tableau algorithm and the need for blocking, let us consider a cyclic fuzzy knowledge base $\Sigma = \langle \mathcal{T}, \mathcal{A} \rangle$, where:

$$\mathcal{T} = \{\langle ObesePerson \sqsubseteq \exists hasParent.ObesePerson \mid [0.7, 1], min, max \rangle\}$$

$$\mathcal{A} = \{\langle John : ObesePerson \mid [0.8, 1], -, - \rangle\}$$

Note that the fuzzy knowledge bases can be expressed in $\mathcal{ALC}_U$ by setting the certainty lattice as $\mathcal{L} = \langle \mathcal{V}, \preceq \rangle$, where $\mathcal{V} = \mathcal{C}[0, 1]$ is the set of closed subintervals $[\alpha, \beta]$ in [0, 1] such that $\alpha \preceq \beta$. We also set the meet operator in the lattice as *inf* (infimum), the join operator as *sup* (supremum), and the negation operator as $\sim(x) = t - x$, where $t = [1, 1]$ is the greatest value in the certainty lattice. Finally, the conjunction function is set to *min*, and the disjunction function is set to *max*.

To find out if Σ is consistent, we first apply the pre-processing steps. For this, we transform the axiom into its normal form:

$$\mathcal{T} = \{\langle \top \sqsubseteq (\neg ObesePerson \sqcup \exists hasParent.ObesePerson \mid [0.7, 1], min, max\rangle\}$$

We then augment the ABox with respect to the TBox. That is, for each individual a in the ABox (in this case, we have only $John$) and for each axiom of the form $\langle \top \sqsubseteq \neg C \sqcup D \mid \alpha, f_c, f_d\rangle$ in the TBox, we add an assertion $\langle a : \neg C \sqcup D \mid \alpha, f_c, f_d\rangle$ to the ABox. Hence, in this step, we add the following assertion to the ABox:

$$\langle John : (\neg ObesePerson \sqcup \exists hasParent.ObesePerson \mid [0.7, 1], min, max\rangle\}$$

Now, we can initialize the extended ABox to be:

$$\mathcal{A}_0^{\mathcal{E}} = \{\langle John : ObesePerson \mid [0.8, 1], -, -\rangle,$$

$$\langle John : (\neg ObesePerson \sqcup \exists hasParent.ObesePerson \mid [0.7, 1], min, max\rangle\}$$

and the constraints set to be $\mathcal{C}_0 = \{\}$.

Once the pre-processing phase is over, we are ready to apply the completion rules. The first assertion is $\langle John : ObesePerson \mid [0.8, 1], -, -\rangle$. Since $ObesePerson$ is an atomic concept, we apply the Concept Assertion Rule, which yields:

$$\mathcal{C}_1 = \mathcal{C}_0 \cup \{(x_{John:ObesePerson} = [0.8, 1])\}$$

$\mathcal{C}_2 = \mathcal{C}_1 \cup \{(x_{John:\neg ObesePerson} = t - x_{John:ObesePerson})\}$, where t is the greatest element in the lattice, $[1, 1]$.

The other assertion in $\mathcal{A}_0^{\mathcal{E}}$ is $\langle John : (\neg ObesePerson \sqcup \exists hasParent.ObesePerson \mid [0.7, 1], min, max\rangle\}$. Since this assertion includes a concept disjunction, the Disjunction Rule applies. This yields:

$$\mathcal{A}_1^{\mathcal{E}} = \mathcal{A}_0^{\mathcal{E}} \cup \{\langle John : \neg ObesePerson \mid x_{John:\neg ObesePerson}, -, -\rangle\}$$

$$\mathcal{A}_2^{\mathcal{E}} = \mathcal{A}_1^{\mathcal{E}} \cup \{\langle John : \exists hasParent.ObesePerson \mid x_{John:\exists hasParent.ObesePerson}, min, max\rangle\}$$

$$\mathcal{C}_3 = \mathcal{C}_2 \cup \{(max(x_{John:\neg ObesePerson}, x_{John:\exists hasParent.ObesePerson}) = [0.7, 1])\}$$

The assertion $\langle John : \neg ObesePerson \mid x_{John:\neg ObesePerson}, -, -\rangle$ in $\mathcal{A}_1^{\mathcal{E}}$ triggers the Negation Rule, which yields:

$$\mathcal{A}_3^{\mathcal{E}} = \mathcal{A}_2^{\mathcal{E}} \cup \{\langle John : ObesePerson \mid x_{John:ObesePerson}, -, -\rangle\}$$

The application of the Concept Assertion Rule to the assertion $\langle John : Obese\ Person \mid x_{John:ObesePerson}, -, -\rangle$ in $\mathcal{A}_3^{\mathcal{E}}$ does not derive any new assertion nor constraint. Next, we apply the Role Exists Restriction Rule to the assertion in $\mathcal{A}_2^{\mathcal{E}}$, and obtain:

$$\mathcal{A}_4^{\mathcal{E}} = \mathcal{A}_3^{\mathcal{E}} \cup \{\langle (John, ind1) : hasParent \mid x_{(John,ind1):hasParent}, -, -\rangle\}$$

$$\mathcal{A}_5^{\mathcal{E}} = \mathcal{A}_4^{\mathcal{E}} \cup \{\langle ind1 : ObesePerson \mid x_{ind1:ObesePerson}, -, -\rangle\}$$

$$\mathcal{C}_4 = \mathcal{C}_3 \cup \{(min(x_{(John,ind1):hasParent}, x_{ind1:ObesePerson}) = x_{John:\exists hasParent.ObesePerson})\}$$

$$\mathcal{A}_6^{\mathcal{E}} = \mathcal{A}_5^{\mathcal{E}} \cup \{\langle ind1 : (\neg ObesePerson \sqcup \exists hasParent.ObesePerson \mid [0.7, 1], min, max\rangle\}$$

The application of the Role Assertion Rule to the assertion in $\mathcal{A}_4^{\mathcal{E}}$ yields:

$$\mathcal{C}_5 = \mathcal{C}_4 \cup \{(x_{(John,ind1):\neg hasParent} = t - x_{(John,ind1):hasParent})\}$$

After applying the Concept Assertion Rule to the assertion in $\mathcal{A}_5^{\mathcal{E}}$, we obtain:

$$\mathcal{C}_6 = \mathcal{C}_5 \cup \{(x_{ind1:\neg ObesePerson} = t - x_{ind1:ObesePerson})\}$$

The assertion in $\mathcal{A}_6^{\mathcal{E}}$ triggers the Disjunction Rule, which yields:

$$\mathcal{A}_7^{\mathcal{E}} = \mathcal{A}_6^{\mathcal{E}} \cup \{\langle ind1 : \neg ObesePerson \mid x_{ind1:\neg ObesePerson}, -, -\rangle\}$$

$$\mathcal{A}_8^{\mathcal{E}} = \mathcal{A}_7^{\mathcal{E}} \cup \{\langle ind1 : \exists hasParent.ObesePerson \mid x_{ind1:\exists hasParent.ObesePerson},$$

$$min, max\rangle\}$$

$$\mathcal{C}_7 = \mathcal{C}_6 \cup \{(max(x_{ind1:\neg ObesePerson}, x_{ind1:\exists hasParent.ObesePerson}) = [0.7, 1])\}$$

Next, the application of the Negation Rule to the assertion in $\mathcal{A}_7^{\mathcal{E}}$ yields:

$$\mathcal{A}_9^{\mathcal{E}} = \mathcal{A}_8^{\mathcal{E}} \cup \{\langle ind1 : ObesePerson \mid x_{ind1:ObesePerson}, -, -\rangle\}$$

We then apply the Concept Assertion Rule to the assertion in $\mathcal{A}_9^{\mathcal{E}}$, and obtain:

$$\mathcal{C}_8 = \mathcal{C}_7 \cup \{(x_{ind1:\neg ObesePerson} = t - x_{ind1:ObesePerson})\}$$

The application of the Role Exists Restriction Rule to the assertion in $\mathcal{A}_8^{\mathcal{E}}$ yields:

$$\mathcal{A}_{10}^{\mathcal{E}} = \mathcal{A}_9^{\mathcal{E}} \cup \{\langle (ind1, ind2) : hasParent \mid x_{(ind1,ind2):hasParent}, -, -\rangle\}$$

$$\mathcal{A}_{11}^{\mathcal{E}} = \mathcal{A}_{10}^{\mathcal{E}} \cup \{\langle ind2 : ObesePerson \mid x_{ind2:ObesePerson}, -, -\rangle\}$$

$$\mathcal{C}_9 = \mathcal{C}_8 \cup \{(min(x_{(ind1,ind2):hasParent}, x_{ind2:ObesePerson}) = x_{ind1:\exists hasParent.}$$

$_{ObesePerson})\}$

$$\mathcal{A}^{\mathcal{E}}_{12} = \mathcal{A}^{\mathcal{E}}_{11} \cup \{\langle ind2 : (\neg ObesePerson \sqcup \exists hasParent.ObesePerson \mid [0.7, 1], min,$$

$$max\rangle\}$$

Next, the Role Assertion Rule is applied to the assertion in $\mathcal{A}^{\mathcal{E}}_{10}$ yields:

$$C_{10} = C_9 \cup \{(x_{(ind1,ind2):\neg hasParent} = t - x_{(ind1,ind2):hasParent})\}$$

After applying the Concept Assertion Rule to the assertion in $\mathcal{A}^{\mathcal{E}}_{11}$, we obtain:

$$C_{11} = C_{10} \cup \{(x_{ind2:\neg ObesePerson} = t - x_{ind2:ObesePerson})\}$$

The assertion in $\mathcal{A}^{\mathcal{E}}_{12}$ triggers the Disjunction Rule, which yields:

$$\mathcal{A}^{\mathcal{E}}_{13} = \mathcal{A}^{\mathcal{E}}_{12} \cup \{\langle ind2 : \neg ObesePerson \mid x_{ind2:\neg ObesePerson}, -, -\rangle\}$$

$$\mathcal{A}^{\mathcal{E}}_{14} = \mathcal{A}^{\mathcal{E}}_{13} \cup \{\langle ind2 : \exists hasParent.ObesePerson \mid x_{ind2:\exists hasParent.ObesePerson},$$

$$min, max\rangle\}$$

$$C_{12} = C_{11} \cup \{(max(x_{ind2:\neg ObesePerson}, x_{ind2:\exists hasParent.ObesePerson}) = [0.7, 1])\}$$

Next, the application of the Negation Rule to the assertion in $\mathcal{A}^{\mathcal{E}}_{13}$ yields:

$$\mathcal{A}^{\mathcal{E}}_{15} = \mathcal{A}^{\mathcal{E}}_{14} \cup \{\langle ind2 : ObesePerson \mid x_{ind2:ObesePerson}, -, -\rangle\}$$

We then apply the Concept Assertion Rule to the assertion in $\mathcal{A}^{\mathcal{E}}_{15}$, and obtain:

$$C_{13} = C_{12} \cup \{(x_{ind2:\neg ObesePerson} = t - x_{ind2:ObesePerson})\}$$

Next, consider the assertion in $\mathcal{A}_{14}^{\mathcal{E}}$. Since $ind1$ is an ancestor of $ind2$ and $\mathcal{L}(ind2) \subseteq \mathcal{L}(ind1)$, individual $ind2$ is blocked. Therefore, we will not continue applying the Role Exists Restriction Rule to the assertion in $\mathcal{A}_{14}^{\mathcal{E}}$, and the completion rule application terminates at this point. Note that without blocking, the tableau algorithm would never terminate since new individual will be generated for each application of the Role Exists Restriction Rule.

Since there is no more rule applicable, the set of constraints in $\mathcal{C}_{13}$ is fed into the constraint solver to check its solvability. Since the constraints are solvable, the knowledge base is consistent.

3.5 Related Work

To compare our $\mathcal{ALC}_U$ framework with related works, we first describe how it differs from the parametric framework for deductive database with uncertainty. We then compare it with existing frameworks for DL with uncertainty.

3.5.1 Deductive Database with Uncertainty

The $\mathcal{ALC}_U$ framework was inspired by parametric framework for deductive database with uncertainty (see Section 2.2). It is therefore important to note the similarities and distinctions between the two frameworks. We first note the similarities as follows:

1. The certainty values are assumed to form a certainty lattice.

2. Each axiom/rule and assertion/fact is attached with some parameters. By tuning these parameters, various forms of uncertainty can be handled.

3. A set of reasonable properties for the combination functions were proposed.

Despite the similarities, there are some major differences between the two frameworks:

1. The $\mathcal{ALC}_U$ framework extended the Description Logic $\mathcal{ALC}$, which is a decidable fragment of FOL that corresponds to the propositional modal logic $\mathbf{K}_m$ [Sch91]. On the other hand, the parametric framework extended Datalog, which is closely related to the first-order Horn clause logic [Mak02]. Although both languages are subsets of FOL, there are some important distinctions:

   - Datalog uses the closed world assumption (CWA), which assumes that the information that is not available in the knowledge base is false. On the other hand, DLs use the open world assumption (OWA), which leaves unstated information open [PH06]. For example, suppose the only fact in the knowledge base is that John knows Mary. If we ask whether John knows David in Datalog, the answer would be false. However, in DLs, we do not know whether this holds or not.

   - DLs model the domain as a set of objects and relationships between them. There can be many models (or interpretations), each describes one possible state of the domain. Datalog also assumes models the domain as objects and relationships between them. However, due to CWA, it assumes that the only objects and relationships that exist in the domain are those that

are explicitly represented in the knowledge base. As a result, there is only one model, and both the data and the model can be thought interchangeably [PH06].

- The inference in DLs boils down to checking whether a certain situation holds in all interpretations that are consistent with the available information [PH06]. For example, if we know from the knowledge base that Mary and John are both students at Concordia University, that they are married to each other, and that only people with opposite gender can get married. Then, we can infer that Concordia University has both male and female students, even though we do not know the gender of Mary or John. This is because in all interpretations where Mary is female, John must be male, and vice versa. On the other hand, the inference in Datalog is much simpler than DLs since the standard Datalog program has only at most one interpretation [PH06]. However, it can model only relatively simple situation in which complete information is stated. For example, the DL example that we just presented cannot be captured using Datalog. It would, for example, be necessary to assert the genders of Mary and John.
- The standard Datalog does not support negation nor allow existentials in the head [GHVD03], but these can be easily expressed in DLs.
- Datalog is unable to assert the existence of individuals whose identity might not be known [GHVD03], such as asserting that all animals have a gender, whether the gender is known or unknown. However, this can be

easily expressed in DLs as $\langle Animal \sqsubseteq \exists hasGender.\top \rangle$.

- The mainstream DLs, such as $\mathcal{SHOIN}$ which is the logic foundation of the ontology language OWL-DL, do not directly support n-ary predicates. This can be easily expressed using Datalog.

- DLs cannot describe classes whose instances are related to an anonymous individual via different properties [GHVD03] (such as expressing individuals who attend local conferences). However, this can be easily expressed in Datalog as:

```
attendLocalConference(x) :- attendConference(x,y),
                            liveIn(x,z), locatedIn(y,l),
                            locatedIn(z,l).
```

- DLs are suitable for representing complex knowledge, whereas Datalog is good at formulating complex queries [CPL97].

2. Although both frameworks attach some parameters to each axiom/rule and assertion/fact, the parameters have different meanings. In the $\mathcal{ALC}_U$ framework, each axiom/assertion has two combination functions – the conjunction and the disjunction functions. The conjunction function is used to interpret the concept conjunction and part of the role exists restriction that appear in the associated axiom/assertion, and the disjunction function is used to interpret the concept disjunction and part of the role value restriction that appear in the associated axiom/assertion. On the other hand, in the parametric framework, each

rule/fact has three combination functions – the conjunction, the propagation, and the disjunction functions. The conjunction function is used to combine certainties of the atoms in the rule body and returns the overall certainty of the rule body; the propagation function combines the certainty of the rule body with that of the rule itself to compute the certainty of the head; the disjunction function combines the certainties associated with different derivations of the same ground atom.

3. In addition to proposing a set of reasonable properties for the combination functions, the $\mathcal{ALC}_U$ framework also proposed desirable properties for other functions, such as those for the negation function, which was not supported in the parametric framework.

3.5.2 Existing Frameworks for Description Logics with Uncertainty

In what follows, we compare each of the components in the $\mathcal{ALC}_U$ framework with that of the existing DL/uncertainty frameworks.

In terms of the description language, recall that the semantics of the $\mathcal{ALC}_U$ language constructors are defined based on a set of properties that the combination functions must satisfy. A similar approach was taken in [PFT+04, Str05a, Str05b]. However, the properties that these frameworks proposed are specific to fuzzy logic.

Note also that, like in our framework, the certainty values in [Str04b] were also assumed to form a certainty lattice.

The $\mathcal{ALC}_U$ knowledge base extension was not based on existing DL/uncertainty frameworks. In fact, it differs from those in the existing frameworks since each axiom/assertion is attached with the combination functions (f_c and f_d) which are used to interpret the axiom/assertion. This allows us to handle various forms of uncertainty within one single framework which is not possible in other frameworks.

The $\mathcal{ALC}_U$ reasoning procedure is a generalization of the ones in [TM98, Str05a, BS07]. Similar to these frameworks, the proposed tableau algorithm generates a set of constraints during the completion rule applications. However, the $\mathcal{ALC}_U$ framework differs from these works in several ways, described as follows.

- The $\mathcal{ALC}_U$ reasoning procedure was designed to deal with various forms of uncertainty knowledge, which are based on different mathematical foundations. On the other hand, others mainly considered one form. For instance, [TM98] supported only fuzzy logic with Zadeh semantics, and [BS07] supported only product t-norm. Although [Str05a] supported both Zadeh and Lukasiewicz semantics, it used two sets of reasoning procedures instead of using one generic reasoning procedure to deal with different semantics.

- General TBoxes are supported in the $\mathcal{ALC}_U$ framework, but not in [TM98, Str05a]. As mentioned in Section 2.1.3, a tableau algorithm that can deal with general TBoxes (i.e., the one that can contain GCIs, concept equations, and

cycles) is much more complicated than the one that cannot.

- The $\mathcal{ALC}_U$ framework is optimized in the generation of the assertions and variable names. For example, the variable that denotes the certainty that individual a is in concept C is always $x_{a:C}$, no matter how many times it is used throughout the reasoning process. This is not the case in [Str05a, BS07], where a new assertion/variable name is always generated for each rule application.

  Consider the extended ABox consisting of the assertions $\langle a : C \sqcap D \mid 0.6, bDiff, bSum\rangle$ and $\langle a : C \sqcup D \mid 0.8, bDiff, bSum\rangle$. That is, both assertions are interpreted using Lukasiewicz logic, with the conjunction function defined as $bDiff(x, y) = max(0, x + y - 1)$, and the disjunction function defined as $bSum(x, y) = min(1, x + y)$.

  With the $\mathcal{ALC}_U$ completion rules, application of the conjunction rule to the first assertion will generate two assertions $\{\langle a : C \mid x_{a:C}, -, -\rangle, \langle a : D \mid x_{a:D}, -, -\rangle\}$ and the constraint $(bDiff(x_{a:C}, x_{a:D}) = 0.6)$. Then, application of the disjunction rule to the second assertion will not generate any new assertion, but will infer the constraint $(bSum(x_{a:C}, x_{a:D}) = 0.8)$. Note that we reuse the generated assertions and variable names as much as possible

  Now, consider the completion rules from [Str05a], application of the conjunction rule to the first assertion will generate two assertions $\{\langle a : C, x_1\rangle, \langle a : D, x_2\rangle\}$, where x_1 and x_2 are two new variables, as well as a set of constraints that capture the semantics of $bDiff$ expressed using variables x_1 and x_2. In addition,

application of the disjunction rule to the second assertion will also generate two new assertions $\{\langle a : C, x_3\rangle, \langle a : D, x_4\rangle\}$, where x_3 and x_4 are two new variables, and a set of constraints that encode the semantics of *bSum* expressed using variables x_3 and x_4. Note that the same assertions are generated twice, and new variable names are created for each rule application.

3.6 Summary and Concluding Remarks

In this chapter, we presented the $\mathcal{ALC}_U$ framework which allows us to represent and reason with various forms of uncertainty knowledge in DLs in a uniformed manner. This is achieved by abstracting away the underlying notion of uncertainty in the description language, the knowledge base, and the reasoning procedure. In particular, we generically represented different certainty values by assuming that they form a certainty lattice. In addition, the semantics of the description language is defined by the combination functions which a set of properties must satisfy to ensure admissibility.

Each axiom and assertion in the $\mathcal{ALC}_U$ knowledge base is associated with a certainty value and a pair of combination functions that are used to interpret the concepts that appear in the axiom/assertion. The advantage of this approach is that, by simply tuning the combination functions, different notions of uncertainty can be modeled.

We also presented a sound, complete, and terminating tableau reasoning procedure for $\mathcal{ALC}_U$, which derives a set of assertions and a set of constraints in the form of linear/nonlinear equations/inequations to encode the semantics of the knowledge

base. Note that there could be infinitely many models for a single $\mathcal{ALC}_U$ knowledge base. Consider, for example, a knowledge base consisting of only one assertion: $\langle Mary : Tall \sqcup Thin \mid 0.8, min, max \rangle$. Then, the only constraint we derive would be $max(x_{Mary:Tall}, x_{Mary:Thin}) = 0.8$. There could be infinitely many possible solutions for this constraint, such as $(x_{Mary:Tall} = 0.8, x_{Mary:Thin} = 0.8)$, $(x_{Mary:Tall} = 0.8, x_{Mary:Thin} = 0.7)$, and so on. As far as checking the consistency of the knowledge base is concerned, all we need from the constraint solver is whether there exists any solution to the given constraints set. Note also that, although the constraint-based approach makes the $\mathcal{ALC}_U$ reasoning process dependent on the constraint solver, it also makes the design of the tableau algorithm elegant, since only one single reasoning procedure is needed to work with different notions of uncertainty.

Chapter 4

Optimizing $\mathcal{ALC}_U$ Reasoning

The $\mathcal{ALC}_U$ reasoning procedure presented in Chapter 3 provides the theoretical basis for developing the core of an $\mathcal{ALC}_U$ system (reasoner). However, for practical purposes, techniques need to be developed to optimize the performance of the $\mathcal{ALC}_U$ system in general, and the $\mathcal{ALC}_U$ reasoning procedure in particular. This chapter first briefly summarizes some of the established runtime optimization techniques known in standard DL systems. Then, optimization techniques developed for the $\mathcal{ALC}_U$ system are presented in detail.

4.1 Optimization Techniques for Standard DL Systems

This section briefly surveys some runtime optimization techniques that are employed in standard DL systems such as FaCT++ [FaC], Pellet [Lab], and RacerPro [KG].

This is by no means a complete list, but it covers the most commonly used techniques.

- **Lexical Normalization:** Lexical normalization is the process of transforming each concept description into a canonical form. The advantage of doing this is that it facilitates another optimization technique, called concept simplification. In addition, it could arrange the sub-concepts in such a way that a clash can be detected early during the tableau expansion [BH91, Hor97b].

- **Concept Simplification:** Concept simplification is the process of removing redundant sub-concepts in a given concept. For example, $(\top \sqcap C)$ can be simplified to C, and $(A \sqcap A)$ can be reduced to A. That is, these sub-concepts are redundant and can be safely removed without any consequences. The advantage of this technique is that it could reduce the number of sub-concepts in a concept description, hence reducing the number of completion rule applications and also enabling the knowledge base be more compactly stored [BH91].

- **Partitioning Based on Connectivity as Individual Groups:** To check consistency of the knowledge base, the ABox can be partitioned into smaller groups based on how individuals are related to each other through role assertions in the initial extended ABox. Each of these individual groups can then be thought of as the initial trees in the forest.

- **Optimized Individual Group Creation:** The process of creating the individual groups we just mentioned is optimized so that no redundant groups are created during the process.

- **Caching:** During the consistency check for which completion rules are applied, there may be many successor nodes created. Some of these nodes can be very similar. Hence, caching satisfiability status of these nodes can save the computation time [Hor97b].

- **Lazy Unfolding:** Lazy unfolding is an optimized unfolding process in which unfolding is done only on demand [BH91].

- **GCI Absorption:** This technique tries to eliminate GCIs in the knowledge base as much as possible by replacing them with primitive definition axioms [Hor97b]. For example, the axioms $\langle A \sqsubseteq C \sqcap D \rangle$ and $\langle A \sqcap B \sqsubseteq E \sqcup F \rangle$ can be transformed into $\langle A \sqsubseteq (C \sqcap D) \sqcap (\neg B \sqcup (E \sqcup F)) \rangle$.

- **Semantic Branching:** Recall that the disjunction rule in standard DLs is nondeterministic by nature, and hence requires search when it is actually implemented. When this rule is applied, semantic branching can be used to avoid repeatedly exploring redundant search spaces [DLL62, GS96].

- **Dependency-Directed Backtracking:** This technique allows fast recovery from bad branching choice by finding the branching points that are responsible for the clash, and jumping back to relevant branching point without exploring any alternatives [SS77, Hor97a]. Note that this technique is designed to deal with the nondeterministic nature of the disjunction rule.

4.2 Optimization Techniques for the $\mathcal{ALC}_U$ System

This section presents some runtime optimization techniques that can be used for the $\mathcal{ALC}_U$ system. These techniques can be divided into three categories:

1. The techniques that were adapted from standard DL systems by taking into account the uncertainty factor. These techniques include lexical normalization, concept simplification, partitioning based on connectivity as Individual Groups, optimized Individual Group creation, and caching (see sections 4.2.1, 4.2.2, 4.2.3, 4.2.4, 4.2.6 respectively). Note that not every optimization technique presented in Section 4.1 can be adapted to the $\mathcal{ALC}_U$ system. In particular, lazy unfolding and GCI absorption are not adaptable since these two techniques take advantage of the unfoldable TBox. However, as mentioned in Note 3.3.6, the notion of unfolding is not applicable to the $\mathcal{ALC}_U$ framework. The other two optimization techniques that are not applicable to the $\mathcal{ALC}_U$ system are semantic branching and dependency-directed backtracking. Both of these techniques are useful to handle the nondeterministic nature of the disjunction rule. However, as we mentioned in Section 3.3.3, the disjunction rule in the $\mathcal{ALC}_U$ framework is deterministic, and hence these optimization techniques are not needed.

2. New optimization techniques that deal with uncertainty reasoning, including partitioning based on connectivity as Assertion Groups, and optimized clash detection (see sections 4.2.3 and 4.2.5).

3. The optimization techniques that are commonly used in software systems, including optimized hashing and optimized string comparison (see sections 4.2.7 and 4.2.8).

In what follows, we explain these optimization techniques in detail.

4.2.1 Lexical Normalization

Lexical normalization is a common optimization technique used in the standard DL systems, and can be directly adopted to the $\mathcal{ALC}_U$ system. With normalization, concepts are transformed into a canonical form. For example, concepts like $(C \sqcap (B \sqcap A))$, $(B \sqcap (C \sqcap A))$, and $((B \sqcap A) \sqcap C)$ can all be transformed into the canonical form $(A \sqcap (B \sqcap C))$.

In the $\mathcal{ALC}_U$ system, lexical normalization is done while the user input is being parsed. Only concept conjunctions (i.e., concepts of the form $C \sqcap D$) and concept disjunctions (i.e., concepts of the form $C \sqcup D$) need to be converted to canonical form, since $\sqcap$ and $\sqcup$ are the only language constructors that combine concepts.

Lexical normalization is realized in the $\mathcal{ALC}_U$ system by sorting the sub-concepts of the concept description:

- In case sub-concepts are of the same type, they are sorted lexicographically. For example, if we have $(\neg B \sqcap \neg A)$, then the concepts $\neg B$ and $\neg A$ will be sorted lexicographically, hence the canonical form is $(\neg A \sqcap \neg B)$. In case of role value restriction ($\forall$) and role exists restriction ($\exists$), the role names are sorted lexicographically if they are different; otherwise, the concept names are sorted

lexicographically. For example, if we have the concept $(\exists R.C \sqcup (\exists S.B \sqcup \exists R.A))$, then its canonical form would be $(\exists R.A \sqcup (\exists R.C \sqcup \exists S.B))$.

- In case sub-concepts are of different type, they are sorted based on the concept types using the order: Top Concept $(\top) \ll$ Bottom Concept $(\bot) \ll$ Atomic Concept $(A) \ll$ Concept Negation $(\neg C) \ll$ Concept Conjunction $(C \sqcap D) \ll$ Concept Disjunction $(C \sqcup D) \ll$ Role Value Restriction $(\forall R.C) \ll$ Role Exists Restriction $(\exists R.C)$. For example, the concept $((\exists R.D) \sqcup (\neg Z))$ would have the canonical form $((\neg Z) \sqcup (\exists R.D))$.

Note that re-ordering of the sub-concepts does not change the semantics of the original concept due to commutativity and associativity properties of the combination functions (see Section 3.1.2 for more details).

The major advantage of lexical normalization is that it allows obvious clashes be detected early. For example, given assertions $\langle Mary : Tall \sqcap Thin \mid [0.8, 1], min, - \rangle$ and $\langle Mary : Thin \sqcap Tall \mid [0, 0.4], min, - \rangle$, the later becomes $\langle Mary : Tall \sqcap Thin \mid [0, 0.4], min, - \rangle$ after the lexical normalization. This allows us to easily notice the conflicting certainty values associated with these two assertions, which would be detectable only later at the constraint solving stage if lexical normalization is not applied. Another advantage of lexical normalization is that it facilitates other optimization methods, namely concept simplification described next.

4.2.2 Concept Simplification

Concept simplification is an optimization technique that removes redundant sub-concepts in a given concept. Let $\leadsto$ denote "can be simplified to". The following simplifications can be applied to the $\mathcal{ALC}_U$ system:

- $\top \sqcap C \leadsto C$
- $\bot \sqcap \cdots \leadsto \bot$
- $\top \sqcup \cdots \leadsto \top$
- $\bot \sqcup C \leadsto C$
- $\forall R.\top \leadsto \top$
- $\exists R.\bot \leadsto \bot$

Note that the above simplifications are valid due to the boundary-condition properties of the combination functions (see Section 3.1.2 for more details).

It is important to note that not all types of simplifications used in the standard DL systems can be applied when uncertainty is present. For example, it is a common practice in the standard DL systems to remove duplicated sub-concepts in a concept conjunction or disjunction, such as simplifying $(A \sqcap A)$ to A. However, this kind of simplification may not always be true in our context, depending on the combination function used. For example, assume that the interpretation of concept A is 0.4. If the conjunction function is *min*, then $(A \sqcap A)$ is A since $min(0.4, 0.4) = 0.4$. However, if

the conjunction function is the algebraic product ($\times$), then $(A \sqcap A)$ is not the same as A, since $\times(0.4, 0.4) = 0.16 \neq 0.4$. Therefore, concept simplification must be applied with care.

The major advantage of the simplification method is that it could potentially reduce the number of sub-concepts in a concept description, hence reducing the number of completion rule applications. In some extreme cases, a complex concept description can be simplified to only $\top$ or $\bot$, hence eliminating the need to apply any completion rule.

4.2.3 Partitioning Based on Connectivity

Since the knowledge bases are usually large in size, it is a common practice to perform inferences on a divide-and-conquer basis to improve the performance. In the $\mathcal{ALC}_U$ system, we partition the knowledge bases into Individual Groups and Assertions Groups based on the notion of connectivity. In what follows, we describe these different bases for partitioning in detail.

Partitioning Based on Connectivity as Individual Groups

Similar to the standard DL systems, the individuals in the ABox are divided into one or more partitions called Individual Groups. Each group consists of individuals that are "related" to each other through role assertions. In general, two individuals a and b are in the same Individual Group if a is an ancestor of b. That is, either there exists a role assertion of the form $\langle (a, b) : R \mid \alpha, -, - \rangle$ in the ABox, or there exists a

chain of role assertions $\langle (a, b_1) : R_1 \mid \alpha_1, -, - \rangle$, $\langle (b_1, b_2) : R_2 \mid \alpha_2, -, - \rangle, \cdots, \langle (b_k, b) : R_{k+1} \mid \alpha_{k+1}, -, - \rangle$ in the ABox, where the role assertion could be either explicitly specified in the user input or derived through the application of Role Exists Restriction Rule.

For example, consider an ABox with the following assertions:

$\langle a : D \sqcap E \mid [0.6, 1], \times, - \rangle$

$\langle a : E \sqcup Q \mid [0.8, 1], -, ind \rangle$

$\langle a : F \mid [0.7, 0.7], -, - \rangle$

$\langle b : \forall R.D \mid [0.6, 1], -, max \rangle$

$\langle (b, c) : R \mid [0.9, 1], -, - \rangle$

$\langle c : G \sqcap H \mid [0.6, 0.8], min, - \rangle$

Since individual a is not related to other individuals in the ABox, it forms a standalone Individual Group. On the other hand, individual b is related to individual c through the role R. Therefore, b and c together form an Individual Group. The result of Individual Group partitioning is shown in Figure 6.

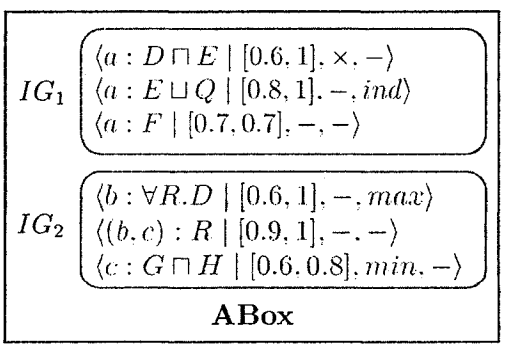

Figure 6: Example of Partitioning Based on Connectivity as Individual Groups

There are some interesting properties about Individual Groups. Let I_U be the set of all individuals in the ABox, I_{IG_i} be all the individuals in the Individual Group IG_i, and IG_1, IG_2, ..., IG_k be all the Individual Groups in the ABox. Then, $I_{IG_1} \cup I_{IG_2} \cup \cdots \cup I_{IG_k} = I_U$ and $I_{IG_i} \cap I_{IG_j} = \emptyset$, for $i \neq j$. In addition, let A_U be the set of all assertions in the ABox, and A_{IG_i} be all the assertions associated with individuals in the Individual Group IG_i. Then, $A_{IG_1} \cup A_{IG_2} \cup \cdots \cup A_{IG_k} = A_U$ and $A_{IG_i} \cap A_{IG_i} = \emptyset$, for $i \neq j$.

By partitioning the ABox this way, inferences can be performed independently for each Individual Group. Once no more completion rule can be applied to a given Individual Group, we could pass the derived assertions and their corresponding constraints to the constraint solver to build the model, and we can be sure that the model built will not be changed even if we perform inference on other Individual Groups in the ABox.

This has several advantages. First, the consistency of the ABox can now be checked incrementally. At any given time, the consistency of one single Individual Group is checked. If the current Individual Group is found to be consistent, we continue checking the remaining Individual Groups. On the other hand, if the current Individual Group is found to be inconsistent, we could stop checking the remaining groups, since this implies that the whole ABox is inconsistent. Therefore, in general, this reduces the reasoning complexity when the knowledge base includes many individuals which could be partitioned as described. To further improve the performance, we could also sort the Individual Groups based on some pre-defined criteria,

such as the initial Individual Group size. Hence, the smaller Individual Groups will be checked for consistency before the larger groups.

Another advantage is that, since the number of constraints and the variables used in the constraints in one single Individual Group is, in general, no more than those in a complete ABox, solving small sets of constraints would be faster than solving one large constraints set. That is, let C_U be all the constraints associated with the ABox, let C_{IG_i} be the set of constraints associated with the Individual Group IG_i, and IG_1, IG_2, ..., IG_k be all the Individual Groups in the ABox. We also use $|C|$ to denote the number of constraints in C, and $v(C)$ to denote the cost (or the speed) of solving the constraints set C. Then, although $|C_{IG_1}| + |C_{IG_2}| + \cdots + |C_{IG_k}| = |C_U|$, we expect in general that $v(C_{IG_1}) + v(C_{IG_2}) + \cdots + v(C_{IG_k}) \leq v(C_U)$.

Partitioning Based on Connectivity as Assertion Groups

As mentioned in Section 3.3, the reasoning procedure for the $\mathcal{ALC}_U$ framework generates a set of linear/nonlinear constraints in the completion rule application. Because the number of constraints is usually large, it is important to optimize the constraint solving process by partitioning constraints into independent subsets. For this, Individual Groups are partitioned into Assertion Groups (see Figure 7). Note that each Assertion Group belongs to only one Individual Group, and each Individual Group can consist of many (and at least one) Assertion Groups.

In general, each Assertion Group consists of assertions with inferred constraints depending on each other. For instance, the first Individual Group in the previous

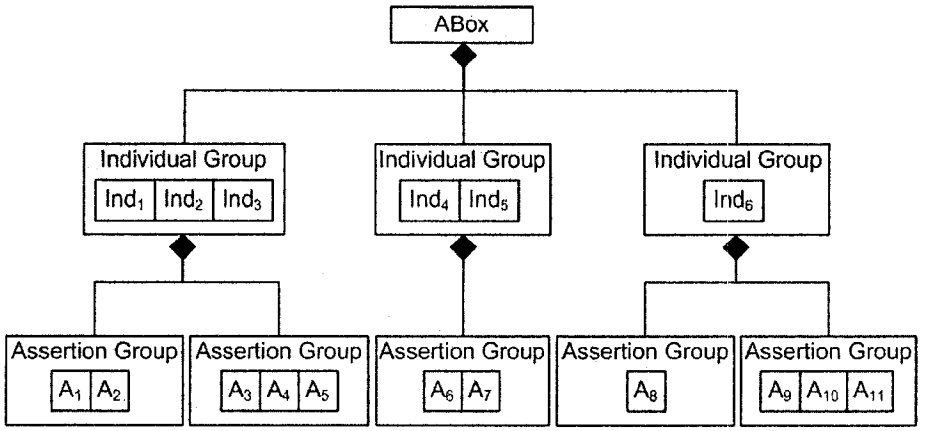

Figure 7: Relationship between ABox, Individual Groups, and Assertion Groups

example contains the following assertions:

$\langle a : D \sqcap E \mid [0.6, 1], \times, - \rangle$

$\langle a : E \sqcup Q \mid [0.8, 1], -, ind \rangle$

$\langle a : F \mid [0.7, 0.7], -, - \rangle$

When the completion rule is applied to the first assertion $\langle a : D \sqcap E \mid [0.6, 1], \times, - \rangle$, we know that individual a is in D to some degree and a is in E to some degree. That is, we infer the following assertions:

$\langle a : D \mid x_{a:D}, -, - \rangle$

$\langle a : E \mid x_{a:E}, -, - \rangle$

Also, the semantics of the concept conjunction in the first assertion is given by the algebraic product, and is encoded as the constraint:

$(x_{a:D} \times x_{a:E} \geq 0.6)$

Similarly, for the second assertion $\langle a : E \sqcup Q \mid [0.8, 1], -, ind \rangle$, we infer the assertions:

$\langle a : E \mid x_{a:E}, -, - \rangle$

$\langle a : Q \mid x_{a:Q}, -, - \rangle$

and the semantics of the concept disjunction is given by the independent function, and is encoded as the constraint:

$(x_{a:E} + x_{a:Q} - (x_{a:E} \times x_{a:Q}) \geq 0.8)$

Because the certainty that a is in E ($x_{a:E}$) appears in both constraints, these two constraints depend on each other, and need to be solved together. So, the first two assertions form an Assertion Group. Finally, with the last assertion $\langle a : F \mid [0.7, 0.7], -, - \rangle$, we infer the constraint:

$(x_{a:F} = 0.7)$

which does not depend on other constraints. So, this assertion forms an Assertion Group on its own. The result of Assertion Group partitioning is shown in Figure 8.

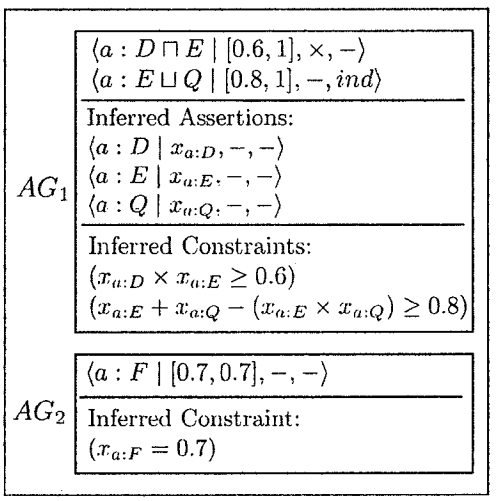

Figure 8: Example of Partitioning Based on Connectivity as Assertion Groups

The Assertion Group(s) are created and merged whenever an assertion A is to be added to the ABox. The decision procedure is illustrated in Figure 9. In this

figure, we use the term *creator Assertion Group* defined as follows: if assertion A is inferred from assertion B, then the Assertion Group to which B belongs is A's *creator Assertion Group*. Note that in case A is a role assertion, the two merge cases in Figure 9 will never happen, since it is impossible for a role assertion to be both inferred and already be present in the ABox. To be more specific, a role assertion A of the form $\langle (a, ind) : R \mid x_{(a,ind):R}, -, - \rangle$ can be inferred from an assertion A_1 only if A_1 contains Role Exists Restriction, and *ind* is a new individual created during the application of Role Exists Restriction rule on A_1. Hence, it is impossible to have a role assertion that is both inferred and already be present in the ABox.

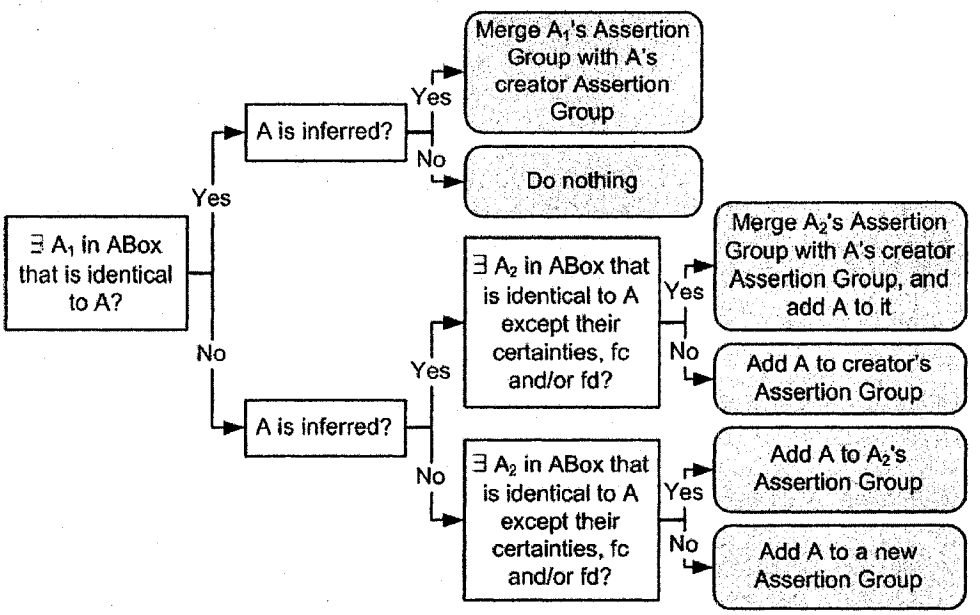

Figure 9: The Creation and Merge of Assertion Group(s)

There are some interesting properties about Assertion Groups. Let A_{IG} be the set of all assertions in a particular Individual Group IG, let A_{AG_i} be all the assertions in the Assertion Group AG_i, and let AG_1, AG_2, ..., AG_k be all the Assertion Groups

in IG. Then, $A_{AG_1} \cup A_{AG_2} \cup \cdots \cup A_{AG_k} = A_{IG}$ and $A_{AG_i} \cap A_{AG_j} = \emptyset$, for $i \neq j$. In addition, let C_{IG} be the set of all constraints that are associated with Individual Group IG, and let C_{AG_i} be all the constraints associated with assertions in the Assertion Group AG_i. Then, $C_{AG_1} \cup C_{AG_2} \cup \cdots \cup C_{AG_k} = C_{IG}$ and $C_{AG_i} \cap C_{AG_j} = \emptyset$, for $i \neq j$. Finally, let V_{IG} be the set of all variables that are used in C_{IG}, and V_{AG_i} be all the variables associated with C_{AG_i}. Then, $V_{AG_1} \cup V_{AG_2} \cup \cdots \cup V_{AG_k} = V_{IG}$ and $V_{AG_i} \cap V_{AG_j} = \emptyset$, for $i \neq j$.

By partitioning the Individual Groups into Assertion Groups, constraints can be solved independently for each Assertion Group. This implies that the model built will not be changed even if we solve constraints in other Assertion Groups in the same Individual Group.

This has several advantages. First, the consistency of the Individual Group can now be checked incrementally by inspecting the consistency of each Assertion Group. At any given time, the constraints in one single Assertion Group are fed into the constraint solver. If the current Assertion Group is found to be consistent, we continue checking the remaining Assertion Groups. On the other hand, if the current Assertion Group is found to be inconsistent, we could stop checking the remaining Assertion Groups, since this implies that the whole Individual Group is inconsistent. Therefore, the consistency checking could be done more effectively by considering the Assertions Groups one by one. In the worst case, all the assertions associated with an Individual Group are in one single Assertion Group. In such case, the performance is the same as performing consistency checking on the whole Individual Group. To further improve

the performance, the Assertion Groups within an Individual Group can be sorted based on some pre-defined criteria, such as the number of constraints plus the number of variables in the Assertion Group. Hence, the smaller Assertion Groups will be checked for consistency before the larger ones are checked.

A related advantage is that, in case an Individual Group is inconsistent, the reasoner will be able to more precisely identify the group of assertions that cause the inconsistency instead of simply returning that the whole Individual Group is inconsistent.

Another advantage is that we are now able to determine the degree to which a particular assertion (say, X) is true by simply solving the constraints in the Assertion Group that X belongs to, without solving those of the whole Individual Group. This is possible since, for each assertion, a link is kept to keep track of which Assertion Group does this particular assertion belongs to. Hence, all we need to do is to solve the constraints in that particular Assertion Group. All the other Assertion Groups can be ignored.

Finally, since the number of constraints (and the variables used in the constraints) in one single Assertion Group is, in general, no more than those of the whole Individual Group, solving a few small constraints sets would be faster than solving one large constraints set.

4.2.4 Optimized Individual Group Creation

The process of creating Individual Groups can be optimized so that no redundant groups will be created. This optimization is particularly useful when we have a deep knowledge base, i.e., when there is a long chain of role assertions in the ABox.

In the $\mathcal{ALC}_U$ system, the process to create Individual Groups is as follows. When a new individual is added to the ABox, it is marked as a standalone individual and its Individual Group is initially set to null. Then, whenever a new role assertion of the form $\langle (a, b) : R \mid \alpha, -, - \rangle$ is added to the knowledge base, we get the Individual Group that individual a belongs to (say, IG_1), and the Individual Group that individual b belongs to (say, IG_2). If both IG_1 and IG_2 are null, it means that both a and b are standalone individuals. Hence, we create a new Individual Group, and add a and b to it. On the other hand, if IG_1 (resp. IG_2) is null and IG_2 (resp. IG_1) is not null, it means that individual a (resp. b) is standalone, but individual b (resp. a) already has an Individual Group. In this case, we simply add a (resp. b) to the Individual Group of b (resp. a). Finally, if both IG_1 and IG_2 are not null, it means both a and b already have Individual Groups. Hence, we need to merge IG_1 and IG_2 into one single Individual Group. Once we finish parsing the user input, we create a new Individual Group for each of the leftover standalone individuals in the ABox.

To illustrate this optimization, assume that we have the following assertions as the input knowledge base:

$\langle a : \exists R.F \mid [0.4, 0.7], min, max \rangle$

$\langle b : E \mid [0.6, 1], -, - \rangle$

$\langle c : G \mid [0.1, 0.9], -, - \rangle$

$\langle (a, b) : R \mid [1, 1], -, - \rangle$

After processing the first assertion $\langle a : \exists R.F \mid [0.4, 0.7], min, max \rangle$, a new individual a is created, and is marked as a standalone individual. Similarly, with the second assertion $\langle b : E \mid [0.6, 1], -, - \rangle$, a new individual b is created, and is marked as a standalone individual. Likewise, after processing the third assertion $\langle c : G \mid [0.1, 0.9], -, -, - \rangle$, a new individual c is created, and is marked as a standalone individual. Finally, the last assertion $\langle (a, b) : R \mid [1, 1], -, - \rangle$ is a role assertion. Since both individuals a and b do not belong to any Individual Group, a new group is created, and both a and b are no longer marked as standalone individuals. At this point, all the assertions are processed, and individual c is the only standalone individual. Hence, we create a new Individual Group for c. Finally, the concept assertion $\langle a : \exists R.F \mid [0.4, 0.7], min, max \rangle$ is the only assertion that is non-atomic in the ABox, hence further processing is required. Based on the Role Exists Restriction Rule, a new individual *ind* is generated, and two assertions are added to the ABox: $\langle (a, ind) : R \mid x_{(a,ind):R}, -, - \rangle$ and $\langle ind : F \mid x_{ind:F}, -, - \rangle$, where x_X is the variable denoting the certainty of assertion X. Since individual a already belongs to an Individual Group, individual *ind* is added to a's group. So, to summarize, there are two Individual Groups in the ABox – individuals a, b, and *ind* form one group, and individual c forms another.

4.2.5 Optimized Clash Detection

The clash is detected as early as possible throughout the reasoning procedure as follows:

- When parsing the input knowledge base, if an obvious contradiction is detected (for instance, if the knowledge base contains both assertions $\langle a : C \mid [0, 0.2], min, max\rangle$ and $\langle a : C \mid [0.6, 1], min, max\rangle$), a clash is detected, and no further processing will be done.

- The non-trivial contradictions can be detected only at the constraint-solving phase. At any time, if an Assertion Group is found to be inconsistent (that is, the constraints set associated with this group is not solvable), then the Individual Group that it belongs to is automatically inconsistent, which in turn implies that the ABox is inconsistent. Therefore, the consistency-checking can be stopped immediately.

4.2.6 Caching

To avoid redundant, repeated work, caching is employed in the $\mathcal{ALC}_U$ system as follows:

- Each assertion and constraint is stored only once. This not only prevents the same assertion being processed multiple times, but also saves the storage space. For example, assume we have the following two assertions in the ABox: $\langle a : C \sqcap D \mid [0.8, 1], min, max\rangle$ and $\langle a : B \sqcup (C \sqcap D) \mid [0.6, 1], min, max\rangle$. The

first one yields the assertions $\{\langle a : C \sqcap D \mid x_{a:C\sqcap D}, min, max\rangle, \langle a : C \mid x_{a:C}, -, -\rangle, \langle a : D \mid x_{a:D}, -, -\rangle\}$ and the constraints $\{(x_{a:C\sqcap D} \geq 0.8), (min\ (x_{a:C}, x_{a:D}) = x_{a:C\sqcap D})\}$. The second assertion will also yield the assertion $\langle a : C \sqcap D \mid x_{a:C\sqcap D}, min, max\rangle$, however, it will not be added to the ABox since it is already there. The only new assertions we derive from the second assertion are $\{\langle a : B \sqcup (C \sqcap D) \mid x_{a:B\sqcup(C\sqcap D)}, min, max\rangle, \langle a : B \mid x_{a:B}, -, -\rangle\}$, and constraints $\{(x_{a:B\sqcup(C\sqcap D)} \geq 0.6), (max(x_{a:B}, x_{a:C\sqcap D}) = x_{a:B\sqcup(C\sqcap D)})\}$, which are added to the ABox.

- A flag is set to indicate whether completion rules have been applied to a given assertion (resp., Individual Group). This prevents the same assertion (resp., Individual Group) being processed multiple times.

- After the constraints in an Assertion Group are solved, the result is cached for later use. As an example, suppose the user decides to first check whether the whole ABox is consistent, and later on, he/she decides to check whether the ABox associated with a particular individual, say a, is consistent. Since the consistency of individual a was already checked when the reasoner was checking for the consistency of the whole ABox, the reasoner will be able to return to the user the cached result right away.

4.2.7 Optimized Hashing

To allow fast access, a hash table is a commonly used data structure in many software systems, including standard DL systems. In the $\mathcal{ALC}_U$ system, all the assertions, constraints, and variables are stored as hash sets, which offer constant time performance for basic operations (including insertion of element, deletion of element, and check if an element is a member of the set), given that the hash function distributes the elements uniformly among the buckets [Sunb].

To minimize collisions, the hash code values are calculated based on the special properties of the object of interest. For example, the hash code of a concept assertion is calculated based on the hash codes of the various components that form a concept assertion, namely the individual, the concept, the conjunction function, and the disjunction function. On the other hand, the hash code of a role assertion is calculated based on the hash codes of the two related individuals as well as the hash code of the role name.

4.2.8 Optimized String Comparison

String comparison is a commonly used operation in the $\mathcal{ALC}_U$ system, for instance, to check if two concept names are equal or if two assertions have clash. Since string comparison is expensive, it is important to optimize this operation. For this, each string used in the knowledge base is assigned a canonical reference. This allows us to compare strings via simple reference comparison instead of a full string comparison.

4.3 Summary and Concluding Remarks

In this chapter, we first surveyed the well know optimization techniques used in the standard DL systems. Note that, to the best of our knowledge, there is no work reported on optimizing DL systems with uncertainty. Therefore, no survey was presented in this regard.

We observed that not all optimization techniques used in the standard DL systems can be adapted to the $\mathcal{ALC}_U$ system. This is because some of these techniques take advantage of the unfoldable TBox which is not applicable to the $\mathcal{ALC}_U$ framework, while others are specific in dealing with the nondeterministic nature of the disjunction rule in standard DL reasoning procedures which is not needed in our context because the $\mathcal{ALC}_U$ reasoning procedure is deterministic.

Nevertheless, we were able to apply many optimization techniques from the standard DL systems and other software systems to the $\mathcal{ALC}_U$ system, including lexical normalization, concept simplification, partitioning based on connectivity as Individual Groups, optimized Individual Group creation, caching, optimized hashing, and optimized string comparison. However, we noted that concept simplification needs to be applied with care, because not all types of simplifications used in the standard DL systems can be applied when uncertainty is present.

We also proposed new optimization techniques that specifically deal with the constraint-based nature of the $\mathcal{ALC}_U$ reasoning procedure. In particular, assertions

and constraints associated with each Individual Group are partitioned into independent sets called Assertion Groups so that constraints in any Assertion Group can be solved independently. This enables incremental checking of the knowledge base consistency, helps in speeding-up the constraint-solving process, and allows clash to be detected as early as possible. Note that, unlike in the standard DL systems where the clash is always detected by applying the completion rules, the $\mathcal{ALC}_U$ completion rules can detect only trivial clashes. The non-trivial clashes can only be detected at the constraint-solving phase. This is why partitioning the constraints into smaller groups helps in optimizing the clash detection.

Chapter 5

URDL - A Prototype System

This chapter presents URDL – an Uncertainty Reasoner for the DL $\mathcal{ALC}_U$. URDL shows the practical merits of the $\mathcal{ALC}_U$ framework. It also serves as a test bed for the optimization techniques described in Chapter 4. To ensure portability, URDL was implemented in Java. Figure 10 gives an overview of the URDL architecture.

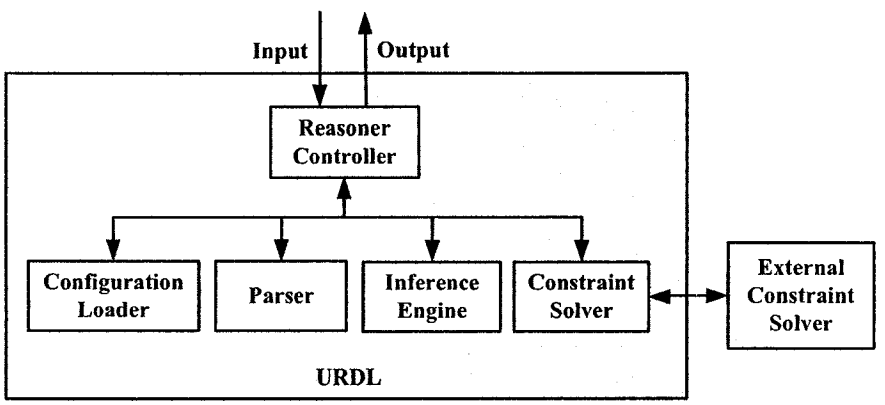

Figure 10: URDL Architecture

As shown in the figure, URDL consists of five main components:

1. **Reasoner Controller** is responsible for the overall program flow. It gets the input from the user, and delegates tasks to Configuration Loader, Parser, Inference Engine, and Constraint Solver in order to fulfill the user query. It then returns the query results back to the user.

2. **Configuration Loader** loads and stores user preferences from the configuration file.

3. **Parser** parses the user input, and stores the parsed axioms, assertions, and combination functions into the knowledge base.

4. **Inference Engine** applies the $\mathcal{ALC}_U$ reasoning procedure to the knowledge base and stores the inferred assertions and constraints into the knowledge base.

5. **Constraint Solver** solves the set of constraints generated by the Inference Engine by calling an external constraint solver. It then returns the result back to the Reasoner Controller.

In what follows, we first describe each of these components in detail in Sections 5.1, 5.2, 5.3, 5.4, and 5.5. We then demonstrate URDL using an example in Section 5.6.

5.1 Reasoner Controller

The Reasoner Controller, which acts as the brain of URDL, is responsible for the overall program flow.

Figure 11 shows how the Reasoner Controller interacts with the user and the other four URDL components in a time sequence. As shown in this figure, the Reasoner Controller first gets the knowledge base and the configuration file from the user. It then calls the Configuration Loader to load the user preferences from the configuration file, and calls the Parser to parse the knowledge base. Once parsing is over, the Reasoner Controller augments the ABox with respect to the TBox (see Figure 43 in Appendix A) and creates Individual Groups for the standalone individuals as described in Section 4.2.4. The Reasoner Controller then gets the user query and calls the Parser to parse the query. Depends on the type of user query, appropriate reasoning must be performed by calling the Inference Engine, and the derived constraints must be solved by calling the Constraint Solver. At the end, the query result is returned back to the user.

The processes of inferencing and solving constraints depend on the type of user query. The main inference services supported by URDL include: the consistency checking which checks the consistency of the ABox with respect to a TBox, the entailment checking which, given a knowledge base, determines to what degree an assertion is true, and the subsumption checking which determines the degree to which concept C is subsumed by concept D with respect to the TBox. In what follows, we explain each of these reasoning services in detail.

Consistency Checking

There are two types of consistency checking supported by URDL. One checks the

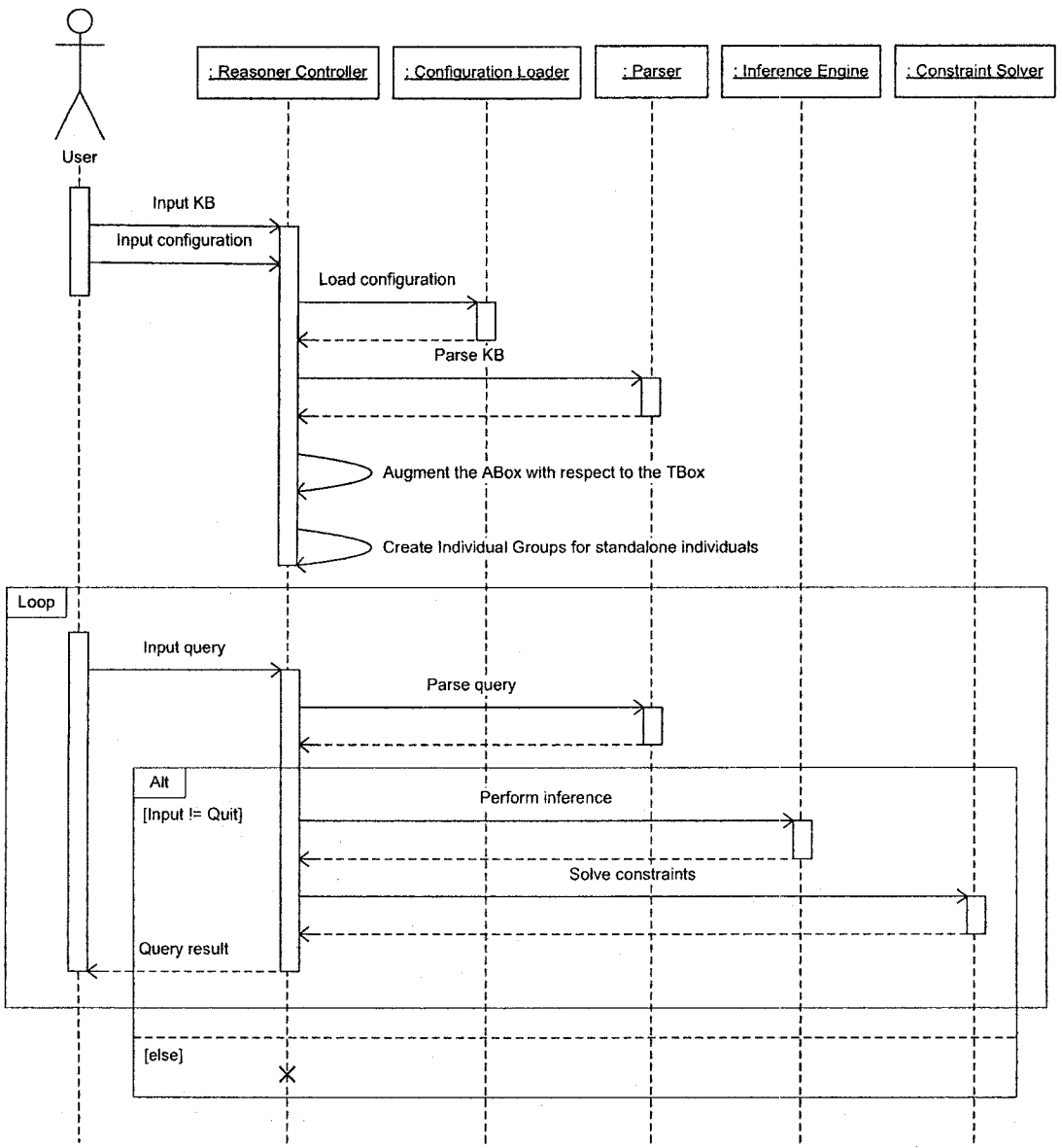

Figure 11: General sequence diagram for URDL

consistency of the entire ABox with respect to the TBox, and the other one checks only the ABox associated with a particular individual (also with respect to the TBox).

If the consistency of the entire ABox is checked, we need to iterate through all the Individual Groups in the ABox. As soon as any of the Individual Groups is found to be inconsistent, the entire ABox is inconsistent. Otherwise, the ABox is consistent with respect to the TBox. Figure 12 shows how the Reasoner Controller, the Inference Engine, and the Constraint Solver interact with each other in a time sequence.

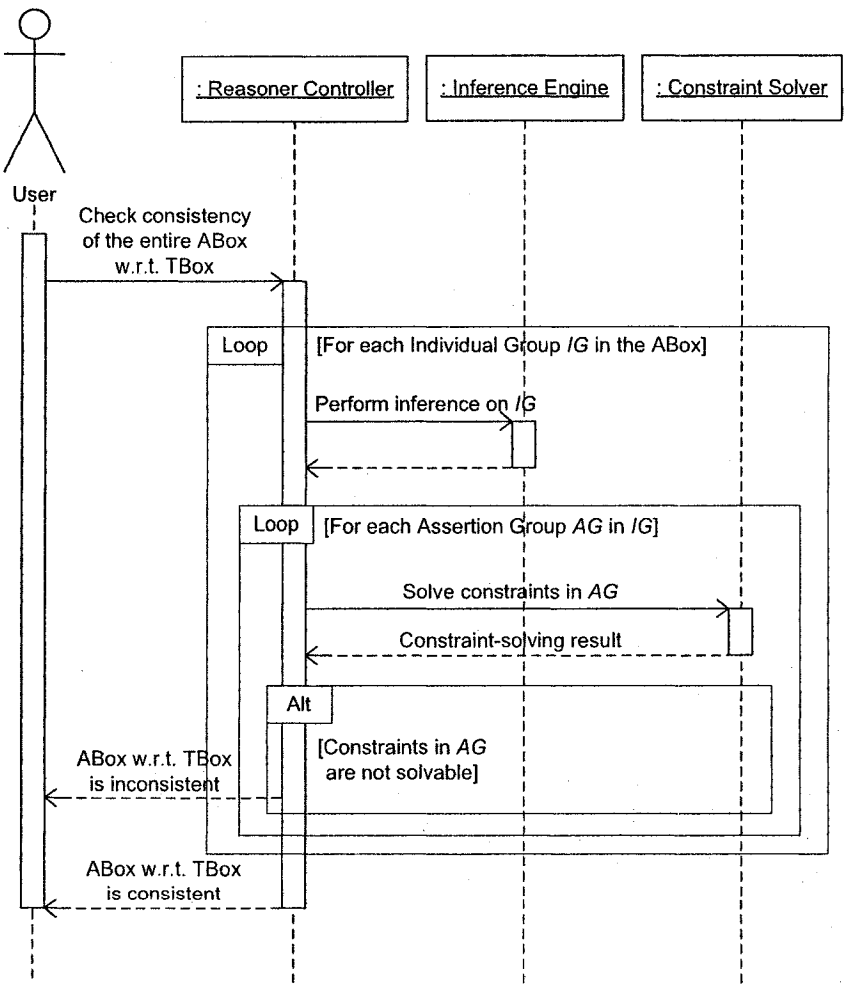

Figure 12: Sequence diagram for consistency checking of ABox with respect to TBox

On the other hand, if only the ABox associated with a particular individual, say *ind*, is to be checked, we need to first get the Individual Group that *ind* belongs to, perform inference on this Individual Group, and solve the constraints associated with the Assertion Groups that are related to *ind*. If any of these constraints sets is not solvable, the ABox associated with *ind* is not consistent. Otherwise, the ABox associated with *ind* is consistent. The sequence diagram illustrating these interactions can be found in Figure 13.

Entailment Checking

Recall that the entailment checking determines the degree to which an assertion A of the form $\langle a : C \mid x_{a:C}, f_c, f_d \rangle$ is true, given an $\mathcal{ALC}_U$ knowledge base. For this, let A' be an assertion of the form $\langle a : \neg C \mid x_{a:\neg C}, f_c, f_d \rangle$. If individual a is already in the ABox, we apply completion rules to the Individual Group that a belongs to, if this group has not been inferred in the past. Then, if A' is not present in the ABox, we temporarily add it and apply the completion rules to its Individual Group. After that, we get the Assertion Group that A' belongs to, and solve the constraints associated with it. Finally, the entailment degree is returned. The sequence diagram for entailment checking is shown in Figure 14.

Subsumption Checking

The subsumption checking determines the degree to which a concept C is subsumed by concept D with respect to a TBox. For this, let $\langle C \sqsubseteq D \mid x_{C \sqsubseteq D}, f_c, f_d \rangle$ be the subsumption relationship to be checked, and let A be an assertion of the form

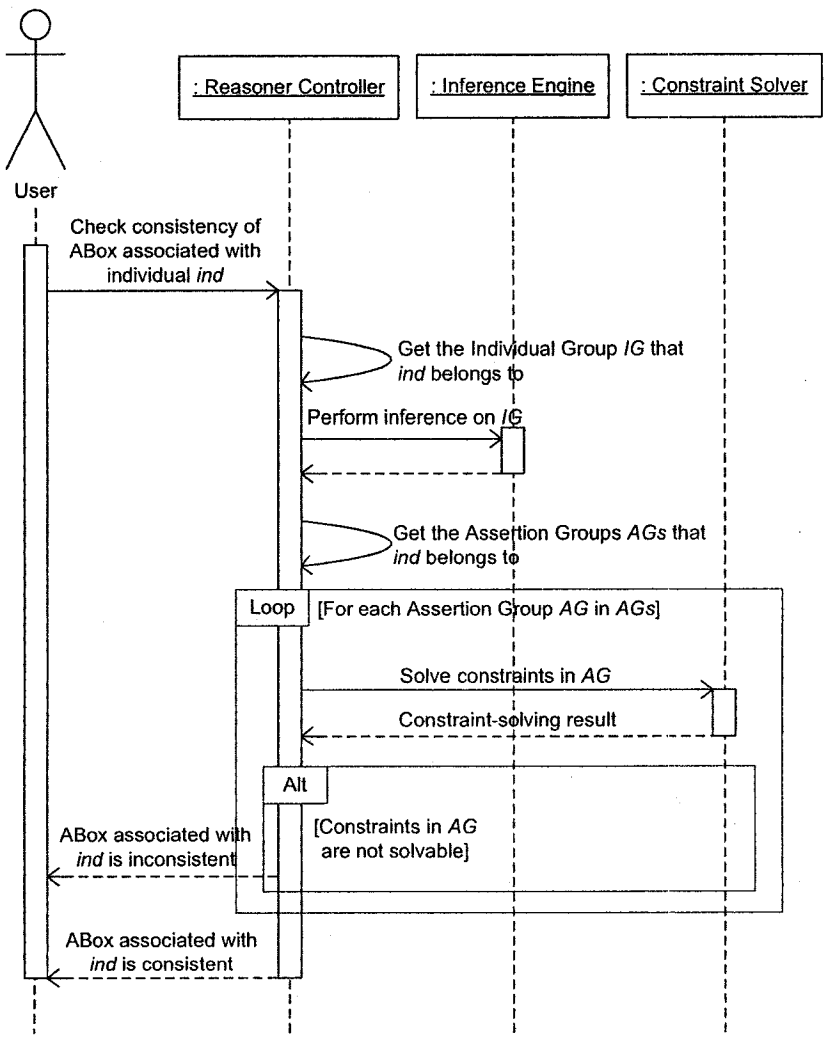

Figure 13: Sequence diagram for consistency checking of ABox associated with an individual

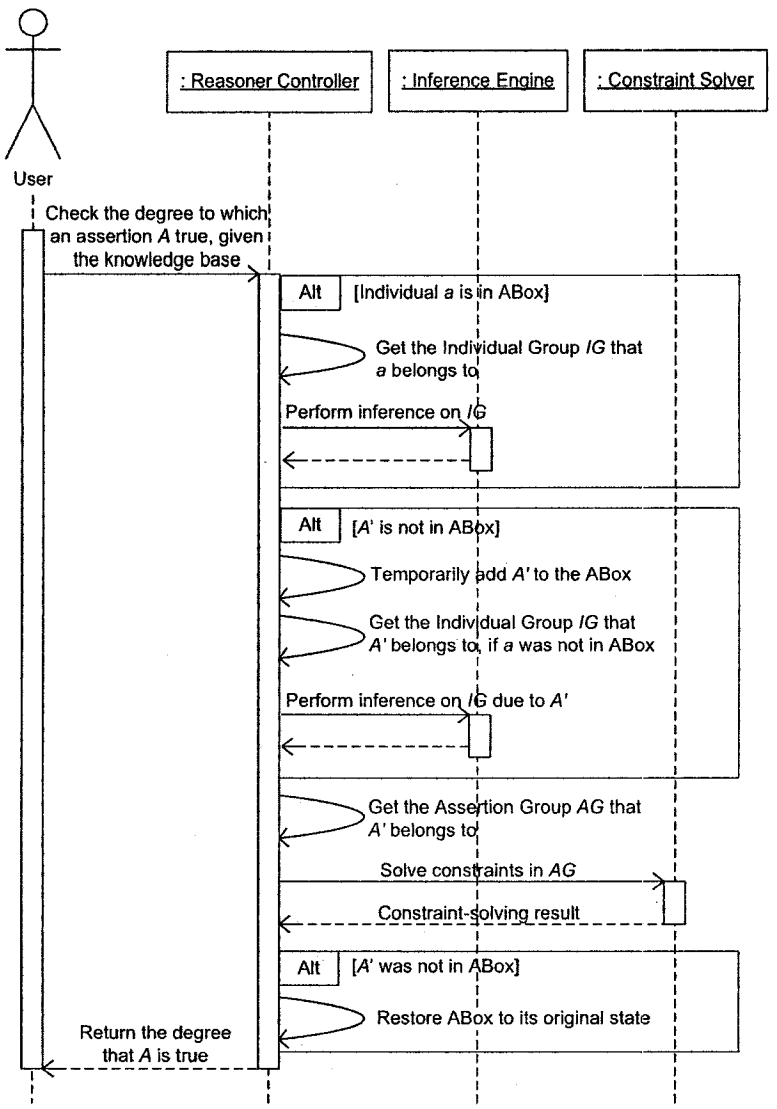

Figure 14: Sequence diagram for entailment checking

$\langle a : C \sqcap \neg D \mid x_{a:C\sqcap\neg D}, f_c, f_d \rangle$, where a is a new individual name. To find the subsumption degree, we temporarily add A to the ABox, and apply completion rules to the Individual Group that A belongs to. After that, we get the Assertion Group that A belongs to, and solve the constraints associated with it. Finally, the subsumption degree is returned. Figure 15 illustrates the sequence diagram for subsumption checking.

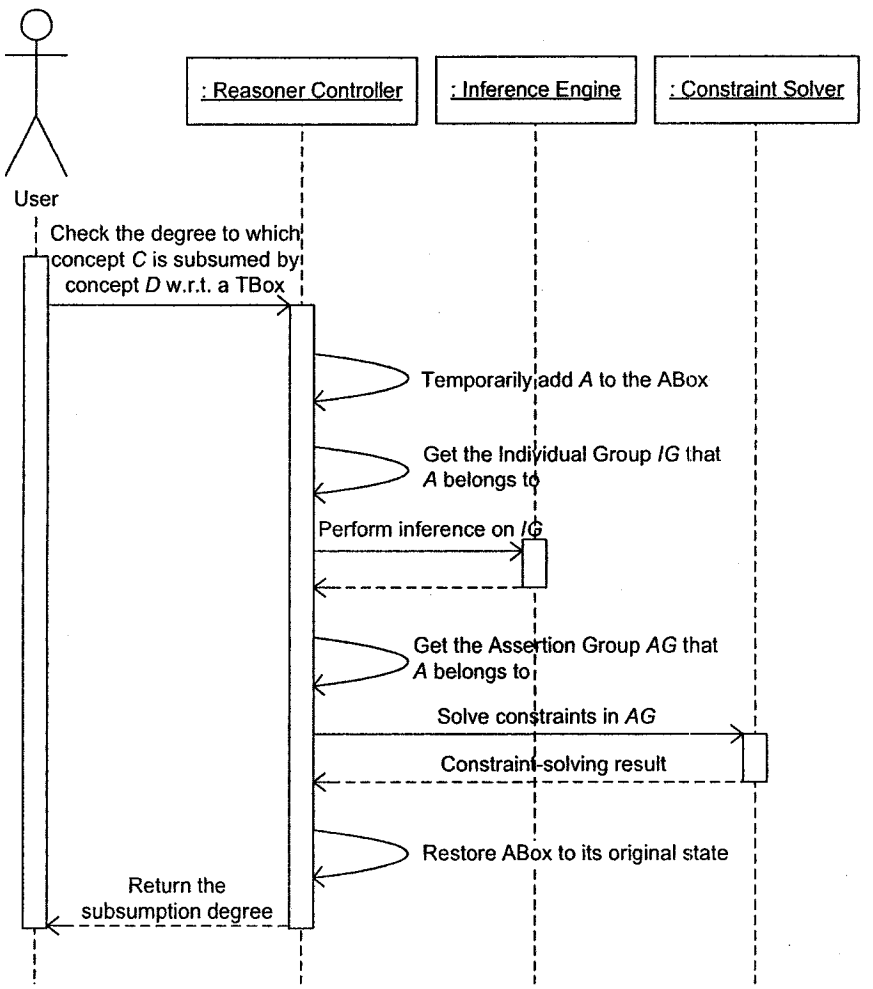

Figure 15: Sequence diagram for subsumption checking

5.2 Configuration Loader

The Configuration Loader is responsible for loading user preferences from the configuration file, and storing the preferences into the reasoner for later use. All the user preferences are stored as name-value pairs. For example,

```
certaintyLattice = [0,1]

enableCaching = true
```

The user preferences can be divided into three categories:

1. **System preferences** are those that are related to the operation of URDL, such as the location of the external constraint solver.

2. **Printing preferences** are concerned with URDL's information display, such as whether or not to print the running time. Each preference has a boolean value.

3. **Optimization preferences** are directives used to define optimization of the reasoner in effect during the reasoning procedure, such as whether caching should be turned on or off.

All the user preferences are optional. In case no user preference is specified, default values are used. The complete list of the user preferences and their descriptions can be found in Tables 33, 34, and 35 in Appendix A.

5.3 Parser

The URDL parser is responsible for parsing the user inputs, and storing them into the knowledge base. The module for parsing the user inputs was generated using JavaCC (Java Compiler Compiler) [Jav], which is an open source parser generator that gets the grammar in EBNF (Extended BackusNaur Form) notation as the input (see Table 36 in Appendix A), and generates a parser in Java as the output.

Like in many standard DL systems, URDL uses the prefix notation to represent the description language. Table 5 shows the correspondence between the $\mathcal{ALC}_U$ description language syntax and the URDL syntax. Note that URDL allows the keywords to be in either lowercases or uppercases, although we show the keywords only in lowercases in the table.

$\mathcal{ALC}_U$ **Syntax**	**URDL Syntax**
$\top$	`*top*`
$\bot$	`*bottom*`
$\neg C$	(`not` C)
$C \sqcap D$	(`and` C D)
$C \sqcup D$	(`or` C D)
$\exists R.C$	(`some` R C)
$\forall R.C$	(`all` R C)

Table 5: Correspondence between the $\mathcal{ALC}_U$ description language syntax and the URDL syntax

The axioms and assertions in URDL are also represented using the prefix notation. Table 6 shows the correspondence between the $\mathcal{ALC}_U$ syntax and the URDL syntax.

To interpret the description language and the knowledge base, the combination functions that are used in the knowledge base must be defined by the user. There are

$\mathcal{ALCC}_U$ Syntax	URDL Syntax
$\langle A \sqsubseteq C \mid [l, u], f_c, f_d \rangle$	(`define-primitive-concept` $A\ C \mid [l, u], f_c, f_d$)
$\langle A \equiv C \mid [l, u], f_c, f_d \rangle$	(`define-concept` $A\ C \mid [l, u], f_c, f_d$)
$\langle C \sqsubseteq D \mid [l, u], f_c, f_d \rangle$	(`implies` $C\ D \mid [l, u], f_c, f_d$)
$\langle C \equiv D \mid [l, u], f_c, f_d \rangle$	(`equivalent` $C\ D \mid [l, u], f_c, f_d$)
$\langle a : C \mid [l, u], f_c, f_d \rangle$	(`instance` $a\ C \mid [l, u], f_c, f_d$)
$\langle (a, b) : R \mid [l, u], -, - \rangle$	(`related` $a\ b\ R \mid [l, u], -, -$)

Table 6: Correspondence between the $\mathcal{ALCC}_U$ knowledge base syntax and the URDL syntax

two types of functions supported in URDL – general functions and special functions.

The general functions are defined using the following syntax:

$FunctionName(parameter_1, parameter_2) = FunctionDefinition$

where $parameter_1$ and $parameter_2$ are two variable names, and *FunctionDefinition* is any expression formed by using numbers, the defined variable names (i.e., $parameter_1$ and/or $parameter_2$), and operators $+, -, *$, and $/$, with the appropriate parenthesis in place. For a more intuitive way of defining functions, we employ the infix notation for combination functions. For example, the independent function can be defined as: $ind(x, y) = (x + y) - (x * y)$.

In addition to general functions, there are two types of special functions that can be defined using the following syntax:

$max(parameter_1, parameter_2) = max(expression_1, expression_2)$, and

$min(parameter_1, parameter_2) = min(expression_1, expression_2)$

where $parameter_1$ and $parameter_2$ are variable names, and $expression_1$ and $expression_2$ are any expressions formed by using numbers, the defined variable names (that is, $parameter_1$ and/or $parameter_2$), and operators $+, -, *$, and $/$, with the appropriate

parenthesis in place. For example, using Lukasiewicz logic, the conjunction function can be defined as: $max(x, y) = max(0, ((x + y) - 1))$, and the disjunction function can be defined as: $min(x, y) = min(1, (x + y))$.

Once the axioms, assertions, and combination functions are parsed, they are stored into the knowledge base. Since the procedures for storing axioms and combination functions are straightforward, we focus mainly on the procedure for storing assertions.

Before an assertion A is added to the ABox, we check whether it clashes with any of the existing assertions in the ABox. If no trivial clash is detected, we need to determine the appropriate Assertion Group to add A, using the decision procedure presented in Figure 9. If A is an atomic assertion, the Concept Assertion rule is applied. On the other hand, if A is a role assertion, the Role Assertion rule is applied. In addition, we need to determine which Individual Group should the individuals in the role assertion be placed, using the decision procedure presented in Section 4.2.4. The pseudo-codes for adding assertions into the ABox are presented in Figures 44, 45, 46, and 47 in Appendix A.

5.4 Inference Engine

In URDL, inference is performed on one Individual Group at a time. The Inference Engine is not only responsible for applying completion rules, but also for ensuring that the rules are being applied in the proper order. In this section, we study the completion rule application policy. We then present the non-generating rules and the

generating rules in detail.

Completion Rule Application Policy

To ensure that the completion rules are applied in a right order, specific completion rule application policy is followed. Figure 16 shows the procedure of completion rule application, where "generating rule" refers to the Role Exists Restrictions rule, and "non-generating rules" refer to all the other completion rules.

```
// Apply the completion rules to an Individual Group, IG
Do
{
  Step 1: Apply non-generating rules as long as possible to IG.

  Step 2: Apply a generating rule and restart with Step 1 as long as possible.

} While there is more change made to the ABox associated with IG.
```

Figure 16: Completion rule application policy

This order of applying completion rules ensures that rules are applied to generated individuals only if no more rule is applicable to the defined individuals. In addition, if I_1 and I_2 are two generated individuals, and if I_1 is a predecessor of I_2, the completion rules are applied to I_2 only if no more rule is applicable to I_1. This enforces the breadth first order.

Generating and Non-Generating Rules

Non-generating rules refer to all the completion rules except the Role Exists Restriction rule. The algorithms for the generating rules are straightforward, since they

can be directly translated from the completion rules described in Section 3.3.3. The pseudo-codes for the Negation, Conjunction, Disjunction, and Value Restriction rules can be found in Figures 48, 49, 50, and 51 respectively in Appendix A.

The Role Exists Restriction rule is also known as the generating rule, since this rule can introduce generated individual. To ensure termination, blocking condition is checked whenever the Role Exists Restriction rule is triggered to determine whether or not a new individual needs to be generated. That is, let *ind* be the individual in the assertion that triggered the Role Exists Restriction rule. If *ind* is a defined individual (refer to Definition 3.2.1), it is not blocked. Otherwise, *ind* is a generated individual, and we need to check if it is directly blocked by its predecessor. If it is not, then check if *ind* is indirectly blocked by its ancestors. The pseudo-code for the Role Exists Restriction rule is presented in Figure 52 in Appendix A.

Due to the optimization employed in URDL, each assertion is processed by the Inference Engine only once. That is, once the completion rule is applied to a given assertion, it is marked as being processed, and will not be processed again in the future. Although this method can avoid the reasoner from doing repeated work, it could potentially cause some missing completion rule applications. Consider the following example. Assume that there are three assertions in the ABox: $\langle a : \forall R.D \mid x_{a:\forall R.D}, min, max\rangle$, $\langle (a,b) : R \mid x_{(a,b):R}, -, -\rangle$, and $\langle a : \exists R.C \mid x_{a:\exists R.C}, min, max\rangle$, and for simplicity, there is no axiom in the TBox. According to the policy for application of completion rules (see Figure 16), we apply all the non-generating rules before the generating rule is applied. Hence, we first apply the Role Value Restriction rule

to $\langle a : \forall R.D \mid x_{a:\forall R.D}, min, max\rangle$ and mark it as being processed. We then apply the Role Exists Restriction rule to $\langle a : \exists R.C \mid x_{a:\exists R.C}, min, max\rangle$. Since individual a is not blocked, a new individual ind is generated, and the assertions $\langle (a, ind) : R \mid x_{(a,ind):R}, -, -\rangle$ and $\langle ind : C \mid x_{ind:C}, -, -\rangle$ are derived. At this point, since the assertions in the ABox are either processed or do not need to be processed (for role assertions and atomic assertions), we would stop the application of completion rules. However, this will cause one missing rule application, since the newly generated role assertion $\langle (a, ind) : R \mid x_{(a,ind):R}, -, -\rangle$ together with the assertion $\langle a : \forall R.D \mid x_{a:\forall R.D}, min, max\rangle$ should trigger the Role Value Restriction rule. However, since the assertion $\langle a : \forall R.D \mid x_{a:\forall R.D}, min, max\rangle$ is already marked as being processed, it will not be processed again. To overcome this problem, at the end of the Role Exists Restriction rule, we need to check whether it is necessary to further process the Role Value Restriction assertions associated with the given individual due to the newly derived role assertion (see the pseudo-codes in Figures 52 and 53).

5.5 Constraint Solver

The Constraint Solver is responsible for solving the constraints generated by the Inference Engine by calling an external constraint solver, and returning the result back to the Reasoner Controller. In URDL, RealPaver [Rea] is used as the external constraint solver, which is capable of solving linear/nonlinear equations and inequations over integer and real variables.

The constraints associated with one Assertion Group are solved at a time. To solve the constraints, the Constraint Solver first checks if the given Assertion Group was solved in the past. If so, it simply returns the cached result. If not, the constraints are transformed into the format used by the external constraint solver, which are then written into a file. Finally, the external constraint solver is called to solve the constraints, and the constraint-solving result is parsed and cached.

The process for transforming the URDL constraints into the format recognized by RealPaver is straightforward. For example, the constraint ($x_{a:C} \in [0.4, 0.6]$) in URDL is transformed into two constraints ($x_{a:C} \geq 0.4$) and ($x_{a:C} \leq 0.6$) in RealPaver. One restriction about RealPaver is that it does not accept special characters such as ":" inside variable names. Inaddition, it allows only variable names of length up to 40 characters. If the variable name is over 40 characters, it will be truncated. This may lead to unexpected results if there are multiple long variable names that have identical prefix in the first 40 characters. To overcome these problems, during the constraint rewriting, we map all the variables into short, generated variable names before they are written to files that are used by RealPaver.

The more complicated constraint rewriting comes when the constraint includes combination functions. For these constraints, the combination function name used in the constraint must be replaced with the corresponding function definition. For example, suppose we have the constraint: $ind(x_{a:C}, x_{a:D}) = x_{a:C \sqcup D}$, where ind is the independent function defined as: $ind(x, y) = (x + y) - (x * y)$. Then, the constraint will be translated into: $(x_{a:C} + x_{a:D}) - (x_{a:C} * x_{a:D}) = x_{a:C \sqcup D}$.

Once the constraints are written to the file, the external constraint solver, RealPaver, is called to solve the constraints. Since it is possible for RealPaver to terminate abnormally (such as in case of stack overflow), a thread is created to monitor such abnormal termination.

5.6 Experimenting with URDL

In this section, we demonstrate URDL by extending the example described in Section 1.3. Let us interpret this example as a fuzzy knowledge base, where the statement "The certainty that an obese person would have heart disease lies between 0.7 and 1" can be expressed in $\mathcal{ALC}_U$ as the axiom $\langle ObesePerson \sqsubseteq HeartPatient \mid [0.7, 1], min, max \rangle$, and the statement "John is obese with a degree between 0.8 and 1" can be captured as the assertion $\langle John : ObesePerson \mid [0.8, 1], -, - \rangle$. Assume that, in addition to the above information, we also know that John is a male person. His mother, Mary, is a diabetes patient. We also know that the certainty of a female person being a breast cancer patient is at least 0.65, and the certainty that somebody who has a diabetes mother is a diabetes patient is at least 0.9. Finally, we also know some general information, such as a male person is disjoint with a female person, and that the range of the role hasMother is a female person. Figure 17 illustrates how this knowledge base is represented using URDL syntax.

To determine the certainty with which John is a heart patient, we apply the entailment checking. As expected, this gives a certainty between 0.7 and 1, as shown

```
(implies ObesePerson HeartPatient | [0.7, 1], min, max)
(instance John ObesePerson | [0.8, 1], -, -)
(instance John MalePerson | [1, 1], -, -)
(related John Mary hasMother | [1, 1], -,-)
(instance Mary DiabetesPatient | [1, 1], -,-)
(implies FemalePerson BreastCancerPatient | [0.65, 1], min, max)
(implies (some hasMother DiabetesPatient) DiabetesPatient | [0.9,1], min, max)
(implies (and MalePerson FemalePerson) *bottom* | [1, 1], min, max)
(implies *top* (all hasMother FemalePerson) | [1,1], min, max)
```

Figure 17: Example represented using URDL syntax

in Figure 18.

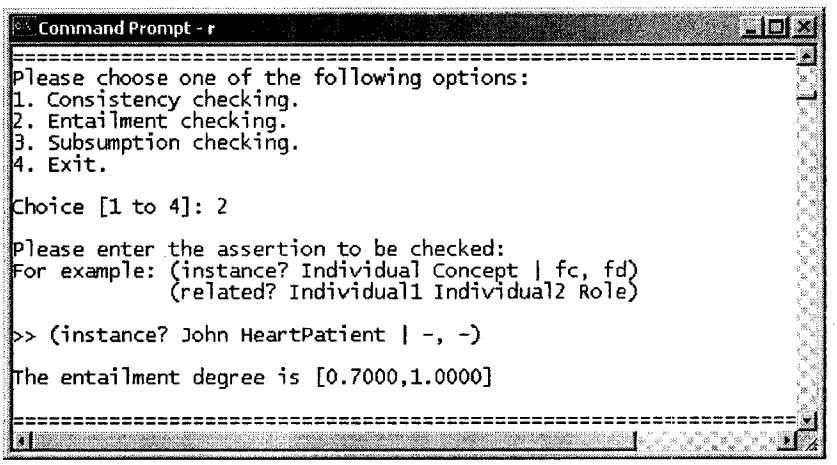

Figure 18: URDL screenshot

Other interesting queries can also be posted. For example, to find out the certainty that John is a diabetes patient or a heart patient, we input the query:

`(instance? John (or HeartPatient DiabetesPatient) | min, max)`

which returns an entailment degree between 0.9 and 1. This is due to the inferred fact that the certainty with which John is a diabetes patient is at least 0.9, whereas the certainty that John is a heart patient is at least 0.7.

Consider another query for which we would like to know the certainty that John has a mother who is both a breast cancer patient and a diabetes patient. For this, we input the query:

```
(instance?  John (some hasMother (and BreastCancerPatient
 DiabetesPatient) | min, max)
```

and we obtain an entailment degree of at least 0.65. It is interesting to note that, although we did not explicitly assert that Mary is a female person, we inferred that Mary is a breast cancer patient with a certainty of at least 0.65 through the fact that Mary is John's mother, a mother is a female person, and a female person is a breast cancer patient with a certainty of at least 0.65.

5.7 Summary and Concluding Remarks

In this chapter, we presented the five components that made up URDL. The Reasoner Controller, which acts as the brain of URDL, first gets the knowledge base and configuration preferences from the user. It then calls the Configuration Loader to load the user preferences and calls the Parser to parse the knowledge base. It then gets the user query and calls the Inference Engine to apply the reasoning procedure. Finally, the derived constraints are solved by calling the Constraint Solver and the query result is returned to the user.

URDL contains roughly 8500 lines of Java code, among which 1150 lines of code are mixed with JavaCC syntax used to generate the URDL parser. The main reason

to use JavaCC (formerly known as Jack, developed by Sun Microsystems) as our parser generator was that it is open source, has intuitive syntax, and the parser it generates conforms well to the Java standard. One problem we encountered during the development of URDL was that, by default, JavaCC emits Java SE version 1.4 source code when generating the parser. However, since URDL was programed using Java's generics feature [Mah04] which is available only for Java SE versions 5.0 and later, we had version incompatible problem [Suna]. This problem was solved by setting the JDK_VERSION flag in JavaCC, which specifies the Java version to be used to generate the parser [Cop07].

URDL used RealPaver as its external constraint solver, since it is open source, is capable of solving linear/nonlinear equations/inequations over integer and real variables, and is freely available. Note, however, that URDL can be easily modified to use other external constraint solvers, since we only need to rewrite the constraints to a syntax accepted by the constraint solver, call the appropriate solver, and parse the result returned from the solver. RealPaver also inspired us to optimize the constraint-solving aspect of URDL. As we will present in Section 6.3.1, when the knowledge base size increases, the constraints set becomes too large for RealPaver to handle. This inspired us to partition the constraints into smaller, independent sets as we presented in Section 4.2.3.

Chapter 6

Performance Evaluation

In this chapter, we study the performance of URDL and show the effectiveness of the optimization techniques described in Chapter 4. All the experiments were conducted under Windows XP Professional on an Intel® 2.40 GHz Core™2 computer with 3.25 GB of RAM. All the time measures are in seconds, and were computed as the average of ten independent runs. In addition, the time measures in the figures are shown in logarithmic-scale.

In what follows, we first describe in Section 6.1 the test cases used to evaluate URDL. We then present the performance of URDL in Section 6.2. Finally, Section 6.3 demonstrates the effect of the proposed optimization techniques.

6.1 Test Cases

Since we were unable to find suitable test cases to evaluate the performance of URDL, we had to either manually create or write test case generators to generate the test cases.

First, to see how different properties of the knowledge base affect URDL's performance, we manually created some test cases that are similar in many aspects but differ in some details. These test cases were created by first carefully building a standard $\mathcal{ALC}$ knowledge base that makes use of all the $\mathcal{ALC}$ description language constructors. Then, different variations of this knowledge base were created by tuning its certainty domain, combination functions, and the type of axioms. The properties of the test cases are shown in Table 7, including: the number of concept assertions (C) in each test case, the number of role assertions (R), the number of axioms with necessary condition (N), the number of axioms with concept definitions (D), the combination functions (F), the certainty domain ($\mathcal{V}$), the number of Individual Groups (IG), the number of Assertion Groups (AG), the width of the ABox (W), and the height of the ABox (H).

Test Case	C	R	N	D	F	$\mathcal{V}$	IG	AG	W	H
`Standard`	15	2	5	0	*min/max*	$\{0, 1\}$	3	84	25	6
`Min-Max`	15	2	5	0	*min/max*	$\mathcal{C}[0, 1]$	3	84	25	6
`Mixed`	15	2	5	0	*min/max/prod/ind*	$\mathcal{C}[0, 1]$	3	159	44	6
`Min-Max/Def`	15	2	0	5	*min/max*	$\mathcal{C}[0, 1]$	3	13	58	6

Table 7: Properties of test cases

Note that all the test cases have the same number of axioms and assertions. The test case `Standard` is a standard $\mathcal{ALC}$ knowledge base, where we use $\{0, 1\}$ for the

certainty domain, and *min* and *max* for its conjunction and disjunction functions. The test case `Min-Max` is a fuzzy knowledge base, which uses the same combination functions as the standard case, and $C[0,1]$ as the certainty domain, where $C[0,1]$ is the set of closed subintervals $[\alpha, \beta]$ in [0, 1] such that $\alpha \preceq \beta$. The test case `Mixed` is very similar to the test case `Min-Max`, except that half of its axioms and assertions use non-linear functions like product (*prod*) and independent (*ind*) functions as the combination functions, and the other half use simple functions like *min* and *max*. The test case `Min-Max/Def` differs from the test case `Min-Max` in the type of axioms it contains. That is, `Min-Max/Def` contains axioms with concept definitions instead of those with necessary conditions.

The second type of test cases is used to test scalability of URDL. Since it is not feasible to create test cases with thousands of axioms/assertions manually, we developed test case generators that automatically generate test cases with different properties, including:

1. **Complexity (simple or complex).** The simple test cases contain only assertions, while complex cases include both axioms and assertions.

2. **Shape (deep or wide).** The deep test cases contain chains of role assertions of the form $\langle (I_0, I_1) : R_0 \mid \alpha_0, f_c, f_d \rangle$, $\langle (I_1, I_2) : R_1 \mid \alpha_1, f_c, f_d \rangle$, $\langle (I_2, I_3) : R_0 \mid \alpha_2, f_c, f_d \rangle$, ... On the other hand, wide test cases contain tree-like role assertions of the form $\langle (I_0, I_1) : R_0 \mid \alpha_0, f_c, f_d \rangle$, $\langle (I_0, I_2) : R_1 \mid \alpha_1, f_c, f_d \rangle$, $\langle (I_0, I_3) : R_0 \mid \alpha_2, f_c, f_d \rangle$, ... These two shapes are shown in Figure 19 (a) and (b), respectively.

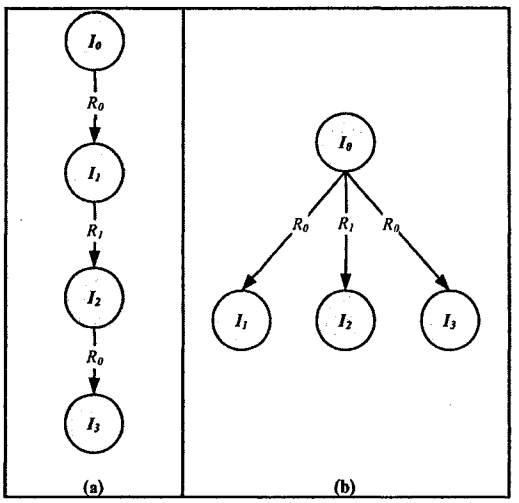

Figure 19: Deep (a) and Wide (b) test cases

3. **Size.** The size of the test cases refers to the number of role assertions in the original ABox, since these assertions form the preliminary shape of the knowledge bases. For example, for a deep test case of size 10, we have a chain of ten role assertions in the knowledge base. On the other hand, for a wide test case of size 10, the ten role assertions form a tree-like structure. Moreover, for simple test cases, additional concept assertions are asserted for each individual in the ABox, and for complex test cases, additional axioms with concept definitions are asserted.

Table 8 lists the properties of some generated test cases. The test cases are named based on the three properties of the test cases. For example, the test case `Simple-Deep-50` refers to a test case that is simple, deep, and its size is 50.

Test Case	C	R	N	D	F	$\mathcal{V}$	IG	AG	W	H
Simple-Deep-10	11	10	0	0	*min/max*	[0, 1]	1	21	1	10
Simple-Deep-50	51	50	0	0	*min/max*	[0, 1]	1	101	1	50
Simple-Deep-100	101	100	0	0	*min/max*	[0, 1]	1	201	1	100
...	...	...	...	...	...	...	...	...	...	...
Simple-Wide-10	11	10	0	0	*min/max*	[0, 1]	1	21	10	1
Simple-Wide-50	51	50	0	0	*min/max*	[0, 1]	1	101	50	1
Simple-Wide-100	101	100	0	0	*min/max*	[0, 1]	1	201	100	1
...	...	...	...	...	...	...	...	...	...	...
Complex-Deep-10	2	10	0	10	*min/max*	[0, 1]	3	527	28	14
Complex-Deep-50	2	50	0	10	*min/max*	[0, 1]	3	2142	28	54
Complex-Deep-100	2	100	0	10	*min/max*	[0, 1]	3	4179	28	104
...	...	...	...	...	...	...	...	...	...	...
Complex-Wide-10	2	10	0	10	*min/max*	[0, 1]	3	527	78	5
Complex-Wide-50	2	50	0	10	*min/max*	[0, 1]	3	2142	318	5
Complex-Wide-100	2	100	0	10	*min/max*	[0, 1]	3	4179	618	5
...	...	...	...	...	...	...	...	...	...	...

Table 8: Properties of some generated test cases

6.2 Performance Evaluation Results

The performance of URDL is evaluated based on the time it takes to load the knowledge base (Load), to apply the completion rules (Inference), to solve the constraints (Constraint-Solving), and other input/output time (I/O). Therefore, the overall running time (Total) can be computed by summing up the above four time measures. Note that the running time reported in this section is measured when all URDL's optimization features are enabled. The comparison between the performance evaluation results with or without an optimization will be given in Section 6.3.

Table 9 shows how URDL behaves for the test cases listed in Table 7. Recall that these test cases are similar in many aspects but differ in some details, which allow us to study the effect of different properties of the knowledge base on the running time. From the above table, we can make the following observations:

Test Case	Load	Inference	Constraint-Solving	I/O	Total
`Standard`	0.014	0.10	2.19	0.22	2.52
`Min-Max`	0.015	0.10	2.47	0.19	2.78
`Mixed`	0.016	0.17	12.92	0.32	13.43
`Min-Max/Def`	0.016	0.54	21.36	0.29	22.20

Table 9: Performance measures in seconds

1. The time spent on solving constraints dominates the overall running time for all the test cases. This shows the importance of optimizing the constraint solving process.

2. When the certainty domain is [0, 1] (for the test case `Min-Max`), it takes longer to solve the constraints than when the certainty domain is $\{0, 1\}$ (for the test case `Standard`). This can be attributed to the way that the external constraint solver, RealPaver, deals with the certainty values.

3. It takes much longer to solve constraints that include some nonlinear functions (for the test case `Mixed`) compared to dealing with only simple constraints (for the test case `Min-Max`). This is due to the way that RealPaver handles different functions.

4. A TBox consisting of axioms with concept definitions (for the test case `Min-Max/Def`) takes much longer to perform the inference and to solve constraints compared to a TBox consisting of axioms with necessary conditions (for the test case `Min-Max`). This is expected since each axiom with concept definition of the form $\langle C \equiv D \mid \alpha, f_c, f_d \rangle$ is equivalent to two axioms with necessary conditions of the form $\langle C \sqsubseteq D \mid \alpha, f_c, f_d \rangle$ and $\langle D \sqsubseteq C \mid \alpha, f_c, f_d \rangle$.

To test scalability of URDL, we measured the running time of the test cases listed in Table 8. Recall that these generated test cases are of different complexities (`simple` or `complex`), shapes (`deep` or `wide`), and sizes.

We first study the performance of the `simple` test cases. Figures 20 and 21 show the performance of the test cases `Simple-Deep` and `Simple-Wide` respectively. The x-axis in a figure indicates the sizes of the test cases, and the y-axis shows the running time in seconds using logarithmic-scale. For example, Figure 20 illustrates how different sizes of the test case `Simple-Deep` behave by showing the running time as the load time, the inference time, the constraint-solving time, other I/O time, and the total time. For instance, consider the test case size 100, which gives the performance of the test case `Simple-Deep-100` (see Table 8). It takes 0.01 second to apply the completion rules, 0.08 second to load the knowledge base, 0.5 second for input and output operations, and 3.35 seconds to solve the constraints. So, the total running time is 3.94 seconds.

From Figures 20 and 21, we can observe that, when the knowledge base is `simple` (i.e., one that has only ABox and no TBox), not much time is spent on performing inferences. In addition, when the knowledge base is `wide`, the time spent on loading the knowledge base increases rapidly. This is because, as the knowledge base becomes wider, the time to access/store the assertions from/to the knowledge base increases (as the parser needs to check for possible duplications, conflicts, and so on). This shows the importance of using an efficient data structure to store the assertions, and efficient string comparisons.

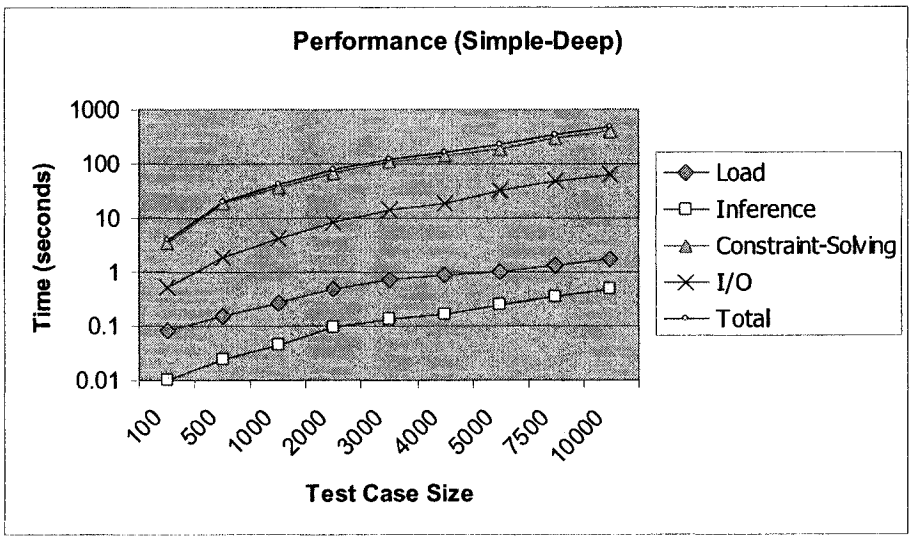

Figure 20: Performance of the test cases `Simple-Deep`

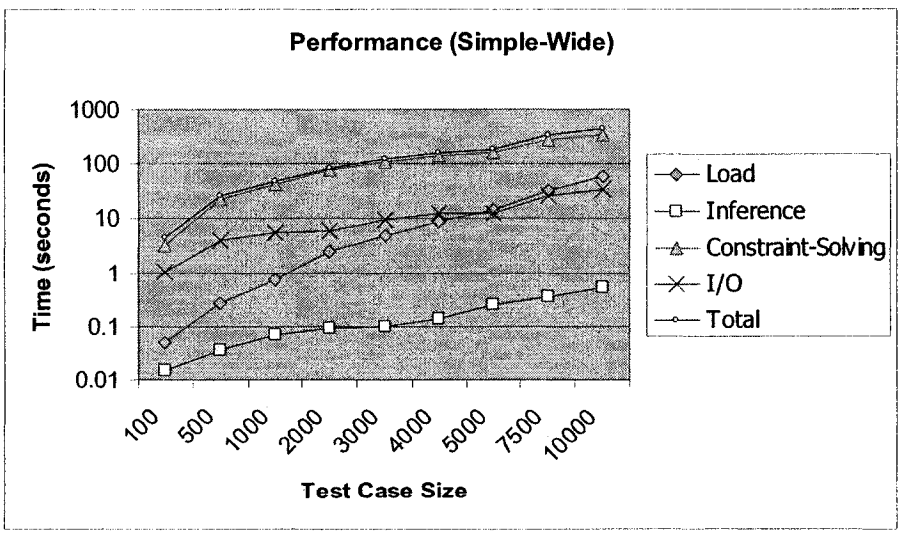

Figure 21: Performance of the test cases `Simple-Wide`

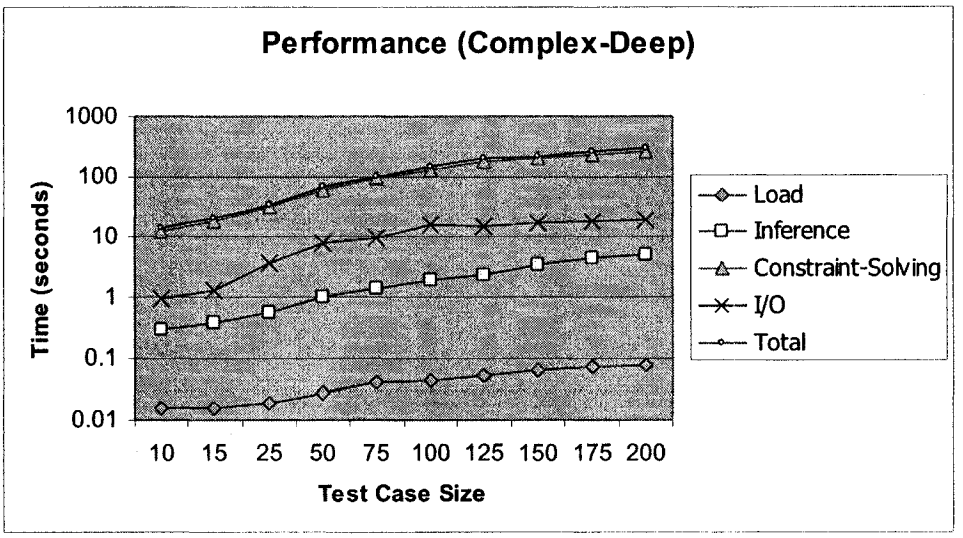

Figure 22: Performance of the test cases `Complex-Deep`

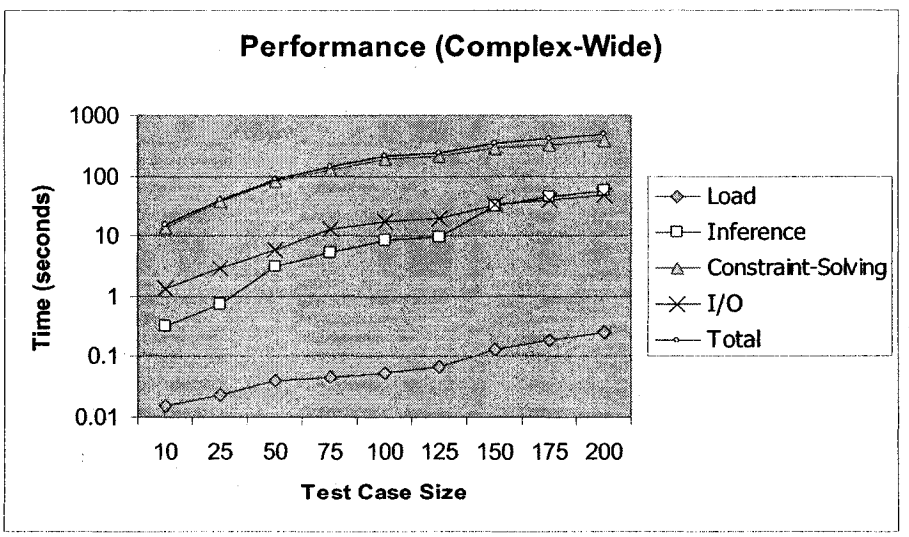

Figure 23: Performance of the test cases `Complex-Wide`

We next study the performance of the `complex` test cases `Complex-Deep` and `Complex-Wide` as shown in Figures 22 and 23 respectively. The experimental results show that, when the knowledge bases are `complex`, the time spent on inferencing exceeds the time spent on loading the knowledge bases. This situation is reversed when the knowledge bases are `simple`. Indeed, it takes much longer to perform inferences when the TBox is not empty. This is because, as described in Section 3.3, for each individual in the ABox, we must assert that it satisfies all the axioms in the TBox. Note also that, when the knowledge bases are `wide`, the time spent on applying inference rules exceeds the time spent on I/O as the test case size increases.

6.3 Optimization Effects

In order to test the effect of the optimization techniques mentioned in Chapter 4, the test cases were run with one or more of the optimizations disabled. The performance improvement was measured using the *speed-up factor*, which is defined by the following formula:

$$\text{Speedup} = \frac{T_{Off}}{T_{On}} \tag{1}$$

where Speedup is the speed-up factor, and T_{Off} and T_{On} are the running times when a particular optimization is turned off and on respectively. We also use "T/O" to denote "Time-Out" when a running was forced to terminate, and "N/A" to denote "Not Applicable."

In what follows, we present the experimental results for optimization techniques

including partitioning based on connectivity, optimized Individual Group creation, optimized clash detection, caching, optimized hashing, and optimized string comparison. Since lexical normalization and concept simplification are optimization techniques directly adopted from the standard DL systems, we suppress the experimental results for these two techniques. At the end of this section, we also present the overall effect of the optimization techniques.

6.3.1 Effect of Partitioning Based on Connectivity

Recall that an ABox can be partitioned based on connectivity into Individual Groups and Assertion Groups, as described in Section 4.2.3. To test the effect of the different bases for partitioning, we measured the total running time when the ABox is partitioned into Assertion Groups (AG), Individual Groups (IG), and when there is no partition at all (ALL). Moreover, we compared these time measures using three different speed-up factors:

$$\text{SpUp}_{IG/AG} = \frac{T_{IG}}{T_{AG}} \tag{2}$$

$$\text{SpUp}_{ALL/IG} = \frac{T_{ALL}}{T_{IG}} \tag{3}$$

$$\text{SpUp}_{ALL/AG} = \frac{T_{ALL}}{T_{AG}} \tag{4}$$

where SpUp denotes the speed-up factor, and T_{AG}, T_{IG}, and T_{ALL} denote the running times when the ABox is partitioned into Assertion Groups, Individual Groups, and when there is no partition respectively.

Figure 24 and Table 10 show the performance improvement of partitioning based

on connectivity for the test cases listed in Table 7. It is interesting to note that, regardless of the certainty domain, the combination functions, and other factors, we achieved the best performance when the ABox was partitioned into Assertion Groups, and the worst performance when the ABox was not partitioned.

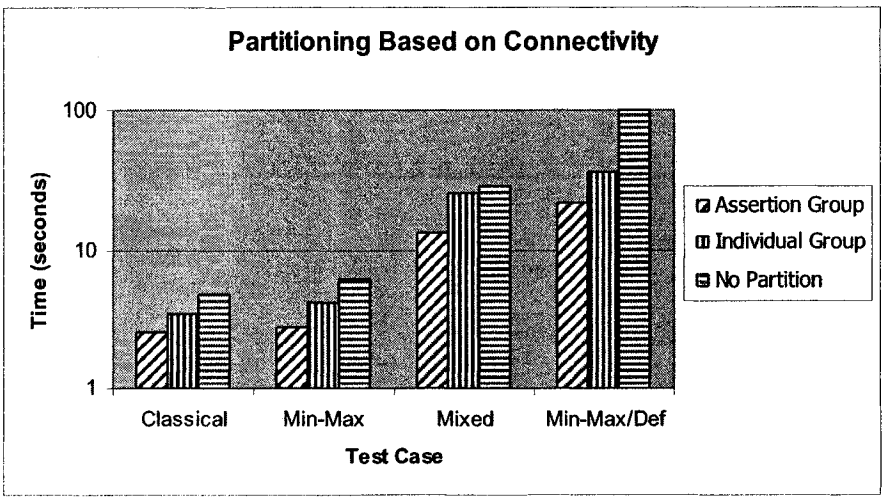

Figure 24: Effect of partitioning based on connectivity

Test Case	AG	IG	ALL	$SpUp_{IG/AG}$	$SpUp_{ALL/IG}$	$SpUp_{ALL/AG}$
`Standard`	2.52	3.42	4.84	1.36	1.42	1.92
`Min-Max`	2.78	4.15	6.17	1.49	1.49	2.22
`Mixed`	13.43	25.54	28.99	1.90	1.14	2.16
`Min-Max/Def`	22.20	37.24	100.58	1.68	2.70	4.53

Table 10: Effect of partitioning based on connectivity (in seconds)

To study how the complexity, the shape, and the size of a test case affect the partitioning technique, we evaluated the performance of the test cases `Simple-Deep`, `Simple-Wide`, `Complex-Deep`, and `Comples-Wide`. Figures 25, 26, 27, 28, and Tables 11, 12, 13, 14 indicate the experimental results.

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

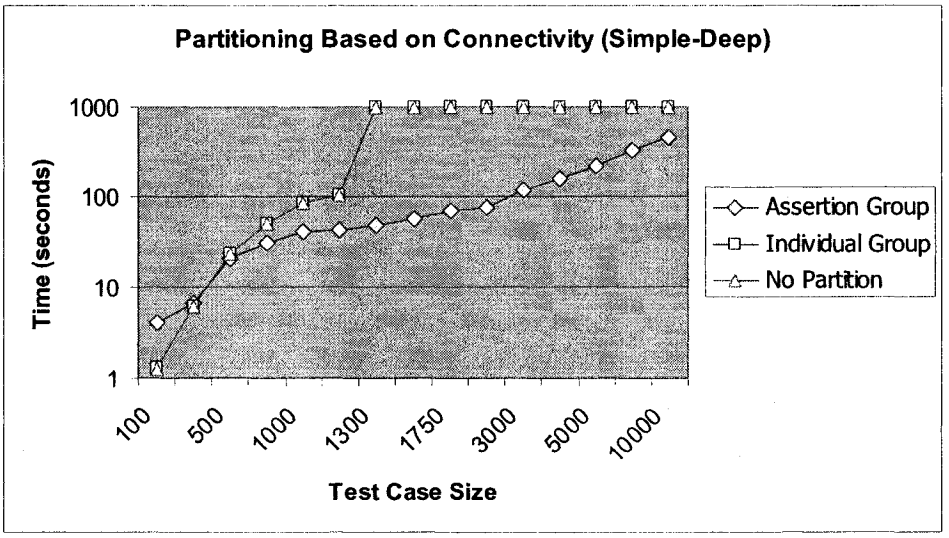

Figure 25: Effect of partitioning based on connectivity - `Simple-Deep`

Test Case	AG	IG	ALL	$SpUp_{IG/AG}$	$SpUp_{ALL/IG}$	$SpUp_{ALL/AG}$
`Simple-Deep-100`	3.94	1.27	1.27	0.32	1.00	0.32
`Simple-Deep-250`	6.64	6.04	6.03	0.91	1.00	0.91
`Simple-Deep-500`	20.28	23.60	23.65	1.16	1.00	1.17
`Simple-Deep-750`	31.25	50.98	50.99	1.63	1.00	1.63
`Simple-Deep-1000`	41.22	87.25	87.61	2.12	1.00	2.13
`Simple-Deep-1100`	43.13	105.22	104.70	2.44	1.00	2.43
`Simple-Deep-1300`	48.09	T/O	T/O	N/A	N/A	N/A
`Simple-Deep-1500`	56.85	T/O	T/O	N/A	N/A	N/A
`Simple-Deep-1750`	69.29	T/O	T/O	N/A	N/A	N/A
`Simple-Deep-2000`	75.75	T/O	T/O	N/A	N/A	N/A
`Simple-Deep-3000`	121.57	T/O	T/O	N/A	N/A	N/A
`Simple-Deep-4000`	156.77	T/O	T/O	N/A	N/A	N/A
`Simple-Deep-5000`	223.28	T/O	T/O	N/A	N/A	N/A
`Simple-Deep-7500`	336.54	T/O	T/O	N/A	N/A	N/A
`Simple-Deep-10000`	461.22	T/O	T/O	N/A	N/A	N/A

Table 11: Effect of partitioning based on connectivity - `Simple-Deep` (in seconds)

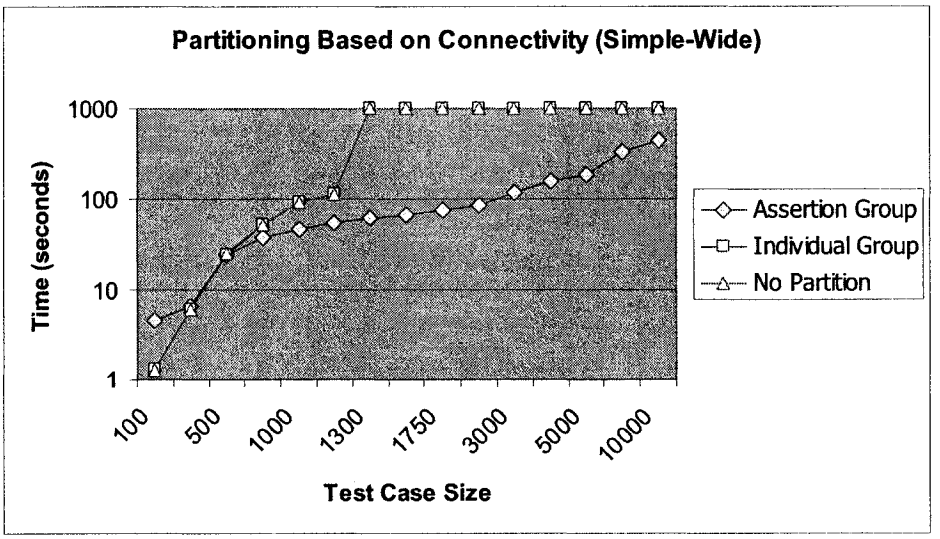

Figure 26: Effect of partitioning based on connectivity - `Simple-Wide`

Test Case	AG	IG	ALL	$SpUp_{IG/AG}$	$SpUp_{ALL/IG}$	$SpUp_{ALL/AG}$
`Simple-Wide-100`	4.49	1.28	1.27	0.28	1.00	0.28
`Simple-Wide-250`	6.50	6.11	6.13	0.94	1.00	0.94
`Simple-Wide-500`	24.15	24.72	24.75	1.02	1.00	1.02
`Simple-Wide-750`	37.43	53.12	53.36	1.42	1.00	1.43
`Simple-Wide-1000`	47.50	93.68	93.71	1.97	1.00	1.97
`Simple-Wide-1100`	55.31	114.59	114.80	2.07	1.00	2.08
`Simple-Wide-1300`	62.70	T/O	T/O	N/A	N/A	N/A
`Simple-Wide-1500`	66.30	T/O	T/O	N/A	N/A	N/A
`Simple-Wide-1750`	75.34	T/O	T/O	N/A	N/A	N/A
`Simple-Wide-2000`	84.85	T/O	T/O	N/A	N/A	N/A
`Simple-Wide-3000`	120.28	T/O	T/O	N/A	N/A	N/A
`Simple-Wide-4000`	162.06	T/O	T/O	N/A	N/A	N/A
`Simple-Wide-5000`	185.51	T/O	T/O	N/A	N/A	N/A
`Simple-Wide-7500`	333.50	T/O	T/O	N/A	N/A	N/A
`Simple-Wide-10000`	434.73	T/O	T/O	N/A	N/A	N/A

Table 12: Effect of partitioning based on connectivity - `Simple-Wide` (in seconds)

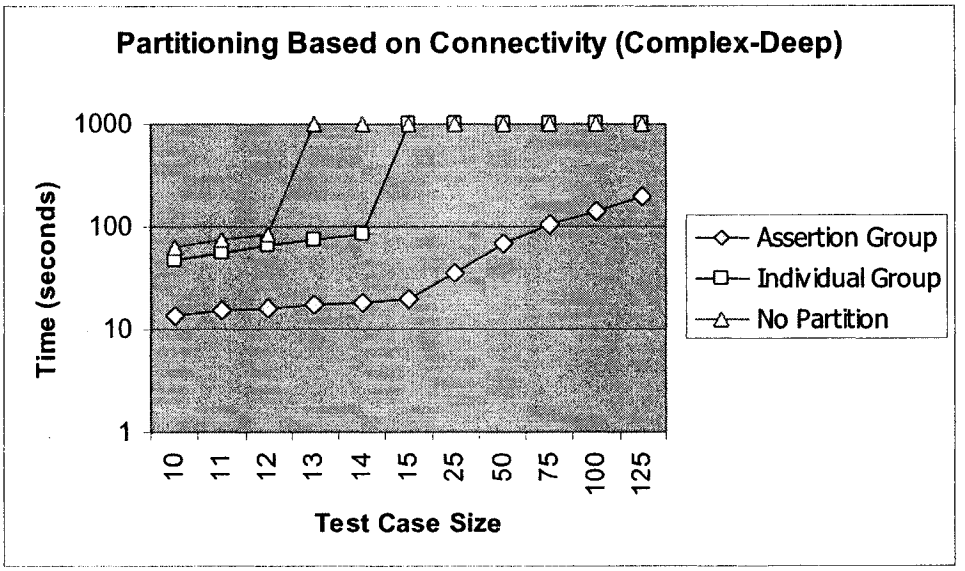

Figure 27: Effect of partitioning based on connectivity - `Complex-Deep`

Test Case	AG	IG	ALL	$SpUp_{IG/AG}$	$SpUp_{ALL/IG}$	$SpUp_{ALL/AG}$
`Complex-Deep-10`	13.64	47.76	63.33	3.50	1.33	4.64
`Complex-Deep-11`	15.18	55.74	74.70	3.67	1.34	4.92
`Complex-Deep-12`	16.02	64.99	85.58	4.06	1.32	5.34
`Complex-Deep-13`	17.18	74.58	T/O	4.34	N/A	N/A
`Complex-Deep-14`	18.20	84.07	T/O	4.62	N/A	N/A
`Complex-Deep-15`	19.80	T/O	T/O	N/A	N/A	N/A
`Complex-Deep-25`	34.79	T/O	T/O	N/A	N/A	N/A
`Complex-Deep-50`	69.54	T/O	T/O	N/A	N/A	N/A
`Complex-Deep-75`	102.91	T/O	T/O	N/A	N/A	N/A
`Complex-Deep-100`	142.63	T/O	T/O	N/A	N/A	N/A
`Complex-Deep-125`	197.51	T/O	T/O	N/A	N/A	N/A

Table 13: Effect of partitioning based on connectivity - `Complex-Deep` (in seconds)

Test Case	AG	IG	ALL	$SpUp_{IG/AG}$	$SpUp_{ALL/IG}$	$SpUp_{ALL/AG}$
`Complex-Wide-10`	15.54	48.43	63.89	3.12	1.32	4.11
`Complex-Wide-11`	17.09	56.86	75.70	3.33	1.33	4.43
`Complex-Wide-12`	18.05	64.01	85.39	3.55	1.33	4.73
`Complex-Wide-13`	22.24	78.95	T/O	3.55	N/A	N/A
`Complex-Wide-14`	23.06	84.57	T/O	3.67	N/A	N/A
`Complex-Wide-15`	24.61	T/O	T/O	N/A	N/A	N/A
`Complex-Wide-25`	40.30	T/O	T/O	N/A	N/A	N/A
`Complex-Wide-50`	89.10	T/O	T/O	N/A	N/A	N/A
`Complex-Wide-75`	141.61	T/O	T/O	N/A	N/A	N/A
`Complex-Wide-100`	209.94	T/O	T/O	N/A	N/A	N/A
`Complex-Wide-125`	242.60	T/O	T/O	N/A	N/A	N/A

Table 14: Effect of partitioning based on connectivity - `Complex-Wide` (in seconds)

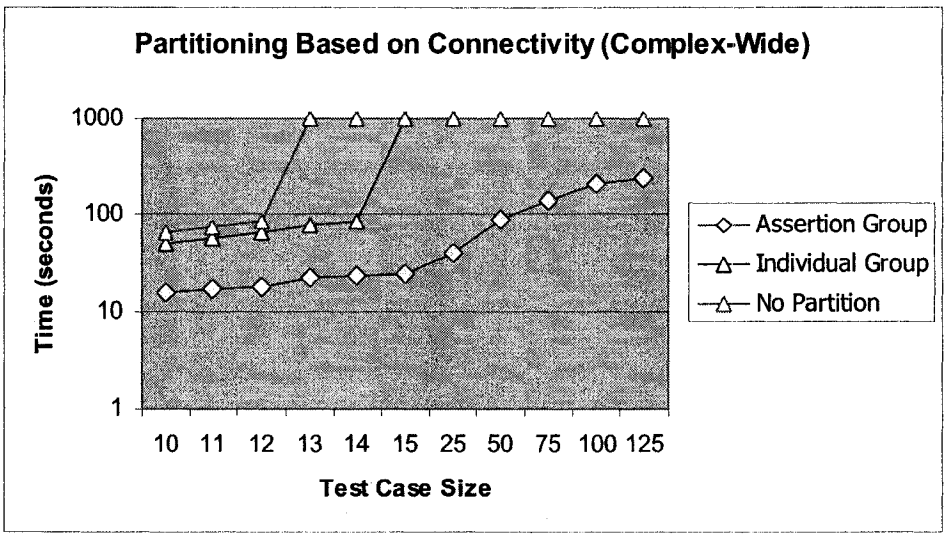

Figure 28: Effect of partitioning based on connectivity - `Complex-Wide`

1. Partitioning the ABox into smaller groups does not always work well. This is particularly the case when the knowledge base is `simple` and small (see test cases `Simple-Deep` and `Simple-Wide`, when the test case size is below 500). This is understandable since when the knowledge base is simple and small, it does not take long to solve the constraints set even without partitioning, because the number of constraints is small. On the other hand, with partitioning, although the overall time spent on solving constraints may be shorter than when there is no partition, the time spent on loading the constraint solver into the memory is longer because it involves loading and offloading many times. Hence, the overall effect is worse compared to when there is no partitioning.

2. When the knowledge base is complex, even if its size is small, partitioning into Assertion Groups always outperforms partitioning into Individual Groups,

which in turn always outperforms no partitioning. This is true regardless of the shape (deep or wide) of the knowledge bases.

3. The larger the knowledge base, the more gain we get by partitioning the ABox. This is shown not just in terms of the running time, but also in terms of the solvability of the constraints. For example, for the test cases `Simple-Deep` and `Simple-Wide`, if the ABox was not partitioned into Assertion Groups when the test case size reaches over 1100, the constraints set was too large for the external constraint solver to handle (we got a stack overflow error). On the other hand, if the same test cases were partitioned into Assertion Groups, they scaled to test cases of size of 10000 and above. This shows the importance of partitioning the ABox to keep the constraints set as small as possible.

4. Since the test cases `Simple-Deep` and `Simple-Wide` have only one Individual Group, the performance of partitioning into Individual Groups is the same as that of ABox with no partition.

6.3.2 Effect of Optimized Individual Group Creation

Optimized Individual Group creation is an optimization technique aimed to avoid redundant creation of Individual Groups when we load the knowledge base. Since this optimization affects only the load time, we compare only the time that URDL takes to load the test cases.

As mentioned in Section 4.2.4, optimized Individual Group creation is particularly

useful if there is long chain of role assertions in the ABox. That is, when we have a deep knowledge base. Figure 29 and Table 15 illustrate this case by comparing the time spent on loading the test case `Simple-Deep` when the optimization is on and off. The experimental results confirm that optimized Individual Group creation clearly improves the load time for deep knowledge bases. Also, the larger the knowledge base, the more we gain from this optimization.

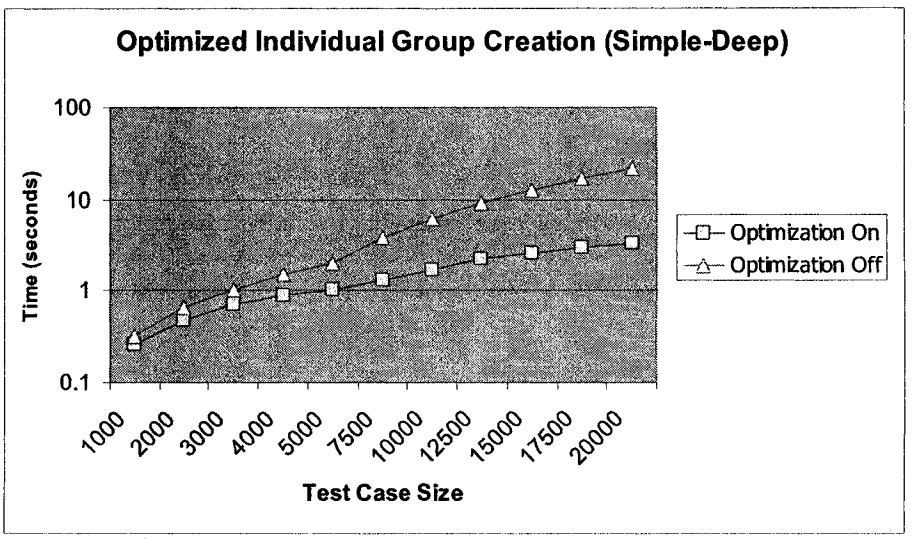

Figure 29: Effect of optimized Individual Group creation - `Simple-Deep`

In addition to the deep knowledge bases, we also studied the effect of optimized Individual Group creation on wide knowledge bases. As shown in Figure 30 and Table 16, this optimization does not improve the performance when the knowledge base is wide. However, note that it does not decrease the performance either.

Test Case	Optimization On	Optimization Off	Speedup
`Simple-Deep-1000`	0.26	0.32	1.24
`Simple-Deep-2000`	0.47	0.66	1.39
`Simple-Deep-3000`	0.70	1.01	1.45
`Simple-Deep-4000`	0.87	1.48	1.71
`Simple-Deep-5000`	1.00	2.01	2.01
`Simple-Deep-7500`	1.28	3.80	2.96
`Simple-Deep-10000`	1.67	6.26	3.75
`Simple-Deep-12500`	2.22	9.16	4.12
`Simple-Deep-15000`	2.61	12.84	4.92
`Simple-Deep-17500`	2.99	16.83	5.63
`Simple-Deep-20000`	3.28	21.67	6.60

Table 15: Effect of optimized Individual Group creation - `Simple-Deep` (in seconds)

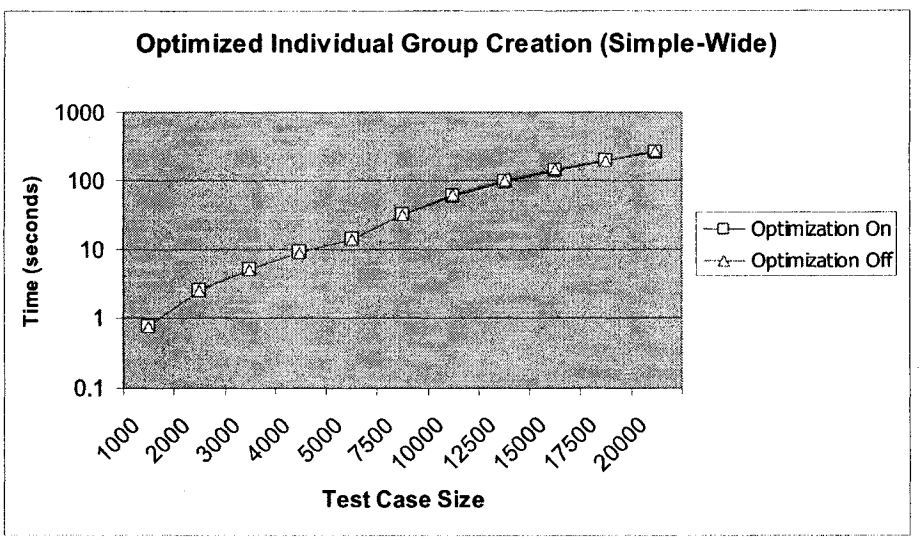

Figure 30: Effect of optimized Individual Group creation - `Simple-Wide`

Test Case	Optimization On	Optimization Off	Speedup
Simple-Wide-1000	0.76	0.77	1.00
Simple-Wide-2000	2.45	2.50	1.02
Simple-Wide-3000	5.00	5.14	1.03
Simple-Wide-4000	9.00	9.05	1.01
Simple-Wide-5000	14.10	14.11	1.00
Simple-Wide-7500	31.30	32.31	1.03
Simple-Wide-10000	58.64	60.55	1.03
Simple-Wide-12500	97.07	97.96	1.01
Simple-Wide-15000	142.26	144.13	1.01
Simple-Wide-17500	198.59	200.20	1.01
Simple-Wide-20000	264.36	267.79	1.01

Table 16: Effect of optimized Individual Group creation - Simple-Wide (in seconds)

6.3.3 Effect of Optimized Clash Detection

In order to test the effectiveness of optimized clash detection, contradicting assertions/axioms were deliberately introduced to the test cases. There were two types of clashes evaluated:

- Trivial clashes (TC) are those that can be detected during the parsing phase. For example, the assertions $\langle a : C \mid [0, 0.2], min, max\rangle$ and $\langle a : C \mid [0.6, 1], min, max\rangle$ would trigger such a clash.

- Non-trivial clashes (NC) are those that can be detected only at the constraint-solving phase. Consider, for example, the assertions $\langle a : C \sqcup D \mid [0, 0.2], min, max\rangle$ and $\langle a : C \sqcap D \mid [0.6, 1], min, max\rangle$. It is not obvious at the parsing phase that these two assertions would trigger a clash. However, such clash can be detected at the constraint-solving phase since there is no solution to the constraints set $\{(max(x_{a:C}, x_{a:D}) \leq 0.2), (min(x_{a:C}, x_{a:D}) \geq 0.6)\}$.

Figures 31, 32, 33, 34, and Tables 17, 18, 19, 20 illustrate the effectiveness of optimized clash detection for the test cases `Simple-Deep`, `Simple-Wide`, `Complex-Deep`, and `Comples-Wide`. Let SpUp denote the speed-up factor, and T_{Off} denote the running time when optimized clash detection is disabled. Also, let T_{TC} (resp. T_{NC}) be the running time for test cases with trivial clash (resp. non-trivial clash), when optimized clash detection is enabled. The speed-up factors are defined as:

$$\text{SpUp}_{Off/TC} = \frac{T_{Off}}{T_{TC}} \tag{5}$$

$$\text{SpUp}_{Off/NC} = \frac{T_{Off}}{T_{NC}} \tag{6}$$

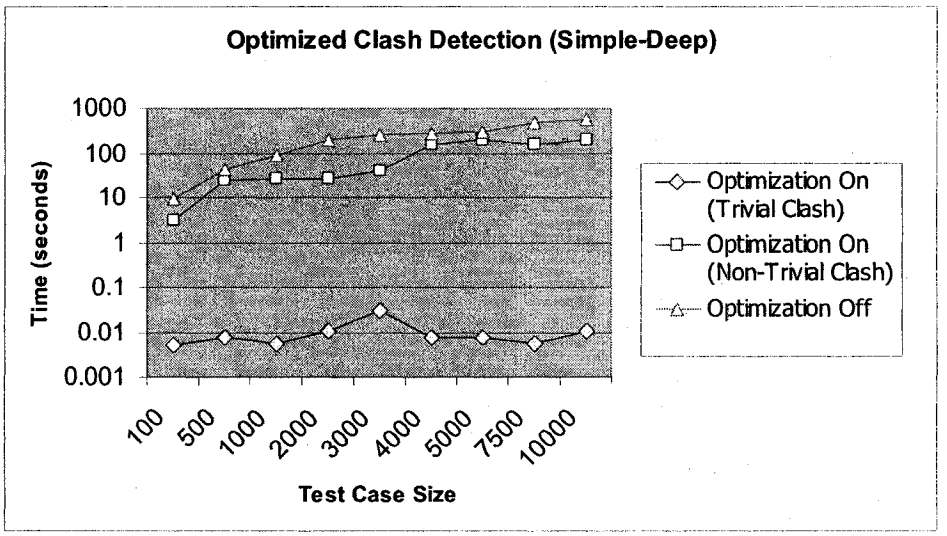

Figure 31: Effect of optimized clash detection - `Simple-Deep`

From the experimental results, it is interesting to notice that:

- When optimized clash detection is enabled, the running time does not appear to be directly correlated with the size of the test cases. This is expected, since the

Test Case	Op. On (TC)	Op. On (NC)	Op. Off	$SpUp_{Off/TC}$	$SpUp_{Off/NC}$
Simple-Deep-100	0.005	2.88	9.39	1817.45	3.26
Simple-Deep-500	0.008	24.31	43.28	5524.91	1.78
Simple-Deep-1000	0.005	25.98	89.88	16852.03	3.46
Simple-Deep-2000	0.010	27.26	199.66	19321.56	7.32
Simple-Deep-3000	0.029	40.43	262.51	9104.48	6.49
Simple-Deep-4000	0.008	157.72	284.37	36302.97	1.80
Simple-Deep-5000	0.008	195.23	301.52	39329.13	1.54
Simple-Deep-7500	0.005	155.41	488.11	91520.00	3.14
Simple-Deep-10000	0.011	199.58	563.56	53587.32	2.82

Table 17: Effect of optimized clash detection - Simple-Deep (in seconds)

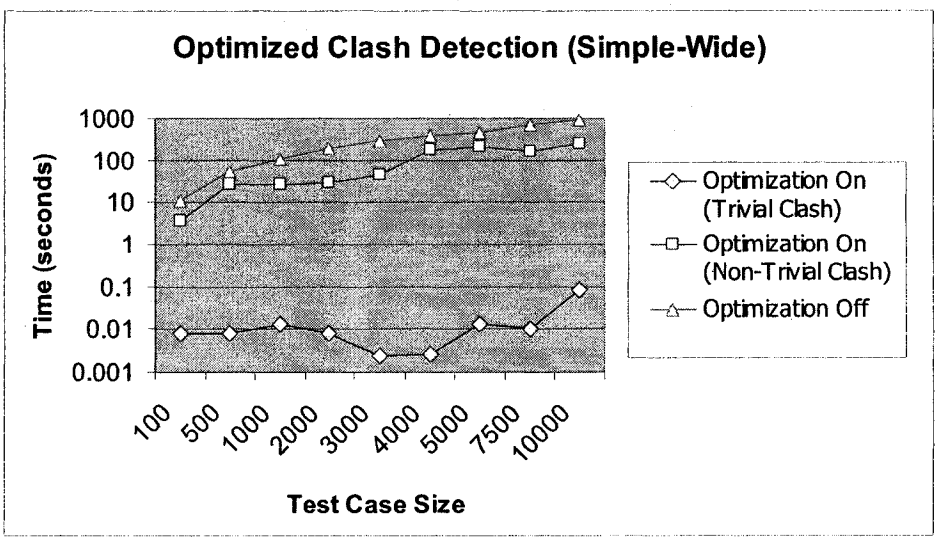

Figure 32: Effect of optimized clash detection - Simple-Wide

Test Case	Op. On (TC)	Op. On (NC)	Op. Off	$SpUp_{Off/TC}$	$SpUp_{Off/NC}$
Simple-Wide-100	0.008	3.53	11.70	1493.38	3.31
Simple-Wide-500	0.008	26.36	51.55	6723.52	1.96
Simple-Wide-1000	0.013	27.03	104.36	8027.42	3.86
Simple-Wide-2000	0.008	30.14	198.47	25887.89	6.59
Simple-Wide-3000	0.002	46.03	297.70	120689.46	6.47
Simple-Wide-4000	0.003	173.96	383.16	143686.56	2.20
Simple-Wide-5000	0.013	214.17	458.91	35300.68	2.14
Simple-Wide-7500	0.010	165.87	737.95	71414.56	4.45
Simple-Wide-10000	0.083	243.46	897.52	10770.19	3.69

Table 18: Effect of optimized clash detection - Simple-Wide (in seconds)

running time depends on where the contradicting assertions/axioms appear in the test case, and/or whether the constraints set that contains the contradicting constraints are solved early or late at the constraint-solving phase.

- Trivial clashes can always be detected quickly (in less than a second for all the test cases evaluated), regardless of the shape, the complexity, and the size of the test cases. In general, it takes no more than the time spent on loading the knowledge base. The speed-up factor for `simple` test cases range from 1493.38 to 143686.56, whereas the speed-up factor for `complex` test cases range from 18188.90 to 441183.33.

- For all the test cases with non-trivial clashes, the performance when optimized clash detection is on is always better than the performance when the optimization is off. This is particularly true for `complex` test cases, where contradicting axioms are present. The speed-up factor for `simple` test cases range from 1.54 to 7.32, and the speed-up factor for `complex` test cases range from 1137.91 to 6385.73.

Test Case	Op. On (TC)	Op. On (NC)	Op. Off	$SpUp_{Off/TC}$	$SpUp_{Off/NC}$
`Complex-Deep-10`	0.005	0.14	23.65	4730.67	172.23
`Complex-Deep-15`	0.007	0.23	36.53	4981.89	158.31
`Complex-Deep-25`	0.010	0.18	60.64	5868.69	343.05
`Complex-Deep-50`	0.011	0.22	108.29	10118.91	498.77
`Complex-Deep-75`	0.015	0.11	171.91	11211.25	1577.12
`Complex-Deep-100`	0.011	0.18	214.57	20435.49	1212.27
`Complex-Deep-125`	0.011	0.19	273.98	25846.74	1410.63
`Complex-Deep-150`	0.013	0.19	339.94	25495.49	1813.55
`Complex-Deep-175`	0.010	0.28	364.80	35882.16	1304.42
`Complex-Deep-200`	0.011	0.14	398.87	37987.38	2837.79

Table 19: Effect of optimized clash detection - `Complex-Deep` (in seconds)

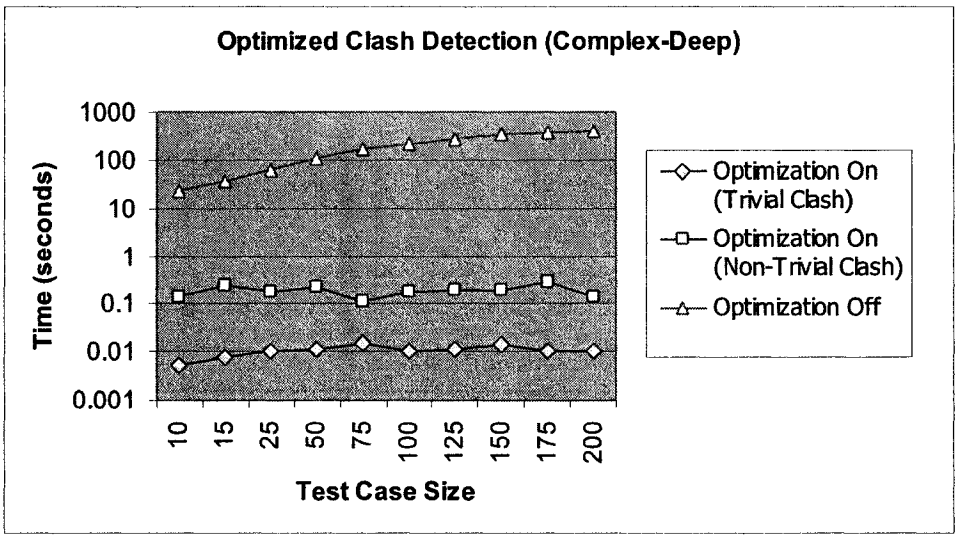

Figure 33: Effect of optimized clash detection - `Complex-Deep`

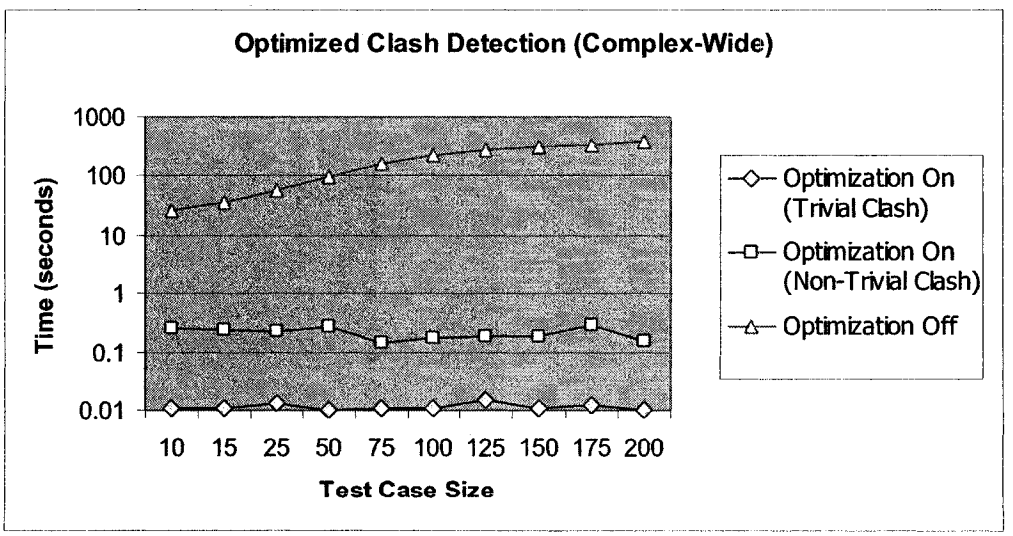

Figure 34: Effect of optimized clash detection - `Complex-Wide`

Test Case	Op. On (TC)	Op. On (NC)	Op. Off	$SpUp_{Off/TC}$	$SpUp_{Off/NC}$
Complex-Wide-10	0.011	0.25	26.32	2467.56	103.76
Complex-Wide-15	0.011	0.24	36.97	3434.09	156.59
Complex-Wide-25	0.013	0.23	58.16	4473.77	255.71
Complex-Wide-50	0.010	0.28	96.40	9640.10	348.86
Complex-Wide-75	0.010	0.14	161.72	15650.24	1123.05
Complex-Wide-100	0.011	0.18	217.80	20743.08	1207.78
Complex-Wide-125	0.015	0.19	272.83	18188.90	1455.54
Complex-Wide-150	0.010	0.19	320.00	30967.52	1692.11
Complex-Wide-175	0.013	0.30	339.98	27198.52	1137.91
Complex-Wide-200	0.010	0.15	370.32	36478.30	2483.49

Table 20: Effect of optimized clash detection - Complex-Wide (in seconds)

In what follows, we present the effect of caching, optimized hashing, and optimized string comparison. Since these optimization techniques do not affect the constraint solving time, we ignore the time spent on writing the constraints to file and solving the constraints.

6.3.4 Effect of Caching

The experimental results presented in this section show only partial effect of the caching techniques, since the first caching technique mentioned in Section 4.2.6 cannot be disabled, and the last caching technique depends mainly on the follow-up user query.

Figures 35, 36 and Tables 21, 22 show the performance evaluation results for the test cases Complex-Deep and Complex-Wide when the caching is on and off. From these results, we observe that:

- When the knowledge base is deep, caching clearly improved the performance. The speed-up factor increases as the test case size increases.

- Wide knowledge bases benefit from caching technique only slightly. Also, unlike the deep knowledge bases, the gap between the optimized case and the non-optimized case does not increase as the test case size increases.

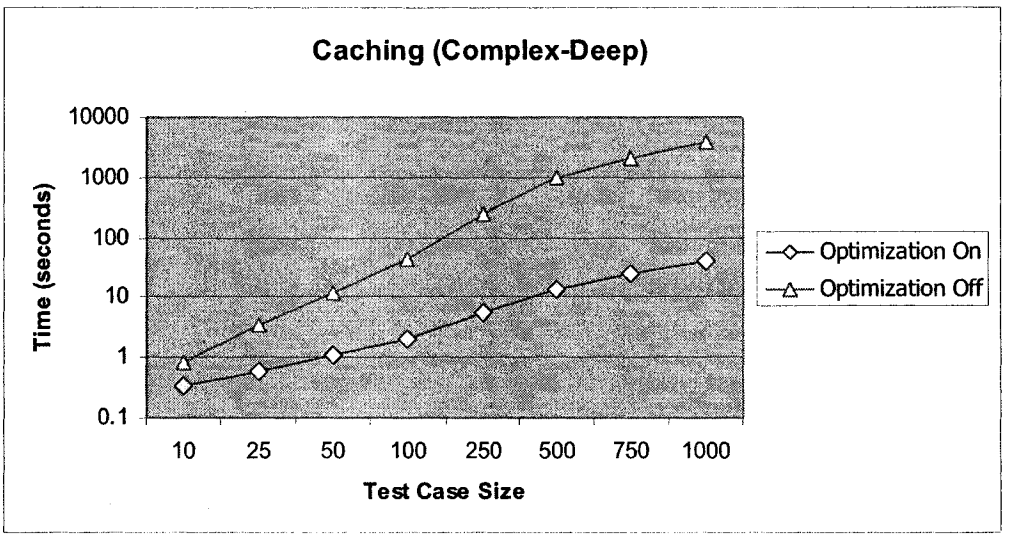

Figure 35: Effect of caching - `Complex-Deep`

Test Case	Optimization On	Optimization Off	Speedup
`Complex-Deep-10`	0.33	0.82	2.46
`Complex-Deep-25`	0.59	3.38	5.74
`Complex-Deep-50`	1.08	11.55	10.67
`Complex-Deep-100`	1.97	42.75	21.68
`Complex-Deep-250`	5.42	251.29	46.35
`Complex-Deep-500`	13.14	968.54	73.74
`Complex-Deep-750`	24.28	2155.69	88.80
`Complex-Deep-1000`	39.19	3914.91	99.90

Table 21: Effect of caching - `Complex-Deep` (in seconds)

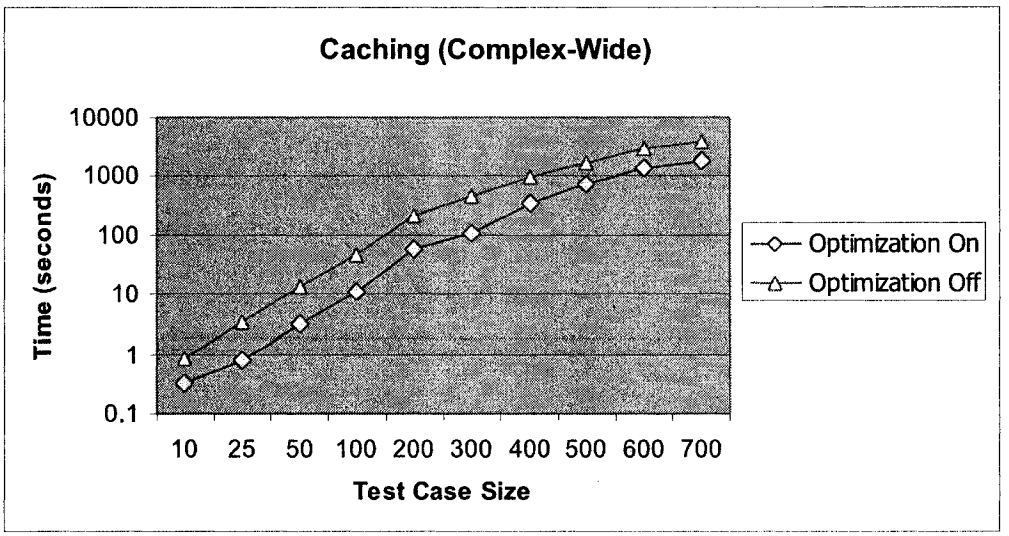

Figure 36: Effect of caching - `Complex-Wide`

Test Case	Optimization On	Optimization Off	Speedup
`Complex-Wide-10`	0.33	0.84	2.53
`Complex-Wide-25`	0.77	3.46	4.49
`Complex-Wide-50`	3.08	13.64	4.43
`Complex-Wide-100`	10.96	47.30	4.31
`Complex-Wide-200`	57.08	213.64	3.74
`Complex-Wide-300`	104.85	451.93	4.31
`Complex-Wide-400`	337.81	952.98	2.82
`Complex-Wide-500`	712.42	1670.01	2.34
`Complex-Wide-600`	1312.31	2818.16	2.15
`Complex-Wide-700`	1757.80	3721.92	2.12

Table 22: Effect of caching - `Complex-Wide` (in seconds)

6.3.5 Effect of Optimized Hashing

Since it is not feasible to disable optimized hashing by changing the underlying data structure, hash sets, to other data structures such as the ordinary sets, we simulated this effect by changing the way hash code values are computed. For example, instead of calculating the hash code of a concept assertion based on the hash codes of the various components that form a concept assertion, we assigned the assertion type as the hash code value when optimized hashing is disabled.

As usual, we tested the effect of optimized hashing with both deep and wide knowledge bases, as shown in Figures 37, 38 and Tables 23, 24 respectively. The interesting things to note here are:

- Optimized hashing improved the performance of deep knowledge bases slightly. In addition, the speed-up factor remains almost constant as the test case size increases.

- When the knowledge base is wide, optimized hashing significantly improved the performance. Indeed, larger knowledge bases show more performance gain from this optimization.

6.3.6 Effect of Optimized String Comparison

Optimized string comparison is the technique that has the least effect on the URDL performance. As shown in Figure 39 and Table 25, it has almost no effect on deep

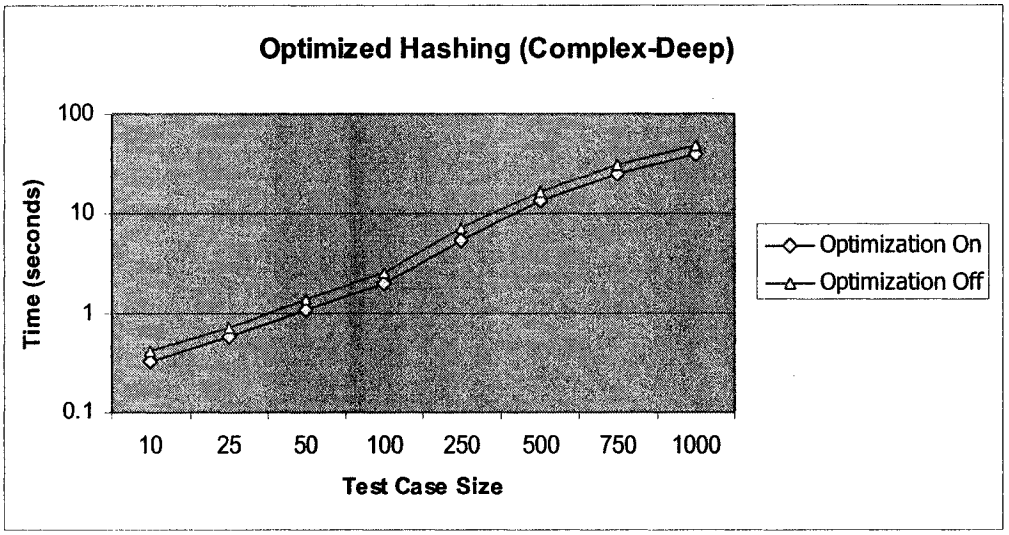

Figure 37: Effect of optimized hashing - `Complex-Deep`

Test Case	Optimization On	Optimization Off	Speedup
`Complex-Deep-10`	0.33	0.42	1.26
`Complex-Deep-25`	0.59	0.72	1.23
`Complex-Deep-50`	1.08	1.41	1.30
`Complex-Deep-100`	1.97	2.59	1.32
`Complex-Deep-250`	5.42	7.10	1.31
`Complex-Deep-500`	13.14	16.65	1.27
`Complex-Deep-750`	24.28	29.96	1.23
`Complex-Deep-1000`	39.19	48.62	1.24

Table 23: Effect of optimized hashing - `Complex-Deep` (in seconds)

Test Case	Optimization On	Optimization Off	Speedup
`Complex-Wide-10`	0.33	0.42	1.28
`Complex-Wide-25`	0.77	2.38	3.09
`Complex-Wide-50`	3.08	26.26	8.52
`Complex-Wide-100`	10.96	139.89	12.76
`Complex-Wide-200`	57.08	1291.87	22.63
`Complex-Wide-300`	104.85	2874.91	27.42
`Complex-Wide-400`	337.81	15231.64	45.09
`Complex-Wide-500`	712.42	49833.11	69.95
`Complex-Wide-600`	1312.31	117686.47	89.68
`Complex-Wide-700`	1757.80	171622.35	97.63

Table 24: Effect of optimized hashing - `Complex-Wide` (in seconds)

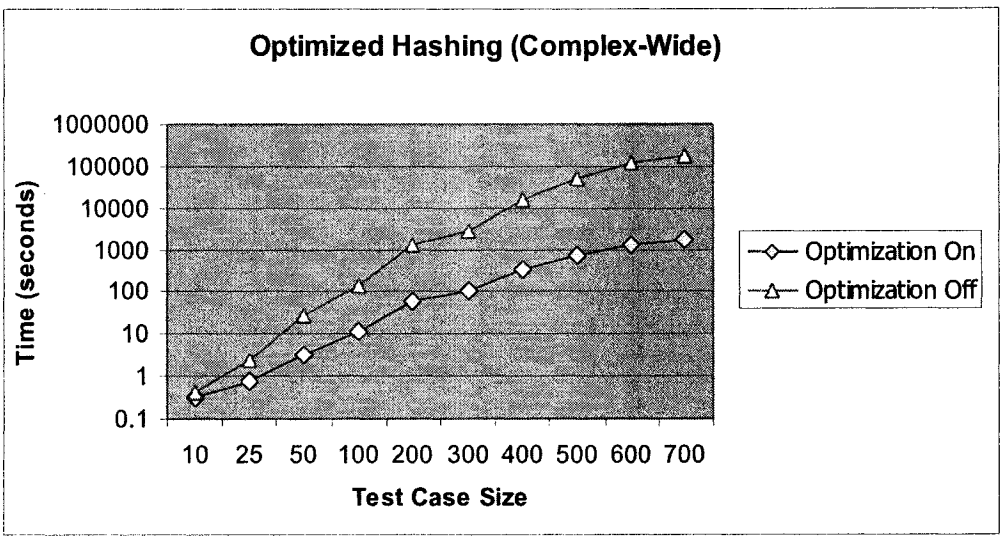

Figure 38: Effect of optimized hashing - `Complex-Wide`

knowledge bases. This is expected, since when the knowledge is not wide, the number of string comparison is low.

Test Case	Optimization On	Optimization Off	Speedup
`Complex-Deep-10`	0.33	0.34	1.01
`Complex-Deep-25`	0.59	0.59	1.01
`Complex-Deep-50`	1.08	1.09	1.00
`Complex-Deep-100`	1.97	1.97	1.00
`Complex-Deep-250`	5.42	5.43	1.00
`Complex-Deep-500`	13.14	13.15	1.00
`Complex-Deep-750`	24.28	24.38	1.00
`Complex-Deep-1000`	39.19	39.85	1.02

Table 25: Effect of optimized string comparison - `Complex-Deep` (in seconds)

We also studied the effect of this optimization on wide knowledge bases. It turned out that optimized string comparison improved the performance, as shown in Figure 40 and Table 26. Indeed, the larger the test case, the more speed-up is gained from this optimization.

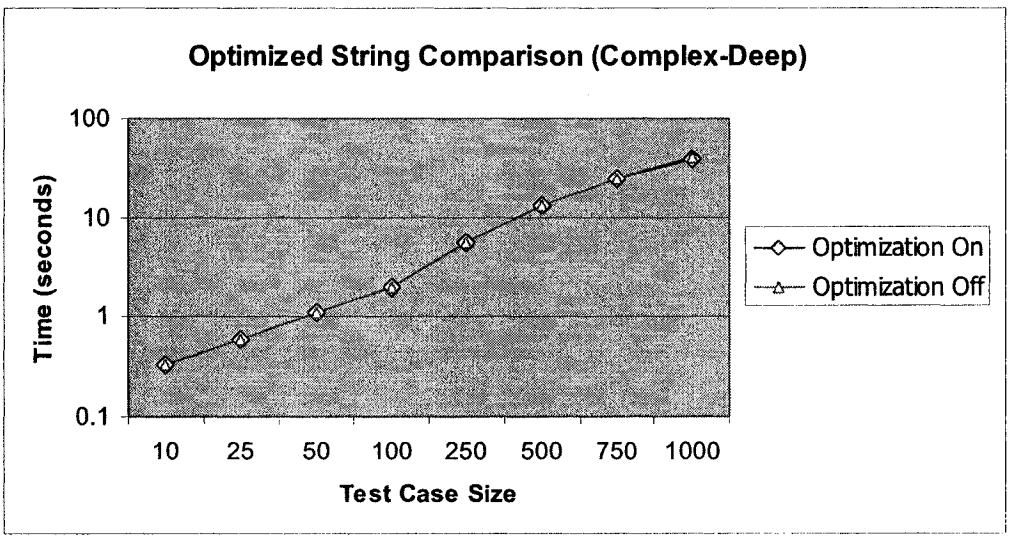

Figure 39: Effect of optimized string comparison - `Complex-Deep`

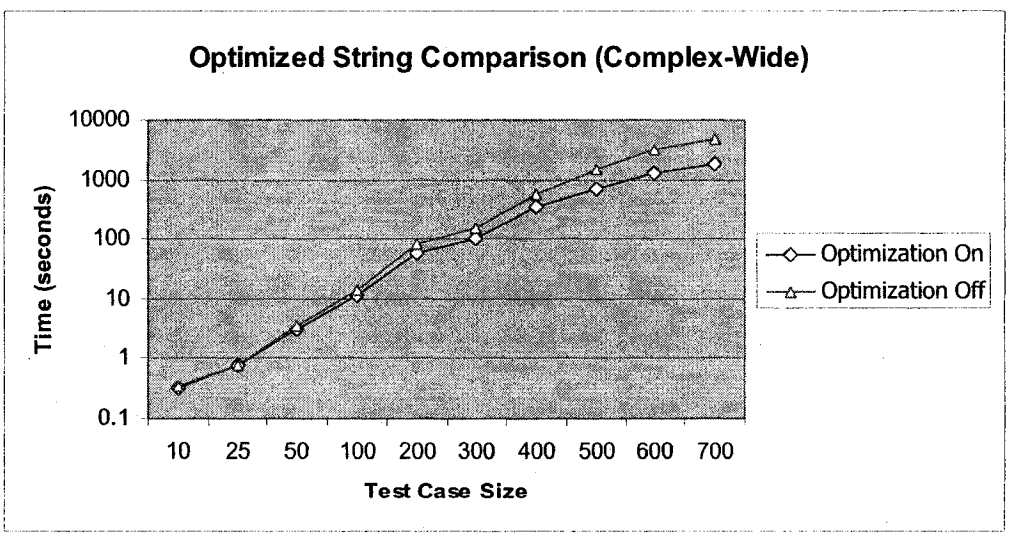

Figure 40: Effect of optimized string comparison - `Complex-Wide`

Test Case	Optimization On	Optimization Off	Speedup
Complex-Wide-10	0.33	0.34	1.03
Complex-Wide-25	0.77	0.80	1.04
Complex-Wide-50	3.08	3.46	1.12
Complex-Wide-100	10.96	13.96	1.27
Complex-Wide-200	57.08	80.10	1.40
Complex-Wide-300	104.85	154.89	1.48
Complex-Wide-400	337.81	561.98	1.66
Complex-Wide-500	712.42	1457.30	2.05
Complex-Wide-600	1312.31	3195.40	2.43
Complex-Wide-700	1757.80	4815.02	2.74

Table 26: Effect of optimized string comparison - Complex-Wide (in seconds)

6.3.7 Overall Optimization Effect

The experimental results showed in the previous subsections were measured by disabling one of the optimizations at a time. We now present the overall effect of the proposed optimization techniques.

Tables 27, 28, 29, and 30 compare the performance when all the optimizations are on and off. It is interesting to note that the performance when all the optimization techniques are disabled is only slightly worse than the performance when partitioning based on connectivity is disabled (see Section 6.3.1). This is not surprising because, when the ABox is not partitioned, URDL can handle only small knowledge bases due to the external constraint solver limitation. As we observed in all of the optimization results presented earlier, the speed-ups from the optimizations are not significant for small knowledge bases, and the performance gain increases as the knowledge base size increases.

In order to better study the overall effect of the optimization techniques, we disregard the constraint-solving factor. That is, the external constraint solver is not

Test Case	Optimization On	Optimization Off	Speedup
Simple-Deep-100	3.94	1.29	0.33
Simple-Deep-250	6.64	6.67	1.00
Simple-Deep-500	20.28	24.33	1.20
Simple-Deep-750	31.25	53.31	1.71
Simple-Deep-1000	41.22	92.66	2.25
Simple-Deep-1100	43.13	111.90	2.59
Simple-Deep-1300	48.09	T/O	N/A
Simple-Deep-1500	56.85	T/O	N/A
Simple-Deep-1750	69.29	T/O	N/A
Simple-Deep-2000	75.75	T/O	N/A
Simple-Deep-3000	121.57	T/O	N/A
Simple-Deep-4000	156.77	T/O	N/A
Simple-Deep-5000	223.28	T/O	N/A
Simple-Deep-7500	336.54	T/O	N/A
Simple-Deep-10000	461.22	T/O	N/A

Table 27: Overall optimization effect - Simple-Deep (in seconds)

Test Case	Optimization On	Optimization Off	Speedup
Simple-Wide-100	4.49	1.32	0.29
Simple-Wide-250	6.50	6.89	1.06
Simple-Wide-500	24.15	26.50	1.10
Simple-Wide-750	37.43	55.59	1.49
Simple-Wide-1000	47.50	98.34	2.07
Simple-Wide-1100	55.31	119.33	2.16
Simple-Wide-1300	62.70	T/O	N/A
Simple-Wide-1500	66.30	T/O	N/A
Simple-Wide-1750	75.34	T/O	N/A
Simple-Wide-2000	84.85	T/O	N/A
Simple-Wide-3000	120.28	T/O	N/A
Simple-Wide-4000	162.06	T/O	N/A
Simple-Wide-5000	185.51	T/O	N/A
Simple-Wide-7500	333.50	T/O	N/A
Simple-Wide-10000	434.73	T/O	N/A

Table 28: Overall optimization effect - Simple-Wide (in seconds)

Test Case	Optimization On	Optimization Off	Speedup
Complex-Deep-10	13.64	65.61	4.81
Complex-Deep-11	15.18	79.79	5.26
Complex-Deep-12	16.02	90.32	5.64
Complex-Deep-13	17.18	T/O	N/A
Complex-Deep-14	18.20	T/O	N/A
Complex-Deep-15	19.80	T/O	N/A
Complex-Deep-25	34.79	T/O	N/A
Complex-Deep-50	69.54	T/O	N/A
Complex-Deep-75	102.91	T/O	N/A
Complex-Deep-100	142.63	T/O	N/A
Complex-Deep-125	197.51	T/O	N/A

Table 29: Overall optimization effect - Complex-Deep (in seconds)

Test Case	Optimization On	Optimization Off	Speedup
Complex-Wide-10	15.54	66.05	4.25
Complex-Wide-11	17.09	80.13	4.69
Complex-Wide-12	18.05	90.19	5.00
Complex-Wide-13	22.24	T/O	N/A
Complex-Wide-14	23.06	T/O	N/A
Complex-Wide-15	24.61	T/O	N/A
Complex-Wide-25	40.30	T/O	N/A
Complex-Wide-50	89.10	T/O	N/A
Complex-Wide-75	141.61	T/O	N/A
Complex-Wide-100	209.94	T/O	N/A
Complex-Wide-125	242.60	T/O	N/A

Table 30: Overall optimization effect - Complex-Wide (in seconds)

called, and the constraints are not written to files. The experimental results show that the proposed optimizations improved the performance of URDL significantly. For deep knowledge bases (see Figure 41 and Table 31), the speed-up factor ranges from 2.83 to 100.22. The performance gain for wide knowledge bases is even greater (see Figure 42 and Table 32), with the speed-up factor ranging from 3.17 to 267.26. Consider, for example, the test case `Complex-Wide-700`. It takes roughly 30 minutes to run this test case with all the optimizations enabled, while it takes 5 days 10 hours and 30 minutes to run the same test case when the optimizations are disabled.

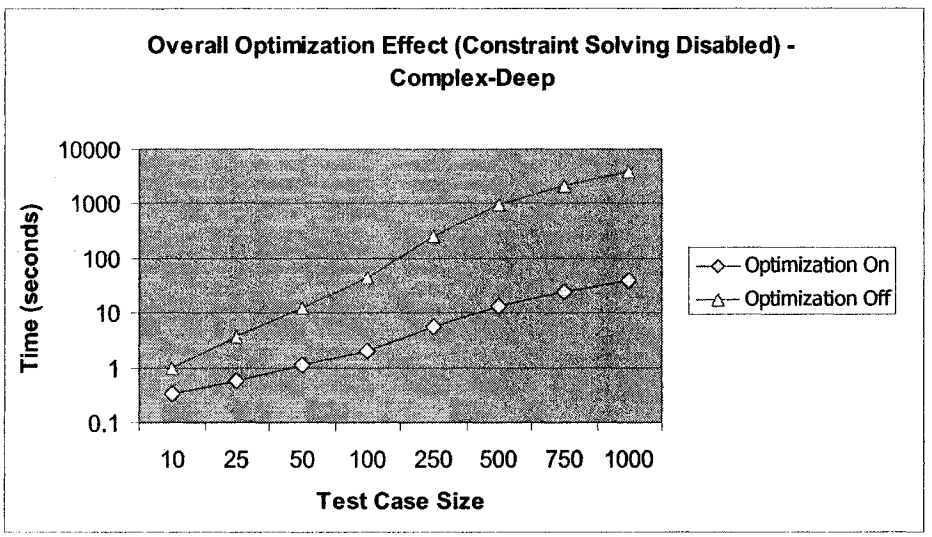

Figure 41: Overall optimization effect (constraint solving disabled) - `Complex-Deep`

6.4 Summary and Concluding Remarks

In this chapter, we studied the performance of URDL and the effectiveness of the proposed optimization techniques. We observed that different characteristics of the

Test Case	Optimization On	Optimization Off	Speedup
Complex-Deep-10	0.33	0.94	2.83
Complex-Deep-25	0.59	3.68	6.26
Complex-Deep-50	1.08	12.22	11.29
Complex-Deep-100	1.97	43.64	22.14
Complex-Deep-250	5.42	254.39	46.92
Complex-Deep-500	13.14	975.79	74.29
Complex-Deep-750	24.28	2180.04	89.80
Complex-Deep-1000	39.19	3927.80	100.22

Table 31: Overall optimization effect (constraint solving disabled) - Complex-Deep (in seconds)

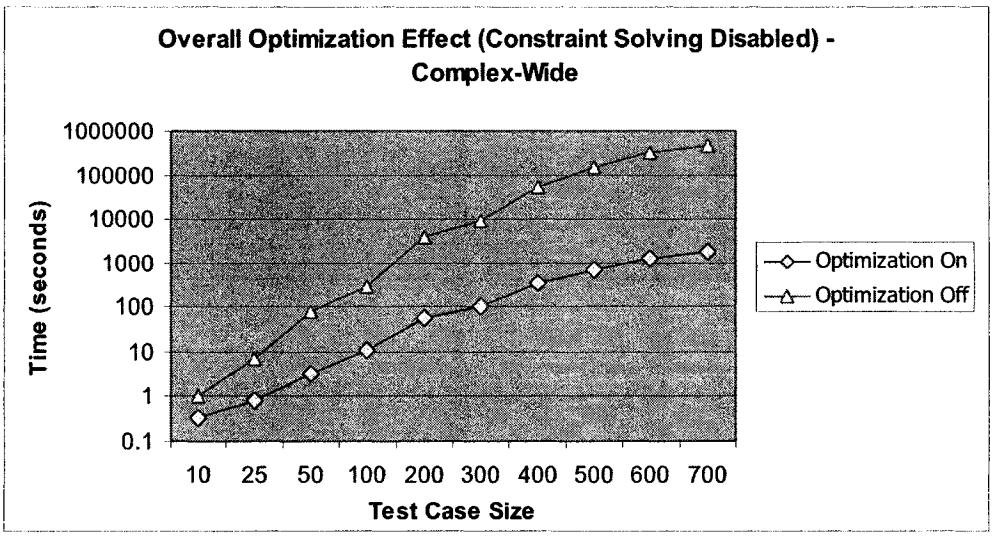

Figure 42: Overall optimization effect (constraint solving disabled) - Complex-Wide

Test Case	Optimization On	Optimization Off	Speedup
Complex-Wide-10	0.33	1.05	3.17
Complex-Wide-25	0.77	7.13	9.25
Complex-Wide-50	3.08	78.46	25.47
Complex-Wide-100	10.96	289.76	26.43
Complex-Wide-200	57.08	3759.39	65.86
Complex-Wide-300	104.85	9637.61	91.92
Complex-Wide-400	337.81	54365.23	160.93
Complex-Wide-500	712.42	146990.19	206.33
Complex-Wide-600	1312.31	321881.03	245.28
Complex-Wide-700	1757.80	469795.34	267.26

Table 32: Overall optimization effect (constraint solving disabled) - Complex-Wide (in seconds)

test cases, such as the certainty domain and the combination functions, can affect the performance of URDL. We also noted constraint solving to be the main bottleneck for URDL. We tackled this weakness by partitioning the constraints into independent subsets. Our experimental results showed that this optimization not only improved the performance in terms of the running time, but also improved the constraints' solvability significantly. We also noted from the experimental results that optimized Individual Group creation and caching techniques work well when the knowledge base is deep, whereas optimized hashing and optimized string comparison work well when the knowledge base is wide. In addition to dealing with consistent knowledge bases, optimized clash detection was also found to be very effective in dealing with inconsistent knowledge bases, with speed-up factor ranging from 1.54 to 441183. For most test cases, larger knowledge bases benefit more from the optimizations. We also noted that the overall effect of the optimizations is significant, with the speed-up factor ranging from 2.83 to 267.26.

Although all the experiments reported in this chapter were conducted under Windows XP, we had also tested URDL under Linux (Fedora). The experimental results showed that the performances on both platforms were roughly the same. Note also that, in order to enable or disable the optimizations, we had included many `if` statements in URDL. Although this was a slight disadvantage to the performance of URDL, we do not expect much effect on the experimental results.

Chapter 7

Conclusions and Future Research

In order to handle many real-world Semantic Web applications, it is essential not only to model and reason with uncertainty knowledge, but also do so efficiently. The goal of this thesis was to develop a DL-based framework that can represent and reason with uncertainty knowledge following a generic approach, and to study query optimization in this context. In this chapter, the work done towards this goal and the contributions of this thesis are summarized. We also present some directions for future research.

7.1 Conclusions

We summarize the contributions of this thesis by assessing the extent to which the objectives set out in Section 1.4 have been met.

The first objective of this thesis was to propose a framework that extends the DL $\mathcal{ALC}$ so that uncertainty knowledge can be represented and reasoned with in

a generic way. To fulfill this goal, we proposed the $\mathcal{ALC}_U$ framework in Chapter 3. The $\mathcal{ALC}_U$ framework extended each component of the $\mathcal{ALC}$ framework, namely the description language, the knowledge base, and the reasoning procedure, while abstracting away the notion of uncertainty in the extension. Inspired by the parametric framework for deductive database with uncertainty, we modeled various forms of certainty values, assuming that they form a certainty lattice. We then defined the semantics of the description language using the combination functions together with the certainty lattice. To ensure admissibility, we also identified a set of properties which the combination functions must satisfy. We defined the $\mathcal{ALC}_U$ knowledge base by associating with each axiom and assertion a certainty value and a pair of combination functions, used to interpret the concepts that appear in the axiom/assertion. We also presented a sound, complete, and terminating tableau-based reasoning procedure for $\mathcal{ALC}_U$, which derives a set of assertions and a set of constraints in the form of linear/nonlinear equations/inequations to capture the semantics of the uncertainty knowledge base. The unique feature of the proposed $\mathcal{ALC}_U$ framework is that, by simply tuning the combination functions associated with the axioms and assertions, different notions of uncertainty can be modeled and reasoned with, using a single reasoning procedure.

With the $\mathcal{ALC}_U$ framework defined, we then dealt with the second objective of this thesis in Chapter 4 by investigating suitable optimization techniques in our context. We adapted many optimization techniques from standard DL systems so that they take into account the presence of uncertainty. These techniques include lexical

normalization, concept simplification, partitioning based on connectivity as Individual Groups, optimized Individual Group creation, and caching. We noted that not all optimization techniques developed for the standard DL systems can be applied to the $\mathcal{ALC}_U$ framework, and even if they could, care must be taken when applying these techniques due to the presence of uncertainty. In addition to the optimization techniques from standard DL systems, we also reused the techniques that are commonly used in software systems, including optimized hashing and optimized string comparison. Finally, we proposed new optimization techniques to deal with the certainty values and the uncertainty constraints generated by the reasoning procedure. In particular, since the number of the generated constraints is usually large, we partitioned these constraints into independent subsets before they are fed into the constraint solver, so that they can be solved independently. In addition, we optimized the handling of inconsistent knowledge bases by detecting contradicting assertions and/or axioms as early as possible so that we do not perform inferences or solve constraints unnecessarily. The proposed techniques could also be adapted and used in other constraint-based systems.

The last objective was fulfilled by developing a running prototype URDL and measuring its performance, as described in Chapters 5 and 6, respectively. To the best of our knowledge, URDL is the first reasoner that employs optimization techniques in the DL/uncertainty context. URDL consists of five main components. The Reasoner Controller gets the input from the user, and delegates tasks to Configuration Loader, Parser, Inference Engine, and Constraint Solver in order to fulfill the user query. It

then returns the query results back to the user. The Configuration Loader loads and stores user preferences from the configuration file. The Parser analyzes the user input, and stores the parsed axioms, assertions, and combination functions into the knowledge base. The Inference Engine applies the reasoning procedure to the knowledge base and stores the inferred assertions and constraints into the knowledge base. Finally, the Constraint Solver solves the set of constraints generated by the Inference Engine by calling an external constraint solver, RealPaver. Note that the main difference between URDL and other standard DL systems is that, URDL relies on both the Inference Engine and the Constraint Solver to complete the reasoning tasks, whereas the standard DL systems need only the Inference Engine since they do not generate linear and/or nonlinear constraints during the completion rule application.

Through performance evaluation of URDL, we learned that different properties of the knowledge bases, such as the certainty domain and the combination functions, can affect the running time. We also identified constraint solving as the main bottleneck for URDL. All the optimization techniques presented in Chapter 4 were shown to be effective, especially when dealing with large knowledge bases. Among these techniques, the newly proposed optimization which partitioned the ABox into Assertion Groups was particularly useful. It not only improved the performance in terms of the running time, but also increased the solvability of the constraints significantly. We also noted that the overall effect of the optimizations was significant, with the speed-up factor ranging from 2.83 to 267.26. In addition to dealing with consistent knowledge bases, optimized clash detection was found to be very effective in dealing

with inconsistent knowledge bases, with the speed-up factor ranging from 1.54 to 441183.

7.2 Future Research Directions

The theoretical as well as practical development in this thesis suggest some promising avenues for future research.

For theoretical aspects, it would be interesting to extend the $\mathcal{ALC}_U$ framework to support more expressive fragments of DLs, such as $\mathcal{SHOIN}$. The DL $\mathcal{SHOIN}$ is the logic foundation for the ontology language OWL DL. In addition to the language constructors supported by $\mathcal{ALC}$, the DL $\mathcal{SHOIN}$ also allows transitive role, role hierarchy, nominal, inverse role, and unqualified number restriction be expressed. The challenge for this extension would be to come up with the completion rules for these additional description language constructors, including the appropriate constraints to denote their semantics.

Another interesting extension to the $\mathcal{ALC}_U$ framework would be to support other forms of uncertainty. Currently, we keep the description language syntax the same as the standard DL while extending only its semantics. However, since probabilistic reasoning usually requires extra information about the events, their relationships, and the facts in the world, it would require syntactical extension to the description language in order to model knowledge bases with more probability modes, such as positive/negative correlation [LS94] and conditional probability [KLP97, GL02b].

The challenge here would be investigating whether it is feasible to extend the syntax of the description language generically to support these uncertainty formalisms, and how such extension can fit into the existing $\mathcal{ALC}_U$ framework.

For practical aspects, it would be interesting to further optimize the reasoning procedure. For example, since constraint-solving is the dominant cost for the reasoning process, in case we have multiple Assertion Groups, we could solve them concurrently by running multiple threads or on different computers. Another related optimization would be to reduce the number of constraints or the number of variables in the constraints generated during the reasoning procedure. These methods are expected to greatly enhance the performance.

To sum up, in this thesis, we proposed the $\mathcal{ALC}_U$ framework which allows uncertainty knowledge be expressed and reasoned with in a generic way. Optimization techniques were developed to improve the performance of the uncertainty reasoning in this context. We also developed a prototype for the proposed framework to show its practicality as well as the effectiveness of the optimization techniques. It is hoped that the framework proposed in this thesis can serve as a foundation for the ongoing research on DL-based uncertainty reasoning in the Semantic Web.

Bibliography

[Baa90] Baader, F. Augmenting concept languages by transitive closure of roles: An alternative to terminological cycles. Technical Report RR-90-13, Deutsches Forschungszentrum für Künstliche Intelligenz GmbH, 1990.

[Bac90] Bacchus, F., editor. *Representing and Reasoning with Probabilistic Knowledge - A Logical Approach to Probabilities.* MIT Press, Cambridge, Massachusetts, 1990.

[BCM+03] Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[BDS93] Buchheit, M., Donini, F. M., and Schaerf, A. Decidable reasoning in terminological knowledge representation systems. *Journal of Artificial Intelligence Research*, 1:109–138, 1993.

[BF00] Berners-Lee, T. and Fischetti, M. *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor.* HarperCollins, New York, 2000.

[BH91] Baader, F. and Hollunder, B. KRIS: Knowledge representation and inference system. *SIGART Bulletin*, 2(3):8–14, 1991.

[BHL01] Berners-Lee, T., Hendler, J., and Lassila, O. The Semantic Web. *Scientific American*, 284(5), May 2001.

[BHS02] Baader, F., Horrocks, I., and Sattler, U. Description logics for the semantic web. *KI – Künstliche Intelligenz*, 16(4):57–59, 2002.

[BHS07] Baader, F., Horrocks, I., and Sattler, U. Description Logics. In Frank van Harmelen, Vladimir Lifschitz, and Bruce Porter, editors, *Handbook of Knowledge Representation*. Elsevier, 2007. To appear.

[BL06] Baader, F. and Lutz, C. Description logic. In Patrick Blackburn, Johan van Benthem, and Frank Wolter, editors, *The Handbook of Modal Logic*. Elsevier, 2006.

[BP02] Beck, H. and Pinto, H. S. Overview of approach, methodologies, standards, and tools for ontologies. Technical report, The Agricultural Ontology Service, United Nations (UN) Food and Agriculture Organization (FAO), 2002.

[BS07] Bobillo, F. and Straccia, U. A fuzzy description logic with product t-norm. In *Proceedings of the IEEE International Conference on Fuzzy Systems (Fuzz IEEE-07)*, pages 652–657. IEEE Computer Society, 2007.

[Cop07] Copeland, T. *Generating Parsers with JavaCC*. Centennial Books, Alexandria, VA., 2007.

[CPL97] Cadoli, M., Palopoli, L., and Lenzerini, M. Datalog and description logics: Expressive power. In *Workshop on Database Programming Languages*, pages 281–298, 1997.

[DAM00] About DAML, 2000. URL: http://www.daml.org/about.html (Last visited September 22nd, 2006).

[DLL62] Davis, M., Logemann, G., and Loveland, D. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.

[DP04] Ding, Z. and Peng, Y. A probabilistic extension to ontology language OWL. In *Proceedings of the 37th Hawaii International Conference on System Sciences (HICSS-37)*, Big Island, Hawaii, January 2004.

[DPP04] Ding, Z., Peng, Y., and Pan, R. A Bayesian approach to uncertainty modeling in OWL ontology. In *Proceedings of 2004 International Conference on Advances in Intelligent Systems - Theory and Applications*, Luxembourg, November 2004.

[DS05] Durig, M. and Studer, T. Probabilistic ABox reasoning: Preliminary results. In *Proceedings of the International Workshop on Description Logics (DL'05)*, pages 104–111, Edinburgh, Scotland, UK, 2005.

[Dur05] Durig, M. PALC: Extending $\mathcal{ALC}$ ABoxes with probabilities. Master's thesis, Institute of Computer Science and Applied Mathematics, Faculty of Science, University of Bern, 2005.

[FaC] FaCT++. URL: http://owl.man.ac.uk/factplusplus/ (Last visited May 3rd, 2007).

[GHVD03] Grosof, N., Horrocks, I., Volz, R., and Decker, S. Description logic programs: Combining logic programs with description logic. In *Proceeding of the Twelfth International World Wide Web Conference (WWW 2003)*, pages 48–57. ACM, 2003.

[GL02a] Giugno, R. and Lukasiewicz, T. P-$\mathcal{SHOQ}$(D): A probabilistic extension of $\mathcal{SHOQ}$(D) for probabilistic ontologies in the Semantic Web. Technical Report INFSYS 1843-02-06, Technische Universität Wien, Wien, Austria, April 2002.

[GL02b] Giugno, R. and Lukasiewicz, T. P-$\mathcal{SHOQ}$(D): A probabilistic extension of $\mathcal{SHOQ}$(D) for probabilistic ontologies in the Semantic Web. In *Proceedings of the European Conference on Logics in Artificial Intelligence*, pages 86–97, Cosenza, Italy, 2002. Springer-Verlag. Lecture Notes In Computer Science; Vol. 2424.

[GP91] Goldszmidt, M. and Pearl, J. On the consistency of defeasible databases. *Artificial Intelligence*, 52(2):121 – 149, December 1991.

[Gru93] Gruber, T. R. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[GS96] Giunchiglia, F. and Sebastiani, R. Building decision procedures for modal logics from propositional decision procedure - the case study of modal k. In *Conference on Automated Deduction*, pages 583–597, 1996.

[Hei94] Heinsohn, J. Probabilistic description logics. In *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence (UAI'94)*, pages 311–318, San Francisco, CA, 1994. Morgan Kaufmann.

[HFB+00] Horrocks, I., Fensel, D., Broekstra, J., Decker, S., Erdmann, M., Goble, C., van Harmelen, F., Klein, M., Staab, S., Studer, R., and Motta, E. OIL: The Ontology Inference Layer. Technical Report IR-479, Vrije Universiteit Amsterdam, Faculty of Sciences, September 2000.

[HKS02] Hölldobler, S., Khang, T. D., and Störr, H.-P. A fuzzy description logic with hedges as concept modifiers. In *Proceedings of the 3rd International Conference on Intelligent Technologies and 3rd Vietnam-Japan Symposium on Fuzzy Systems and Applications*, pages 25–34, Hanoi, Vietnam, 2002. Science and Technics Publishing House.

[Hod01] Hodges, W. *Classical Logic I: First Order Logic.* Blackwell, 2001.

[Hol94] Hollunder, B. An alternative proof method for possibilistic logic and its application to terminological logics. In *Proceedings of the 10th Annual Conference on Uncertainty in Artificial Intelligence*, pages 327–335, San Francisco, CA, 1994. Morgan Kaufmann.

[Hor97a] Horrocks, I. Optimisation techniques for expressive description logics. Technical Report UMCS-97-2-1, University of Manchester, Department of Computer Science, 1997.

[Hor97b] Horrocks, I. *Optimising Tableaux Decision Procedures for Description Logics.* PhD thesis, University of Manchester, 1997.

[HPS] Haarslev, V., Pai, H.I., and Shiri, N. A constraint-based reasoning procedure for description logic with uncertainty. *Submitted to International Journal of Approximate Reasoning.*

[HPS05] Haarslev, V., Pai, H.I., and Shiri, N. A generic framework for description logics with uncertainty. In *Proceedings of the 2005 Workshop on Uncertainty Reasoning for the Semantic Web (URSW) at the 4th International Semantic Web Conference*, pages 77–86, Galway, Ireland, November 2005.

[HPS06a] Haarslev, V., Pai, H.I., and Shiri, N. Completion rules for uncertainty reasoning with the description logic $\mathcal{ALC}$. In *Proceedings of the Canadian Semantic Web Working Symposium - Series: Semantic Web and Beyond: Computing for Human Experience, Vol. 4*, pages 205–225, Quebec City, Canada, June 2006. Springer Verlag.

[HPS06b] Haarslev, V., Pai, H.I., and Shiri, N. Uncertainty reasoning in description logics: A generic approach. In *Proceedings of the 19th International FLAIRS Conference*, pages 818–823, Melbourne Beach, Florida, May 2006. AAAI Press.

[HPS07] Haarslev, V., Pai, H.I., and Shiri, N. Optimizing tableau reasoning in $\mathcal{ALC}$ extended with uncertainty. In *Proceedings of the International Workshop on Description Logics (DL'07)*, pages 307–314, Brixen-Bressanone, Italy, June 2007.

[HPS09] Haarslev, V., Pai, H.I., and Shiri, N. Semantic web uncertainty management. In *Encyclopedia of Information Science and Technology, 2nd edition.* Information Science Reference, 2009.

[HST99] Horrocks, I., Sattler, U., and Tobies, S. A description logic with transitive and converse roles, role hierarchies and qualifying number restrictions. LTCS-Report 99-08, LuFg Theoretical Computer Science, RWTH Aachen, Germany, 1999.

[HST00] Horrocks, I., Sattler, U., and Tobies, S. Reasoning with individuals for the description logic $\mathcal{SHIQ}$. In David McAllester, editor, *Proceedings of the 17th International Conference on Automated Deduction (CADE 2000)*, volume 1831 of *Lecture Notes in Computer Science*, pages 482–496. Springer, 2000.

[HV05] Haase, P. and Völker, J. Ontology learning and reasoning - dealing with uncertainty and inconsistency. In *Proceedings of Uncertainty Reasoning for the Semantic Web*, pages 45–55, Galway, Ireland, November 2005.

[Jae94a] Jaeger, M. A probabilistic extension of terminological logics. Research Report MPI-I-94-208, Max-Planck-Institut für Informatik, March 1994.

[Jae94b] Jaeger, M. Probabilistic reasoning in terminological logics. In *Proceedings of the 4th International Conference on Principles of Knowledge Representation and Reasoning (KR'94)*, pages 305–316, 1994.

[Jav] JavaCC. URL: https://javacc.dev.java.net/ (Last visited May 21st, 2007).

[KG] Racer Systems GmbH Co. & KG. RacerPro. URL: http://www.racer-systems.com/ (Last visited June 22nd, 2007).

[KLP97] Koller, D., Levy, A. Y., and Pfeffer, A. P-CLASSIC: A tractable probablistic description logic. In *Proceedings of the 14th National Conference on Artificial Intelligence*, pages 390–397, Providence, Rhode Island, July 1997. AAAI Press.

[Lab] Mindswap Lab. Pellet. URL: http://pellet.owldl.com/ (Last visited August 2nd, 2007).

[LS94] Lakshmanan, L.V.S. and Sadri, F. Probabilistic deductive databases. In *Proceedings of Workshop on Design and Implementation of Parallel Logic Programming Systems*, pages 254–268, Ithaca, NY, November 1994. MIT Press.

[LS01a] Lakshmanan, L.V.S. and Shiri, N. Logic programming and deductive databases with uncertainty: A survey. In *Enclyclopedia of Computer Science and Technology*, volume 45, pages 153–176. Marcel Dekker, Inc., New York, 2001.

[LS01b] Lakshmanan, L.V.S. and Shiri, N. A parametric approach to deductive databases with uncertainty. *IEEE Transactions on Knowledge and Data Engineering*, 13(4):554–570, 2001.

[Luk07] Lukasiewicz, T. Probabilistic description logics for the Semantic Web. Technical Report INFSYS 1843-06-05, Technische Universität Wien, Wien, Austria, March 2007.

[Mah04] Q.H. Mahmoud. Using and programming generics in J2SE 5.0, 2004. URL: http://java.sun.com/developer/technicalArticles/J2SE/generics/ (Last visited January 3rd, 2007).

[Mak02] Makowsky, J.A. *Theory of Horn clauses – Encyclopaedia of Mathematics.* Springer-Verlag, 2002.

[Mar05] Markoff, J. Debating the size of the Web. The New York Times, August 2005.

[Mol01] Moller, R. Expressive description logics: Foundations for practical applications, habilitation thesis. University of Hamburg, Computer Science Department, July 2001.

[MR05] Martn-Recuerda, F. and Robertson, D. Discovery and uncertainty in semantic web services. In *Proceedings of Uncertainty Reasoning for the Semantic Web*, Galway, Ireland, November 2005.

[MS97] Motro, A. and Smets,Ph., editors. *Uncertainty Management in Information Systems: From Needs to Solutions.* Kluwer Academic Publishers, Boston, 1997.

[Par96] Parsons, S. Current approaches to handling imperfect information in data and knowledge bases. *IEEE Transactions on Knowledge and Data Engineering*, 8(3):353–372, 1996.

[Pea88] Pearl, J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Francisco, CA, USA, 1988.

[PFT+04] Pan, J. Z., Franconi, E., Tessaris, S., Stamou, G., Tzouvaras, V., Serafini, L., Horrocks, I., and Glimm, B. Specification of coordination of rule and ontology languages. Technical Report KWEB/2004/D2.5.1/v1.0, The Knowledge Web project, June 2004.

[PH06] Patel-Schneider, P.F. and Horrocks, I. A comparison of two modelling paradigms in the semantic web. In *Proceeding of the Fifteenth International World Wide Web Conference (WWW 2006)*, pages 3–12. ACM, 2006.

[QPJ07] Qi, G., Pan, J. Z., and Ji, Q. Possibilistic extension of description logics. In *Proceedings of the International Workshop on Description Logics (DL'07)*, pages 435–442, 2007.

[Rea] RealPaver. URL: http://sourceforge.net/projects/realpaver (Last visited August 15th, 2007).

[SB07] Straccia, U. and Bobillo, F. Mixed integer programming, general concept inclusions and fuzzy description logics. In *Proceedings of the 5th Conference of the European Society for Fuzzy Logic and Technology (EUSFLAT-07)*, volume 2, pages 213–220, Ostrava, Czech Republic, 2007. University of Ostrava.

[Sch91] Schild, K. A correspondence theory for terminological logics: preliminary report. In *Proceedings of IJCAI-91, 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sidney, AU, 1991.

[Sch94] Schaerf, A. Reasoning with individuals in concept languages. *Data Knowledge Engineering*, 13(2):141–176, 1994.

[SEW+07] Stevens, R., Egana Aranguren, M., Wolstencroft, K., Sattler, U., Drummond, N., Horridge, M., and Rector, A.L. Using owl to model biological knowledge. *International Journal of Human-Computer Studies*, 65(7):583–594, 2007.

[She00] Sherman, C. Google announces largest index. Search Engine Report, July 2000.

[SS77] Stallman, R. M. and Sussman, G. J. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9(2):135–196, 1977.

[SS91] Schmidt-Schaubß, M. and Smolka, G. Attributive concept descriptions with complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[SSP+07] Stoilos, G., Stamou, G., Pan, J.Z., Tzouvaras, V., and Horrocks, I. Reasoning with very expressive fuzzy description logics. *Journal of Artificial Intelligence Research*, 30:273–320, 2007.

[SSSP06] Stoilos, G., Straccia, U., Stamou, G., and Pan, J.Z. General concept inclusions in fuzzy description logics. In *Proceedings of the 17th European Conference on Artificial Intelligence (ECAI 06)*, pages 457–461, Riva del Garda, Italy, August 2006. IOS Press.

[ST04] Sánchez, D. and Tettamanzi, A. G. B. Generalizing quantification in fuzzy description logics. In *Proceedings of the 8th Fuzzy Days in Dortmund - Advances in Soft Computing Series*. Springer-Verlag, 2004.

[Str98] Straccia, U. A fuzzy description logic. In *Proceedings of the 15th National Conference on Artificial Intelligence*, pages 594–599, Menlo Park, CA, USA, 1998. AAAI Press.

[Str01] Straccia, U. Reasoning within fuzzy description logics. *Journal of Artificial Intelligence Research*, 14:137–166, 2001.

[Str04a] Straccia, U. Transforming fuzzy description logics into classical description logics. In *Proceedings of the 9th European Conference on Logics in Artificial Intelligence*, pages 385–399. Springer-Verlag, 2004. Lecture Notes In

Computer Science; Vol. 3229.

[Str04b] Straccia, U. Uncertainty in description logics: a lattice-based approach. In *Proceedings of the 10th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (JIPMU'04)*, pages 251–258, 2004.

[Str05a] Straccia, U. Fuzzy description logic with concrete domains. Technical Report 2005-TR-03, Istituto di Elaborazione dell'Informazione, January 2005.

[Str05b] Straccia, U. Towards a fuzzy description logic for the semantic web (preliminary report). In *Proceedings of the 2nd European Semantic Web Conference (ESWC'05)*, pages 167–181. Springer Verlag, 2005. Lecture Notes in Computer Science; Vol. 3532.

[Suna] Sun Microsystems. Incompatibilities in J2SE 5.0 (since 1.4.2).
URL: http://java.sun.com/j2se/1.5.0/compatibility.html (Last visited January 3rd, 2007).

[Sunb] Sun Microsystems. Java Platform Standard Edition 6 API Specification - HashSet.
URL: http://java.sun.com/javase/6/docs/api/java/util/HashSet.html (Last visited January 3rd, 2007).

[Tau06] Tauberer , J. What is RDF. In *XML.com*, July 2006.

[TM98] Tresp, C. and Molitor, R. A description logic for vague knowledge. In *Proceedings of ECAI-98*, pages 361–365, Brighton, UK, 1998. John Wiley and Sons.

[Van86] Van Emden, M. H. Quantitative deduction and its fixpoint theory. *Journal of Logic Programming*, 3(1):37–53, 1986.

[VSP05] Van Ossenbruggen, J., Stamou, G., and Pan, J. Z. Multimedia annotations and the semantic web. In *Proceedings of Semantic Web Case Studies and Best Practices for eBusiness*, Galway, Ireland, November 2005.

[W3C99] W3C. HTML 4.01 specification, 1999.
URL: http://www.w3.org/TR/html4/ (Last visited September 22nd, 2006).

[W3C01] W3C. DAML+OIL reference description, 2001.
URL: http://www.w3.org/TR/daml+oil-reference (Last visited September 22nd, 2006).

[W3C04a] W3C. OWL web ontology language overview, 2004.
URL: http://www.w3.org/TR/owl-features/ (Last visited September 14th, 2007).

[W3C04b] W3C. RDF primer, February 2004.
URL: http://www.w3.org/TR/rdf-primer/ (Last visited September 22nd, 2006).

[W3C04c] W3C. RDF vocabulary description language 1.0: Rdf schema, 2004. URL: http://www.w3.org/TR/rdf-schema/ (Last visited September 22nd, 2006).

[W3C04d] W3C. RDF/XML syntax specification, 2004. URL: http://www.w3.org/TR/rdf-syntax-grammar/ (Last visited September 22nd, 2006).

[W3C06a] W3C. Extensible Markup Language (XML) 1.0, 2006. URL: http://www.w3.org/TR/2006/REC-xml-20060816/ (Last visited September 22nd, 2006).

[W3C06b] W3C. Naming and addressing: URIs, URLs, ..., 2006. URL: http://www.w3.org/Addressing/ (Last visited September 22nd, 2006).

[Wik07] Wikipedia. Syntactic Web — Wikipedia, the free encyclopedia, 2007. URL: http://en.wikipedia.org/wiki/Syntactic_web (Last visited December 2nd, 2006).

[Yel99] Yelland, P. M. Market analysis using a combination of bayesian networks and description logics. Technical Report SMLI TR-99-78, Sun Microsystems Laboratories, August 1999.

[Zad65] Zadeh, L. A. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[Zad78] Zadeh, L. A. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1(1):3–28, 1978.

Appendix A

URDL Implementation Details

```
// Augment the ABox with respect to the TBox
augmentABoxWrtTBox()
{
  If the ABox is empty, add an individual ind to the ABox.

  For each individual ind in the ABox
  {
    For each axiom in the TBox of the form ⟨⊤ ⊑ ¬C ⊔ D | α, f_c, f_d⟩
    {
      Create an assertion A of the form ⟨ind : ¬C ⊔ D | α, f_c, f_d⟩.
      Add A to the ABox.
    }
  }
}
```

Figure 43: Pseudo-code for augmenting the ABox with respect to the TBox

System Preference	Description
certaintyLattice	The certainty lattice used in the input knowledge base.
certaintyPrecision	The precision of the certainty value.
generatedIndividualName	The prefix of the generated individual name.
generatedVariableName	The prefix of the variable name.
constraintInputFileName	The location and the prefix of the file name where the generated constraints will be saved.

constraintOutputFileName	The location and the prefix of the file name where the constraint solving result will be stored.
consistencySolverCommand	The location of the constraint solver and the solver preference used for consistency checking.
entailmentSolverCommand	The location of the constraint solver and the solver preference used for entailment checking.

Table 33: System preferences

Printing Preference	**Description**
printMenu	Display the menu. If menu is disabled, the reasoner will check the consistency of the knowledge base by default.
debug	Display the debugging messages.
printIndividualsInIndividualGroup	Print the individual names in each Individual Group.
printIndividualsInAssertionGroup	Print the individual names in each Assertion Group.
printAssertions	Print the assertions, including the derived assertions.
printVariables	Print the generated variables.
printConstraints	Print the derived constraints.
printModel	Print the model.
printConflictingAssertions	Print the conflicting assertions if the knowledge base is inconsistent.
printDerivedConflictingAssertions	Print the conflicting assertions, including the derived ones, if the knowledge base is inconsistent.
printRunningTime	Print the running time.
printKBHeightWidth	Print the height and the width of the knowledge base.

Table 34: Printing preferences

Optimization Preference	Description
partitionOption	Specifies whether the ABox should be partitioned into Assertion Groups, Individual Groups, or no partition.
optimizeIndividualGroupCreation	Specifies whether optimized Individual Group creation should be turned on or off.
enableCaching	Specifies whether caching should be enabled or disabled.
optimizedClashDetection	Specifies whether optimized clash detection should be enabled or disabled.
stopCheckConsistencyOnFail	Specifies whether to stop consistency checking in case one Assertion Group is found to be inconsistent.
optimizeHashing	Specifies whether optimized hashing should be enabled or disabled.
optimizeStringComparison	Specifies whether optimized string comparison should be enabled or disabled.
callConstraintSolver	Specifies whether the reasoner should call the constraint solver to solve the constraints.
writeConstraintsToFile	Specifies whether generated constraints should be written to files. (For example, in case callConstraintSolver is disabled, there is no need to write the constraints to file.)

Table 35: Optimization preferences

KB	:=	(Statement (EOL)$^+$)* EOF
Statement	:=	Assertion
	|	Axiom
	|	FunctionDef
EOL	:=	"\n"
	|	"\r"
	|	"\r\n"
Assertion	:=	"(" Instance Identifier Concept "|" CertaintyValue "," CombinationFunction "," CombinationFunction")"
	|	"(" Related Identifier Identifier Identifier "|"

		CertaintyValue ", -, -)"
Axiom	:=	"(" DefineConcept Identifier Concept "|"
		CertaintyValue "," CombinationFunction ","
		CombinationFunction")"
	|	"(" DefinePrimitiveConcept Identifier Concept "|"
		CertaintyValue "," CombinationFunction ","
		CombinationFunction")"
	|	"(" Equivalent Concept Concept "|" CertaintyValue ","
		CombinationFunction "," CombinationFunction")"
	|	"(" Implies Concept Concept "|" CertaintyValue ","
		CombinationFunction "," CombinationFunction")"
FunctionDef	:=	Identifier "(" Identifier "," Identifier ")" "="
		AdditiveExpression
	|	Max "(" Identifier "," Identifier ")" "=" Max "("
		AdditiveExpression "," AdditiveExpression ")"
	|	Min "(" Identifier "," Identifier ")" "=" Min "("
		AdditiveExpression "," AdditiveExpression ")"
Instance	:=	"instance"
	|	"INSTANCE"
Identifier	:=	["a"-"z","A"-"Z"] (["a"-"z","A"-"Z","0"-"9","-"])*
Concept	:=	AtomicConcept
	|	"(" Not Concept ")"
	|	"(" And Concept Concept ")"
	|	"(" Or Concept Concept ")"
	|	"(" Some Identifier Concept ")"
	|	"(" All Identifier Concept ")"
CertaintyValue	:=	"[" Number "," Number "]"
CombinationFunction	:=	"_"
	|	Max
	|	Min
	|	Identifier
Related	:=	"related"
	|	"RELATED"
DefineConcept	:=	"define-concept"
	|	"DEFINE-CONCEPT"
DefinePrimitiveConcept	:=	"define-primitive-concept"
	|	"DEFINE-PRIMITIVE-CONCEPT"
Equivalent	:=	"equivalent"
	|	"EQUIVALENT"
Implies	:=	"implies"

	|	"IMPLIES"
AtomicConcept	:=	Identifier
	|	Top
	|	Bottom
Not	:=	"not"| "NOT"
And	:=	"and"| "AND"
Or	:=	"or" | "OR"
Some	:=	"some" | "SOME"
All	:=	"all" | "ALL"
Top	:=	"*top*" | "*TOP*"
Bottom	:=	"*bottom*"| "*BOTTOM*"
Number	:=	(["0"-"9"])* (".")? (["0"-"9"])+
Max	:=	"min" | "MIN"
Min	:=	"max" | "MAX"
AdditiveExpression	:=	MultiplicativeExpression (("+" MultiplicativeExpression) |
		("-" MultiplicativeExpression))*
MultiplicativeExpression	:=	UnaryExpression (("*" UnaryExpression) |
		("/" UnaryExpression))*
UnaryExpression	:=	Identifier
	|	(AdditiveExpression)
	|	Number

Table 36: EBNF grammar used by parser

```
// Add the atomic concept assertion A_s of the form (a : A | Γ, −, −) to the ABox.
// If A_s is a derived assertion, AG is the Assertion Group of the assertion that
// A_s derived from. Otherwise, AG is null.
addAtomicConceptAssertion(Assertion A_s, Assertion Group AG)
{
  If individual a is not yet in the ABox, add it.

  If Γ is a numerical certainty value
  {
    certaintyOfA = Γ
    If Γ clashes with any existing certainty values for x_{a:A}, return.
  }
  Else
    certaintyOfA = x_{a:A}

  // Decide which Assertion Group A_s should belong to.
  If A_s is not present in the ABox
  {
    If AG is null
      Create a new Assertion Group for A_s.
    Else
      A_s is in AG.

    Add A_s to the list of atomic concept assertions associated with a.
  }
  Else // There is an assertion A' in the ABox that is identical to A_s
  {
    If AG is null
      A_s is in the same Assertion Group as A'.
    Else
      Merge the Assertion Group of A' with AG.
  }

  If certaintyOfA is a numerical certainty value
  {
    Create a constraint C of the form (x_{a:A} = Γ)
    Add C to the Assertion Group of A_s.

    Create a constraint C' of the form (x_{a:¬A} =~ Γ)
    Add C' to the Assertion Group of A_s.
  }
  Else
  {
    Create a constraint C of the form (x_{a:¬A} = ⊤ − x_{a:A})
    Add C to the Assertion Group of A_s.
  }
  Add new variables to the Assertion Group of A_s.
}
```

Figure 44: Pseudo-code for adding atomic concept assertion into the ABox

```
// Add the non-atomic concept assertion A of the form ⟨a : C | Γ, f_c, f_d⟩ to the ABox.
// If A is a derived assertion, AG is the Assertion Group of the assertion that
// A derived from. Otherwise, AG is null.
addNonAtomicAssertion(Assertion A, Assertion Group AG)
{
  If individual a is not yet in the ABox, add it.

  If Γ is a numerical certainty value
  {
    certaintyOfA = Γ
    If Γ clashes with any existing certainty values for x_{a:C}, return.
  }
  Else
    certaintyOfA = x_{a:C}

  Check if there is an assertion A' associated with a that is either identical to A or
  has the same concept name as A.

  // Decide which Assertion Group A should belong to.
  If there is no A' found
  {
    If AG is null
      Create a new Assertion Group for A.
    Else
      A is in AG.
  }
  Else // There is A' found
  {
    If AG is null
      A is in the same Assertion Group as A'.
    Else
      Merge the Assertion Group of A' with AG.
  }

  If there is no assertion associated with a that is identical to A
    Add A to the list of concept assertions associated with a.

  If A is of type Role Value Restriction
    Add A to the list of Role Value Restriction assertions associated with a.

  If certaintyOfA is a numerical certainty value
  {
    Create a constraint C of the form (x_{a:C} = Γ)
    Add C to the Assertion Group of A.
  }
  Add new variables to the Assertion Group of A.
}
```

Figure 45: Pseudo-code for adding non-atomic concept assertion into the ABox

```
// Add the role assertion A of the form ((a, b) : R | Γ, −, −) to the ABox.
// If A is a derived assertion, AG is the Assertion Group of the assertion that
// A derived from. Otherwise, AG is null.
addRoleAssertion(Assertion A, Assertion Group AG)
{
  If individuals a and b are not yet in the ABox, add them.
  If Γ is a numerical certainty value
  {
    certaintyOfA = Γ
    If Γ clashes with any existing certainty values for x_(a,b):R, return.
  }
  Else
    certaintyOfA = x_(a,b):R

  // Decide which Assertion Group A should belong to.
  If A is not present in the ABox
  {
    If AG is null
      Create a new Assertion Group for A.
    Else
      A is in AG.

    Add A to the list of role assertions associated with a.
  }
  Else // There is an assertion A' in the ABox that is identical to A
  {
    If AG is null
      A is in the same Assertion Group as A'.
    // There is no Else, since we cannot have a role assertion that is both
    // inferred and is already present in the ABox.
  }

  If certaintyOfA is a numerical certainty value
  {
    Create a constraint C of the form (x_(a,b):R = Γ)
    Add C to the Assertion Group of A.

    Create a constraint C' of the form (x_¬(a,b):R =~ Γ)
    Add C' to the Assertion Group of A.
  }
  Else
  {
    Create a constraint C of the form (x_¬(a,b):R = ⊤ − x_(a,b):R)
    Add C to the Assertion Group of A.
  }
  Add new variables to the Assertion Group of A.
  Call determineIGForRoleAssertion(A) to decide the Individual Group for A
}
```

Figure 46: Pseudo-code for adding role assertion into the ABox

```
// Create or merge Individual Group for individuals a and b in the role
// assertion A of the form ⟨(a, b) : R | Γ, −, −⟩.
// StandaloneIndividuals is list consists of individuals that currently do not
// belong to any Individual Group.
determineIGForRoleAssertion(Assertion A)
{
  Let IG1 be the Individual Group of individual a.
  Let IG2 be the Individual Group of individual b.

  If both IG1 and IG2 are null
  {
    // IG1 and IG2 do not yet belong to any Individual Group
    Remove IG1 and IG2 from StandaloneIndividuals
    Create a new Individual Group for a and b.    }
  Else if IG1 is not null, and IG2 is null
  {
    // a already belongs to the Individual Group IG1, but b does not yet
    // belong to any Individual Group.
    Remove b from StandaloneIndividuals
    Add b to IG1.
  }
  Else if IG1 is null, and IG2 is not null
  {
    // b already belongs to the Individual Group IG2, but a does not yet
    // belong to any Individual Group.
    Remove a from StandaloneIndividuals
    Add a to IG2.
  }
  Else // Both IG1 and IG2 are not null
  {
    // a already belongs to the Individual Group IG1, and b already
    // belongs to the Individual Group IG2.
    Merge IG2 to IG1 to form one single Individual Group.
}
```

Figure 47: Pseudo-code for determining Individual Group for role assertion

```
// Apply the negation rule to assertion A of the form ⟨a : ¬A | x_{a:¬A}, −, −⟩
negationRule(Assertion A)
{
  Create an assertion A′ of the form ⟨a : A | x_{a:A}, −, −⟩

  Assign the line number of A to the line number of A′ to indicate
  that A′ was derived from A.

  Add A′ to the ABox.
}
```

Figure 48: Pseudo-code for the negation rule

```
// Apply the conjunction rule to assertion A of the form ⟨a : C ⊓ D | x_{a:C⊓D}, f_c, f_d⟩
conjunctionRule(Assertion A)
{
  Create an assertion A′ of the form ⟨a : C | x_{a:C}, f_c, f_d⟩
  Assign the line number of A to the line number of A′ to indicate
  that A′ was derived from A.
  Add A′ to the ABox.

  Create an assertion A″ of the form ⟨a : D | x_{a:D}, f_c, f_d⟩
  Assign the line number of A to the line number of A″ to indicate
  that A″ was derived from A.
  Add A″ to the ABox.

  Create a constraint C of the form (f_c(x_{a:C}, x_{a:D}) = x_{a:C⊓D})
  Add C to the ABox.

  Add new variables to the ABox.
}
```

Figure 49: Pseudo-code for the conjunction rule

```
// Apply the disjunction rule to assertion A of the form ⟨a : C ⊔ D | x_{a:C⊔D}, f_c, f_d⟩
disjunctionRule(Assertion A)
{
  Create an assertion A' of the form ⟨a : C | x_{a:C}, f_c, f_d⟩
  Assign the line number of A to the line number of A' to indicate
  that A' was derived from A.
  Add A' to the ABox.

  Create an assertion A'' of the form ⟨a : D | x_{a:D}, f_c, f_d⟩
  Assign the line number of A to the line number of A'' to indicate
  that A'' was derived from A.
  Add A'' to the ABox.

  Create a constraint C of the form (f_d(x_{a:C}, x_{a:D}) = x_{a:C⊔D})
  Add C to the ABox.

  Add new variables to the ABox.
}
```

Figure 50: Pseudo-code for the disjunction rule

```
// Apply the role value restriction rule to assertion A of the form ⟨a : ∀R.C | x_{a:∀R.C}, f_c, f_d⟩
roleValueRestrictionRule(Assertion A)
{
  Let RoleAssertions be the set of role assertions associated with individual a.

  For each role assertion A_R in RoleAssertions
  {
    If A_R is of the form ⟨(a, b) : R | x_{(a,b):R}, −, −⟩ where b is any individual that is
    related to individual a through role R
    {
      Create an assertion A' of the form ⟨b : C | x_{b:C}, f_c, f_d⟩
      Assign the line numbers of A and A_R to the line number of A' to indicate
      that A' was derived from A and A_R.
      Add A' to the ABox.

      Create a constraint C of the form (f_d(x_{¬(a,b):R}, x_{b:c}) = x_{a:∀R.C})
      Add C to the ABox.

      Add new variables to the ABox.
    }
  }
}
```

Figure 51: Pseudo-code for the role value restriction rule

```
// Apply the role exists restriction rule to assertion A of the form ⟨a : ∃R.C | x_{a:∃R.C}, f_c, f_d⟩
roleExistsRestrictionRule(Assertion A)
{
  If individual a is blocked by its ancestor, return.

  Let b be a newly generated individual.

  Create an assertion A' of the form ⟨(a, b) : R | x_{(a,b):R}, −, −⟩
  Assign the line number of A to the line number of A' to indicate
  that A' was derived from A.
  Add A' to the ABox.

  Create an assertion A'' of the form ⟨b : C | x_{b:C}, f_c, f_d⟩
  Assign the line number of A to the line number of A'' to indicate
  that A'' was derived from A.
  Add A'' to the ABox.

  Create a constraint C of the form (f_c(x_{(a,b):R}, x_{b:C}) = x_{a:∃R.C})
  Add C to the ABox.

  Add new variables to the ABox.

  // Abstract the ABox w.r.t. TBox
  For each axiom in the TBox of the form ⟨⊤ ⊑ ¬C ⊔ D | α, f_c, f_d⟩
  {
    Create an assertion A''' of the form ⟨b : ¬C ⊔ D | α, f_c, f_d⟩.
    Add A''' to the ABox.
  }
  Call processRoleValueRestriction(A, A') to process any Role Value Restriction
  assertions associated with a due to the newly derived role assertion A'.
}
```

Figure 52: Pseudo-code for the role exists restriction rule

```
// Process the Role Value Restriction assertions associated with a due to the new
// role assertion A' of the form ⟨(a, b) : R | x_{(a,b):R}, −, −⟩ generated by the application
// of Role Exists Restriction rule to assertion A of the form ⟨a : ∃R.C | x_{a:∃R.C}, f_c, f_d⟩
processRoleValueRestriction(Assertion A, Assertion A')
{
  Let RoleValueRestrictions be the set of assertions with Role Value Restrictions
  that are associated with individual a.

  For each assertion A_R in RoleValueRestrictions
  {
    If A_R is of the form ⟨a : ∀R.D | x_{a:∀R.D}, f'_c, f'_d⟩ where A_R has the same
    role name as A', and D is any concept name
    {
      Create an assertion A'' of the form ⟨b : D | x_{b:D}, f'_c, f'_d⟩
      Assign the line numbers of A' and A_R to the line number of A'' to indicate
      that A'' was derived from A' and A_R.
      Add A'' to the ABox.

      Create a constraint C of the form (f'_d(x_{¬(a,b):R}, x_{b:D}) = x_{a:∀R.D})
      Add C to the ABox.

      Add new variables to the ABox.

      Merge the Assertion Groups of A and A_R.
    }
  }
}
```

Figure 53: Pseudo-code for checking whether role value restriction rule needs to be re-applied

Appendix B

Glossary

ABox = Assertional Box

AG = Assertion Group

$\mathcal{ALC}$ = A Description Logic fragment that provides the following language constructors: atomic concept, atomic role, top concept, bottom concept, concept negation, concept conjunction, concept disjunction, role exists restriction, and role value restriction

$\mathcal{ALC}_U$ = The Description Logic $\mathcal{ALC}$ extended with uncertainty

BDB = Best Degree Bound

bMIP = Bounded, Mixed Integer Programming

DL = Description Logic

EBNF = Extended BackusNaur Form

FOL = First-Order Logic

GCI = General Concept Inclusion

JavaCC = Java Compiler Compiler

Java SE = Java Standard Edition, also known as J2SE

HTML = HyperText Markup Language

IG = Individual Group

NNF = Negation Normal Form

OWL = Web Ontology Language

OWL DL = An OWL specie that has close correspondence with the Description Logic $\mathcal{SHOIN}$

RDF = Resource Description Framework

RDFS = Resource Description Framework Schema

$\mathcal{SHOIN}$ = $\mathcal{ALC}$ extended with transitive role axioms, role hierarchy, nominal, inverse role, and unqualified number restriction

TBox = Terminological Box

URDL = Uncertainty Reasoner for the DL $\mathcal{ALC}_U$

URI = Universal Resource Identifier

URL = Uniform Resource Locators

W3C = World Wide Web Consortium

XML = Extensible Markup Language