Optimizing Inconsistency-tolerant Description Logic Reasoning

Mokarrom Hossain and Wendy MacCaull

Department of Mathematics, Statistics and Computer Science
St. Francis Xavier University, Antigonish, NS, Canada
{mokarrom.hossain, wmaccaul}@stfx.ca

Abstract

The study of inconsistency-tolerant description logic reasoning is of growing importance for the Semantic Web since knowledge within it may not be logically consistent. The recently developed quasi-classical description logic has proved successful in handling inconsistency in description logic. To achieve a high level of performance when using tableau-based algorithms requires the incorporation of a wide range of optimizations. In our previous work, we developed a naive inconsistency-tolerant reasoner that can handle inconsistency directly. Here, we investigate a set of well known, state-of-the-art optimization techniques for inconsistency-tolerant reasoning. Our experimental results show significant performance improvements over our naive reasoner for several well-known ontologies. To the best of our knowledge, this is the first attempt to apply optimizations for inconsistency-tolerant tableau-based reasoning.

1 Introduction

Since the environment of the Semantic Web (SW) is open, constantly changing and collaborative, it is unreasonable to expect knowledge bases (KBs) to be always logically consistent [1]. With the explosive growth of the SW the probability of introducing inconsistencies in ontologies is increasing day by day due to ontology merging, ontology evolution, ontologies created using machine learning or data mining, migration from other formalisms, modeling errors, knowledge from distributed sources, etc. [2]. The Web Ontology Language (OWL) is a family of knowledge representation languages for authoring ontologies in SW applications. The family of description logics (DLs), decidable fragments of First Order Logic, is the logical foundation of OWL. Classical DLs are unable to tolerate inconsistencies occurring in ontologies due to the *principle of explosion* [3]. Therefore, it is important to study the ways of dealing with inconsistencies in DL based ontologies [4, 5, 6].

There are several approaches to deal with inconsistencies. These approaches may be roughly categorized into two different classes. One class assumes that inconsistencies indicate errors in data. The main view of this class is that an ontology should not be inconsistent, and thus researchers try to eliminate inconsistency from an ontology in order to obtain a consistent ontology [7]. In this approach the idea, therefore, is, first detect and then, repair, inconsistencies. This class of approaches could be called "removing inconsistencies". In this approach, inconsistency could be removed by pinpointing the part of the ontology which causes inconsistencies and removing or weakening axioms in these parts to restore consistency. The main assumption of the other class of approaches is that inconsistency is a natural phenomenon of realistic data and should be tolerated in reasoning [3, 8, 9]. In this class of approaches, inconsistency is not simply avoided, rather, the idea is to employ non standard reasoning methods in order to obtain meaningful information from an inconsistent KB. This class of approaches could be called "living with inconsistencies". Though inconsistency in small ontologies may be dealt with using

the former approach, the latter one is better for large and complex ontologies [10]. Moreover, in the first class of approaches, we may lose useful information during the process of removing inconsistencies [3]. In this work we focus on the second class of approaches.

In the last decades, a number of researchers have extended DL in different ways in order to cope with inconsistency [3, 8, 10]. Nguyen et al. studied three-valued DL based on Kleene's three-valued semantics [8]. Another three-valued DL is paradoxical DL [11] based on Priest's paradoxical semantics. Kaminski et al. presented a paraconsistent version of the three-valued semantics for hybrid knowledge bases which allows full paraconsistent reasoning [12]. Indeed, three-valued DLs are usually appropriate for handling the inconsistency but not the incompleteness of a KB. In many KBs, information is not only inconsistent but is also incomplete. Four-valued DL, based on four-valued logic [13], is studied by Ma et al. in [3] and has received a lot of attention. Though it can handle both inconsistent and incomplete KBs, it is not widely accepted due to its weak inference power [10]. For example, four-valued DL does not fully support a few important properties about inference such as *modus tollens* (MT), *disjunctive syllogism* (DS), *resolution*, etc. Recently, Zhang et al., proposed a paraconsistent version of DL, called QCDL, in [10], based on *quasi-classical propositional logic* (QC-logic) originally presented in [14]. Weak inference power, one of the common problems of the family of paraconsistent logics, is overcome by QCDL. Most inference rules like MT, DS are valid in QCDL.

In the previous work [15], we presented a sound, complete, and decidable tableau algorithm for QCDL and implemented a tableau based paraconsistent reasoner called QC-OWL based on this algorithm. However, we did not address any optimization techniques in QC-OWL. Most modern reasoners implement a set of state-of-the-art optimization techniques and these techniques are the keys to the enhanced performance of a modern tableau-based reasoner. Here, we extend a set of state-of-the-art optimization techniques for classical tableau based reasoner, namely normalization and simplification, unfolding, absorption, semantic branching, and dependency directed backtracking, to paraconsistent reasoning (background and motivation of these well-known techniques may be found in [16]). We implement those techniques on top of QC-OWL and compare the performance with that of our previous work, where no optimizations were addressed.

The rest of this paper is organized as follows: Section 2 briefly introduces the syntax and semantics of QCDL, Section 3 describes the optimization techniques, Section 4 presents the implementation and evaluation of those optimizations, Section 5 outlines some related works and future research directions, and finally, Section 6 concludes the paper.

2 Preliminaries

In this section we briefly introduce the syntax and semantics of QCDL. We focus on an expressive DL fragment, $\mathcal{SHIQ}$, and call it quasi-classical $\mathcal{SHIQ}$ (in short, QC-$\mathcal{SHIQ}$). We assume that the readers are familiar with the basic DL formalism; for more comprehensive background knowledge of DLs and QCDL, the reader is referred to [10, 15, 16].

2.1 QCDL syntax and semantics

The syntax and semantics of QCDL is presented in [10] by extending the semantics of quasi-classical logic (QC-logic) proposed in [14]. The syntax of QC-$\mathcal{SHIQ}$ is slightly different from the syntax of classical $\mathcal{SHIQ}$. In QC-$\mathcal{SHIQ}$, the negation of a concept, i.e., $\neg C$, is taken as a different concept from C, rather than the complement of C. QC-negation, denoted by

$\overline{C}$, is used to represent the complement (set-theoretic complement) concept of C [10]. For a comprehensive background and motivation of QC-negation, the reader is referred to [10, 17].

Let N_C, N_R and N_I be non-empty and pair-wise disjoint sets of concept names, role names, and individual names, respectively. Let $\mathbf{R}$ be a set of role names with a subset $\mathbf{R}_+ \subseteq \mathbf{R}$ of transitive role names. The set of roles is $\mathbf{R} \cup \{R^- \mid R \in \mathbf{R}\}$. The function $Inv(.)$ is defined on roles such that $Inv(R) = R^-$ and $Inv(R^-) = R$, where R is a role name.

Let $R_1, R_2 \in \mathbf{R}$. A role axiom is a role inclusion of the form $R_1 \sqsubseteq R_2$. An RBox or role hierarchy $\mathcal{R}$ is a finite set of role axioms. For a role hierarchy $\mathcal{R}$, the relation $\sqsubseteq$ is defined to be the transitive-reflexive closure of $\sqsubseteq$ on $\mathcal{R} \cup \{Inv(R) \sqsubseteq Inv(S) \mid R \sqsubseteq S \in \mathcal{R}\}$. A role R is called *sub-role* (respectively, *super-role*) of a role S if $R \sqsubseteq S$ (respectively, $S \sqsubseteq R$). A role S is *simple* if it is neither transitive nor has any transitive sub-roles.

The set of complex concepts is the smallest set such that

- each concept name $A \in N_C$ is a concept;
- if C, D are concepts, R is a role, S is a simple role, and n is a nonnegative integer, then $C \sqcap D \mid C \sqcup D \mid \neg C \mid \overline{C} \mid \forall R.C \mid \exists R.C \mid \geqslant nS.C \mid \leqslant nS.C$ are also concepts.

A general concept inclusion (GCI) is an expression in the form $C \sqsubseteq D$, where C, D are concepts. A TBox is a finite set of GCIs. An assertion is of the form $C(a)$ (concept assertion), $R(a, b)$ (role assertion), or $a \neq b$ (inequality assertion), where $a, b \in N_I$. An ABox contains a finite set of assertions. A QC-$\mathcal{SHIQ}$ KB is a triple $\mathcal{K} = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ where $\mathcal{T}$, $\mathcal{A}$, and $\mathcal{R}$ are the TBox, ABox, and RBox, respectively.

Two types of interpretations, called *weak interpretations* and *strong interpretations*, are proposed by QCDL semantics. The former is the reformulation of that for four-valued logic. Before introducing these two types of interpretations, we first define a notion called *base interpretations* [10].

A *base interpretation* $\mathcal{I}$ is a pair $(\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ where the domain $\Delta^{\mathcal{I}}$ is a set of individuals and the assignment function, $.^{\mathcal{I}}$, assigns:

- each concept name A to an ordered pair $\langle +A, -A \rangle$ where $\pm A \subseteq \Delta^{\mathcal{I}}$;
- each role R to an ordered pair $\langle +R, -R \rangle$ where $\pm R \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$;
- each inverse role R^- to an ordered pair $\langle +R^-, -R^- \rangle$ where $\pm R^- = \{(y, x) \mid (x, y) \in \pm R\}$;

Note that each base interpretation maps an object X, when $X \in \{A, R, R^-\}$, to a pair of sets of elements, unlike classical interpretation where an object is mapped to a set of elements. $+X$ and $-X$ are not necessarily disjoint. Intutively, $+X$ is the set of elements known to be in X while $-X$ is the set of elements known to be not in X.

A *weak interpretation* $\mathcal{I}$ is a base interpretation $(\Delta^{\mathcal{I}}, .^{\mathcal{I}})$ such that the assignment function $.^{\mathcal{I}}$ satisfies the following conditions [15], where $\#M$ denotes the cardinality of a set M:

$\top^{\mathcal{I}} = \langle\, \Delta^{\mathcal{I}},\, \emptyset \,\rangle$; $\quad \bot^{\mathcal{I}} = \langle\, \emptyset,\, \Delta^{\mathcal{I}} \,\rangle$;

$(\neg C)^{\mathcal{I}} = \langle\, -C,\, +C \,\rangle$; $\quad (\overline{C})^{\mathcal{I}} = \langle\, \Delta^{I} \setminus +C,\, \Delta^{I} \setminus -C \,\rangle$;

$(C \sqcap D)^{\mathcal{I}} = \langle\, +C \cap +D,\ -C \cup -D \,\rangle$; $\quad (C \sqcup D)^{\mathcal{I}} = \langle\, +C \cup +D,\ -C \cap -D \,\rangle$;

$(\forall R.C)^{\mathcal{I}} = \langle\, \{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x, y) \in +R \text{ implies } y \in +C\},$
$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in +R \text{ and } y \in -C\} \,\rangle$;

$(\exists R.C)^{\mathcal{I}} = \langle\, \{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in +R \text{ and } y \in +C\},$
$\{x \in \Delta^{\mathcal{I}} \mid \forall y \in \Delta^{\mathcal{I}} : (x, y) \in +R \text{ implies } y \in -C\} \,\rangle$;

$(\geqslant nS.C)^{\mathcal{I}} = \langle\, \{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} : (x, y) \in +S \text{ and } y \in +C\} \geqslant n\},$
$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} : (x, y) \in +S \text{ and } y \in (\Delta^{\mathcal{I}} \setminus -C)\} < n\} \,\rangle$;

$(\leqslant nS.C)^{\mathcal{I}} = \langle\, \{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} : (x, y) \in +S \text{ and } y \in (\Delta^{\mathcal{I}} \setminus -C)\} \leqslant n\},$
$\{x \in \Delta^{\mathcal{I}} \mid \#\{y \in \Delta^{\mathcal{I}} : (x, y) \in +S \text{ and } y \in +C\} > n\} \,\rangle$.

Let $\mathcal{I}$ be a weak interpretation. A *weak satisfaction relation*, denoted by $\models_w$, is defined as follows: $\mathcal{I} \models_w C(a)$ if $a^{\mathcal{I}} \in +C$; $\mathcal{I} \models_w R(a,b)$ if $(a^{\mathcal{I}},\ b^{\mathcal{I}}) \in +R$; $\mathcal{I} \models_w C \sqsubseteq D$ if $+C \subseteq +D$; $\mathcal{I} \models_w R_1 \sqsubseteq R_2$ if $+R_1 \subseteq +R_2$; and $\mathcal{I} \models_w a \neq b$ if $a^{\mathcal{I}} \neq b^{\mathcal{I}}$; where $X^{\mathcal{I}} = \langle +X, -X\rangle$ for any $X \in \{C,\ D,\ R,\ R_1,\ R_2\}$ [15].

A *strong interpretation* is as similar to a weak interpretation except that the conjunction and disjunction of concepts are interpreted as follows [15]:

$(C \sqcap D)^{\mathcal{I}} = \langle +C \cap +D,\ \ (-C \cup -D) \cap (-C \cup \overline{+D}) \cap (\overline{+C} \cup -D)\rangle$;

$(C \sqcup D)^{\mathcal{I}} = \langle (+C \cup +D) \cap (\overline{-C} \cup +D) \cap (+C \cup \overline{-D}),\ \ -C \cap -D\rangle$.

The definition of *strong satisfaction relation*, denoted by $\models_s$, is the same as that of the weak satisfaction relation except for GCIs [15]. For GCIs, $\models_s$ is defined as $\mathcal{I} \models_s C \sqsubseteq D$ if $\overline{-C} \subseteq +D,\ +C \subseteq +D,\ -D \subseteq -C$.

Let $\mathcal{K}$ be a KB and ϕ be an axiom. $\mathcal{K}$ *quasi-classically entails* (QC entails) ϕ, denoted by $\mathcal{K} \models_Q \phi$, if for every base interpretation $\mathcal{I}$, $\mathcal{I} \models_s \mathcal{K}$ implies $\mathcal{I} \models_w \phi$. In this case, $\models_Q$ is called *QC-entailment*. A base interpretation $\mathcal{I}$ is a *QC-model* of $\mathcal{K}$ if for all axioms φ in $\mathcal{K}$, $\mathcal{I} \models_s \varphi$. $\mathcal{K}$ is *QC-consistent* if there exists some QC-model $\mathcal{I}$ of $\mathcal{K}$, else it is *QC-inconsistent* [15].

2.2 A tableau algorithm for QC-$\mathcal{SHIQ}$ Abox

State of the art DL systems typically use tableaux algorithms [18] to decide the consistency of a KB, i.e., to determine whether a given KB has a model. Consistency checking is one of the main inference problems to which all other inferences can be reduced [16]. A sound, complete and decidable tableau algorithm (called *QC-tableau*) for checking the QC-consistency of an ABox is proposed in [15] by modifying and extending the standard tableau algorithm for $\mathcal{SHIQ}$ [19]. We outline the main steps as follows.

First we note that, in order to work efficiently, the TBox is reduced to an empty TBox with the *internalization* technique and the ABox is transformed into *Negation Normal Form* (NNF).

Internalization: Let U be a universal role, that is, a transitive super role consisting of all roles occurring in $\mathcal{T}$ together with their respective inverses. The base interpretation of U is defined as follows: $U^{\mathcal{I}} = \langle \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}},\ \emptyset\rangle$ for any base interpretation $\mathcal{I}$. Given $\mathcal{T}$, a concept $C_{\mathcal{T}}$ is defined as

$$C_{\mathcal{T}} := \bigsqcap_{C_i \sqsubseteq D_i \in \mathcal{T}} (\neg C_i \sqcup D_i).$$

Any individual x in any model of $\mathcal{T}$ will be an instance of $C_{\mathcal{T}}$. Let $\mathcal{R}_U = \mathcal{R} \cup \{R \sqsubseteq U,\ Inv(R) \sqsubseteq U \mid R$ occurs in $C, D, \mathcal{T}, \mathcal{A}$ or $\mathcal{R}\}$. Now, $\mathcal{A}$ is QC-consistent w.r.t. $\mathcal{R}$ and $\mathcal{T}$ iff $\mathcal{A} \cup \{C_{\mathcal{T}} \sqcap \forall U.C_{\mathcal{T}}(a) \mid a$ occurs in $\mathcal{A}\}$ is QC-consistent w.r.t. $\mathcal{R}_U$.

Negation Normal Form (NNF): A concept expressions is in NNF if all the negations directly precede concept names. Let C, D be two concepts. We say C is *equivalent to* D, denoted by $C \equiv_s D$, if for any strong interpretation $\mathcal{I}$, $C^{\mathcal{I}} = D^{\mathcal{I}}$. That is, $+C = +D$ and $-C = -D$ where $C^{\mathcal{I}} = \langle +C, -C\rangle$ and $D^{\mathcal{I}} = \langle +D, -D\rangle$. Each QCDL concept is equivalent to a QCDL concept in NNF. NNF of a concept expression in QCDL can be computed by applying the equivalences from Table 1.

The QC-tableau algorithm works on a data structure called a *completion forest*. The algorithm starts with the input ABox, $\mathcal{A}$, and applies consistency preserving expansion rules from Table 2 (due to space limitation only five rules are presented here; for the remaining rules the reader is referred to [15, 17]) until no more rules are applicable (the tableau is *complete*) or an obvious contradiction (called a *clash*) is found in every branch. If a complete and clash-free completion forest is obtained, $\mathcal{A}$ is QC-consistent; otherwise it is QC-inconsistent. In the completion forest, the label of a node x is denoted by $\mathcal{L}(x)$.

$$
\begin{array}{rcl||rcl}
\overline{\overline{C}} & \equiv_s & C & & & \\
\neg\overline{C} & \equiv_s & \overline{\neg C} & \neg\neg C & \equiv_s & C \\
\neg(C \sqcup D) & \equiv_s & \neg C \sqcap \neg D & \neg(C \sqcap D) & \equiv_s & \neg C \sqcup \neg D \\
\neg\exists R.C & \equiv_s & \forall R.\neg C & \neg\forall R.C & \equiv_s & \exists R.\neg C \\
\overline{\exists R.C} & \equiv_s & \forall R.\overline{C} & \overline{\forall R.C} & \equiv_s & \exists R.\overline{C} \\
\neg(\leqslant n\, S.C) & \equiv_s & \geqslant n+1\, S.C & \neg(\geqslant n+1\, S.C) & \equiv_s & \leqslant n\, S.C \\
\overline{(\leqslant n\, S.C)} & \equiv_s & \geqslant (n+1)\, S.\overline{C} & \overline{(\geqslant n+1)\, S.C} & \equiv_s & \leqslant n\, S.\overline{\neg C}
\end{array}
$$

Table 1: QC-NNF equivalences.

$\sqcap$-rule	**if** (1) $C_1 \sqcap C_2 \in \mathcal{L}(x)$, x is not blocked, and (2) $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$, **then** $\mathcal{L}(x) := \mathcal{L}(x) \cup \{C_1, C_2\}$.
R-rule	**if** (1) $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and (2) $\sim C_i \in \mathcal{L}(x)$ for some $i \in \{1, 2\}$, **then** $\mathcal{L}(x) := \mathcal{L}(x) \cup \{C_{3-i}\}$.
$\sqcup$-rule	if (1) $C_1 \sqcup C_2 \in \mathcal{L}(x)$, x is not blocked, and (2) $\{C_1, C_2, \sim C_1, \sim C_2\} \cap \mathcal{L}(x) = \emptyset$, **then** $\mathcal{L}(x) := \mathcal{L}(x) \cup \{E\}$ for some $E \in \{C_1, C_2\}$.
$\overline{\sqcap}$-rule	**if** (1) $\overline{C_1 \sqcap C_2} \in \mathcal{L}(x)$, x is not blocked, and (2) $\{\overline{C_1}, \overline{C_2}\} \not\subseteq \mathcal{L}(x)$, **then** $\mathcal{L}(x) := \mathcal{L}(x) \cup \{\overline{E}\}$ for some $E \in \{C_1, C_2\}$.
$\overline{\sqcup}$-rule	**if** (2) $\overline{C_1 \sqcup C_2} \in \mathcal{L}(x)$, and x is not blocked, **then** $\mathcal{L}(x) := \mathcal{L}(x) \cup W$ for some $W \in \{\{\overline{C_1}, \overline{C_2}\}, \{\sim C_1, \overline{C_2}\}, \{\overline{C_1}, \sim C_2\}\}$

Table 2: QC-tableau expansion rules.

3 Optimizations

The basic algorithm for inconsistency-tolerant reasoning discussed in the previous section is too slow for use in practice. We have investigated and employed a range of optimizations that improve the performance of standard tableau algorithms. These optimizations include: normalization and simplification, unfolding, absorption, semantic branching search, dependency directed backtracking. The first three techniques are performed directly on the input which serve to pre-process and simplify the input into a form more amenable to later processing, while the remaining techniques are applied during the search for a model. In the following subsections, each of these techniques will be revised for QCDL by modifying and extending the techniques presented in [16] for classical DLs.

3.1 Normalization and Simplification

Normalization is an optimization technique performed in pre-processing. It allows the detection of contradictions involving complex concepts early during tableaux expansion. Theoretical descriptions of tableau algorithms generally assume that the concept expression to be tested is in negation normal form. Though it simplifies the algorithm, this means that a contradiction will be detected only when an atomic concept and its negation occur in the same node label.

Example 1 Consider the concept expression $\exists R.(C \sqcap D) \sqcap \overline{\exists R.C}$, where C is an atomic

concept. When the algorithm creates an R-successor using the $\exists$-rule and applies the $\overline{\exists}$-rule to $\overline{\exists R.C}$, a clash will be detected due to the fact that $\{C, \overline{C}\} \in \mathcal{L}(y)$. However, if C is a concept expression then C will be transformed into NNF. Thence, the clash would not be detected immediately. If C is a complex concept this may cause a lot of unnecessary expansion. It is possible to detect contradictions caused by non-atomic concepts early by transforming all concepts into a syntactic normal form, which we define here.

Logics that include full negation often provide pairs of operators, either one of which can be eliminated in favor of the other, by using negation. In syntactic normal form, all concepts are transformed so that only one of each such pair appears in the KB. In QC-$\mathcal{SHIQ}$, as in classical $\mathcal{SHIQ}$, all concepts could be transformed into atomic concepts, negations, conjunctions, value restrictions and QC-negations. For example, $\exists R.C$ is transformed into $\neg\forall R.\neg C$. It is important to note that within the normalization process, conjunctions are considered as sets; this simplifies the elimination of redundant conjuncts. For example, $\neg C \sqcup \neg D$ is transformed into $\neg \sqcap \{C, D\}$. The normalization process can also include a range of simplifications that detect obvious clashes during the normalization process and also gets rid of redundant elements of a concept expression.

$$
\begin{array}{rcl}
Norm(A) & = & A \text{ for an atomic concept A} \\
Norm(\neg C) & = & Simp(\neg(Norm(C))) \\
Norm(\overline{C}) & = & Simp(\overline{Norm(C)}) \\
Norm(C_1 \sqcap ... \sqcap C_n) & = & Simp(\sqcap(\{Norm(C_1)\} \cup ... \cup \{Norm(C_n)\})) \\
Norm(C_1 \sqcup ... \sqcup C_n) & = & Norm(\neg(\neg C_1 \sqcap ... \sqcap \neg C_n)) \\
Norm(\forall R.C) & = & Simp(\forall R.Norm(C)) \\
Norm(\exists R.C) & = & Norm(\neg\forall R.\neg C) \\
Norm(\geqslant n\, R.C) & = & Simp(\geqslant n\, R.Norm(C)) \\
Norm(\leqslant n\, R.C) & = & Norm(\neg \geqslant (n+1)\, R.C) \\
\hline
Simp(A) & = & A \text{ for an atomic concept A} \\
Simp(\neg C) & = & \begin{cases} Simp(D) & \text{if } C = \neg D \\ \overline{Simp(\neg D)} & \text{if } C = \overline{D} \\ \neg C & \text{otherwise.} \end{cases} \\
Simp(\overline{C}) & = & \begin{cases} Simp(D) & \text{if } C = \overline{D} \\ \overline{C} & \text{otherwise.} \end{cases} \\
Simp(\sqcap \mathbf{S}) & = & \begin{cases} Simp(\sqcap \mathbf{P} \cup \mathbf{S} \backslash \{\sqcap\{\mathbf{P}\}\}) & \text{if } \sqcap\{\mathbf{P}\} \in \mathbf{S} \\ clash & \text{if } \{C, \overline{C}\} \subseteq \mathbf{S} \\ \sqcap \mathbf{S} & \text{otherwise.} \end{cases} \\
Simp(\forall R.C) & = & \forall R.C \\
Simp(\geqslant n\, R.C) & = & \geqslant n\, R.C \qquad [\mathbf{S}, \mathbf{P} \text{ are sets of concepts}]
\end{array}
$$

Table 3: *Norm* and *Simp* functions for QC-$\mathcal{SHIQ}$.

Table 3 describes normalization and simplification functions *Norm* and *Simp* for QC-$\mathcal{SHIQ}$. Normalized and simplified concepts may not be in negation normal form, but they can be dealt with by treating them exactly like their non-negated counterparts. For example, $\neg \sqcap \{C, D\}$ can be treated as $\neg C \sqcup \neg D$ and $\neg\forall R.\neg C$ can be treated as $\exists R.C$ during the tableau expansion. The normalization and simplification procedure is implemented by a recursive function, applied to the concept expression to check. The normalized and simplified concepts for QC-$\mathcal{SHIQ}$ are presented in Table 4.

Example 2 Consider the Example 1 again. The expression $\exists R.(C \sqcap D) \sqcap \overline{\exists R.C}$ is transformed into $\sqcap\{\neg(\forall R.\neg \sqcap \{C, D\}), \overline{\neg\forall R.\neg C}\}$. The term $\neg(\forall R.\neg \sqcap \{C, D\})$ allows the creation of an R-

Expression	Normalized & Simplified	Expression	Normalized & Simplified
$\neg\neg C$	C		
$\neg\overline{C}$	$\overline{\neg C}$	$\overline{\overline{C}}$	C
$C_1 \sqcap C_2$	$\sqcap\{C_1, C_2\}$	$\overline{C_1 \sqcap C_2}$	$\overline{\sqcap\{C_1, C_2\}}$
$C_1 \sqcup C_2$	$\neg \sqcap \{\neg C_1, \neg C_2\}$	$\overline{C_1 \sqcup C_2}$	$\overline{\neg \sqcap \{\neg C_1, \neg C_2\}}$
$\forall R.C$	$\forall R.C$	$\overline{\forall R.C}$	$\overline{\forall R.C}$
$\exists R.C$	$\neg\forall R.\neg C$	$\overline{\exists R.C}$	$\overline{\neg\forall R.\neg C}$
$\geqslant n\, R.C$	$\geqslant n\, R.C$	$\leqslant n\, R.C$	$\neg \geqslant (n+1)\, R.C$
$\overline{\geqslant n\, R.C}$	$\overline{\geqslant n\, R.C}$	$\overline{\leqslant n\, R.C}$	$\overline{\neg \geqslant (n+1)\, R.C}$

Table 4: Normalized and simplified concepts in QC-$\mathcal{SHIQ}$

successor using the $\exists$-rule whose label contains both C and $\overline{C}$ (by applying the $\overline{\exists}$-rule to the term $\overline{\neg\forall R.\neg C}$). Since the two occurrences of C are in the syntactic normal form, a clash will be detected immediately, regardless of the structure of C.

3.2 Unfolding

Unfolding is a recursive substitution procedure that allows the testing of the satisfiability of a given concept C w.r.t. $\mathcal{T}$ by eliminating from C all concept names occurring in $\mathcal{T}$ [16]. If $A \equiv D$ is an axiom in $\mathcal{T}$, where A is a non-primitive (defined in $\mathcal{T}$) concept name, the procedure simply substitutes (i.e., unfolds) A with D wherever it occurs in C, and then recursively unfolds D in the same manner. If $A \sqsubseteq D$ is an axiom in $\mathcal{T}$, where A is a primitive concept name, A is substituted by $A' \sqcap D$, where A' is a new concept name that does not occur in $\mathcal{T}$ or C. The concept name A' represents the primitiveness of A, i.e., the unspecified characteristics that differentiate A from D. $\texttt{Unfold}(C, \mathcal{T})$ denotes the concept C after unfolding w.r.t. $\mathcal{T}$ [16].

Subsumption testing can be made independent of $\mathcal{T}$ using the same technique. The problem of determining if C is subsumed by D w.r.t. a TBox $\mathcal{T}$ is the same as the problem of determining if $\texttt{Unfold}(C, \mathcal{T})$ is subsumed by $\texttt{Unfold}(D, \mathcal{T})$ w.r.t. an empty TBox; in other words, $\mathcal{T} \models C \sqsubseteq D$ iff $\emptyset \models \texttt{Unfold}(C, \mathcal{T}) \sqsubseteq \texttt{Unfold}(D, \mathcal{T})$ [16].

Generally there are two problems regarding concept unfolding: (1) an unrestricted recursive unfolding could possibly produce a resulting concept expression of exponential size; (2) unfolding would not be possible if $\mathcal{T}$ contains (i) multiple definitions for some concept name A, e.g., if $\{A \equiv C,\ A \equiv D\} \subseteq \mathcal{T}$, or (ii) cyclical axioms, e.g., if $(A \sqsubseteq \exists R.A) \in \mathcal{T}$. The former problem can be addressed by a technique called *lazy unfolding* which unfolds concepts only when required during the progress of the algorithm. In other words, lazy unfolding does not expand the occurrences of concept names which follow $\exists$ or $\forall$. For example, when testing the satisfiability of an expression $\exists R.E$, where E is a concept name, the unfolding of E can be delayed until the $\exists$-rule has created an R-successor y with $\mathcal{L}(y) = \{E\}$. By imposing this restriction, lazy unfolding may prevent the exponential increase of a concept expression.

Example 3 Consider, testing the satisfiability of the concept expression: $\exists R.E \sqcap \forall R.\overline{E}$. The optimized algorithm will detect a contradiction immediately when the $\exists$-rule creates an R-successor y and applies the $\forall$-rule because $\{E, \overline{E}\} \subseteq \mathcal{L}(y)$. This may save a lot of unnecessary work if unfolding E produces a large and complex expression.

As we have just noticed, all axioms in an arbitrary TBox are not amenable to unfolding. The solution to this problem is to divide the Tbox $\mathcal{T}$ into two components, a general part $\mathcal{T}_g$ and an unfoldable part $\mathcal{T}_u$ such that $\mathcal{T} = \mathcal{T}_u \cup \mathcal{T}_g$; where $\mathcal{T}_u$ contains unique, acyclical, definition axioms and $\mathcal{T}_g$ contains the rest of $\mathcal{T}$. This can be achieved easily by initializing $\mathcal{T}_u$ to $\emptyset$, then

for each axiom X in $\mathcal{T}$, adding X to $\mathcal{T}_u$ if $\mathcal{T}_u \cup X$ is still unfoldable, adding X to $\mathcal{T}_g$ otherwise. In this way, reasoning tasks w.r.t. $\mathcal{T}$ can be considered as reasoning tasks w.r.t. $\mathcal{T}_u$ and $\mathcal{T}_g$: use lazy unfolding to deal with $\mathcal{T}_u$ and internalization to deal with $\mathcal{T}_g$ [16].

3.3 Absorption

As we have seen in the previous section, an arbitrary TBox $\mathcal{T}$ is divided into two parts, $\mathcal{T}_u$ and $\mathcal{T}_g$; unfolding is applied to $\mathcal{T}_u$ and internalization is applied to $\mathcal{T}_g$. The reasoning performance for $\mathcal{T}_u$ can be very good, while the reasoning performance for $\mathcal{T}_g$ might be bad, because internalization may introduce many disjunctions which increases the search space exponentially. For example, if a $\mathcal{T}_g$ contains 10 GCIs with 10 nodes, there are already 100 disjunctions, and they can be non-deterministically expanded in 2^{100} different ways. Therefore, it is a good strategy to eliminate as many GCIs from $\mathcal{T}_g$ as possible.

Absorption is a technique that tries to eliminate GCIs by absorbing them into primitive definitions. By considering $\mathcal{T}_u$ and $\mathcal{T}_g$, if one can move axioms from $\mathcal{T}_g$ to $\mathcal{T}_u$ while keeping the semantics of $\mathcal{T}$ unchanged, one should be able to improve the reasoning performance. The absorption technique presented here is analogous to that of classical DL described in [16] except for handling the QC-negations. The basic idea is that a GCI of the form $C \sqsubseteq D$, where C may be a non-atomic concept, is transformed into the form of a primitive definition $A \sqsubseteq D'$, where A is an atomic concept, using the axiom equivalences (1) and (2) below. Then, $A \sqsubseteq D'$ together with an existing primitive definition $A \sqsubseteq C'$ may be replaced by $A \sqsubseteq C' \sqcap D'$.

$$C_1 \sqcap C_2 \sqsubseteq D \iff C_1 \sqsubseteq D \sqcup \neg C_2 \tag{1}$$

$$C \sqsubseteq D_1 \sqcap D_2 \iff C \sqsubseteq D_1 \text{ and } C \sqsubseteq D_2 \tag{2}$$

Given $\mathcal{T}_u$ and $\mathcal{T}_g$, absorbing the axioms from $\mathcal{T}_g$ into the primitive definitions in $\mathcal{T}_u$ can be done according to the following procedure. First, each axiom of the form $C \equiv D$ is replaced by an equivalent pair of axioms $C \sqsubseteq D$ and $\neg C \sqsubseteq \neg D$, and $\mathcal{T}_g'$ is set to $\emptyset$. Then for each axiom $(C \sqsubseteq D) \in \mathcal{T}_g$ [16]:

(A) Initialize a set $\mathbf{G} = \{\neg D, C\}$, which represents the axiom in the form $T \sqsubseteq \neg \sqcap \{\neg D, C\}$ (i.e., $T \sqsubseteq D \sqcup \neg C$).

(B) If there is a primitive definition axiom $(A \sqsubseteq C) \in \mathcal{T}_u$ for some $A \in \mathbf{G}$, then absorb the general axiom into the primitive definition axiom so that it becomes $A \sqsubseteq \sqcap\{C, \neg \sqcap (\mathbf{G} \backslash \{A\})\}$, and exit.

(C) If there is a primitive definition axiom $(A \equiv D) \in \mathcal{T}_u$ for some $A \in \mathbf{G}$, then substitute A with D, $\mathbf{G} \rightarrow \{D\} \cup \mathbf{G} \backslash \{A\}$, and return to step (B).

(D) If there is a primitive definition axiom $(A \equiv D) \in \mathcal{T}_u$ for some $\neg A \in \mathbf{G}$, then substitute $\neg A$ with $\neg D$, $\mathbf{G} \rightarrow \{\neg D\} \cup \mathbf{G} \backslash \{\neg A\}$, and return to step (B).

(E) If there is a primitive definition axiom $(A \equiv D) \in \mathcal{T}_u$ for some $\overline{A} \in \mathbf{G}$, then substitute $\overline{A}$ with $\overline{D}$, $\mathbf{G} \rightarrow \{\overline{D}\} \cup \mathbf{G} \backslash \{\overline{A}\}$, and return to step (B).

(F) If there is some $C \in \mathbf{G}$ such that C is of the form $\sqcap \mathbf{S}$, then use associativity to simplify $\mathbf{G}$, $\mathbf{G} \rightarrow \mathbf{S} \cup \mathbf{G} \backslash \{\sqcap \mathbf{S}\}$, and return to step (B).

(G) If there is some $C \in \mathbf{G}$ such that C is of the form $\neg \sqcap \mathbf{S}$, then for every $D \in \mathbf{S}$ determine if C can be absorbed (recursively) in $\mathbf{G}$, $\{\neg D\} \cup \mathbf{G} \backslash \{\neg \sqcap \mathbf{S}\}$, and exit.

(H) Otherwise, the axiom could not be absorbed, so add $\neg \sqcap \mathbf{G}$ to $\mathcal{T}_g'$, $\mathcal{T}_g' \rightarrow \mathcal{T}_g' \cup \neg \sqcap \mathbf{G}$, and exit.

In the above absorption technique, step (E) is new and has been added to the technique in [16] to handle QC-negations.

3.4 Semantic branching

Standard tableau algorithms are inherently inefficient because they use a search technique based on syntactic branching. When expanding the label of a node x, $\mathcal{L}(x)$, syntactic branching works by choosing an unexpanded disjunction $(C_1 \sqcup C_2 \sqcup ... \sqcup C_n)$ in $\mathcal{L}(x)$ and searching the different models obtained by adding each of the disjuncts $C_1, C_2, ..., C_n$ to $\mathcal{L}(x)$ [16]. Since the alternative branches of the search tree are not disjoint, the recurrence of an unsatisfiable disjunct in different branches can occur. This can lead to a lot of wasted expansions. For example, consider the tableau expansion of a node x, where $\mathcal{L}(x) = \{(C \sqcup D_1), (C \sqcup D_2)\}$ and C leads to a clash. The syntactic branching technique could lead a wasted expansion as shown in Figure 1, where a clash due to C must be demonstrated twice. This problem can be dealt with by using a semantic branching technique analogous to that of classical DL [16].

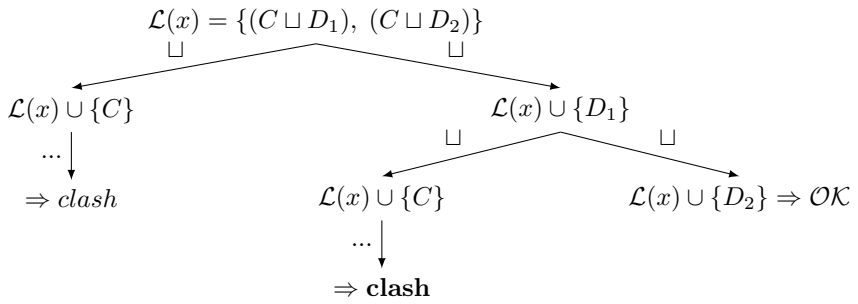

Figure 1: Syntactic branching with wasted expansion.

With semantic branching, a single disjunct D is chosen from one of the unexpanded disjunctions in $\mathcal{L}(x)$. The two possible sub-trees obtained by adding either D or $\overline{D}$ to $\mathcal{L}(x)$ are then searched (recall, $\overline{D}$ is the QC-negation of D, i.e., $D \cap \overline{D} = \emptyset$ w.r.t. the domain, and $\{D, \overline{D}\}$ is the clash). Now we have two disjoint subtrees and the possibility of wasted expansions such as in syntactic branching is avoided. As shown in Figure 2, with the semantic branching, only one exploration of the expression C was needed whereas, with the syntactic branching, two explorations are needed.

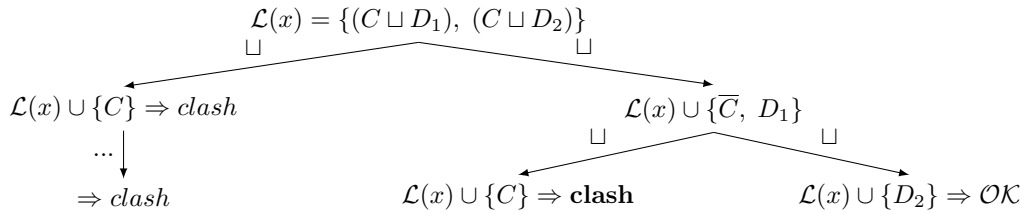

Figure 2: Semantic branching search.

3.5 Dependency directed backtracking

If a sub-problem leads to a clash, this clash can be detected only when the sub-problem is expanded. So inherent unsatisfiability concealed in sub-problems can lead to large amounts

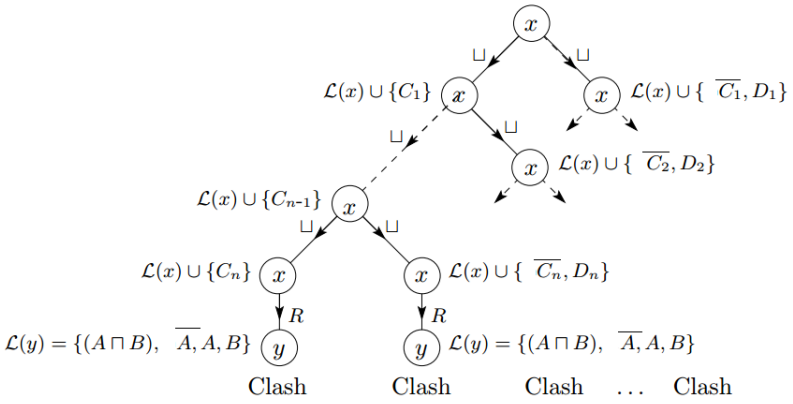

Figure 3: Thrashing with normal backtracking.

of unproductive backtracking search known as *thrashing*. The problem becomes worse when blocking is used to guarantee termination, because blocking may require that sub-problems be expanded only after all other forms of expansions have been performed. For example, expanding a node x, where $\mathcal{L}(x) = \{(C_1 \sqcup D_1), ..., (C_n \sqcup D_n), \exists R.(A \sqcap B), \forall R.\overline{A}\}$ could lead to the fruitless exploration of 2^n possible R successors of x until the inherent unsatisfiability is discovered. The search tree created by tableau algorithm (using semantic branching) is presented in Figure 3. This problem can be addressed by adopting a form of dependency-directed backtracking called backjumping. The technique is essentially the same as that for classical DL presented in [16]; here we review the techniques for handling the QC-negations.

Backjumping is a crucial optimization technique that can effectively prune irrelevant alternatives of non-deterministic branching decisions. If a branching point is not involved in a clash, other alternatives of the branching point may be bypassed, because they cannot eliminate the cause of clash. So the challenge is to locate the cause of a clash which will allow one to downsize the search space. In order to identify the branching point involved in a clash, all concepts are labeled with a dependency set containing information about the branching points on which they depend. A concept $C \in \mathcal{L}(x)$ depends on a branching point if C was added to $\mathcal{L}(x)$ at the branching point or if C depends on another concept D and D depends on that branching point. A concept $C \in \mathcal{L}(x)$ depends on a concept D when C was added to $\mathcal{L}(x)$ by the application of a deterministic expansion rule that used D. For example, if $A \in \mathcal{L}(x)$ was derived from the expansion of $(A \sqcap B) \in \mathcal{L}(x)$, then $A \in \mathcal{L}(x)$ depends on $(A \sqcap B) \in \mathcal{L}(x)$ [16].

When a concept is added to a node by applying the tableau expansion rules, it inherits the dependencies from the concepts it was generated by. If the concept is added by the application of a non-deterministic rule, a dependency from the new branching point is also added. When a clash is discovered, a new dependency set is created by the union of dependency sets of the clashing concepts and backtracking is initiated. In the backtracking, each branching point is checked against the dependency set to see whether it is in the dependency set. If a branching point is not in the dependency set, then the other branching points are ignored and backtracking continues. If the branching point is in the dependency set, and the other branches are not explored yet, then backtracking stops and searching proceeds with the exploration of the other branches. When all branches of a branching point are explored, the union of the dependency sets from the branches is taken and backtracking continues [16].

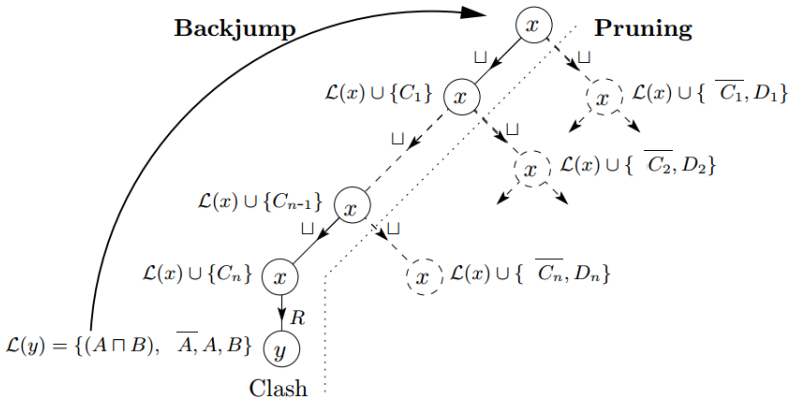

Figure 4: Pruning the search using backjumping.

For example, consider the previous example; when expanding a node x, where $\mathcal{L}(x) = \{(C_1 \sqcup D_1), ..., (C_n \sqcup D_n), \exists R.(A \sqcap B), \forall R.\overline{A}\}$, by using backjumping we could reduce the search space dramatically. In the search algorithm, after creating n branches, the label of the n^{th} node x_n will contain $\{\exists R.(A \sqcap B), \forall R.\overline{A}\}$. When $\exists R.(A \sqcap B)$ is expanded, the algorithm creates an R-successor y_1 with $\mathcal{L}(y_1) = \{(A \sqcap B)\}$ using the $\exists$-rule, and then applies the $\forall$-rule to $\forall R.\overline{A}$ which results with $\mathcal{L}(y_1) = \{(A \sqcap B), \overline{A}\}$. Now, applying the $\sqcap$-rule to $A \sqcap B$ leads to a clash, because $\{A, \overline{A}\} \subset \mathcal{L}(y_1)$. Since neither A nor $\overline{A}$ in $\mathcal{L}(y_1)$ depend on the branching point from x to x_n, the algorithm backtracks to the most recent branching point on which one of A or $\overline{A}$ did depend without exploring alternative branches at any branching point between x to x_n. We show in Figure 4 how the search tree below x is pruned by backjumping, and thus the number of R-successors explored is reduced by 2^n - 1.

4 Evaluation

To check the effectiveness of the optimization techniques discussed in the previous section for inconsistency-tolerant reasoning, i.e., QCDL, we implemented those techniques top of QC-OWL. QC-OWL [17] is an inconsistency-tolerant reasoner that can handle inconsistency directly with reasonable inference power. It can perform reasoning over both consistent and inconsistent ontologies with acceptable performance for the DL $\mathcal{SHIQ}$. It was designed and developed by following the Strategy Pattern, a behavioral design pattern, and is based on the core framework of Pellet [20], a widely used complete OWL-DL reasoner.

In this section, we compare our results with those of our previous version of QC-OWL. The benchmark ontologies used in the experiments are presented in Table 5. The first 12 ontologies (ID# 1 to ID# 12) were found in [15] while 13 (ID# 13) was found in [10]. The ontologies 14, 15, 16, 17, 18 (ID# 14 to ID# 18) were collected from the TONES Ontology Repository [21] while the remaining two (ID# 19 to ID# 20) were collected from ISG Ontology Repository [22]. For the experiments each ontology was processed five times and the time required to perform QC-consistency test was recorded. The average time of the five independent runs is displayed in the table. In the table, the column *Con* is for the consistency and *Opt* with optimizations in QC-OWL. All experiments were performed on a Notebook with Intel Core i7 CPU and 8G

ID	KB name	DL expressivity	Concept count	Axiom count
1	amino-acid	ALCF(D)	46	563
2	heart	SHI	75	448
3	bad-food	ALCO(D)	18	52
4	buggyPolicy	ALCHO	15	41
5	tambis-patched	SHIN	395	1090
6	uma-025-arctan	ALCRIF(D)	366	153403
7	0.01-arctan	ALCRF(D)	366	14816
8	0.03-arctan	ALCRIF(D)	366	26421
9	0.01-arctan-inc	ALCRIF(D)	366	14829
10	0.03-arctan-inc	ALCRIF(D)	366	26421
11	0.04-arctan-inc	ALCRF(D)	366	32231
12	0.07-arctan-inc	ALCRIF(D)	366	49592
13	chem-a	ALCHOF(D)	48	196
14	goslim	AL	161	485
15	transportation	ALCH(D)	445	2364
16	economy	ALCH(D)	339	2817
17	numerics	SHIF(D)	2364	7268
18	yowl-complex	SHIF(D)	336	2212
19	00390	SHIF	16311	366495
20	00786	SH(D)	93413	1212604

Table 5: Characteristics of the benchmark KBs.

memory on Windows 8 platform. The maximum allocated memory for JVM was 512M.

KB name	Con	QC-OWL	QC-OWL(Opt)
amino-acid.owl	Y	33	26
0.01-arctan.owl	Y	36	33
0.03-arctan.owl	Y	39	36
heart.owl	Y	27	25
tambis-patched.owl	Y	39	27
uma-025-arctan.owl	Y	30	24

Table 6: QC-consistency test results (consistent ontologies).

KB name	Con	QC-OWL	QC-OWL(Opt)
bad-food.owl	N	34	26
buggyPolicy.owl	N	22	19
0.01-arctan-inc.owl	N	14761	44
0.03-arctan-inc.owl	N	10342	38
0.04-arctan-inc.owl	N	20044	32
0.07-arctan-inc.owl	N	185803	202

Table 7: QC-consistency test results (inconsistent ontologies).

The experiments were conducted in two steps. In the first step, QC-consistency tests were performed for the same set of ontologies as found in [15]. The results are presented in Table 6 and Table 7 for consistent and inconsistent ontologies, respectively. As it is shown in Table 6, QC-OWL(Opt) marginally outperforms QC-OWL for consistent ontologies. Since QC-OWL already shows good performance for this set of ontologies, the performance improvement is not

significant. Indeed, the search spaces of these ontologies are small due to the characteristics of these ontologies. However, the results presented in Table 7 show that QC-OWL(Opt) significantly outperforms QC-OWL for inconsistent ontologies. For example, in the case of the *0.04-arctan-inc* ontology, QC-OWL takes around 20 seconds whereas QC-OWL(Opt) takes only 32 milliseconds. It is important to note that the performance improvement through optimization is greater for inconsistent ontologies than consistent ontologies. The reason is, for an inconsistent ontology, every branch must be explored before returning the result. However, in the case of a consistent ontology, a model can be found before exploring all alternative branches. When a model is found in a branch, the algorithm returns immediately without exploring the remaining branches. Therefore in general, the performance improvement for inconsistent ontologies can be expected to be greater than that for consistent ontologies.

In the second step, QC-consistency test were performed for another set of popular ontologies and the results are presented in Table 8. For this experiments, the maximum allocated memory for JVM was 2G and *mem-out* stands for *OutOfMemoryError* in Java. The results presented in Table 8 show that optimizations play a significant role for the performance improvement. As an example, for the *transportation* ontology, QC-OWL takes 470 milliseconds whereas QC-OWL(Opt) takes only 45 milliseconds. It is motivating to note for the *00786* ontology, QC-OWL gets *mem-out* while QC-OWL(Opt) gets result in 325 milliseconds. The performance improvement in Table 8 is significant because the search spaces of these ontologies are larger than the ontologies in Table 6 (i.e., the ontologies in Table 8 contain more individuals than the ontologies in Table 6).

KB name	**Con**	**QC-OWL**	**QC-OWL(Opt)**
chem-a.owl	N	94	32
goslim.owl	Y	260	44
transportation.owl	Y	470	45
economy.owl	Y	661	71
numerics.owl	Y	2075	100
yowl-complex.owl	Y	12262	96
00390.owl	Y	*mem-out*	32
00786.owl	Y	*mem-out*	325

Table 8: QC-consistency test results.

5 Related and Future Work

In the past decades, numerous techniques have been developed for optimizing standard tableau-based reasoning, but none of them are directed to inconsistency-tolerant reasoning. In this section, we outline some work that is related to optimizing the tableau-based reasoning for classical DLs and discuss some future research directions for improving the performance of an inconsistency-tolerant tableau-based reasoner.

Most state-of-the-art optimization techniques in tableau-based DL reasoning have been discussed in [16]. Apart from optimizing the tableau algorithm, a few researchers also attempted to parallelize the tableau algorithm itself by applying a thread-based strategy in a shared-memory environment (see, for example [17]). Although thread-based strategies such as multi-threading in a multi-cored processor are often the easiest and simplest way to achieve high performance, speed gain via thread-level parallelism is limited by the number of available cores. A process-based strategy discussed in [17] is another option for achieving scalable performance.

In order to provide efficiency in reasoning, 3 profiles for OWL 2 offer important advantages depending on the application scenario: OWL 2 EL, OWL 2 QL and OWL 2 RL [23]. For example, OWL 2 EL is useful for ontologies that contain very large numbers of properties and/or classes, while OWL 2 QL is useful for dealing with ontologies with very large volumes of instance data, and where query answering is the most important reasoning task. OWL 2 RL is aimed at applications that require scalable reasoning without sacrificing expressivity [23].

For the experiments, our reasoner has been implemented on top of Pellet using Java. Although Java has many appealing features, it is not strongly recommended for high performance computing due to some design features associated with this language, such as garbage collection, etc. Better performance can be achieved by implementing this algorithm in C or C++. Prolog, a general purpose logic programming language, could be another good choice. Prolog's backtracking strategy may be well suited implementing the dependency-directed backtracking.

Recently, Faddoul and MacCaull [24] investigated algebraic tableau reasoning for the DL $\mathcal{ALCQ}$. Preliminary results motivate the application of algebraic reasoning for paraconsistent reasoning. However, in order to work with an algebraic reasoning component, a standard tableau calculus needs to be modified and extended.

6 Conclusion

In this work, we discuss a set of widely used optimization techniques developed for classical tableau-based reasoners which we implemented on top of QC-OWL, our inconsistency-tolerant reasoner, and compare its performance with our naive implementation. The experimental results show a significant runtime improvement for a wide range of both consistent and inconsistent ontologies. While it is possible to measure the performance of each optimization individually, we did not do so, as these optimizations are benchmarked for classical DLs. In future, we shall investigate other optimizations for QC-OWL, e.g., boolean constraint propagation, heuristic guided search, etc. We also plan to parallelize the QC-tableau algorithm itself incorporating a set of optimization techniques which is hoped to significantly improve the performance of the QC-tableau algorithm.

Acknowledgments: The second author wishes to thank the Natural Sciences and Engineering Research Council of Canada for financial support.

References

[1] M. Horridge, B. Parsia, and U. Sattler, "Explaining inconsistencies in OWL ontologies," in *Proceedings of the 3rd International Conference on Scalable Uncertainty Management*. Berlin, Heidelberg: Springer-Verlag, 2009, pp. 124–137.

[2] D. Bell, G. Qi, and W. Liu, "Approaches to inconsistency handling in description-logic based ontologies," in *On the Move to Meaningful Internet Systems 2007: OTM 2007 Workshops*, ser. Lecture Notes in Computer Science, R. Meersman, Z. Tari, and P. Herrero, Eds. Springer Berlin Heidelberg, 2007, vol. 4806, pp. 1303–1311.

[3] F. Maier, Y. Ma, and P. Hitzler, "Paraconsistent OWL and related logics," *Semantic Web*, vol. 4, no. 4, pp. 395–427, 2013.

[4] N. Kamide, "A comparison of paraconsistent description logics," *International Journal of Intelligence Science*, vol. 3, no. 2, pp. 99–109, 2013.

[5] J. Fang and Z. Huang, "Reasoning with inconsistent ontologies," *Tsinghua Science and Technology*, vol. 15, no. 6, pp. 687–691, 2010.

[6] C. Cocos, F. Imam, and W. MacCaull, "Ontology merging and reasoning using paraconsistent logics," *International Journal of Knowledge-Based Organizations (IJKBO)*, vol. 2, pp. 35–51, 2012.

[7] G. Qi, W. Liu, and D. A. Bell, "A revision-based approach for handling inconsistency in description logics," in *Proceedings of the 11th International Workshop on NonMonotonic Reasoning (NMR-06)*, 2006.

[8] L. Nguyen and A. Szalas, "Three-valued paraconsistent reasoning for Semantic Web agents," in *Agent and Multi-Agent Systems: Technologies and Applications*, ser. Lecture Notes in Computer Science, P. Jdrzejowicz, N. Nguyen, R. Howlet, and L. Jain, Eds. Springer Berlin Heidelberg, 2010, vol. 6070, pp. 152–162.

[9] S. Odintsov and H. Wansing, "Inconsistency-tolerant description logic. part ii: A tableau algorithm for CALCC," *Journal of Applied Logic*, vol. 6, no. 3, pp. 343–360, 2008.

[10] X. Zhang, G. Xiao, Z. Lin, and J. Van Den Bussche, "Inconsistency-tolerant reasoning with OWL DL," *International Journal of Approximate Reasoning*, vol. 55, no. 2, pp. 557–584, 2014.

[11] X. Zhang, Z. Lin, and K. Wang, "Towards a paradoxical description logic for the Semantic Web," in *Foundations of Information and Knowledge Systems*, ser. Lecture Notes in Computer Science, S. Link and H. Prade, Eds. Springer Berlin Heidelberg, 2010, vol. 5956, pp. 306–325.

[12] T. Kaminski, M. Knorr, and J. Leite, "Efficient paraconsistent reasoning with ontologies and rules," in *Proceedings of the 24th International Conference on Artificial Intelligence (IJCAI 2015)*, 2015, pp. 3098–3105.

[13] J. Belnap, Nuel D., "A useful four-valued logic," in *Modern Uses of Multiple-Valued Logic*, ser. Episteme, M. Dunn and G. Epstein, Eds. Springer Netherlands, 1977, vol. 2, pp. 5–37.

[14] A. Hunter, "Reasoning with contradictory information using quasi-classical logic," *Journal of Logic and Computation*, vol. 10, pp. 677–703, 1999.

[15] X. Zhang, Z. Feng, W. Wu, M. Hossain, and W. MacCaull, "On the satisfiability of qasi-classical description logics," *Computing and Informatics*, 2015, to appear. [Online]. Available: http://cs.tju.edu.cn/faculty/zhangxiaowang/publication/CAI15.pdf

[16] F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, Eds., *The Description Logic Handbook: Theory, Implementation, and Applications*. New York, NY, USA: Cambridge University Press, 2003.

[17] M. Hossain, "Inconsistency-tolerant description logic reasoning," M.Sc. thesis, St. Francis Xavier University, 2016.

[18] F. Baader and U. Sattler, "An overview of tableau algorithms for description logics," *Studia Logica*, vol. 69, no. 1, pp. 5–40, 2001.

[19] I. Horrocks, U. Sattler, and S. Tobies, "Reasoning with individuals for the description logic $\mathcal{SHIQ}$," in *Proceedings of the 17th International Conference on Automated Deduction*. London, UK: Springer-Verlag, 2000, pp. 482–496.

[20] E. Sirin, B. Parsia, B. C. Grau, A. Kalyanpur, and Y. Katz, "Pellet: A practical OWL-DL reasoner," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 5, no. 2, pp. 51–53, 2007.

[21] (2008) TONES: Ontology repository. University of Manchester. [last accessed: December 2015]. [Online]. Available: http://owl.cs.manchester.ac.uk/repository/

[22] Information Systems Group: Ontology repository. University of Oxford. [last accessed: April 2016]. [Online]. Available: http://www.cs.ox.ac.uk/isg/ontologies/

[23] P. Hitzler, M. Krötzsch, B. Parsia, P. F. Patel-Schneider, and S. Rudolph, Eds., *OWL 2 Web Ontology Language: Primer*. W3C Recommendation, 27 October 2009. [Online]. Available: http://www.w3.org/TR/owl2-primer/.

[24] J. Faddoul and W. MacCaull, "Handling non-determinism with description logics using a fork/join approach," *International Journal of Networking and Computing*, vol. 5, no. 1, pp. 61–85, 2015.