TABLEAU-BASED REASONING FOR DESCRIPTION LOGICS WITH INVERSE ROLES AND NUMBER RESTRICTIONS

Yu Ding

A thesis
in
The Department
of
Computer Science & 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: 978-0-494-37744-4
Our file Notre référence
ISBN: 978-0-494-37744-4

ABSTRACT

Tableau-based Reasoning for Description Logics with Inverse Roles and Number Restrictions

Yu Ding, Ph.D.
Concordia University, 2008

The tableaux algorithm is a general technique for deciding concept satisfiability problems in description logics (DLs). It is useful not only for practical implementations, but also for studying the correctness and complexity of concrete decision procedures.

There is a family of DLs that currently lack appropriate optimization techniques. The research focuses on these DLs which typically have inverse roles and number restrictions (corresponding to ontology languages OWL-lite and OWL-DL respectively). We provide solutions to known problems such as the unsoundness of global tableaux caching, and present new tableau-based algorithms for concept satisfiability problems in these DLs. The research presented in this thesis is significant in several aspects. Firstly, based on an equivalence discovered during the course of the research, we are able to show an elimination of inverse roles for a sub-family of DLs. Our experiments have confirmed the practicality of this technique. Secondly, we provide three sub-tableaux caching techniques that are sound and global (but with different power in caching functionality). Finally, we present two ExpTime tableau-based decision procedures, with the one for $\mathcal{SHIQ}$ achieving an improved worst-case upper bound in the strong sense of binary coding of numbers (based on the integer linear programming technique).

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Description logics (DLs) are a family of *knowledge representation formalisms* suitable for representing the *terminological knowledge* in a wide range of applications [BCM$^+$03]. The *Tableaux algorithm* [BS01] is a general technique for deciding the *concept satisfiability* problems in description logics. Historically, the *tableaux algorithm* provides an algorithmic framework that is parametric with respect to *language constructors* and is useful for studying both correctness and complexity of concrete *decision procedures* [DLNS96, BHLW03].

1.1 Background

Description logics were designed as an extension to *frames* [Min85] and *semantic networks* [LL00], which were criticized as "not equipped with logic-based semantics" [BCM$^+$03]. *Description logics* were given their current name in late 1980s. Prior to this they were called, among others, *terminological systems* and *concept languages.* The research about finding the right "fragments" that are both expressive for applications and "practical" for computation can be found in [Baa90, DLNS94, Sat96, BCM$^+$03]. The computational properties of various description formalisms have been thoroughly investigated in the literature, e.g., [DLNN91]. The name *description logic*, on the one hand, refers to *concept descriptions* used to describe a domain and, on the other hand, to the *logic-based semantics* which can be given by a translation into first-order predicate logic [BCM$^+$03]. Notably, today description logic has become a cornerstone of several areas for its use in the design of ontologies, e.g., the Semantic Web, the DL-based software information system (SIS), practical software engineering, etc.

There are various implemented DL systems based on tableaux algorithms, offering a palette of description formalisms with different expressive power. In the history, the first DL-like system was KL-ONE [BS85]. KRIS [BHNP94] is one of the first *description logic* reasoners that implemented a highly optimized *tableaux algorithm.* A worst-case optimal tableau-based procedure for the *concept satisfiability* problem of the DL $\mathcal{ALC}$ was first given in [DM00] in details. The proposed *global sub-tableaux caching* technique is quite influential on practical tableau-based DL systems. Dozens of different tableau-based DL systems[1], with sophisticated optimization techniques, have been implemented since the mid-1980s.

1.2 Syntax and Semantics

The fundamental representation unit of *description logics* is the so-called *concept description.* In the literature, a *concept description* is also called a *concept expression* or a *concept formula*, but the most common name is its abbreviation *concept. Concepts* are built from atomic *concept names* (and individual names called *nominals*) using certain constructors, e.g., the common *boolean operators*[2] such as $\sqcap$ (for "conjunction"), and $\sqcup$ (for "disjunction"), and $\neg$ (for "negation"), and others with more description logic flavor. *Concepts* can be used at the conceptual level for describing *terminological knowledge*, e.g., the *terminological axioms* of a Tbox. *Concepts* can also be used at the assertional level for describing knowledge of *individuals.* In Section 1.3.2, we will introduce the notion of Abox *individuals.*

In the following, we review the DL $\mathcal{SHOIQ}$ proposed in [HS05, HS07]. Formally, *concepts* are inductively defined through a set of *concept constructors*, starting with a set N_C of *concept names* (and with a special *nominal* set N_O) and a set N_R of *role names.* The available *concept constructors* determine the expressiveness of the DL.

[1]A list of DL reasoners is maintained at http://www.cs.man.ac.uk/~sattler/reasoners.html.

[2]A DL that provides all the boolean operators is called propositionally closed. DLs that are not propositionally closed are called sub-boolean DLs (e.g., the $\mathcal{EL}$-family [BBL05] DLs).

Definition 1. (Concept) Let N_C, N_O, and N_R be pair-wise disjoint sets of *concept names*, *nominals*, and *role names*. *Atoms* are defined as the union of N_C and N_O, i.e., $N_{Atom} = N_C \cup N_O$. $N_R \cup \{R^- | R \in N_R\}$ is the set of *roles*. R^- stands for the *inverse* of a role name R. The set of (well-formed) $\mathcal{SHOIQ}$ concepts is the set such that:

- each $A \in N_{Atom}$ is a $\mathcal{SHOIQ}$-concept; and
- if C and D are $\mathcal{SHOIQ}$-concept formulae, R a role, and n a non-negative integer number, then $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $(\exists^{\geq n}\ R.C)$ and $(\exists^{\leq n}\ R.C)$ are also $\mathcal{SHOIQ}$-concepts.

$\exists R.C$ is called the *existential restriction*, $\forall R.C$ the *universal restriction*, $(\exists^{\geq n}\ R.C)$ the *at-least restriction*, and $(\exists^{\leq n}\ R.C)$ the *at-most restriction*. They are generally called *modal constraints*. We use $\top$ for denoting $A \sqcup \neg A$, and $\bot$ for $A \sqcap \neg A$.

The semantics of description logics is defined by interpreting *concepts* as sets of individuals and *roles* as sets of pairs of individuals.

Definition 2. (Interpretation) An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ consists of a non-empty set $\Delta^{\mathcal{I}}$, called the interpretation domain, and a mapping $.^{\mathcal{I}}$ which associates with each $A \in N_{Atom}$ a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, and with each role name R a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. The interpretation is defined as follows:

- $(C \sqcap D)^{\mathcal{I}} = C^{\mathcal{I}} \cap D^{\mathcal{I}}$
- $(C \sqcup D)^{\mathcal{I}} = C^{\mathcal{I}} \cup D^{\mathcal{I}}$
- $\neg C^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
- $(\exists R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} | \exists e \in \Delta^{\mathcal{I}}$ with $(d, e) \in R^{\mathcal{I}}$ and $e \in C^{\mathcal{I}}\}$
- $(\forall R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} |$ for all $e \in \Delta^{\mathcal{I}}$, if $(d, e) \in R^{\mathcal{I}}$ then $e \in C^{\mathcal{I}}\}$
- $(\exists^{\geq n}\ R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} | \sharp\{e \in C^{\mathcal{I}} | (d, e) \in R^{\mathcal{I}}\} \geq n\}$
- $(\exists^{\leq n}\ R.C)^{\mathcal{I}} = \{d \in \Delta^{\mathcal{I}} | \sharp\{e \in C^{\mathcal{I}} | (d, e) \in R^{\mathcal{I}}\} \leq n\}$
- for $o \in N_O$, $o^{\mathcal{I}} = \{o' \in \Delta^{\mathcal{I}}\}$ and $\sharp\{o' \in \Delta^{\mathcal{I}}\} = 1$
- for a role R, $(a, b) \in R^{\mathcal{I}}$ iff $(b, a) \in (R^-)^{\mathcal{I}}$

- for a *transitive role* S, $(a,b) \in S^{\mathcal{I}} \wedge (b,c) \in S^{\mathcal{I}} \Rightarrow (a,c) \in S^{\mathcal{I}}$

An element $d \in C^{\mathcal{I}}$ is called an instance of *concept* C. For two elements d and e, if $(d,e) \in R^{\mathcal{I}}$, then e is called an R-neighbor of d, and d is called an R^{-}-neighbor of e. Note that a *nominal* is interpreted as a singleton set (i.e., its cardinality is 1).

Definition 3. (Satisfiability, Subsumption, Equivalence, Disjointness) A concept description D subsumes a concept description C (written $C \sqsubseteq D$) iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ for all interpretations $\mathcal{I}$. We say that C is satisfiable iff there exists an interpretation $\mathcal{I}$ such that $C^{\mathcal{I}} \neq \emptyset$; C and D are equivalent iff $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ and $D^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ for each $\mathcal{I}$; C and D are disjoint iff $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for all $\mathcal{I}$.

As stated in [BCM^{+}03], we have the following three propositions.

Proposition 4. (Reduction to Subsumption) *For concepts C and D:*

- *C is unsatisfiable $\Leftrightarrow$ $C \sqsubseteq \bot$;*
- *C and D are equivalent $\Leftrightarrow$ $C \sqsubseteq D$ and $D \sqsubseteq C$;*
- *C and D are disjoint $\Leftrightarrow$ $C \sqcap D \sqsubseteq \bot$.*

Proposition 5. (Reduction to Unsatisfiability) *For concepts C and D:*

- *C is subsumed by D $\Leftrightarrow$ $C \sqcap \neg D$ is unsatisfiable;*
- *C and D are equivalent $\Leftrightarrow$ both $(C \sqcap \neg D)$ and $(\neg C \sqcap D)$ are unsatisfiable;*
- *C and D are disjoint $\Leftrightarrow$ $C \sqcap D$ is unsatisfiable.*

Proposition 6. (Reducing Unsatisfiability) *Let C be a concept. Then the following are equivalent:*

- *C is unsatisfiable;*
- *C is subsumed by $\bot$;*
- *C and $\bot$ are equivalent;*
- *C and $\top$ are disjoint;*

The *concept satisfiability* (subsumption, equivalence and disjointness) problem defined above is commonly called the *pure concept satisfiability* (subsumption, equivalence and disjointness) problem because no Tbox is taken into consideration[3].

	Propositional Constructs	**Universal/ Existential Restriction**	**Functional Restriction**	**Qualified Number Restriction**	**Role Hierarchy**	**Inverse Role**	**Transitive Role**	**Nominal**
DL Name	$\sqcap$ $\sqcup$ $\neg$	$\forall R.C$ $\exists R.C$	$\leq 1R$ $\geq 1R$	$\exists^{\leq n}R.C$ $\exists^{\geq n}R.C$	$R_1 \sqsubseteq R_2$	R^-	S	$\{o\}$
$\mathcal{ALC}$	√	√						
$\mathcal{ALCI}$	√	√				√		
$\mathcal{ALCOI}$	√	√				√		√
$\mathcal{ALCFI}$	√	√	√			√		
$\mathcal{ALCHIQ}$	√	√		√	√	√		
$\mathcal{SH}$	√	√			√		√	
$\mathcal{SHOI}$	√	√			√	√	√	√
$\mathcal{SHIQ}$	√	√		√	√	√	√	

Figure 1.1: Description Logics and Their Language Elements

Figure 1.1 lists the description logics that will be discussed in this thesis.

1.3 Tbox, Abox and Role Hierarchy

Besides the *concept descriptions* for describing sets of individuals or objects, the second major representation mechanism of *description logics* is the *knowledge base*, which consists of a Tbox and an Abox. In this part, we introduce the Tbox, Abox, and Role hierarchy.

1.3.1 Tbox

The first component of a DL *knowledge base* is the Tbox ("T" for terminological). A Tbox can be either a *simple Tbox* or a *general Tbox*. A simple Tbox was also called a *terminology* in the past.

[3]For description logics such as $\mathcal{ALCOI}$ or $\mathcal{SH}$, it is known that these logics are so expressive that a whole Tbox can be "internalized" as a single concept expression [BCM+03]. For these description logics, the pure concept satisfiability problem is as general as the concept satisfiability problem in a Tbox because the Tbox itself can be expressed in a concept description. For description logics such as $\mathcal{ALCHI}$, it is impossible to "internalize" a whole Tbox, therefore we need to differentiate the case of the pure satisfiability problem and the case of the satisfiability problem in a Tbox.

Definition 7. (Simple Tbox) The elements of a simple Tbox are either *concept inclusions* (e.g., $A \sqsubseteq C$) or *concept definitions* (e.g., $A \equiv C$).

Both *inclusions* and *definitions* introduce symbolic names for complex *concept descriptions.* In a *simple Tbox*, at most one *concept definition* is allowed per concept name. The *concept inclusions* or *concept definitions* of a *simple Tbox* can serve as "rewrite rules" to expand concept names to their definition without compromising soundness. If concept names are not allowed to refer to themselves, neither directly nor indirectly, then we have an acyclic simple Tbox. Otherwise, it is called a cyclic simple Tbox.

As an example, the concept "MOTHER", as interpreted in English as "a WOMAN who has a child", could be introduced by a *description* like:

$\text{Mother} \sqsubseteq \text{Woman} \sqcap \exists \textit{has\_child}.\text{Person}$

To state that a "WOMAN is a PERSON", we could use a second *concept inclusion* like:

$\text{Woman} \sqsubseteq \text{Person}$

Among these two *concept inclusions* there is no cyclic reference relationship, therefore together they are acyclic.

Definition 8. (General Concept Inclusion) A concept inclusion is called general (a.k.a. general concept inclusion, or GCI for short) if it is of the form $C \sqsubseteq D$, where C and D are arbitrary $\mathcal{SHOIQ}$-concepts. A Tbox is a finite set of GCIs.

A Tbox having *general concept inclusions* is a *general Tbox.* As usually perceived by a knowledge engineer in the knowledge representation area: the *unfoldable* part of a *general Tbox* provides definitions; the GCI part (containing *general concept inclusions*) provides background knowledge.

The definition of the interpretation given in the previous section is extended for Tboxes as follows. An interpretation $\mathcal{I}$ satisfies a GCI $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. $\mathcal{I}$ satisfies a Tbox T if $\mathcal{I}$ satisfies all GCIs in T; in this case, $\mathcal{I}$ is called a model of T, and T is *coherent.*

Likewise, the notions of **Satisfiability, Subsumption, Equivalence** and **Disjointness** are extended for a Tbox as follows. A concept C is *satisfiable* w.r.t a Tbox T if there is a model $\mathcal{I}$ of T with $C^{\mathcal{I}} \neq \emptyset$. A concept C is *subsumed* by a concept D w.r.t T (written $C \sqsubseteq_T D$) if, for each model $\mathcal{I}$ of T, we have that $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. Two concepts are *equivalent* if each one subsumes the other. Two concepts are *disjoint* if $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \emptyset$ for every model $\mathcal{I}$ of T. Any one of the two reasoning problems, *satisfiability* and *subsumption*, can be reduced to the other: C is satisfiable w.r.t T iff C is not subsumed by $\bot$ w.r.t T; $C \sqsubseteq_T D$ iff $C \sqcap \neg D$ is not satisfiable w.r.t T.

Moreover, $C \equiv D$ is an abbreviation for $C \sqsubseteq D$ and $D \sqsubseteq C$.

The *taxonomy* of a Tbox T is a partial order (i.e. subsumption hierarchy) of the concept names w.r.t. $\sqsubseteq_T$, and may be viewed as a mathematical structure, such as a lattice or Hasse diagram. The process of computing the *taxonomy* of concept names is also called the *classification* of terminologies.

Checking Tbox *coherence*, computing the Tbox *taxonomy*, testing *concept subsumption* and *concept satisfiability* with respect to a Tbox are Tbox-related reasoning tasks, among which the *concept satisfiability problem* is the fundamental one [BCM+03].

1.3.2 Abox

The second component of the *knowledge base* is the Abox ("A" for assertional). An Abox describes named *individuals* and their relations while possibly referring to the *concept descriptions* in the Tbox.

Definition 9. (Abox Assertion) Given a set of individual names N_I, an Abox *assertion* is of either of the following two forms:

- $a : C$
- $(a, b) : R$

where $a, b \in N_I$ are individual names, C is a concept, and R is a role. An Abox is a finite set of assertions.

The definition of an interpretation $\mathcal{I}$ is also extended to an Abox. It associates with each $a \in N_I$ some $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. An interpretation $\mathcal{I}$ satisfies an Abox if: (1) for each assertion $a : C$, we have that $a^{\mathcal{I}} \in C^{\mathcal{I}}$; and (2) for each assertion $(a, b) : R$, we have that $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. An interpretation that satisfies all the assertions in an Abox is called a *model* of the Abox. An Abox that has a model is said to be *consistent*.

The Abox *consistency test* is the fundamental Abox-related reasoning task, which checks if a given Abox is consistent.

1.3.3 Role Hierarchy

A *role hierarchy* (denoted by $\mathcal{H}$ as in $\mathcal{ALCHIQ}$ and $\mathcal{SHIQ}$) is a mechanism for specifying the subsumption relationships between roles. In addition to Tbox and ABox, a role hierarchy is sometimes regarded as a third component of a DL knowledge base and is called Rbox ("R" for Role). However, there is no recognized reasoning task on a Rbox itself. A role hierarchy specifies the subsumption relation between a pair of *roles* by using a *role inclusion* in the form of $R_1 \sqsubseteq R_2$, where R_1 and R_2 are roles. An interpretation $\mathcal{I}$ satisfies a role inclusion $R_1 \sqsubseteq R_2$ if $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$.

We should point out that this thesis is not intended for delineating the expressive differences for different description logic formalisms. General information about expressiveness can be found in [BCM+03] or at the website of "the DL Complexity Navigator" at www.cs.man.ac.uk/~ezolin/dl/. Therefore, a comparison between the expressiveness of *nominals* versa *Abox individuals* and others alike are out of the scope. In Section 1.4, small examples using description logic formalisms for knowledge representation will be given. By working through those small examples, inexperienced readers are expected to gain a better understanding of the notions central to description logics such as *concepts*, *concept inclusions*, and named *individuals* (which can be equivalently represented either in a *nominal* or in an *Abox individual*). One thing to notice is that (general) concept inclusions are used in all examples. We also expect that readers have no difficulty in finding out that an *Abox assertion* is a special form of a *concept inclusion* (Example 10 tries to give such a hint).

1.4 Examples

The convenience of using *inverse roles* has been long recognized. For example, it is difficult to directly express a sentence[4] such as "mary likes all cats" even in $\mathcal{SHO}$. By contrast, its equivalence to "all cats are liked by mary" can be easily expressed[5] in $\mathcal{ALCOI}$ by using the inverse $likes^-$ of the role $likes$.

[4]There is a paper [LS00] discussing the right description logic to express "mary likes all cats".

[5]In this thesis, we will show that inverse roles in examples like this can be eliminated, which means there exists an automatic and "polynomial" translation from $\mathcal{ALCOI}$ to $\mathcal{ALCO}$.

In the following, $\textsc{Cat}$, $\textsc{Mouse}$, $\textsc{Animal}$ and $\textsc{Female}$ are concepts, each of which is to be interpreted as a set of individuals; $\{Mary\}$, $\{Jessica\}$, and $\{Tom\}$ are named individuals (i.e., *nominals*, each of which is a singleton set); *likes* and *has_child* are role names, and $likes^-$ and has_child^- are their inverse roles respectively.

Example 1. "`Cat likes mouse`" can be expressed simply in $\mathcal{ALC}$ as:

$\textsc{Cat} \sqsubseteq \exists likes.\textsc{Mouse}$

Example 2. "No `Dog` likes a `Cat`" can be expressed either by

$\textsc{Dog} \sqsubseteq \forall likes.\neg\textsc{Cat}$ or by

$\exists likes^-.\textsc{Dog} \sqsubseteq \neg\textsc{Cat}$

Example 3. "`Mary likes` all `cats`, `Mary` is a `female` person and `cat` is an `animal`."

$\textsc{Cat} \sqsubseteq \textsc{Animal} \sqcap \exists likes^-.\{Mary\}$ and $\{Mary\} \sqsubseteq \textsc{Female}$

Example 4. "`Jessica likes` everything that `Mary likes`."

$\exists likes^-.\{Mary\} \sqsubseteq \exists likes^-.\{Jessica\}$

Example 5. "`Jessica` does not `like Mary`".

$\exists likes^-.\{Jessica\} \sqsubseteq \neg\{Mary\}$

Example 6. Given only the `has_child` relation, to express "`Mary` has at least five siblings", it is straightforward to use *inverse roles* and *number restrictions*:

$\{Mary\} \sqsubseteq \exists has\_child^-.\exists^{\geq 6} has\_child.\top$

Example 7. "`Mary likes` all her siblings" can be expressed in

$\neg\{Mary\} \sqcap \exists has\_child^-.(\exists has\_child.\{Mary\}) \sqsubseteq \exists likes^-.\{Mary\}$

Example 8. "Each of `Mary`'s siblings, if any, likes `Mary`". This can be expressed as:

$\neg\{Mary\} \sqcap \exists has\_child^-.(\exists has\_child.\{Mary\}) \sqsubseteq \exists likes.\{Mary\}$

or as:

$\{Mary\} \sqsubseteq \forall has\_child^-.(\forall has\_child.(\{Mary\} \sqcup \exists likes.\{Mary\}))$

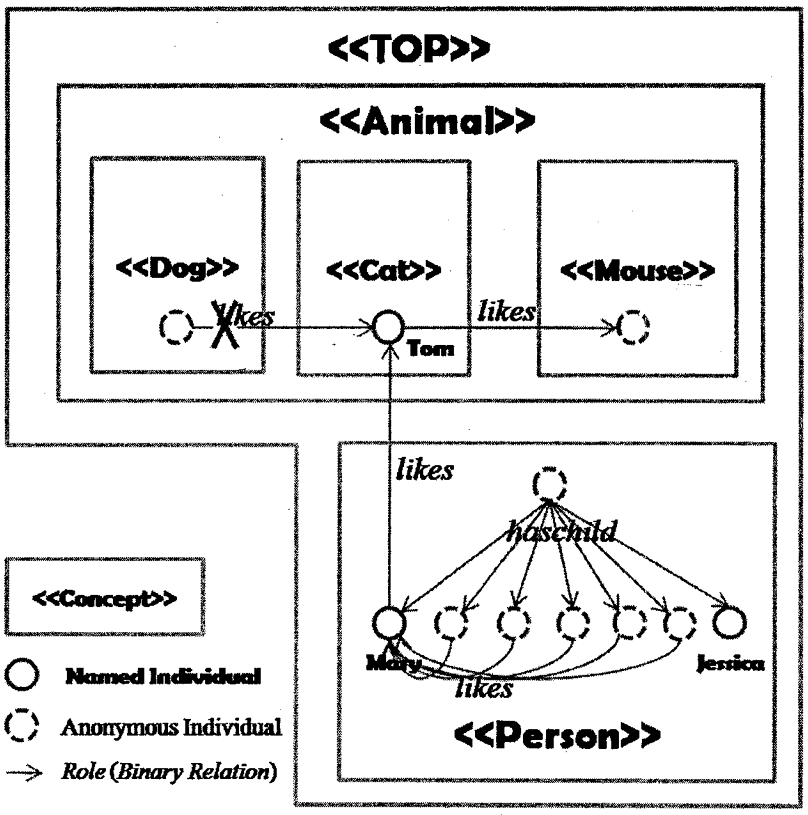


Figure 1.2: A Pictorial Presentation of (Partial) Knowledge of the Examples

Example 9. "`Jessica` and `Mary` share parents". This can be expressed by either:

$\{Mary\} \sqsubseteq \exists has\_child^{-}.(\exists has\_child.\{Jessica\})$ or by

$\{Jessica\} \sqsubseteq \exists has\_child^{-}.(\exists has\_child.\{Mary\})$

Example 10. "`Tom is a Cat`" can be expressed[6] either as an Abox assertion

$Tom : \textsc{Cat}$

or as a *concept inclusion* for a named individual (i.e., nominal)

$\{Tom\} \sqsubseteq \textsc{Cat}$

Given the above "knowledge" as encoded in the form of description logic *concept inclusions*, a logical consequence is that `Jessica` and `Mary` are the same person: `Jessica` and `Mary` share a common parent (according to Example 9) but can not be siblings (for otherwise there will be a contradiction by Example 8 and Example 5).

[6]This example shows that an *Abox assertion* about an Abox individual is equivalent to a *concept inclusion* about that named individual.

1.5 Research Motivation, Contribution and Thesis Organization

1.5.1 Research Motivation

The tableaux algorithm is a general technique for deciding *concept satisfiability problems* in description logics. It provides an algorithmic framework that is parametric with respect to language constructors, and is useful for studying the correctness and complexity of concrete decision procedures as well as for designing practical implementations. In practice highly optimized tableau-based algorithms have been implemented in DL inference engines. In the presence of *inverse roles* and *number restrictions* (corresponding to the ontology languages OWL-lite and OWL-DL respectively), however, most successful tableau-based DL reasoning systems exhibit problems, for example, degraded performance. The current loss of performance is largely due to the missing applicability of some well-known optimization techniques, especially the one for caching the satisfiability status of modal successors, and the new *absorption technique* which transforms GCIs into unfoldable axioms by considering the Ramsey-rule [Ram31] for a role and its inverse. For more discussions please see next subsection.

Our research initially started from carrying out a thorough investigation of the "current status" (about which techniques become invalid or less efficient) and performing an in-depth cause-analysis (of why current techniques for *inverse roles* and *number restrictions* are inefficient). During the course of this research, we have found and proposed several new techniques (including an equivalence about inverse relations, the *elimination of inverse roles*, and the *sound global tableaux caching* technique) as solutions. For *number restrictions*, an algebraic method was proposed in [HM01a] for $\mathcal{SHQ}$ and was implemented in the DL reasoner RACER [HM01b]. It is known from practice that the *algebraic method* has a good run-time performance. However, there was no theoretical explanation for this, thus it is questionable if the *algebraic method* is better than other approaches. Indeed in [Tob01] a question was raised about whether a tableau-based approach could lead to an optimal decision procedure for *concept satisfiability tests* of *number restrictions*. It was also unclear about how to use the *algebraic method* for DLs having *number restrictions* and *inverse roles*. These questions are answered in this thesis.

1.5.2 Contributions

This thesis is based on a continuous research being carried out at Concordia University since five years ago. Some research results in this thesis are collected from my thesis proposal and several publications [DH07b, DHW07, DH07a, DH06, DH05]. The contributions are as follows:

(1) The "dynamic blocking" technique [HS02] has been adapted to a *dynamic global tableaux caching* technique (as shown for the DL $\mathcal{ALCI}$ in this thesis). The proposed *dynamic global tableaux caching* technique (first appeared in [DH05]) exemplifies a general mechanism of "anywhere blocking"[7] for a family of DLs with *inverse roles.*

(2) An *equivalence*[8] about a *role* and its *inverse* was (re)discovered in the area of description logics during the course of this research. In [DH05] we pointed out its promising application to *absorption algorithms* [HT00]. A new framework [HW06] considered this equivalence and extended [HT00] in many aspects for designing *absorption algorithms.* Recently in [WH08], by combining with planning techniques, a general usage of this equivalence (allowing a recursive extraction of usable concept names) successfully transformed several very hard ontologies (from the model checking field [BDTW07]) into KBs without GCIs and showed orders of magnitude performance gain in experiments. Also related to this equivalence is a result [DHW07] showing that any unfoldable Tbox in $\mathcal{SHOI}$ can be transformed into an unfoldable Tbox in $\mathcal{SHO}$ while preserving *concept satisfiability.*

(3) A methodology to eliminate *inverse roles* for a family of DLs having inverse roles (e.g., from $\mathcal{SHOI}$ to $\mathcal{SHO}$, and from $\mathcal{ALCI}$ Abox to $\mathcal{ALC}$ Abox).

(4) A worst-case optimal (ExpTime) tableau-based decision procedure for $\mathcal{ALCFI}$ is established. This is achieved by a sound global tableaux caching technique that is the most powerful of the three tableaux caching techniques proposed in this thesis. It is quite promising that the same technique can be extended to $\mathcal{SHIF}$, a DL corresponding to the ontology language OWL-lite (see www.w3c.org).

(5) A simplified condition for the termination (a.k.a. blocking) of tableaux algorithms for DLs with *inverse roles* even when extended by *qualified number restrictions.* In contrast to the *dynamic double*

[7]Recently a *pairwise anywhere blocking* technique was introduced in [MSH07] for $\mathcal{SHIQ}$.

[8]This equivalence and the dynamic global tableaux caching technique were reported in the thesis proposal and were later presented in [DH05]. In 2006, the author of this thesis found out that this equivalence is a variant of the Ramsey Rule [Ram31] in modal logics. Related research for developing new *absorption algorithms* include [HW06, SGP06].

blocking conditions previously known and popularly recognized today [HS99], the new termination condition is not only conceptually simple but also (potentially) practically easy for implementations. A straightforward application of this termination condition is a sound global tableaux caching for DLs with *qualified number restrictions* and *inverse roles* (see below for $\mathcal{SHIQ}$). When connecting to the *depth-2 pattern* brought up in [BHLW03, HM04, Hla04], the new condition is intuitively understandable and fits right to the notion of the *depth-2 pattern.*

(6) Based on the *algebraic method* [CL94, OK99, HM01a] and the *integer linear programming* technique [Pap81], we demonstrate a worst-case ExpTime (with improved upper bound) tableau-based decision procedure for $\mathcal{SHIQ}$ in the strong sense of a binary coding of numbers. This is achieved by the use of sound global tableaux caching[9] and also by the use of a *feasibility test* for *integer linear inequalities.*

1.5.3 Thesis Organization

This thesis has seven chapters plus two appendices.

Chapter 1 is the present introduction.

Chapter 2 introduces a dynamic *global tableaux caching* technique for the description logic $\mathcal{ALCI}$. This is the first (but the least powerful) of three global tableaux caching techniques to be presented in this thesis.

In Chapter 3, we first show an equivalence about inverse relations which is a variant of the Ramsey-rule [Ram31]. Then we introduce a translation technique to convert an $\mathcal{ALCI}$ knowledge base (an Abox plus an acyclic Tbox) to an $\mathcal{ALC}$ knowledge base. The Abox consistency problem with regard to an acyclic Tbox for a description logic with inverse roles is reduced to the same problem in a description logic without inverse roles.

In Chapter 4, we present a second *global tableaux caching* technique, the most powerful one of the three caching techniques proposed in this thesis. We further show an ExpTime decision procedure for the satisfiability problem in $\mathcal{ALCFI}$.

[9]The *pairwise anywhere blocking* technique introduced in [MSH07] for $\mathcal{SHIQ}$ is not able to guarantee a worst-case ExpTime tableau-based decision procedure even in unary coding of numbers.

Chapter 5 presents several reductions for eliminating the role hierarchy and inverse roles. This chapter is related to Chapter 3 and also shows that satisfiability problems in $\mathcal{SHI}$ can be reduced to those in $\mathcal{ALC}$.

Chapter 6 covers an upper-bound improved ExpTime decision procedure for $\mathcal{SHIQ}$. We introduce the *algebraic method* [OK99, HM01a], the *integer linear programming* technique, the simplified termination condition as mentioned above, and then provide a third *global tableaux caching* technique, which is the most general one of the three variants (of the *global tableaux caching* technique) to be presented in this thesis.

Chapter 7 summarizes the thesis and discusses future work.

Appendix A is the result from the first phase of this research and reviews optimized tableaux algorithms for description logics.

Appendix B presents the experimental results about the elimination of inverse roles.

Figure 1.3 illustrates the connections among different chapters and appendices of this thesis. For example, Appendix A provides some extra background to Chapter 1, hence it might be better to have a look at Appendix A when reading Chapter 1. Chapter 2, Chapter 4 and Chapter 6 share commonalities in presenting tableau-based algorithms; Chapter 3 and Chapter 5 share commonalities in using a reduction technique. Therefore, it might be helpful for readers to read them in groups. It is hoped that a simple navigational map like this would help readers go through this thesis in a more efficient way.

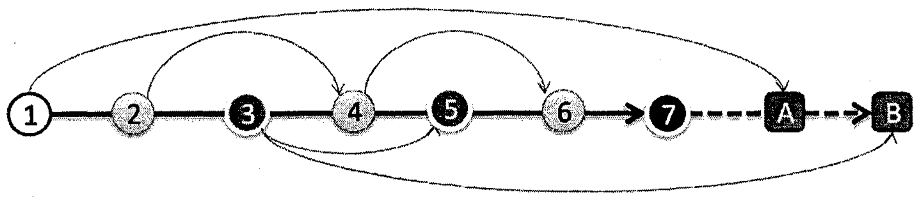


Figure 1.3: Organization of the Chapters and Appendices

Chapter 2

Dynamic Tableaux Caching for $\mathcal{ALCI}$

Modern description logic (DL) reasoners are known to be less efficient for DLs with inverse roles [DH05, GN07]. The current loss of performance is usually caused by the missing applicability of well-known optimization techniques such as caching the satisfiability status of modal successors. The "unsoundness" of (global) tableaux caching in the presence of inverse roles is the very reason for people to dismiss its "applicability" [HST99a, DM00, HM00a]. To recognize and avoid unsound cases in tableaux caching is therefore a first step toward a correct use of the global tableaux caching technique. This chapter studies the correct conditions for using tableaux caching in $\mathcal{ALCI}$ and shows a *global tableaux caching* technique that is dynamic[1] and sound.

As shown in Chapter 1, $\mathcal{ALCI}$ has propositional constructs, universal restrictions, and existential restrictions. We assume readers are familiar with its syntax and semantics.

In this chapter (and throughout this thesis) we will use two conventions.

The first is about a shorthand notation for inverse roles. To express the inverse of a role R, we simply denote it by R^-. In some published work, an auxiliary function $\texttt{Inv}(.)$ is used instead for denoting the inverse of a role. Though $\texttt{Inv}(R)$ could possibly denote an inverse of the role R in a more precise way, it is less convenient in writing and does not give better clarity than the simple

[1]The name "dynamic caching" is taken after the "dynamic blocking" technique in [HS99]. In here and there the word "dynamic" is used to denote a relationship (in tableaux structures) that should be dynamically updated and maintained by the proposed techniques.

notation R^-. For a role S, if it is the inverse of a role R, then its inverse S^- will mean R. Therefore, readers are not expected to see notations such as R^{--} in this thesis.

The second convention is related to discussing tableaux structures. It is sometimes inevitable to refer to neighborhood relations between tableaux nodes. In the literature, it is common to see statements like "x is an R-neighbor of y", which can be easily understood as saying "x is y's R-neighbor", or put in another way, x and y are two adjacent nodes and there is an R relation from y to x. To be consistent, when we say that x is an R-predecessor of y, we mean that x is y's predecessor and there is a relation R from y to x. Likewise, "x is an R-successor of y" means that x is y's successor and there is a relation R from y to x.

2.1 The Tableaux Rules

The notions of the $\mathcal{C}$-label (initial label) and the *unsat-cached clash* will be used.

The $\mathcal{C}$-label is used to keep those information propagated down the tableaux tree; while the $\mathcal{L}$-label is conventional and contains all information. It is required that $\mathcal{C}(x) \subseteq \mathcal{L}(x)$. As usual, a tableaux node x is called *completed* or *saturated* if no *tableaux expansion rule* is applicable to $\mathcal{L}(x)$. In $\mathcal{ALCI}$, the two-way computations on a *tableaux tree*, as caused by the back-propagation due to *universal restrictions*, should be taken into account to accordingly reflect the satisfiability of $\mathcal{ALCI}$ concepts.

Recall that in [BCM+03] the basic *clash trigger* is defined as $\{A, \neg A\}$, where A is some concept name. As a reminder, $\bot$ is used as an abbreviation for $A \sqcap \neg A$. To integrate a *global tableaux caching* functionality into the *tableau expansion rules*, a second *clash trigger* is needed. When creating a successor node y for a node x, if there exists some label $\mathcal{L}(z)$ known to be unsatisfiable and $\mathcal{L}(z) \subseteq \mathcal{L}(y)$, then a clash trigger is applicable to node y's $\mathcal{L}$ label; likewise, if there exists some label $\mathcal{C}(z)$ known to be unsatisfiable and $\mathcal{C}(z) \subseteq \mathcal{C}(y)$, then a clash trigger is applicable to node y's $\mathcal{C}$ label. A clash of this kind is called an *unsat-cached clash.*

In the following, we present several auxiliary functions that will be used later.

Definition 1. (Negation Normal Form) The Negation Normal Form (NNF) $nnf(.)$ of a concept expression is defined as

(1) $nnf(A) = A$ and $nnf(\neg A) = \neg A$ for concept name $A \in N_C$,

(2) $nnf(\neg(\neg C)) = nnf(C)$,

(3) $nnf(C \sqcap D) = nnf(C) \sqcap nnf(D)$ and $nnf(\neg(C \sqcap D)) = nnf(\neg C) \sqcup nnf(\neg D)$

(4) $nnf(C \sqcup D) = nnf(C) \sqcup nnf(D)$ and $nnf(\neg(C \sqcup D)) = nnf(\neg C) \sqcap nnf(\neg D)$

(5) $nnf(\forall R.C) = \forall R.(nnf(C))$ and $nnf(\neg(\forall R.C)) = \exists R.(nnf(\neg C))$

(6) $nnf(\exists R.C) = \exists R.(nnf(C))$ and $nnf(\neg(\exists R.C)) = \forall R.(nnf(\neg C))$

Definition 2. (function sub(.)) The set of concepts *sub*(.) is defined as follows:

(1) $sub(A) = \{A\}$ and $sub(\neg A) = \{\neg A\}$ for concept name $A \in N_C$

(2) $sub(C \sqcap D) = \{C \sqcap D\} \cup sub(C) \cup sub(D)$

(3) $sub(C \sqcup D) = \{C \sqcup D\} \cup sub(C) \cup sub(D)$

(4) $sub(\forall R.C) = \{\forall R.C\} \cup sub(C)$

(5) $sub(\exists R.C) = \{\exists R.C\} \cup sub(C)$

The following is a definition of the *tableaux structure.* As usual, in the definition only $\mathcal{L}$-labels are used. It should be clear that $\mathcal{C}$-labels are used in identifying safe conditions (for sound tableaux caching) in Definition 6.

Definition 3. ($\mathcal{ALCI}$ **Tableau Structure)** Let E be an $\mathcal{ALC}$ concept in NNF, R_E be the set of roles occurring in E. A *tableau structure* for E is a triple $(\mathcal{S}, \mathcal{L}, \mathcal{E})$, where $\mathcal{S}$ is a set of individuals, $\mathcal{L} : \mathcal{S} \to 2^{sub(E)}$, $\mathcal{E} : R_E \to 2^{\mathcal{S} \times \mathcal{S}}$, and the following properties hold:

(1) $\bot \notin \mathcal{L}(s)$, and if $C \in \mathcal{L}(s)$, then $\neg C \notin \mathcal{L}(s)$,

(2) if $C \sqcap D \in \mathcal{L}(s)$, then $C \in \mathcal{L}(s)$ and $D \in \mathcal{L}(s)$,

(3) if $C \sqcup D \in \mathcal{L}(s)$, then $C \in \mathcal{L}(s)$ or $D \in \mathcal{L}(s)$,

(4) if $\forall R.C \in \mathcal{L}(s)$ and $\langle s, t\rangle \in \mathcal{E}(R)$, then $C \in \mathcal{L}(t)$,

(5) if $\exists R.C \in \mathcal{L}(s)$, then there is some $t \in \mathcal{S}$ s.t. $\langle s, t\rangle \in \mathcal{E}(R)$ and $C \in \mathcal{L}(t)$.

(6) $\langle s, t\rangle \in \mathcal{E}(R)$ iff $\langle t, s\rangle \in \mathcal{E}(R^-)$.

Below are two functions *succ*(.,.) and *watch*(.). Both functions help in characterizing the influence of *universal restrictions* in a tableaux structure. The function *succ*(.,.) specifies a set of role fillers; the function *watch*(.) specifies a set of roles. They are used to form the (sound tableaux caching) preconditions in Definition 6. A label $\mathcal{L}(x)$ is locally completed if the ⊓-rule and the ⊔-rule are not applicable to it. In subsequent chapters, this local completion will also be called propositionally completed to reflect its relation to the propositional expansion rules.

Definition 4. (function succ(.,.)) Given a node x with a pair of labels $\langle \mathcal{C}(x), \mathcal{L}(x) \rangle$, where $\mathcal{L}(x)$ is a locally completed label, for R a role, the function $succ(\mathcal{L}(x), R) = \{C \mid \forall R^-.C \in \mathcal{L}(x)\}$ specifies a set of role fillers of the relevant universal restrictions.

Definition 5. (function watch(.)) Given a tableau node x with $\{\mathcal{C}(x), \mathcal{L}(x)\}$, where $\mathcal{L}(x)$ is the (locally) completed label as prescribed above, the set of watched incoming-edges for x is $watch(\mathcal{L}(x)) = \{R^- \mid \forall R.C \in \mathcal{L}(x)\}$.

Definition 6. (Sat-Cached) Given a node x with $\{\mathcal{C}(x), \mathcal{L}(x)\}$, and its R-successor y with $\{\mathcal{C}(y), \mathcal{L}(y)\}$, and some node z with $\{\mathcal{C}(z), \mathcal{L}(z)\}$ such that $\mathcal{L}(z)$ is completed and does not contain a clash. The node y is sat-cached by node z if one of the following conditions[2] holds:

(1) $\mathcal{C}(y) \subseteq \mathcal{C}(z)$ and $R \notin watch(\mathcal{L}(z))$; or

(2) $\mathcal{L}(y) \subseteq \mathcal{L}(z)$ and $R \notin watch(\mathcal{L}(z))$; or

(3) $\mathcal{C}(y) \subseteq \mathcal{C}(z)$ and $succ(\mathcal{L}(z), R) \subseteq \mathcal{L}(x)$; or

(4) $\mathcal{C}(y) \subseteq \mathcal{C}(z)$, z is a R-successor to w, $\mathcal{L}(w) \subseteq \mathcal{L}(x)$, $succ(\mathcal{L}(z), R) \subseteq \mathcal{L}(w)$; or

(5) Both y and z are x's R-successors and $\mathcal{C}(y) \subseteq \mathcal{C}(z)$.

Table-2.1 is a set of tableaux expansion rules for $\mathcal{ALCI}$ with caching integrated. The first half of the ∀-rule is for backward propagation and the second half for forward propagation. The ∃∀-rule generates successor nodes and creates their initial labels.

[2]Following the fundamental idea of "dynamic blocking", it is sufficient to use condition-3 as the only terminating condition for the underlying tableaux procedure. Nonetheless, considering several other cases might benefit run time performance further. For example, condition-5 is quite special and in some situations it can significantly prune the search space; while condition-1 and condition-2 could lead to static global tableaux caching [DH05].

$\sqcap$-rule	if	1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, and
		2. $\{C_1, C_2\} \cap \mathcal{L}(x) \neq \{C_1, C_2\}$
	then	$\mathcal{L}(x) = \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule	if	1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, and
		2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
	then	$\mathcal{L}(x) = \mathcal{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$
$\forall$-rule	if	1.1 $\forall R.C \in \mathcal{L}(x)$, and
		1.2 there is an R-predecessor y of x with $C \notin \mathcal{L}(y)$
	then	$\mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
	if	2.1 $\forall R.C \in \mathcal{L}(x)$, and
		2.2 there is an R-successor y of x with $C \notin \mathcal{C}(y)$
	then	$\mathcal{C}(y) = \mathcal{C}(y) \cup \{C\}, \mathcal{L}(y) = \mathcal{L}(y) \cup \{C\}$
$\exists\forall$-rule	if	1. $\exists R.C \in \mathcal{L}(x)$, and
		2. x is not sat-cached, and
		3. x has no R-neighbor y with $C \in \mathcal{C}(y)$
	then	create a R-successor y such that
		$\mathcal{L}(y) = \mathcal{C}(y) = \{C\} \cup \{D \mid \forall R.D \in \mathcal{L}(x)\}$ and $\mathcal{L}(\langle x, y\rangle) = R$

Table 2.1: Tableau rules for $\mathcal{ALCI}$ with an integrated generalized cache.

Proposition 7. The above tableau procedure is sound for the satisfiability testing of $\mathcal{ALCI}$ concepts.

Proof. The tableaux rules given in Table 2.1 are commonly seen in the literature except the sat-cached conditions. Its soundness therefore largely relies on the correctness of the five sat-cached conditions. The following is a proof for each case.

(case 1) Figure 2.1 illustrates the first case and gives an example.

Considering the facts that $\mathcal{L}(z)$ is a completed label supporting the satisfiability of $\mathcal{C}(z)$ and the first precondition $\mathcal{C}(y) \subseteq \mathcal{C}(z)$, it is trivial to see $\mathcal{L}(z)$ supports the satisfiability of $\mathcal{C}(y)$. Unless z can not be completed[3], it is guaranteed that no clash is introduced for y in the final tableau structure.

The second precondition $R \notin watch(\mathcal{L}(z))$ further guarantees that no *universal restriction* like $\forall R^{-}.C$ occurs in $\mathcal{L}(y)$ (an $\mathcal{L}$-label for y supporting the satisfiability of $\mathcal{C}(y)$). Therefore, the expansion of $\mathcal{C}(y)$ (to $\mathcal{L}(y)$) does not incur backward propagation of constraints to its predecessor node x.

[3]If $\mathcal{L}(z)$ can not be completed in the end, it means that $\mathcal{L}(z)$ is not satisfiable. In this case, the "relation" of y being cached on z should be invalidated. A label $\mathcal{L}(y)$ will be computed from $\mathcal{C}(y)$ and subject to tableaux expansion rules. Readers are now expected to understand better the meaning of "dynamic" as in the name dynamic tableaux caching.

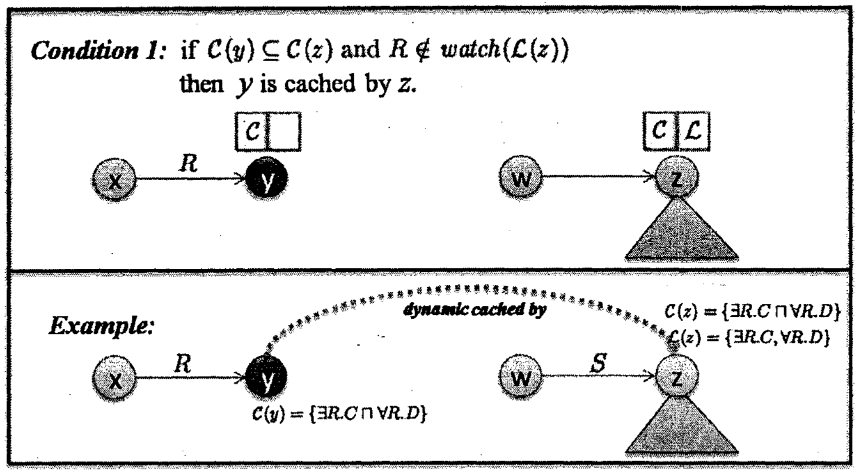

Condition 1: if $\mathcal{C}(y) \subseteq \mathcal{C}(z)$ and $R \notin watch(\mathcal{L}(z))$
then y is cached by z.
$\mathcal{C}$
$\mathcal{C}$ $\mathcal{L}$
x
R
y
w
z
Example:
dynamic cached by
$\mathcal{C}(z) = \{\exists R.C \sqcap \forall R.D\}$
$\mathcal{L}(z) = \{\exists R.C, \forall R.D\}$
S
$\mathcal{C}(y) = \{\exists R.C \sqcap \forall R.D\}$

Figure 2.1: Dynamic Caching: Condition 1

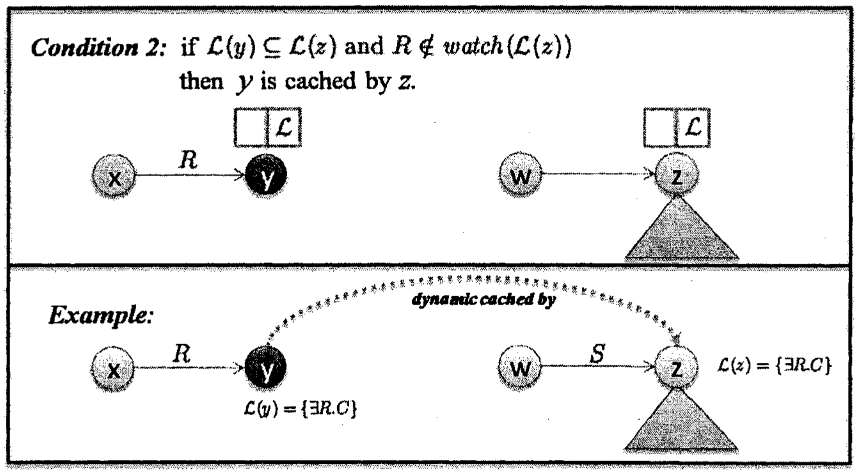

Condition 2: if $\mathcal{L}(y) \subseteq \mathcal{L}(z)$ and $R \notin watch(\mathcal{L}(z))$
then y is cached by z.
$\mathcal{L}$
$\mathcal{L}$
x
R
y
w
z
Example:
dynamic cached by
S
$\mathcal{L}(z) = \{\exists R.C\}$
$\mathcal{L}(y) = \{\exists R.C\}$

Figure 2.2: Dynamic Caching: Condition 2

(case 2) Figure 2.2 shows the second case and gives an example. The proof relies on the construction of a sub-tableau T' for y such that no extra constraints will be introduced to $\mathcal{L}(x)$ due to backward propagation. Since $\mathcal{L}(z)$ is given as a (locally) completed label, $T'(y)$ can be copied from a subset of $T(z)$ so that no backward propagation to x will be introduced.

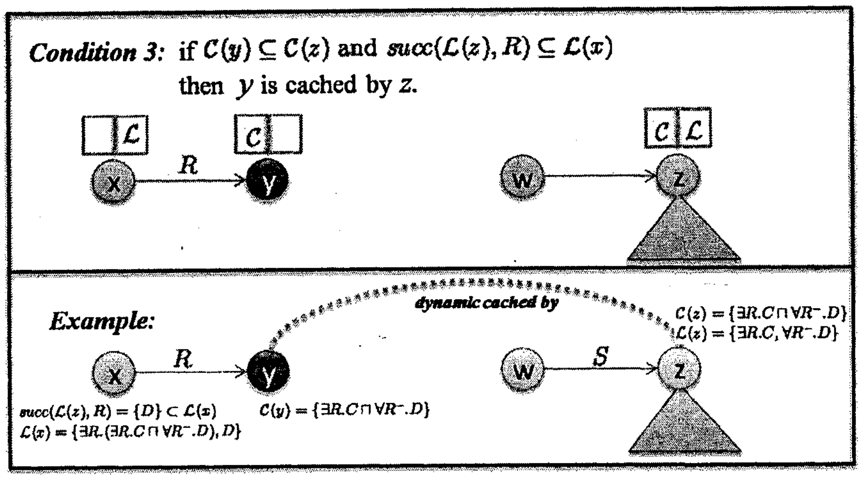

Condition 3: if $\mathcal{C}(y) \subseteq \mathcal{C}(z)$ and $succ(\mathcal{L}(z), R) \subseteq \mathcal{L}(x)$
then y is cached by z.
$\mathcal{L}$
$\mathcal{C}$
$\mathcal{C}$ $\mathcal{L}$
x
R
y
w
z
Example:
dynamic cached by
$\mathcal{C}(z) = \{\exists R.C \sqcap \forall R^-.D\}$
$\mathcal{L}(z) = \{\exists R.C, \forall R^-.D\}$
x
R
y
w
S
z
$succ(\mathcal{L}(z), R) = \{D\} \subset \mathcal{L}(x)$
$\mathcal{L}(x) = \{\exists R.(\exists R.C \sqcap \forall R^-.D), D\}$
$\mathcal{C}(y) = \{\exists R.C \sqcap \forall R^-.D\}$

Figure 2.3: Dynamic Caching: Condition 3

(case 3) Figure 2.3 shows the third case[4] and gives an example. Given $\mathcal{C}(y) \subseteq \mathcal{C}(z)$ and a completed label $\mathcal{L}(z)$, there must exist an $\mathcal{L}$-label for y s.t. $\mathcal{L}(y) \subseteq \mathcal{L}(z)$. So, for any $\forall R^-.C$, if $\forall R^-.C \in \mathcal{L}(y)$, it must be the case that $\forall R^-.C \in \mathcal{L}(z)$. Given $succ(z, R) \subseteq \mathcal{L}(x)$ as a known pre-condition, it means that for any $\forall R^-.C \in \mathcal{L}(z)$ it holds that $C \in \mathcal{L}(x)$. So, for any $\forall R^-.C$, if $\forall R^-.C \in \mathcal{L}(y)$, then it is true that $C \in \mathcal{L}(x)$. To construct the *tableau structure* T, we need $\mathcal{E}(R) \supseteq \{(x, z) \in \mathcal{S} \times \mathcal{S} \mid \mathcal{L}(\langle x, y \rangle)$ is an R-edge and y is Sat-Cached by $z\}$ to reflect **case 3**.

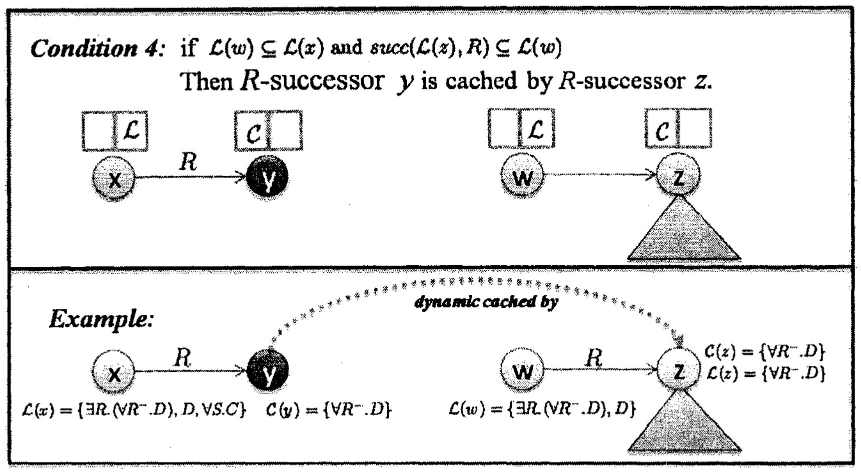

Condition 4: if $\mathcal{L}(w) \subseteq \mathcal{L}(x)$ and $succ(\mathcal{L}(z), R) \subseteq \mathcal{L}(w)$
Then R-successor y is cached by R-successor z.
$\mathcal{L}$
$\mathcal{C}$
$\mathcal{L}$
$\mathcal{C}$
x
R
y
w
z
Example:
dynamic cached by
x
R
y
w
R
z
$\mathcal{C}(z) = \{\forall R^-.D\}$
$\mathcal{L}(z) = \{\forall R^-.D\}$
$\mathcal{L}(x) = \{\exists R.(\forall R^-.D), D, \forall S.C\}$
$\mathcal{C}(y) = \{\forall R^-.D\}$
$\mathcal{L}(w) = \{\exists R.(\forall R^-.D), D\}$

Figure 2.4: Dynamic Caching: Condition 4

[4]A variant of this case is $\mathcal{L}(y) \subseteq \mathcal{L}(z)$ and $succ(\mathcal{L}(z), R) \subseteq \mathcal{L}(x)$, which resembles condition 2.

(case 4) Figure 2.4 shows the fourth case and gives an example. As already stated in the definition, the witness z is locally saturated (for no rule is applicable to $\mathcal{L}(z)$ any more) and does not contain a clash, and so does its predecessor w. This means that $succ(\mathcal{L}(z), R) \subseteq \mathcal{L}(w) \subseteq \mathcal{L}(x)$. According to **case 3**, y is sat-cached by z.

(case 5) This is a case in which two nodes share a common parent and each of them has the same role relation to the parent. If the two sibling nodes z and y satisfy $\mathcal{C}(y) \subseteq \mathcal{C}(z)$, then only node z need to be expanded. In other words, if a label $\mathcal{C}(z)$ is satisfiable, then so is the label $\mathcal{C}(y)$. In DLs having no inverse roles, global tableaux caching techniques commonly use set inclusion relationship among initial labels. In $\mathcal{ALCI}$, generally it would be unsound to use initial labels for global tableaux caching. However, this case shows an exception.

The above constitutes the soundness proof. □

2.2 Summary

In this chapter, a dynamic global *sub-tableaux caching* technique is presented for $\mathcal{ALCI}$. The relation between a witness node and its blocked nodes is specified in a global scope so that it is possible to cache intermediate computations (satisfiable results) during the execution of tableau-based procedures. However, the "blocking" relation here is dynamic in that whenever the precondition changes the "node" being "blocked" then it should be unblocked and the *tableaux expansion rules* should be applied to them. This caching technique is dynamic in nature and is a generalization of the *dynamic blocking* technique [HS99].

The proposed technique uses a pair of labels per tableau node. The idea of using more than one label per node is not new and readers are referred to [HST98]. The label $\mathcal{L}(y)$ is required to be locally (i.e., propositionally) completed, while $\mathcal{C}(y)$ is the initial label of y. The obvious weakness is that no inconsistency propagation is provided. In this sense, this dynamic caching technique is the weakest of the three caching techniques proposed in this thesis. The next chapter (Chapter 3) presents a conversion from $\mathcal{ALCI}$ to $\mathcal{ALC}$ so that a full *inconsistency propagation* in $\mathcal{ALC}$ can be exploited. In Appendix B, we show that by utilizing a fully-fledged *global tableaux caching* capability (with inconsistency propagation enabled), the tableaux algorithm runs faster in orders of magnitude. In

Chapter 4, we will present a second *global tableaux caching* technique for $\mathcal{ALCFI}$ which integrates the capability of inconsistency propagation as if there were no inverse roles.

Chapter 3

Inverse Role and $\mathcal{ALCI}$ Abox

In this chapter, first we show an equivalence on inverse relations, and then show we may convert an $\mathcal{ALCI}$ Abox to an $\mathcal{ALC}$ Abox.

3.1 An Equivalence on Inverse Relation

Due to lack of appropriate optimization techniques in the presence of *inverse roles*, DL tableau-based procedures are very sensitive in their performance but resolution-based approaches to DL reasoning are relatively unaffected [Tsa03]. Therefore it might be beneficial for us to consider the fragment of *first order logic* (FO) translated from expressions of the basic *description logics* with *inverse roles*, i.e., $\mathcal{ALCI}$.

(1) Firstly, let us consider $C \sqsubseteq \forall R.D$.

Based on the standard semantics, a translation into FO would look like:

$\forall x(c(x) \rightarrow (\forall y(r(x,y) \rightarrow d(y))))$.

Its *prenex form* is $\forall x \forall y(\neg c(x) \vee \neg r(x,y) \vee d(y))$.

(2) Secondly, let us consider $\neg D \sqsubseteq \forall R^{-}.\neg C$. The corresponding FO expression is:

$\forall z(\neg d(z) \rightarrow (\forall w(r^{-}(z,w) \rightarrow \neg c(w))))$.

Its *prenex form* is $\forall z \forall w(d(z) \vee \neg r^{-}(z,w) \vee \neg c(w))$.

It is easy to see that $\forall z \forall w(d(z) \vee \neg r^{-}(z,w) \vee \neg c(w))$

$$\Leftrightarrow \forall z \forall w(d(z) \vee \neg r(w,z) \vee \neg c(w))$$

$$\Leftrightarrow \forall z \forall w(\neg c(w) \vee \neg r(w,z) \vee d(z)).$$

Based on the *variable substitution* $\{x/w, y/z\}$, it is easy to see that the two FO formulae are logically equivalent. In this sense, we can conclude that $C \sqsubseteq \forall R.D$ and $\neg D \sqsubseteq \forall R^{-}.\neg C$ are equivalent w.r.t. the above mentioned translation into first-order logic.

The above analysis shows a property that the *universal restriction* has and that regards the *inverse relationship* in a context of *concept inclusion axioms* or *general concept inclusions*. We also notice that it is the *role filler* of a *universal restriction* that plays a function in the inverse relation. This equivalence was first presented in [DH05] and was motivated by observing that current absorption algorithms largely fail to consider this equivalence in their absorption process[1].

In this chapter, we show how to use the equivalence to eliminate *inverse relations* for an $\mathcal{ALCI}$ Abox in presence of an acyclic Tbox. It consists of three simple steps (i.e., *tagging*, *recording*, and *polarisation*).

3.2 Abox Consistency with Acyclic Tbox

3.2.1 The Intuition Behind the Conversion

A Tbox (a.k.a. *terminological box*) is a set of unfoldable axioms (*concept definitions* or *concept inclusions*). By the standard semantics, *concept definitions* like $A \equiv C$ are expressed as two *inclusion axioms* $A \sqsubseteq C$ and $\neg A \sqsubseteq \neg C$ [BCM+03]. For an acyclic Tbox, if it has only *concept inclusions* and the right-hand-side of each *concept inclusion* is in *negation norm form*, then it is a *simplified Tbox* [Lut99]. Recall that an acyclic Tbox does not allow a concept name (or its negation) to refer to itself (or its negation) either directly or indirectly. This restriction at the syntax level is an important factor to be considered when it comes to the PSPACE decidability for a family of DLs. Below, we introduce two notions that characterize such "reference relationship" occurring at the syntax level for an acyclic Tbox.

Definition 1. (Modal Depth) The depth *depth*(.) of a concept is defined as:

(1) for each axiom $A \sqsubseteq C$, it holds that $depth(A) = depth(C)$;

[1] A presentation was given in DL Workshop 2005 with an example to show the "optimal" result of an old *absorption algorithm* can actually be further optimized when taking the equivalence into consideration. This equivalence was named as C-rule by the author in various events. In November 2006, the author found that this equivalence actually is a syntactic variant of the well-known Ramsey-rule pointed out by Frank P. Ramsey [Ram31] in 1925. New *absorption algorithms* [HW06, WH08, SGP06] considered this equivalence.

(2) $depth(C \sqcap D) = max(depth(C), depth(D))$;

(3) $depth(C \sqcup D) = max(depth(C), depth(D))$;

(4) $depth(\exists R.C) = depth(C) + 1$;

(5) $depth(\forall R.C) = depth(C) + 1$.

(6) for an atomic concept A not occurring on the lhs of any axiom, $depth(A) = 0$.

Also related to Tboxes is the notion of *acyclic ordering*, which is defined as follows.

Definition 2. (Acyclic Ordering) The ordering relation[2] $\succ$ is as follows:

(1) for each axiom $A \sqsubseteq C$ or $\neg A \sqsubseteq C$, there is $ord(A) \succ ord(C)$;

(2) $ord(C \sqcap D) \succ ord(C)$ and $ord(C \sqcap D) \succ ord(D)$;

(3) $ord(C \sqcup D) \succ ord(C)$ and $ord(C \sqcup D) \succ ord(D)$;

(4) $ord(\exists R.C) \succ ord(C)$;

(5) $ord(\forall R.C) \succ ord(C)$.

The notion of *modal depth* and the notion of *acyclic ordering* are extensively used in the literature for proving termination in decision procedures dealing with *acyclic Tboxes.* The *acyclic Tbox* is commonly regarded as a shorthand and a compact notation for *concept expressions.* It is clear that at the syntax level a *concept expression* (with respect to an acyclic Tbox) can be put in a tree shape, as Figure 3.1 shows. For convenience, *existential restrictions* and *universal restrictions* are also called *modal constraints*; others are called *propositional constraints.*

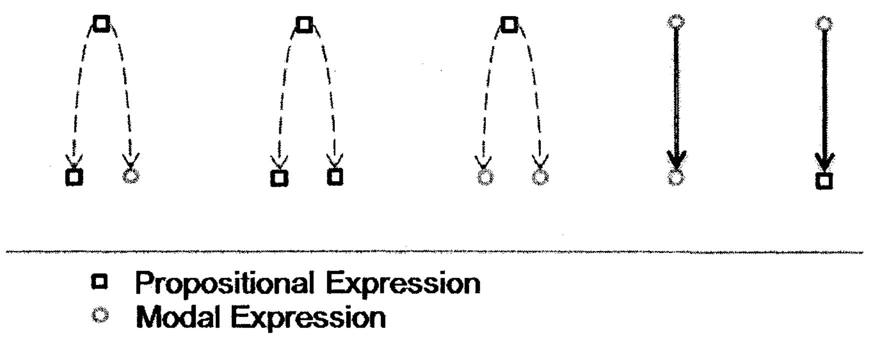


Figure 3.1: The Syntax Tree of Concept Expressions

[2]Due to acyclicity, $ord(A) \succ ord(A)$ is not induced.

For all possible *concept expressions*, Figure 3.1 shows several common patterns. The small square stands for "propositional expression" and the small circle stands for "modal expression." Recalling from Chapter 1 about DL syntax, it is the mutually recursive reference (of a propositional expression to a modal expression and vice versa) that makes up a complex *concept expression*. For acyclic Tboxes, it is important to use notions of *modal depth* and *acyclic ordering* when considering (direct) "reference relationships" among concept expressions at the syntax level [Lut99, BML+05]. However, for an acyclic Tbox in a DL with *inverse roles*, we need to consider the *modal depth* or *acyclic ordering* in a slightly extended way. We need to consider all "reference relationships" (e.g., indirect reference) at the syntax level except the impossible references such as a sub-expression referring to its (syntactical) parent expression. For example, because $ord(\exists R.\exists S.\forall S^{-}.C) \succ ord(C)$, we need to consider the "indirect reference" of $\exists R.\exists S.\forall S^{-}.C$ to C.

An Abox (a.k.a. *Assertional box*) consists of *individual assertions* and *role assertions*.

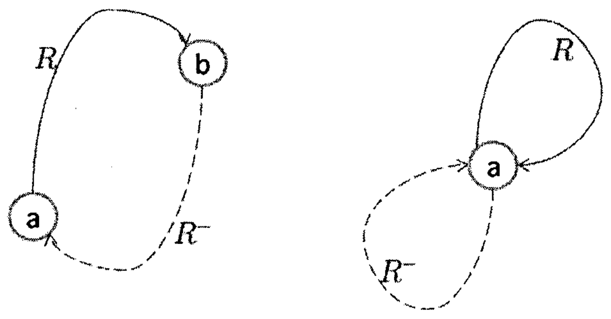


Figure 3.2: The Role Assertions In Abox

Figure 3.2 shows how changes are made to Abox individuals a and b. On the left, two Abox individual a and b are connected with a role assertion $(a, b) : R$. By conversion, we additionally append a second role assertion which says $(b, a) : R^{-}$. For the figure on the right, an Abox individual a has a self-loop $(a, a) : R$, similarly one role assertion $(a, a) : R^{-}$ is added. Note that the newly introduced role assertions are redundant.

Figure 3.3 shows the *polarisation* of Figure 3.2. Note that each R-edge and R^{-}-edge is replaced with R^{a}-edge and R^{b}-edge respectively, where R^{a} is a new role name in $\mathcal{ALC}$ for role R in $\mathcal{ALCI}$, and R^{b} is a new role name in $\mathcal{ALC}$ for role R^{-} in $\mathcal{ALCI}$.

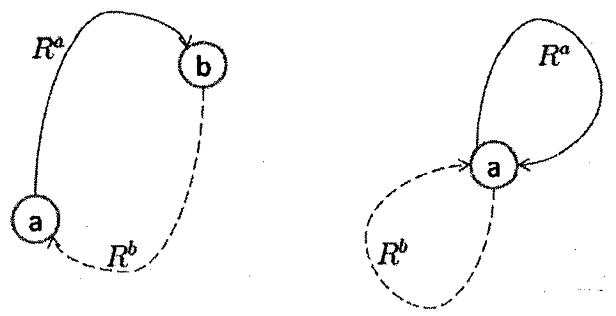


Figure 3.3: The Role Assertions In Abox After Polarisation

We assume one individual has at most one *individual assertion*, because two *individual assertions* $d : C$ and $d : D$ can be replaced by a single *assertion* $d : C \sqcap D$. If an Abox has several unconnected components, each of them can be treated alike separately. Without loss of generality, we consider[3] a single-component Abox $\mathcal{A}_0$ and each individual has at most one assertion. We denote the label (i.e., individual assertion) of d_i as $\mathcal{L}(d_i)$.

Below we introduce three simple steps (i.e., *tagging*, *recording*, and *polarisation*) and show how to perform them sequentially on an Abox and an acyclic Tbox.

3.2.2 The Three Steps

The first step is called *tagging.* Intuitively, it assigns each *modal constraint* a new *concept name.* It should be clear that the tagging operation changes neither the acyclic ordering nor the modal depth.

Definition 3. (Tagging) The function $tag(x)$ is defined as follows:

(1) if x is individual d_i with $\mathcal{L}(d_i)$, then $tag(x) = d_i : P(x) \sqcap\ tag(\mathcal{L}(d_i))$;

(2) if x is an individual d_i without a label, then $tag(x) = d_i : P(x)$;

(3) if x is $C \sqcap D$, then $tag(x) = tag(C) \sqcap tag(D)$;

(4) if x is $C \sqcup D$, then $tag(x) = tag(C) \sqcup tag(D)$;

(5) if x is $\exists R.C$, then $tag(x) = P(x) \sqcap\ \exists R.(tag(C))$;

(6) if x is $\forall R.C$, then $tag(x) = P(x) \sqcap \forall R.(tag(C))$;

(7) if x is $A \sqsubseteq C$, then $tag(x) = A \sqsubseteq tag(C)$;

(8) otherwise, $tag(x) = x$;

[3]Without making such an assumption, an alternative to get a single component is to introduce a special *individual* and connect it to all other *individuals* with a special *role.* This resembles the use of a *nominal* to *internalize* an Abox.

where $P(x)$ is a unique name for each x.

The original Abox/Tbox are denoted as $\mathcal{A}_0/\mathcal{T}_0$, and their tagged counterparts are denoted as $\mathcal{A}_1/\mathcal{T}_1$. Notice that we do not tag *role assertions*. We also write $P(d_i)$ instead of $P(x)$ if the tag is for an individual d_i. The set of tags for all individuals of the Abox is $\mathcal{D} = \{P(d_i) | d_i \in \mathcal{A}_0\}$. We introduce two sets of constraints $\mathcal{U}(.)$ and $\mathcal{E}(.)$. Let C denote any (sub)formula. Then by the tagging operation above, we have:

($\star$) for $x = \exists R.C$, there is $P(x) \sqcap \exists R.tag(C) \in \mathcal{E}(R)$;

($\star$) for $y = \forall R.C$, there is $P(y) \sqcap \forall R.tag(C) \in \mathcal{U}(R)$;

($\star$) for an Abox individual d_i, there is the tag $P(d_i) \in \mathcal{D}$;

We additionally require $ord(P(d_i)) \succ ord(tag(\mathcal{L}(d_j)))$, for any individual d_i and d_j (even when $i = j$). This forces $P(d_i)$ to get a higher order than $tag(\mathcal{L}(d_j))$ (and higher than subformulae of $tag(\mathcal{L}(d_j))$ but it does not introduce cycles. Please note that $\succ$ is transitive and that extra requirement forces $ord(P(d_i) \oplus tag(\mathcal{L}(d_i))) \succ ord(P(d_i)) \succ ord(tag(\mathcal{L}(d_i)))$. Therefore, for $i \neq j$ we have: (1) $ord(tag(\mathcal{L}(d_i)))$ and $ord(tag(\mathcal{L}(d_j)))$ are incomparable; (2) $ord(P(d_i))$ and $ord(P(d_j))$ are incomparable; (3) $ord(P(d_i)) \succ ord(tag(\mathcal{L}(d_j)))$.

Next, we introduce the second step called *recording*. This step respects acyclic ordering and modal depth and will generate new axioms (concept inclusions).

Definition 4. (Recording) Initially $\mathcal{T}(a) = \emptyset$. For two tuples $\alpha \in (\mathcal{E}(*) \bigcup \mathcal{U}(*) \bigcup \mathcal{D})$ and $\beta \in \mathcal{U}(*)$, where $*$ denotes any role, if the following conditions are met:

(1) $ord(\alpha) \succ ord(\beta)$; and

(2) $\alpha = P(x) \sqcap \forall R_1.tag(C)$ or

$\alpha = P(x) \sqcap \exists R_1.tag(C)$ or

$\alpha = P(x)$ and x is some Abox individual d_i; and

(3) $\beta = P(y) \sqcap \forall R_2.tag(D)$;

then we perform the operation: $\mathcal{T}_a = \mathcal{T}_a \cup \{P(x) \sqsubseteq (\forall R_2^-.\neg P(y)) \sqcup tag(D)\}$.

Figure 3.4 shows two examples. The left part shows a relational structure in which two different nodes x and y are associated with different *modal constraints*. The right part shows a relational structure concerning a self-loop. In both cases, the recorded axioms are $\{P(x) \sqsubseteq (\forall R.\neg P(y)) \sqcup tag(C)\}$.

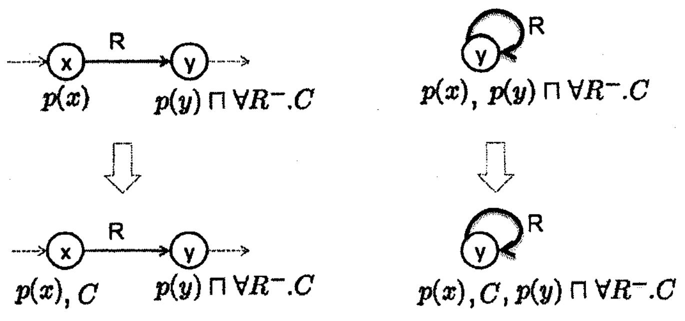

R
x
y
$p(x)$
$p(y) \sqcap \forall R^-.C$
R
y
$p(x)$, $p(y) \sqcap \forall R^-.C$
R
x
y
$p(x), C$
$p(y) \sqcap \forall R^-.C$
R
y
$p(x), C, p(y) \sqcap \forall R^-.C$

Figure 3.4: Backward Constraint Propagation

It should be clear that the axioms newly introduced for $\mathcal{T}_a$ change neither the *acyclic ordering* nor the *modal depth*. The final step is called *polarization*.

Definition 5. (Polarisation) $Pol(x)$ is performed on the tagged Abox $\mathcal{A}_1$ to get $\mathcal{A}_2$, and performed on the augmented Tbox $\mathcal{T}_1 \cup \mathcal{T}_a$ to get $\mathcal{T}_2$:

(1) if x is $(c,d) : R \in \mathcal{A}_1$, then $\mathcal{A}_2 = \mathcal{A}_2 \cup \{(c,d) : R^a,\ (d,c) : R^b\}$;

(2) if x is $(c,d) : R^- \in \mathcal{A}_1$, then $\mathcal{A}_2 = \mathcal{A}_2 \cup \{(d,c) : R^a,\ (c,d) : R^b\}$;

(3) if x is $d_i : P(d_i) \oplus \mathcal{L}(d_i) \in \mathcal{A}_1$, then $\mathcal{A}_2 = \mathcal{A}_2 \cup \{d_i : P(d_i) \sqcap Pol(\mathcal{L}(d_i))\}$;

(4) if x is $C \sqcap D$, then $Pol(x) = Pol(C) \sqcap Pol(D)$;

(5) if x is $C \sqcup D$, then $Pol(x) = Pol(C) \sqcup Pol(D)$;

(6) if x is $\exists R.C$, then $Pol(x) = \exists R^a.Pol(C)$;

(7) if x is $\forall R.C$, then $Pol(x) = \forall R^a.Pol(C)$;

(8) if x is $\exists R^-.C$, then $Pol(x) = \exists R^b.Pol(C)$;

(9) if x is $\forall R^-.C$, then $Pol(x) = \forall R^b.Pol(C)$;

(10) if x is $A \sqsubseteq C$, then $Pol(x) = A \sqsubseteq Pol(C)$;

(11) otherwise, $Pol(x) = x$.

where R^a (R^b) is a fresh role name unique for R (R^-).

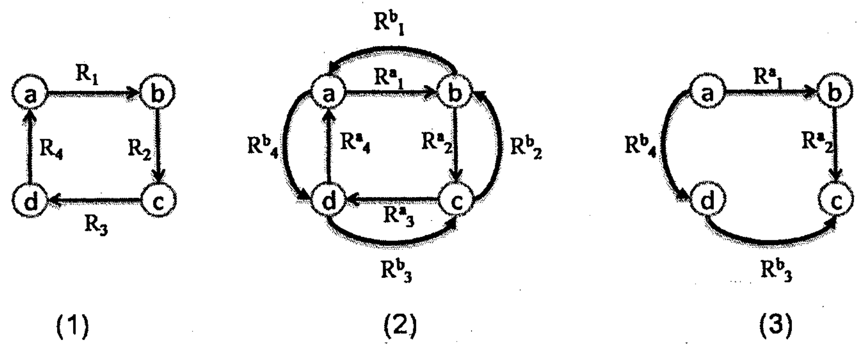


Figure 3.5: An Example Abox

Example 1. Consider the Abox as shown in Figure 3.5 (1).

The Abox *individual assertions* are as follows:

$a : \top$

$b : \forall R_1{}^{-}.C$

$c : B \sqcap \forall R_3.D$

$d : \exists R_4.(A \sqcap \forall R_4{}^{-}.B)$

The Abox *role assertions* are as shown in the figure.

The acyclic Tbox $\mathcal{T}_0$ has one *concept inclusion* $B \sqsubseteq \forall R_3{}^{-}.\exists R_2{}^{-}.A$.

(1) Perform the *tagging operation*. After tagging, the Abox will be

$a : P_a \sqcap \top$

$b : P_b \sqcap Q_3 \sqcap \forall R_1{}^{-}.C$

$c : P_c \sqcap B \sqcap Q_4 \sqcap \forall R_3.D$

$d : P_d \sqcap Q_5 \sqcap \exists R_4.(A \sqcap Q_6 \sqcap \forall R_4{}^{-}.B)$

The tagged Tbox has one *concept inclusion* $B \sqsubseteq Q_1 \sqcap \forall R_3{}^{-}.(Q_2 \sqcap \exists R_2{}^{-}.A)$.

The newly introduced concept names are Q_i ($i \in \{1,2,3,4,5,6\}$) for *modal constraints*, and P_a, P_b, P_c, P_d for Abox individuals.

(2) Perform the *recording operation*, $\mathcal{T}_a$ will have the following *concept inclusions*:

$P_a \sqcup P_b \sqcup P_c \sqcup P_d \sqsubseteq \forall R_1.\neg Q_3 \sqcup C$

$P_a \sqcup P_b \sqcup P_c \sqcup P_d \sqsubseteq \forall R_3{}^{-}.\neg Q_4 \sqcup D$

$P_a \sqcup P_b \sqcup P_c \sqcup P_d \sqsubseteq \forall R_4.\neg Q_6 \sqcup B$

$P_a \sqcup P_b \sqcup P_c \sqcup P_d \sqcup Q_6 \sqcup Q_5 \sqsubseteq \forall R_3.\neg Q_1 \sqcup (Q_2 \sqcap \exists {R_2}^-.A)$

$Q_5 \sqsubseteq \forall R_4.\neg Q_6 \sqcup B$

In the above, $A_1 \sqcup A_2 ... \sqcup A_n \sqsubseteq C$ is a shorthand for $A_i \sqsubseteq C$ $(i \in \{1, 2, 3, ..., n\})$.

(3) Perform the *polarisation operation*, the final Abox *individual assertions* will be

$a : P_a \sqcap \top$

$b : P_b \sqcap Q_3 \sqcap \forall {R_1}^b.C$

$c : P_c \sqcap B \sqcap Q_4 \sqcap \forall {R_3}^a.D$

$d : P_d \sqcap Q_5 \sqcap \exists {R_4}^a.(A \sqcap Q_6 \sqcap \forall {R_4}^b.B)$

The final Abox *role assertions*[4] are depicted in Figure 3.5 (2).

The final Tbox will be

$B \sqsubseteq Q_1 \sqcap \forall {R_3}^b.(Q_2 \sqcap \exists {R_2}^b.A)$

$P_a \sqcup P_b \sqcup P_c \sqcup P_d \sqsubseteq \forall {R_1}^a.\neg Q_3 \sqcup C$

$P_a \sqcup P_b \sqcup P_c \sqcup P_d \sqsubseteq \forall {R_3}^b.\neg Q_4 \sqcup D$

$P_a \sqcup P_b \sqcup P_c \sqcup P_d \sqsubseteq \forall {R_4}^a.\neg Q_6 \sqcup B$

$P_a \sqcup P_b \sqcup P_c \sqcup P_d \sqcup Q_6 \sqcup Q_5 \sqsubseteq \forall {R_3}^a.\neg Q_1 \sqcup (Q_2 \sqcap \exists {R_2}^b.A)$

$Q_5 \sqsubseteq \forall {R_4}^a.\neg Q_6 \sqcup B$

The roles ${R_1}^a$, ${R_3}^a$ and ${R_4}^a$ are newly introduced for R_1, R_3 and R_4; the roles ${R_2}^b$, ${R_3}^b$ and ${R_4}^b$ are newly introduced for ${R_2}^-$, ${R_3}^-$ and ${R_4}^-$.

3.2.3 Proofs

Lemma 6. *$\mathcal{A}_0$ and $\mathcal{T}_0$ have a model iff $\mathcal{A}_1$ and $\mathcal{T}_1$ have a model.*

Proof. It is obvious to see that introducing new conjunct of fresh concept names is harmless to consistency. □

Lemma 7. *If $\mathcal{A}_1$ and $\mathcal{T}_1$ have a model, then $\mathcal{T}_a$ is satisfiable in that model.*

Proof. We prove it by contradiction. Assume $\mathcal{T}_a$ can not be satisfied in the interpretation $\mathcal{I}$ which is a model for both $\mathcal{A}_1$ and $\mathcal{T}_1$. Then, there must exist at least one axiom of $\mathcal{T}_a$ being violated at some domain element of the interpretation. Without loss of generality, suppose the violated axiom is

[4]Actually for Abox consistency problems, it is sufficient for the final Abox to have *role assertions* as depicted in 3.5 (3) (by taking off self-loops and cycles). However, for conjunctive query problems, according to [Lut07], it is necessary to form the *role assertions* as Figure 3.5 (2).

$P(x) \sqsubseteq \forall R.\neg P(y) \sqcup tag(C)$, and suppose the domain element of $\mathcal{I}$ causing the violation (w.r.t. that axiom) is a. Consider two element $a, b \in \Delta^{\mathcal{I}}$ such that $a \in P(x)^{\mathcal{I}}$ and $b \in P(y)^{\mathcal{I}}$ and $(a, b) \in R^{\mathcal{I}}$.

(case 1) $a \neq b$: In this case, we are considering two different individuals. By recalling the tagging operation given in Definition 3, we can see that $P(y)$ is a tag introduced for the universal restriction $\forall R^{-}.tag(C)$. Given $b \in P(y)^{\mathcal{I}}$, it follows that $b \in (\forall R^{-}.tag(C))^{\mathcal{I}}$. Given $(a, b) \in R^{\mathcal{I}}$, it is true that $(b, a) \in (R^{-})^{\mathcal{I}}$. Therefore $a \in tag(C)^{\mathcal{I}}$. This means that the given axiom $P(x) \sqsubseteq \forall R.\neg P(y) \sqcup tag(C)$ actually is satisfied at the domain element a, which contradicts the assumption.

(case 2) $a = b$: In this case, we are discussing one single individual a. The conditions to be considered are: $a \in \Delta^{\mathcal{I}}$, $a \in P(x)^{\mathcal{I}}$, $a \in P(y)^{\mathcal{I}}$, $(a, a) \in R^{\mathcal{I}}$ and $P(x) \neq P(y)$. Similar to case 1, it is able to show that the given axiom $P(x) \sqsubseteq \forall R.\neg P(y) \sqcup tag(C)$ is satisfied at the domain element a.

Readers are referred to Figure 3.4 for an illustration of the above two cases. Since each axiom of $\mathcal{T}_a$ is satisfied at all domain elements of $\mathcal{I}$, the lemma is proved. □

Lemma 8. *$\mathcal{A}_1$ is satisfiable w.r.t. $\mathcal{T}_1 \cup \mathcal{T}_a$ iff $\mathcal{A}_1$ is satisfiable w.r.t. $\mathcal{T}_1$.*

Proof. ($\Rightarrow$) It is trivial.

($\Leftarrow$) Let $\mathcal{M}_2$ be a model for $\mathcal{A}_1$ and $\mathcal{T}_1$. According to the lemma above, $\mathcal{T}_a$ is always satisfied in the model for both $\mathcal{T}_1$ and $\mathcal{A}_1$. It follows that $\mathcal{M}_2$ is a model for $\mathcal{A}_1$ and $\mathcal{T}_1 \cup \mathcal{T}_a$. □

Lemma 9. *$\mathcal{A}_1$ is satisfiable w.r.t. $\mathcal{T}_1 \cup \mathcal{T}_a$ iff $\mathcal{A}_2$ is satisfiable w.r.t. $\mathcal{T}_2$.*

Proof. Note $\mathcal{A}_2 = Pol(\mathcal{A}_1)$ and $\mathcal{T}_2 = Pol(\mathcal{T}_1 \cup \mathcal{T}_a)$.

(If Direction) Let $\mathcal{M}_2 = (\Delta^{\mathcal{I}_2}, \cdot^{\mathcal{I}_2})$ be a model (possibly non-tree) for $\mathcal{A}_2$ and $\mathcal{T}_2$. For $m', n' \in \Delta^{\mathcal{I}_2}$, consider a mapping to $m, n \in \Delta^{\mathcal{I}_1}$ such that

(1) if $(m', n') \in (R^a)^{\mathcal{I}_2}$, then $(m, n) \in R^{\mathcal{I}_1}$;

(2) if $(m', n') \in (S^b)^{\mathcal{I}_2}$, then $(m, n) \in (S^{-})^{\mathcal{I}_1}$;

(3) if $m', n' \in Pol(C)^{\mathcal{I}_2}$, then $m, n \in C^{\mathcal{I}_1}$.

(Only If Direction) Let $\mathcal{M}_1 = (\Delta^{\mathcal{I}_1}, \cdot^{\mathcal{I}_1})$ be a model (possibly non-tree) for $\mathcal{A}_1$ and $\mathcal{T}_1$. For $m, n \in \Delta^{\mathcal{I}_1}$, consider a mapping that maps them to $m', n' \in \Delta^{\mathcal{I}_2}$

(1) if $(m, n) \in R^{\mathcal{I}_1}$, then $(m', n') \in (R^a)^{\mathcal{I}_2}$ and $(n', m') \in (R^b)^{\mathcal{I}_2}$;

(2) if $(m, n) \in (S^{-})^{\mathcal{I}_1}$, then $(m', n') \in (S^b)^{\mathcal{I}_2}$ and $(n', m') \in (S^a)^{\mathcal{I}_2}$;

(3) if $m, n \in C^{\mathcal{I}_1}$ then $m', n' \in Pol(C)^{\mathcal{I}_2}$.

R, S^- are roles in $\mathcal{ALCI}$; R^a, R^b, S^a, S^b are roles in $\mathcal{ALC}$.

In both directions, for each element of the target interpretation, all constraints are satisfied both locally, and w.r.t. its neighbor elements provided the given $(\Delta^{\mathcal{I}_k}, .^{\mathcal{I}_k})$ (where $k = 1, 2$) is a model. This concludes that the polarisation operation preserves equisatisfiability (for a tagged and recorded problem). $\square$

Theorem 10. *The above tagging, recording and polarization converts an $\mathcal{ALCI}$ Abox to an $\mathcal{ALC}$ Abox and preserves consistency.*

3.3 Discussion

By eliminating inverse relations, we obtain an $\mathcal{ALC}$ Abox from an $\mathcal{ALCI}$ Abox. In [DHW07], we have shown a proof that the consistency problem of an $\mathcal{ALCI}$ Abox w.r.t. an acyclic Tbox is PSPACE decidable by using this elimination technique. By available techniques [Lut07, DHW07, BML+05], it is simple to convert an $\mathcal{ALC}$ Abox to an acyclic $\mathcal{ALCO}$ Tbox. So, it is not surprising for one to see that any $\mathcal{ALCI}$ Abox (with respect to an acyclic Tbox) is representable in an (acyclic) $\mathcal{ALCO}$ Tbox. According [BML+05], the *consistency problem* of $\mathcal{ALCOQ}$ with an acyclic Tbox is PSPACE decidable by tableaux algorithms. Therefore, it is easy to see that the *consistency problem* of an $\mathcal{ALCI}$ Abox with respect to an acyclic Tbox is still PSPACE decidable.

Experiments have been carried out to test practicality of the technique described in this chapter (currently only for concept satisfiability test in $\mathcal{ALCI}$). A concise report can be found in Appendix B. Though the sizes of the new ontologies were slightly increased (less than a constant factor of 5 times the sizes of the original ontologies), RACER [HM01b] solved all converted ontologies within an acceptable time (see Appendix B), demonstrating a "robustness" behavior for realistic problems due to switching on *global tableaux caching.*

In the next chapter, we will use a variant of the same technique to take away the effect of "backward propagation" of (universal and functional) constraints for the DL $\mathcal{ALCFI}$ instead.

Chapter 4

A Tableaux Procedure for $\mathcal{ALCFI}$

4.1 Introduction

4.1.1 A Brief Review

For Description Logics (DLs) having no inverse roles, the *global (sub-)tableaux caching* functionality is known to be an effective *runtime optimization* and was built into some highly-optimized tableau-based DL systems, e.g., FaCT [Hor98], DLP [Pet98], and RACER [HM01b]. Roughly speaking, tableaux caching is a looking-up mechanism consisting of consistency caching and inconsistency caching. The use of cached results can effectively avoid repeated searches by a fast retrieval of previously stored information about a label and its satisfiability status. However, for DLs with inverse roles, the soundness of tableaux caching is problematic [BCM+03] because conventionally tableaux caching mechanisms do not take backward-propagation of constraints into consideration[1]. It is evident that the above mentioned DL systems and their successors when reasoning for DLs with inverse roles do not use the full-fledged tableaux caching functionality previously implemented for DLs without inverse roles.

There is another approach to DLs with inverse roles [TH06, SPG+07]. By contrast to the *global tableaux caching*, this approach uses some optimizations (e.g., *dynamic blocking* and *pseudo model merging* [THPS07]) that adapt and scale better to DLs with inverse roles. For example, FaCT++ [TH06] implemented a "ToDo List" architecture and is able to schedule propagation of constraints

[1]Though it is possible to do so, the implementation would be complicated and the run-time penalty might be high. $\exists R^-.(\forall R.\bot)$ is not consistent though its sub-concept $\forall R.\bot$ is; an overlooking of backward propagation of constraints ($\bot$ here) leads to unsoundness. A second source of unsoundness as was pointed out in [HM00a] is an inadvertent use of caching for Abox *individual*. The two sources of unsoundness might occur in $\mathcal{ALCI}$ Abox problems.

[THPS07]. This approach is influential and is a prominent feature of some new DL systems. It can be understood that managing constraint propagation in the presence of inverse roles is an important issue for tableaux systems.

A sound and restricted global *tableaux caching* was reported in [DH05]; the idea is to compute labels potentially unsafe for reuse and then exclude them from caching. An experimental system has been implemented as a hybrid of an $\mathcal{ALC}$-style (restricted) *tableaux caching* functionality with a standard $\mathcal{ALCI}$-style tableaux system. But the preliminary results were mixed [DH06]. Therefore, this chapter continues on developing a new way that can fully utilize the power of caching (available for DLs without inverse roles) rather than being restrictive as in [DH05] (see also Chapter 2).

4.1.2 Why Inverse Relation Is The Problem

Here, let us take an inside look at some common issues from tableau-based systems for DLs, and see the difference between an $\mathcal{ALC}$ system and an $\mathcal{ALCI}$ system. Recall that to restore the tableaux structure (a.k.a. completion graph) to a previous state during a backtracking phase, it is necessary to make a copy of (portions of) the tableaux structure for each sequence of deterministic operations. In $\mathcal{ALC}$ the scope of the undoing area (i.e. the portion of changes to be reverted) is bounded below a subtree rooted at the source of a conflict; in $\mathcal{ALCI}$ the scope of undoing is hard to calculate even when sources of conflicts can be optimally located. Moreover, it is more difficult to efficiently propagate inconsistency results in $\mathcal{ALCI}$, which also means that the inconsistency caching functionality is weaker in $\mathcal{ALCI}$. The left graph below shows an inconsistent structure where no nontrivial inconsistency is inferred.

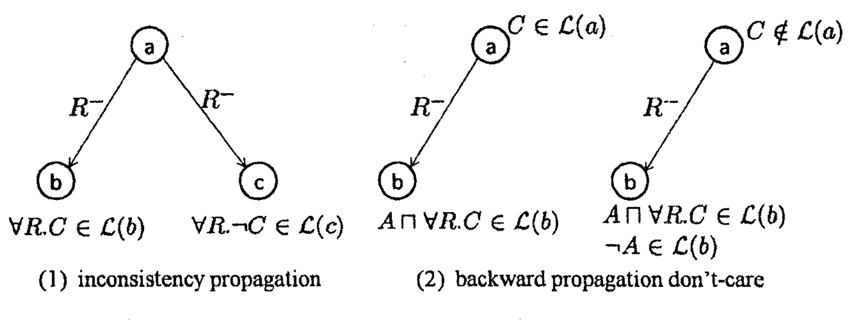


Figure 4.1: Inconsistency Propagation and Backward Constraint Propagation

In an $\mathcal{ALC}$ tableaux structure, a node does not propagate constraints to its predecessor, and its sibling nodes can be handled independently; this is not true in $\mathcal{ALCI}$. It is the issue of *backward propagation* of constraints that makes an $\mathcal{ALCI}$ tableaux system behave differently from $\mathcal{ALC}$'s. The two rightmost pictures above show that *backward propagation* of constraints can be circumvented by way of enforcing some *general concept inclusions*[2] (a variant of the technique presented in Chapter 3).

$\mathcal{ALCFI}$ has been well studied in the literature [CGR98, HS99, Tob00, Lut04]. It extends the basic DL $\mathcal{ALC}$ with *inverse roles* and *functional restrictions*. A functional restriction is of the form $(\leq_1 R)$ which restricts the number of R-neighbors to be at most one, and can be viewed as a partial function over the interpretation domain [Lut99]. When considering *general concept inclusions*, this logic still has the *tree model* property but no *finite model* property. For example, it is only possible with an infinite model to satisfy the concept $\neg A$ and a *general concept inclusion* $\top \sqsubseteq (\leq_1 R^-) \sqcap \exists R.A$. To avoid incorrect infinite models, the dynamic *pair-wise blocking technique* [HS99] is used to guarantee soundness.

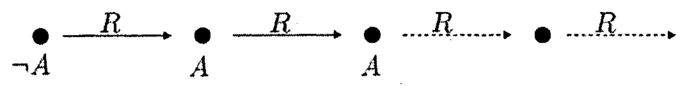


Figure 4.2: A Model for $\neg A$ w.r.t. $\mathcal{T} = \{\top \sqsubseteq (\leq_1 R^-), \top \sqsubseteq \exists R.A\}$

It is appropriate in this context to discuss *blocking techniques*, which are also known as *cycle detection and termination* mechanisms. Generally speaking, a node x is blocked if none of its *ancestors* are blocked, and x has a *witness* x' (one of x's ancestor nodes) such that the labels of x and x' meet certain prior conditions. For example in the *equality blocking technique*, if $\mathcal{L}(x) = \mathcal{L}(x')$ (note that $\mathcal{L}(.)$ is a set of concept expressions denoting a label for a tableau node), then it is said that x is blocked by x'. Provably tableau-based decision procedures can safely ignore any blocked node without compromising soundness. It was shown in [HS99] that, even for $\mathcal{ALCI}$, *static blocking* is inadequate and unsound. Therefore it is necessary to use *dynamic blocking* for DLs with inverse

[2]When the predecessor node does not get backward propagated constraints, a primitive clash like $\{A, \neg A\}$ will be triggered at the successor node (when the recorded axioms are enforced at the predecessor node). This is called "backward propagation don't-care", see Figure 4.1 and Lemma 12.

roles. If the original conditions have been changed (due to backward constraint propagation), it is necessary to break the established blockings to reflect the changes[3]. Also presented in [HS99] is a dynamic *pair-wise blocking* technique (and its optimized variants) designed for searching infinite models. The dynamic *pair-wise blocking* is also known as "double blocking" because the blocking condition is specified in patterns of depth 2 [BHLW03, HM04, Hla04], i.e., the matching is specified by the relation of two pairs of predecessor-successor nodes (for an illustration and explanation please see "double blocking" in Section 6.4.3 of Chapter 6).

It is known that the global caching of both consistency and inconsistency label sets is sufficient to get an ExpTime tableau-based decision procedure [DM00], however, no example has been set up for DLs with inverse roles so far[4]. We will show a worst-case exponential time decision procedure for $\mathcal{ALCFI}$.

Let us connect two related notions, namely, the *blocking technique* and the *global tableaux caching* (in the context of DLs having no inverse roles). This connection is plausible because the primary purpose of both is to ensure the termination of tableau-based decision procedures. These two notions, usually discussed in a tree or forest structure[5], differ in their applicable scope: *blocking* is applicable only between a node and (one or many of) its ancestor nodes; *caching* is globally applicable as long as its application does not destroy soundness. The second difference is that *caching* is applicable to both satisfiable and unsatisfiable situations; but *blocking* is only applicable to satisfiable situations, which means *blocking* is not able to prune away "unpromising parts" of the search space.

4.1.3 A New Approach

In this chapter, the backward propagation of constraints (on a tableau structure) is characterized as *general concept inclusions* in the language of the logic itself, similar to the *encoding method* mentioned before. During the run of the decision procedure, a cyclic tableaux structure might be built and the *unraveling technique* [HS99] is used to construct infinite models. It should be clear that the *tree model* property for the description logic $\mathcal{ALCFI}$ is used. Please note that a *restart strategy* is used in the decision procedure for an illustration purpose. In other words, the correctness of the

[3]A broken blocking might later be re-established again [HS99].

[4]It was recently shown in [GN07] and [DH07a] that ExpTime tableau-based decision procedures were proposed for DLs with inverse roles.

[5]And also including a finite-sized non-tree structure in DLs containing $\mathcal{ALCOIQ}$ as a fragment.

decision procedure does not depend on a use of this strategy. The decision procedure is required to be capable of propagating *inconsistency*: at least one new inconsistency need to be found when a model cannot be constructed. The tableaux caching technique proposed in this chapter is inspired by and uses similar data structures (`Nogood` and `Witness`) as in [DM00].

Before starting the major part of this chapter, let us take a look at a simple example. Suppose $\{\forall R.C\}$ is cached as a witness. However, this *cached witness* cannot be reused to test the satisfiability of $\neg C \sqcap \exists R^{-}.(\forall R.C)$. An improper use of the cached witness in this scenario (and others alike) would lead to an unsound conclusion that $\neg C \sqcap \exists R^{-}.(\forall R.C)$ is satisfiable, which is actually not true. The culprit is the unanticipated propagations of constraints from the cached witness, i.e., $\{\forall R.C\}$ propagates along the R^{-} edge backward a constraint C. This phenomenon is the very reason behind the unsoundness of global tableaux caching. By a preprocessing step, the source problem is converted to a target problem with a concept $\neg C \sqcap \exists R^{-}.(A \sqcap \forall R.C)$ and a *general concept inclusion* $\top \sqsubseteq C \sqcup \forall R^{-}.\neg A$. Working on the converted problem prevents inadvertent uses of cached witnesses. The soundness of the global (sub)tableaux caching here relies on those explicit *general concept inclusions* (characterizing backward propagation of constraints on a tableau structure).

We will present a satisfiability-preserving conversion (in Section 4.3) that transforms general problems in $\mathcal{ALCFI}$ to target problems in the same logic, and continue to describe a tableau-based decision procedure (composed of two cooperative sub-procedures and a restart strategy) that decides the satisfiability of the target problems in deterministic exponential time. Thus, it will be shown how a combination of these two ingredients enables a sound use of the global (sub)tableaux caching technique as freely as in the case of $\mathcal{ALC}$ [DM00] for a logic with inverse roles.

4.2 $\mathcal{ALCFI}$ Syntax and Semantics

Conventional notations are used: A for *atomic concept*, C and D for arbitrary concepts, and R for a role. The concept formulae in $\mathcal{ALCFI}$ are formed as follows:

$$C, D := \top | A | \neg C | C \sqcap D | C \sqcup D | \exists R.C | \forall R.C | \leq_1 R | \geq_2 R$$

An interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, .^{\mathcal{I}} \rangle$ consists of a non-empty domain $\Delta^{\mathcal{I}}$, and an interpretation function $.^{\mathcal{I}}$ which maps every concept to a subset of $\Delta^{\mathcal{I}}$ and every role name to a subset of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ such that:

$$(\top)^{\mathcal{I}} = \Delta^{\mathcal{I}} \qquad (C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}}$$

$$(C \sqcup D)^{\mathcal{I}} := C^{\mathcal{I}} \cup D^{\mathcal{I}} \qquad (\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$$

$$(\exists R.C)^{\mathcal{I}} := \{a \in \Delta^{\mathcal{I}} | \text{ There is } b \in \Delta^{\mathcal{I}} \text{ with } (a, b) \in R^{\mathcal{I}} \text{ and } b \in C^{\mathcal{I}}\}$$

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

$$(\leq_1 R)^{\mathcal{I}} := \{a \in \Delta^{\mathcal{I}} | \forall b, c \in \Delta^{\mathcal{I}}, (a, b) \in R^{\mathcal{I}} \text{ and } (a, c) \in R^{\mathcal{I}} \text{ implies } b = c\}$$

$$(\geq_2 R)^{\mathcal{I}} := \{a \in \Delta^{\mathcal{I}} | \exists b, c \in \Delta^{\mathcal{I}} \text{ s.t. } (a, b) \in R^{\mathcal{I}} \text{ and } (a, c) \in R^{\mathcal{I}} \text{ and } b \neq c\}$$

A *role* is a role name or the inverse of a role, which is interpreted as a binary relation on a subset of elements of the interpretation domain $\Delta^{\mathcal{I}}$. The related *inverse role* is interpreted as the inverse of that binary relation. For example, let $x, y \in \Delta^{\mathcal{I}}$, $(x, y) \in R^{\mathcal{I}}$ iff $(y, x) \in (R^-)^{\mathcal{I}}$, where R^- and R are two roles in the inverse relationship. Simply, each binary relation has a unique inverse relation; and the inverse of an inverse relation is the original relation itself. In this chapter, it is assumed that each role has a unique inverse role. For a role R, for example, R^- is considered as the only *inverse role*[6] of R, and vice versa.

A Tbox $\mathcal{T}$ of general axioms is a set of axioms of the form $\top \sqsubseteq C$[7]. An $\mathcal{ALCFI}$ concept C is satisfiable w.r.t. a Tbox $\mathcal{T}$ iff C and $\mathcal{T}$ have a model (i.e. non-empty interpretation) in common.

[6]It takes a linear cost to identify equivalent roles that are implied by inverse relationship declarations in a namespace (of role names).

[7]Equality axioms of the form $C \equiv D$ are converted to two inclusion axioms like $C \sqsubseteq D$ and $D \sqsubseteq C$. An inclusion axiom of the form $C \sqsubseteq D$ can be converted to $\top \sqsubseteq \neg C \sqcup D$.

4.3 A Preprocessing Step

A variant of the technique given in Chapter 3 will be presented to take *functional restrictions* into account. For convenience, concept formulae/expressions are required to be in the Negation Normal Form (NNF) (by pushing negation signs inward to concept names through the use of De-Morgan's law and dualities, see also its definition in Chapter 2). For an input concept formula E_0 and a Tbox $\mathcal{K}_0$ consisting of general axioms of the form $\top \sqsubseteq C$, where E_0 and C are in NNF, a preprocessing step is introduced as follows.

Definition 1. The operation $tag(.)$ on an expression (incl. axiom) x is:

(1) if x is $C \sqcap D$, then $tag(x) = tag(C) \sqcap tag(D)$;

(2) if x is $C \sqcup D$, then $tag(x) = tag(C) \sqcup tag(D)$;

(3) if x is $\exists R.C$, then $tag(x) = P(x) \sqcap \exists R.(tag(C))$;

(4) if x is $\forall R.C$, then $tag(x) = Q(x) \sqcap \forall R.(tag(C))$;

(5) if x is $\top \sqsubseteq C$, then $tag(x) = \top \sqsubseteq tag(C)$;

(6) otherwise $tag(x) = x$.

where $P(x)$, $Q(x)$ are fresh names unique for each x; C, D are subformulae.

E_0/E_1 denote the formulae before/after tagging; $\mathcal{K}_0/\mathcal{K}_1$ denote the Tboxes before/after tagging. By using the tagging operation, it is possible to collect tagged *universal constraints* of the form $Q(x) \sqcap \forall R.(tag(C))$ into a set $\mathcal{U}(R)$, and tagged *existential constraints* of the form $P(x) \sqcap \exists R.(tag(C))$ into a set $\mathcal{E}(R)$, where R is a role, and $\mathcal{U}$ and $\mathcal{E}$ are sets indexed by different roles. Let x denote any (sub)formula of E_0 and $\mathcal{K}_0$, we have:

- for $x = \exists R.C$, there is $P(x) \sqcap \exists R.tag(C) \in \mathcal{E}(R)$;

- for $y = \forall R.D$, there is $Q(x) \sqcap \forall R.tag(D) \in \mathcal{U}(R)$;

Definition 2. Initialize $\mathcal{K}_a = \emptyset$. For $\alpha \in \mathcal{E}(*)$ and $\beta \in \mathcal{U}(*)$, where $*$ denotes any role, perform the following operations:

(1) if α is $P(x) \sqcap \exists R.tag(C) \in \mathcal{E}(R)$, then

$\mathcal{K}_a := \mathcal{K}_a \cup \{\top \sqsubseteq (\forall R^{-}.(\neg P(x) \sqcup \geq_2 R)) \sqcup tag(C)\}$;

(2) if β is $Q(x) \sqcap \forall R.tag(D) \in \mathcal{U}(R)$, then

$\sqcap$-rule:	if	1. $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, and
		2. not $\{C_1, C_2\} \subseteq \mathcal{L}(x)$
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if	1. $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and
		2. $\{C_1, C_2\} \cap \mathcal{L}(x) = \emptyset$
	then	$\mathcal{L}(x) := \mathcal{L}(x) \cup \{C_i\}$ for some $C_i \in \{C_1, C_2\}$
$\exists$-rule:	if	1. $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and
		2. $(\leq_1 R) \notin \mathcal{L}(x)$ and x has no R-neighbor y with $C \in \mathcal{L}(y)$
	then	create a successor node y with $\mathcal{L}(\langle x, y\rangle) := \{R\}$ and $\mathcal{L}(y) := \{C\}$;
	if	1. $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and
		2. $(\leq_1 R) \in \mathcal{L}(x)$ and x has no R^--predecessor, and
		3. x has no R-successor y
	then	create a successor node y with $\mathcal{L}(\langle x, y\rangle) := \{R\}$ and $\mathcal{L}(y) := \{C\}$;
	if	1. $\exists R.C \in \mathcal{L}(x)$, x is not blocked, and
		2. $(\leq_1 R) \in \mathcal{L}(x)$ and x has no R^--predecessor, and
		3. x has an R-successor y
	then	$\mathcal{L}(y) := \mathcal{L}(y) \cup \{C\}$
$\geq$-rule:	if	1. $(\geq_2 R) \in \mathcal{L}(x)$, x is not blocked, and
		2. x has no R-neighbor y
	then	create a successor node y with $\mathcal{L}(\langle x, y\rangle) := \{R\}$ and $\mathcal{L}(y) := \emptyset$
$\forall$-rule:	if	1. $\forall R.C \in \mathcal{L}(x)$, x is not blocked, and
		2. there is an R-successor y of x with $C \notin \mathcal{L}(y)$
	then	$\mathcal{L}(y) := \mathcal{L}(y) \cup \{C\}$

Table 4.1: The tableaux expansion rules for $\mathcal{ALCFI}$.

$$\mathcal{K}_a := \mathcal{K}_a \cup \{\top \sqsubseteq (\forall R^-.\neg Q(x)) \sqcup tag(D)\}.$$

Similar to the $\mathcal{T}_a$ (used in the recording step) in Chapter 3, $\mathcal{K}_a$ consists of axioms that will help achieve an effect of "backward propagation don't-care" in the tableau-based decision procedure to be introduced shortly. A small example (of the preprocessing) was given in the middle of Section 4.1.3, and we will present a correctness proof in Section 4.6 for this conversion. Hereafter, $\mathcal{K}_2$ is used to denote $\mathcal{K}_a \cup \mathcal{K}_1$, and E_2 to denote E_1. In the following, we will introduce a tableaux procedure working on E_2 and $\mathcal{K}_2$ (both of which are obviously expressed in the description logic $\mathcal{ALCFI}$).

4.4 $\mathcal{ALCFI}$ Tableau Rules

We consider the converted problem, i.e., the satisfiability problem of E_2 w.r.t. $\mathcal{K}_2$. Without loss of generality, the Tbox of a set of general axioms is simplified and expressed as one general axiom $\top \sqsubseteq G_2$. To be precise, $G_2 = \sqcap r_i$ for all $\top \sqsubseteq r_i \in \mathcal{K}_2$. Table 4.1 is a set of tableaux expansion rules for (the converted problem in) $\mathcal{ALCFI}$.

We use the notion of a *completion tree* (see Appendix A.1). Given a completion tree, a node y

$\bot$-0:		$\{\neg\top\} \in \bot$-set.
$\bot$-1:		$\{C, \neg C\} \in \bot$-set.
$\bot$-2:		$\{(\leq_1 R), (\geq_2 R)\} \in \bot$-set.
$\bot$-3:	if	$\alpha \in \bot$-set and $\alpha \subseteq \beta$, then $\beta \in \bot$-set.
$\bot$-4:	if	$\alpha \cup \{C\} \in \bot$-set and $\alpha \cup \{D\} \in \bot$-set, then $\alpha \cup \{C \sqcup D\} \in \bot$-set.
$\bot$-5:	if	$\alpha \in \bot$-set and $\beta = \{\exists R.E_0, \forall R.E_1, ..., \forall R.E_n\}$(where R is any role) and $\alpha \subseteq \{E_0, E_1, E_2, ..., E_n\}$, then $\beta \in \bot$-set.
$\bot$-6:	if	$\alpha \in \bot$-set and $\beta = \{(\geq_2 R), \forall R.E_1, ..., \forall R.E_n\}$(where R is any role) and $\alpha \subseteq \{E_1, E_2, ..., E_n\}$, then $\beta \in \bot$-set.

Table 4.2: The inconsistency propagation rule for $\mathcal{ALCFI}$.

is called an *R-neighbor* of a node x if y is an R-successor of x, or if x is an R^--successor of y. Predecessor nodes are deliberately ignored in some of the rules above. The correctness relies on the *back-propagation-don't-care property* (a proof will be given in Section 4.7.2).

To obtain a potentially infinite model, it is necessary to apply the *unraveling technique* on the completion tree using the notion of a path [HS99]. Instead of going to a directly blocked node, the path goes to the blocking node. Thus, if the blocking occurred, an infinite model can be obtained through the *unraveling technique* (in cases when an ancestor node and a descendant node are involved).

To generalize the notion of the *clash triggers* [BCM+03] in $\mathcal{ALCFI}$, the notion of a $\bot$-set (a.k.a. Nogood) presented in [DM00] is adapted here. To avoid generating irrelevant $\bot$-sets (which would be the case if the $\bot$-rules were considered as a parallel calculus to derive inconsistent concept sets bottom-up rather than top-down), inconsistency inferences based on the $\bot$-rules are carried out on demand in the decision procedure. This "on demand" inconsistency propagation[8] also follows from [DM00]. Table 4.2 shows the inconsistency propagation rules for $\bot$-sets.

[8]The author is aware of a bottom-up approach (see reference in [DM00]) that performs inconsistency inferences "aggressively". However, that approach is irrelevant to the decision procedure here.

4.5 $\mathcal{ALCFI}$ Decision Procedure

To get an exponential-time sound and complete decision procedure, certain search strategies must be integrated with the expansion and propagation rules given above. In the following we outline two procedures that cooperate and decide the concept satisfiability problem w.r.t. a set of general axioms in $\mathcal{ALCFI}$. The procedure $TEST(E_2, G_2)$ answers the satisfiability testing of E_2 w.r.t. $\mathcal{K}_2$ (Recall that G_2 is RHS of the single GCI of $\mathcal{K}_2$, see the discussion at the beginning of Section 4.4).

4.5.1 The Procedure TEST(.,.)

```
PROCEDURE TEST(C, G)
[01]  Nogood := ∅;
[02]  WHILE (true)
[03]     Witness := ∅;
[04]     len := sizeof(Nogood);
[05]     allocate a tableau node x_0 and let L(x_0) := {C};
[06]     SAT(x_0, G, null, null);
[07]     IF (C ∈ Nogood) RETURN unsatisfiable;
[08]     IF (len == sizeof(Nogood)) RETURN satisfiable;
[09]  ENDWHILE
END-PROCEDURE {TEST}
```

when restarting, line 07
will be run next

The procedure $TEST(\cdot,\cdot)$ takes a concept C and the RHS of a general axiom $\top \sqsubseteq G$ as input. It invokes the sub-procedure $SAT(\cdot,\cdot,\cdot,\cdot)$ in the WHILE-loop. Nogood and Witness are two abstract global data structures considered as sets (of set of formulae). Nogood is initialized only once and memorizes everything augmented by the sub-procedure $SAT(\cdot,\cdot,\cdot,\cdot)$. Witness, used for the *static blocking* is, however, emptied in each loop before invoking $SAT(\cdot,\cdot,\cdot,\cdot)$.

The utility *sizeof*(.) gets the number of elements of a set. The variable *len* is used for keeping the number of current elements in Nogood. There is a special tableau node named x_0 and $\mathcal{L}(x_0)$ is a label (a set of concepts that is subject to *satisfiability test*).

There are two cases to exit the infinite WHILE-loop. The first is when the input concept C is found in the global data structure Nogood, then $TEST(\cdot,\cdot)$ decides the problem as unsatisfiable. The second is when Nogood is not augmented by the sub-procedure $SAT(\cdot,\cdot,\cdot,\cdot)$, then $TEST(\cdot,\cdot)$ decides the problem as satisfiable.

4.5.2 The Procedure SAT(.,.,.,.)

```
PROCEDURE $SAT(x, G, parent, edge2parent)$
[10]  IF ($\mathcal{L}(x) \in$ Nogood) RETURN $false$;
[11]  IF ($\mathcal{L}(x) \in$ Witness) RETURN $true$;
[12]  $\mathcal{BS}$ := all the propositional branches of $\mathcal{L}(x) \cup \{G\}$;
[13]  selected := $false$;
[14]  FOR EACH $\mathcal{B} \in \mathcal{BS}$
[15]      IF ($\mathcal{B} \in$ Nogood) CONTINUE;
[16]      IF ($hasclash(\mathcal{B})$) THEN
[17]          Nogood := Nogood $\cup\{\mathcal{B}\}$;
[18]          ABORT;                                  restart
[19]      ENDIF
[20]      IF ($\mathcal{B} \in$ Witness) THEN                   initial label used for
[21]          Witness := Witness $\cup\{\mathcal{L}(x)\}$;           witness (global caching)
[22]          RETURN $true$;
[23]      ENDIF
[24]      selected := $true$;
[25]      BREAK;
[26]  ENDFOR
[27]  IF ($\neg$ selected ) THEN
[28]      Nogood := Nogood $\cup\{\mathcal{L}(x)\}$;               initial label used for
[29]      ABORT;                          restart     inconsistency inference
[30]  ENDIF
[31]  $Successors(\mathcal{B}, Succ(x,.), edge2parent)$;
[32]  $AllSuccessors(\mathcal{B}, Succ(x,.))$;
[33]  Witness := Witness $\cup$ $\{\mathcal{B}\}$;
[34]  Witness := Witness $\cup$ $\{\mathcal{L}(x)\}$;
[35]  FOR EACH successor $s \in Succ(x, R)$    ($comment$: $R$ is any role)
[36]      ret := $SAT(s, G, x, R^-)$;
[37]      IF ($\neg$ ret) THEN
[38]          Nogood := Nogood $\cup$ $\{\mathcal{B}\}$;
[39]          ABORT;                                    restart
[40]      ENDIF
[41]  ENDFOR
[42]  RETURN $true$;
END-PROCEDURE {SAT}
```

The instruction Abort facilitates the *restart strategy.* An execution of Line-18 or Line-29 or Line-39 will transfer the program control to Line-07 in procedure $TEST(\cdot,\cdot)$. Note, a new element is surely inserted into Nogood just before the execution of Abort.

When Line-15 is executed, the program control goes to Line-14 (because the use of Continue).

Line-35 to Line-41 performs a depth-first exploration of the tableau tree.

Line-17, Line-28, and Line-38 implement the $\perp$-rules. The utility *hasclash*(.) implements the detection of primitive clashes.

Line-11, and the code snippet from Line-20 to Line-23 implement the specific *static blocking* and works on a new notion of *propositional branches.* A propositional branch abstracts the exhaustive application of the $\sqcap$-rule and the $\sqcup$-rule (at the tableaux node x) over the topmost $\sqcap$ and $\sqcup$ operators in concept expressions of $\mathcal{L}(x) \cup \{G\}$.

$Succ(x,.)$ is a data structure holding a set of successors of node x, indexed on different roles.

4.5.3 The Sub-Procedure `Successors(.,.,.)`

```
PROCEDURE Successors($\mathcal{B}$, $Succ(x,.)$, $edge2parent$)
[43]  FOR EACH existential-restriction $e \in \mathcal{B}$   (comment: $e$ is of form $\exists R.C$)
[44]      IF (($\leq_1 R$) $\in \mathcal{B}$) and ($edge2parent \neq R$) THEN
[45]          IF ($Succ(x,R) == \emptyset$) THEN
[46]              allocate a new tableau node $s$;
[47]              $Succ(x,R) := Succ(x,R) \cup \{s\}$;
[48]              $\mathcal{L}(s) := \{C\}$;
[49]              $\mathcal{L}(\langle x,s \rangle) := \{R\}$;
[50]          ELSE
[51]              $s :=$ the single element in $Succ(x,R)$;
[52]              $\mathcal{L}(s) := \mathcal{L}(s) \cup \{C\}$;
[53]          ENDIF
[54]      ELSE IF (($\leq_1 R$) $\notin \mathcal{B}$) THEN
[55]          IF ($x$ has no $R$-neighbor $y$ s.t. $C \in \mathcal{L}(y)$) THEN
[56]              allocate a new tableau node $s$;
[57]              $Succ(x,R) := Succ(x,R) \cup \{s\}$;
[58]              $\mathcal{L}(s) := \{C\}$;
[59]              $\mathcal{L}(\langle x,s \rangle) := \{R\}$;
[60]          ENDIF
[61]      ENDIF
[62]  ENDFOR
END-PROCEDURE {Successors}
```

Successors($\mathcal{B}$, $Succ(x,.)$, *edge2parent*) implements the $\exists$-rule and the $\leq$-rule. $Succ(x,.)$ is an output parameter. $\mathcal{B}$ and *edge2parent* are input parameters.

4.5.4 The Sub-Procedure `AllSuccessors(.,.)`

```
PROCEDURE AllSuccessors($\mathcal{B}$, $Succ(x,.)$)
[63]  FOR EACH universal-restriction $v \in \mathcal{B}$   (comment: $v$ is of form $\forall R.C$)
[64]      IF ($x$ has no $R$-neighbor) and (($\geq_2 R$) $\in \mathcal{B}$) THEN
[65]          allocate a new tableau node $s$;
[66]          $\mathcal{L}(\langle x,s \rangle) := \{R\}$;
[67]          $\mathcal{L}(s) := \emptyset$;
[68]          $Succ(x,R) := Succ(x,R) \cup \{s\}$;
[69]      ENDIF
[70]      FOR EACH $s \in Succ(x,R)$   (comment: $R$ stands for any role)
[71]          $\mathcal{L}(s) := \mathcal{L}(s) \cup \{C\}$;
[72]      ENDFOR
[73]  ENDFOR
END-PROCEDURE {AllSuccessors}
```

AllSuccessors($\mathcal{B}$, $Succ(x,.)$) implements a combination of the $\forall$-rule and the $\geq$-rule, where $Succ(x,.)$ is an input and output parameter and $\mathcal{B}$ is an input parameter.

4.6 Equisatisfiability of the Preprocessing Step

A few comments are necessary: (1) Formulae like $\exists R.C$ are called *existential constraints*, and formulae like $\forall R.C$ *universal constraints.* Both of them are also called *modal constraints*, generally denoted as $\overset{\exists}{\underset{\forall}{}} R.tag(C)$ hereafter; (2) The *tag*(.) operation (recursively) maps each occurrence of a specific *modal constraint* to a conjunction of two conjuncts, one of which is a Ramsey atom[9]; (3) In each tagged constraint we stipulate that the Ramsey atom is on the left and the modal constraint on the right. This simplifies our discussion by excluding (artificial) cases like $\overset{\exists}{\forall}R.tag(C) \sqcap Q(x)$.

Definition 3. ***(p-model)*** Given $E_1/\mathcal{K}_1$ a tagged formula/Tbox, $Q(x) \sqcap \overset{\exists}{\forall}R.tag(C)$ a tagged constraint in E_1 and $\mathcal{K}_1$, a p-model for E_1 and $\mathcal{K}_1$ is a model $(\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ such that for any $n \in \Delta^{\mathcal{I}}$ it holds that

(1) if $n \in (Q(x))^{\mathcal{I}}$, then $n \in (\overset{\exists}{\forall}R.tag(C))^{\mathcal{I}}$; and

(2) if $n \in (\overset{\exists}{\forall}R.tag(C))^{\mathcal{I}}$ where $\overset{\exists}{\forall}R.tag(C)$ was tagged, then there is some Ramsey atom $Q(x)$ for $\overset{\exists}{\forall}R.tag(C)$ such that $n \in (Q(x))^{\mathcal{I}}$.

Lemma 4. *If E_1 and $\mathcal{K}_1$ are satisfiable, then they are satisfiable in a p-model.*

Proof. Given $E_1/\mathcal{K}_1$ as the converted formula/axioms, let $(\Delta^{\mathcal{I}_1}, .^{\mathcal{I}_1})$ be a model for them. To construct a p-model $(\Delta^{\mathcal{I}_2}, .^{\mathcal{I}_2})$, consider each $n \in \Delta^{\mathcal{I}_1}$ having:

(1) $n \in (Q_i(x))^{\mathcal{I}_1}$ but there is no $\overset{\exists}{\forall}R.tag(C)$ s.t. $n \in (\overset{\exists}{\forall}R.tag(C))^{\mathcal{I}_1}$; or

(2) $n \in (\overset{\exists}{\forall}R.tag(C))^{\mathcal{I}_1}$, but there is no $Q_i(x)$ s.t. $n \in (Q_i(x))^{\mathcal{I}_1}$.

Use the sub-model generating technique[10] to exclude the superfluous $\overset{\exists}{\forall}R.tag(C)$ or $Q_i(x)$ in both cases when constructing $\mathcal{I}_2$ from $\mathcal{I}_1$: let n be an element of $\mathcal{I}_1$, we construct a corresponding element n' in $\mathcal{I}_2$ by extracting other interpretations (i.e., concepts) from n unchanged to n' (except those superfluous concepts), extracting other elements in the domain of $\mathcal{I}_1$ in the same way, and extracting role interpretations for elements in $\Delta^{\mathcal{I}_2}$. Starting from a root node n_0 (corresponding to the input concept E_1) of $\mathcal{I}_1$, one is able to construct a $(\Delta^{\mathcal{I}_2}, .^{\mathcal{I}_2})$ to meet the two requirements of being a p-model. So, $(\Delta^{\mathcal{I}_2}, .^{\mathcal{I}_2})$ is a p-model provided that $(\Delta^{\mathcal{I}_1}, .^{\mathcal{I}_1})$ be a model. $\square$

Lemma 5. *If E_1 and $\mathcal{K}_1$ have a model, then $\mathcal{K}_a$ is satisfiable in that model.*

Proof. Suppose E_1 and $\mathcal{K}_1$ has a p-model $\mathcal{M} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ in which $\mathcal{K}_a$ is not satisfied. This means that the existence of at least one axiom $s \in \mathcal{K}_a$ being violated at some element $n \in \Delta^{\mathcal{I}}$.

[9]Let us call the unique concept name introduced as Ramsey atom. It is in honor of Frank P. Ramsey, who in 1926 observed the C-rule (a.k.a. the Ramsey's Rule [Ram31]) which is about the equivalence of converse modalities in modal logics.

[10]See [DM00] and the reference therein for a foundational theory that also supports the *subset tableaux caching technique* used in tableau-based decision procedures for DLs.

Case-1: By the definition of $\mathcal{K}_a$, this s corresponds to some tuple $Q(x) \sqcap \forall R.tag(C) \in \mathcal{U}(R)$ for some role R. Suppose the axiom being violated at element n be $\top \sqsubseteq tag(C) \sqcup \forall R^{-}.\neg Q(x)$. This means there is some $m \in \Delta^{\mathcal{I}}$ such that

(1) $(m,n) \in R^{\mathcal{I}}$, and

(2) $m \in (Q(x))^{\mathcal{I}}$.

From the definition of the p-model, we know $(Q(x))^{\mathcal{I}} \subseteq (\forall R.tag(C))^{\mathcal{I}}$. So,

(3) $m \in (\forall R.tag(C))^{\mathcal{I}}$.

By (3) and (1), it follows

(4) $n \in (tag(C))^{\mathcal{I}}$.

By (4), the axiom $\top \sqsubseteq tag(C) \sqcup \forall R^{-}.\neg Q(x)$ is satisfied at n.

Case-2: Suppose this axiom s corresponds to some tuple $P(x) \sqcap \exists R.tag(C) \in \mathcal{E}(R)$ for some role R. In this case, $\top \sqsubseteq tag(C) \sqcup \forall R^{-}.(\neg P(x) \sqcup \geq_2 R)$ is the axiom being violated at element n, which requires an element $m \in \Delta^{\mathcal{I}}$ such that

(1′) $(m,n) \in R^{\mathcal{I}}$, and

(2′) $m \in (P(x))^{\mathcal{I}}$, and

(3′) $m \in (\leq_1 R)^{\mathcal{I}}$.

By the definition of the p-model, we know $(P(x))^{\mathcal{I}} \subseteq (\exists R.tag(C))^{\mathcal{I}}$. So,

(4′) $m \in (\exists R.tag(C))^{\mathcal{I}}$.

Consider (1′) (3′) (4′), we have

(5′) $n \in (tag(C))^{\mathcal{I}}$.

By (5′), the axiom $\top \sqsubseteq tag(C) \sqcup \forall R^{-}.(\neg P(x) \sqcup \geq_2 R)$ is satisfied at n.

Since in both cases the supposedly violated axioms are actually satisfied, this concludes that a p-model for both E_1 and $\mathcal{K}_1$ is also a model for $\mathcal{K}_a$. $\square$

Lemma 6. *E_1 is satisfiable w.r.t. $\mathcal{K}_1 \cup \mathcal{K}_a$ iff E_1 is satisfiable w.r.t. $\mathcal{K}_1$.*

Proof. (only if direction) Let $\mathcal{M}_1$ be a model for E_1 and $\mathcal{K}_1 \cup \mathcal{K}_a$. Observe that $\mathcal{K}_1 \cup \mathcal{K}_a$ is a superset of $\mathcal{K}_1$, E_1 is satisfiable w.r.t. $\mathcal{K}_1$ (trivially in $\mathcal{M}_1$).

(if direction) Let $\mathcal{M}_2$ be a p-model for E_1 and $\mathcal{K}_1$. According to the previous lemma, $\mathcal{K}_a$ is always satisfied in the p-model for both E_1 and $\mathcal{K}_1$. It follows that $\mathcal{M}_2$ is a model for E_1 and $\mathcal{K}_1 \cup \mathcal{K}_a$. $\square$

Lemma 7. *Given a concept formula E_0 and a Tbox $\mathcal{K}_0$, let the tagged formula and Tbox be E_1 and $\mathcal{K}_1$. E_0 is satisfiable w.r.t. $\mathcal{K}_0$ iff E_1 is satisfiable w.r.t. $\mathcal{K}_1$.*

Proof. Given a model $\mathcal{M}_1$ of E_1 and $\mathcal{K}_1$, E_0 and $\mathcal{K}_0$ are trivially satisfied in $\mathcal{M}_1$. In the other direction, given a model $\mathcal{M}_0$ of E_0 and $\mathcal{K}_0$, adding Q_i accordingly as guided by $\mathcal{M}_0$ will produce a model for E_1 and $\mathcal{K}_1$. This concludes that the tagging operation preserves satisfiability. □

Theorem 8. *E_0 is satisfiable w.r.t. $\mathcal{K}_0$ iff E_1 is satisfiable w.r.t. $\mathcal{K}_1 \cup \mathcal{K}_a$.*

Proof. This immediately follows from Lemma 6 and Lemma 7. □

4.7 Correctness of the Decision Procedure

4.7.1 Completeness

For the completeness, it is proper to prove the correctness for what regards concept unsatisfiability. Recall the procedure $SAT(\cdot,\cdot,\cdot,\cdot)$ and that `Nogood` is a global and permanent data structure. Taking the approach in [DM00], we start with a lemma saying that $\perp$-rules correctly propagate inconsistencies.

Lemma 9. *The $\perp$-rules generate only unsatisfiable sets.*

Proof. By induction on the application of $\perp$-rules.

Base cases. Consider rules $\perp$-0, $\perp$-1 and $\perp$-2. They are clearly unsatisfiable.

Inductive cases. Suppose the claim holds for the antecedent of each $\perp$-rule. Let's analyze the application of each $\perp$-rule.

- ($\perp$-3): We prove the claim by contradiction. Suppose $\alpha \subseteq \beta$, α is unsatisfiable and β is satisfiable. Let M be a model for β. Using the sub-model generating technique, there is a sub-model N of M which satisfies α, and this contradicts the hypothesis that α is unsatisfiable.
- ($\perp$-4): We prove the claim by contradiction. Suppose $\alpha \sqcap C$ and $\alpha \sqcap D$ are unsatisfiable, but $\alpha \sqcap (C \sqcup D)$ is satisfiable. Let M be a model for $\alpha \sqcap (C \sqcup D)$, then either $\alpha \sqcap C$ or $\alpha \sqcap D$ is satisfied in M. This contradicts the hypothesis.

- ($\perp$-5): Assume α is unsatisfiable and $\alpha \subseteq \{E_0, E_1, ..., E_n\}$. By $\perp$-3, $\{E_0, E_1, ..., E_n\}$ is unsatisfiable. Suppose β is satisfiable in a model M. This gives a sub-model for $\{E_0, E_1, ..., E_n\}$. This results in a contradiction.

- ($\perp$-6): Assume α is unsatisfiable and $\alpha \subseteq \{E_1, ..., E_n\}$. By $\perp$-3, we have $\{E_1, ..., E_n\}$ unsatisfiable. Suppose β is satisfied in a model M. This gives a sub-model for $\{E_1, ..., E_n\}$ and is a contradiction.

□

Lemma 10. *If* $\alpha \in$ *Nogood, then* α *is unsatisfiable.*

Proof. Recall that Nogood is a data structure for storing $\perp$-sets that are obtained by an application of only the $\perp$-rules in the sub-procedure $SAT(\cdot,\cdot,\cdot,\cdot)$. □

Theorem 11. (Completeness) *If* α *is satisfiable, then* $\alpha \notin$ *Nogood.*

Proof. This follows from Lemma 10. □

4.7.2 Soundness

Let $\mathbf{T}$ denote the tree constructed by the decision procedure. For each node $t_i \in \mathbf{T}$, we use $\mathcal{L}(t_i)$ to denote t_i's initial label, and use $\mathcal{B}(t_i)$ to denote t_i's label for its current propositional branch.

$SAT(\cdot,\cdot,\cdot,\cdot)$ takes a depth-first traversal to expand $\mathbf{T}$ starting from its root. If t_i is explored (i.e., expanded, completed) before t_j, let's denote it as $t_i \succ t_j$. The blocking relationship of nodes, written in the symbol $\tilde{\succ}$, is required to conform to the node exploration ordering[11] (in symbol $\succ$). If t_i blocks t_j, this is denoted in $t_i \tilde{\succ} t_j$.

Transitive blocking is allowed as long as the blocking node can be *propositionally completed* (i.e., it has a current propositional branch not known to be unsatisfiable and to which the $\sqcap$-rule and the $\sqcup$-rule are no longer applicable.).

Only for a completed node its label is added to Witness. If an element $w \in$ Witness is the label of a tableau node x, we write $node(w) = x$. Recall the procedure $SAT(\cdot,\cdot,\cdot,\cdot)$, if neither $\mathcal{L}(x)$ nor $\mathcal{B}(x)$ is in Nogood or has clashes, they are added to Witness right before exploring x's successors.

[11] $\succ$ and $\tilde{\succ}$ are two binary relations on tree nodes. Since $\succ$ is acyclic, $\tilde{\succ}$ is acyclic.

When visiting/exploring/expanding a new node y, the procedure first checks if either $\mathcal{L}(y)$ or $\mathcal{B}(y)$ matches any element of `Witness`. If there is a $w \in$ `Witness` equal to either $\mathcal{B}(y)$ or $\mathcal{L}(y)$, then y is blocked by $node(w)$. Clearly $node(w)$ has been explored prior to y, and the blocking is recorded as $node(w) \tilde{\succ} y$. Notice that if $\mathcal{B}(y) \in$ `Witness`, then $\mathcal{L}(y)$ will be added to `Witness`[12]. However, `Witness` is emptied (by the *restart strategy*) whenever one new `Nogood` is inferred.

Lemma 12. (Backward Propagation Don't-care) *Let Q_1 and P_1 be the Ramsey atoms respectively for $\forall R^-.tag(C)$ and $\exists R^-.tag(C)$. Given a node x that is completed and one of its R-successor nodes y that is only propositionally completed:*

- *(1) if $\mathcal{B}(y) \supseteq \{Q_1, \forall R^-.tag(C)\}$, then $tag(C) \in \mathcal{L}(x)$;*
- *(2) if $\mathcal{B}(y) \supseteq \{P_1, \exists R^-.tag(C), (\leq_1 R^-)\}$, then $tag(C) \in \mathcal{L}(x)$.*

Proof. (1) Consider enforcing the *general concept inclusion* $\top \sqsubseteq \forall R.\neg Q_1 \sqcup tag(C)$ at node x. It is impossible for x to choose $\forall R.\neg Q_1$, for otherwise y (an R-successor to x) will contain a primitive clash $\{Q_1, \neg Q_1\}$. Therefore, it is only possible for x to choose $tag(C)$.

(2) Consider enforcing $\top \sqsubseteq \forall R.(\neg P_1 \sqcup (\geq_2 R^-)) \sqcup tag(C)$ at node x. It is impossible for x to choose $\forall R.(\neg P_1 \sqcup (\geq_2 R^-))$, for otherwise y (an R-successor to x) will contain a clash $\{\neg P_1 \sqcup (\geq 2R^-), (\leq 1R^-), P_1\}$. $\square$

The above lemma states that, in the course of constructing a tableaux structure (a pre-model) for a given *concept satisfiability problem*, "backward propagation" of constraints can be circumvented if one considers a set of extra *general concept inclusions* (which are obtained in the preprocessing step, see Section 4.3). In $\mathcal{ALCI}$ (and also $\mathcal{SHI}$), the backward propagation of constraints is only caused by *universal constraints*. However, in $\mathcal{ALCFI}$, *functional constraints* can also cause backward propagation of constraints. Lemma 12 considered these two cases of "backward propagation" in $\mathcal{ALCFI}$ (and also $\mathcal{SHIF}$).

Lemma 13. (Soundness) *If there is a tableau tree $\boldsymbol{T}$ for E_2 w.r.t. $\top \sqsubseteq G_2$, then there is a model T for E_2 w.r.t. $\top \sqsubseteq G_2$.*

[12]See line-33 and line-34 of procedure $SAT(\cdot,\cdot,\cdot,\cdot)$.

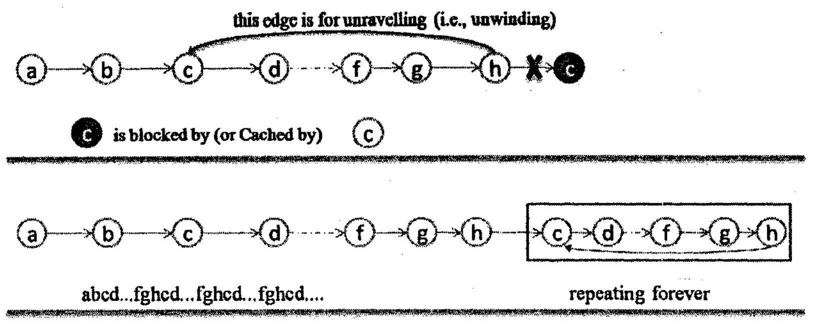

this edge is for unravelling (i.e., unwinding)
a
b
c
d
f
g
h
c
is blocked by (or Cached by)
abcd...fghcd...fghcd...fghcd....
repeating forever

Figure 4.3: Blocking and Unravelling

Proof. Recall that (see the very beginning of Section 4.4) we discuss the converted problem which is to test the satisfiability of E_2 w.r.t. $\mathcal{K}_2$ ($\top \sqsubseteq G_2$). It takes two steps. First, the tableau (completion tree) $\mathbf{T}$ is updated to get $\mathbf{T}'$, and then the unraveling technique is applied to $\mathbf{T}'$ to get a model.

(1) For each node $x \in \mathbf{T}$ that is not blocked, if $(\geq_2 R) \in \mathcal{B}(x)$ and x has only one R-neighbor y, then make a copy y' of y, put y' as x's R-successor, and establish the blocking $y \tilde{\succ} y'$[13]. This results in the tableau $\mathbf{T}'$.

(2) To admit an infinite model, it is necessary to consider paths in $\mathbf{T}'$. The mapping $\mathbf{Tail}(p)$ is used to return the last element in a path p. Give a path $p = [x_0, ..., x_n]$, where x_i are nodes in $\mathbf{T}'$, $\mathbf{Tail}(p)= x_n$. Paths in $\mathbf{T}'$ are defined inductively as follows:

- for the root node x_0 in $\mathbf{T}'$, $[x_0]$ is a path in $\mathbf{T}'$.
- for a path p and a node x_i in $\mathbf{T}'$, $[p, x_i]$ is a path in $\mathbf{T}'$ iff
  - x_i is not blocked, and
    * x_i is a successor of $\mathbf{Tail}(p)$, or
    * y is a successor of $\mathbf{Tail}(p)$ and x_i (transitively) blocks y.

The model $T = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ can be defined with:

$\Delta = \{x_p \mid p \text{ is a path in } \mathbf{T}' \}$

$x_p \in (\mathcal{L}(\mathbf{Tail}(p)) \sqcap \mathcal{B}(\mathbf{Tail}(p)))^{\mathcal{I}}$

$\{\langle x_p, x_q\rangle \mid \langle x_p, x_q\rangle \in (R)^{\mathcal{I}}\} = \{\langle x_p, x_q\rangle \in \Delta \times \Delta \mid \; q = [p, \mathbf{Tail}(q)]$ and

[13]The case $(\geq_2 R) \in \mathcal{B}(x)$ and x has no R-neighbor does not exist, see the $\geq$-rule and also line-64 of procedure $AllSuccessors(\cdot,\cdot)$.

1. $\mathbf{Tail}(q)$ is an R-successor of $\mathbf{Tail}(p)$, or

2. $\exists y \in \mathbf{T}'$, y is an R-successor of $\mathbf{Tail}(p)$ and $\mathbf{Tail}(q) \tilde{\succ} y$ }

$\bigcup$ $\{\langle x_p, x_q \rangle \in \Delta \times \Delta \mid p = [q, \mathbf{Tail}(p)]$ and

1. $\mathbf{Tail}(p)$ is an R^--successor of $\mathbf{Tail}(q)$, or

2. $\exists y \in \mathbf{T}'$, y is an R^--successor of $\mathbf{Tail}(q)$ and $\mathbf{Tail}(p) \tilde{\succ} y$ }

T is a model for E_1 w.r.t. G_2. □

4.7.3 Complexity

Lemma 14. *Each execution of the procedure $SAT(\cdot,\cdot,\cdot,\cdot)$ takes a cost of $2^{O(n)}$, where n is the problem size.*

Proof. Notice that the procedure $SAT(\cdot,\cdot,\cdot,\cdot)$ terminates if a new `Nogood` is found or if the tableau (completion tree) is saturated.

(1) For any tableau node x, considering the concept G_2 (the RHS of a single general concept inclusion $\top \sqsubseteq G_2$) and the concepts in $\mathcal{L}(x)$. The number of topmost $\sqcap$ and $\sqcup$ operators (i.e., the $\sqcap$ and $\sqcup$ operators that are subject to the $\sqcap$-rule and the $\sqcup$-rule) is bounded by n (where n is the size of the given problem and $n \geq \|G_2\|$) for x. This implies that the number of potential propositional branches for node x is $2^{O(n)}$.

(2) The number of expanded nodes (i.e., those nodes having successors due to the application of the $\exists$-rule) is bounded by 2^n due to the blocking strategy (using `Witness` to avoid repetitive propositional branch or initial label) in $SAT(\cdot,\cdot,\cdot,\cdot)$. The number of nodes in the final tableau structure (including blocked nodes which have no successors) is bounded by $2^n + 2^n * n$.

(3) The cost per node could not surpass $2^{O(n)}$ (considering the cost for selecting propositional branches and the cost for cache look-up operations).

Considering all these three factors, the total cost for one execution of $SAT(\cdot,\cdot,\cdot,\cdot)$ is thus bounded by $2^{O(n)}$. □

Lemma 15. *The procedure $TEST(\cdot,\cdot)$ stops after at most $2^{O(n)}$ loops, where n is the problem size.*

Proof. (1) The size of `Nogood` can be bounded by $h(n) = 2^{O(n)}$. This is justifiable when considering the powerset of the space of sub-formulae for the given problem in question.

(2) Each run of $TEST(\cdot,\cdot)$ will report at least one new Nogood if that run does not produce a saturated tableaux structure. After at most $h(n)$ runs of $TEST(\cdot,\cdot)$, the size of Nogood will not grow (because the whole sub-formulae space is exhausted). At this point (i.e., in the worst case), $TEST(\cdot,\cdot)$ is able to decide the satisfiability of the problem in question and terminates. $\square$

Theorem 16. *The concept satisfiability test w.r.t. general axioms in $\mathcal{ALCFI}$ can be decided in deterministic exponential time by the presented tableau decision procedure.*

4.8 Discussion

The DL $\mathcal{ALCFI}$ is chosen not simply because it was extensively studied in the past, but because this logic is the juncture point of several interesting research results established in the past as well.

For example, it is the logic where the *finite model* property is lost and the dynamic *pair-wise blocking* technique was proposed for; it is the logic where *functional restrictions* can be eliminated; and it is also the logic where the complexity of its satisfiability problem goes up to NExpTime-hard when *nominals* are added. Each of the above mentioned results is an advancement and has impact on either practical or theoretical research.

The *encoding method* for eliminating *functional restrictions* from an $\mathcal{ALCFI}$ knowledge base was presented[14] in [Gia96, CGR98, GM00]. Our approach shares similarity with their work in introducing new concept names and encoding new constraints in the form of *general concept inclusions.* The difference is that we keep functional restrictions untouched but eliminate the effect of *backward propagation* of constraints on a tableaux structure.

[Tob00, Lut04] proved that the DL $\mathcal{ALCFIO}$ is NExpTime-hard. The conversion would be incorrect if there is a *nominal.* This limitation coincides with the "merging pattern" as pointed out in the improved hardness proof in [Lut04]. If there are only *nominals* and *inverse roles* but no *functional restrictions,* the presented conversion is satisfiability-preserving[15].

[14]In one of those works, the connection between an "infinite model" in $\mathcal{ALCFI}$ and its corresponding "finite model" in $\mathcal{ALCF}$ was briefly pointed out. There exist further extensions for number restrictions, interested readers are referred to [CGLN01].

[15]Recently in [Lut07], it was proved that the *conjunctive query entailment* problem is NExpTime-hard in $\mathcal{ALCI}$ but remains ExpTime-complete in $\mathcal{SHQ}$.

For DLs supporting inverse roles, the *termination mechanism* (a.k.a. cycle detection and blocking techniques) for tableau-based decision procedures is required to be dynamic to accommodate *backward propagation* of constraints [HS99]. To further support *number restrictions,* the more sophisticated dynamic *pair-wise blocking* technique [HS99] and its optimized variants [HS02] were introduced to guarantee soundness. A study of the model properties of DLs with inverse roles shows that various dynamic termination mechanisms are sufficient. Generally speaking, the bidirectional traversal on a relational structure is a natural consequence of enforcing *tableau expansion rules* over inverse relations. However, there are some undesirable side effects from bidirectional propagation of constraints, e.g., one major problem is the unsoundness of the *tableau caching* technique [BCM+03], and the second problem is the save and restore problem (as mentioned before in Section 4.1.1 and Section 4.1.2), the third problem is the lack of efficient propagation of unsatisfiability, and the final problem is that the size of the tableaux structure could become unnecessarily large. Further, though the book-keeping cost for a sequence of establishing, breaking and re-establishing of blockings is minor for small knowledge bases, the cost could be amplified for larger knowledge bases, and it also entails complicated design and implementation issues. This chapter presented a way to adapt a *static termination mechanism* to tableau-based decision procedures for the concept satisfiability test w.r.t. general axioms in $\mathcal{ALCFI}$. The *static termination mechanism* is intuitively simple and easily implementable. The major run-time performance advantage for *static blocking* and one-way traversal is that it requires a small footprint algorithm, thus it is well suited to resource limited computer systems[16]. The independence of sibling nodes of a tableaux structure also implies implementations that explore the multi-threading mechanism and multi-core computer systems more efficiently.

In the beginning of this chapter, a transformation is introduced to convert a source problem to a target problem and to preserve equisatisfiability in $\mathcal{ALCFI}$. The search of a model in the converted problem can safely use the *static equality termination* mechanism (a.k.a. the *equality blocking* technique [BCM+03]) globally. Based on the static blocking mechanism and (a very weak form of) the inconsistency propagation capability, one is then able to show that the decision procedure runs in exponential time in the worst case for the concept satisfiability problem in $\mathcal{ALCFI}$.

[16]The virtual memory might be huge but is often much slower than physical memory, and frequent page swapping can cause low-level performance thrashing. Small footprint algorithms can be significantly faster than memory intensive algorithms.

The decision procedure relies on two functionalities: the *static equality blocking* which is globally applicable but transient, and the unsatisfiability caching which is globally applicable and persistent. [DM00] is credited for this idea. As shown in [GN07], the data structure `Witness` here can possibly be persistent as well under certain conditions. We demonstrated a decision procedure using the *restart strategy.* Whenever one new unsatisfiable set (of labels) is obtained and it is not the given problem itself, the decision procedure restarts. In other cases, the procedure terminates and is able to decide the satisfiability of the problem. Though the use of the *restart strategy* here is for an easy analysis of the complexity, it is also of potential advantage to the run-time performance for realistic problems. For example, it is possible to assert artificial elements in `Nogood` (as a penalty) to bypass the searching of certain branches which were timed out before. Possibly this might help in practice because the reasoning system can somehow choose to explore "easier" branches first.

An ExpTime decision procedure for a DL with both *inverse roles* and *functional restrictions* would not be ready but for a polynomial-time conversion. This is because the decision procedure is correct only for problems with a special property (which is called "backward propagation don't-care"). By a satisfiability-preserving conversion, nonetheless this decision procedure works for $\mathcal{ALCFI}$ in general (actually for $\mathcal{SHIF}$ which corresponds to the ontology language OWL-lite). The converted problem is of size $O(n^2)$ where n is the size of the source problem. However, a linear conversion is possible in theory according to [Lut99].

A transformation system[17] was developed for the conversion and comparison tests were carried out. It is observed that, as expected, fresh concept names and new *general concept inclusions* have a modest size expansion by a factor of 5, and a run-time performance penalty of a factor of around 3 to 4. But for a set of hard problems, this penalty is well offset by the global tableaux caching functionality, and magnitudes of performance gain have been achieved for half of the test cases. As already discussed in the background section (at the end of Section 4.1.2), the *tableaux caching* technique can serve not only as a static *blocking technique*, it can also prune away "bad parts" of the search space. By contrast with running problems in DLs with inverse roles, running the converted

[17]Our current transformation system is not able to process $\mathcal{ALCFI}$. The tests were carried out in $\mathcal{ALCI}$ only. Recently we noticed a report [MSH07] on using the "anywhere dynamic pairwise blocking" technique to classify the GALEN (Simplified) Ontology in 104 seconds. Recall that the "anywhere dynamic pairwise blocking" is more or less a variant of what was presented in Chapter 2 (with the least caching power of the three caching techniques proposed in this thesis). The GALEN ontology is specified in $\mathcal{SHIF}$. Once we have completed the extended transformation for $\mathcal{SHIF}$, it will be quite promising for us to test the proposed technique in this chapter against GALEN.

problems in DLs without inverse roles likely demands less memory (for there is no need to make an extra copy of the tableaux structure before branching so that an undoing will be possible). It is quite promising that the *global (sub)tableaux caching* technique can be used in tableau-based DL systems to try some realistic problems yet unsolved because of inverse roles.

In Chapter 6, we will deal with $\mathcal{SHIQ}$ and present a more general, but less powerful[18], global tableaux caching technique.

[18] In the sense that *initial labels* are not used for the propagation of inconsistency.

Chapter 5

$\mathcal{SHOI}$, $\mathcal{SHQ}$, and $\mathcal{ALCHQI}$ Acyclic Tbox

This chapter presents three different conversions that eliminate either a *role hierarchy* or *inverse roles.* The first is to convert $\mathcal{SHQ}$ to $\mathcal{ALCQ}$ by eliminating a role hierarchy and transitive roles. The second is to convert $\mathcal{SHOI}$ to $\mathcal{SHO}$ by eliminating *inverse roles*. The last one is to convert the satisfiability problem from $\mathcal{ALCHQI}$ to $\mathcal{ALCHQ}$ while keeping Tbox axioms in an acyclic form.

5.1 Reducing $\mathcal{SHQ}$ to $\mathcal{ALCQ}$

A conversion from a $\mathcal{SHQ}$ concept to an $\mathcal{ALCQ}$ Tbox is described as follows. It eliminates a role hierarchy and transitive roles by introducing new *concept names* and *concept inclusions.*

(1) First, introduce a new role name g for the conversion;

(2) Introduce a new concept name for each role name occurred in the given concept;

(3) Process the *role hierarchy* as follows: replace each *role inclusion* $R_1 \sqsubseteq R_2$ with a *concept inclusion* $C_{R_1} \sqsubseteq C_{R_2}$, where C_{R_i} is the *concept* introduced for role R_i.

(4) Process *modal constraints* on transitive roles as follows:

(4.1) for each $\forall S.F$ (where S is *transitive* and the concept assigned to it is C_S), replace it with a fresh *concept name* Q;

(4.1.1) introduce a new *concept name* A_F, and add the following two axioms:

(4.1.1.1) $\neg Q \sqsubseteq \exists g.(C_S \sqcap \neg A_F)$

(4.1.1.2) $Q \sqsubseteq \forall g.(\neg C_S \sqcup (Q \sqcap A_F))$

(4.1.2) add two axioms $A_F \sqsubseteq F$ and $\neg A_F \sqsubseteq \neg F$, and recursively deal with F and $\neg F$.

(4.2) for each $\exists S.F$ (where S is *transitive* and the concept assigned for it is C_S), replace it with a fresh *concept name* P;

(4.2.1) introduce a new *concept name* A_F, and add the following two axioms:

(4.2.1.1) $\neg P \sqsubseteq \forall g.(\neg C_S \sqcup (\neg P \sqcap \neg A_F))$

(4.2.1.2) $P \sqsubseteq \exists g.(C_S \sqcap A_F)$

(4.2.2) add two axioms $A_F \sqsubseteq F$ and $\neg A_F \sqsubseteq \neg F$, recursively process F and $\neg F$.

(5) Process *modal constraints*[1] on non-transitive roles. Replace $\exists^{\leq n} R.D$ with $\exists^{\leq n} g.(C_R \sqcap A_D)$; replace $\exists^{\geq n} R.D$ with $\exists^{\geq n} g.(C_R \sqcap A_D)$, where A_D is a fresh *concept name* for D. Then add two *concept inclusions* $A_D \sqsubseteq D$ and $\neg A_D \sqsubseteq \neg D$, and recursively process D and $\neg D$.

Example 1. Let the role hierarchy be $\{R_1 \sqsubseteq R_3, R_2 \sqsubseteq R_3\}$. Given a set of *number restrictions* $\{\exists^{\geq 3} R_1.C, \exists^{\geq 5} R_3.C, \exists^{\leq 5} R_3.C, \exists^{\geq 4} R_2.C\}$ for satisfiability testing.

First, to eliminate the given role hierarchy, we introduce new concept names C_{R_1} for R_1, C_{R_2} for R_2, and C_{R_3} for R_3, and introduce concept inclusions $C_{R_1} \sqsubseteq C_{R_3}$ and $C_{R_2} \sqsubseteq C_{R_3}$.

Second, the set of *number restrictions* is converted to $\{\exists^{\geq 3} g.(C_{R_1} \sqcap C), \exists^{\geq 5} g.(C_{R_3} \sqcap C), \exists^{\leq 5} g.(C_{R_3} \sqcap C), \exists^{\geq 4} g.(C_{R_2} \sqcap C)\}$.

The converted problem has one role g, and a solution for the converted problem is:

2 g-successors with role filler $\{C_{R_1}, C_{R_2}, C_{R_3}, C\}$

2 g-successors with role filler $\{C_{R_2}, C_{R_3}, C\}$

1 g-successor with role filler $\{C_{R_1}, C_{R_3}, C\}$

A solution to the converted problem corresponds to a solution to the original problem (and vice versa):

2 $\{R_1, R_2, R_3\}$-successors with role filler $\{C\}$

2 $\{R_2, R_3\}$-successors with role filler $\{C\}$

1 $\{R_1, R_3\}$-successor with role filler $\{C\}$

The idea behind an elimination of a *role hierarchy* is quite intuitive: introducing concept names to simulate a *role hierarchy*. However, it would be difficult to eliminate a role hierarchy in the presence

[1]Please note that a *universal constraint* like $\forall R.C$ can be expressed in $\exists^{\leq 0} R.\neg C$ and an *existential constraints* like $\exists R.C$ can be expressed in $\exists^{\geq 1} R.C$. Therefore, for *modal constraints* we only consider *number restrictions* in general.

of *inverse roles* because *inverse relations* are not considered as Boolean expressible [OK99]. In the following we show a way to eliminate *inverse roles* from a *role hierarchy.*

5.2 Reducing $\mathcal{SHOI}$ to $\mathcal{SHO}$

In Chapter 3, we showed how to eliminate inverse relations from an Abox and an acyclic Tbox in the DL $\mathcal{ALCI}$. In [DHW07], we used a similar technique to eliminate inverse relations for a general Tbox in $\mathcal{ALCI}$. We will use the same idea in this section to eliminate inverse relations from $\mathcal{SHOI}$, a DL having expressive roles and nominals. We convert a $\mathcal{SHOI}$ concept to a $\mathcal{SHO}$ Tbox.

(1) For each *role inclusion* in the *role hierarchy*, replace it with two axioms:

(1.1) case $R_i \sqsubseteq R_j$, then introduce $R_i^a \sqsubseteq R_j^a$ and $R_i^b \sqsubseteq R_j^b$

(1.2) case $R_i^- \sqsubseteq R_j$, then introduce $R_i^b \sqsubseteq R_j^a$ and $R_i^a \sqsubseteq R_j^b$

(1.3) case $R_i \sqsubseteq R_j^-$, then introduce $R_i^a \sqsubseteq R_j^b$ and $R_i^b \sqsubseteq R_j^a$

(1.4) case $R_i^- \sqsubseteq R_j^-$, then introduce $R_i^a \sqsubseteq R_j^a$ and $R_i^b \sqsubseteq R_j^b$

Note: R_i^a (R_i^b) is a unique role name introduced for role R (R^-).

(2) For each *universal constraint* $\forall S.C$ (where S is *transitive*), replace it with a new *atomic concept name* P. Add the following axioms:

(2.1) $\exists S^b.P \sqsubseteq P \sqcap A_C$ and $P \sqsubseteq \forall S^a.(P \sqcap A_C)$

(2.2) $A_C \sqsubseteq C$, where A_C is a fresh *concept name*

(2.3) $\neg P \sqsubseteq \exists S^a.(\neg A_C)$ and $\neg A_C \sqsubseteq \neg C$

(2.4) recursively process C (and $\neg C$)

Similarly process *universal constraints* of the form $\forall S^-.C$ for transitive role S.

(3) For each *universal constraint* $\forall R.C$ (R is not *transitive*), replace it with a new *atomic concept name* Q. Add axioms as follows:

(3.1) $\exists R^b.Q \sqsubseteq A_C$, and $Q \sqsubseteq \forall R^a.A_C$

(3.2) $A_C \sqsubseteq C$, where A_C a fresh concept name

(3.3) $\neg Q \sqsubseteq \exists R^a.(\neg A_C)$, and $\neg A_C \sqsubseteq \neg C$

(3.4) recursively process C (and $\neg C$)

Similarly process *universal constraints* of the form $\forall R^-.C$ for non-transitive R.

Remark. The above steps convert a $\mathcal{SHOI}$ concept to a $\mathcal{SHO}$ concept and a set of $\mathcal{SHO}$ *general concept inclusions*. Because in $\mathcal{SHO}$ *general concept inclusions* can be internalized to a $\mathcal{SHO}$ concept, one can see that the above transformation results in a $\mathcal{SHO}$ concept from a $\mathcal{SHOI}$ concept (if further using the internalization technique [BCM+03]).

5.3 Reducing $\mathcal{ALCHQI}$ Acyclic Tbox

In [Lut99] and [BML+05] PSPACE-tableau-based decision procedures have been shown respectively for acyclic Tboxes in $\mathcal{ALC}$ and in $\mathcal{ALCOQ}$. The notion of *modal depth* [Lut99, BML+05] is critical in giving a polynomial upper bound on the length of any *trace* the tableaux decision procedure explores. Here, based on a similar observation, one is able to convert an $\mathcal{ALCHQI}$ acyclic Tbox to an $\mathcal{ALCHQ}$ acyclic Tbox[2].

Let us consider a given acyclic Tbox $\mathcal{T}_0$ and a given concept expression C in $\mathcal{ALCHQI}$. Assume one wishes to test the *concept satisfiability* of C w.r.t. $\mathcal{T}_0$. In the following, to weave a "web" of modal-reference relations for a concept and its *role filler*, it is necessary to clearly specify the position of each modal constraint (a.k.a. qualified number restriction) in terms of "levels".

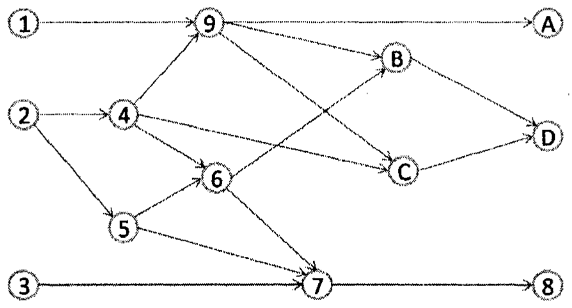


Figure 5.1: The Modal-reference Relation of an Imaginary Acyclic Tbox

Figure 5.1 shows a network of modal-reference relation for an exemplary acyclic Tbox (modulo propositional reference relations). A circle represents one *modal concept*, and an arrow represents a modal reference. Assign a "level number" to each *concept* so that for all paths in the network

[2]The conversion here extends our previous work in [DHW07] and [DH07b].

starting from a source node to a sink node, the "level" of a predecessor node is higher than the "level" of a successor node. Coded in hexadecimal numbers, Figure 5.1 shows a "linearization" of the *partial order* induced by the graph. Nodes numbered with 2, 4, 6, B, and D are ordered in $0x02 \leq 0x04 \leq 0x06 \leq 0x0B \leq 0x0D$.

(1) Introduce a new *role name* g (a super-role of any roles occurring in $\mathcal{T}_0$ and C), and for other role R, add two *role inclusions*: $R^a \sqsubseteq g$ and $R^b \sqsubseteq g$. Further, process the *role hierarchy* of *role inclusions* according to the step-1 of Section 5.2.

(2) Denote the lowest level (of the "modal reference" relation) as d.

(2.1) introduce new *concept names* L_0, and $L_1, L_2, ..., L_d$;

(2.2) add axioms $L_1 \sqsubseteq \forall g.L_2$, and $L_2 \sqsubseteq \forall g.L_3$, ..., and $L_{d-1} \sqsubseteq \forall g.L_d$;

(2.3) introduce an axiom $L_0 \sqsubseteq \forall g.L_1$, and replace C with $L_0 \sqcap \exists g.C$

(3) Assign a level number to each modal constraint as described above (and as exemplified in Figure 5.1). For each *modal constraint* at level $i > 0$, we need to define its role filler. Syntactically, each *modal constraint* has a unique *role filler* and vice versa. The definition of a role filler A_α is based on its propositional composition (of a *modal constraint* A_β at level i). In other words, A_α is composed by propositional connectives (e.g., $\sqcap$, $\sqcup$ and $\neg$ constructors) only; each of its sub-concepts is either a concept name (or a negated concept name) or a lower-level *modal constraint*. For example, let A_β be $\exists R_1.(C \sqcap \exists R_2^-.D)$, then its role filler A_α is $C \sqcap \exists R_2^-.D$ and its definition is $A_\alpha \equiv C \sqcap M_1$ where $M_1 \equiv \exists R_2^-.D$ is a second *modal constraint*.

(4) Let A_β be a *modal constraint* with an assigned level number i (for $i > 1$), and let A_α be its role R filler. Add more axioms as follows:

(4.1) $L_1 \sqcup L_2 \sqcup ... \sqcup L_{i-1} \sqsubseteq \neg A_\alpha \sqcup (\forall R^b.(A_\beta \equiv \exists^{\bowtie n-1} R^a.A_\alpha) \sqcap MD)$

(4.2) $L_1 \sqcup L_2 \sqcup ... \sqcup L_{i-1} \sqsubseteq A_\alpha \sqcup (\forall R^b.(A_\beta \equiv \exists^{\bowtie n} R^a.A_\alpha) \sqcap MD)$

(4.3) $MD \sqsubseteq \forall \widetilde{R_b}.(A_\beta \equiv \exists^{\bowtie n} R^a.A_\alpha)$

In the above, $A_\beta \equiv \exists^{\bowtie n} R^a.A_\alpha$ is a shorthand for $(\neg A_\beta \sqcup \exists^{\bowtie n} R^a.A_\alpha) \sqcap (A_\beta \sqcup \neg(\exists^{\bowtie n} R^a.A_\alpha))$. Note that $\exists^{\bowtie n} R^a.A_\alpha$ is the original definition of A_β; while $\exists^{\bowtie n-1} R^a.A_\alpha$ is its fine-tuned constraint (see an example in Section 6.3.1 of Chapter 6). The notation $\forall \widetilde{R}.X$ is a shorthand for $\forall S_1.X \sqcap \forall S_2.X \sqcap ... \sqcap \forall S_n.X$ where S_i $(i = 1, ..., n)$ is a role and $S_i \not\sqsubseteq R$ (according to the *role hierarchy*).

(5) Repeat step-3 and step-4 until the maximum level d has been reached.

Below, we give a small example to show elimination of *inverse roles* for an acyclic $\mathcal{ALCHQI}$ Tbox. The language used in this example is a strict subset of $\mathcal{ALCHQI}$ (i.e., $\mathcal{ALCI}$) for which a simple technique was given in Chapter 3. However, for an illustration purpose, let us just take it as if it is in $\mathcal{ALCHQI}$.

Example 1. Suppose we want to test satisfiability of $\exists R.(\forall R^-.B \sqcap \exists S.A)$ w.r.t. an acyclic simple Tbox $\{A \equiv \forall S^-.E \sqcap D, E \equiv \forall R^-.W\}$ of two *concept definitions* (see Section 1.3.1 in Chapter 1 about the notion of a simple Tbox).

(1) To convert the concept $\exists R.(\forall R^-.B \sqcap \exists S.A)$ that is subject to a *satisfiability test*, extra concept names (to weave a web of modal-reference relations) are introduced. We introduce new concept names C, M_1 and M_2 and use them as follows:

$$
\begin{aligned}
&C \equiv M_3\\
&M_3 \equiv \exists^{\geq 1} R.(M_1 \sqcap M_2)\\
&M_1 \equiv \exists^{\leq 0} R^-.\neg B\\
&M_2 \equiv \exists^{\geq 1} S.A
\end{aligned}
$$

(2) To convert the given acyclic Tbox, we further introduce new concept names (for *modal constraints* in the Tbox) as follows.

$$
\begin{aligned}
&A \equiv M_4 \sqcap D\\
&M_4 \equiv \exists^{\leq 0} S^-.\neg E\\
&E \equiv M_5\\
&M_5 \equiv \exists^{\leq 0} R^-.\neg W
\end{aligned}
$$

(3) Based on the previous two steps, we are able to assign a "level" for each concept name (corresponding to the five modal constraints) as follows: M_5 is at level-4, M_4 at level-3, M_1 and M_2 at level-2, while M_3 at level-1.

(4) According to the "levels" introduced above, we introduce new *concept inclusions* as follows.

$$
\begin{aligned}
&L_1 \sqcup L_2 \sqcup L_3 \sqsubseteq W \sqcup (\ \forall R.(M_5 \equiv (\exists^{\leq -1} R^-.\neg W))\ \sqcap\ MD_5)\\
&L_1 \sqcup L_2 \sqcup L_3 \sqsubseteq \neg W \sqcup (\ \forall R.(M_5 \equiv (\exists^{\leq 0} R^-.\neg W))\ \sqcap\ MD_5)\\
&MD_5 \sqsubseteq \forall S.(M_5 \equiv (\exists^{\leq 0} R^-.\neg W))\ \sqcap\ \forall S^-.(M_5 \equiv (\exists^{\leq 0} R^-.\neg W))\\
&\qquad\qquad \sqcap \forall R^-.(M_5 \equiv (\exists^{\leq 0} R^-.\neg W))
\end{aligned}
$$

$$
\begin{aligned}
&L_1 \sqcup L_2 \sqsubseteq \neg E \sqcup (\forall S.(M_4 \equiv (\exists^{\leq -1} S^-.\neg E))\ \sqcap\ MD_4)\\
&L_1 \sqcup L_2 \sqsubseteq E \sqcup (\forall S.(M_4 \equiv (\exists^{\leq 0} S^-.\neg E))\ \sqcap\ MD_4)\\
&MD_4 \sqsubseteq \forall R.(M_4 \equiv \exists^{\leq 0} S^-.\neg E) \sqcap \forall R^-.(M_4 \equiv \exists^{\leq 0} S^-.\neg E) \sqcap \forall S^-.(M_4 \equiv \exists^{\leq 0} S^-.\neg E)
\end{aligned}
$$

$$
\begin{aligned}
&L_1 \sqsubseteq A \sqcup (\forall S^-.(M_2 \equiv (\exists^{\geq 1} S.A))\ \sqcap\ MD_2)\\
&L_1 \sqsubseteq \neg A \sqcup (\forall S^-.(M_2 \equiv (\exists^{\geq 0} S.A))\ \sqcap\ MD_2)
\end{aligned}
$$

$MD_2 \sqsubseteq \forall R.(M_2 \equiv \exists^{\geq 1}S.A) \sqcap \forall R^-.(M_2 \equiv \exists^{\geq 1}S.A) \sqcap \forall S.(M_2 \equiv \exists^{\geq 1}S.A)$

$L_1 \sqsubseteq B \sqcup (\forall R^-.(M_1 \equiv (\exists^{\leq -1}R.\neg B)) \sqcap MD_1)$
$L_1 \sqsubseteq \neg B \sqcup (\forall R^-.(M_1 \equiv (\exists^{\leq 0}R.\neg B)) \sqcap MD_1)$
$MD_1 \sqsubseteq \forall S.(M_1 \equiv \exists^{\leq 0}R.\neg B) \sqcap \forall S^-.(M_1 \equiv \exists^{\leq 0}R.\neg B) \sqcap \forall R.(M_1 \equiv \exists^{\leq 0}R.\neg B)$

(5) Introducing a new role g which is a super role of all other roles

$S^b \sqsubseteq g$
$S^a \sqsubseteq g$
$R^a \sqsubseteq g$
$R^b \sqsubseteq g$

The converted problem is reformulated as a *satisfiability testing* of the concept $L_0 \sqcap \exists g.C$ w.r.t. an acyclic Tbox consisting of following concept definitions/inclusions.

$C \equiv M_3$
$M_3 \equiv \exists^{\geq 1}R^a.(M_1 \sqcap M_2)$
$A \equiv M_4 \sqcap D$
$E \equiv M_5$

$L_0 \sqsubseteq \forall g.L_1$
$L_1 \sqsubseteq \forall g.L_2$
$L_2 \sqsubseteq \forall g.L_3$

$L_1 \sqcup L_2 \sqcup L_3 \sqsubseteq W \sqcup (\forall R^a.(M_5 \equiv (\exists^{\leq -1}R^b.\neg W)) \sqcap MD_5)$
$L_1 \sqcup L_2 \sqcup L_3 \sqsubseteq \neg W \sqcup (\forall R^a.(M_5 \equiv (\exists^{\leq 0}R^b.\neg W)) \sqcap MD_5)$
$MD_5 \sqsubseteq \forall S^a.(M_5 \equiv (\exists^{\leq 0}R^b.\neg W)) \sqcap \forall S^b.(M_5 \equiv (\exists^{\leq 0}R^b.\neg W))$
$\sqcap \forall R^b.(M_5 \equiv (\exists^{\leq 0}R^b.\neg W))$

$L_1 \sqcup L_2 \sqsubseteq \neg E \sqcup (\forall S^a.(M_4 \equiv (\exists^{\leq -1}S^b.\neg E)) \sqcap MD_4)$
$L_1 \sqcup L_2 \sqsubseteq E \sqcup (\forall S^a.(M_4 \equiv (\exists^{\leq 0}S^b.\neg E)) \sqcap MD_4)$
$MD_4 \sqsubseteq \forall R^a.(M_4 \equiv (\exists^{\leq 0}S^b.\neg E)) \sqcap \forall R^b.(M_4 \equiv (\exists^{\leq 0}S^b.\neg E)) \sqcap \forall S^b.(M_4 \equiv (\exists^{\leq 0}S^b.\neg E))$

$L_1 \sqsubseteq A \sqcup (\forall S^b.(M_2 \equiv (\exists^{\geq 1}S^a.A)) \sqcap MD_2)$
$L_1 \sqsubseteq \neg A \sqcup (\forall S^b.(M_2 \equiv (\exists^{\geq 0}S^a.A)) \sqcap MD_2)$
$MD_2 \sqsubseteq \forall R^a.(M_2 \equiv (\exists^{\geq 1}S^a.A)) \sqcap \forall R^b.(M_2 \equiv (\exists^{\geq 1}S^a.A)) \sqcap \forall S^a.(M_2 \equiv (\exists^{\geq 1}S^a.A))$

$L_1 \sqsubseteq B \sqcup (\forall R^b.(M_1 \equiv (\exists^{\leq -1}R^a.\neg B)) \sqcap MD_1)$
$L_1 \sqsubseteq \neg B \sqcup (\forall R^b.(M_1 \equiv (\exists^{\leq 0}R^a.\neg B)) \sqcap MD_1)$
$MD_1 \sqsubseteq \forall S^a.(M_1 \equiv (\exists^{\leq 0}R^a.\neg B)) \sqcap \forall S^b.(M_1 \equiv (\exists^{\leq 0}R^a.\neg B)) \sqcap \forall R^a.(M_1 \equiv (\exists^{\leq 0}R^a.\neg B))$

In summary, the above conversion relies on a hard encoding of "level" into new concept names and a way of ensuring that in newly introduced axioms of the converted Tbox an original i-th level *modal constraint* will not appear in any level lower than the i-th level. In contrast to Chapter 3 and Section 5.2, the "encoding technique" presented in this section introduces high non-determinism and therefore would not be practical for implementations (in its current presented form).

5.4 Summary

This chapter indirectly gives one conversion from $\mathcal{SHI}$ to $\mathcal{ALC}$ (by eliminating the inverse roles and the role hierarchy in $\mathcal{SHI}$). The conversion from $\mathcal{SHI}$ to $\mathcal{ALC}$ is straightforward by composing the two conversions shown in Sections 5.1 and 5.2 in sequence (See Figure 5.2).

$$\begin{tikzcd}
\mathcal{SHOI} \Longrightarrow \mathcal{SHO} \arrow[d, bend right] & & \mathcal{SHQ} \Longrightarrow \mathcal{ALCQ} \arrow[d, bend left] \\
\mathcal{SHI} \Longrightarrow \mathcal{SH} \arrow[dr, Rightarrow] & & \mathcal{SH} \Longrightarrow \mathcal{ALC} \arrow[dl, Rightarrow] \\
& \mathcal{SHI} \Longrightarrow \mathcal{ALC} &
\end{tikzcd}$$

Figure 5.2: A Conversion of $\mathcal{SHI}$ to $\mathcal{ALC}$

This chapter also showed a conversion of an $\mathcal{ALCHQI}$ acyclic Tbox to an $\mathcal{ALCHQ}$ acyclic Tbox. It relies on introducing "explicit level" concepts which are implicitly implied in the original Tbox. This conversion heavily uses non-determinism and is not intended for practical reasoning. This conversion is not applicable also when *general concept inclusions* are present.

In the next chapter, we take *general concept inclusions* into consideration and (by using a *global tableaux caching technique* and the *integer linear programming* technique) will show a worst-case ExpTime decision procedure for $\mathcal{SHIQ}$ — a description logic more expressive than $\mathcal{ALCHQI}$.

Chapter 6

A Tableaux Procedure for $\mathcal{SHIQ}$

The use of the *atomic decomposition* principle and the *mathematical programming* method is typical of the so called *algebraic methods* [CL94, OK99, HM01a]. The *atomic decomposition* introduced in [CL94] works on role fillers, while in [OK99] another style of *atomic decomposition* of "roles" was introduced. Soundness and completeness proofs on decompositions were given therein. In [HM01a] the algebraic method of [OK99] was combined with a tableaux calculus for $\mathcal{SHQ}$.

This chapter shows that the *algebraic method* [OK99, HM01a] leads to a worst-case optimal tableaux algorithm for the *concept satisfiability* problem in $\mathcal{SHIQ}$.

6.1 Two Questions

6.1.1 Big Number Values

Will a *concept satisfiability* test of $\exists^{\geq 3}R.D_1 \sqcap \exists^{\leq 3}R.D_2$ be easier than that of $\exists^{\geq 311}R.D_1 \sqcap \exists^{\leq 311}R.D_2$? This is the numerical scalability question for *number restrictions.*

In practice, the answer to this question varies depending on the type of DL reasoners used. For DL reasoners designed to support only very small numbers, very likely the answer is yes. For *algebraic reasoners* which take advantage of *integer linear programming* techniques, the answer is no. [HM01a] pointed out that even when numbers are increased by a factor of 100, the increased reasoning time is negligible[1]. Therefore, the *algebraic method* scales better than other available methods for number restrictions. Related to this scalability, this chapter will present that the algebraic method is worst

[1]There is no comparable report from non-algebraic DL reasoners.

case optimal in the strong sense (of binary coding of numbers).

6.1.2 Soundness of Tableaux Caching

Let D_1 and D_2 be two satisfiable concepts w.r.t. one *general concept inclusion* asserting $D_1 \sqsubseteq D_2$. As shown in Figure 6.1, $\exists^{\geq 3}R.D_1 \sqcap \exists^{\leq 3}R.D_2$ is satisfiable.

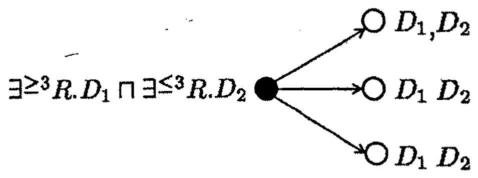


Figure 6.1: $\exists^{\geq 3}R.D_1 \sqcap \exists^{\leq 3}R.D_2$ has a model w.r.t. $D_1 \sqsubseteq D_2$

The dark node in the figure represents the given problem, the 3 grey nodes (R-neighbors to the predecessor node) represent three identical subproblems. It is obvious to see that the given problem $\exists^{\geq 3}R.D_1 \sqcap \exists^{\leq 3}R.D_2$ is satisfiable.

Is $\neg D_1 \sqcap D_2 \sqcap \exists R^-.\exists^{\geq 3}R.D_1 \sqcap \forall R^-.\exists^{\leq 3}R.D_2$ satisfiable or not? In Figure 6.2, the dark node represents a subproblem $\exists^{\geq 3}R.D_1 \sqcap \exists^{\leq 3}R.D_2$, which has already been tested in Figure 6.1.

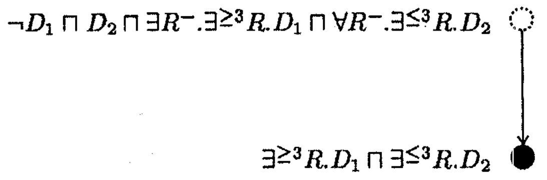

$\neg D_1 \sqcap D_2 \sqcap \exists R^-.\exists^{\geq 3}R.D_1 \sqcap \forall R^-.\exists^{\leq 3}R.D_2$
$\exists^{\geq 3}R.D_1 \sqcap \exists^{\leq 3}R.D_2$

Figure 6.2: A repeated subproblem $\exists^{\geq 3}R.D_1 \sqcap \exists^{\leq 3}R.D_2$

Unfortunately for the "repeated subproblem" $\exists^{\geq 3}R.D_1 \sqcap \exists^{\leq 3}R.D_2$, it is not "safe" to "reuse" the previous tested result. It would be incorrect if we directly "copy" the satisfiability status of the dark node from Figure 6.1 and reuse it here. The correct answer is that the "repeated subproblem" is not satisfiable in the new case, and accordingly $\neg D_1 \sqcap D_2 \sqcap \exists R^-.\exists^{\geq 3}R.D_1 \sqcap \forall R^-.\exists^{\leq 3}R.D_2$ is not satisfiable. This chapter will also present a *global tableaux caching* that is sound for $\mathcal{SHIQ}$.

6.2 Syntax and Semantics

We stipulate *concept expressions* are in the Negation Normal Form (NNF) (see Section 2.1 of Chapter 2). For a concept C (in NNF), we denote the NNF of $\neg C$ by $\widetilde{C}$.

The notion of the inverse of a role R is defined as follows: if R is a role name, then $\texttt{Inv}(R) = R^-$; if $R = S^-$ for a role name S, then $\texttt{Inv}(R) = S$.

To facilitate the use of the *algebraic method* to *modal constraints* on transitive roles, we perform a transformation[2] as follows: for *universal restrictions* like $\forall S.C$ where S is transitive, replace it with a new name A, and add three axioms: (1) $A \sqsubseteq \forall S.(C \sqcap A)$; (2) $\neg A \sqsubseteq \exists S.\neg C$; (3) $\exists(\texttt{Inv}(S)).A \sqsubseteq A \sqcap C$. Recursively process C and $\neg C$. Similarly process *existential restrictions* on transitive roles in a dual way (see Section 5.1 of Chapter 5).

It is known that *universal restrictions* and *existential restrictions* can be expressed in *qualified number restrictions* by using $\exists R.C \Rightarrow \exists^{\geq 1} R.C$ and $\forall R.C \Rightarrow \exists^{\leq 0} R.\neg C$. Therefore, it is justified to assume[3] that only *qualified number restrictions* (also called *modal constraints* hereafter) of the form $\exists^{\geq n} R.C$ and $\exists^{\leq n} R.C$ occur in the language of the DL $\mathcal{SHIQ}$. We use $\bowtie$ as a placeholder for either $\exists^{\leq}$ or $\exists^{\geq}$, so an expression of the form $\exists^{\bowtie n} R.C$ might stand for either $\exists^{\leq n} R.C$ or $\exists^{\geq n} R.C$.

Definition 1. (Syntax) We use A for *atomic concept*, C and D for arbitrary concepts, R (and R^-) for a *role*, and n for a non-negative integer. Concept formulae in $\mathcal{SHIQ}$ are formed according the following grammar:

$$C, D := \top | A | \neg C | C \sqcap D | C \sqcup D | \exists^{\leq n} R.C | \exists^{\geq n} R.C$$

Definition 2. (Semantics) An interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a set $\Delta^{\mathcal{I}}$ (the domain) and an interpretation function $\cdot^{\mathcal{I}}$. The interpretation function maps each concept name C to a subset $C^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, each role name R to a subset $R^{\mathcal{I}}$ of $\Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Let the symbols C, D be concept formulae, R and R^- be two roles in converse. The interpretation function is inductively defined as follows:

[2]This transformation is not new and can be found in the literature (e.g. in [GN07, MSH07]).

[3]Clearly the two forms have only a syntactical difference. Although recently proved in [KSZ07] that "number restrictions on *transitive roles*" is ExpTime decidable, it was shown that "number restrictions on a role having *transitive sub-roles* (w.r.t. a role hierarchy)" is undecidable [HST99a]. Consequently in the DL $\mathcal{SHOIQ}$ and its fragments, a role to have number restrictions is stipulated to be non-transitive and having no transitive sub-roles (a.k.a. a simple role). It should be clear that the undecidability result of [HST99a] is not violated when we represent $\forall S.C$ in $\exists^{\leq 0} S.\neg C$ and $\exists S.C$ in $\exists^{\geq 1} S.C$ for a transitive role S. Moreover, we are not considering the latest result in [KSZ07] because it would make (slightly) different semantics for $\mathcal{SHIQ}$.

$\top^{\mathcal{I}} := \Delta^{\mathcal{I}} \quad (\neg C)^{\mathcal{I}} := \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}} \quad (C \sqcap D)^{\mathcal{I}} := C^{\mathcal{I}} \cap D^{\mathcal{I}} \quad (C \sqcup D)^{\mathcal{I}} := C^{\mathcal{I}} \cup D^{\mathcal{I}}$

$(\exists^{\leq n} R.C)^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \leq n\}$

$(\exists^{\geq n} R.C)^{\mathcal{I}} := \{x \in \Delta^{\mathcal{I}} \mid \sharp\{y \in \Delta^{\mathcal{I}} : (x,y) \in R^{\mathcal{I}} \text{ and } y \in C^{\mathcal{I}}\} \geq n\}$

and additionally, $(x,y) \in R^{\mathcal{I}}$ if and only if $(y,x) \in (R^{-})^{\mathcal{I}}$. For a transitive role S, if $(x,y) \in S^{\mathcal{I}}$ and $(y,z) \in S^{\mathcal{I}}$ then $(x,z) \in S^{\mathcal{I}}$. For a *role inclusion* $R_1 \sqsubseteq R_2$, it satisfies $R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$.

A tableau structure is a "completed" tree (or graph in general) representing binary relations about concept formulae [BS01]. Below are notions commonly employed in discussing "elements" and their relations in a tableau structure.

Definition 3. (Successor, Predecessor, Neighbor) In a tableau structure, if $R \in \mathcal{L}(\langle x, y \rangle)$, then y is called an R-successor of x, and x is called an R^{-}-predecessor of y. The R-neighbors of a node x is a set of R-predecessors or R-successors of x.

Definition 4. (Specific Neighbor) For a role R and a node x in a tableau structure G, we define the set of x's R-neighbors with C in their labels, $R^G(x, C)$: $R^G(x,C) := \{y \mid y \text{ is an } R\text{-neighbor of } x \text{ and } C \in \mathcal{L}(y)\}$. In this thesis whenever G is obvious from context, we simply write $R(x, C)$.

Definition 5. (Closure) The definition of *clos*(.,.) in the DL $\mathcal{SHIQ}$ is as follows. For an $\mathcal{SHIQ}$ Tbox $\mathcal{T}$ and a concept formula C, we define $clos(\mathcal{T}, C)$ as the smallest set X that contains C and satisfies the following properties:

- for every subformula D, $nnf(D) \in X$ and $nnf(\neg D) \in X$;
- for every $\top \sqsubseteq D \in \mathcal{T}$, $nnf(D) \in X$ and $nnf(\neg D) \in X$;
- X is closed under subformulae and the application of $nnf(.)$ and $\neg$.

Definition 6. (Successor Constraints) The set of concepts (i.e., role fillers) occurring in *number restrictions*, which is to be considered at successor and predecessor nodes, is defined by $\mathsf{succ}(\mathcal{T}, C) = \{D \mid (\exists^{\bowtie n} R.D) \in clos(\mathcal{T}, C) \text{ where } R \text{ is any role}\}$.

Definition 7. (Locally Consistent) A subset $\phi \subseteq clos(\mathcal{T}, C)$ is locally consistent if and only if the following hold

- for every $D \in \text{succ}(\mathcal{T}, C)$, it is that $\phi \cap \{D, \widetilde{D}\} \neq \emptyset$ and $\{D, \widetilde{D}\} \subsetneq \phi$,
- for every $\top \sqsubseteq D \in \mathcal{T}$, it is that $nnf(D) \in \phi$,
- if $C_1 \sqcap C_2 \in \phi$ then $\{C_1, C_2\} \subseteq \phi$, and
- if $C_1 \sqcup C_2 \in \phi$ then $\phi \cap \{C_1, C_2\} \neq \emptyset$.

Below we assume $\mathcal{T}$ contains only one *general concept inclusion* of the form $\top \sqsubseteq G$, i.e., $\mathcal{T} = \{\top \sqsubseteq G\}$. For convenience, we abuse the definition $clos(\mathcal{T}, .)$ and $succ(\mathcal{T}, .)$ by parameterizing their second parameter for concept sets too. We overload $clos(., .)$ and use the notion of $clos(\mathcal{H}, .)$ on a role hierarchy $\mathcal{H}$.

6.3 The Algebraic Method for $\mathcal{SHIQ}$

The *algebraic method* first performs the *atomic decomposition* on a group of *modal constraints* and thus generates subproblems to solve. What is special about *atomic decomposition* is that an *integer linear program* will also be constructed thereof. This is achieved more or less by first arranging *modal constraints* in some preferred order, according to which an integer vector $\vec{\mathbf{b}}$ of the numbers (occurring in the *modal constraints*) will be formed. A coefficient matrix A will also be formed according to that order, and its columns will be adjusted later depending on the satisfiability status of the subproblems. The feasibility of the *integer program* $A \cdot \vec{v} = b$ (modulo slack and surplus variables) indicates the satisfiability of the group of *modal constraints.*

Several modifications to the original *algebraic method* are introduced below for description logics having both *inverse roles* and *qualified number restrictions.* These include *fine-tuning* of modal constraints, an extended *atomic decomposition*, and an extended *tableau structure.* As a reminder, in Section 6.6.3 an algorithm adapted from [Pap81] for testing the feasibility of special *linear integer inequalities* will be shown, which is critical to the complexity proof in the sense of binary coding of numbers.

6.3.1 Fine-tuning on Modal Constraints

For the satisfiability of *modal constraints* in $\mathcal{SHIQ}$, there is a step to take the "neighborhood information" into account before performing the *atomic decomposition.* Let us consider such a situation in which partial "neighborhood" information is given. For example in Figure 6.3, a, b and c are distinct individuals (i.e., $a \doteqdot b$, $b \doteqdot c$, $c \doteqdot a$), there is one R-edge from a to c and one R^--edge from b to a respectively, and $\mathcal{L}(b) = \{A_1, A_2, \neg A_3, \neg A_4\}$ and $\mathcal{L}(c) = \{\neg A_1, A_2, A_3, \neg A_4\}$. Now consider to satisfy $\{\exists^{\geq 4} R.A_1, \exists^{\leq 3} R.A_2\}$ at a, which requires at least 4 R-neighbors of type A_1 (the *specific neighbor* $\sharp \|R(a, A_1)\| \geq 4$) and at most 3 R-neighbors of type A_2 (the *specific neighbor* $\sharp\|R(a, A_2)\| \leq 3$).

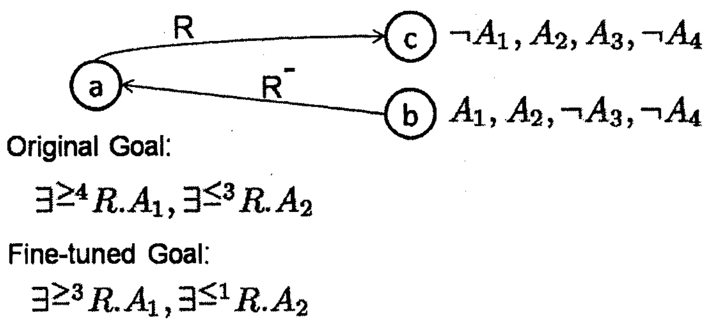

R
R⁻
c
a
b
$\neg A_1, A_2, A_3, \neg A_4$
$A_1, A_2, \neg A_3, \neg A_4$
Original Goal:
$\exists^{\geq 4} R.A_1, \exists^{\leq 3} R.A_2$
Fine-tuned Goal:
$\exists^{\geq 3} R.A_1, \exists^{\leq 1} R.A_2$

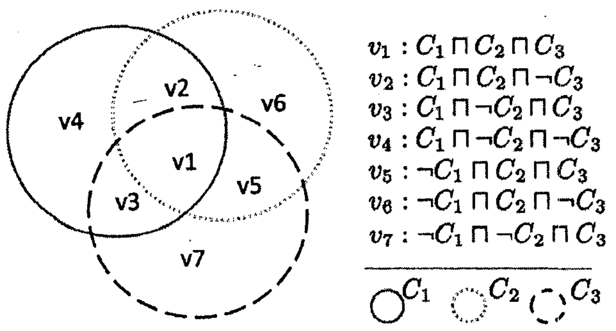
Figure 6.3: Fine-Tune of Modal Constraints

What is the "told" (or explicit) information that should be taken into consideration? As Figure 6.3 shows, $\mathcal{L}(b)$ and $\mathcal{L}(c)$ respectively are supersets of an *atomic partition* of the modal constraints in question. In this example, because b already counts as one R-neighbor of type A_1, a only needs 3 more R-successors of type A_1. It is also clear that a is only allowed to have one more R-successor of type A_2, since the two distinct individuals b and c already count as two different R-neighbors of type A_2 for a. Therefore, the subproblem to explore at node a is adjusted to be "to satisfy $\exists^{\leq 1} R.A_2$ and $\exists^{\geq 3} R.A_1$".

The adjustment of number values for modal constraints in a label is called the **fine-tuning of modal constraints**. We denote the fine-tuned modal constraints by a designated label $\mathcal{F}$. For the above example, the *fine-tuned label* for a (according to its neighborhood information) is $\mathcal{F}(a) = \{\exists^{\geq 3} R.A_1, \exists^{\leq 1} R.A_2\}$.

6.3.2 Atomic Decomposition and Integer Linear Program

Typical of the *algebraic method* is a principle called *atomic decomposition.* The *atomic decomposition* is usually depicted in Venn diagrams [OK99, HM01a]. For example, Figure 6.4 shows a decomposition on C_1, C_2, C_3.



$v_1 : C_1 \sqcap C_2 \sqcap C_3$
$v_2 : C_1 \sqcap C_2 \sqcap \neg C_3$
$v_3 : C_1 \sqcap \neg C_2 \sqcap C_3$
$v_4 : C_1 \sqcap \neg C_2 \sqcap \neg C_3$
$v_5 : \neg C_1 \sqcap C_2 \sqcap C_3$
$v_6 : \neg C_1 \sqcap C_2 \sqcap \neg C_3$
$v_7 : \neg C_1 \sqcap \neg C_2 \sqcap C_3$

Figure 6.4: An Atomic Decomposition on 3 Objects

For more objects, it is not easy to show its decomposition pictorially, e.g., consider the complex diagram known as the Edwards-Venn-diagram below.

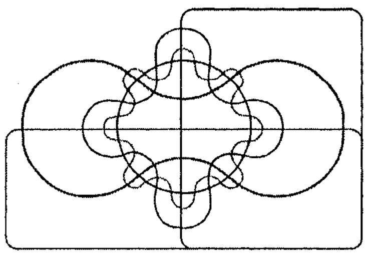


Figure 6.5: An Atomic Decomposition on 6 Objects [Edw04]

The problem of solving a system of linear inequalities dates back at least as far as Fourier [Vas83, Sch86]. The system of linear inequalities is called a *program.* Many practical problems in *operational research* can be expressed as integer linear programming (ILP) problems. The algebraic method works by solving ILP problems as sub-problems.

Example 1. Let us consider an example. Let x be a tableaux node, and its current label $\mathcal{L}(x) =$

$\{\exists^{\leq 3}R.C_1, \exists^{\geq 2}R.C_2, \exists^{\geq 4}R.C_3\}$. The *atomic decomposition* on role-fillers of the modal constraints gives 7 pairwise disjoint combinations, see Figure 6.4.

The decomposition results in a coefficient matrix $A = |w_{i,j}| = (\vec{w}_1, \vec{w}_2, ..., \vec{w}_7)$. The variable v_i is associated with the i-th *atomic decomposition.* The resulting *integer inequalities* $(\vec{w}_1, \vec{w}_2, ..., \vec{w}_7) \cdot \vec{v} \gtreqless \vec{b}$ is as follows:

$$\begin{array}{lllllllll} w_1 \cdot v_1 + & w_2 \cdot v_2 + & w_3 \cdot v_3 + & w_4 \cdot v_4 + & 0 \cdot v_5 + & 0 \cdot v_6 + & 0 \cdot v_7 & \leq 3 \\ w_1 \cdot v_1 + & w_2 \cdot v_2 + & 0 \cdot v_3 + & 0 \cdot v_4 + & w_5 \cdot v_5 + & w_6 \cdot v_6 + & 0 \cdot v_7 & \geq 2 \\ w_1 \cdot v_1 + & 0 \cdot v_2 + & w_3 \cdot v_3 + & 0 \cdot v_4 + & w_5 \cdot v_5 + & 0 \cdot v_6 + & w_7 \cdot v_7 & \geq 4 \end{array}$$

Each w_i takes a value of 0 (in the backtrack phase of the recursive process) if its "atomic decomposition" (i.e., the i-th decomposition) turns out to be unsatisfiable and takes a value of 1 otherwise.

By introducing slack variables and surplus variables, the vector $\vec{v}$ of variables is augmented to $\vec{v}'$ and the integer linear inequalities above can be expressed as $[A|B] \cdot \vec{v}' = \vec{b}$, where B is a $m \times m$ matrix. For example, $\vec{v}' = (v_1, v_2, v_3, v_4, v_5, v_6, v_7, s_1, s_2, s_3)^{\mathrm{T}}$, and A_E has three more columns $\vec{A_E}[8] = (1,0,0)^{\mathrm{T}}$, $\vec{A_E}[9] = (0,-1,0)^{\mathrm{T}}$, $\vec{A_E}[10] = (0,0,-1)^{\mathrm{T}}$. The extended *integer program* is usually abbreviated as $A_E \cdot \vec{v}_E = \vec{b}$.

To ease the presentation of a $\mathcal{SHIQ}$ algorithm in Section 6.5, we introduce a function $id(.)$ mapping a decomposition p to its predefined order. When writing $id(p) = i$, it means p is the i-th decomposition. Similarly, the *column vector* of the *coefficient matrix* A relating to the decomposition p is denoted by $A[id(p)]$; the variable of the *variable vector* $\vec{v}$ corresponding to the decomposition p is denoted by $\vec{v}[id(p)]$; and the number in the *constant vector* $\vec{b}$ corresponding to the decomposition p is denoted by $\vec{b}[id(p)]$.

An *integer program* having no integer solution is called **non-feasible**, otherwise **feasible**. If the i-th decomposition is unsatisfiable, then $\vec{w}_i$ will be set to $\vec{0}$. If the satisfiability status of a sufficient number of decompositions is known, the feasibility of an *integer linear program* can be decided by *integer linear programming* methods [Vas83, Sch86].

Notice that for a number of m *modal constraints* the *atomic decomposition* generates a *variable vector* of size $2^{O(m)}$ and a matrix A of size $m \times 2^{O(m)}$. It is interesting to ask whether an *integer* *linear program* of such a form could be solved in an exponential time to the total size of the m constraints (considering a binary coding of the *constant vector* $\vec{\mathbf{b}}$). According to [Pap81], the answer is yes (even if the element a of matrix A is from the whole integer domain, i.e., allowing $|a| \geq 1$). A proof will be given in Section 6.6.3.

6.3.3 A Concatenated Two-phase Decomposition

The *algebraic method* in [OK99, HM01a] works in a *role hierarchy* and successfully handles $\mathcal{SHQ}$. In [CL94], the *algebraic method* decomposes solely on "concepts" instead. Here, we combine the two styles in a sequence of two phases. In the first phase, an atomic decomposition is performed on roles; in the second phase, an atomic decomposition is performed on role fillers.

The upper portion of an element of the resulting set $\mathcal{P}$ is called a **role atom**; the lower portion of an element is called a **concept atom**. Clearly, elements are pair-wise disjoint, i.e., for $x, y \in \mathcal{P}$ and $x \neq y$, $x \sqcap y \sqsubseteq \bot$.

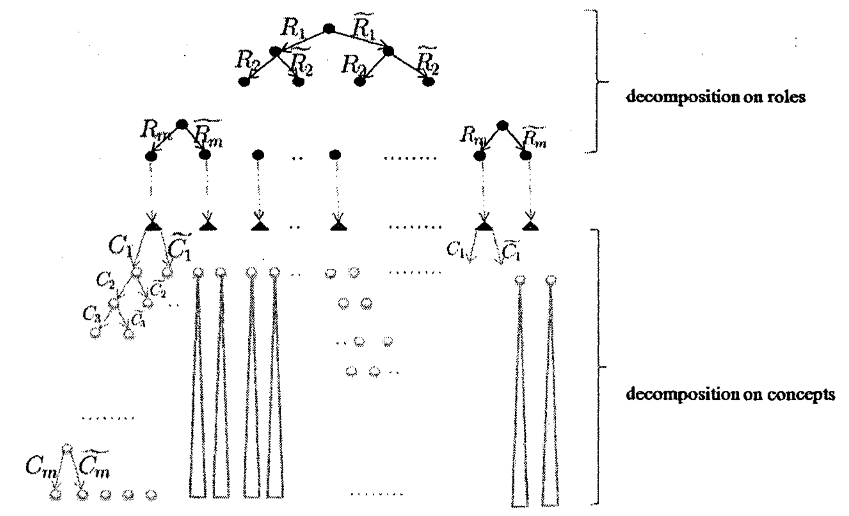


Figure 6.6: A Two-phase Concatenated Decomposition on Roles and Role Fillers

To ease the presentation of a $\mathcal{SHIQ}$ algorithm in Section 6.5, we introduce two functions *roles*(.) and *concepts*(.). The first function returns the roles (upper portion) for a given *decomposition*, while the second one returns the decomposed concepts (lower portion) of a given *decomposition*.

choose-rule:	if	$\exists^{\bowtie n}R.C \in clos(\mathcal{T}, C)$ is not in $\mathcal{C}(x)$,
	then	choose some $E \in \{C, \widetilde{C}\}$ s.t. $\mathcal{C}(x) = \mathcal{C}(x) \cup \{E\}$
$\sqcap$-rule:	if	1. $C_1 \sqcap C_2 \in \mathcal{C}(x)$, and
		2. $\{C_1, C_2\} \cap \mathcal{C}(x) \neq \{C_1, C_2\}$
	then	$\mathcal{C}(x) = \mathcal{C}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule:	if	1. $C_1 \sqcup C_2 \in \mathcal{C}(x)$, and
		2. $\{C_1, C_2\} \cap \mathcal{C}(x) = \emptyset$
	then	$\mathcal{C}(x) = \mathcal{C}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$
$\bowtie$-rule:	if	x is not sat-cached
	then	perform the concatenated atomic decomposition, and generate upto 2^n successors

Table 6.1: The tableau expansion rules for $\mathcal{SHIQ}$.

6.3.4 Tableaux Structure

The *tableaux structure* usually used is a *labeled tree*; and the label is either a set of *concepts* or a set of *roles*. Each tableaux node is labeled with a set of *concepts*; each tableaux edge is labeled with a set of *roles*.

For $\mathcal{SHIQ}$, the *tableaux structure* is also considered a *labeled tree*. However, additional labels are provided as follows.

- each tableaux node is labeled with 0 or 1 integer linear program $A_E \cdot \vec{\mathbf{v}}_E = \vec{\mathbf{b}}$;
- each tableaux edge to the i-th decomposition is associated with a tuple $\langle A_i, v_i \rangle$ of two variables, where A_i is the i-th column vector of the 0/1 matrix A, and v_i stores the solution value for v_i of the integer linear program.

If i-th decomposition $\mathcal{P}[i]$ is unsatisfiable, then the vector $\overrightarrow{\mathbf{A}}_E\,[i]$ will be set to $\vec{\mathbf{0}}$, a vector of all 0s; otherwise, the vector $\overrightarrow{\mathbf{A}_E}[i]$ remains unchanged. If the integer program is *feasible*, one solution to $A_E \cdot \vec{\mathbf{v}}_E = \vec{\mathbf{b}}$ will be recorded at the second parts of the tuples $\langle A_i, v_i \rangle$. A_i is a shorthand for $\overrightarrow{\mathbf{A}}[i]$, and v_i is a shorthand for $\overrightarrow{\mathbf{v}}\,[i]$.

6.4 $\mathcal{SHIQ}$ Tableaux

6.4.1 Tableau Expansion Rules

Table 6.1 shows the tableau expansion rules that explicitly process the *modal constraints* and *propositional constraints* at a tableau node x by manipulating its initial label $\mathcal{L}(x)$, its local consistent label $\mathcal{C}(x)$ and the fine-tuned $\mathcal{F}(x)$. The choose-rule is nondeterministic and implements the first condition of the local consistency. Note that the choose-rule is different from that of [BS01], but is also highly non-deterministic. The $\sqcap$-rule and the $\sqcup$-rule are as usual. The $\bowtie$-rule implements the *atomic decomposition* principle.

6.4.2 Inconsistency Propagation Rules

$\perp$-0-rule:		$\{\neg\top\} \in \perp$-set.
$\perp$-1-rule:		$\{C, \neg C\} \in \perp$-set.
$\perp$-2-rule:	if	(1) $\alpha \in \perp$-set, and
		(2) $\alpha \subseteq \beta$
	then	$\beta \in \perp$-set.
$\perp$-3-rule:	if	(1) $\alpha \cup \{C\} \in \perp$-set, and
		(2) $\alpha \cup \{D\} \in \perp$-set
	then	$\alpha \cup \{C \sqcup D\} \in \perp$-set.
$\perp$-4-rule:	if	(1) the set of fine-tuned modal constraints is $\mathcal{M}$, and
		(2) $\mathcal{M}$'s atom decompositions is $\mathcal{P}$, and
		(3) $\mathcal{P}$'s integer linear program $A_E \cdot \vec{\mathbf{x}}_E = \vec{\mathbf{b}}$ is *nonfeasible*
	then	$\mathcal{M} \in \perp$-set.

Table 6.2: The inconsistency propagation rule for $\mathcal{SHIQ}$.

Table 6.2 shows the *inconsistency propagation rules* for $\perp$-set. The *primitive clashes* [BCM+03] in $\mathcal{SHIQ}$ include any superset of $\{\neg\top\}$, $\{C, \neg C\}$ (as implied by $\perp$-0, $\perp$-1 and $\perp$-2 here). Rule $\perp$-3 is same as Chapter 4. Here to consider *number restrictions*, the patterns for *clash triggers* are necessarily more complex. For example, $\{\exists^{\leq 1}R.A, \exists^{\leq 2}.R.\neg A, \exists^{\geq 4}R.B\}$ is inconsistent. Also, for example the consistency of $\{\exists R.A, \exists R.B, \exists R.C, \exists R.D, \exists^{\leq 3}R.\top\}$ depends on a successful partition that divides $\{A, B, C, D\}$ into at most 3 parts without any conflict. The new rule $\perp$-4 is for a set of *modal constraints* and generalizes simple clash patterns such as $\{\exists^{\leq -1}R.C\}$ (possibly resulting from a fine-tuning of the *modal constraint* $\{\exists^{\leq 0}R.C\}$), and $\{\exists^{\leq n}R.C,\ \exists^{\geq n+1}R.C\}$, and more complex ones as well. Inconsistency inference is performed on demand during the backtracking phase.

6.4.3 Blocking

It is known from the literature that lots of efforts have been spent in investigating "optimized blocking" for DLs having both *inverse roles* and *qualified number restrictions*, e.g. $\mathcal{SHIQ}$. The "blocking" method widely recognized nowadays is the "**dynamic double blocking**" technique and its optimized version [HS02] as shown in Figure 6.7.

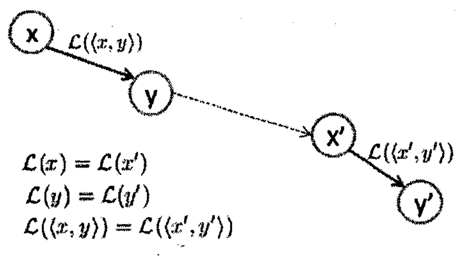

x
$\mathcal{L}(\langle x,y\rangle)$
y
x'
$\mathcal{L}(\langle x',y'\rangle)$
y'
$\mathcal{L}(x) = \mathcal{L}(x')$
$\mathcal{L}(y) = \mathcal{L}(y')$
$\mathcal{L}(\langle x,y\rangle) = \mathcal{L}(\langle x',y'\rangle)$

Figure 6.7: Dynamic Double Blocking

The dashed line in Figure 6.7 indicates that node x' is a descendant of node y. The blocking condition is specified in a pairwise manner: for node y' to be blocked, it is sufficient to find an ancestor node x and x's direct successor node y such that $\mathcal{L}(x)$ equals to $\mathcal{L}(x')$, and $\mathcal{L}(y)$ equals to $\mathcal{L}(y')$; moreover the labels of the two edges are also equal. As commented in [HS02], the conditions required to trigger a "blocking" are more complex than in earlier tableaux algorithms. Below we introduce a new style of block conditions that generalizes the idea of the *dynamic double blocking.*

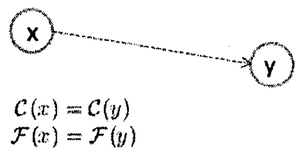

x
y
$\mathcal{C}(x) = \mathcal{C}(y)$
$\mathcal{F}(x) = \mathcal{F}(y)$

Figure 6.8: New Blocking

The new one supports simplified pre-conditions for the blocking and potentially leads to an easy implementation. In the following paragraphs about "blocking", both blocking nodes and blocked nodes are required to be "propositionally completed", i.e., the propositional rules (incl. the ⊓-rule and the ⊔-rule) are not applicable to them (a.k.a. being locally consistent). Those tableau nodes that serve as "blocking" nodes are also called *witness* nodes. A node y is blocked if none of its ancestors is blocked, and there is a *witness* node x such that

- $\mathcal{C}(x) = \mathcal{C}(y)$, and

- $\mathcal{F}(x) = \mathcal{F}(y)$

where $\mathcal{F}(x)$ is the fine-tuned label of $\mathcal{C}(x)$ and $\mathcal{C}(x)$ is one *locally consistent set* of $\mathcal{L}(x)$. When the two equalities hold, we say x blocks y. This blocking is static and is based on *label equality* (in the sense of "set equality"). For a node x, obviously $\mathcal{C}(x)$ and $\mathcal{F}(x)$ should not be known to be inconsistent. The dashed line in Figure 6.8 from node x to node y simply means x has been locally saturated before y has been done.

Note that the fine-tuned modal constraints might not necessarily belong to the $clos(\mathcal{T}, \mathcal{C})$. Nonetheless, the number of fine-tuned labels are of the same order of the original labels. In a tree-like tableau structure, the number of tuples (i.e., combinations) of a label and its fine-tuned one $\langle \mathcal{F}, \mathcal{C} \rangle$ is $2^{O(n)}$ where $n = \|\mathcal{T}\| + \|\mathcal{C}\|$.

6.5 $\mathcal{SHIQ}$ Algorithm

The algorithm starts building a tableau structure (TS) from a root node labeled with the concept subject to the satisfiability test. The decision procedure uses a *restart strategy*. It switches to explore a different TS if the current one cannot be saturated without causing local inconsistency[4]. Since it might restart, there could be many runs of the decision procedure. In each run, the decision procedure takes a depth-first traversal to construct a tableaux tree.

There are two major data structures: `Nogood` (for unsat *caching* across different tableau structures) and `Witness` (for static *blocking* of tableau nodes within one tableau structure). The two data structures are described as follows:

- `Nogood`: `Nogood` permanently holds labels like $\mathcal{C}(x)$ or $\mathcal{F}(x)$ for inconsistent concept sets encountered. Recall that $\mathcal{F}(x)$ is the fine-tuned version of $\mathcal{C}(x)$; while $\mathcal{C}(x)$ is one of the *locally consistent sets* of $\mathcal{L}(x)$.

- `Witness`: `Witness` holds intermediate results like $\langle \mathcal{F}(x), \mathcal{C}(x) \rangle$, and is used for *blocking*.

The restart strategy resets `Witness` to empty whenever the decision procedure can infer a new `Nogood` element bottom-up by using the $\perp$-rules. The inconsistency inference is triggered by primitive

[4]To explore one TS, either DFS or BFS suffices. There is no assumption about the order in which TSs are explored. The switching could be implemented either as restarting or backjumping, or any sound method that shares previous computations.

clashes or by a (cache) hit of the global data set Nogood. On the next page, we show two procedures $TEST(\cdot,\cdot)$ and $SAT(\cdot,\cdot,\cdot,\cdot,\cdot)$ for the concept satisfiability test in $\mathcal{SHIQ}$. $TEST(\cdot,\cdot)$ is similar to Chapter 4; while $SAT(\cdot,\cdot,\cdot,\cdot,\cdot)$ has been significantly changed to use the *algebraic method*, i.e., *atomic decomposition* and *integer linear programming*. Also note that for the purpose of a clear presentation, the checking of *primitive clashes* is omitted.

The procedure decides C as *unsatisfiable* if $\{C\} \in$ Nogood; or otherwise decides C as *satisfiable* if the size of Nogood is not changed. In all other cases, the procedure restarts. The termination is guaranteed since the size of Nogood is bounded and each restart is triggered only when a new (nontrivial) inconsistency set is found and added to Nogood.

```
PROCEDURE TEST($C, GCI$)
[01]  Nogood := $\emptyset$;
[02]  WHILE (true)
[03]      Witness := $\emptyset$;
[04]      $len$ := $sizeof$(Nogood);
[05]      allocate a tableau node $x_0$ and let $\mathcal{L}(x_0) := \{C\}$;
[06]      $SAT(x_0, GCI, \emptyset, \{C\}, null)$;
[07]      IF ($C \in$ Nogood) RETURN unsatisfiable;              when restarting this line
[08]      IF ($len$ == $sizeof$(Nogood)) RETURN satisfiable;     will run next
[09]  ENDWHILE
END-PROCEDURE {TEST}


PROCEDURE SAT($x, GCI, edge2me, \mathcal{L}(x), parent$)
[10]  IF ($\mathcal{L}(x) \in$ Nogood) RETURN $false$;
[11]  selected := $false$;
[12]  FOR EACH $local\ consistent\ set$ $\mathcal{C}(x)$ of $\mathcal{L}(x)$
[13]      IF ($\mathcal{C}(x) \in$ Nogood) CONTINUE;
[14]      $\mathcal{F}(x)$ := $fine\text{-}tune$ of $\mathcal{C}(x)$;
[15]      IF ($\mathcal{F}(x) \in$ Nogood) CONTINUE;
[16]      IF ($\langle\mathcal{F}(x), \mathcal{C}(x)\rangle \in$ Witness) RETURN $true$;
[17]      selected := $true$;
[18]      BREAK;
[19]  ENDFOR
[20]  IF ($\neg$ selected) THEN
[21]      IF ($parent$ == $null$) THEN
[22]          Nogood := Nogood $\cup$ $\{C\}$;
[23]          ABORT;                                   restart
[24]      ENDIF
[25]      RETURN $false$;
[26]  ENDIF
[27]  Witness := Witness $\cup$ $\{\langle\mathcal{F}(x), \mathcal{C}(x)\rangle\}$;
[28]  perform $atomic\ decomposition$ about $\mathcal{F}(x)$ and build an $integer\ linear\ program$ $A \cdot \vec{v} = \vec{b}$;
[29]  FOR EACH $atomic\ decomposition$ $p \in NewAtomDecompose(x, \mathcal{H})$
[30]      allocate a tableau node $y$ and build an edge (labeled with roles) from $x$ to $y$;
[31]      ret := $SAT(y, GCI, roles(p), concepts(p), x)$;
[32]      IF ($\neg$ ret) THEN $A[id(p)] = \vec{0}$; ENDIF
[33]  ENDFOR
[34]  IF ($A \cdot \vec{v} = \vec{b}$ is infeasible) THEN
[35]      Nogood := Nogood $\cup$ $\{\mathcal{C}(x)\}$;
[36]      ABORT;                               restart
[37]  ENDIF
[38]  IF ($A \cdot \vec{v} = \vec{b}'$ is infeasible) THEN
[39]      Nogood := Nogood $\cup$ $\{\mathcal{F}(x)\}$;
[40]      ABORT;                               restart
[41]  ENDIF
[42]  RETURN $true$;
END-PROCEDURE {SAT}
```

6.6 Proof

6.6.1 Completeness

For the completeness, we need to prove the correctness for concept unsatisfiability. We start with a lemma saying that $\perp$-rules correctly propagate inconsistencies.

Lemma 8. (Completeness) *The $\perp$-rules generate only unsatisfiable sets.*

Proof. By induction on the application of $\perp$-rules. We follow the proof steps as given in Chapter 4. Here we only consider the inconsistency propagation rule $\perp$-4.

- ($\perp$-4): The *atom-decomposition* exhausts all combinations of role fillers (and their negations). The *column vector* of the *coefficient matrix* of the program takes a value $\vec{0}$ if the corresponding role-filler combination is *unsatisfiable*; otherwise it keeps its initial value. We prove the claim by contradiction. Suppose $\mathcal{M}$ is satisfiable, then this leads to a feasible configuration of combinations of (negated) role fillers. This corresponds to a non-$\vec{0}$ integer solution to the integer program, which contradicts the hypothesis. □

The data structure Nogood implements $\perp$-sets in the procedure $TEST(\cdot, \cdot)$. By Lemma 8, no *satisfiable concept* (set) will be added to Nogood (i.e., $\perp$-sets), therefore any satisfiable concept will be decided as "satisfiable".

As a reminder, $SAT(\cdot, \cdot, \cdot, \cdot)$ does not provide a functionality of *clash trigger* simply to avoid cluttering the space. Of course, it is necessary to have a mechanism to recognize fundamental clashes (esp. propositional clashes). However, it should be clear that the functionality of primitive clash triggers can be implemented by making minor modifications to line-10, line-13 and line-15.

Also, we would like to point out that in $\mathcal{SHIQ}$ the "inconsistency propagation" is weaker than in $\mathcal{SHI}$ (and $\mathcal{ALCFI}$, see Chapter 4). The $SAT(\cdot, \cdot, \cdot, \cdot)$ for $\mathcal{SHIQ}$ propagates an inconsistent set through the use of the $\mathcal{F}$ label but not the $\mathcal{L}$ label; by contrast in $\mathcal{ALCFI}$ the $\mathcal{L}$ label can be used for the inconsistency propagation.

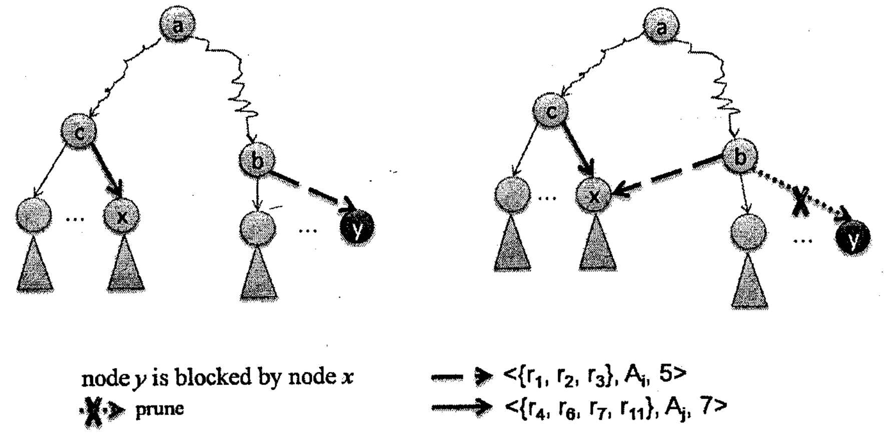


Figure 6.9: Global Tableaux Caching

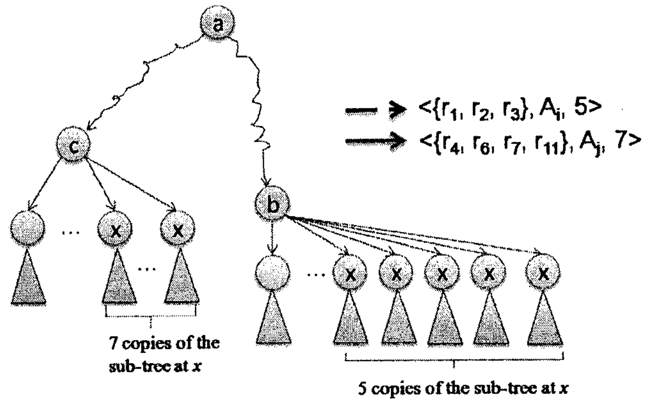

a
c
b
x
x
x
x
x
x
x
<{r₁, r₂, r₃}, Aᵢ, 5>
<{r₄, r₆, r₇, r₁₁}, Aⱼ, 7>
7 copies of the sub-tree at x
5 copies of the sub-tree at x

Figure 6.10: Unravelling and Copying

6.6.2 Soundness

The algorithm takes a DFS traversal to build a *completion tree* starting from the root node x_0, and uses the global data structures `Witness` and `Nogood`. Let $\mathbf{T}$ denote the completed tree. For a node $x_i \in \mathbf{T}$, denote its propositionally completed label by $\mathcal{C}(x_i)$, and the fine-tuned label by $\mathcal{F}(x_i)$.

If a node x_i is expanded and completed before node x_j does, we denote this ordering by $x_i \succ x_j$. The blocking relationship conforms to this (node expansion) ordering (similar to Section 4.7.2 of Chapter 4). As indicated before, the witness is required to be *propositionally completed* (which is equivalent to say the $\sqcap$-rule and the $\sqcup$-rule are no longer applicable), and is fine-tuned and is not in `Nogood`. Only for a propositionally completed node x it is possible to add a tuple of two labels $\langle \mathcal{F}(x), \mathcal{C}(x) \rangle$ to `Witness`.

Lemma 9. (Soundness) *If there is a tableau tree T for $\mathcal{L}(x_0) = \{C\}$ w.r.t. $\top \sqsubseteq G$, then there is a model M for C w.r.t. $\top \sqsubseteq G$.*

Proof. It takes two steps.

(1) To admit infinite models, we consider paths in $\mathbf{T}$. The mapping $\mathbf{Tail}(p)$ returns the last element in a path p. Give a path $p = [x_0, ..., x_n]$, where x_i are nodes in $\mathbf{T}$, $\mathbf{Tail}(p) = x_n$. Paths in $\mathbf{T}$ are defined inductively as follows:

- for the root node x_0 in $\mathbf{T}$, $[x_0]$ is a path in $\mathbf{T}$.
- for a path p and a node x_i in $\mathbf{T}$, $[p, x_i]$ is a path in $\mathbf{T}$ iff
  - x_i is not blocked, and
    - * x_i is a successor of $\mathbf{Tail}(p)$ and the variable[5] $v_{x_i} > 0$, or
    - * y is a successor of $\mathbf{Tail}(p)$ and x_i blocks y and the variable $v_y > 0$.
  - x_i is not known to be unsat (i.e., $\mathcal{C}(x_i), \mathcal{F}(x_i) \notin$ `Nogood`)

The pre-model $M' = (\Delta^{\mathcal{I}'}, \cdot^{\mathcal{I}'})$ can be defined with:

$\Delta = \{x_p | \ p \text{ is a path in } \mathbf{T} \ \}$

[5]Recall that (see Section 6.3.4) each tableaux edge is labeled with a tuple $\langle A_i, v_i \rangle$, where v_i is one variable of the integer program at the predecessor node. The v_i corresponds to the "number of copies" required for the node at the endpoint of the edge.

$x_p \in (C(\mathbf{Tail}(p)))^{\mathcal{I}'}$

$\{\langle x_p, x_q\rangle \mid \langle x_p, x_q\rangle \in (R)^{\mathcal{I}'}\} = \{\langle x_p, x_q\rangle \in \Delta \times \Delta \mid q = [p,\ \mathbf{Tail}(q)]$ and

1. $\mathbf{Tail}(q)$ is an R-successor of $\mathbf{Tail}(p)$, or
2. $\exists y \in \mathbf{T}$, y is an R-successor of $\mathbf{Tail}(p)$ and $\mathbf{Tail}(q)$ blocks y $\}$

$\bigcup$ $\{\langle x_p, x_q\rangle \in \Delta \times \Delta \mid\ p = [q,\ \mathbf{Tail}(p)]$ and

1. $\mathbf{Tail}(p)$ is an R^--successor of $\mathbf{Tail}(q)$, or
2. $\exists y \in \mathbf{T}$, y is an R^--successor of $\mathbf{Tail}(q)$ and $\mathbf{Tail}(p)$ blocks y $\}$

(2) For a tuple like $\langle A_i, v_x\rangle$, which is a label on the incoming edge to a tableau node x in M', if $v_x > 1$, then generate new tableau nodes $x_1, ..., x_{v_x-1}$ by replicating x. Each new nodes takes the same label as x's and share x's-predecessor, and has the same successors as the node x does. This leads to $M'' = (\Delta^{\mathcal{I}''}, \cdot^{\mathcal{I}''})$. Each element of M'' is locally consistent (see Definition 7) and is saturated w.r.t. local cardinality restrictions. M'' is a model for C and $\top \sqsubseteq G$.

By step-2, the compact pre-model from step-1 is exponentially expanded. □

6.6.3 Complexity

Lemma 10. *The feasibility of the integer linear inequalities can be solved in $2^{O(L^2)}$ in the worst case, where L is the size of the given modal constraints (in binary coding of numbers).*

Proof. Denote the number of *modal constraints* by m; denote the maximum value of numbers (encoded in binary) by 2^p; denote the number of variables by n. It is that $m + p \leq L$ and $n \leq 2^m$.

Recall that for a group of *integer linear inequalities* there is a matrix A which has n columns and m rows, and there is a variable vector $\vec{\mathbf{v}}$ of n variables.

(1) Let us consider the i-th column $\overrightarrow{\mathbf{A}[i]}$ of A. If we fix i but change the value of the variable $\vec{\mathbf{v}}[i]$, there are (at most) $1 + 2^p$ states for $\overrightarrow{\mathbf{A}[i]}$ without exceeding the limit.

(2) Let us introduce a m-dimension COUNTER, which will be used to add up from $\overrightarrow{\mathbf{A}[0]}$ to $\overrightarrow{\mathbf{A}[n]}$. It is sufficient for us to allow each element of the COUNTER to take a range of values in $[0, n \cdot 2^p]$. So, the m-dimension COUNTER has at most $(1 + n \cdot 2^p)^m$ states.

(3) To add up one column, there is at most $(1 + n \cdot 2^p)^m \star (1 + 2^p)$ combinations. So, to add up to n-th column, it takes $c = (1 + n \cdot 2^p)^m \star (1 + 2^p) \star (n - 1)$ steps.

It is sufficient for each summation step to have a cost proportional to the size of the input. Therefore we have:

$c = (1 + n \cdot 2^p)^m \star (1 + 2^p) \star (n-1) \leq d \cdot 2^{L^2}$ for some constant d. $\square$

Lemma 11. *The number of nodes of any tableaux structure generated by the procedure $TEST(\cdot,\cdot)$ is bounded by $2^{O(n)}$, where n is the problem size.*

Proof. $TEST(\cdot,\cdot)$ calls $SAT(\cdot,\cdot,\cdot,\cdot,\cdot)$ and relies on the latter for termination. Notice that the only point where the program fails to infer a Nogood but still performs backtracking is **line** 25 in $SAT(\cdot,\cdot,\cdot,\cdot,\cdot)$.

(1) Suppose **line** 25 is executing at a tableaux node x, the current label $\langle \mathcal{F}(x), \mathcal{C}(x) \rangle$ will not be recorded in Witness, because the only line that enters a current label to Witness is **line** 27, which is not executed yet. Further for node x, the program will not perform the *atomic decomposition*, therefore, x must be a leaf node having no successors.

(2) By executing **line** 25, the program will return to the previous calling point located at **line** 31. Suppose this tableaux node is y. Since the program working at node y has executed line-31, it must also have executed **line** 27 and **line** 30. Therefore, y is not a leaf node and for any non-leaf node its label is cached in Witness. This implies that the number of non-leaf nodes is bounded by $2^{O(n)}$ (the size of Witness). Observe also that in the program between **line** 28 and **line** 42 there are only two possibilities for non-leaf nodes: the program must either "ABORT" or "RETURN *true*", so, eventually one Nogood will be reported or otherwise the tableau structure will be saturated.

(3) In the worst case, the number of tableaux leaf nodes like x might be of $2^{O(n)}$ and their labels are neither in Nogood nor Witness. Other tableaux nodes that are not leaf nodes must be in Witness (according to the second point above), thus the total number of tableaux nodes is bounded by $2^{O(n)}$. $\square$

Three factors are considered : (1) the maximum cost per *tableau node*; (2) the maximum size of a *tableau structure*; (3) the maximum number of tableau structures required to be explored. We allow a cost of $2^{O(n^c)}$ per node for some constant c, but require the size of each tableau structure to be of $2^{O(n)}$ and only $2^{O(n)}$ such tableau structures can be formed.

The procedure starts building a *tableau structure* (TS) from a root node and switches to explore a different TS if the current one can not be saturated without conflicts. When switching to explore another TS, at least one new Nogood will be collected. Since the size of Nogood is bounded by $2^{O(n)}$, at most $2^{O(n)}$ TSs will be explored.

By [Pap81], the *integer programs* resulting from *atomic decompositions* can be solved in the worst case in an exponential time of the problem size even when numbers are binary coded. It is also clear that it takes an exponential cost to perform *atomic decompositions*. These are taken care of by allowing a cost of $2^{O(n^c)}$ per node for some constant c.

By using Witness, the size of a tableaux structure is bounded by $2^{O(n)}$.

Therefore, the total cost is $2^{O(n^c)} * 2^{O(n)} * 2^{O(n)}$. By lemma 10, the constant $c = 2$.

Lemma 12. **(Termination)** *The algorithm terminates in* $2^{O(n^c)}$, *where* n *is the size of the problem and the constant* $c = 2$.

Theorem 13. *The tableau-based decision procedure decides* $\mathcal{SHIQ}$ *concept satisfiability problems in ExpTime in the worst case for binary coding of numbers.*

6.7 Discussion

6.7.1 Integer Linear Inequation

Recall that a system of integer linear inequations is called feasible if it has a solution. Otherwise it is called infeasible. The following lemma says that an infeasible system of integer linear inequations can not become feasible for any possible fine-tuning.

Lemma 14. *Given a system of integer linear inequations* $A_E \cdot \vec{x} = \vec{b}$ *and its fine-tuned counterpart* $A_E \cdot \vec{x} = \vec{b}'$. *If* $A_E \cdot \vec{x} = \vec{b}$ *is infeasible, then* $A_E \cdot \vec{x} = \vec{b}'$ *is infeasible.*

Proof. Recall that A_E is the extension of the coefficient matrix A after introducing surplus and slack variables. Observe that $\vec{\mathbf{b}}'$ results from a fine-tune by subtracting a certain positive vector $\vec{\mathbf{A}}[i]$ of A from $\vec{\mathbf{b}}$, i.e., $\vec{\mathbf{b}}' = \vec{\mathbf{b}} - \vec{\mathbf{A}}[i]$ for some index i. Therefore, if $A_E \cdot \vec{\mathbf{x}} = \vec{\mathbf{b}}'$ has a solution $\vec{\mathbf{x}} = (x_1, x_2, ..., x_i, ...)$, then $A_E \cdot \vec{\mathbf{x}} = \vec{\mathbf{b}}$ will have a solution $\vec{\mathbf{x}} = (x_1, x_2, ..., 1 + x_i, ...)$. $\square$

6.7.2 Atomic Decomposition

The *atomic decomposition* process is known to have another form which is more economic than what is presented in this chapter [HMT01]. The idea for an "economic decomposition" is to process *at-least restrictions* $\exists^{\geq n} R.C$ in a slightly different way, i.e., not introducing the negated *role filler* $\widetilde{C}$ [HMT01]. Besides the optimization of omitting negated role filler for at-least constraints, further optimizations are possible.

As pointed out in [OK99], a *number restriction* over a role-filler composed of the "$\sqcup$" operator has relation to the algebraic operation "+" in *integer domain*. For example, assume A_1 and A_2 have been "decomposed" already, i.e., for all (pair-wise disjoint) atoms $p \in \mathcal{P}$ it is that $(A_1 \in p) \vee (\widetilde{A_1} \in p)$ and $(A_2 \in p) \vee (\widetilde{A_2} \in p)$. Then a *number restriction* like $\exists^{\leq 3} R.(A_1 \sqcup A_2)$ can be directly encoded in an inequality as $\sum_{q \in \mathcal{Q}} \vec{v}[id(q)] \leq 3$ by reusing available variables, where $\mathcal{Q}$ stands for the set of atoms which either A_1 or A_2 is an element of. Similarly, $\exists^{\geq 3} R.(A_1 \sqcup A_2)$ can be encoded in $\sum_{q \in \mathcal{Q}} \vec{v}[id(q)] \geq 3$.

Recall that the *atomic decomposition* process generates a vector of variables and a coefficient matrix. Each atom corresponds to a distinct variable as well as a distinct column of the constant matrix. It is easy to see that in some cases reusing variables and matrix columns could restrict the unnecessary exponential explosion.

6.7.3 Reachability Analysis

In [DH07b], for $\mathcal{ALCQI}$ w.r.t. *general concept inclusions*, restricted *cut-formulae* were introduced for the soundness of the *global (sub)-tableaux caching* [DM00]. It is clear that the cut-formulae can be treated like GCIs. In this chapter, instead of using *cut-formulae* [DH07b], the notion of a *locally consistent set* (corresponding to the choose-rule) is used.

The non-determinism from guessing "backward propagation" of constraints due to a combination of *inverse roles* and *qualified number restrictions* could possibly be suppressed by a *reachability analysis* similar to [DH06]. It is conjectured that a combination of the *reachability analysis* and a more restrained use of cut-formulae might lead to "acceptable" run-time performance for "realistic application problems".

The usefulness of the *algebraic method* has been manifested in providing theoretical tableaux calculi and in supporting practical reasoning tasks when dealing with a role hierarchy and transitive roles as previously shown in [HM01a]. It can provide worst-case optimal algorithms for *concept satisfiability problems* in both the PSpace class[6] and the ExpTime class. We are quite convinced that the *algebraic method* is a general tool for satisfiability problems involving *qualified number restrictions.*

In Section 6.8, we will give two small examples. The first example illustrates how to use the *atomic decomposition principle* (Section 6.3.3) and how to form the corresponding *integer linear inequalities* for a set of *number restrictions.* The second example illustrates how to use the simplified termination condition to realize the global tableaux caching (Section 6.4.3).

6.8 Examples

Example 1. (Decomposition and Integer Linear Inequalities) Given a role hierarchy $\mathcal{H}$ consisting of two *role inclusions* $\{R_2 \sqsubseteq R_1, R_3 \sqsubseteq R_1\}$, and $\mathcal{M}$ a set of 4 *number restrictions* $\{\exists^{\leq 4}R_1.(\neg C_3 \sqcap (C_1 \sqcup C_2)), \exists^{\geq 3}R_1.(C_1 \sqcap \neg C_3), \exists^{\geq 2}R_2.(C_1 \sqcap C_3), \exists^{\geq 2}R_3.(C_2 \sqcap C_3)\}$. The question is to test the satisfiability of $\mathcal{M}$ w.r.t. $\mathcal{H}$.

First, we use the *two-phase decomposition* (see Section 6.3.3) on the role hierarchy $\mathcal{H}$. This gives the following decompositions:

(H1) $\{R_1, R_2, R_3\}$
(H2) $\{R_1, R_2\}$
(H3) $\{R_1, R_3\}$
(H4) $\{R_1\}$

Second, we use the *two-phase decomposition* on role fillers of $\mathcal{M}$ (based on the discussion given in Section 6.7.2). This gives the following decompositions (modulo some decompositions not used):

(F1) $\{C_1, C_2, C_3\}$
(F2) $\{C_1, C_2, \neg C_3\}$
(F3) $\{C_1, \neg C_2, C_3\}$
(F4) $\{C_1, \neg C_2, \neg C_3\}$
(F5) $\{\neg C_1, C_2, C_3\}$
(F6) $\{\neg C_1, C_2, \neg C_3\}$

[6]For the *algebraic method* to use only a polynomial space, the *atomic decomposition* should be performed in a space-economic way, i.e., generating a fixed number of atoms at a time and the *integer linear program* should be implicitly formed and solved.

Third, for each *number restriction* in $\mathcal{M}$, we form one *integer linear inequality*. There are 4 *integer linear inequalities* as follows (one for each *number restriction*).

$$\begin{aligned}
(1)\ & v_{<H1,F2>} + v_{<H1,F4>} + v_{<H1,F6>} + v_{<H2,F2>} + v_{<H2,F4>} + v_{<H2,F6>} + v_{<H3,F2>} + \\
& + v_{<H3,F4>} + v_{<H3,F6>} + v_{<H4,F2>} + v_{<H4,F4>} + v_{<H4,F6>} \leq 4; \\
(2)\ & v_{<H1,F2>} + v_{<H1,F4>} + v_{<H2,F2>} + v_{<H2,F4>} + v_{<H3,F2>} + v_{<H3,F4>} + v_{<H4,F2>} + \\
& + v_{<H4,F4>} \geq 3; \\
(3)\ & v_{<H1,F1>} + v_{<H1,F3>} + v_{<H2,F1>} + v_{<H2,F3>} \geq 2; \\
(4)\ & v_{<H1,F1>} + v_{<H1,F5>} + v_{<H3,F1>} + v_{<H3,F5>} \geq 2;
\end{aligned}$$

Fourth, the set of the above 4 inequalities is *feasible* for there is a solution as follows.

$v_{<H1,F1>} = 2$, $v_{<H1,F2>} = 3$, and other variables take a value of 0.

Fifth, the solution gives a configuration of modal successors (to satisfy $\mathcal{M}$ w.r.t. $\mathcal{H}$): 2 $\{R_1, R_2, R_3\}$-successors with a *role filler* $\{C_1, C_2, C_3\}$ and 3 $\{R_1, R_2, R_3\}$-successors with a *role filler* $\{C_1, C_2, \neg C_3\}$

Example 2. (Generalized Blocking)

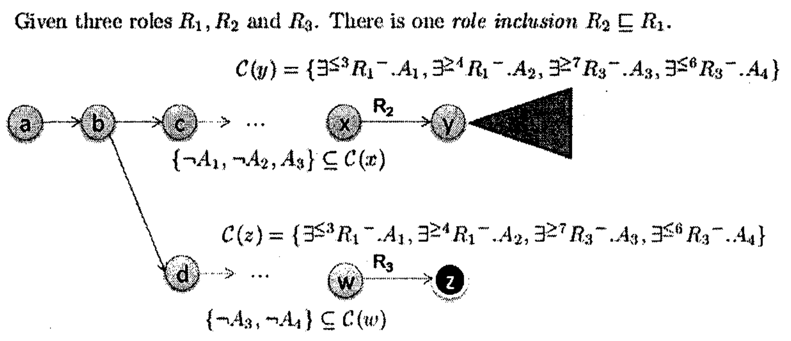

Given three roles R_1, R_2 and R_3. There is one *role inclusion* $R_2 \sqsubseteq R_1$.
$C(y) = \{\exists^{\leq 3} R_1^-.A_1, \exists^{\geq 4} R_1^-.A_2, \exists^{\geq 7} R_3^-.A_3, \exists^{\leq 6} R_3^-.A_4\}$
a
b
c
x
R_2
y
$\{\neg A_1, \neg A_2, A_3\} \subseteq C(x)$
$C(z) = \{\exists^{\leq 3} R_1^-.A_1, \exists^{\geq 4} R_1^-.A_2, \exists^{\geq 7} R_3^-.A_3, \exists^{\leq 6} R_3^-.A_4\}$
d
w
R_3
z
$\{\neg A_3, \neg A_4\} \subseteq C(w)$

Figure 6.11: An Example of the Generalized Blocking (Global Tableaux Caching)

In this example, we give an example of the simplified termination condition (introduced in Section 6.4.3 as a notion of "blocking"). In the tableaux structure as shown in Figure 6.11, node y is an R_2-successor of node x, node z is an R_3-successor of node w. We assume the *role hierarchy* requires $R_2 \sqsubseteq R_1$. It is known that $C(y) = \{\exists^{\leq 3} R_1^-.A_1, \exists^{\geq 4} R_1^-.A_2, \exists^{\geq 7} R_3^-.A_3, \exists^{\leq 6} R_3^-.A_4\}$, and $C(z) = \{\exists^{\leq 3} R_1^-.A_1, \exists^{\geq 4} R_1^-.A_2, \exists^{\geq 7} R_3^-.A_3, \exists^{\leq 6} R_3^-.A_4\}$. Also, it is given that $\{\neg A_1, \neg A_2, A_3\} \subseteq C(x)$ and $\{\neg A_3, A_2, \neg A_4\} \subseteq C(w)$.

Recall the notion of *fine-tuning* introduced in Section 6.3.1. By performing fine-tuning on *modal constraints* in $C(y)$ and $C(z)$ respectively, we get $\mathcal{F}(y) = C(y) = \mathcal{F}(z) = C(z)$. According to Section 6.4.3, node z is blocked by node y. Note that in contrast to the "dynamic double blocking" technique

[HS02], it is not required that $\mathcal{L}(\langle x, y\rangle) = \mathcal{L}(\langle w, z\rangle)$ nor is it required that y must be an ancestor node of z.

6.9 Summary

In many areas of the engineering world, we frequently encounter various constraints regarding large integer values[7]. For example, the IPv4 (Inernet Protocol Version 4) specification stipulates that the header portion of each Transport-Control Protocol (TCP) segment should have *at least* 10 fields and *at most* 11 fields; it is necessary to implement *at least* 4 different timers to develop the TCP protocol in IPv4; and the *minimum transfer unit* (MTU) of *Ethernet Network* must be *at least* 576 bytes, etc. It was pointed out in [HM01a] that the `algebraic reasoner` scales smoothly for large number values by implementing algorithms such as the *simplex methods.*

A *binary coding* of integer numbers is exponentially succinct than a *unary coding*; the *coding* of numbers has been playing a critical role when it comes to examine if a *decision procedure* is indeed *worst-case optimal* for the *satisfiability problems* in DLs with *number restrictions.* In the past, it was not clear how the tableau-based approach could possibly achieve a worst-case ExpTime decision procedure in the strong sense of binary coding of number values. So far, the known ExpTime decision procedure, allowing *binary coding* of numbers, is automaton-based [Tob01]. In that approach, a *looping tree automaton* is constructed for the given concept and the GCIs. The automaton is exponential in the size of the input and uses abstract "limiting functions" that are considered impractical for implementations [Tob01].

This chapter investigated the *concept satisfiability problem* in the DL $\mathcal{SHIQ}$ (w.r.t. *general concept inclusions*), and also investigates the applicability of the global *(sub)-tableaux caching* technique in *tree structures* restricted by local cardinality constraints and inverse relations. $\mathcal{SHIQ}$ is known to have the tree model property [BCM+03]. The global (sub)tableaux caching technique is inspired by [DM00]. We borrowed ideas and techniques from [Pap81, CL94, DM00], and [OK99, HM01a, Tob99, Tob01, HS02], and [BHLW03, HM04, Hla04, HM04]. The *algebraic method* has been extended and combined with the global (sub)-tableaux caching technique for a DL having *number restrictions* and *inverse roles.* The worst case complexity for the satisfiability problem in $\mathcal{SHIQ}$ is $2^{O(n^2)}$, where

[7]Yet there are applications using only very small numbers such as 1 or 2, as argued in [Hor02].

$n = \|\mathcal{T}\| + \|C\|$ in the strong sense of binary coding of numbers. This upper bound is also applicable to $\mathcal{SHIQ}$ Abox consistency problems (even not assuming *unique name assumption*).

Chapter 7

Conclusion and Future Work

Description Logics (DLs) are gaining popularity. In the presence of *inverse roles* and *number restrictions*, however, most successful tableau-based DL reasoning systems exhibit problems, for example, degraded performance. The current loss of performance is largely due to lack of some well-known optimization techniques, especially the one for caching the satisfiability status of modal successors.

Based on an investigation of the current research status in optimization techniques and an analysis of why current approaches to handling *inverse roles* are either inefficient or invalid, several new solutions (including *elimination of inverse roles* and the *sound global tableaux caching* techniques) are proposed in this thesis. For *number restrictions*, it is known from the RACER [HM01b] experience that the *algebraic method* has better run-time performance. This thesis has extended the previous approach of [HM01a] to $\mathcal{SHIQ}$ and proved that the *algebraic method* indeed leads to a worst-case optimal decision procedure. This could be the first and important step to a plausible explanation to the practical superiority of the *algebraic method* over other tableau-based approaches [BCM+03].

7.1 Summary

The following summarizes our research results.

- A dynamic *global sub-tableaux caching technique* was presented in Chapter 2.

  It addresses the soundness of the conventional sub-tableaux caching techniques. This technique first appeared in [DH05].

  The technique presented in Chapter 2 uses two labels per tableau-node, and relies on the notion of "dynamic" which was previously seen in various "dynamic" blocking techniques [HS02]. If further using the *reachability analysis* technique (also presented in [DH05]), the proposed *dynamic tableaux caching* technique can be "static" (in certain situations).

The availability of the caching technique can speed up the reasoning process. First, the witness space is no longer restricted to ancestor nodes. Second, the "static" version of the caching technique [DH05] requires no re-checking and thus it is safe to discard nodes once they are successfully cached. By doing this, the tableau algorithm is more space-economic[1]. Our approach can successfully deal with a set of (possibly cyclic) unfolding rules and a role hierarchy, thus it well meets the typical requirement from real applications [DH06]. However, the proposed "dynamic caching" technique (as well as the popular "dynamic blocking technique") has a serious drawback, i.e., it has only a very weak *inconsistency propagation* capability. A weak inconsistency propagation[2] would possibly result in, as also observed in [DH06], a relatively lower cache hit rate for a group of test cases specially designed.

- To benefit from an effective reuse of previously computed satisfiability status as well as unsatisfiability status (as supported by global tableaux caching techniques [DM00]), it is desirable to eliminate inverse roles. The *elimination of inverse roles* was presented in Chapter 3. The empirical results were shown in Appendix B. This conversion technique is different from the method of [Gia96] and [Sch91] in that ours relies on the Ramsey-rule [Ram31]. According to [HST99a, HST99b], FaCT [Hor98] was able to classify a UML terminology (19 concepts and 42 axioms) in less than 0.1s of (266MHz Pentium) CPU time, but eliminating inverse roles using an *embedding technique* [Gia96, Sch91] gives an equisatisfiable FaCT terminology with an additional 84 axioms which FaCT was unable to classify in 12 hours of CPU time. By contrast (see Appendix B), our *elimination of inverse roles* only slightly increased (less than a constant factor of 5 times) the size of the original ontologies, and RACER [HM01b] solved all converted ontologies within an acceptable time (see **Appendix B**).

- In Chapter 4, a worst-case ExpTime tableau-based decision procedure was presented for the

[1]This is also observed in the experiments of a recent paper [MSH07] where a *pairwise anywhere blocking* technique was proposed and implemented for $\mathcal{SHIQ}$.

[2]We conjecture that the weakness in inconsistency propagation is inherent for tableau-based decision procedures when reasoning for DLs with inverse roles. Available techniques (such as the "pseudo model merging" technique [Hor95, THPS07] which is considered very useful for almost all DLs) do not prevent that tableau-based procedures suffer from these deficiencies.

DL $\mathcal{ALCFI}$. Part of this chapter is based on [DH07a].

It is known that $\mathcal{ALCFI}$ lacks the *finite model property*, and consequently a sophisticated *double blocking technique* is used to terminate the search for models [HS02]. This chapter used the commonly-used *equality blocking* technique. This idea behind such a blocking shares the intuition of the elimination of *functional roles* from [CGLN01]. Moreover, this chapter showed that *recorded axioms* are sufficient for the (sub)tableaux caching technique to be truly "static" without the help of a *reachability analysis* (as proposed in [DH05]). Further, it is quite promising that the technique presented in this chapter can be extended to $\mathcal{SHIF}$, a DL corresponding to the ontology language OWL-lite.

- In Chapter 5 we showed three different reductions that convert *concept satisfiability problems* from $\mathcal{SHQ}$ to $\mathcal{ALCQ}$, from $\mathcal{SHOI}$ to $\mathcal{SHO}$, and from acyclic $\mathcal{ALCHQI}$ Tbox to acyclic $\mathcal{ALCQ}$ Tbox. This chapter improves our previous work in [DHW07] and [DH07b].

- In Chapter 6, the *algebraic method* [OK99, HM01a] was extended for $\mathcal{SHIQ}$. By applying the *integer linear programming* technique presented in [Pap81], we also showed that the integer linear programs (resulted from applying the *atomic decomposition* principle [OK99, HM01a] on *modal constraints*) can be solved in $2^{O(n^2)}$ where n is the size of the constraints (under binary coding of numbers).

  The decision procedure is worst-case exponential time (with an improved upper bound) in the strong sense of binary coding of numbers. This solved the question raised (e.g., in [Tob01]) about how a tableau-based approach could possibly lead to an optimal algorithm for concept satisfiability tests about *number restrictions* with respect to GCIs.

  Previously only Tobies [Tob01] had given an automata-based procedure in $2^{O(n^6)}$ for binary coding of numbers in $\mathcal{ALCQI}_b$. Rajeev and Linh [GN07] recently gave a $2^{O(n^2)}$ tableaux procedure for $\mathcal{SHI}$. Our upper bound for $\mathcal{SHIQ}$ is $2^{O(n^2)}$. This upper bound also applies to $\mathcal{SHIQ}$ Abox consistency problems (even not assuming the *unique name assumption* [BCM+03] for Abox individuals).

7.2 Conclusion and Future Research Direction

This thesis has illustrated our research work in two aspects.

The first aspect is the search for new reasoning techniques (including a revitalizing of the *global tableaux caching technique* known to be unsound in the past for DLs with *inverse roles*, a (re)discovering of a variant of the Ramsey-rule [Ram31], and the application of this rule to *elimination of inverse relations*). We have demonstrated a use of the variant of the Ramsey-rule for elimination of inverse relations for a sub-family of DLs with inverse roles (i.e., $\mathcal{SHIF}$ and $\mathcal{SHOI}$), and have also showed its usefulness by carrying out experiments on realistic ontologies (corresponding to knowledge bases in the DL $\mathcal{ALCI}$). In contrast to simplicity of elimination of *inverse roles* in $\mathcal{ALCFI}$ and $\mathcal{SHOI}$, we have also shown elimination of *inverse roles* for $\mathcal{ALCHQI}$ which otherwise becomes more complex and less practical for reasoning (though it is possible to implement this transformation). By combining with tableau-based decision procedures, three variants of the *global tableaux caching* technique have been shown with different strength in "inconsistency propagation." In terms of "inconsistency propagation", the *dynamic global tableaux caching* is of the least power, and the *global tableaux caching* for $\mathcal{SHIQ}$ (being the most general of the three variants) is relatively stronger than the former. It should be pointed out that it is *number restrictions* (other than *inverse roles*) that make the tableaux caching in $\mathcal{SHIQ}$ less powerful. Consequently, the largest fragment in $\mathcal{SHIQ}$ that can benefit from the *global tableaux caching* technique as powerful as $\mathcal{ALC}$ is the DL $\mathcal{SHIF}$, which corresponds to OWL-Lite (an ontology language adopted in Semantic Web, see www.w3c.org.).

The second aspect of our research is to provide for a theoretical explanation for an existing technique (i.e., the *algebraic method* scaling better in practice). Essentially, this thesis has shown that the combination of the *global tableaux caching* technique and the *integer linear programming* technique is sufficient for the *algebraic method* to be worst case optimal for binary coding of numbers.

This thesis has covered major fragments[3] (i.e., $\mathcal{SHIQ}$ and $\mathcal{SHOI}$) of the DL $\mathcal{SHOIQ}$ [HS05, HS07]. For the *concept satisfiability problem* in this language, an NEXPTIME hardness proof was given in [Tob01] and refined in [Lut04]. A tableau-based decision procedure for $\mathcal{SHOIQ}$ appeared recently in [HS05, HS07]. The DL $\mathcal{SHOIQ}$ corresponds to the ontology language OWL-DL, which is expected

[3]Actually, as was pointed out in [DH07b], the presented tableau-based algorithm for $\mathcal{SHIQ}$ can be adapted to a worst-case optimal decision procedure for $\mathcal{SHOQ}$ also.

to be the mainstream language in the ontology design and implementation. Given the successful experience of RACER [HM01b] in using the *algebraic method* for $\mathcal{SHQ}$ [HM01a] and the proposed extension for $\mathcal{SHIQ}$, it is therefore meaningful to start investigating how to extend the *algebraic method* for $\mathcal{SHOIQ}$.

Bibliography

[AG07] Pietro Abate and Rajeev Gore. System Description: The Tableau Work Bench. In *Proceedings of the 5nd World Conference on the Methods for Modalities*, Paris, France, 2007. Electronic Notes in Theoretical Computer Science, www.elsevier.nl/locate/entcs.

[Baa90] Franz Baader. Augmenting Concept Languages by Transitive Closure of Roles: An Alternative to Terminological Cycles. *DFKI-RR-90-13*, pages 1–33, December 1990.

[BBL05] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ Envelope. In Leslie Pack Kaelbling and Alessandro Saffiotti, editors, *IJCAI*, pages 364–369. Professional Book Center, 2005.

[BCM+03] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications.* Cambridge University Press, 2003.

[BDTW07] Shoham Ben-David, Richard J. Trefler, and Grant E. Weddell. Bounded Model Checking with Description Logic Reasoning. In Nicola Olivetti, editor, *TABLEAUX*, volume 4548 of *Lecture Notes in Computer Science*, pages 60–72. Springer, 2007.

[BHLW03] Franz Baader, Jan Hladik, Carsten Lutz, and Frank Wolter. From Tableaux to Automata for Description Logics. In Moshe Y. Vardi and Andrei Voronkov, editors, *LPAR*, volume 2850 of *Lecture Notes in Computer Science*, pages 1–32. Springer, 2003.

[BHNP94] Franz Baader, Bernhard Hollunder, Bernhard Nebel, and Hans-Jurgen Profitlich. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS Get a Move On. *Applied Intelligence 4(2)*, pages 109–132, 1994.

[BML$^+$05] F. Baader, M. Milicic, C. Lutz, U. Sattler, and F. Wolter. Integrating Description Logics and Action Formalisms for Reasoning about Web Services. LTCS-Report LTCS-05-02, Chair for Automata Theory, Institute for Theoretical Computer Science, Dresden University of Technology, Germany, 2005. See http://lat.inf.tu-dresden.de/research/reports.html.

[BS85] Ronald J. Brachman and James G. Schmolze. An Overview of the KL-ONE Knowledge Representation System. *Cognitive Science*, 9(2), 1985.

[BS01] F. Baader and U. Sattler. An Overview of Tableau Algorithms for Description Logics. *Studia Logica*, pages 5–40, 2001.

[CGLN01] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Daniele Nardi. Reasoning in Expressive Description Logics. In John Alan Robinson and Andrei Voronkov, editors, *Handbook of Automated Reasoning*, pages 1581–1634. Elsevier and MIT Press, 2001.

[CGR98] Diego Calvanese, Giuseppe De Giacomo, and Riccardo Rosati. A Note on Encoding Inverse Roles and Functional Restrictions in $\mathcal{ALC}$ Knowledge Bases. In Enrico Franconi, Giuseppe De Giacomo, Robert M. MacGregor, Werner Nutt, and Christopher A. Welty, editors, *Description Logics*, volume 11 of *CEUR Workshop Proceedings*. CEUR-WS.org, 1998.

[CL94] Diego Calvanese and Maurizio Lenzerini. Making Object-oriented Schemas More Expressive. *PODS '94: Proceedings of the thirteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 243–254, 1994.

[DH05] Yu Ding and Volker Haarslev. Toward Efficient Reasoning for Description Logics with Inverse Roles. *DL-Workshop'05*, 2005.

[DH06] Yu Ding and Volker Haarslev. Tableau Caching for Description Logics with Inverse and Transitive Roles. *DL-Workshop'06*, 2006.

[DH07a] Yu Ding and Volker Haarslev. A Decision Procedure for Description Logic $\mathcal{ALCFI}$. *TABLEAUX'07*, 2007.

[DH07b] Yu Ding and Volker Haarslev. An ExpTime Tableau-based Decision Procedure for $\mathcal{ALCQI}$. *DL-Workshop'07*, 2007.

[DHW07] Yu Ding, Volker Haarslev, and Jiewen Wu. A New Mapping from $\mathcal{ALCI}$ to $\mathcal{ALC}$. *DL-Workshop'07*, 2007.

[DLNN91] Donini, Lenzerini, Nardi, and Nutt. The Complexity of Concept Languages. *KR-91*, pages 151–162, 1991.

[DLNS94] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in Concept Languages: From Subsumption to Instance Checking. *J. Log. Comput.*, 4(4):423–452, 1994.

[DLNS96] F. M. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. *Reasoning in Description Logics.* in Gerhard Brewka, editor, Foundation of Knowledge Representation, CSLI Publication, 1996.

[DM00] Francesco M. Donini and Fabio Massacci. ExpTime Tableaux for $\mathcal{ALC}$. *Artifical Intelligence*, 124(1):87–138, 2000.

[Edw04] Anthony W. F. Edwards, editor. *Cogwheels of the Mind: the story of Venn diagrams.* Johns Hopkins University Press, Baltimore and London, 2004.

[Fre95] Jon William Freeman. *Improvements to Propositional Satisfiability Search Algorithms.* PhD thesis, University of Pennsylvania, 1995.

[GDM96] Giuseppe De Giacomo, Francesco M. Donini, and Fabio Massacci. Exptime tableaux for alc. In Lin Padgham, Enrico Franconi, Manfred Gehrke, Deborah L. McGuinness, and Peter F. Patel-Schneider, editors, *Description Logics*, volume WS-96-05 of *AAAI Technical Report*, pages 107–110. AAAI Press, 1996.

[Gia95] Giuseppe De Giacomo. *Decidability of Class-based Knowledge Representation Formalisms.* PhD thesis, Universita' di Roma "La Sapienza", Roma, Italy, 1995.

[Gia96] Giuseppe De Giacomo. Eliminating "Converse" from Converse PDL. *Journal of Logic, Language and Information*, 5(2):193–208, 1996.

[Gin93] Matthew L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research (JAIR)*, 1:25–46, 1993.

[GM00] Giuseppe De Giacomo and Fabio Massacci. Combining Deduction and Model Checking into Tableaux and Algorithms for Converse-PDL. *Inf. Comput.*, 162(1-2):117–137, 2000.

[GN07] Rajeev Gore and Linh Anh Nguyen. EXPTIME Tableaux with Global Caching for Description Logics with Transitive Roles, Inverse Roles and Role Hierarchies. *Automated Reasoning with Analytic Tableaux and Related Methods*, LNCS 4548:133–148, 2007.

[Haa00] Volker Haarslev. *Theory and Practice of Visual Languages and Description Logics.* Habilitationsschrift, Universität Hamburg, Fachbereich Informaik, 2000.

[HB91] Hollunder and Baader. Qualifying Number Restriction in Concept Languages. *KR'91*, pages 335–346, 1991.

[Hla04] Jan Hladik. A Tableau System for the Description Logic $\mathcal{SHIO}$. In Ulrike Sattler, editor, *IJCAR Doctoral Programme*, volume 106 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[HM00a] Volker Haarslev and Ralf Möller. Consistency Testing: The RACE Experience. *Proc. of TABLEAUX'2000, International Conference, Automated Reasoning with Analytic Tableaux and Related Methods*, pages 57–61, July 2000.

[HM00b] Volker Haarslev and Ralf Möller. High Performance Reasoning with Very Large Knowledge Bases. *Proceedings of the International Workshop in DL2000*, pages 143–152, 2000.

[HM01a] Volker Haarslev and Ralf Möller. Combining Tableaux and Algebraic Decision Procedures for Dealing with Qualified Number Restrictions in Description Logics. *IJCAR-2001*, pages 39–48, 2001.

[HM01b] Volker Haarslev and Ralf Möller. RACER System Description. *IJCAR'2001, Vol 2083 of LNAI, Springer*, pages 701–706, 2001.

[HM04] Jan Hladik and Jörg Model. Tableau Systems for $\mathcal{SHIO}$ and $\mathcal{SHIQ}$. In Volker Haarslev and Ralf Möller, editors, *Description Logics*, volume 104 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2004.

[HMT01] Volker Haarslev, Ralf Möller, and Anni-Yasmin Turhan. Exploiting Pseudo Models for TBox and ABox Reasoning in Expressive Description Logics. *Proceedings of International Joint Conference on Automated Reasoning, IJCAR'2001*, pages 61–75, June 2001.

[HN90] Bernhard Hollunder and Werner Nutt. Subsumption Algorithms for Concept Languages. *DFKI Report: RR-90-04*, 1990.

[Hor95] I. Horrocks. A Comparison of Two Terminological Knowledge Representation Systems. Master's thesis, University of Manchester, 1995.

[Hor98] Ian Horrocks. Using an Expressive Description Logic: FaCT or Fiction? In *Proc. of the 6th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR'98)*, pages 636–647, 1998.

[Hor02] Ian Horrocks. Backtracking and Qualified Number Restrictions: Some Preliminary Results. In *Proc. of the 2002 Description Logic Workshop (DL 2002)*, volume 63 of *CEUR*, pages 99–106, 2002.

[Hor03] Ian Horrocks. Implementation and Optimization Techniques. In Baader et al. [BCM+03], pages 306–346.

[HPS98] Ian Horrocks and Peter F. Patel-Schneider. FaCT and DLP. *In Automated Reasoning with Analytic Tableaux and Related Methods: Proceedings of Tableaux'98*, pages 27–30, May 1998.

[HS99] Ian Horrocks and Ulrike Sattler. A Description Logic with Transitive and Inverse Roles and Role Hierarchies. *J. of Logic and Computation*, 9(3):385–410, 1999.

[HS02] Ian Horrocks and Ulrike Sattler. Optimised Reasoning for $\mathcal{SHIQ}$. In *Proc. of the 15th Eur. Conf. on Artificial Intelligence (ECAI 2002)*, pages 277–281, July 2002.

[HS05] Ian Horrocks and Ulrike Sattler. A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. In *Proc. of the 19th Int. Joint Conf. on Artificial Intelligence (IJCAI 2005)*, pages 448–453, 2005.

[HS07] Ian Horrocks and Ulrike Sattler. A Tableaux Decision Procedure for $\mathcal{SHOIQ}$. *J. of Automated Reasoning*, pages 249–276, 2007. Vol. 39, Issue 3.

[HST98] Ian Horrocks, Ulrike Sattler, and Stephan Tobies. A PSPACE-Algorithm for Deciding $\mathcal{ALCNI}_{\mathcal{R}^+}$-Satisfiability. *LTCS-Report 9808*, page 35, August 1998.

[HST99a] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Description Logics with Functional Restrictions, Inverse and Transitive Roles, and Role Hierarchies. In *Proceedings of the first workshop on Methods for Modalities (M4M-1)*, May 1999.

[HST99b] I. Horrocks, U. Sattler, and S. Tobies. Practical Reasoning for Expressive Description Logics. In H. Ganzinger, D. McAllester, and A. Voronkov, editors, *Proc. of the 6th Int. Conf. on Logic for Programming and Automated Reasoning (LPAR'99)*, number 1705 in Lecture Notes in Artificial Intelligence, pages 161–180. Springer, 1999.

[HT00] I. Horrocks and S. Tobies. Reasoning with Axioms: Theory and Practice. *Proc. of the 7th Int. Conf. on Principles of Knowledge Representation and Reasoning (KR 2000)*, pages 285–296, 2000.

[HW06] Alexander K. Hudek and Grant Weddell. Binary Absorption in Tableaux-Based Reasoning for Description Logics. *DL-Workshop'06*, 2006.

[KSZ07] Yevgeny Kazakov, Ulrike Sattler, and Evgeny Zolin. How Many Legs Do I Have? Non-Simple Roles in Number Restrictions Revisited. In Nachum Dershowitz and Andrei Voronkov, editors, *LPAR*, volume 4790 of *Lecture Notes in Computer Science*, pages 303–317. Springer, 2007.

[LL00] Hector J. Levesque and Gerhard Lakemeyer. *The Logic of Knowledge Bases.* The MIT Press, 2000.

[LS00] C. Lutz and U. Sattler. Mary likes all Cats. In F. Baader and U. Sattler, editors, *Proceedings of the 2000 International Workshop in Description Logics (DL2000)*, pages 213–226, Aachen, Germany, August 2000.

[Lut99] Carsten Lutz. Complexity of Terminological Reasoning Revisited. *LPAR-1999*, 99:181–200, 1999.

[Lut04] Carsten Lutz. An Improved NExpTime-hardness Result for the Description Logic $\mathcal{ALC}$ Extended with Inverse Roles, Nominals, and Counting. *LTCS-Report 04-07, Technical University Dresden*, 2004.

[Lut07] Carsten Lutz. Inverse Roles Make Conjunctive Queries Hard. In *Description Logics*, 2007.

[Mas00] Fabio Massacci. Single step tableaux for modal logics. *J. Autom. Reason.*, 24(3):319–364, 2000.

[Min85] M. Minsky. A Framework for Representing Knowledge. In *Readings in Knowledge Representation*, pages 245–262. Morgan Kaufmann, 1985.

[MSH07] Boris Motik, Rob Shearer, and Ian Horrocks. Optimized Reasoning in Description Logics using Hypertableaux. In Frank Pfenning, editor, *Proc. of the 21st Conference on Automated Deduction (CADE-21)*, volume 4603 of *LNAI*, pages 67–83, Bremen, Germany, July 17–20 2007. Springer.

[OK99] H. J. Ohlbach and J. Köhler. Modal Logics, Description Logics and Arithmetic Reasoning. *Artificial Intelligence*, 109:1–31, 1999.

[Pap81] Christos H. Papadimitriou. On the Complexity of Integer Programming. *J. ACM*, 28(4):765–768, 1981.

[Pet98] Patel-Schneider Peter. DLP System Description. *DL-98*, pages 87–89, 1998.

[Ram31] F. Plank Ramsey. *Truth and Probability.* In Foundations of Mathematics and Other Logical Essays, R.B. Braithwaite, ed., New York., 1931.

[Sat96] Ulrike Sattler. A Concept Language Extended with Different Kinds of Transitive Roles. *Deutsche Jahrestagung fur Kunstliche Intelligenz*, pages 333–345, 1996.

[Sch86] Alexander Schrijver. *Theory of Linear and Integer Programming.* A Wiley-Interscience Publication, 1986.

[Sch91] Klaus Schild. A Correspondence Theory for Terminological Logics: Preliminary Report. *IJCAI*, pages 466–471, 1991.

[SGP06] Evren Sirin, Bernardo Cuenca Grau, and Bijan Parsia. From Wine to Water: Optimizing Description Logic Reasoning for Nominals. In Patrick Doherty, John Mylopoulos, and Christopher A. Welty, editors, *KR*, pages 90–99. AAAI Press, 2006.

[SP06] Evren Sirin and Bijan Parsia. Pellet System Description. *DL-Workshop'06*, 2006.

[SPG$^+$07] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz. Pellet: A Practical OWL-DL Reasoner. *Journal of Web Semantics*, 2007.

[SSS91] Manfred Schmidt-Schaubßand Gert Smolka. Attributive Concept Descriptions with Complements. *Artificial Intelligence*, 48(1):1–26, 1991.

[TH04] Dmitry Tsarkov and Ian Horrocks. Efficient Reasoning with Range and Domain Constraints. *DL-Workshop'04*, 2004.

[TH06] Dmitry Tsarkov and Ian Horrocks. FaCT++ Description Logic Reasoner: System Description. In *Proc. of the Int. Joint Conf. on Automated Reasoning (IJCAR 2006)*, volume 4130 of *Lecture Notes in Artificial Intelligence*, pages 292–297. Springer, 2006.

[THPS07] Dmitry Tsarkov, Ian Horrocks, and Peter F. Patel-Schneider. Optimising Terminological Reasoning for Expressive Description Logics. *J. Autom. Reason.*, 39(3):277–316, 2007.

[Tob99] Stephan Tobies. On the Complexity of Counting in Description Logics. *DL-Workshop'99*, 1999.

[Tob00] Stephan Tobies. The Complexity of Reasoning with Cardinality Restrictions and Nominals in Expressive Description Logics. *J. Artif. Intell. Res. (JAIR)*, 12:199–217, 2000.

[Tob01] Stephan Tobies. *Complexity Results and Practical Algorithms for Logics in Knowledge Representation*. PhD thesis, RWTH Aachen, 2001.

[Tsa03] Dmitry Tsarkov. DL Reasoner vs. First-Order Prover. *DL-Workshop'03*, pages 152–159, 2003.

[Vas83] Chavatal Vasek. *Linear Programming*. W.H. Freeman & Company, 1983.

[WH08] Jiewen Wu and Volker Haarslev. Planning of Axiom Absorptions. In *DL-Workshop'08, submitted.*, 2008.

Appendix A

Tableau-based Decision Procedures and Optimizations

A.1 Tableau-based Decision Procedures

When discussing tableau-based decision procedures for description logics, many papers use a form of constraint systems, while others use a form of labeled deduction on relational structures [Haa00, BS01, BCM$^+$03]. As indicated before, *tableau-based decision procedures* were first applied to description logics as *constraint systems* [SSS91, HN90]. In practice, a majority of DL systems implement their DL reasoners based on tableaux procedures, usually with sophisticated optimizations, e.g., RACER [HM01b], TWB [AG07], HermiT [MSH07], FaCT++ [TH06], and Pellet [SP06]. It should be noted that all these different formulations (or implementations) of tableau-based decision procedure are more or less equivalent and their differences are inessential.

In general, tableau-based decision procedures work on labeled graphs (or labeled trees) whose nodes stand for individuals of an interpretation. Each node is labeled with a set of concepts, namely those it is assumed to be an instance of. Concepts (in each tableaux node's label) are usually assumed to be in *negation normal form*[1] [BS01]. Each edge between two tableaux nodes is labeled with a role or a set of roles, namely those that hold between the corresponding individuals. The completed and saturated graph structure, in the form of a completion tree (or forest), serves as a partial description of a model whose individuals correspond to tableaux nodes, and whose binary relations are taken

[1]Abbr. NNF, negation signs are recursively pushed inward until they appear only in front of atomic names, normally by use of the De Morgan's law and the duality property. For example, the well known disjunctive normal form (DNF) and conjunctive normal form (CNF) belong to the NNF.

from the (labels of the) tableaux edges.

To test the satisfiability of a concept C, a *tableau-based algorithm* starts (usually in a top-down manner) with an instance x_0 of C, i.e., the root of a graph with C as an element of its label (written as $\mathcal{L}(x_0) = \{C\}$). The algorithm breaks down concepts in the label of each node syntactically (according to a set of the so-called *tableaux expansion rules*), infers new constraints (a.k.a. subgoals) to be further satisfied by possibly generating new tableaux nodes (and their labels) or by merging existing ones, and ultimately constructs a saturated relational structure that could serve as a pre-model for C. If no such structure can be found, then C is said to be "unsatisfiable". For a tableaux procedure to be correct, soundness and completeness are critical issues to consider.

The first component of a tableau-based decision procedure is the *tableaux expansion rules.* There exists at least one *expansion rule* for each syntactic construct. If no more completion rule is applicable, i.e., no application of any rule can change the tableau-structure (graph or tree) under construction, then the final *tableaux structure* is said "completed", hence called a *completion.* A completion free of *clash* serves as a pre-model that can be mapped into a model for C.

The second important notion is the so-called *clash triggers*, which are used for detecting obvious contradictions in labels. There are several *clash triggers*, each of which is usually designed for a specific restriction type and is used in specific inference algorithms. The most fundamental *clash trigger* is the *name clash*, which recognizes situations in which a certain concept name and its negation occur in the label of one common node. For an arbitrary concept name A and an individual x, this *name clash* is triggered whenever $\{A, \neg A\} \subseteq \mathcal{L}(x)$, a situation in which x is inconsistent because x can not be interpreted as an instance of A and an instance of $\neg A$ at the same time without causing a contradiction.

When *number restrictions* are considered, the *clash triggers* are usually not specifiable in the above form. For example, $\{\exists^{\leq 1} R.A, \exists^{\leq 2}.R.\neg A, \exists^{\geq 4} R.B\}$ is inconsistent. This inconsistency cannot be detected unless by a combinatorial reasoning over a number of potential combinations. Consider $\{\exists R.A,\ \exists R.B,\ \exists R.C,\ \exists R.D,\ \exists^{\leq 3} R.\top\}$ as a second example, its consistency depends on if there is a consistent partitioning of $\{A, B, C, D\}$ of size less than or equal to 3. When *nominals* are considered, it is necessary to ensure that a *nominal* can only be counted once (in the interpretation), for example

$\exists^{\geq 3}R.\{o\}$ is not satisfiable because $\{o\}$ can only be interpreted as a singleton (see Chapter 1 for semantics). It is very clear that the *integer linear programming* technique can be used for checking (and reasoning about) the existence of a specific combination (see Chapter 6).

To guarantee termination, tableau-based decision procedures need to stop explicitly, e.g. by the *caching* or the *blocking techniques* or simply rely on the *sub-expression property*. It will be clear that for description logics with inverse roles, the termination condition is usually more complex than their counterparts having no inverse roles.

Tableau procedures non-deterministically build and work on labeled (tree-like or graph-like) structures, if there exists a run with a completion free of clashes, then the input concept is *satisfiable*, otherwise *unsatisfiable*. The non-determinism has to be circumvented for a better implementation, usually by the *global sub-tableaux caching technique*, the optimized *backtracking techniques* and some heuristics for *branch selection*. In summary, the don't-care-non-determinism of tableau expansion rules is good for the proof of soundness and completeness of tableau algorithms; the don't-know-non-determinism of disjunctions and number restrictions increases the complexity in searching for a pre-model of the concept.

A.2 General Optimizations

It is well-known from practical experiences that the *tableau-based decision procedure* can achieve better run-time performance by applying a wide range of *optimization techniques*[2]. In the following, the focus will be on language-independent optimization techniques, i.e., those techniques not relying on specific language features. This kind of techniques could be considered as general techniques. We introduce some fundamental reasoning techniques that underly the tableau procedure of description logics, and also discuss the possible problems they might have. A comprehensive and impartial review on optimization techniques investigated so far is a complex task hard to attain. For another review we refer to [THPS07].

[2]All optimization techniques are designed to improve the run-time performance of the *tableau procedure*. However, not all of them are optimizing the tableau procedure directly. One simple criteria about whether a specific technique optimizes the tableau procedure is to see if this technique will be called in the execution of the tableau procedure. In this vein, *internalization* or *absorption* are techniques not for optimizing *tableau procedure*, but *lexical normalization and simplification, unfolding, caching, branching and backtracking* could be. The relationship is very much like a problem concerning the road, the cars and the transportation (car v.s. tableau procedure, road v.s. Tbox, transportation v.s. DL system).

A.2.1 Concept Unfolding

Given a KB $\mathcal{T}$ and a concept C whose satisfiability is to be tested w.r.t $\mathcal{T}$, it is possible to eliminate from C all concept names occurring in $\mathcal{T}$ using a recursive substitution procedure called the *unfolding* of concept names [BCM$^+$03].

For a non-primitive concept name A defined in $\mathcal{T}$ by an axiom $A \equiv D$, the procedure simply substitutes (also called unfolds) A with D wherever it occurs in C, and then recursively unfolds D in the same manner.

For a primitive concept name A defined in $\mathcal{T}$ by an axiom $A \sqsubseteq D$, the substitution to A is by $A' \sqcap D$, where A' is a new concept name which represents the "primitiveness" of A, i.e., the unspecified characteristic that differentiates A from D. Using $unfold(C, \mathcal{T})$ to denote the concept C after unfolding w.r.t $\mathcal{T}$, we have $\mathcal{T} \models C \sqsubseteq D \iff \emptyset \models unfold(C, \mathcal{T}) \sqsubseteq unfold(D, \mathcal{T})$, i.e., to test the satisfiability of a concept C w.r.t $\mathcal{T}$, it is equivalent to a satisfiability test of the unfolded concept w.r.t an empty Tbox.

Generally there are two problems regarding *concept name unfolding*[3]: (1) a unrestricted recursive unfolding could possibly produce a resulting concept expression of exponential size; (2) unfolding would not be possible if $\mathcal{T}$ contains (2.1) multiple definitions for one concept name, e.g. $\{A \equiv C, A \equiv D\} \subseteq \mathcal{T}$, or (2.2) axioms like $\exists^{\geq 3} R \sqsubseteq \exists S.D$. Axioms that can not be transformed into the *unfoldable* form, a format amenable for the unfolding process, will be *internalized* during the execution of tableaux procedures[4].

The *unfolding* of a concept name into its definition according to a given unfoldable Tbox is a basic process in the *tableau-based procedure*, however, unrestricted unfoldings might result in an expanded label of exponential size. The *lazy unfolding* technique addresses this problem by imposing a restriction on the unfolding process, i.e. to replace only those occurrences of names outside the scope of any modal operators with their definitions. In other words, *lazy unfolding* does not expand the occurrences of concept names which are behind $\exists$ or $\forall$.[5] With this restriction, *lazy unfolding*

[3] *Label retaining* is necessary when the *unfolding process* is dealing with the *concept introduction axioms*. To be precise, the RHS must be added rather than replacing the LHS. For example, for an *concept introduction axiom* like $A \sqsubseteq C$, when it comes to unfolding A, A should be retained with its RHS C rather than be replaced.

[4] The absorption technique tries to transform most of the non-unfoldable axioms into unfoldable axioms in an equivalence-preserving way.

[5] I.e., lazy unfolding only applies to those names in the *propositional part* of a label. The *modal parts* (incl. those

will not result in an exponentially sized label.

The *unfolding* and *lazy unfolding* are dependency elimination processes. It is also considered as a lightweight *constraint propagation* mechanism local to each node of the *tableau-structure*. In a sense, *lazy unfolding* is a requisite rather than an optimization. But a combination of *lazy unfolding* with *label retaining* is considered an optimization because it has been observed that clashes can usually be detected earlier [Hor03]. For discussions on keeping the LHS when the unfolding process expands it to its RHS definition, see [BHNP94, Haa00, Hor03].

A.2.2 Normalization and Simplification

The *normalization and simplification* mechanism [Hor95, Mas00, Hor03] provides a simple way for structure sharing of concept expressions for a knowledge base. Comparatively speaking, it is a simplified and lightweight version of those techniques identifying syntactic equivalence, contradictions, and tautologies, and so on.

The *lexical normalization* produces the syntactic *norm form* by using DeMorgan's law and the duality between existential and universal restrictions, e.g., converting disjunctions to conjunctions, existential restrictions to universal restrictions, at-most number restrictions to at-least number restrictions, and so on. For example, $\neg D \sqcup \neg C$ is expressed/stored as $\neg \sqcap \{C, D\}$ where a conjunction is treated as a set so that reordering or repeating the conjuncts does not effect equivalence. Likewise, it is possible to have $\exists R.C \rightarrow \neg\forall R.\neg C$ and $\exists^{\leq n} R.C \rightarrow \neg\exists^{\geq(n+1)} R.\neg C$. The *lexical simplification* simplifies $\exists R.\bot$ to $\bot$, and $\exists^{\geq 0}R.C$ to $\top$, and so on. For *nominals*, e.g., $\exists^{\geq 2}R.(\{o\} \sqcap C)$ can be simplified to $\bot$, $\exists^{\leq 3}R.(\{o\} \sqcap C)$ to $\top$. Further in $\mathcal{SHOQ}$, $\exists^{\geq 3}R.(\{o\} \sqcup C)$ can be converted to $(\exists R.(\{o\} \sqcap \neg C) \sqcup \exists R.(\{o\} \sqcap C)) \ \sqcap\ \exists^{\geq 2}R.(C \sqcap \neg\{o\})$; similarly $\exists^{\leq 3}R.(\{o\} \sqcup C)$ can be converted to $(\exists R.(\{o\} \sqcap \neg C) \sqcup \exists R.(\{o\} \sqcap C)) \ \sqcap\ \exists^{\leq 2}R.(C \sqcap \neg\{o\})$. This simplification of *number restrictions* in the presence of *nominals* relies on a notion of *constraint fine-tuning* (see Chapter 6). To the best knowledge of the author of this thesis, there is no similar discussion available in the published literature from the optimization field.

The elimination of redundancy and the sharing of syntactically equivalent structures may lead to a compact KB. It complements *lazy unfolding* and improves early clash detection [Hor03]. However,

generated by lazy unfolding if any) lies behind each $\forall$ or $\exists$ are handled only by the tableaux completion rules.

some criticism says that for very unstructured KBs there may be no benefit and it might even slightly increase the size of a KB.

A.2.3 Internalization

By using "internalization", the whole Tbox can be "internalized" into a single concept, i.e., it is possible to build a concept that expresses all the axioms of the Tbox, $C_\mathcal{T} \equiv \sqcap_{(A_i \equiv B_i \in \mathcal{T})}((A_i \sqcup \neg B_i) \sqcap (\neg A_i \sqcup B_i)) \quad \sqcap \quad \sqcap_{(A_j \sqsubseteq B_j \in \mathcal{T})}(\neg A_j \sqcup B_j)$. Testing concept D w.r.t the Tbox $\mathcal{T}$ is equivalent to testing a single concept $D \sqcap C_\mathcal{T} \sqcap \forall U.C_\mathcal{T}$, where U is a transitively closed super-role of all the roles in $\mathcal{T}$ (or equivalently, U can be considered as the universal modality).

The notion of *internalization* [CGLN01, BCM+03] has its origin from the research in *propositional dynamic logics* (PDLs) [CGLN01]. The notion of *internalization* is more of theoretical interests than of practical interests. It is usually used, for example, to demonstrate that the *concept satisfiability* w.r.t a Tbox is in the same complexity class as the *pure concept satisfiability* problem. In practice, however, if some axioms of a Tbox can not be converted to a form amenable for concept *name unfolding*, usually they are required to be internalized.

The problem with *internalization* is that the concept resulting from the *internalization* process might contain too many disjunctions, which is known to increase non-determinism in tableau-based reasoning, thus causes inefficiency in reasoning.

A.2.4 Branching

In the presence of a *disjunction*, possibly there are several branches open for search. Pruning away fruitless choices beforehand is a key to achieve a good run-time performance. Obviously, the exhaustive search of every branch should be avoided. However, it is necessary to keep a balance, say, not sacrificing the completeness of the underlying decision procedure when trying to avoid an exhaustive exploration of the search space. The *semantic branching technique* has been adapted from a technique commonly used in the SAT community, namely, the Davis-Putnam-Logemann-Loveland (DPLL) procedure [Fre95, Hor03].

When expanding the label of a node x, *syntactic branching* works by choosing an unexpanded disjunction $(C_1 \sqcup ... \sqcup C_n)$ in $\mathcal{L}(x)$ and searching the different models obtained by adding each of the disjuncts $C_1, ..., C_n$ to $\mathcal{L}(x)$. As the alternative branches of the search tree are not disjoint, there is nothing to prevent the recurrence of an unsatisfiable disjunct in different branches.

The *semantic branching* works by choosing one disjunct D from one of the unexpanded disjunctions in $\mathcal{L}(x)$. The two sub-trees obtained by adding either D or $\neg D$ to $\mathcal{L}(x)$ are then searched. The two sub-trees are strictly disjoint, there is no possibility of wasted search as in syntactic branching. In contrast to *syntactic branching*, where redundant search space may be repeatedly explored, *semantic branching* uses a splitting rule which replaces the original problem by two disjoint sub-problems. Usually, the *semantic branching* technique is supported by various techniques to speed up the search [Haa00]: (1) a *look-ahead algorithm* or a *constraint propagator* tries to reduce orders of magnitude of the open search space; (2) various *heuristics*, e.g., MOMS (Maximum number of Occurrences in disjunctions of Minimum Size) [Fre95], are used to select the next open disjunct in a disjunction, and a dynamic selection scheme is often employed.

A.2.5 Backtracking

"Intelligent" *branching and backtracking* in the case of "dead-ends" is a key to system performance. Both select the "right" open sub-problem for further exploration. *Syntactic branching* for a disjunction might suffer from redundant sub-problems; the *backtracking* mechanism supported by compilers and operating systems is chronological in nature and suffers from a local *thrashing* phenomenon [Hor03]. Both are not efficient for implementation. The *conflict-directed backjumping* is an optimized backtracking technique that enables the decision procedure to backtrack to the relevant part (clashing participants) rather than chronological backtracking, see [Gin93, Haa00, Hor03]. A good backtracking subsystem relies heavily on the efficiency of the underlying data structure, the *dependency maintenance* and the *inconsistency propagator* [Haa00].

A.2.6 Axiom Transformation

When reasoning with description logic knowledge bases which contain a large number of *general concept inclusions* (GCIs), performance is the key concern in real applications. The *absorption technique* is a general term used for techniques that transform a set of given GCIs into another form so that much of it is *unfoldable*. This transformation makes tableau-based reasoning less dependent on the *internalization technique*, thus the GCI absorption improves run-time performance of the tableau procedures and has been a central research issue for DL KBs, see [Hor95, HT00, Haa00, HM00b, Hor03].

In contrast to the conventional absorption centered around concept names (the unary predicates), there is another notion about absorption on role names (the binary predicates), which originated in [Haa00, HM00b]. Later, this notion was slightly extended and was called the *domain and range absorption* in [TH04].

There is a recent trend in utilizing certain properties of *inverse roles* for GCI absorption, see [HW06, SGP06]. Different from previous absorption techniques, [HW06] is the first to show the new idea to integrate the pattern matching algorithms into tableau-based procedures, very likely this would lead to a new framework that allows for unfolding of a conjunction of concept names rather than unfolding over a single concept name.

In summary, there are three kinds of absorption methods with different degree of emphasis on concept names, role names, or conjunctions.

A.2.7 Blocking

The tableau-based decision procedure checks the satisfiability of an input *concept formula* by trying to construct a saturated (a.k.a. completed) and clash-free *tableau structure*. Its termination should be guaranteed. In most cases, for example for *unfoldable acyclic Tboxes* or empty *Tboxes*, the tableaux procedure (for testing concept satisfiability) would naturally terminate (due to syntactical restrictions). However, for complex situations (e.g., expressive DLs), especially when *cyclic Tboxes* and *general concept inclusions* (GCIs) are considered, the decision procedure might not terminate. The *blocking* mechanism is designed to ensure the termination of a tableau-based decision procedure[6]. The termination of a tableau-based decision procedure should be guaranteed and this is done by detecting cyclic computations and preventing rules from running into cycles. The difference of various *blocking techniques* are characterized by their corresponding *blocking conditions*[7].

[6]In this sense, *blocking* falls in the requisite category.

[7]This dissertation introduced a new "blocking" technique that complements those well-known "blocking techniques" introduced in the literature.

Typical blocking techniques in the literature include: (1) subset blocking; (2) equality blocking; (3) pairwise blocking; and (4) dynamic blocking. The *blocking* mechanisms used [Baa90, HST98, HST99b, HM00a, HM00b, Haa00] in Abox reasoning generally do not allow *old individuals* being blocked by other individuals.

In brief, the *blocking techniques* characterize the run-time conditions the tableau structure under construction must meet for the decision procedure to stay sound[8]. All blocking techniques guarantee the termination of each *trace* in the tableau structure. It is observed that earlier termination of a trace is desirable for a better performance.

A.2.8 Caching Technique

Sub-tableaux caching is a proof reuse technique. The effect of exploiting already computed *subsumption relations* (i.e. *unsatisfiability* results) was carefully investigated in [BHNP94]. That is one of the earliest practical work that proposed the idea of using *caching* for the tableau-based procedures in DLs. In real world problems, it was observed later that the non-subsumption relations (i.e. *satisfiability* results) prevails over subsumption relations (i.e. *unsatisfiability* results). According to some statistics, the ratio of satisfiable problems v.s. unsatisfiable problems is 95% to 5% or more. And there might be such a problem that the number of distinct sub-problems are few but this small number of sub-problem patterns repeat numerous times and each of them is satisfiable. For problems of this kind, caching unsatisfiability is impractical. As pointed out in [Haa00, HM00a], caching both *satisfiable* and *unsatisfiable* intermediate computation results is a necessary prerequisite to prove the (in)consistency of many concepts terms.

A theoretical explanation of the usefulness of the *global sub-tableaux caching* was made by Donini and Massacci in [DM00] (and by Giacomo, Donini and Massacci in [GDM96]). They formally introduced notions of *nogood list* and *visited list.* These two notions generally characterized the caching technique. In [DM00], they presented a tableau-based algorithm using a permanent caching of (*inconsistent*, or *unsatisfiable*) sub-results and a somewhat restricted *caching* of satisfiable *witnesses.* A worst-case optimal ExpTime tableau-based algorithm matching the lower bound was achieved for the description logic $\mathcal{ALC}$ w.r.t general Tbox. Their work theoretically manifested the power of

[8]In this thesis, all blocking techniques are viewed as special cases of the *global sub-tableaux caching technique.*

caching and the necessity of using it in expressive description logics.

There are many successful applications of the caching technique in real description logic systems such as RACER [Haa00, HM00a, HMT01], FaCT and DLP [HPS98], and two kinds of strategies are generally employed: *(pseudo) model caching* [Hor95] which reuses pseudo models of concepts, and *subtableaux caching* which reuses sub-problems [DM00, Haa00, HM00a]. Empirical tests show that global *sub-tableaux caching* is very effective for many Abox consistency problems [Haa00, P.167], e.g. TANCS benchmark (TANCS'2000 comparison problems). Refined *caching technique* [HM00b] can have maximum performance by using both an *equal cache* and a *subset* as well as a *superset cache*. Also, *pseudo model merging* [Hor95, Haa00, HMT01] extends the equal/subset/superset comparison (look-up) to checking of a combination of several cached entries, more complex operations on cached entries than simple cache look-ups. Recently there is a revived trend in using the *pseudo model merging* technique on arbitrary graphs, e.g., [SGP06] showed that a combination of *nominal absorption* and nominal-based graph model merging led to a successful consistency checking of the Wine ontology (see www.w3c.org). A recent work [GN07] also discussed how a global tableaux caching technique could also work on "graph-based models".

It is always safe to cache the inconsistency of a sub-tableau [Haa00]. However, caching non-inconsistent sub-tableaux is not trivial any more because it depends on the context [Haa00, HM00a], i.e., in the case there is a blocking witness the dependency between the blocking witness and the current (sub)problem should be carefully handled to ensure correct behavior. One solution is provided in [HM00a], i.e., a dependency tracking mechanism for cache entries is implemented in RACE (precursor of RACER). Once the system detects the inconsistency of a concept (or constraint system) on which a cached entry is dependent, the corresponding cache entries are (recursively) removed. The *blocking technique*, originally designed for the termination of a trace in one branch of the tableau tree, could be viewed as a special *caching technique* working only on one trace. The *witness* in [DM00] is more general than the notion of blocking nodes presented in the literature.[9] Recently, Rajeev and Linh have improved the work of [DM00], readers are referred to [GN07] for more information.

[9]Blocking only happens on one trace, but the notion of a witness (i.e., element of the *visited list*) in [DM00] is not restricted to one trace.

A.3 Reasoning About Inverse Roles

The *inverse role* is regarded as one of the convenient, even indispensable, constructs of expressive description languages. For example, it has been recognized long ago in the DL community that one is able to use "inverse roles" to "indirectly" describe a concept like "a person has at least five siblings" as $\exists has\_child^{-}.\exists^{\geq 6} has\_child.\top$, by reusing available roles instead of resorting to new roles. For more examples, readers are referred to Chapter 1 (Section 1.4).

The convenience in using inverse roles, however, introduces a number of problems, inefficiency in reasoning in particular, for description logics. The first problem is very general and is the fundamental one: computation is two-way rather than one-way, a notorious phenomenon also identified in other logics such as converse-PDL[10]. The second problem is that a label can not be explicated at the time it is created. The third problem is that the global sub-tableaux caching techniques[11] well designed for $\mathcal{ALC}$ might be unsound for $\mathcal{ALCI}$. The fourth problem is that the so-called *pre-completion technique* is unsound for Abox reasoning in the presence of *inverse roles.* The fifth problem is that inconsistency propagation and conflict-directed backjumping are less effective in the presence of *inverse roles.*

The current trend for reasoning issues of DLs with inverse roles takes an architectural approach, e.g., the "ToDo List" architecture as proposed in [THPS07]. This approach is typical of using "dynamic double blocking" for the termination of its underlying tableaux algorithm, and lacks a powerful inconsistency propagation capability. As pointed out in [Tob01], algorithms based on "dynamic double blocking" are nondeterministic in nature and would take double exponential cost in the worst case.

A.4 Reasoning About Number Restrictions

Number restrictions are concept constructors that are available in almost all implemented description logic systems. They allow to restrict the number of role-successors (role-neighbors) of an individual with respect to a given role. For example, if *has_child* is an atomic role and *person* is a concept,

[10]I.e., converse propositional dynamic logic, denoted as CPDL [Gia96]. PDL is originally a logic for program verification and now is a general logic about action.

[11]Also other well-designed caching related techniques turn unsound in the presence of *inverse roles.*

then we can describe "persons having at least 3 children" by $person \sqcap (\exists^{\geq 3} has\_child.person)$.

- [HB91] provided a worst-case ExpTime tableau-based decision procedure based on unary coding of numbers.
- [Gia95] used "reification" to translate *qualified number restrictions* into *functional restrictions.* Again it is based on an unary coding of numbers, and this translation gives a new KB with a polynomial expansion of the original KB.
- Later, Tobies [Tob99] gave a PSpace algorithm based on a notion of "counters" for the pure concept satisfiability problem in $\mathcal{ALCQ}$ and $\mathcal{ALCQI}$.
- A later work [BML+05] provided a PSpace algorithm for $\mathcal{ALCQO}$ w.r.t. acyclic Tboxes. Based on a similar notion of "counters", the algorithm of [BML+05] is PSpace even when numbers have a binary coding.
- Instead of providing tableau-based algorithms, [Tob01] provided a method to construct a looping tree automaton for the concept satisfiability of a concept w.r.t. GCIs. Based on the introduction of "limiting functions", this construction is exponential to the size of the input when numbers are in binary coding.
- The algebraic method appeared in [OK99] and [HM01a]. However, the algebraic method is not known to work for DLs with both *qualified number restrictions* and *inverse roles.* In Chapter 6, we have presented a worst-case ExpTime tableau-based decision procedure, with an improved upper bound than the automata-based approach [Tob01], based on the algebraic method.

Appendix B

Empirical Results

We have implemented the elimination of inverse roles as presented in Chapter 3 and have evaluated its practicality. All satisfiability tests were performed with RacerPro 1.9.0 on a Pentium PC with 3.5 GB memory. The tested ontologies (in the ontology language OWL) were also converted on the same machine. Note that the expressivity of the original ontologies corresponds to the DL $\mathcal{ALCI}$.

Table B.1 shows some empirical results (coherence check only), where the time indicated is the average of 5 independent runs of the conversion system. It can been seen that although more time is spent for testing Tbox coherence for the converted versions of the first two KBs, the performance is still acceptable since the KB sizes after conversion are less than five times larger than original. Evidently, for the UML ontology the runtime after conversion is quite impressive. Besides, we have also divided the UML ontology into two sub-ontologies, both of which, if converted, require less time to compute the satisfiability of all the concepts. Dramatic increase of performance is shown in the last case, where the ontology contains one major class extracted from the ontology "revised-9-alci".

KB Name	Coherence Check (original Tbox)	**Coherence Check (converted Tbox)**	No. of Axioms (original/converted)
galen-ir1-alci-new1	9.141	84.625	4645/5495
galen-ir2-alci-new1	9.549	76.156	4666/5508
uml-no-max-min-new4	timeout after 1 hour	0.110	524/739
revised-9-alci (partial)	timeout after 1 hour	3.297	3077/3099

Table B.1: Experimental results (all times are given in seconds)

Index