Decomposition-Based Optimization for Debugging of Inconsistent OWL DL Ontologies

Jianfeng Du[1,2] and Guilin Qi[3]

[1] Guangdong University of Foreign Studies, Guangzhou 510006, China
jfdu@mail.gdufs.edu.cn
[2] State Key Laboratory of Computer Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China
[3] School of Computer Science and Engineering, Southeast University, Nanjing 211189, China

Abstract. Ontology debugging plays an important role in tackling inconsistency in OWL DL ontologies. It computes all minimal inconsistent sub-ontologies (MISs) of an inconsistent ontology. However, the computation of all MISs is costly. Existing module extraction methods that optimize the debugging of ontology entailments are not sufficient to optimize the computation of all MISs, because a module (i.e. a sub-ontology) that covers all MISs can be too large to be handled by existing OWL DL debugging facilities. In order to generate smaller sub-ontologies, we propose a novel method for computing a set of sub-ontologies from an inconsistent OWL DL ontology so that the computation of all MISs can be separately performed in each resulting sub-ontology and the union of computational results yields exactly the set of all MISs. Experimental results show that this method significantly improves the scalability for computing all MISs of an inconsistent ontology.

1 Introduction

OWL DL[1] is a standard language proposed by the W3C organization for modeling ontologies in the Semantic Web. It is a syntactic variant of Description Logic (DL) $\mathcal{SHOIN}(\mathbf{D})$ [5]. As $\mathcal{SHOIN}(\mathbf{D})$ is a fragment of First-Order Logic (FOL), $\mathcal{SHOIN}(\mathbf{D})$ (i.e. OWL DL) inherits from FOL the property *ex falso sequitur quodlibet* (from falsehood/contradiction follows anything), thus inconsistency in an OWL DL ontology is fatal. As inconsistency can be caused by unsatisfiable concepts, existing work pays much attention to debugging of unsatisfiable concepts in an ontology (e.g. [10,8]), which computes all minimal sub-ontologies in which a given concept is unsatisfiable. However debugging of unsatisfiable concepts does not directly deal with inconsistency. To handle inconsistency directly, Haase *et al.* [4] define the task of inconsistency debugging in an ontology as computing all minimal inconsistent sub-ontologies (MISs) of the ontology. MISs are very useful in tackling inconsistency because they contain erroneous axioms

[1] `http://www.w3.org/TR/owl-semantics/`

in an inconsistent ontology. Inconsistency can be tackled by fixing or removing axioms according to their appearance in MISs. Furthermore, MISs can be used to compute all maximal consistent sub-ontologies by using Reiter's Hitting Set Tree (HST) algorithm [9], some of which can be used as repaired results for an inconsistent ontology.

Though MISs are useful in tackling inconsistency, they are computationally intensive due to the high complexity of consistency checking in OWL DL. To optimize the computation of all MISs, the current approach to optimizing ontology debugging, namely module extraction, can possibly be adapted. Existing module extraction methods [12,1] extract from a given ontology a sub-ontology that covers all minimal subsets of axioms responsible for a given ontology entailment. One may want to adapt existing module extraction approach to finding justifications to optimize the computation of all MISs. However, even if this approach is possible, it may not take effect when the union set of all MISs is very large. Take an ontology $\mathcal{O}$ for example, which consists of the following axioms.

$\mathsf{ChiefActress} \sqsubseteq \mathsf{Actress}$	$\mathsf{Actress} \sqcap \mathsf{Actor} \sqsubseteq \bot$	$\mathsf{ChiefActress}(a_1)$	$\mathsf{Actor}(a_1)$
...	...	$\mathsf{ChiefActress}(a_n)$	$\mathsf{Actor}(a_n)$

It can be seen that the set of MISs of $\mathcal{O}$ is $\{\mathsf{ChiefActress} \sqsubseteq \mathsf{Actress}, \mathsf{Actress} \sqcap \mathsf{Actor} \sqsubseteq \bot, \mathsf{ChiefActress}(a_i), \mathsf{Actor}(a_i) \mid 1 \leq i \leq n\}$. Hence a sub-ontology covering all these MISs is exactly $\mathcal{O}$. This example shows that module extraction is insufficient in optimizing the computation of all MISs. We therefore need a new method that can generate smaller sub-ontologies from which MISs are computed.

In order to make the computation of all MISs easier, we propose a new method for computing a set of sub-ontologies from which MISs can be separately computed (see Section 3). The method is based on a similar process as the one given in [1], i.e. compilation from OWL DL to Propositional Logic, but extends the compilation to yield a *labeled propositional program* in which every ground clause is associated with a label. The label stands for the axiom from which the corresponding ground clause is compiled. After compiling the labeled propositional program, our proposed method decomposes it into disjoint components and constructs sub-ontologies based on the labels in the resulting components. We prove that MISs can be separately computed from each resulting sub-ontology and the union of computational results is exactly the set of all MISs. For the aforementioned example, our proposed method computes from $\mathcal{O}$ the set of sub-ontologies $\{\mathsf{ChiefActress} \sqsubseteq \mathsf{Actress}, \mathsf{Actress} \sqcap \mathsf{Actor} \sqsubseteq \bot, \mathsf{ChiefActress}(a_i), \mathsf{Actor}(a_i) \mid 1 \leq i \leq n\}$, each of which is exactly a MIS of $\mathcal{O}$.

We implement the proposed method as a prototype system and test on two groups of inconsistent ontologies (see Section 4). The ontologies in the first group are modified from existing incoherent ontologies. For two out of the three ontologies in this group, the computation of all MISs cannot be done in four hours by the well-known Pellet system [11] (simply Pellet), but can be computed in a few minutes by our system. The ontologies in the second group are modified from the University Benchmark (UOBM-Lite) [7] by inserting conflicts. Experimental results on this group show that our system significantly outperforms Pellet and scales well up to hundreds of conflicts when the number of conflicts increases.

2 Preliminaries

OWL DL and Debugging. OWL DL corresponds to DL $\mathcal{SHOIN}$ [5].[2] We assume that the reader is familiar with OWL DL and thus we do not describe it in detail, but recall that an OWL DL ontology $\mathcal{O}$ consists of a set of axioms. These axioms include *terminological axioms* (i.e. *concept inclusion axioms* $C \sqsubseteq D$, *transitivity axioms* $\mathsf{Trans}(R)$ and *role inclusion axioms* $R \sqsubseteq S$) and *assertional axioms* (i.e. *concept assertions* $C(a)$, *role assertions* $R(a, b)$, *equality assertions* $a \approx b$ and *inequality assertions* $a \not\approx b$), where C and D are OWL DL concepts, R and S roles, and a and b individuals.

We briefly introduce the direct model-theoretic semantics for an OWL DL ontology $\mathcal{O}$. An *interpretation* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a *domain* $\Delta^{\mathcal{I}}$ and a function $\cdot^{\mathcal{I}}$ that maps every atomic concept A to a set $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$, every atomic role R to a binary relation $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and every individual a to $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. $\mathcal{I}$ is called a *model* of $\mathcal{O}$ if every axiom in $\mathcal{O}$ is satisfied by $\mathcal{I}$. $\mathcal{O}$ is said to be *consistent* if it has a model. A concept C is said to be *satisfiable* in $\mathcal{O}$ if there exists a model $\mathcal{I}$ of $\mathcal{O}$ such that $C^{\mathcal{I}} \neq \emptyset$. $\mathcal{O}$ is said to be *coherent* if every atomic concept in it is satisfiable.

A *sub-ontology* $\mathcal{O}'$ of $\mathcal{O}$ is a subset of axioms in $\mathcal{O}$. A *minimal inconsistent sub-ontology* (MIS) $\mathcal{O}'$ of $\mathcal{O}$ is an inconsistent sub-ontology of $\mathcal{O}$ that has not any proper subset $\mathcal{O}''$ such that $\mathcal{O}''$ is also inconsistent. The task of debugging of inconsistency in an inconsistent ontology $\mathcal{O}$ is to compute all MISs of $\mathcal{O}$ [4].

First-order Logic and Axiomatizing Equality. *Terms* are variables, constants or functional terms of the form $f(t_1, \ldots, t_n)$, where f is a function symbol of arity n and $t_1, ..., t_n$ are terms.[3] We only consider unary function symbols because only unary function symbols occur in first-order logic programs that are translated from OWL DL ontologies. *Atoms* are of the form $T(t_1, \ldots, t_n)$ where T is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms. A *literal* is a positive or negative atom and a *clause* is a disjunction of literals. Terms, atoms and clauses that do not contain variables are called *ground*.

A *first-order logic program* (FOL program) is a set of clauses in which all variables are universally quantified. For a clause $cl = \neg A_1 \vee \ldots \vee \neg A_n \vee B_1 \vee \ldots \vee B_m$, the set of atoms $\{A_1, \ldots, A_n\}$ is denoted by cl^-, whereas the set of atoms $\{B_1, \ldots, B_m\}$ is denoted by cl^+. By $|S|$ we denote the cardinality of a set S. A clause cl is called a *fact* if $|cl^-| = 0$.

A *propositional program* Π is a FOL program consisting of only ground clauses. The set of ground atoms occurring in Π is denoted by $\mathsf{atoms}(\Pi)$.

For a FOL program P, the set of ground terms (resp. ground atoms) defined from the first-order signature of P is called the *Herbrand universe* (resp. *Herbrand base*) of P, denoted by $\mathsf{HU}(P)$ (resp. $\mathsf{HB}(P)$). The set of ground clauses

[2] OWL DL also involves datatypes which can be handled by our method with some extensions. But we do not complicate our presentation by considering them here.

[3] Throughout this paper, we use (possibly with subscripts) x, y, z for variables, a, b, c for constants, and s, t for terms.

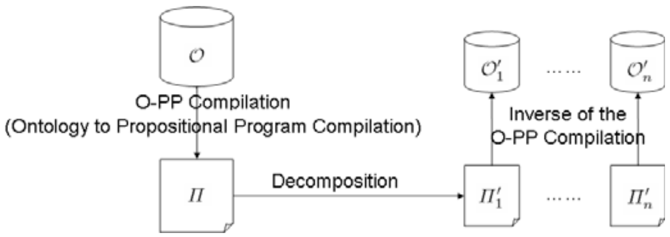
obtained by replacing all variables occurring in each clause in P with ground terms from $\mathsf{HU}(P)$ is called the *primary grounding* of P, denoted by $\mathcal{G}(P)$. An *interpretation* M of P is a set of ground atoms in $\mathsf{HB}(P)$; it is a *model* of P if for any ground clause $cl \in \mathcal{G}(P)$ such that $cl^- \subseteq M$, $cl^+ \cap M \neq \emptyset$; it is a *minimal model* of P if there is no model M' of P such that $M' \subset M$. P is said to be *satisfiable* if P has a model.

The FOL program P translated from an OWL DL ontology may contain the *equality predicate* $\approx$, which is interpreted as a *congruence relation* and different from ordinary predicates. This difference is not captured by the above first-order semantics. However, the equality predicate $\approx$ can be explicitly axiomatized via a well-known transformation from [3]. Let $\mathcal{E}(P)$ denote the FOL program consisting of the following clauses: (1) $t \approx t$, for each ground term $t \in \mathsf{HU}(P)$; (2) $\neg(x \approx y) \vee y \approx x$; (3) $\neg(x \approx y) \vee \neg(y \approx z) \vee x \approx z$; (4) $\neg(x \approx y) \vee f(x) \approx f(y)$, for each function symbol f occurring in P; (5) $\neg T(x_1, \ldots, x_i, \ldots, x_n) \vee \neg(x_i \approx y_i) \vee T(x_1, \ldots, y_i, \ldots, x_n)$, for each predicate symbol T other than $\approx$ occurring in P and each position i. Appending $\mathcal{E}(P)$ to P allows to treat $\approx$ as an ordinary predicate, i.e., M is a model of P that interprets $\approx$ as a congruence relation, iff for any ground clause $cl \in \mathcal{G}(P \cup \mathcal{E}(P))$ such that $cl^- \subseteq M$, $cl^+ \cap M \neq \emptyset$.

3 Finding Sub-ontologies to Optimize MIS Computation

Given an OWL DL ontology $\mathcal{O}$, we intend to compute a set of sub-ontologies $\{\mathcal{O}'_i\}_{1 \leq i \leq n}$ of $\mathcal{O}$ such that the union of sets of MISs of these sub-ontologies yields the set of all MISs of $\mathcal{O}$, i.e., $\bigcup_{i=1}^{n} \mathcal{M}_i$ is the set of MISs of $\mathcal{O}$, where $\mathcal{M}_i$ is the set of MISs of $\mathcal{O}'_i$ for $1 \leq i \leq n$. The direct computation of these sub-ontologies from $\mathcal{O}$ is hard as we need to consider the internal of ontology reasoning algorithms. Hence we follow the idea given in our previous work [1] and consider compiling $\mathcal{O}$ into a propositional program. It is easier to analyze propositional reasoning algorithms because a model of a propositional program is a subset of ground atoms occurring in it. That is, it is easier to analyze the compiled propositional program and use it to compute sub-ontologies of $\mathcal{O}$ that preserve MISs.

The idea exploited in the current work is depicted in the following figure.



First, the given ontology $\mathcal{O}$ is compiled to a propositional program Π using a so-called O-PP (Ontology to Propositional Program) compilation process. Then, Π is decomposed into a set of disjoint components $\Pi'_1, \ldots, \Pi'_n$. Finally, a set of sub-ontologies $\{\mathcal{O}'_i\}_{1 \leq i \leq n}$ of $\mathcal{O}$ is extracted from $\{\Pi'_i\}_{1 \leq i \leq n}$ respectively using the inverse process of the O-PP compilation. Suppose $\mathcal{M}_i$ is the set of MISs computed from $\mathcal{O}'_i$ by applying existing OWL DL debugging facilities ($1 \leq i \leq n$). Then $\bigcup_{i=1}^{n} \mathcal{M}_i$ is the set of MISs of $\mathcal{O}$.

There are two key points in the proposed method. The first one is how to define the O-PP compilation process and its inverse process. The second one is how to define the decomposition process. We will describe our solutions to these key points in the following subsections.

3.1 The O-PP Compilation Process and Its Inverse Process

As $\mathcal{SHOIN}$ is a fragment of FOL, an OWL DL (i.e. $\mathcal{SHOIN}$) ontology can be translated to a FOL program using standard translation methods. Then the FOL program can be further transformed to a propositional program using standard grounding methods. Thus an OWL DL ontology can be compiled to a propositional program using standard methods. However, the standard methods are insufficient to facilitate the inverse process because any manipulation of the final propositional program is hardly mapped to the original ontology.

To make the inverse process easy, we develop a method for translating an OWL DL ontology $\mathcal{O}$ to a *labeled FOL program* $\mathcal{L}(\mathcal{O})$, which consists of *labeled clauses* of the form (cl, ax), where ax is the axiom from which the clause cl is translated; ax is the empty label $\diamond$ if cl is a clause used to axiomatize equality. The labeled FOL program $\mathcal{L}(\mathcal{O})$ is defined through a translation from an axiom ax to a set $\mathcal{L}(ax)$ of labeled clauses. $\mathcal{L}(ax)$ is defined recursively in the following frame, where A and B are atomic concepts or $\top$, a and b are individuals, C, C_1, C_2 and D are $\mathcal{SHOIN}$ concepts, R and S are roles, and Q_X is X if X is an atomic concept, or a new globally unique concept name for X otherwise; $\mathsf{NNF}(C)$ denotes the result of translating C to the negation normal form; $\mathsf{ar}(R, s, t)$ denotes $S(t, s)$ if $R = S^-$ for some role name S, or $R(s, t)$ otherwise.

if $ax \equiv R \sqsubseteq S$, $\mathcal{L}(ax) = \{(\neg\mathsf{ar}(R, x, y) \vee \mathsf{ar}(S, x, y), ax)\}$;
if $ax \equiv \mathsf{Trans}(R)$, $\mathcal{L}(ax) = \{(\neg\mathsf{ar}(R, x, y) \vee \neg\mathsf{ar}(R, y, z) \vee \mathsf{ar}(R, x, z), ax)\}$;
if $ax \equiv C \sqsubseteq D$, $\mathcal{L}(ax) = \{(cl, ax) \mid cl \in \mathcal{F}(\top \sqsubseteq \mathsf{NNF}(\neg C \sqcup D))\}$;
if $ax \equiv C(a)$, $\mathcal{L}(ax) = \{(Q_D(a), ax)\} \cup \{ (cl, ax) \mid cl \in \mathcal{F}(Q_D \sqsubseteq D)\}$ for $D = \mathsf{NNF}(C)$;
if $ax \equiv R(a, b)$, $\mathcal{L}(ax) = \{(\mathsf{ar}(R, a, b), ax)\}$;
if $ax \equiv a \approx b$, $\mathcal{L}(ax) = \{(a \approx b, ax)\}$; if $ax \equiv a \not\approx b$, $\mathcal{L}(ax) = \{(\neg(a \approx b), ax)\}$;
$\mathcal{F}(A \sqsubseteq B) = \{\neg A(x) \vee B(x)\}$; $\mathcal{F}(A \sqsubseteq \neg B) = \{\neg A(x) \vee \neg B(x)\}$;
$\mathcal{F}(A \sqsubseteq \{a\}) = \{\neg A(x) \vee x \approx a\}$; $\mathcal{F}(A \sqsubseteq \neg\{a\}) = \{\neg A(a)\}$;
$\mathcal{F}(A \sqsubseteq C_1 \sqcap C_2) = \mathcal{F}(A \sqsubseteq C_1) \cup \mathcal{F}(A \sqsubseteq C_2)$;
$\mathcal{F}(A \sqsubseteq C_1 \sqcup C_2) = \{\neg A(x) \vee Q_{C_1}(x) \vee Q_{C_2}(x)\} \cup \mathcal{F}\,(Q_{C_1} \sqsubseteq C_1) \cup \mathcal{F}(Q_{C_2} \sqsubseteq C_2)$;
$\mathcal{F}(A \sqsubseteq \exists R.C) = \{\neg A(x) \vee \mathsf{ar}(R, x, f(x)), \neg A(x) \vee Q_C(f(x))\} \cup \mathcal{F}\,(Q_C \sqsubseteq C)$;
$\mathcal{F}(A \sqsubseteq \forall R.C) = \{\neg A(x) \vee \neg\mathsf{ar}(R, x, y) \vee Q_C(y)\} \cup \mathcal{F}\,(Q_C \sqsubseteq C)$;
$\mathcal{F}(A \sqsubseteq \geq_n R) = \{\neg A(x) \vee \mathsf{ar}(R, x, f_i(x)) \mid 1 \leq i \leq n\} \cup$
$\{\neg(f_j(x) \approx f_k(x)) \mid 1 \leq j < k \leq n\}$;
$\mathcal{F}(A \sqsubseteq \leq_n R) = \{\neg A(x) \vee \bigvee_{i=1}^{n+1} \neg\mathsf{ar}(R, x, y_i) \vee \bigvee_{j=1}^{n} \bigvee_{k=j+1}^{n+1} y_j \approx y_k\}$.

Let $\mathcal{L}^*(\mathcal{O})$ denote $\bigcup_{ax \in \mathcal{O}} \mathcal{L}(ax)$. Then $\mathcal{L}(\mathcal{O})$ is defined as $\mathcal{L}^*(\mathcal{O}) \cup \{(cl, \diamond) \mid cl \in \mathcal{E}(\{cl \mid (cl, ax) \in \mathcal{L}^*(\mathcal{O})\})\}$ if some equational atom $s \approx t$ occurs positively in $\mathcal{L}^*(\mathcal{O})$, or defined as $\mathcal{L}^*(\mathcal{O})$ otherwise. Recall that $\mathcal{E}(P)$ for a FOL program P is used to axiomatize equality in P, as described in Section 2. Such translation from

$\mathcal{O}$ to $\mathcal{L}(\mathcal{O})$ is based on the well-known processes of *structural transformation* [6], clausification and equality axiomatization.

Example 1. Our running example is adapted from the example given in Section 1 by adding ax_2, ax_3 and ax_5 and fixing $n = 2$. Let $\mathcal{O} = \{ax_1, ..., ax_9\}$.

$ax_1 \equiv \mathsf{ChiefActress} \sqsubseteq \mathsf{Actress}$ $\quad ax_2 \equiv \mathsf{ChiefActress} \sqcap \mathsf{Man} \sqsubseteq \bot$

$ax_3 \equiv \mathsf{Actress} \sqsubseteq \mathsf{Person}$ $\quad ax_4 \equiv \mathsf{Actress} \sqcap \mathsf{Actor} \sqsubseteq \bot$

$ax_5 \equiv \mathsf{Person} \sqsubseteq \mathsf{Man} \sqcup \mathsf{Woman}$ $\quad ax_6 \equiv \mathsf{Actor}(a_1)$ $\quad ax_7 \equiv \mathsf{Actor}(a_2)$

$ax_8 \equiv \mathsf{ChiefActress}(a_1)$ $\quad ax_9 \equiv \mathsf{ChiefActress}(a_2)$

Then $\mathcal{L}(\mathcal{O})$ consists of the following labeled clauses.

$lcl_1 \equiv (\neg\mathsf{ChiefActress}(x) \vee \mathsf{Actress}(x), ax_1)$ $\quad lcl_2 \equiv (\neg\mathsf{ChiefActress}(x) \vee \neg\mathsf{Man}(x), ax_2)$

$lcl_3 \equiv (\neg\mathsf{Actress}(x) \vee \mathsf{Person}(x), ax_3)$ $\quad lcl_4 \equiv (\neg\mathsf{Actress}(x) \vee \neg\mathsf{Actor}(x), ax_4)$

$lcl_5 \equiv (\neg\mathsf{Person}(x) \vee \mathsf{Man}(x) \vee \mathsf{Woman}(x), ax_5)$ $\quad lcl_6 \equiv (\mathsf{Actor}(a_1), ax_6)$

$lcl_7 \equiv (\mathsf{Actor}(a_2), ax_7)$ $\quad lcl_8 \equiv (\mathsf{ChiefActress}(a_1), ax_8)$ $\quad lcl_9 \equiv (\mathsf{ChiefActress}(a_2), ax_9)$

For a labeled FOL program P, by $\mathsf{Cl}(P)$ and $\mathsf{Ax}(P)$ we respectively denote the clause part and axiom part of P, i.e., $\mathsf{Cl}(P) = \{cl \mid (cl, ax) \in P\}$ and $\mathsf{Ax}(P) = \{ax \in \mathcal{O} \mid (cl, ax) \in P\}$. For a sub-ontology $\mathcal{O}'$ of $\mathcal{O}$, by $\rho(\mathcal{L}(\mathcal{O}), \mathcal{O}')$ we denote $\{(cl, ax) \in \mathcal{L}(\mathcal{O}) \mid ax \in \{\diamond\} \cup \mathcal{O}'\}$, which is the subset of $\mathcal{L}(\mathcal{O})$ consisting of clauses translated from $\mathcal{O}'$ and clauses used to axiomatize equality. The following lemma shows a relationship between $\mathcal{O}'$ and $\rho(\mathcal{L}(\mathcal{O}), \mathcal{O}')$, which follows from the first-order semantics of OWL DL [6] and the fact that structural transformation and clausification preserve satisfiability.

Lemma 1. *For any subset $\mathcal{O}'$ of $\mathcal{O}$, $\mathcal{O}'$ is consistent iff* $\mathsf{Cl}(\rho(\mathcal{L}(\mathcal{O}), \mathcal{O}'))$ *is satisfiable.* □

To complete the O-PP compilation process, we adapt the notion of bottom-up grounding given in [1] to a labeled FOL program P and define the *bottom-up grounding* of P, denoted by $\mathcal{G}_{bu}(P)$, as the least fixpoint of $\Pi^{(n)}$ such that $\Pi^{(0)} = \emptyset$ and for $n \geq 1$, $\Pi^{(n)} = \{(cl\,\sigma, ax) \mid (cl, ax) \in P,\ \sigma$ is a ground substitution such that $cl^-\sigma \subseteq \mathsf{atoms}(\mathsf{Cl}(\Pi^{(n-1)}))$ and $cl^+\sigma \subseteq \mathsf{HB}(\mathsf{Cl}(P))\}$. Note that a propositional program is a special FOL program, so we also use $\mathsf{Cl}(\Pi)$ and $\mathsf{Ax}(\Pi)$ to respectively denote the clause part and axiom part of a labeled propositional program Π. This notion of bottom-up grounding only differs from the original one given in [1] on considering labels. The following lemma shows a relation between P and $\mathcal{G}_{bu}(P)$, proved analogously as in Lemma 3 of [1].

Lemma 2. *For a labeled FOL program P where the equality predicate has been axiomatized,* $\mathsf{Cl}(\mathcal{G}_{bu}(P))$ *has the same set of minimal models as* $\mathsf{Cl}(P)$ *has.* □

Note that $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ can be infinite due to function symbols in $\mathcal{L}(\mathcal{O})$ introduced by *skolemization* in the course of clausification. We will present an approximate grounding method to tackle this problem in Subsection 3.3. But here we first consider an O-PP compilation process which compiles $\mathcal{O}$ to $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$.

For a subset Π of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$, we can naturally define the inverse process of the O-PP compilation (on Π) as computing $\mathsf{Ax}(\Pi)$. Clearly this inverse process is easy; this explains why we introduce labels to the O-PP compilation process. The following lemma shows a relation between $\mathsf{Cl}(\Pi)$ and $\mathsf{Ax}(\Pi)$.

Lemma 3. *For any subset Π of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$, $\mathsf{Ax}(\Pi)$ is inconsistent if $\mathsf{Cl}(\Pi)$ is unsatisfiable.*

Proof. Since each clause in $\mathsf{Cl}(\Pi)$ is instantiated from $\mathsf{Cl}(\rho(\mathcal{L}(\mathcal{O}), \mathsf{Ax}(\Pi)))$, $\mathsf{Cl}(\Pi)$ is a subset of $\mathcal{G}(\mathsf{Cl}(\rho(\mathcal{L}(\mathcal{O}), \mathsf{Ax}(\Pi))))$, i.e. the primary grounding of $\mathsf{Cl}(\rho(\mathcal{L}(\mathcal{O}), \mathsf{Ax}(\Pi)))$. When $\mathsf{Cl}(\Pi)$ is unsatisfiable, $\mathcal{G}(\mathsf{Cl}(\rho(\mathcal{L}(\mathcal{O}), \mathsf{Ax}(\Pi))))$ is also unsatisfiable and so is $\mathsf{Cl}(\rho(\mathcal{L}(\mathcal{O}), \mathsf{Ax}(\Pi)))$. By Lemma 1, $\mathsf{Ax}(\Pi)$ is inconsistent. □

Lemma 3 implies that some MISs of $\mathcal{O}$ can be computed from the axiom part of a subset Π of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ such that $\mathsf{Cl}(\Pi)$ is unsatisfiable. Hence we can first compute all small subsets of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ whose clause parts are unsatisfiable, then compute MISs of $\mathcal{O}$ from them. The computation of such subsets of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ can be realized by decomposing $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$, as shown in the next subsection.

3.2 The Decomposition Process

The basic idea of decomposing $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ is to first remove a *satisfiable core* from $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$, then compute *maximal connected components* of the remaining subset, where a subset Π_{sc} of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ is called a *satisfiable core* of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ if for any subset $\mathcal{O}'$ of $\mathcal{O}$, $\mathsf{Cl}(\mathcal{G}_{bu}(\rho(\mathcal{L}(\mathcal{O}), \mathcal{O}')) \cap \Pi_{sc})$ has a model M_0 such that for all models M of $\mathsf{Cl}(\mathcal{G}_{bu}(\rho(\mathcal{L}(\mathcal{O}), \mathcal{O}')) \setminus \Pi_{sc})$, $M_0 \cup M$ is a model of $\mathsf{Cl}(\mathcal{G}_{bu}(\rho(\mathcal{L}(\mathcal{O}), \mathcal{O}')))$. This idea is similar with the one given in [1], which is used to compute a sub-ontology that preserves all minimal subsets of axioms responsible for a given ontology entailment. Here the notion of maximal connected component is adapted from the homonymous notion given in [1]. That is, a *connected component* of a labeled propositional program Π is a subset of Π such that any two clauses in its clause part are connected through common ground atoms, while a *maximal connected component* of Π is a connected component of Π which cannot be a proper subset of any connected component of Π. The following theorem shows the correctness of a method exploiting the above idea.

Theorem 1. *Let Π_{sc1} be a subset of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ such that for all $(cl, ax) \in \Pi_{sc1}$, $cl^+ \not\subseteq \mathsf{atoms}(\mathsf{Cl}(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \setminus \Pi_{sc1}))$. Let Π_{sc2} be a subset of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \setminus \Pi_{sc1}$ such that for all $(cl, ax) \in \Pi_{sc2}$, $cl^- \not\subseteq \bigcup_{(cl,ax)\in\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))\setminus(\Pi_{sc1}\cup\Pi_{sc2})} cl^+$. Let $\{\Pi_i\}_{1\leq i\leq m}$ be the set of maximal connected components of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \setminus (\Pi_{sc1} \cup \Pi_{sc2})$. Let $\mathcal{M}_i$ be the set of MISs of $\mathsf{Ax}(\Pi_i)$ for $1 \leq i \leq m$. Then $\bigcup_{i=1}^m \mathcal{M}_i$ is the set of all MISs of $\mathcal{O}$.*

Proof. (i) It is clear that a MIS of $\mathsf{Ax}(\Pi_i)$ for any $1 \leq i \leq m$ is a MIS of $\mathcal{O}$. (ii) Consider a MIS $\mathcal{O}'$ of $\mathcal{O}$. By Lemma 1, $\mathsf{Cl}(\rho(\mathcal{L}(\mathcal{O}), \mathcal{O}'))$ is unsatisfiable. Let $\Pi' = \mathcal{G}_{bu}(\rho(\mathcal{L}(\mathcal{O}), \mathcal{O}'))$. By Lemma 2, $\mathsf{Cl}(\Pi')$ is also unsatisfiable. Let $\Pi'_i = \Pi' \cap \Pi_i$ for all $1 \leq i \leq m$. Suppose $\mathsf{Cl}(\Pi'_i)$ is satisfiable for all $1 \leq i \leq m$. Let M_i be a model of $\mathsf{Cl}(\Pi'_i)$ for $1 \leq i \leq m$ and $M_0 = \mathsf{atoms}(\mathsf{Cl}(\Pi')) \cap \bigcup_{(cl,ax)\in\Pi_{sc1}} cl^+ \setminus \mathsf{atoms}(\mathsf{Cl}(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))\setminus\Pi_{sc1}))$. Let $\Pi'_0 = \Pi' \setminus \bigcup_{i=1}^m \Pi'_i$. Then $\Pi'_0 \subseteq \Pi_{sc1} \cup \Pi_{sc2}$. For all $cl \in \mathsf{Cl}(\Pi_{sc1} \cap \Pi'_0)$, since $cl^+ \not\subseteq \mathsf{atoms}(\mathsf{Cl}(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))\setminus\Pi_{sc1}))$, we have $\emptyset \subset cl^+ \setminus \mathsf{atoms}(\mathsf{Cl}(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))\setminus\Pi_{sc1})) \subseteq M_0$, thus $M_0 \cap cl^+ \neq \emptyset$. For all $cl \in \mathsf{Cl}(\Pi_{sc2} \cap \Pi'_0)$, since $M_0 \subseteq \bigcup_{(cl,ax)\in\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))\setminus(\Pi_{sc1}\cup\Pi_{sc2})} cl^+$, we have $cl^- \not\subseteq M_0$. It follows that

Algorithm 1. `Decompose`(Π)

```
1. $\Pi_{sc1} := \Pi$; $\Pi'_{sc1} := \emptyset$;
2. while $\Pi'_{sc1} \neq \Pi_{sc1}$ do
3.     $\Pi'_{sc1} := \Pi_{sc1}$;
4.     for each $(cl, ax) \in \Pi_{sc1}$ s.t. $cl^+ \subseteq \mathsf{atoms}(\mathsf{Cl}(\Pi \setminus \Pi_{sc1}))$ do
5.         $\Pi_{sc1} := \Pi_{sc1} \setminus \{(cl, ax)\}$;
6. $\Pi_{sc2} := \emptyset$; $\Pi'_{sc2} := \Pi$;
7. while $\Pi'_{sc2} \neq \Pi_{sc2}$ do
8.     $\Pi'_{sc2} := \Pi_{sc2}$;
9.     for each $(cl, ax) \in \Pi \setminus (\Pi_{sc1} \cup \Pi_{sc2})$ s.t. $cl^- \not\subseteq \bigcup_{(cl,ax)\in\Pi\setminus(\Pi_{sc1}\cup\Pi_{sc2})} cl^+$ do
10.        $\Pi_{sc2} := \Pi_{sc2} \cup \{(cl, ax)\}$;
11. return the set of maximal connected components of $\Pi \setminus (\Pi_{sc1} \cup \Pi_{sc2})$;
```

M_0 is a model of $\mathsf{Cl}(\Pi'_0)$. Since $\bigcup_{i=1}^m M_i \subseteq \bigcup_{(cl,ax)\in\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))\setminus(\Pi_{sc1}\cup\Pi_{sc2})} cl^+$ and for all $0 \leq i \leq m$ and $1 \leq j \leq m$, $M_i \cap \mathsf{atoms}(\Pi'_j) = \emptyset$ if $i \neq j$, $\bigcup_{i=0}^m M_i$ is a model of $\bigcup_{i=0}^m \mathsf{Cl}(\Pi'_i) = \mathsf{Cl}(\Pi')$, contradicting that $\mathsf{Cl}(\Pi')$ is unsatisfiable. Hence $\mathsf{Cl}(\Pi'_k)$ is unsatisfiable for some $1 \leq k \leq m$. Since Π'_k is a subset of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$, by Lemma 3, $\mathsf{Ax}(\Pi'_k)$ is inconsistent. Since $\mathsf{Ax}(\Pi'_k) \subseteq \mathsf{Ax}(\Pi') = \mathcal{O}'$ and $\mathcal{O}'$ is a MIS of $\mathcal{O}$, we have $\mathcal{O}' = \mathsf{Ax}(\Pi'_k) \subseteq \mathsf{Ax}(\Pi_k)$, thus $\mathcal{O}' \in \mathcal{M}_k$. □

Algorithm 1 shows how to decompose $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ (where the input Π is set as $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$) using the method given in Theorem 1. The set of maximal connected components of $\Pi\setminus(\Pi_{sc1}\cup\Pi_{sc2})$ returned by the algorithm can be efficiently computed by the well-known union-find algorithm (cf. `http://en.wikipedia.org/wiki/Union-find_algorithm`). It can be seen that Π_{sc1} and Π_{sc2} computed in `Decompose`($\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$) satisfy the conditions of the homonymous subsets of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ given in Theorem 1, thus we have the following theorem.

Theorem 2. *Let $\{\Pi_i\}_{1\leq i\leq m}$ be returned by `Decompose`($\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$). Let $\mathcal{M}_i$ be the set of MISs of $\mathsf{Ax}(\Pi_i)$. Then $\bigcup_{i=1}^m \mathcal{M}_i$ is the set of all MISs of $\mathcal{O}$.* □

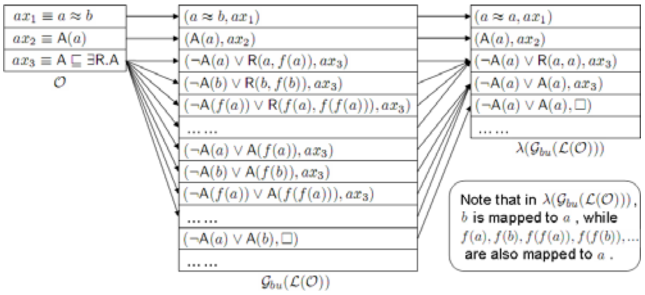
Example 2. Continue Example 1. By applying a fixpoint-evaluation manner, we can compute that $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) = \{lcl'_1, \ldots, lcl'_{14}\}$. (Note that the labeled clauses are ordered to save space.)

$lcl'_1 \equiv (\neg\mathsf{ChiefActress}(a_1) \vee \mathsf{Actress}(a_1), ax_1)$ $lcl'_2 \equiv (\neg\mathsf{ChiefActress}(a_1) \vee \neg\mathsf{Man}(a_1), ax_2)$
$lcl'_3 \equiv (\neg\mathsf{Actress}(a_1) \vee \mathsf{Person}(a_1), ax_3)$ $lcl'_4 \equiv (\neg\mathsf{Actress}(a_1) \vee \neg\mathsf{Actor}(a_1), ax_4)$
$lcl'_5 \equiv (\neg\mathsf{Person}(a_1) \vee \mathsf{Man}(a_1) \vee \mathsf{Woman}(a_1), ax_5)$ $lcl'_6 \equiv (\mathsf{Actor}(a_1), ax_6)$
$lcl'_7 \equiv (\neg\mathsf{ChiefActress}(a_2) \vee \mathsf{Actress}(a_2), ax_1)$ $lcl'_8 \equiv (\neg\mathsf{ChiefActress}(a_2) \vee \neg\mathsf{Man}(a_2), ax_2)$
$lcl'_9 \equiv (\neg\mathsf{Actress}(a_2) \vee \mathsf{Person}(a_2), ax_3)$ $lcl'_{10} \equiv (\neg\mathsf{Actress}(a_2) \vee \neg\mathsf{Actor}(a_2), ax_4)$
$lcl'_{11} \equiv (\neg\mathsf{Person}(a_2) \vee \mathsf{Man}(a_2) \vee \mathsf{Woman}(a_2), ax_5)$ $lcl'_{12} \equiv (\mathsf{Actor}(a_2), ax_7)$
$lcl'_{13} \equiv (\mathsf{ChiefActress}(a_1), ax_8)$ $lcl'_{14} \equiv (\mathsf{ChiefActress}(a_2), ax_9)$

By applying the algorithm `Decompose`($\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$), we can compute in turn that $\Pi_{sc1} = \{lcl'_3, lcl'_5, lcl'_9, lcl'_{11}\}$ and $\Pi_{sc2} = \{lcl'_2, lcl'_8\}$, and compute the two maximal connected components of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \setminus (\Pi_{sc1} \cup \Pi_{sc2})$ as $\Pi_1 = \{lcl'_1, lcl'_4, lcl'_6, lcl'_{13}\}$ and $\Pi_2 = \{lcl'_7, lcl'_{10}, lcl'_{12}, lcl'_{14}\}$. It is not hard to see that the set $\mathcal{M}_1$ of MISs of $\mathsf{Ax}(\Pi_1)$ is $\{\{ax_1, ax_4, ax_6, ax_8\}\}$, while the set $\mathcal{M}_2$ of MISs of $\mathsf{Ax}(\Pi_2)$ is $\{\{ax_1, ax_4, ax_7, ax_9\}\}$. By Theorem 2, the set of all MISs of $\mathcal{O}$ is $\mathcal{M}_1 \cup \mathcal{M}_2 = \{\{ax_1, ax_4, ax_6, ax_8\}, \{ax_1, ax_4, ax_7, ax_9\}\}$.

3.3 Using Approximate Compilation

The remaining problem of our proposed method is on the possible infiniteness of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$. We address this problem by using the same method given in [1]. That is, we introduce the notion of *convergent mapping function* for a labeled propositional program Π and compute a superset of $\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$ for λ a convergent mapping function for $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$. A convergent mapping function λ for a labeled propositional program Π is defined as a mapping function from ground terms occurring in Π to constants occurring in Π, such that (i) for every functional term $f_1(...f_n(a))$ (where a is a constant) occurring in Π, $\lambda(f_1(...f_n(a))) = \lambda(a)$, and (ii) for every equational atom $s \approx t$ occurring in Π, $\lambda(s) = \lambda(t)$. The mapping function λ is further extended to a clause cl (resp. a labeled propositional program Π) by defining $\lambda(cl)$ (resp. $\lambda(\Pi)$) as the result obtained from cl (resp. Π) by replacing every ground term t occurring in it with $\lambda(t)$. The idea for introducing the convergent mapping function is illustrated the following figure.


$ax_1 \equiv a \approx b$
$ax_2 \equiv \mathsf{A}(a)$
$ax_3 \equiv \mathsf{A} \sqsubseteq \exists \mathsf{R.A}$
$\mathcal{O}$
$(a \approx b, ax_1)$
$(\mathsf{A}(a), ax_2)$
$(\neg\mathsf{A}(a) \vee \mathsf{R}(a, f(a)), ax_3)$
$(\neg\mathsf{A}(b) \vee \mathsf{R}(b, f(b)), ax_3)$
$(\neg\mathsf{A}(f(a)) \vee \mathsf{R}(f(a), f(f(a))), ax_3)$
……
$(\neg\mathsf{A}(a) \vee \mathsf{A}(f(a)), ax_3)$
$(\neg\mathsf{A}(b) \vee \mathsf{A}(f(b)), ax_3)$
$(\neg\mathsf{A}(f(a)) \vee \mathsf{A}(f(f(a))), ax_3)$
……
$(\neg\mathsf{A}(a) \vee \mathsf{A}(b), \square)$
……
$\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$
$(a \approx a, ax_1)$
$(\mathsf{A}(a), ax_2)$
$(\neg\mathsf{A}(a) \vee \mathsf{R}(a, a), ax_3)$
$(\neg\mathsf{A}(a) \vee \mathsf{A}(a), ax_3)$
$(\neg\mathsf{A}(a) \vee \mathsf{A}(a), \square)$
……
$\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$
Note that in $\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$, b is mapped to a, while $f(a), f(b), f(f(a)), f(f(b)), ...$ are also mapped to a.

As shown in the above figure (where arrows denote the sources of labeled clauses), $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ is infinite even when $\mathcal{O}$ consists of only three axioms. But after we map functional terms occurring in $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$ to constants and map a constant to another one if they occur in the same equational atom, we can obtain a small labeled propositional program $\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$ for λ a convergent mapping function for $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$. In fact, $\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$ has polynomial number of labeled clauses wrt the size of $\mathcal{O}$ because structural transformation guarantees that $\mathcal{L}(\mathcal{O})$ has polynomial number of labeled clauses, and there are at most two distinct constants occurring in every labeled clause in $\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$.

The following theorem shows that a set of sub-ontologies, from which all MISs of $\mathcal{O}$ can be computed, are also computable from a superset of $\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$.

Theorem 3. *Let λ be a convergent mapping function for $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$. Let Π be a superset of $\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$. Let $\{\Pi\}_{1 \le i \le m}$ be returned by* `Decompose(`Π`)`*. Let $\mathcal{M}_i$ be the set of MISs of $\mathsf{Ax}(\Pi_i)$ for $1 \le i \le m$. Then $\bigcup_{i=1}^{m} \mathcal{M}_i$ is the set of all MISs of $\mathcal{O}$.*

Proof. Let Π_{sc1} and Π_{sc2} be the homonymous sets of labeled clauses computed in `Decompose`(Π). Let $\Pi'_{sc1} = \{(cl, ax) \in \mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \mid (\lambda(cl), ax) \in \Pi_{sc1}\}$ and $\Pi'_{sc2} = \{(cl, ax) \in \mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \mid (\lambda(cl), ax) \in \Pi_{sc2}\}$. It can be seen that, for all $(cl, ax) \in \Pi'_{sc1}$, $cl^+ \not\subseteq \mathsf{atoms}(\mathsf{Cl}(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \setminus \Pi'_{sc1}))$, and that for all $(cl, ax) \in \Pi'_{sc2}$, $cl^- \not\subseteq \bigcup_{(cl,ax) \in \mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \setminus (\Pi'_{sc1} \cup \Pi'_{sc2})} cl^+$. Let $\{\Pi'_i\}_{1 \le i \le m'}$ be the set of maximal connected components of $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})) \setminus (\Pi'_{sc1} \cup \Pi'_{sc2})$. Let $\mathcal{M}'_i$ be the set of MISs of $\mathsf{Ax}(\Pi'_i)$ for $1 \leq i \leq m'$. By Theorem 1, $\bigcup_{i=1}^{m'} \mathcal{M}'_i$ is the set of all MISs of $\mathcal{O}$. Consider each $1 \leq i \leq m'$. Let $\Pi''_i = \{(\lambda(cl), ax) \mid (cl, ax) \in \Pi'_i\}$. It can be seen that $\Pi''_i \subseteq \Pi \setminus (\Pi_{sc1} \cup \Pi_{sc2})$. Since Π'_i is a connected component, Π''_i is a connected component of $\Pi \setminus (\Pi_{sc1} \cup \Pi_{sc2})$. Hence there exists Π_k $(1 \leq k \leq m)$ such that $\Pi''_i \subseteq \Pi_k$. It follows that $\mathsf{Ax}(\Pi'_i) = \mathsf{Ax}(\Pi''_i) \subseteq \mathsf{Ax}(\Pi_k)$, then $\mathcal{M}'_i \subseteq \mathcal{M}_k$. Thus $\bigcup_{i=1}^{m'} \mathcal{M}'_i \subseteq \bigcup_{i=1}^{m} \mathcal{M}_i$. Since $\mathcal{M}_i$ consists of only MISs of $\mathcal{O}$, $\bigcup_{i=1}^{m} \mathcal{M}_i$ is the set of all MISs of $\mathcal{O}$. □

To summarize, our proposed O-PP compilation process is to compile $\mathcal{O}$ to a superset Π of $\lambda(\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O})))$ for λ a convergent mapping function for $\mathcal{G}_{bu}(\mathcal{L}(\mathcal{O}))$, while the inverse process of the O-PP compilation is to extract axiom parts from the result of `Decompose`(Π). To compute such Π, we adapt Algorithm 2 given in [1] by considering labels. Due to the space limitation, we do not provide the adapted algorithm here but refer the interested reader to [1] for more details.

4 Experimental Evaluation

We implemented a prototype system[4] that applies the Pellet [11] (version 2.0.1) debugging facility to compute MISs in every extracted sub-ontology. There is an optimization in our implementation. That is, before computing MISs in extracted sub-ontologies, the sub-ontologies having the same set of terminological axioms are combined into a bin and the consistency of the bin is checked. Only sub-ontologies not in any consistent bin are considered when computing MISs.

We used two groups of inconsistent ontologies. The test ontologies in the first group, including University, Chemical and MiniTambis, are modified from homonymous incoherent ontologies[5] by adding a concept assertion for every atomic concept. The test ontologies in the second group are modified from the University Benchmark (UOBM-Lite) [7] ontologies[6] by inserting a specified number of conflicts using the `Injector` tool described in [2], where a conflict is a set of axioms violating a functional role restriction or a disjointness constraint. By UOBM-Liten_{+m} we denote an UOBM-Lite ontology with assertional axioms of n universities and with m conflicts inserted. The characteristics of all test ontologies are given in Table 1. All experiments were conducted on a PC with Pentium Dual Core 2.60GHz CPU and 2GB RAM, running Windows XP, where the maximum Java heap size was set to (max) 1280MB for applying Pellet.

We compared our implemented system with Pellet [11] (version 2.0.1)[7] on computing MISs of all test ontologies. Both systems were set to use the glass box debugging method. Typical comparison results on execution time and some runtime statistics of our system are shown in Table 2. For two out of the three

[4] See `http://jfdu.limewebs.com/dbo-debug/` for more details on our system.

[5] `http://www.mindswap.org/ontologies/debugging/`

[6] `http://www.alphaworks.ibm.com/tech/semanticstk/`

[7] To the best of our knowledge, Pellet is the only existing OWL DL reasoner that provides debugging facilities.

Table 1. The characteristics of all test ontologies

$\mathcal{O}$	Expressivity	#C	#R	#I	#Ax
University	$\mathcal{SOIF}(\mathbf{D})$	30	12	34	74
Chemical	$\mathcal{ALCHF}$	48	20	48	162
MiniTambis	$\mathcal{ALCF}$	183	44	183	350
UOBM-Lite1$_{+50\sim+250}$	$\mathcal{SHIF}(\mathbf{D})$	51	43	95,113–95,522	246,144–246,744
UOBM-Lite5$_{+50\sim+250}$				420,251–420,662	1,075,340–1,075,940
UOBM-Lite10$_{+50\sim+250}$				820,358–820,958	2,097,253–2,097,853

Note: "#C", "#R"", "#I"" and "#Ax" are respectively the numbers of atomic concepts, atomic roles, individuals and axioms in $\mathcal{O}$.

Table 2. Typical comparison results and some runtime statistics

$\mathcal{O}$	Pellet	Ours	Compile	Decompose	#MIS	#Sub	#A-S	#M-S
University	0:02:40	0:00:25	0:00:04	0:00:01	8	8	10	1
Chemical	>4:00:00	0:02:39	0:00:16	0:00:01	362	37	31	21
MiniTambis	>4:00:00	0:03:10	0:00:59	0:00:01	37	30	19	2
UOBM-Lite1$_{+50}$	>4:00:00	0:06:47	0:01:33	0:00:03	50	47	7	2
UOBM-Lite1$_{+250}$	>4:00:00	0:20:04	0:01:38	0:00:04	250	140	29	9
UOBM-Lite5$_{+50}$	>4:00:00	0:35:02	0:10:08	0:00:28	50	50	5	1
UOBM-Lite10$_{+50}$	out of mem	1:14:33	0:15:36	0:00:32	50	50	5	1

Note: "Pellet" (resp. "Ours") is the total time Pellet (resp. our system) spends to compute all MISs of $\mathcal{O}$; "Compile" (resp. "Decompose") is the time spent in the O-PP compilation process (resp. the decomposition process); "#MIS" is the number of MISs of $\mathcal{O}$; "#Sub" is the number of extracted sub-ontologies that are not in any consistent bin; "#A-S" (resp. "#M-S") is the maximum number of axioms (resp. MISs) in every extracted sub-ontology that is not in any consistent bin.

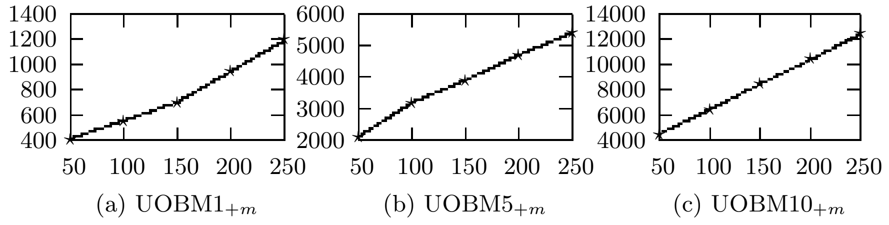


Fig. 1. The total execution time (seconds) against increasing numbers of conflicts

ontologies in the first group, Pellet cannot compute all MISs in four hours, while our system computes all MISs in a few minutes. For all ontologies in the second group, Pellet cannot compute all MISs in four hours or runs out of memory when loading the test ontology, while our system works well too. We also used Pellet to debug the original incoherent ontologies in the first group and found that Pellet works well when there is no assertional axioms. These results show that Pellet is hard to debug an inconsistent ontology having hundreds of axioms and some dozens of MISs where some axioms are assertional ones. In all our experiments, however, the Pellet debugging facility applied in our system only works on some extracted sub-ontology having at most 31 axioms and 21 MISs. This explains why our system works well and why it can significantly outperform Pellet.

We also tested our system against different numbers of inserted conflicts. As shown in Figure 1, our system scales well up to hundreds of conflicts when the number of conflicts increases.

5 Concluding Remarks

Debugging an inconsistent OWL DL ontology (i.e. computing all MISs of it) is an important problem in the Semantic Web, but currently lacks solutions that scale to large ontologies. For this situation, we proposed a solution to optimize the computation of all MISs and empirically showed its effectiveness in improving the scalability. The solution is based on a new compilation method from OWL DL to Propositional Logic (PL) and a new decomposition method on PL. Though our solution provides optimizations to existing OWL DL debugging facilities, the optimizations may not be effective in all possible cases. Hence we will research on approximate debugging methods in the future work.

References

1. Du, J., Qi, G., Ji, Q.: Goal-directed module extraction for explaining OWL DL entailments. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 163–179. Springer, Heidelberg (2009)
2. Du, J., Shen, Y.: Computing minimum cost diagnoses to repair populated DL-based ontologies. In: Proc. of WWW 2008, pp. 265–274 (2008)
3. Fitting, M.: First-order Logic and Automated Theorem Proving, 2nd edn. Springer, Secaucus (1996)
4. Haase, P., van Harmelen, F., Huang, Z., Stuckenschmidt, H., Sure, Y.: A framework for handling inconsistency in changing ontologies. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 353–367. Springer, Heidelberg (2005)
5. Horrocks, I., Patel-Schneider, P.F., van Harmelen, F.: From $\mathcal{SHIQ}$ and RDF to OWL: the making of a web ontology language. Journal of Web Semantics 1(1), 7–26 (2003)
6. Kazakov, Y., Motik, B.: A resolution-based decision procedure for $\mathcal{SHOIQ}$. Journal of Automated Reasoning 40(2-3), 89–116 (2008)
7. Ma, L., Yang, Y., Qiu, Z., Xie, G., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Sure, Y., Domingue, J. (eds.) ESWC 2006. LNCS, vol. 4011, pp. 125–139. Springer, Heidelberg (2006)
8. Parsia, B., Sirin, E., Kalyanpur, A.: Debugging OWL ontologies. In: Proc. of WWW 2005, pp. 633–640 (2005)
9. Reiter, R.: A theory of diagnosis from first principles. Artificial Intelligence 32(1), 57–95 (1987)

10. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Proc. of IJCAI 2003, pp. 355–362 (2003)
11. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5(2), 51–53 (2007)
12. Suntisrivaraporn, B., Qi, G., Ji, Q., Haase, P.: A modularization-based approach to finding all justifications for owl dl entailments. In: Domingue, J., Anutariya, C. (eds.) ASWC 2008. LNCS, vol. 5367, pp. 1–15. Springer, Heidelberg (2008)