PROSE: A Plugin-based Framework for Paraconsistent Reasoning on Semantic Web

Xiaowang Zhang, Zhiyong Feng*, Wenrui Wu, Xin Wang, Guozheng Rao

School of Computer Science and Technology, Tianjin University, China
Tianjin Key Laboratory of Cognitive Computing and Application, Tianjin, China
{xiaowangzhang, zyfeng, wenruiwu, wangx, rgz}@tju.edu.cn

Abstract. The study of paraconsistent reasoning with ontologies is especially important for the Semantic Web since knowledge is not always perfect within it. However, classical OWL reasoners cannot support reasoning with inconsistent ontologies. In this paper, we present a plugin-based framework called PROSE to provide rich paraconsistent reasoning services for OWL ontologies, whose architecture contains the three following parts: a classical OWL reasoner (for reasoning), a multi-valued transformer (for paraconsistent transformation), and OWL API connecting with them. Within the proposed framework PROSE, we implement different multi-valued paraconsistent reasoning in OWL such as quasi-classical reasoning, four-valued material reasoning, four-valued internal reasoning, and four-valued strong reasoning. Moreover, we select three popular classical OWL reasoners (i.e., Pellet, HermiT, and FaCT++) and two typical kinds of reasoning services (i.e., consistency checking and classification) for users. As we excepted, PROSE does exactly enable current classical OWL reasoners to tolerate inconsistency in a simple and convenient way. Finally, we evaluate the three reasoners in an united framework (PROSE) and, as a result, those results can amend the analysis of the three reasoners on inconsistent ontologies.

1 Introduction

As an extension of the World Wide Web (WWW), the Semantic Web [3] becomes more constantly changing and highly collaborative. Ontologies considered one of the pillars of the Semantic Web will rarely be perfect due to many reasons, such as modeling errors, migration from other formalisms, merging ontologies, and ontology evolution [36, 32, 25]. As a fragment of predicate logic, description logic (DL), which is the logical foundation of the Web Ontology Language [27] (e.g., sublanguages OWL Lite and OWL DL correspond to $SHIF(D)$ and $SHOIN(D)$ respectively), is unable to tolerate inconsistencies occurring in ontologies [4]. Thus, the topic of inconsistency handling in OWL and DL has received extensive interests in the community in recent years [36, 32, 25, 22].

There are several approaches to handling inconsistencies in DLs. All of them can be functionally roughly classified into two different types. One type is based

* Corresponding author.
on the assumption that inconsistencies indicate erroneous data which are to be removed in order to obtain a consistent ontology. In these approaches, researchers hold a common view that ontologies should be completely free of inconsistencies, and thus try to eliminate inconsistencies from them to recovery consistency immediately by any means possible. However, there are some different opinions about the first type of treating inconsistency. And argues that inconsistencies in knowledge are the norm in the real world, and so should be formalized and used, rather than always rejected. The other, called inconsistency-tolerant (or paraconsistent) approaches, is not to simply avoid inconsistencies but to apply non-standard reasoning methods (e.g., non-standard inference or non-classical semantics) to obtain meaningful answers. In the second type of approaches, inconsistency treated as a natural phenomenon in realistic data, should be tolerated in reasoning. So far, the main idea of existing paraconsistent methods for handling inconsistency is introducing either non-standard inference or non-classical semantics to draw meaningful conclusions from inconsistent KBs. Those paraconsistent approaches with non-standard inference presented by are employing argument principles where consistent subsets are selected from an inconsistent KB as substitutes in reasoning. Those approach with preferred semantics by are introducing some preference between interpretations and collecting minimal interpretations as candidate models. In this sense, this approach is employing a non-monotone strategy to handle inconsistency. Those paraconsistent approaches are based on multi-valued semantics (a popular kind of non-classical semantics) such as four-valued DL. Because four-valued logic is a basic member of the family of multi-valued logics, four-valued semantics of DL has got a lot of attention. However, the inference power of four-valued DL is rather weak as noted/argued by although three kinds of implications (namely, material inclusion, internal inclusion, and strong inclusion) are introduced in four-valued DL to improve inference power. Some important properties about inference such as disjunctive syllogism, resolution, and intuitive equivalence are invalid in four-valued DL.

Indeed, the weak inference power is one of common characteristics of the family of paraconsistent logics where some important inference rules are prohibited in order to avoid the explosion of inference. As a result, this topic of making more properties about inference valid under preventing the explosion of inference becomes interesting and important since more useful information can be inferred from inconsistent ontologies. To avoid the shortcomings of four-valued DL, in our pervious work, we presented a quasi-classical description logic (QCDL), based on quasi-classical semantics proposed in.

2
It is proven that QCDL can be applied to tolerate inconsistency in reasoning on OWL ontologies with the stronger inference power of paraconsistent reasoning comparing with four-valued DL. In other words, QCDL can bring more conclusions than four-valued DL.

In our previous work [47], we have developed a paraconsistent reasoning based on quasi-classical transformation whose aim is to employ Pellet [38] to implement a prototype reasoning system named PROSE. In this paper, based on the architecture of PROSE, we present a framework (we still say PROSE) in which several popular multi-valued reasoning mechanisms including quasi-classical reasoning, four-valued material reasoning, four-valued internal reasoning, and four-valued strong reasoning and three classical OWL reasoners, namely, Pellet [38], HermiT [37], and FaCT++ [41] can be enriched in paraconsistent reasoning on OWL ontologies. In this sense, our implementation can be seen as a significant extension of current classical DL reasoners. Furthermore, we evaluate PROSE by employing the three classical reasoners for inconsistent ontologies in two reasoning problems. Those results provide a way to observe performances of the three classical reasoners in the case of inconsistent ontologies to amend existing results of comparison of OWL reasoners on consistent ontologies [6]. Additionally, we offer several alternatives for classical reasoners to meet multiple requirements in a practical world.

Compared with the previous version of PROSE which is implemented only in Pellet, the current implementation of PROSE has several advantages:

− PROSE is a self-decision system. For classical consistent ontologies, PROSE can give answers in considerable time. However, PROSE has the ability to judge whether the input ontology is consistent or not and then decide to whether to shutdown the process of transformation or not. PROSE is more powerful in terms of reasoning abilities.

− PROSE supports three widely used classical OWL reasoners. We can select the best candidate for our reasoning tasks among Pellet, HermiT, and FaCT++. So each reasoner can do its adept jobs as possible.

− The implementation of PROSE is commonly used. Since PROSE strongly relies on the OWL-API, it can be easily integrated with other OWL reasoners which are compatible with OWL-API.

This paper is an extended version of our previous conference paper in the proceedings of JIST 2015 [42]. In this extension, we have improved this paper in the following three aspects:

− We have additionally evaluated PROSE on LUBM benchmark ontologies [12] in three OWL reasoners (i.e., Pellet, HermiT, and FaCT++) and two typical kinds of reasoning services (i.e., consistency checking and classification) (see §4).

− We have re-implemented all three scenarios (PROSE_M, PROSE_I, and PROSE_S) of ParOWL in PROSE by revising our multi-valued transformer: ParOWL_M for material inclusions, ParOWL_I for internal inclusions, and ParOWL_S for strong inclusions (see Section 3.3) respectively.
We have further evaluated three scenarios of ParOWL on LUBM benchmark ontologies \[12\] on our generated ontologies in three OWL reasoners (i.e., Pellet, HermiT, and FaCT++) and two typical kinds of reasoning services (i.e., consistency checking and classification) (see Section 4).

The rest of this paper is organized as follows: Section 2 introduces briefly DLs and quasi-classical semantics. Section 3 implements PROSE and re-implements ParOWL within this framework. Section 4 evaluates experimental results. Finally, Section 5 discusses related works and Section 6 summarizes the paper.

2 Preliminaries

In this section, we briefly introduce description logics, as a logical foundation of OWL, and quasi-classical semantics. For more comprehensive background knowledge of DLs and quasi-classical semantics, we refer the reader to some basic references \[1,47\].

2.1 Syntax of description logics

In description logics (DLs), elementary descriptions are concept names (unary predicates) and role names (binary predicates). Complex descriptions are built from them inductively using concept and role constructors provided by the particular DLs under consideration. In this section, we review the syntax and semantics of DLs.

Let \( N_C, N_R, \) and \( N_I \) be countably infinite sets of concept names, role names, and individual names. \( N_R = R_A \cup R_D \) where \( R_A \) is a set of abstract role names and \( R_D \) is a set of concrete role names. The set of roles is then \( N_R \cup \{ R^- | R \in N_R \} \) where \( R^- \) is the inverse role of \( R \). The function \( Inv(\cdot) \) is defined on the sets of roles as follows, where \( R \) is a role name:

\[ Inv(R) = R^- \quad \text{and} \quad Inv(R^-) = R \]

For roles \( R_1 \) and \( R_2 \), a role axiom is either a role inclusion, which is of the form \( R_1 \sqsubseteq R_2 \) for \( R_1, R_2 \in R_A \) or \( R_1, R_2 \in R_D \), or a transitivity axiom, which is of the form \( \text{Trans}(R) \) for \( R \in R_A \). A role hierarchy \( R \) (or an RBox) is a finite set of role axioms. Let \( \mathbb{R} \) be the reflexive-transitive closure of \( \sqsubseteq \) on \( R \) as follows:

\[ \{(R_1, R_2) | R_1 \subseteq R_2 \in \mathbb{R} \text{ or } Inv(R_1) \subseteq Inv(R_2) \in \mathbb{R}\} \]

A role \( R \) is transitive in \( \mathbb{R} \), if a role \( R' \) exists such that \( R' \mathbb{R} R, R \mathbb{R} R', \) and either \( \text{Trans}(R') \in \mathbb{R} \) or \( \text{Trans}(Inv(R')) \in \mathbb{R} \). A role \( S \) is simple if no transitive role \( R \) exists such that \( R \mathbb{R} S \). \( R^{\mathbb{R}} \) denotes the transitive closure of \( R \).

Concrete datatypes are used to represent literal values such as numbers and strings. A type system typically defines a set of “primitive” datatypes, such as \textit{string} or \textit{integer}, and provides a mechanism for deriving new datatypes from existing ones. To represent concepts such as “a person whose age is at least 21”, a set of concrete datatypes \( D \) is given, and, with each \( d \in D \), a set \( d^D \subseteq \Delta_D \) is associated, where \( \Delta_D \) is the domain of all datatypes.

A set of datatypes is \textit{conforming} if it satisfies the above criteria.

\[
C, D \rightarrow A | d | \top | \bot | \neg C | C \cup D | C \cap D | \exists R.C | \forall R.C |
\{\} | \geq nR | \leq nR | \exists T.d | \forall T.d,
\]

(1)
where \( a \in N_I \), \( C, D \) concepts, \( R \) an abstract role, \( T \) a concrete role, \( S \) a simple role and \( d \in D \) a concrete datatype.

Note that the disjunction of nominals \( \{ o_1 \} \sqcup \cdots \sqcup \{ o_m \} \), where \( o_i \) (\( 1 \leq i \leq m \)) and \( m \) is a positive integer, is still taken as a nominal, denoted by \( \{ o_1, \ldots, o_m \} \). Indeed, nominals can be technically treated as complex concepts.

<table>
<thead>
<tr>
<th>Table 1: Syntax and semantics of DLs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Elements</strong></td>
</tr>
<tr>
<td>individual ((N_I))</td>
</tr>
<tr>
<td>atomic concept ((N_C))</td>
</tr>
<tr>
<td>abstract role ((R_A))</td>
</tr>
<tr>
<td>concrete role ((R_D))</td>
</tr>
<tr>
<td>datatype ((D))</td>
</tr>
<tr>
<td>inverse abstract role</td>
</tr>
<tr>
<td>transitive abstract role</td>
</tr>
<tr>
<td><strong>Complex Concepts</strong></td>
</tr>
<tr>
<td>top concept</td>
</tr>
<tr>
<td>bottom concept</td>
</tr>
<tr>
<td>conjunction</td>
</tr>
<tr>
<td>disjunction</td>
</tr>
<tr>
<td>exist restriction</td>
</tr>
<tr>
<td>value restriction</td>
</tr>
<tr>
<td>existential restriction</td>
</tr>
<tr>
<td>function ((F))</td>
</tr>
<tr>
<td>nominal ((O))</td>
</tr>
<tr>
<td>number restriction</td>
</tr>
<tr>
<td>least restriction</td>
</tr>
<tr>
<td>atmost restriction</td>
</tr>
<tr>
<td>atmost restriction</td>
</tr>
<tr>
<td>datatype exists</td>
</tr>
<tr>
<td>datatype value</td>
</tr>
<tr>
<td>equality ((\text{UNA}))</td>
</tr>
<tr>
<td>inequality ((\text{UNA}))</td>
</tr>
<tr>
<td><strong>Axioms</strong></td>
</tr>
<tr>
<td>concept inclusion ((\mathcal{AC}))</td>
</tr>
<tr>
<td>concept definition ((\mathcal{AC}))</td>
</tr>
<tr>
<td>nominals inclusion ((\mathcal{O}))</td>
</tr>
<tr>
<td>role inclusion ((\mathcal{R}))</td>
</tr>
</tbody>
</table>
The set of concepts is the smallest set such that each concept name \( A \in N_C \) is a concept, and complex concept in OWL are formed according to the following syntax rules by using the operators shown in Table 1.

A terminology or a TBox \( T \) is a finite set of general concept inclusion axioms (GCIs) \( C \sqsubseteq D \) (possibly contains nominals and datatypes in the language of \( O \)). In an ABox, one describes a specific state of affairs of an application domain in terms of concept and roles. It is the statement about how concepts are related to each other. An ABox \( A \) is a finite set of assertions of the forms \( C(a) \) (concept assertion), \( R(a,b) \) (role assertion), \( a = b \) (equality assertion), and \( a \neq b \) (inequality assertion). A knowledge base (KB) \( K \) (or ontology) is a triple \( (R,T,A) \).

The semantics is given by means of interpretations. An interpretation \( I_c = (\Delta I_c, \cdot I_c) \) consists of a non-empty domain \( \Delta I_c \), disjoint from the concrete domain \( \Delta D \), and a mapping \( \cdot I_c \) which maps concepts, roles, and nominals according to Table 1 (\( \# \) denotes set cardinality).

Let \( K \) be a KB. We use \( \text{Mod}(K) \) to denote the collection of all models of \( K \). \( K \) is consistent if \( \text{Mod}(K) \neq \emptyset \) and inconsistent otherwise. Consistency is an important property of KBs. For instance, consider a KB \( K = (\{A \sqsubseteq B\}, \{A(a), \neg B(a)\}) \). Clearly, \( K \) is inconsistent. Note that most reasoning services can be reduced to the consistency problem.

### 2.2 Quasi-classical semantics

In syntax, quasi-classical description logic (QCDL) slightly extends the syntax of classical DLs by introducing the quasi-classical negation (QC negation) of a concept. The QC negation of a concept \( C \) is denoted by \( \neg C \).

The quasi-classical semantics is characterized by two interpretations, namely, weak interpretation and strong interpretation, which are built on base interpretation defined as follows.

A base interpretation \( \mathcal{I} \) is a pair \( (\Delta^\mathcal{I}, \mathcal{I}) \) where the domain \( \Delta^\mathcal{I} \) is a set of individuals, \( \Delta_D \) a concrete domain of datatypes and the assignment function \( \cdot^\mathcal{I} \) assigns each individual to an element of \( \Delta^\mathcal{I} \) and assigns:

\[
\begin{align*}
\top^\mathcal{I} &= (\Delta^\mathcal{I}, \emptyset); \\
\bot^\mathcal{I} &= (\emptyset, \Delta^\mathcal{I}); \\
\neg C^\mathcal{I} &= (+A, A); \\
R^\mathcal{I} &= (+R, -R); \\
(R^-)^\mathcal{I} &= (+R^-, -R^-); \\
(R^{tc})^\mathcal{I} &= (+R^{tc}, -R^{tc}); \\
d^\mathcal{I} &= (+d, -d); \\
T^\mathcal{I} &= (+T, -T).
\end{align*}
\]

Here \( \pm A, N \subseteq \Delta^\mathcal{I}, \pm R, \pm R^- \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}, \pm T \subseteq \Delta^\mathcal{I} \times \Delta_D \), and \( \pm d^D \subseteq \Delta_D \).

Note that \( +X \) and \( -X \) are not necessarily disjoint when \( X \in \{C, R, T, d\} \). Intuitively, \( +X \) is the set of elements known to be in the extension of \( X \) while \( -X \) is the set of elements known to be not in the extension of \( X \).
A weak interpretation \( I \) is a base interpretation \( (\Delta^I, I) \) such that the assignment function \( -^I \) satisfies the following conditions:

\[
(C \cap D)^I = (\{ +C \cap +D, -C \cup -D \}); \\
(C \cup D)^I = (\{ +C \cup +D, -C \cap -D \}); \\
(\neg C)^I = (\{ -C, +C \}); \\
(\neg C)^I = (\{ \Delta^I \setminus +C, \Delta^I \setminus -C \}); \\
(\exists R.C)^I = (\{ \{ x \mid \exists y, \langle x, y \rangle \in +R \text{ and } y \in +C \} \\ x \mid \forall y, \langle x, y \rangle \in +R \text{ implies } y \in -C \}); \\
(\forall R.C)^I = (\{ \{ x \mid \forall y, \langle x, y \rangle \in +R \text{ implies } y \in +C \} \\ x \mid \exists y, \langle x, y \rangle \in +R \text{ and } y \in -C \}); \\
\{ a \}^I = (\{ a^I \}, N) \text{ where } N \subseteq \Delta^I; \\
(\neg d)^I = (\{ -d^I, +d^I \}) \text{ where } d^I = (\{ +d^I, -d^I \}); \\
(\neg nR.C)^I = (\{ \{ x \mid \exists \{ \langle y, x \rangle \in +R \} \geq n \}, \{ x \mid \exists \{ \langle y, x \rangle \in +R \} < n \} \\ x \mid \forall \{ \langle y, x \rangle \in +R \} \leq n \}, \{ x \mid \forall \{ \langle y, x \rangle \in +R \} > n \}); \\
(\forall T.d)^I = (\{ \{ x \mid \forall \{ \langle y, x \rangle \in +T \text{ and } y \in +d^I \} \\
\{ x \in \Delta^I \mid \forall y, \langle x, y \rangle \in +T \text{ implies } y \in -d^I \}; \\
(\exists T.d)^I = (\{ \{ x \mid \exists \{ \langle y, x \rangle \in +T \text{ implies } y \in +d^I \}, \{ x \in \Delta^I \mid \forall y, \langle x, y \rangle \in +T \text{ and } y \in -d^I \}. \\
\}

Let \( I \) be a weak interpretation. A weak satisfaction \((=_w)\) is defined as follows: let \( X^I = (\{ +X, -X \}) \) where \( X \in \{ C, D, d, R, S \} \),

- \( I =_w C(a) \text{ if } a^I \in +C; \)
- \( I =_w C \subseteq D, \text{ if } +C \subseteq +D; \)
- \( I =_w R(a, b), \text{ if } \langle a^I, b^I \rangle \in +R; \)
- \( I =_w d_1 \subseteq d_2 \text{ if } +d_1 \subseteq +d_2; \)
- \( I =_w R \subseteq S \text{ if } +R \subseteq +S, \text{ for any role } R, S \in R_A \text{ or } R, S \in R_D; \)
- \( I =_w \text{Trans}(R) \text{ if } +R = (+R)^w, \text{ for any abstract } R, S \in R_A; \)
- \( I =_w a \dashv b \text{ if } a^I = b^I; \)
- \( I =_w a \not\dashv b \text{ if } a^I \neq b^I. \)

A strong interpretation \( I \) is a base interpretation \( (\Delta^I, I) \) such that the assignment function \( -^I \) satisfies the conditions in the definition of weak interpretation except that the conjunction and the disjunction of concepts are interpreted as follows: let \( C^I = (\{ +C, -C \}) \) and \( D^I = (\{ +D, -D \}); \\
(C \cap D)^I = (\{ +C \cap +D, -C \cup -D \}) \cap (\{ +C \cup +D \}) \cap (\{ +C \cup -D \}) \cap (\{ +C \cup -D \}); \\
(C \cup D)^I = (\{ +C \cup +D, -C \cap -D \}) \cap (\{ +C \cup +D \}) \cap (\{ +C \cup -D \}) \cap (\{ +C \cup -D \}). \quad (2) \)

Let \( I \) be a strong interpretation. A strong satisfaction \((=_s)\) is defined as the same of the weak satisfaction on axioms except for concept/role inclusions as follows: let \( X^I = (\{ +X, -X \}) \) where \( X \in \{ C, D, d, d_1, d_2 \}, \)

\[
7
\]
– \( \mathcal{I} \models s C \subseteq D \), if \( -C \subseteq +D, +C \subseteq +D \) and \( -D \subseteq -C \);
– \( \mathcal{I} \models s d_1 \subseteq d_2 \), if \( -d_1 \subseteq +d_2, +d_1 \subseteq +d_2 \) and \( -d_2 \subseteq -d_1 \);
– \( \mathcal{I} \models s \{o_1, \ldots, o_s\} \subseteq \{o'_1, \ldots, o'_t\} \) if \( \mathcal{N}_1 \subseteq \{o''_1, \ldots, o''_r\} \), \( N_2 \subseteq N_1 \) and \( \{o''_1, \ldots, o''_r\} \subseteq \{o'_1, \ldots, o'_t\} \);
– \( \mathcal{I} \models s R \subseteq S \), if \( -R \subseteq +S, +R \subseteq +S \) and \( -S \subseteq -R \), for any role \( R, S \in R_A \) or \( R, S \in R_D \).

Let \( \mathcal{K} \) be a QC KB and \( \mathcal{I} \) be a strong interpretation. \( \mathcal{I} \) is a quasi-classical model (QC model) of \( \mathcal{K} \) if \( \mathcal{I} \models s \varphi \) for all axiom \( \varphi \) of \( \mathcal{K} \). We use \( \text{Mod}^s(\mathcal{K}) \) to denote the collection of all QC models of \( \mathcal{K} \). \( \mathcal{K} \) is QC consistent if \( \text{Mod}^s(\mathcal{K}) \neq \emptyset \) and QC inconsistent otherwise. The QC-consistency problem is to determine whether a QC KB is QC consistent. Note that the QC-consistency problem is ExpTime-complete [43]. We call \( \mathcal{K} \) quasi-classically entails (QC entails) \( \varphi \), denoted \( \mathcal{K} \models Q \varphi \), if for every base interpretation \( \mathcal{I} \), \( \mathcal{I} \models s \mathcal{K} \) implies \( \mathcal{I} \models w \varphi \). In this case, \( \models Q \) is called QC entailment.

**Lemma 1.** [47] Let \( \mathcal{T} \) be a terminology, \( \mathcal{R} \) a role hierarchy, \( \mathcal{A} \) an ABox, and \( C, D \) concepts. For any base interpretation \( \mathcal{I} \), we interpret \( U^\mathcal{I} = (\Delta^\mathcal{I} \times \Delta^\mathcal{I}, \emptyset) \).

Then

1. \( (\mathcal{T}, \mathcal{R}, \mathcal{A}) \models Q C(a) \) if and only if \( (\mathcal{T}, \mathcal{R}, \mathcal{A} \cup \{C(a)\}) \) is QC inconsistent w.r.t. \( \mathcal{R}_U \);
2. \( (\mathcal{T}, \mathcal{R}, \emptyset) \models Q C \subseteq D \) if and only if \( (\mathcal{T}, \mathcal{R}, \{(C \cap \overline{D})(t)\}) \) is QC inconsistent w.r.t. \( \mathcal{R}_U \) for some new individual \( t \in \Delta \).

By Lemma 1, we can find that two reasoning problems, namely, instance checking and classification (or subsumption), can be reduced to the consistency problem.

### 3 PROSE: a plugin-based framework

In this section, we firstly introduce a framework called PROSE for quasi-classical paraconsistent reasoning for OWL ontologies where inconsistency is tolerated in a plugin via a paraconsistent transformation which is independent of reasoning engine. Finally, we implement PROSE and re-implement ParOWL by employing three OWL reasoners: Pellet [38], HermiT [37], and FaCT++ [41].

#### 3.1 PROSE architecture

This framework is called PROSE (paraconsistent reasoning on semantic web). The architecture of PROSE is shown in Figure 1. PROSE is designed in the Decorator Pattern [10] extending the inner classical OWL reasoner so that paraconsistent reasoning on inconsistent KBs can be performed. The module Strong Transformer rewrites the input KB \( \mathcal{K} \) to a new KB \( \mathcal{S}(\mathcal{K}) \) while the module Weak Transformer changes the input query \( \varphi \) into a new query \( \mathcal{W}(\varphi) \). Then the inner classical OWL reasoner is called.
In the initial version of PROSE [47], the only classical OWL reasoner Pellet is implemented, which is an open source Java based reasoner for OWL 2 DL developed by the Mind Swap group. PROSE is thus limited by some disadvantages of Pellet which are also limitations in reasoning on consistent ontologies.

In the current version of PROSE, we have extended PROSE by including the other two classical OWL reasoners: HermiT and FaCT++. Both of them are widely used and have outstanding performance in specific reasoning tasks [6]. PROSE has good performance in reasoning over both consistent and inconsistent ontologies. It is evaluated empirically on various ontologies (e.g., containing large-scale of classes or properties or individuals) in doing specific reasoning task and results are presented in the following section. We give a critical estimation before using PROSE to perform paraconsistent reasoning. So, we can take full advantage of the highlights of different OWL reasoners through PROSE.

Based on the transformation algorithms, PROSE rewrites the input ontology to a new ontology which is then taken as input by the inner classical reasoner.

### 3.2 Quasi-classical transformer

Firstly, we introduce quasi-classical transformer serving for transformation as a plugin in the architecture of PROSE. Note that the quasi-classical transformer contains two transformers: weak transformer and strong transformer. The weak transformation is stated in Table 2 where NA is a new concept name.

The strong transformer $S(\cdot)$ is identical with the weak transformation $W(\cdot)$ except for disjunctions, the negation of conjunctions, and GCI which are defined as follows: where $X_i \in \{C_i, R_i\} \ (i = 1, 2)$,

$$S(\neg (C \sqcup D)) = S(\neg C \sqcup \neg D);$$
$$S(C \sqcup D) = (S(C) \sqcup S(D)) \sqcap (\neg S(\neg C) \sqcup \neg S(\neg D)) \sqcap (C \sqcup \neg S(\neg D));$$
$$S(X_1 \sqsubseteq X_2) = \{S(X_1) \sqsubseteq S(X_2), S(\neg X_1) \sqsubseteq S(\neg X_2), S(\neg X_1) \sqsubseteq S(\neg X_1) \sqsubseteq X_2\};$$
$$S(\{o_1, \ldots, o_m\}) = S(\{o_1\} \sqcup \ldots \sqcup \{o_m\});$$
$$S(\neg \{o_1, \ldots, o_m\}) = S(\neg \{o_1\}) \sqcap \ldots \sqcap S(\neg \{o_m\}).$$
Table 2: Weak Transformation Rules for OWL

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Weak transformation $W(\cdot)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\top$</td>
<td>$NA \sqcup \neg NA$</td>
</tr>
<tr>
<td>$\bot$</td>
<td>$NA \sqcap \neg NA$</td>
</tr>
<tr>
<td>$\neg \top$</td>
<td>$NA \sqcap \neg NA$</td>
</tr>
<tr>
<td>$\neg \bot$</td>
<td>$NA \sqcup \neg NA$</td>
</tr>
<tr>
<td>$X$ ($X \in {A, R, S, T, d}$)</td>
<td>$X^+$</td>
</tr>
<tr>
<td>$\neg X$ ($X \in {A, R, S, T, d}$)</td>
<td>$X^-$</td>
</tr>
<tr>
<td>${\emptyset}$</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>$\neg \emptyset$</td>
<td>$\neg W(C)$</td>
</tr>
<tr>
<td>$C$</td>
<td>$W(C)$</td>
</tr>
<tr>
<td>$\neg C$</td>
<td>$W(C)$</td>
</tr>
<tr>
<td>$C \sqcap D$</td>
<td>$W(C) \sqcap W(D)$</td>
</tr>
<tr>
<td>$\neg (C \sqcap D)$</td>
<td>$W(\neg C) \sqcup W(\neg D)$</td>
</tr>
<tr>
<td>$C \sqcup D$</td>
<td>$W(C) \sqcup W(D)$</td>
</tr>
<tr>
<td>$\neg (C \sqcup D)$</td>
<td>$W(\neg C) \sqcap W(\neg D)$</td>
</tr>
<tr>
<td>$\exists R.C$</td>
<td>$\exists W(R).W(C)$</td>
</tr>
<tr>
<td>$\forall R.C$</td>
<td>$\forall W(R).W(C)$</td>
</tr>
<tr>
<td>$\neg (\exists R.C)$</td>
<td>$\forall W(R).W(\neg C)$</td>
</tr>
<tr>
<td>$\neg (\forall R.C)$</td>
<td>$\exists W(R).W(\neg C)$</td>
</tr>
<tr>
<td>$\exists T.d$</td>
<td>$\exists W(T).W(d)$</td>
</tr>
<tr>
<td>$\forall T.d$</td>
<td>$\forall W(T).W(d)$</td>
</tr>
<tr>
<td>$\neg (\exists T.d)$</td>
<td>$\forall W(T).W(\neg d)$</td>
</tr>
<tr>
<td>$\neg (\forall T.d)$</td>
<td>$\exists W(T).W(\neg d)$</td>
</tr>
<tr>
<td>$\geq n.S$</td>
<td>$\geq n.W(S)$</td>
</tr>
<tr>
<td>$\leq n.S$</td>
<td>$\leq n.W(S)$</td>
</tr>
<tr>
<td>$\neg (\geq n.S)$</td>
<td>$\leq (n - 1).W(S)$</td>
</tr>
<tr>
<td>$\neg (\leq n.S)$</td>
<td>$\geq (n + 1).W(S)$</td>
</tr>
<tr>
<td>$\geq n.R.C$</td>
<td>$\geq n.W(R).W(C)$</td>
</tr>
<tr>
<td>$\leq n.R.C$</td>
<td>$\leq n.W(R).W(C)$</td>
</tr>
<tr>
<td>$\neg (\leq n.R.C)$</td>
<td>$\geq (n + 1).W(R).W(\neg C)$</td>
</tr>
<tr>
<td>$\neg (\geq n.R.C)$</td>
<td>$\leq (n - 1).W(R).W(\neg C)$</td>
</tr>
<tr>
<td>$C(a)$</td>
<td>$W(C)[a]$</td>
</tr>
<tr>
<td>$C \sqsubseteq D$</td>
<td>$W(C) \sqsubseteq W(D)$</td>
</tr>
<tr>
<td>$R(a,b)$</td>
<td>$W(R)[a,b]$</td>
</tr>
<tr>
<td>$R_1 \sqsubseteq R_2$</td>
<td>$W(R_1) \sqsubseteq W(R_2)$</td>
</tr>
<tr>
<td>$\text{Trans}(R)$</td>
<td>$\text{Trans}(W(R))$</td>
</tr>
<tr>
<td>$a \neq b$</td>
<td>$a \neq b$</td>
</tr>
<tr>
<td>$a \equiv b$</td>
<td>$a \equiv b$</td>
</tr>
</tbody>
</table>

We have the following lemma [47].

**Lemma 2.** [47] Let $K$ be a QC KB and $\varphi$ a QC axiom. Then

1. $K$ is QC consistent if and only if $S(K)$ is consistent;
2. \( \mathcal{K} \models Q \varphi \) if and only if \( S(\mathcal{K}) \models W(\varphi) \).

By Lemma 2, we can reduce the reasoning problems of QCDL to the reasoning problems of DL.

Next, let us consider an example in [47] to show how the quasi-classical transformer work.

For instance, let \( \mathcal{K} = \{ \{ \text{Bird} \sqsubseteq \text{Fly}, \text{Penguin} \sqsubseteq \text{Bird}, \text{Penguin} \sqsubseteq \neg \text{Fly}, \text{Swallow} \sqsubseteq \text{Bird} \}, \{ \text{Penguin}(\text{tweety}), \text{Swallow}(\text{fred}) \} \) be a KB.

(1) \( \mathcal{K} \) will be transformed to

\[
S(\mathcal{K}) = \{ \{ \text{Bird}^+, \text{Fly}^-, \text{Bird}^-, \neg \text{Bird}^+ \sqsubseteq \text{Fly}^+, \\
\text{Penguin}^+ \sqsubseteq \text{Bird}^+, \text{Bird}^+ \sqsubseteq \text{Penguin}^-, \neg \text{Penguin}^- \sqsubseteq \text{Bird}^+, \\
\text{Penguin}^+ \sqsubseteq \neg \text{Fly}, \text{Fly}^+ \sqsubseteq \text{Penguin}^-, \neg \text{Penguin}^- \sqsubseteq \text{Fly}^-, \\
\text{Swallow}^+ \sqsubseteq \text{Bird}^+, \text{Bird}^+ \sqsubseteq \text{Swallow}^-, \neg \text{Swallow}^- \sqsubseteq \text{Bird}^+, \\
\{ \text{Penguin}^+(\text{tweety}), \text{Swallow}^+(\text{fred}) \} \}.
\]

(2) The query \( \varphi = \text{Fly}(\text{fred}) \) will be transformed to \( W \varphi = \text{Fly}(\text{fred})^+ \).

(3) Querying \( \mathcal{K} \models Q \text{Fly}(\text{fred}) \) can be reduced to \( S(\mathcal{K}) \models \text{Fly}^+(\text{fred}) \). As a result, the answer to \( \text{Fly}(\text{fred}) \) is “yes” since \( S(\mathcal{K}) \cup \{ \neg \text{Fly}^+(\text{fred}) \} \) is inconsistent, that is, \( S(\mathcal{K}) \models \text{Fly}^+(\text{fred}) \). These knowledge (\( \text{Swallow}(\text{fred}) \) and \( \text{Swallow} \sqsubseteq \text{Bird} \) and \( \text{Bird} \sqsubseteq \text{Fly} \)) representing \( \text{fred} \) does not contain conflict although those knowledge (\( \text{Penguin}(\text{tweety}), \text{Penguin} \sqsubseteq \text{Bird}, \text{Bird} \sqsubseteq \text{Fly} \) and \( \text{Penguin} \sqsubseteq \neg \text{Fly} \)) representing \( \text{tweety} \) contains some conflict. Intuitively, it is reasonable that \( \text{fred} \) can fly.

### 3.3 Implementation of ParOWL in PROSE

ParOWL [31] is a paraconsistent OWL reasoner for four-valued semantics [24,25,31].

There are three kinds of inclusions, namely material inclusion (\( C \rightarrow D \)), internal inclusion (\( C \sqsubseteq D \)), and strong inclusion (\( C \rightarrow D \)). Semantically, for any base interpretation \( \mathcal{I} \),

- material inclusion: \( \mathcal{I} \models_w C \rightarrow D \) if \( \Delta^+ \setminus (-C) \subseteq +D \);
- internal inclusion: \( \mathcal{I} \models_w C \sqsubseteq D \) if \( +C \subseteq +D \);
- strong inclusion: \( \mathcal{I} \models_w C \rightarrow D \) if \( +C \subseteq +D \) and \( -D \subseteq -C \);

where \( C^+ = (+C, -C) \) and \( D^+ = (+D, -D) \).

To characterize the three inclusions of four-valued semantics, based on quasi-classical transfer, we firstly introduce three transfers called material inclusion transfer (\( W_M \)), internal inclusion transfer (\( W_I \)), and strong inclusion transfer (\( W_S \)) respectively. The three new transfers are obtained from weak transfer by revising the definition of inclusions as follows [25]:

- material inclusion transformer: \( W_M(C \rightarrow D) = \neg W_M(\neg C) \sqsubseteq W_M(D) \);
- internal inclusion transformer: \( W_I(C \sqsubseteq D) = W_I(C) \sqsubseteq W_I(D) \);
- strong inclusion transformer: \( W_S(C \rightarrow D) = W_I(C) \sqsubseteq W_I(D) \) and \( W_I(\neg D) \sqsubseteq W_I(C) \).
Based on PROSE architecture, we can construct three frameworks for ParOWL named ParOWL$_M$, ParOWL$_I$, and ParOWL$_S$ respectively (see Figure 2) where $X$ is a placeholder of $M, I, S$.

![Diagram of ParOWL in PROSE architecture]

**Fig. 2: ParOWL in PROSE architecture**

Compared with two transformers in PROSE architecture (see Figure 1), ParOWL contains only one transformer.

### 4 Experimental evaluations

We have implemented PROSE as a plugin-based framework for paraconsistent reasoning. Besides the QC-transformation proposed in Section 3.2, PROSE also re-implement three transformers of ParOWL in its unified framework according to the architecture in Fig 2. For all transformations, PROSE provide three popular OWL reasoners (e.g., Pellet, HermiT, and FaCT++) to perform the reasoning tasks after transforming. In addition, other OWL reasoners can be easily integrated into PROSE and so we get two benefits a) user can select the best one among classical OWL reasoners for its specific reasoning task through PROSE and b) PROSE can take full advantage of any improvement of OWL reasoners automatically.

The transformation algorithms are implemented in Java 7 and built on OWL-API 3.5.0 [14], which is a Java programming interface for manipulating OWL ontologies. The procedure is designed in the Visitor Pattern [10] which is widely used in OWL-API. Since OWL-API is a general implementation for OWL ontologies and contains functions that do not have a tightly specified functionality, we use it to link PROSE with different classical reasoners. We have developed different programs for Pellet version 2.3.0, HermiT version 1.3.8, and FaCT++ version 1.6.2, but all of the programs are through specific interfaces in OWL-API 3.5.0.
The three kinds of transformations of ParOWL implemented in PROSE are denoted by ParOWL_M, ParOWL_I, and ParOWL_S, corresponding to material, internal, and strong transformer in Fig 2.

4.1 Evaluation setup

Since PROSE is a unified framework for paraconsistent reasoning, it can give further evaluation and comparison among different transformations and reasoners. We performed a series of experiments by applying PROSE to inconsistent ontologies for paraconsistent reasoning from multiple dimensions including:

- transformation: both QC-transformation and three transformations of ParOWL were evaluated for handling inconsistencies in ontologies;
- reasoner: three popular reasoners Pellet, HermiT, and FaCT++ were used to perform reasoning tasks;
- reasoning task: classification and consistency checking were as two reasoning tasks;
- ontology: LUBM [12] ontologies and generated ontologies were used as benchmark to capture the rich characteristics of ontologies.

Two groups of ontologies were used as benchmark and modified for testing. One was created by using the Lehigh Benchmark (LUBM) suite [12] which provides a tool for generating ontologies populated with a variable number of universities, departments, professors, etc. Since ontologies created by LUBM suite vary in the number of individuals and have a fixed number of classes and properties, we generated another group of ontologies according to the increasing number of one component (e.g., classes) with the fixed number of others (e.g., properties and individuals) for further evaluation. Details about the benchmark ontologies are shown in the following subsections.

Since PROSE is independent on OWL reasoners and has integrated Pellet, HermiT, and FaCT++, we evaluated the performance of these three classical OWL reasoners in paraconsistent reasoning. The experimental results can give more insight how different OWL reasoners functions. An online demo is developed for intuitively presenting the framework of PROSE.

The experiments were performed under Ubuntu Server 14.04 on 16 Intel Xeon E5620, 2.40GHz processors with 32GB memory. The program was run on Java 1.7 with maximum 16GB heap space allocated for JVM. We performed two classical reasoning tasks, namely consistency checking and classification, on each ontology and measured the runtime. All stated runtimes are averaged over five independent runs. Note that all results are in milliseconds and a 60 minute time-limit is used.

4.2 Evaluation results

In this subsection, we evaluate the four transformations ParOWL_M, ParOWL_I, ParOWL_S, and QC-transformation (denoted by PROSEQ) under two reasoning

1http://123.56.79.184/prose
tasks: checking consistency (i.e., the classical consistency of ontologies after transformed) and classification.

**LUBM ontologies** Table 3 describes the ontologies generated by LUBM suite. For the evaluations, the default settings of the tools are used and five ontologies are created with the number of universities from one to five. Inconsistent versions of the resulting ontologies are created by manually adding disjoint class axioms (e.g., person and organization).

In Table 3, we use NC, NP, and NI to denote the number of classes, properties, and individuals of ontologies, respectively.

The growth of classes refer to original size after using different transformations is given in Table 4. We can see that the number of classes increases by different factors depending on which transformation is used. The first two transformations (material transformation and internal transformation) bring 1.65 and 1.96 times in size respectively while QC-transformation in PROSE and strong transformation bring 2.65 times in size. It is reasonable that the growth is due to their more complicated transformation rules in QC-transformation and strong transformation. We also can notice that the growth of classes in size is invariable among five LUBM ontologies since the strategy in ontologies generated by LUBM suite mainly works on individuals. In other words, the scale of LUBM ontologies only depends on the scale of individuals (see Table 3).

**Table 3: The inconsistent LUBM ontologies**

<table>
<thead>
<tr>
<th>Ontology</th>
<th>DL expressivity</th>
<th>#Axioms</th>
<th>NC</th>
<th>NP</th>
<th>NI</th>
</tr>
</thead>
<tbody>
<tr>
<td>lubm1-inc</td>
<td>ALEHI+(D)</td>
<td>100793</td>
<td>58</td>
<td>55</td>
<td>17175</td>
</tr>
<tr>
<td>lubm2-inc</td>
<td>ALEHI+(D)</td>
<td>230311</td>
<td>58</td>
<td>55</td>
<td>38335</td>
</tr>
<tr>
<td>lubm3-inc</td>
<td>ALEHI+(D)</td>
<td>337377</td>
<td>58</td>
<td>55</td>
<td>55665</td>
</tr>
<tr>
<td>lubm4-inc</td>
<td>ALEHI+(D)</td>
<td>478034</td>
<td>58</td>
<td>55</td>
<td>78580</td>
</tr>
<tr>
<td>lubm5-inc</td>
<td>ALEHI+(D)</td>
<td>624782</td>
<td>58</td>
<td>55</td>
<td>102369</td>
</tr>
</tbody>
</table>
Table 4: Growth of Classes after transformation (relative to original)

<table>
<thead>
<tr>
<th>Ontology</th>
<th>ParOWL₂₅₀</th>
<th>ParOWL₋₁</th>
<th>ParOWL₊₀</th>
<th>PROSE₂₅₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>lubm1-inc</td>
<td>1.65</td>
<td>1.96</td>
<td>2.50</td>
<td>2.50</td>
</tr>
<tr>
<td>lubm2-inc</td>
<td>1.65</td>
<td>1.96</td>
<td>2.50</td>
<td>2.50</td>
</tr>
<tr>
<td>lubm3-inc</td>
<td>1.65</td>
<td>1.96</td>
<td>2.50</td>
<td>2.50</td>
</tr>
<tr>
<td>lubm4-inc</td>
<td>1.65</td>
<td>1.96</td>
<td>2.50</td>
<td>2.50</td>
</tr>
<tr>
<td>lubm5-inc</td>
<td>1.65</td>
<td>1.96</td>
<td>2.50</td>
<td>2.50</td>
</tr>
</tbody>
</table>

Analysis  Figure 3 shows the average time required to reasoning over inconsistent LUBM ontologies with PROSE₂₅₀ for handling inconsistency, using Pellet, HermiT, and FaCT++. We find that HermiT exceeds the time-limit in dealing with the inconsistent LUBM ontologies. However, for classification, neither HermiT nor FaCT++ can produce the final results in the limit of time. In other words, both HermiT and FaCT++ take disadvantage of reasoning over ontologies with the big scale of individuals.

Figure 4 and 5 describe the performance of ParOWL from multiple perspectives, based on LUBM benchmark ontologies. Figure 4 displays the average time required to check consistency after three transformations in ParOWL, using three classical OWL reasoners. Classification time is shown in Figure 5 in the same way. It is shown that, no matter which kind of transformation is used, the consuming time increases significantly with the growth of ontologies. This is clearly due to more axioms in larger size of ontologies, which have fixed number of classes though.

The results show that PROSE, as a framework of paraconsistent reasoning, has exactly the capacity for comparing the different methods of handling inconsistency from multiple dimensions in a unified and convenient way.
Fig. 3: Reasoning on inconsistent LUBM ontologies using PROSEQ

Fig. 4: Consistency checking on inconsistent LUBM ontologies using ParOWL
Generated ontologies LUBM suite can only characterize ontologies with large number of individuals. However, ontologies with lots of classes or properties (e.g., biological ontologies) are widely used in practice. It is not enough to characterize the full graph of performance of existing OWL reasoners among all three dimensions (individual, class, property) by LUBM suite. To compare them in a fair way, we generate another group of ontologies according to the increasing number of one component (e.g., classes) with the fixed number of others (e.g., properties and individuals) for further evaluation on diverse ontologies. Inconsistent versions of the resulting ontologies are created in the similar way with inconsistent LUBM ontologies.

Table 5 describes the ontologies of rich characteristics. Nine tested ontologies are generated according to the increasing number of one component (e.g., classes) with the fixed number of others (e.g., properties and individuals). For instance, three ontologies (namely 100Class, 1000Class, and 10000Class in the table) consist of 15 properties, 3 individuals, and 100, 1000, 10000 classes respectively. Disjoint class axioms and some individuals are added for inconsistent versions. The relative number of classes after different transformations is given in Table 6.
Table 5: Generated inconsistent ontologies

<table>
<thead>
<tr>
<th>Ontology</th>
<th>DL Expressivity</th>
<th>#Axioms</th>
<th>NC</th>
<th>NP</th>
<th>NI</th>
</tr>
</thead>
<tbody>
<tr>
<td>100Class</td>
<td>ALCHI</td>
<td>318</td>
<td>100</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>1000Class</td>
<td>ALCHI</td>
<td>2732</td>
<td>1000</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>10000Class</td>
<td>ALCHI</td>
<td>29269</td>
<td>10000</td>
<td>15</td>
<td>3</td>
</tr>
<tr>
<td>100Property</td>
<td>ALCHI</td>
<td>479</td>
<td>100</td>
<td>105</td>
<td>3</td>
</tr>
<tr>
<td>1000Property</td>
<td>ALCHI</td>
<td>2407</td>
<td>100</td>
<td>1005</td>
<td>3</td>
</tr>
<tr>
<td>10000Property</td>
<td>ALCHI</td>
<td>21625</td>
<td>100</td>
<td>10005</td>
<td>3</td>
</tr>
<tr>
<td>100Individual</td>
<td>ALCHI</td>
<td>473</td>
<td>100</td>
<td>15</td>
<td>100</td>
</tr>
<tr>
<td>1000Individual</td>
<td>ALCHI</td>
<td>2283</td>
<td>100</td>
<td>15</td>
<td>1000</td>
</tr>
<tr>
<td>10000Individual</td>
<td>ALCHI</td>
<td>20271</td>
<td>100</td>
<td>15</td>
<td>10000</td>
</tr>
</tbody>
</table>

Table 6: Growth of Classes after transformation (relative to original)

<table>
<thead>
<tr>
<th>Ontology</th>
<th>ParOWL1</th>
<th>ParOWL4</th>
<th>ParOWL8</th>
<th>PROSE4</th>
</tr>
</thead>
<tbody>
<tr>
<td>100Class</td>
<td>1.71</td>
<td>1.93</td>
<td>2.77</td>
<td>2.77</td>
</tr>
<tr>
<td>1000Class</td>
<td>2.35</td>
<td>2.50</td>
<td>2.84</td>
<td>2.84</td>
</tr>
<tr>
<td>10000Class</td>
<td>2.84</td>
<td>2.88</td>
<td>2.96</td>
<td>2.96</td>
</tr>
<tr>
<td>100Property</td>
<td>1.79</td>
<td>2.08</td>
<td>2.75</td>
<td>2.75</td>
</tr>
<tr>
<td>1000Property</td>
<td>1.71</td>
<td>2.00</td>
<td>2.72</td>
<td>2.72</td>
</tr>
<tr>
<td>10000Property</td>
<td>1.84</td>
<td>2.10</td>
<td>2.76</td>
<td>2.76</td>
</tr>
<tr>
<td>100Individual</td>
<td>1.95</td>
<td>2.05</td>
<td>2.71</td>
<td>2.71</td>
</tr>
<tr>
<td>1000Individual</td>
<td>2.14</td>
<td>2.14</td>
<td>2.78</td>
<td>2.78</td>
</tr>
<tr>
<td>10000Individual</td>
<td>2.03</td>
<td>2.03</td>
<td>2.71</td>
<td>2.71</td>
</tr>
</tbody>
</table>

18
Analysis  Figure 6 and 7 show the average time for \texttt{PROSEQ} required to checking consistency and classification respectively, over the generated ontologies and using the three popular OWL reasoners. As shown there, FaCT++ has the best performance in most cases, for both consistency checking and classification.

Figure 8, 9, and 10 display the average time required to check consistency on generated ontologies after three transformations in ParOWL, using Pellet, HermiT, and FaCT++ respectively. Classification time is given in Figure 11, 12 and 13 corresponding to Pellet, HermiT, and FaCT++ respectively.

Fig. 6: Consistency checking using \texttt{PROSEQ} on three OWL reasoners

Fig. 7: Classification using \texttt{PROSEQ} on three OWL reasoners
**Fig. 8:** Consistency checking using ParOWL on Pellet

**Fig. 9:** Consistency checking using ParOWL on HermiT

**Fig. 10:** Consistency checking using ParOWL on FaCT++
Fig. 11: Classification using ParOWL on Pellet

Fig. 12: Classification using ParOWL on HermiT

Fig. 13: Classification using ParOWL on FaCT++
As a result, the full comparison of four paraconsistent reasoners (i.e., PROSEQ, ParOWL-M, ParOWL-I, and ParOWL-S) in 18 cases of three dimensions (i.e., individual, class, property), two reasoning tasks (i.e., consistency checking and classification), and three OWL reasoners (i.e., Pellet, HermiT, and FaCT++) can be intuitively given in our unified framework. Additionally, LUBM and generated ontologies can amend each other in order to a fair and complete evaluation.

At the end of this section, we also notice that though PROSEQ takes a little more time than three scenarios of ParOWL (i.e., ParOWL-M, ParOWL-I, and ParOWL-S), the performance of PROSEQ is still in acceptable range (see Figure 6 and 7). This is due to the cost of the QC-transformation [47]. The QC-transformation contains six transformations while those transformations applied in ParOWL are a few of the six transformations [25, Table 5]. For instance, ParOWL-M contains two of six transformations as well as ParOWL-I and ParOWL-S contains three of six transformations. As we well known, it is reasonable to make a trade off between inference power and efficiency. PROSEQ brings a stronger inference power than ParOWL and the performance loss is in acceptable range.

5 Related Works

In this section, we mainly compare this work with existing related works. We will simplify those comparisons between QCDL and existing paraconsistent approaches in OWL since they are well discussed in [47].

Compared with three scenarios of ParOWL [25], PROSE is proven more powerful in paraconsistent reasoning due to its strong transformer. Analogously, compared with two kinds of three-valued DLs, namely, paradoxical DL [46] and three-valued DL [28] are based four-valued DL, PROSE is still more powerful since they are based on four-valued semantics. Additionally, there exist some variants of four-valued DL such as \( \mathcal{PALC} \) presented in [20]. \( \mathcal{PALC} \) is obtained from a description logic (called \( \mathcal{ALC_n} \)) with such a dual (or multiple)-interpretation semantics by adding a weak negation in order to tolerate inconsistency where the weak negation is identical to the classical negation and the classical negation is identical to the QC negation in QCDL. The weak negation is used to tolerate inconsistency and the classical negation is used to implement paraconsistent reasoning. In \( \mathcal{PALC} \), the satisfaction of GCIs is defined by the internal inclusion. In this sense, \( \mathcal{PALC} \) can be taken as our weak semantics for DL. Recently, [16] presents a quasi-classical semantics for DL where each quasi-classical model is a subset of Herbrand base, which is obtained by grounding all concepts and roles in a Herbrand Universe (a set of constants). In this sense, we think that the semantics could be taken as some kind of restricted version of our proposal semantics.

Compared with those preferred semantics based on two-valued interpretations [7,48,49], the QC semantics is based on four-valued semantics. Compared with paraconsistent approaches based on repairing [17,23,8] where a new consistent KB or models of KB are restored from an inconsistent KB by removing some
knowledge causing inconsistency, our approach does not reject any knowledge but tolerate inconsistent knowledge in reasoning. Similarly, those paraconsistent approaches based on argumentation presented by \cite{11,44} introduce some partial orders (argument principles) of all consistent subsets of an inconsistent KB to select some expected consistent subsets for reasoning. Our approach adopts a totally different principle from those approaches. In addition, QCDL, including paradoxical DL, four-valued DL and \(PALC\), is monotonic.

As an important member of the multi-valued DL family, fuzzy description logics \cite{40} can reason with uncertain knowledge in DL. Fuzzy DL admits truth values different from “true” and “false”, each of which is intuitively taken as a certain degree. The big difference between fuzzy logic and multi-valued logic like our work is in the aims.

Additionally, there are some treatments in inconsistent OWL ontologies. Zhou et al \cite{50} present a three-based approach to measuring inconsistent ontologies. Ji et al \cite{19} propose an approach to debug inconsistent ontologies. Du et al \cite{7} develop an approach to answering queries on inconsistent ontologies. Compared with them, our work focus on reasoning on inconsistent ontologies.

6 Conclusions

In this paper, we have implemented PROSE as a plugin-based framework for paraconsistent reasoning for OWL ontologies and, within this framework, then we can select variable classical OWL reasoners to serve for inconsistent ontologies. Moreover, our experiment results show that classical OWL reasoners as the engine of PROSE have different performance in reasoning between consistent ontologies and inconsistent ontologies. In some sense, we provide a handy tool which gives more insight how different DL reasoners functions. Additionally, we have re-implemented ParOWL in our framework PROSE which gives a good proof to show that our framework is feasible in characterizing different paraconsistent approaches in a unified way. In the future work, we are going to optimize QC-transformer so that PROSE can support more general ontologies in an efficient way.

Acknowledgements

We would like to thank Jeff Z. Pan, Juanzi Li, and Kouji Kozaki for their useful comments which helped us to improve the paper. We would also like to thank Zuoquan Lin, Kewen Wang, Guilin Qi, Yue Ma, and Guohui Xiao for discussions and their critical comments on our previous work of quasi-classical description logics and thank Daoxun Fu and Qiong Li for help in experiment. This work is supported by the program of the National Natural Science Foundation of China (NSFC) under 61502336, 61572353, 61373035 and the National High-tech R&D Program of China (863 Program) under 2013AA013204. Xiaowang Zhang is supported by the project of Tianjin Thousand Young Talents Program.
References

33. Prot`eg´e: The prot´eg´e ontology editor and knowledge acquisition system. Website, http://protege.stanford.edu/