Goal-Directed Module Extraction for Explaining OWL DL Entailments

Jianfeng Du[1,2], Guilin Qi[3,4], and Qiu Ji[3]

[1] Institute of Business Intelligence & Knowledge Discovery,
Guangdong University of Foreign Studies, Guangzhou 510006, China
[2] State Key Laboratory of Computer Science, Institute of Software,
Chinese Academy of Sciences, Beijing 100190, China
jfdu@ios.ac.cn
[3] AIFB, Universität Karlsruhe, D-76128 Karlsruhe, Germany
{gqi,qiji}@aifb.uni-karlsruhe.de
[4] School of Computer Science and Engineering, Southeast University, Nanjing, China

Abstract. Module extraction methods have proved to be effective in improving the performance of some ontology reasoning tasks, including finding justifications to explain why an entailment holds in an OWL DL ontology. However, the existing module extraction methods that compute a *syntactic locality-based module* for the sub-concept in a subsumption entailment, though ensuring the resulting module to preserve all justifications of the entailment, may be insufficient in improving the performance of finding all justifications. This is because a syntactic locality-based module is independent of the super-concept in a subsumption entailment and always contains all concept/role assertions. In order to extract smaller modules to further optimize finding all justifications in an OWL DL ontology, we propose a goal-directed method for extracting a module that preserves all justifications of a given entailment. Experimental results on large ontologies show that a module extracted by our method is smaller than the corresponding syntactic locality-based module, making the subsequent computation of all justifications more scalable and more efficient.

1 Introduction

As the Semantic Web (SW) is evolving, ontologies become more and more important because they provide formal representation of knowledge shared within the SW. To enable users to understand the knowledge in a consistent way, ontologies need to be logically constructed and have well-defined semantics. To this end, the W3C organization proposed the Web Ontology Language (OWL)[5] for representing ontologies, which is based on Description Logics (DLs) [1]. With well-defined semantics from DLs, some important ontology reasoning tasks emerged, including classical reasoning tasks like subsumption checking and instance retrieval [1], and non-standard reasoning tasks like finding *justifications* to explain why an entailment holds in a given ontology [12, 17], where a justification of an entailment is a minimal subset of axioms in the given ontology that are responsible for the entailment.

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

Reasoning tasks in OWL ontologies are in general computationally hard, thus rely on optimization techniques. To improve the performance of ontology reasoning, module extraction methods are often employed and their effectiveness has been adequately verified. A module extraction method reduces a given ontology to a specific module depending on specific tasks. These tasks include ontology reuse [16, 4], subsumption checking [21], incremental classification [7] and finding of justifications [2, 22].

In this paper, we focus on the problem of extracting small modules to improve the performance of an important reasoning task, namely finding all justifications of a given entailment. The idea of using modules to optimize finding justifications is not new. It has been shown in [2] and [22] that the syntactic locality-based module introduced in [8] can be used to improve the performance of finding all justifications. As proved in [22], a syntactic locality-based module is a module that preserves all subsumption entailments $A \sqsubseteq B$ w.r.t. a concept name A and is independent of the super-concept B. It follows that the extraction of a syntactic locality-based module is not goal-directed; it may yield an unexpectedly large module as illustrated below.

Example 1. Let $\mathcal{O}$ be an ontology consisting of the following axioms.
ax_1 : $\mathsf{ChiefActress} \sqsubseteq \mathsf{Person}$ ax_2 : $\mathsf{ChiefActress} \sqsubseteq \mathsf{Actress}$ ax_3 : $\mathsf{Actress} \sqsubseteq \mathsf{Woman}$
ax_4 : $\mathsf{Person} \sqsubseteq \mathsf{Man} \sqcup \mathsf{Woman}$ ax_5 : $\mathsf{ChiefActress} \sqsubseteq \neg\mathsf{Man}$

Consider the above example. There are two justifications $\{ax_1, ax_4, ax_5\}$ and $\{ax_2, ax_3\}$ of the subsumption entailment $\mathsf{ChiefActress} \sqsubseteq \mathsf{Woman}$ in $\mathcal{O}$. The syntactic locality-based module w.r.t. $\mathsf{ChiefActress}$ in $\mathcal{O}$ is exactly $\mathcal{O}$. This module cannot be used to optimize finding all justifications of a subsumption entailment of the form $\mathsf{ChiefActress} \sqsubseteq B$ no matter what the concept name B is. On the other hand, a syntactic locality-based module must contain all concept/role assertions because these assertions can never be *local* [8]. This means that extracting a syntactic locality-based module can hardly optimize finding justifications of a membership entailment $A(a)$ or $R(a, b)$, as all concept/role assertions in the given ontology must be involved.

In order to extract modules smaller than the syntactic locality-based ones to further optimize finding justifications, we propose a goal-directed method for module extraction in $\mathcal{SHOIQ}$ ontologies (see Section 4). The DL $\mathcal{SHOIQ}$ corresponds to OWL DL allowing qualifying number restrictions. Since the problem of finding justifications of a subsumption entailment can be reduced to the problem of finding justifications of a membership entailment by introducing new concept assertions, our method focuses on finding a module that preserves all justifications of a given membership entailment. The basic idea is to start from a given membership entailment and backward traverse a set of axioms that might be responsible for the given entailment. To ensure the traversed set to be an intended module, our method compiles a propositional program from the given ontology in which each clause may have a flag corresponding to an axiom in the given ontology, such that the traversal of clauses corresponds to the traversal of axioms, i.e., the set of flags appearing in the traversed clauses corresponds to an intended module.

We implement the proposed method and test on large ontologies (see Section 5). Experimental results show that our method can extract modules that are much smaller than syntactic locality-based modules. They also show that, with our proposed module extraction method, it is possible to efficiently find all justifications of a membership entailment in a large ontology with millions of ABox axioms.

2 Related Work

Module extraction methods form an important category in ontology modularization. Some methods rely on syntactically traversing axioms in the ontology and exploiting different heuristics to determine which axioms belong to the extracted module, such as those given in [16], [18] and [4]. Other methods formally specify the intended outputs by considering the semantics of the ontology language, such as the method proposed by Cuenca Grau et al. [8], which computes a so-called locality-based module that preserves all subsumption entailments $A \sqsubseteq B$ for a given concept name A. There are two kinds of locality-based modules, namely semantic ones and syntactic ones, both contain all concept/role assertions [8]. The method proposed by Suntisrivaraporn [21] computes so-called reachability-based modules in $\mathcal{EL}^+$ ontologies, which coincide with syntactic locality-based modules. The above methods do not concern extracting a suitable module to optimize finding all justifications.

Another important category in ontology modularization is formed by ontology partitioning methods, each of which divides a given ontology into a set of modules. For example, Stuckenschmidt and Klein [20] propose a method to partition all concepts in an ontology to facilitate visualization of the ontology; Cuenca Grau *et al.* [9] propose a method for dividing an ontology into a set of modules such that all inferences about the signature contained in a module can be made locally. These methods are less relevant to ours because they yield a set of modules other than a single one.

3 Preliminaries

$\mathcal{SHOIQ}$ Syntax and Semantics. A $\mathcal{SHOIQ}$ ontology $\mathcal{O}$ consists of a *terminological box* (*TBox*) $\mathcal{O}_\mathcal{T}$ and an *assertional box* (*ABox*) $\mathcal{O}_\mathcal{A}$. The TBox $\mathcal{O}_\mathcal{T}$ consists of a finite set of *concept inclusion axioms* $C \sqsubseteq D$, *transitivity axioms* $\mathrm{Trans}(R)$ and *role inclusion axioms* $R \sqsubseteq S$, where C and D are $\mathcal{SHOIQ}$ concepts, and R and S roles. A role is either a role name or an *inverse role* R^- for some role name R; it is called *simple* if it has not any transitive subrole. The set of $\mathcal{SHOIQ}$ concepts is the smallest set such that each concept name A (also called *atomic concept*) is a $\mathcal{SHOIQ}$ concept and, for C and D $\mathcal{SHOIQ}$ concepts, R a role, S a simple role, a an individual, and n a nonnegative integer, $\top$, $\bot$, $\{a\}$, $\neg C$, $C \sqcap D$, $C \sqcup D$, $\exists R.C$, $\forall R.C$, $\leq_n S.C$ and $\geq_n S.C$ are also $\mathcal{SHOIQ}$ concepts. The ABox $\mathcal{O}_\mathcal{A}$ consists of a finite set of *concept assertions* $C(a)$, *role assertions* $R(a, b)$ or $\neg R(a, b)$, *equality assertions* $a \approx b$ and *inequality assertions* $a \not\approx b$, where C is a $\mathcal{SHOIQ}$ concept, R a role, and a and b individuals. In this paper, both $\mathcal{O}_\mathcal{T}$ and $\mathcal{O}_\mathcal{A}$ are treated as sets of axioms, so is $\mathcal{O} = \mathcal{O}_\mathcal{T} \cup \mathcal{O}_\mathcal{A}$.

As a fragment of first-order logic, $\mathcal{SHOIQ}$ inherits its semantics from first-order logic by the standard translations known e.g. from [15]. Let π denote the operator for mapping a $\mathcal{SHOIQ}$ ontology (resp. axiom) to a first-order logic program (resp. clause) as given in [15]. Then $\mathcal{O}$ is called *consistent* or *satisfiable* if $\pi(\mathcal{O})$ has a model. An atomic concept A is called *satisfiable* in $\mathcal{O}$ if there exists a model M of $\pi(\mathcal{O})$ such that $A(a) \in M$ for some constant a. $\mathcal{O}$ is called *coherent* if all atomic concepts in $\mathcal{O}$ are satisfiable. When $\mathcal{O}$ has atomic concepts, $\mathcal{O}$ being coherent implies $\mathcal{O}$ being consistent. An axiom ax is called an *entailment* of $\mathcal{O}$, denoted by $\mathcal{O} \models ax$, if $\pi(ax)$ is satisfied by all models of $\pi(\mathcal{O})$. An entailment of $\mathcal{O}$ is called a *subsumption entailment* (resp. a *membership entailment*) if it is of the form $C \sqsubseteq D$ (resp. $C(a)$ or $R(a, b)$).

First-Order Logic. We use the standard clausal form to represent a first-order logic program. *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. Throughout this paper, we use (possibly with subscripts) x, y, z for variables, a, b, c for constants, and s, t for terms. We only consider unary function symbols because only unary function symbols occur in first-order logic programs that are translated from $\mathcal{SHOIQ}$ 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* 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 first-order logic program consisting of only ground clauses. The set of ground atoms occurring in Π is denoted by $\mathsf{atoms}(\Pi)$.

For a first-order logic 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 $\mathrm{HU}(P)$ (resp. $\mathrm{HB}(P)$). The set of ground clauses obtained by replacing all variables occurring in each clause in P with ground terms from $\mathrm{HU}(P)$ is called the *primary grounding* of P, denoted by $\mathcal{G}(P)$. An *interpretation* M of P is a set of ground atoms in $\mathrm{HB}(P)$; 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 first-order logic program P translated from a $\mathcal{SHOIQ}$ 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 [6]. Let $\mathcal{E}(P)$ denote the first-order logic program consisting of the following clauses: (1) $t \approx t$, for each ground term $t \in \mathrm{HU}(P)$; (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 $cl^- \subseteq M$, $cl^+ \cap M \neq \emptyset$.

4 Goal-Directed Module Extraction

We in this paper address the following problem: Given a coherent $\mathcal{SHOIQ}$ ontology $\mathcal{O}$ which contains atomic concepts (thus $\mathcal{O}$ is consistent too) and an entailment ax of the form $A \sqsubseteq B$, $A(a)$ or $R(a, b)$ in $\mathcal{O}$, where A and B are concept names, R a role name, and a and b individuals, extract a *just-preserving module* of $\mathcal{O}$ for ax, which preserves all *justifications* of ax.

Definition 1 (Justification). For $\mathcal{O}$ an ontology and ax an axiom such that $\mathcal{O} \models ax$, a subset $\mathcal{J}$ of axioms in $\mathcal{O}$ is called a *justification* of ax in $\mathcal{O}$ if $\mathcal{J} \models ax$ and for all proper subsets $\mathcal{J}'$ of $\mathcal{J}$, $\mathcal{J}' \not\models ax$.

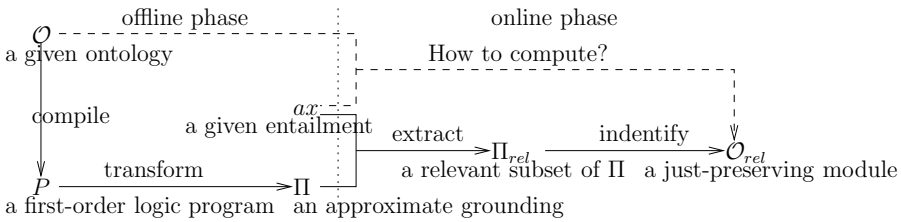

Fig. 1. Illustration of our proposed method

Definition 2 (Just-Preserving Module). For $\mathcal{O}$ an ontology and ax an axiom such that $\mathcal{O} \models ax$, a subset $\mathcal{M}$ of axioms in $\mathcal{O}$ is called a *just-preserving module* of $\mathcal{O}$ for ax if $\mathcal{J} \subseteq \mathcal{M}$ for all justifications $\mathcal{J}$ of ax in $\mathcal{O}$.

We present a goal-directed method for extracting just-preserving modules, depicted in Figure 1. The method consists of two phases. In the first phase, the method first compiles a given $\mathcal{SHOIQ}$ ontology $\mathcal{O}$ to a first-order logic program P (see Subsection 4.1), then transforms P to a propositional program Π. Both P and Π are independent of any given entailment, thus this phase can be performed offline. To show the correctness of the method in a clearer way, we present two algorithms for transforming P to Π, though only one of them is used in the method. One algorithm transforms P to a propositional program which has the same set of minimal models as P has (see Subsection 4.2). The other algorithm transforms P to a finite variant of the above propositional program (see Subsection 4.4); it is actually used in the proposed method. In the second phase, for every given entailment ax, the method extracts a relevant subset Π_{rel} from Π in a goal-directed manner and then identifies a just-preserving module for ax from Π_{rel} (see Subsection 4.3). Due to the space limitation, we do not provide proofs of lemmas and theorems in this paper, but refer the interested reader to our technical report[6].

4.1 Compiling a Diagnosing Program

Let $\mathcal{O}^\dagger$ denote $\mathcal{O} \cup \{A(a_A) \mid A$ is an atomic concept in $\mathcal{O}$ and a_A is a new globally unique individual corresponding to $A\}$. Recall that $\mathcal{O}$ is assumed coherent and having concept names in our addressing problem, so $\mathcal{O}^\dagger$ is consistent. Since the new individuals introduced in $\mathcal{O}^\dagger$ is unique, we have $\mathcal{O} \models A \sqsubseteq B$ iff $\mathcal{O}^\dagger \models B(a_A)$. Hence, $\mathcal{J}$ is a justification of $A \sqsubseteq B$ in $\mathcal{O}$ iff $\mathcal{J} \cup \{A(a_A)\}$ is a justification of $B(a_A)$ in $\mathcal{O}^\dagger$. It follows that $\mathcal{M}$ is a just-preserving module of $\mathcal{O}$ for $A \sqsubseteq B$ iff $\mathcal{M} \cup \{A(a_A)\}$ is a just-preserving module of $\mathcal{O}^\dagger$ for $B(a_A)$. That is, our addressing problem can be reduced to a problem of extracting a just-preserving module for a membership entailment. We therefore, in what follows, focus on membership entailments.

Note that, for an concept/role assertion ax, $\mathcal{O}^\dagger \models ax$ iff $\mathcal{O}^\dagger \cup \{\neg ax\}$ is inconsistent, so $\mathcal{J}$ is a justification of ax in $\mathcal{O}^\dagger$ iff $\mathcal{J} \cup \{\neg ax\}$ is a minimally inconsistent subset of axioms in $\mathcal{O}^\dagger \cup \{\neg ax\}$ (simply called a *MIS* of $\mathcal{O}^\dagger \cup \{\neg ax\}$). It follows that $\mathcal{M}$ is a just-preserving module of $\mathcal{O}^\dagger$ for ax iff $S \subseteq \mathcal{M} \cup \{\neg ax\}$ for all MISs S of $\mathcal{O}^\dagger \cup \{\neg ax\}$. To extract a just-preserving module of $\mathcal{O}^\dagger$ for ax, we characterize all MISs of $\mathcal{O}^\dagger \cup \{\neg ax\}$

[6] `http://www.aifb.uni-karlsruhe.de/WBS/gqi/jp-module/module-long.pdf`

by a first-order logic program, called the *diagnosing program* of $\mathcal{O}^\dagger$, denoted by $\mathcal{D}(\mathcal{O}^\dagger)$, in which every clause may have a positive literal $\hbar_{ax}$ that corresponds to an axiom $ax \in \mathcal{O}^\dagger$. The atom $\hbar_{ax}$ is called the *decision atom* of ax. It is nullary and works as a flag to associate the clauses containing it with the axiom ax. We detail below the method for computing the diagnosing program of $\mathcal{O}^\dagger$.

We first translate $\mathcal{O}^\dagger$ to a first-order logic program. Since a direct translation from $\mathcal{SHOIQ}$ to first-order clauses may incur exponential blowup [13], we apply the well-known *structural transformation* [11, 13] to an axiom before translating it to first-order clauses. By $\Theta(ax)$ we denote the result of applying structural transformation and clausification (i.e. translating to clauses) to an axiom ax. As both structural transformation and clausification are well-known, we do not give their definition here but refer the reader to [13] or our technical report[6].

Let $\Xi(ax)$ denote the set of clauses obtained from $\Theta(ax)$ by adding the decision atom $\hbar_{ax}$ to every clause in $\Theta(ax)$, i.e., $\Xi(ax) = \{cl \vee \hbar_{ax} \mid cl \in \Theta(ax)\}$. Let $\Xi(\mathcal{O}^\dagger) = \bigcup_{ax \in \mathcal{O}^\dagger} \Xi(ax)$. Then the *diagnosing program* of $\mathcal{O}^\dagger$, i.e. $\mathcal{D}(\mathcal{O}^\dagger)$, is defined as follows: if some equational atom $s \approx t$ occurs positively in $\Xi(\mathcal{O}^\dagger)$, then $\mathcal{D}(\mathcal{O}^\dagger) = \Xi(\mathcal{O}^\dagger) \cup \mathcal{E}(\Xi(\mathcal{O}^\dagger))$, otherwise $\mathcal{D}(\mathcal{O}^\dagger) = \Xi(\mathcal{O}^\dagger)$. Recall that $\mathcal{E}(\Xi(\mathcal{O}^\dagger))$ is used to axiomatize the equality predicate in $\Xi(\mathcal{O}^\dagger)$ (see Section 3).

Example 2 (Example 1 continued). Consider the ontology $\mathcal{O}$ in Example 1. We compute $\mathcal{O}^\dagger$ as $\mathcal{O} \cup \{ax_6 : \mathsf{ChiefActress}(a_1), ax_7 : \mathsf{Person}(a_2), ax_8 : \mathsf{Actress}(a_3), ax_9 : \mathsf{Woman}(a_4), ax_{10} : \mathsf{Man}(a_5)\}$, where $a_1, \ldots, a_5$ are all new individuals. We then compute $\Xi(\mathcal{O}^\dagger)$ as $\{cl_1, \ldots, cl_{10}\}$ given below.

$cl_1 : \neg\mathsf{ChiefActress}(x) \vee \mathsf{Person}(x) \vee \hbar_{ax_1}$ $\quad$ $cl_2 : \neg\mathsf{ChiefActress}(x) \vee \mathsf{Actress}(x) \vee \hbar_{ax_2}$
$cl_3 : \neg\mathsf{Actress}(x) \vee \mathsf{Woman}(x) \vee \hbar_{ax_3}$ $\quad$ $cl_4 : \neg\mathsf{Person}(x) \vee \mathsf{Man}(x) \vee \mathsf{Woman}(x) \vee \hbar_{ax_4}$
$cl_5 : \neg\mathsf{ChiefActress}(x) \vee \neg\mathsf{Man}(x) \vee \hbar_{ax_5}$ $\quad$ $cl_6 : \mathsf{ChiefActress}(a_1) \vee \hbar_{ax_6}$
$cl_7 : \mathsf{Person}(a_2) \vee \hbar_{ax_7}$ $\quad$ $cl_8 : \mathsf{Actress}(a_3) \vee \hbar_{ax_8}$
$cl_9 : \mathsf{Woman}(a_4) \vee \hbar_{ax_9}$ $\quad$ $cl_{10} : \mathsf{Man}(a_5) \vee \hbar_{ax_{10}}$

Since there is no equational atom occurring positively in $\Xi(\mathcal{O}^\dagger)$, the diagnosing program of $\mathcal{O}^\dagger$, i.e. $\mathcal{D}(\mathcal{O}^\dagger)$, is exactly $\Xi(\mathcal{O}^\dagger)$.

Before showing how to use the diagnosing program, we introduce some notions. Given a first-order logic program P, a set X of ground atoms never occurring negatively in P and a truth assignment Φ_X on X, we define the *reduction* of P w.r.t. Φ_X, denoted by $P \downarrow \Phi_X$, as a first-order logic program obtained from P by deleting every clause $cl \in P$ that contains a ground atom $\alpha \in X$ such that $\Phi_X(\alpha) = \mathrm{true}$, and by removing all ground atoms in X from the remaining clauses. Since any ground atom in X does not occur negatively in P, it is clear that $P \downarrow \Phi_X$ is satisfiable iff there exists a model M of P such that $M \cap X = \{\alpha \in X \mid \Phi_X(\alpha) = \mathrm{true}\}$.

In the remainder of this section, let X denote the set of decision atoms in $\mathcal{D}(\mathcal{O}^\dagger)$, i.e., $X = \{\hbar_{ax} \mid ax \in \mathcal{O}^\dagger\}$, and ax a membership entailment of $\mathcal{O}^\dagger$ on concept/role names if not otherwise specified. Consider an ontology $\mathcal{O}'$ such that $\mathcal{O}^\dagger \subseteq \mathcal{O}'$. For Φ_X a truth assignment on X, we define the *reduction* of $\mathcal{O}'$ w.r.t. Φ_X, denoted by $\mathcal{O}' \downarrow \Phi_X$, as $\mathcal{O}' \backslash \{ax' \in \mathcal{O}^\dagger \mid \Phi_X(\hbar_{ax'}) = \mathrm{true}\}$, i.e. the subset of $\mathcal{O}'$ where all axioms $ax' \in \mathcal{O}^\dagger$ such that $\Phi_X(\hbar_{ax'}) = \mathrm{true}$ are absent. We have the following relationship.

Lemma 1. *Let $P = \mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$ and $\mathcal{O}' = \mathcal{O}^\dagger \cup \{\neg ax\}$. For any truth assignment Φ_X on X, $P \downarrow \Phi_X$ is satisfiable iff $\mathcal{O}' \downarrow \Phi_X$ is satisfiable.*

The above relationship implies a correspondence between MISs of $\mathcal{O}^\dagger \cup \{\neg ax\}$ and *maximally unsatisfiable X-assignments* for $\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$ (see Lemma 2), where a maximally unsatisfiable X-assignment for $\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$ is a truth assignment Φ_X on X such that $(\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}) \downarrow \Phi_X$ is unsatisfiable and $\{\alpha \in X \mid \Phi_X(\alpha) = \text{true}\}$ is maximal, formally defined below.

Definition 3 (Maximally Unsatisfiable X-Assignment). For P a first-order logic program and X a set of ground atoms never occurring negatively in P, a truth assignment Φ_X on X for P is called an *unsatisfiable X-assignment* for P if $P \downarrow \Phi_X$ is unsatisfiable; an unsatisfiable X-assignment Φ_X for P is called a *maximally unsatisfiable X-assignment* for P if there is no other unsatisfiable X-assignment Φ'_X for P such that $\{\alpha \in X \mid \Phi_X(\alpha) = \text{true}\} \subset \{\alpha \in X \mid \Phi'_X(\alpha) = \text{true}\}$.

Lemma 2. *Let $P = \mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$ and $\mathcal{O}' = \mathcal{O}^\dagger \cup \{\neg ax\}$. For a MIS S of $\mathcal{O}'$, the truth assignment Φ_X on X, such that $\Phi_X(\hbar_{ax'}) = \text{true}$ iff $ax' \in \mathcal{O}' \setminus S$ for all $ax' \in \mathcal{O}^\dagger$, is a maximally unsatisfiable X-assignment for P. For a maximally unsatisfiable X-assignment Φ_X for P, $\mathcal{O}' \setminus \{ax' \in \mathcal{O}^\dagger \mid \Phi_X(\hbar_{ax'}) = \text{true}\}$ is a MIS of $\mathcal{O}'$.*

According to Lemma 2, we can characterize a just-preserving module of $\mathcal{O}^\dagger$ for ax with the help of maximally unsatisfiable X-assignments for $\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$.

Theorem 1. *$\mathcal{M}$ is a just-preserving module of $\mathcal{O}^\dagger$ for ax iff $\{ax' \in \mathcal{O}^\dagger \mid \Phi_X(\hbar_{ax'}) = \text{false}\} \subseteq \mathcal{M}$ for all maximally unsatisfiable X-assignments Φ_X for $\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$.*

4.2 Exact Grounding of the Diagnosing Program

According to Theorem 1, we need to consider satisfiability problems on $\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$. This can be done by considering a propositional program that is transformed from $\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$ and has the same set of minimal models as $\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}$ has. We extend the well-known intelligent grounding (IG) technique [5] which computes, in a fixpoint-evaluation manner, a semantically equivalent propositional program containing only derivable ground atoms from a function-free first-order logic program. By generalizing the idea of the IG technique, we define the so-called *bottom-up grounding* of a general first-order logic program P, denoted by $\mathcal{G}_{bu}(P)$, as the least fixpoint of $\Pi^{(n)}$ such that $\Pi^{(0)} = \emptyset$ and for $n \geq 1$, $\Pi^{(n)} = \{cl\,\sigma \mid cl \in P,\ \sigma$ is a ground substitution such that $cl^-\sigma \subseteq \mathsf{atoms}(\Pi^{(n-1)})$ and $cl^+\sigma \subseteq \text{HB}(P)\}$.

Lemma 3. *Let P be a first-order logic program in which the equality predicate $\approx$ has been axiomatized. Then (1) $\mathcal{G}_{bu}(P)$ is the least subset S of $\mathcal{G}(P)$ such that $cl\,\sigma \in S$ for any clause $cl \in P$ and any ground substitution σ such that $cl^-\sigma \subseteq \mathsf{atoms}(S)$ and $cl^+\sigma \subseteq \text{HB}(P)$; (2) $\mathcal{G}_{bu}(P)$ has the same set of minimal models as P has.*

Example 3 (Example 2 continued). This example shows the steps of computing the bottom-up grounding of $\mathcal{D}(\mathcal{O}^\dagger)$ given in Example 2. By applying the fixpoint-evaluation process for defining $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$, we have $\Pi^{(0)} = \emptyset$, $\Pi^{(1)} = \{cl'_1, ..., cl'_5\}$, $\Pi^{(2)} = \{cl'_1, ..., cl'_9\}$, $\Pi^{(3)} = \{cl'_1, ..., cl'_{11}\}$, $\Pi^{(4)} = \{cl'_1, ..., cl'_{12}\}$ and $\Pi^{(5)} = \Pi^{(4)}$, so $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) = \{cl'_1, ..., cl'_{12}\}$, where $cl'_1, \ldots, cl'_{12}$ are given below.

cl'_1: $\mathsf{ChiefActress}(a_1) \vee \hbar_{ax_6}$
cl'_2: $\mathsf{Person}(a_2) \vee \hbar_{ax_7}$
cl'_3: $\mathsf{Actress}(a_3) \vee \hbar_{ax_8}$
cl'_4: $\mathsf{Woman}(a_4) \vee \hbar_{ax_9}$
cl'_5: $\mathsf{Man}(a_5) \vee \hbar_{ax_{10}}$
cl'_6: $\neg\mathsf{ChiefActress}(a_1) \vee \mathsf{Person}(a_1) \vee \hbar_{ax_1}$
cl'_7: $\neg\mathsf{ChiefActress}(a_1) \vee \mathsf{Actress}(a_1) \vee \hbar_{ax_2}$
cl'_8: $\neg\mathsf{Actress}(a_3) \vee \mathsf{Woman}(a_3) \vee \hbar_{ax_3}$
cl'_9: $\neg\mathsf{Person}(a_2) \vee \mathsf{Man}(a_2) \vee \mathsf{Woman}(a_2) \vee \hbar_{ax_4}$
cl'_{10}: $\neg\mathsf{Actress}(a_1) \vee \mathsf{Woman}(a_1) \vee \hbar_{ax_3}$
cl'_{11}: $\neg\mathsf{Person}(a_1) \vee \mathsf{Man}(a_1) \vee \mathsf{Woman}(a_1) \vee \hbar_{ax_4}$
cl'_{12}: $\neg\mathsf{ChiefActress}(a_1) \vee \neg\mathsf{Man}(a_1) \vee \hbar_{ax_5}$

Since ax is a membership entailment of $\mathcal{O}^\dagger$, by lemma 3, $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \models ax$. Hence ax occurs positively in some clauses in $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$. Note that $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\})$ can be computed by first grounding $\mathcal{D}(\mathcal{O}^\dagger)$ then adding the clause $\{\neg ax\}$, and that the adding of clause $\neg ax$ to $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$ does not introduce new ground atoms to $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$, so $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger) \cup \{\neg ax\}) = \mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \cup \{\neg ax\}$. It follows from Theorem 1 and Lemma 3 that $\mathcal{M}$ is a just-preserving module of $\mathcal{O}^\dagger$ for ax iff $\{ax' \in \mathcal{O}^\dagger \mid \Phi_X(\hbar_{ax'}) = \text{false}\} \subseteq \mathcal{M}$ for all maximally unsatisfiable X-assignments Φ_X for $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \cup \{\neg ax\}$. Subsequently, a just-preserving module of $\mathcal{O}^\dagger$ for ax can be computed as $\{ax' \in \mathcal{O}^\dagger \mid \hbar_{ax'} \in \mathsf{atoms}(\Pi_{rel})\}$, where Π_{rel} is a subset of $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \cup \{\neg ax\}$ such that $\{\hbar_{ax'} \in X \mid \Phi_X(\hbar_{ax'}) = \text{false}\} \subseteq \mathsf{atoms}(\Pi_{rel})$ for all maximally unsatisfiable X-assignments Φ_X for $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \cup \{\neg ax\}$. We call such Π_{rel} a *just-preserving relevant subset* of $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \cup \{\neg ax\}$ for ax.

4.3 Extracting a Justification-Preserving Module

In this subsection, by Π we denote $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \cup \{\neg ax\}$ if not otherwise specified. Our method for extracting a just-preserving relevant subset of Π is based on the notion of a *connected component* (see Definition 4 below), which is a subset of a propositional program such that any two clauses in it have common ground atoms. This notion has been used to confine the search space in solving SAT problems [3] because an unsatisfiable propositional program must have a maximal connected component that is unsatisfiable.

Definition 4 (Connected Component). Let Π be a propositional program. Two ground clauses cl and cl' are called *connected* in Π if there exists a sequence of clauses $cl_0 = cl, cl_1, \ldots, cl_n = cl'$ in Π such that cl_{i-1} and cl_i have common ground atoms for any $1 \leq i \leq n$. A *connected component* Π_c of Π is a subset of Π such that any two clauses cl and cl' in Π_c are connected in Π_c. Π_c is called *maximal* if there is no connected component Π'_c of Π such that $\Pi_c \subset \Pi'_c$.

The basic idea employed in our method is that the maximal connected component of Π where ax occurs is a just-preserving relevant subset of Π for ax. To obtain a smaller just-preserving relevant subset, our method extends the basic idea by first removing two subsets Π_{ur1} and Π_{ur2} from Π in turn, then extracting a maximal connected component from the remaining set. The description and the correctness of the method are shown in Lemma 4, while the idea is illustrated in Example 4.

Lemma 4. *Let Π_{ur1} be a subset of Π such that for all clauses $cl \in \Pi_{ur1}$, cl^+ contains at least one ground atom not in $X \cup \mathsf{atoms}(\Pi \setminus \Pi_{ur1})$. Let Π_{ur2} be a subset of $\Pi \setminus \Pi_{ur1}$ such that $cl^- \not\subseteq \bigcup_{cl \in \Pi \setminus (\Pi_{ur1} \cup \Pi_{ur2})} cl^+$ for all clauses $cl \in \Pi_{ur2}$. Let Π_{rel} be the maximal connected component of $\Pi \setminus (\Pi_{ur1} \cup \Pi_{ur2})$ where ax occurs. Then Π_{rel} is a just-preserving relevant subset of Π for ax, i.e., $\{\hbar_{ax'} \in X \mid \Phi_X(\hbar_{ax'}) = \text{false}\} \subseteq \mathsf{atoms}(\Pi_{rel})$ for all maximally unsatisfiable X-assignments Φ_X for Π.*

According to Lemma 4, we develop an algorithm for extracting a just-preserving relevant subset of Π for ax, shown in Figure 2. In the algorithm, lines 1–5 compute Π_{ur1} given in Lemma 4, lines 6–10 compute Π_{ur2} (where $\Pi_0 \setminus \Pi_{ur1}$ in the algorithm corresponds to Π_{ur2} given in Lemma 4), and lines 11-15 compute Π_{rel}. Note that Π_{rel} is computed in a goal-directed (backward traversal) manner.

Example 4 (Example 3 cont.). Let $\Pi = \mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \cup \{\neg\mathsf{Person}(a_1)\}$ for $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$ given in Example 3. This example shows the sets Π_{ur1}, Π_{ur2} and Π_{rel} that may be computed by our method.

cl'_2 : $\boxed{\mathsf{Person}(a_2)} \vee \hbar_{ax_7}$
cl'_3 : $\boxed{\mathsf{Actress}(a_3)} \vee \hbar_{ax_8}$
cl'_4 : $\boxed{\mathsf{Woman}(a_4)} \vee \hbar_{ax_9}$
cl'_5 : $\boxed{\mathsf{Man}(a_5)} \vee \hbar_{ax_{10}}$
cl'_7 : $\neg\mathsf{ChiefActress}(a_1) \vee \boxed{\mathsf{Actress}(a_1)} \vee \hbar_{ax_2}$
cl'_8 : $\neg\mathsf{Actress}(a_3) \vee \boxed{\mathsf{Woman}(a_3)} \vee \hbar_{ax_3}$
cl'_9 : $\neg\mathsf{Person}(a_2) \vee \boxed{\mathsf{Man}(a_2)} \vee \boxed{\mathsf{Woman}(a_2)} \vee \hbar_{ax_4}$
cl'_{10} : $\neg\mathsf{Actress}(a_1) \vee \boxed{\mathsf{Woman}(a_1)} \vee \hbar_{ax_3}$
cl'_{11} : $\neg\mathsf{Person}(a_1) \vee \mathsf{Man}(a_1) \vee \boxed{\mathsf{Woman}(a_1)} \vee \hbar_{ax_4}$

Π_{ur1} : for all clauses $cl \in \Pi_{ur1}$, cl^+ contains at least one ground atom not in $X \cup \mathsf{atoms}(\Pi \setminus \Pi_{ur1})$.

The set $M_0 = \bigcup_{cl \in \Pi_{ur1}} cl^+ \setminus \mathsf{atoms}(\Pi \setminus \Pi_{ur1})$, i.e. the set of framed ground atoms, is a model of Π_{ur1}.

cl'_{12} : $\neg\mathsf{ChiefActress}(a_1) \vee \neg\mathsf{Man}(a_1) \vee \hbar_{ax_5}$

Π_{ur2} : for all clauses $cl \in \Pi_{ur2}$, $cl^- \not\subseteq \bigcup_{cl \in \Pi \setminus (\Pi_{ur1} \cup \Pi_{ur2})} cl^+$.

Note that $M_0 \cup \mathcal{A}$ is a model of Π_{ur2} for any subset $\mathcal{A}$ of ground atoms occurring in $\Pi \setminus (\Pi_{ur1} \cup \Pi_{ur2})$.

$\neg\mathsf{Person}(a_1)$
cl'_6 : $\neg\mathsf{ChiefActress}(a_1) \vee \mathsf{Person}(a_1) \vee \hbar_{ax_1}$
cl'_1 : $\mathsf{ChiefActress}(a_1) \vee \hbar_{ax_6}$

Π_{rel} = the maximal connected component of $\Pi \setminus (\Pi_{ur1} \cup \Pi_{ur2})$ where $\mathsf{Person}(a_1)$ occurs.

Π_{rel} is a just-preserving subset of Π for $\mathsf{Person}(a_1)$, and $\{ax' \in \mathcal{O}^\dagger \mid \hbar_{ax'} \in \mathsf{atoms}(\Pi_{rel})\} = \{ax_1, ax_6\}$ is a just-preserving module of $\mathcal{O}^\dagger$ for $\mathsf{Person}(a_1)$.

In this example, $\Pi = \Pi_{ur1} \cup \Pi_{ur2} \cup \Pi_{rel}$. The key point to prove that Π_{rel} is a just-preserving subset of Π is that for any maximally unsatisfiable X-assignment Φ_X of Π, $\Phi_X(\alpha) = \mathtt{true}$ for all $\alpha \in X \setminus \mathsf{atoms}(\Pi_{rel})$, otherwise $\Pi \downarrow \Phi'_X$ is satisfiable for Φ'_X the truth assignment on X such that $\Phi'_X(\alpha) = \Phi_X(\alpha)$ for all $\alpha \in X \cap \mathsf{atoms}(\Pi_{rel})$ and $\Phi'_X(\alpha) = \mathtt{true}$ for all $\alpha \in X \setminus \mathsf{atoms}(\Pi_{rel})$; this implies that $\Pi_{rel} \downarrow \Phi_X = \Pi_{rel} \downarrow \Phi'_X$ is satisfiable and $\Pi_{rel} \downarrow \Phi_X$ has a minimal model M, but then $M \cup M_0$ is model of $\Pi \downarrow \Phi_X$ (contradiction).

Algorithm 1. $\mathrm{Extract}(\Pi_{in}, X, \alpha)$

Input: A propositional program Π_{in} with decision atoms, the set X of decision atoms occurring in Π_{in}, and a ground atom α occurring in Π_{in}.

Output: A subset of $\Pi_{in} \cup \{\neg\alpha\}$.

1. $\Pi_{ur1} := \Pi_{in}$; $\Pi'_{ur1} := \emptyset$;
2. **while** $\Pi'_{ur1} \neq \Pi_{ur1}$ **do**
3. $\quad \Pi'_{ur1} := \Pi_{ur1}$;
4. $\quad$ **for** each clause $cl \in \Pi_{ur1}$ such that $cl^+ \subseteq X \cup \mathsf{atoms}(\Pi_{in} \setminus \Pi_{ur1})$ **do**
5. $\quad\quad \Pi_{ur1} := \Pi_{ur1} \setminus \{cl\}$;
6. $\Pi_0 := \Pi_{ur1}$; $\Pi'_0 := \emptyset$;
7. **while** $\Pi'_0 \neq \Pi_0$ **do**
8. $\quad \Pi'_0 := \Pi_0$;
9. $\quad$ **for** each clause $cl \in \Pi_{in} \setminus \Pi_0$ such that $cl^- \not\subseteq \bigcup_{cl \in \Pi_{in} \setminus \Pi_0} cl^+$ **do**
10. $\quad\quad \Pi_0 := \Pi_0 \cup \{cl\}$;
11. $\Pi_{rel} := \{\neg\alpha\}$; $\Pi'_{rel} := \emptyset$;
12. **while** $\Pi'_{rel} \neq \Pi_{rel}$ **do**
13. $\quad \Pi'_{rel} := \Pi_{rel}$;
14. $\quad$ **for** each clause $cl \in \Pi_{in} \setminus \Pi_0$ such that $(cl^+ \cup cl^-) \cap \mathsf{atoms}(\Pi_{rel}) \neq \emptyset$ **do**
15. $\quad\quad \Pi_{rel} := \Pi_{rel} \cup \{cl\}$;
16. **return** Π_{rel};

Fig. 2. The algorithm for extracting a just-preserving relevant subset of $\Pi_{in} \cup \{\neg\alpha\}$ for α

Consider $\mathrm{Extract}(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)), X, \mathsf{Person}(a_1))$ for $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$ given in Example 3. It can be checked that the values of Π_{ur1}, Π_{ur2} (i.e. $\Pi_0 \setminus \Pi_{ur1}$ in the algorithm) and Π_{rel} are exactly those given in Example 4. By Lemma 4, Π_{rel} is a just-preserving subset of Π for $\mathsf{Person}(a_1)$, thus $\{ax' \in \mathcal{O}^\dagger \mid \hbar_{ax'} \in \mathsf{atoms}(\Pi_{rel})\} = \{ax_1, ax_6\}$ is a just-preserving module of $\mathcal{O}^\dagger$ for $\mathsf{Person}(a_1)$. Note that $\mathsf{ChiefActress}(a_1) \in \mathcal{O}^\dagger \setminus \mathcal{O}$,

so $\{ax_1\}$ is a just-preserving module of $\mathcal{O}$ for $\mathsf{ChiefActress} \sqsubseteq \mathsf{Person}$. Recall that the syntactic locality-based module of $\mathsf{ChiefActress}$ in $\mathcal{O}$ is exactly $\mathcal{O}$. This shows that the just-preserving module extracted by our goal-directed method can be much smaller than the syntactic locality-based module.

The following theorem shows the correctness and complexity of the algorithm.

Theorem 2. $\mathrm{Extract}(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)), X, ax)$ *returns a just-preserving relevant subset of* $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)) \cup \{\neg ax\}$ *for* ax *in time polynomial in* $|\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))|$.

It follows from the above theorem that $\{ax' \in \mathcal{O}^\dagger \mid \hbar_{ax'} \in \mathsf{atoms}(\Pi_{rel})\}$ is a just-preserving module of $\mathcal{O}^\dagger$ for ax, where Π_{rel} is the result of $\mathrm{Extract}(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)), X, ax)$. The remaining issue for extracting just-preserving modules is that $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$ can be infinite when there are function symbols occurring in $\mathcal{D}(\mathcal{O}^\dagger)$. We in the next subsection show that a just-preserving module of $\mathcal{O}^\dagger$ for ax can also be extracted from a finite variant of $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$ whose size is polynomial in the size of $\mathcal{O}$.

4.4 Approximate Grounding of the Diagnosing Program

To tackle the problem that the bottom-up grounding Π of a first-order logic program can be infinite, we consider a mapping function on ground terms occurring in Π such that the range of this function is finite and small. We call a mapping function $\lambda : \mathrm{terms}(\Pi) \mapsto \mathrm{terms}(\Pi)$, where $\mathrm{terms}(\Pi)$ is the set of ground terms occurring in Π, a *convergent mapping function* for Π, if 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 for every equational atom $s \approx t$ occurring in Π, $\lambda(s) = \lambda(t)$. For example, given a propositional program $\Pi^\dagger = \{\neg\mathsf{hasFather}(a,b) \vee \neg\mathsf{hasFather}(a, f(a)) \vee b \approx f(a), \neg\mathsf{Person}(a) \vee \mathsf{hasFather}(a, f(a)), \neg\mathsf{Person}(a) \vee \mathsf{Person}(f(a))\}$, the mapping function $\lambda = \{a \mapsto a, b \mapsto a, f(a) \mapsto a\}$ is a convergent mapping function for $\Pi^\dagger$, but the mapping function $\lambda' = \{a \mapsto b, b \mapsto a, f(a) \mapsto a\}$ is not since $\lambda'(f(a)) \neq \lambda'(a)$.

Given a mapping function λ on ground terms, by $\lambda(\alpha)$, $\lambda(cl)$, $\lambda(\mathcal{A})$ and $\lambda(\Pi)$ we respectively denote the results obtained from a ground atom α, a clause cl, a set $\mathcal{A}$ of ground atoms and a propositional program Π by replacing every ground term t occurring in it with $\lambda(t)$. Take $\Pi^\dagger$ given above for example, for the mapping function $\lambda = \{a \mapsto a, b \mapsto a, f(a) \mapsto a\}$, $\lambda(\Pi^\dagger) = \{\neg\mathsf{hasFather}(a,a) \vee \neg\mathsf{hasFather}(a,a) \vee a \approx a, \neg\mathsf{Person}(a) \vee \mathsf{hasFather}(a,a), \neg\mathsf{Person}(a) \vee \mathsf{Person}(a)\}$.

It is obvious that for any convergent mapping function λ for Π, $\lambda(\Pi)$ is finite when Π is infinite but the number of constants and predicate symbols occurring in Π is finite. Even when Π is finite and does not contain function symbols, $\lambda(\Pi)$ can be much smaller than Π because the subset of ground clauses in Π that form a congruence relation is collapsed in $\lambda(\Pi)$.

Lemma 5. *Let* λ *be an convergent mapping function for* $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$ *where decision atoms, treated as nullary atoms, are not in the domain of* λ, Π *be a superset of* $\lambda(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)))$, *and* Π_{rel} *be returned by* $\mathrm{Extract}(\Pi, X, \lambda(ax))$. *Then* $\{ax' \in \mathcal{O}^\dagger \mid \hbar_{ax'} \in \mathsf{atoms}(\Pi_{rel})\}$ *is a just-preserving module of* $\mathcal{O}^\dagger$ *for* ax.

To employ the above lemma to extract a just-preserving module of $\mathcal{O}^\dagger$ for ax, we develop an algorithm, shown in Figure 3, to compute a superset of $\lambda(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)))$ for some convergent mapping function λ for $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$.

Algorithm 2. $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$

Input: An $\mathcal{SHOIQ}$ ontology $\mathcal{O}^\dagger$.

Output: A propositional program and a set of sets of constants.

1. Let P be obtained from $\mathcal{D}(\mathcal{O}^\dagger)$ by stripping all function symbols from functional terms;
2. $\Pi := \emptyset$; $\mathcal{S} := \emptyset$;
3. **repeat**
4. $\quad \Pi' := \Pi$;
5. $\quad$ **for** each clause $cl \in P$ and each ground substitution σ such that $cl^-\sigma \subseteq \mathsf{atoms}(\Pi) \cap \mathrm{HB}(P)$ and $cl^+\sigma \subseteq \mathrm{HB}(P)$ **do**
6. $\quad\quad cl_{inst} := cl\,\sigma$;
7. $\quad\quad$ **for** each equational atom $a \approx b \in cl^+_{inst}$ such that a and b are different constants **do**
8. $\quad\quad\quad \mathrm{MergeConstants}(a, b, \mathcal{S}, cl_{inst}, P)$;
9. $\quad\quad \Pi := \Pi \cup \{cl_{inst}\}$;
10. **until** $\Pi' = \Pi$;
11. $\Pi := \{cl \in \Pi \mid cl^+ \cup cl^- \subseteq \mathrm{HB}(P)\}$;
12. **return** $(\Pi, \mathcal{S})$;

Subprocedure $\mathrm{MergeConstants}(a, b, \mathcal{S}, cl, P)$

1. Let $\mathcal{C}_a$ be $\{a\}$ if a does not occur in $\mathcal{S}$, or the congruence class in $\mathcal{S}$ where a belongs to;
2. Let $\mathcal{C}_b$ be $\{b\}$ if b does not occur in $\mathcal{S}$, or the congruence class in $\mathcal{S}$ where b belongs to;
3. $\mathcal{C}_{new} := \mathcal{C}_a \cup \mathcal{C}_b$; $\mathsf{rep}(\mathcal{C}_{new}) := b$; $\mathcal{S} := (\mathcal{S} \setminus \{\mathcal{C}_a, \mathcal{C}_b\}) \cup \mathcal{C}_{new}$;
4. Update cl and P by replacing every occurrence of a with b;

Fig. 3. The algorithm for approximately grounding the diagnosing program $\mathcal{D}(\mathcal{O}^\dagger)$

In the algorithm, P is a first-order logic program obtained from $\mathcal{D}(\mathcal{O}^\dagger)$ by stripping all function symbols from functional terms, i.e., by replacing every functional term $f_1(...f_n(t))$ with t where t is a variable or a constant; $\mathrm{HB}(P)$ is the Herbrand base of P, which does not contain any function symbol and is changed whenever P is changed; $\mathcal{S}$ is the set of sets of constants such that for any constant a in any element $\mathcal{C} \in \mathcal{S}$, there exists a constant $b \in \mathcal{C}$ such that the equational atom $a \approx b$ appears in the execution of the algorithm. We call an element in $\mathcal{S}$ a *congruence class*. Each congruence class $\mathcal{C}$ is associated with a *representative constant*, denoted by $\mathsf{rep}(\mathcal{C})$, which is an element of $\mathcal{C}$.

The algorithm does not directly compute a convergent mapping function λ for $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$, because such mapping function can be constructed from $\mathcal{S}$. By $\mathsf{map}(t, \mathcal{S})$ we denote the function for mapping a term $t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))$ to a constant occurring in $\mathcal{D}(\mathcal{O}^\dagger)$ based on $\mathcal{S}$, recursively defined as follows, where a is a constant.

- $\mathsf{map}(f_1(...f_n(a)), \mathcal{S}) = \mathsf{map}(a, \mathcal{S})$, where $n > 0$;
- $\mathsf{map}(a, \mathcal{S}) = b$, where $b = \mathsf{rep}(\mathcal{C})$ if $a \in \mathcal{C}$ for some $\mathcal{C} \in \mathcal{S}$, or $b = a$ otherwise.

We naturally extend the function map to (sets of) ground atoms and ground clauses, i.e., by $\mathsf{map}(\alpha, \mathcal{S})$, $\mathsf{map}(\mathcal{A}, \mathcal{S})$ and $\mathsf{map}(cl, \mathcal{S})$ we respectively denote the results obtained from a ground atom α, a set $\mathcal{A}$ of ground atoms, and a clause cl by replacing every ground term t occurring in it with $\mathsf{map}(t, \mathcal{S})$.

We call a mapping function $\lambda : \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger)) \mapsto \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))$ *induced from* the function map w.r.t. $\mathcal{S}$ if $\lambda(t) = \mathsf{map}(t, \mathcal{S})$ for all ground terms $t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))$. One goal of the algorithm is to ensure the mapping function λ induced from map w.r.t. $\mathcal{S}$ to be an convergent mapping function for $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$, i.e., ensure $\mathsf{map}(s, \mathcal{S}) =$

$\mathsf{map}(t, \mathcal{S})$ for all equational atoms $s \approx t$ occurring positively in $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$. Another goal of the algorithm is to return a superset Π of $\lambda(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)))$. Consider how to achieve this goal. By Lemma 3, $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$ is the least subset S of $\mathcal{G}(\mathcal{D}(\mathcal{O}^\dagger))$ such that $cl\,\sigma \in S$ for any clause $cl \in \mathcal{D}(\mathcal{O}^\dagger)$ and any ground substitution σ such that $cl^-\sigma \subseteq \mathsf{atoms}(S)$ and $cl^+\sigma \subseteq \mathrm{HB}(\mathcal{D}(\mathcal{O}^\dagger))$. It follows that $\lambda(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger)))$ is the least subset S' of $\lambda(\mathcal{G}(\mathcal{D}(\mathcal{O}^\dagger)))$ such that $\lambda(cl\,\sigma) \in S'$ for any clause $cl \in \mathcal{D}(\mathcal{O}^\dagger)$ and any ground substitution σ such that $\lambda(cl^-\sigma) \subseteq \mathsf{atoms}(S')$ and $\lambda(cl^+\sigma) \subseteq \lambda(\mathrm{HB}(\mathcal{D}(\mathcal{O}^\dagger)))$. If Π is a subset of $\lambda(\mathcal{G}(\mathcal{D}(\mathcal{O}^\dagger)))$ such that $\lambda(cl\,\sigma) \in \Pi$ for any clause $cl \in \mathcal{D}(\mathcal{O}^\dagger)$ and any ground substitution σ such that $\lambda(cl^-\sigma) \subseteq \mathsf{atoms}(\Pi)$ and $\lambda(cl^+\sigma) \subseteq \lambda(\mathrm{HB}(\mathcal{D}(\mathcal{O}^\dagger)))$, then we have $\lambda(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))) \subseteq \Pi$. Hence we refine the second goal to the goal of finding the above subset Π.

In the following descriptions, we use λ to denote a mapping function induced from map w.r.t. $\mathcal{S}$. At the beginning (before line 3), it is clear that $\mathrm{HU}(P) = \{\mathsf{map}(t, \mathcal{S}) \mid t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))\}$. Hence we can use $\mathrm{HU}(P)$ to represent $\{\mathsf{map}(t, \mathcal{S}) \mid t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))\}$ in the algorithm. To this end, we should ensure $\{\mathsf{map}(t, \mathcal{S}) \mid t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))\} = \mathrm{HU}(P)$ throughout the algorithm.

We now describe the main part of the algorithm. All clauses in P are instantiated iteratively until a fixpoint is reached (lines 3–10), making the instantiated set Π satisfy (†) $\lambda(cl\,\sigma) \in \Pi$ for any clause $cl \in \mathcal{D}(\mathcal{O}^\dagger)$ and any ground substitution σ s.t. $\lambda(cl^-\sigma) \subseteq \mathsf{atoms}(\Pi)$ and $\lambda(cl^+\sigma) \subseteq \lambda(\mathrm{HB}(\mathcal{D}(\mathcal{O}^\dagger)))$. Consider any clause $cl \in \mathcal{D}(\mathcal{O}^\dagger)$ and any ground substitution σ. Let cl' be obtained from cl by stripping all function symbols from functional terms and by replacing every occurrence of constant a with $\lambda(a)$, and σ' be obtained from σ by replacing each mapping $x \mapsto t$ with $x \mapsto \mathsf{map}(t, \mathcal{S})$. Then $cl' \in P$ and $\lambda(cl\,\sigma) = \lambda(cl'\sigma')$. Hence we only need to consider adding $\lambda(cl\,\sigma)$ to Π for every clause $cl \in P$ and every ground substitution σ such that every ground term occurring in $cl\,\sigma$ also occurs in $\{\mathsf{map}(t, \mathcal{S}) \mid t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))\} = \mathrm{HU}(P)$. Note that $\lambda(cl\,\sigma) = cl\,\sigma$ when every ground term occurring in $cl\,\sigma$ also occurs in $\mathrm{HU}(P)$ and that $\mathrm{HB}(P) = \lambda(\mathrm{HB}(\mathcal{D}(\mathcal{O}^\dagger)))$, so in order to satisfy the condition (†), every clause $cl \in P$ and every ground substitution σ such that $cl^-\sigma \subseteq \mathsf{atoms}(\Pi) \cap \mathrm{HB}(P)$ and $cl^+\sigma \subseteq \mathrm{HB}(P)$ are handled as follows (lines 5–9). Let $cl_{inst} = cl\,\sigma$. Every ground term $a \approx b \in cl^+_{inst}$ such that a and b are different constants is handled by merging a and b to the same congruence class in $\mathcal{S}$ and by updating cl_{inst} and P accordingly (see the subprocedure $\mathrm{MergeConstants}$). The merge of a and b is to ensure $\mathsf{map}(a, \mathcal{S}) = \mathsf{map}(b, \mathcal{S})$. The update of cl_{inst} is to ensure $\mathsf{map}(cl_{inst}, \mathcal{S}) = cl_{inst}$. The update of P is to ensure again $\{\mathsf{map}(t, \mathcal{S}) \mid t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))\} = \mathrm{HU}(P)$. Since $\mathcal{S}$ is only updated in the subprocedure $\mathrm{MergeConstants}$, $\{\mathsf{map}(t, \mathcal{S}) \mid t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))\} = \mathrm{HU}(P)$ holds throughout the algorithm. After every ground term $a \approx b \in cl^+_{inst}$ is handled, the possibly updated cl_{inst} is added to Π. After Π reaches a fixpoint, since Π should be a subset of $\lambda(\mathcal{G}(\mathcal{D}(\mathcal{O}^\dagger)))$, every ground term occurring in Π should be in $\{\lambda(t) \mid t \in \mathrm{HU}(\mathcal{D}(\mathcal{O}^\dagger))\} = \mathrm{HU}(P)$, so those clauses in Π that contain ground atoms not in $\mathrm{HB}(P)$ are removed from Π (line 11). Since the propositional program Π returned by the algorithm satisfies the above condition (†), we have the following lemma.

Lemma 6. *Let $(\Pi, \mathcal{S})$ be returned by $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$ and λ be a mapping function induced from the function map w.r.t. $\mathcal{S}$. Then (1) $\lambda(\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))) \subseteq \Pi$ and λ is a convergent mapping function for $\mathcal{G}_{bu}(\mathcal{D}(\mathcal{O}^\dagger))$; (2) $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$ works in time*

polynomial in s^m and $|\Pi|$ is polynomial in s, where m is the maximum number in all qualifying number restrictions in $\mathcal{O}$ and s is the size of $\mathcal{O}$.

The following theorem shows the correctness of our proposed method for extracting a just-preserving module, which is immediately follows from Lemma 5 and Lemma 6.

Theorem 3. *Let $(\Pi, \mathcal{S})$ be returned by $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$, and Π_{rel} be returned by $\mathrm{Extract}(\Pi, X, \mathsf{map}(ax, \mathcal{S}))$. Then $\{ax' \in \mathcal{O}^\dagger \mid \hbar_{ax'} \in \mathsf{atoms}(\Pi_{rel})\}$ is a just-preserving module of $\mathcal{O}^\dagger$ for ax.*

To summarize, our proposed method works as follows. In the first and offline phase, we first compute $\mathcal{O}^\dagger = \mathcal{O} \cup \{A(a_A) \mid A$ is an atomic concept in $\mathcal{O}$ and a_A is a new globally unique individual corresponding to $A\}$, then compute $(\Pi, \mathcal{S})$ as the result of $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$. In the second and online phase, for every coming entailment ax, we first set $\alpha = B(a_A)$ if the given entailment ax is $A \sqsubseteq B$, or $\alpha = ax$ otherwise, then compute $\Pi_{rel} = \mathrm{Extract}(\Pi, X, \mathsf{map}(\alpha, \mathcal{S}))$, and finally compute a just-preserving module of $\mathcal{O}$ for ax as $\{ax' \in \mathcal{O}^\dagger \mid \hbar_{ax'} \in \mathsf{atoms}(\Pi_{rel})\} \cap \mathcal{O}$.

5 Experimental Evaluation

We implemented the proposed method in GNU C++, using MySQL as the back-end SQL engine. In the implementation, ABox axioms are maintained in databases, instantiated clauses in the course of grounding are maintained in disk files, and Pellet [19] (version 2.0.0-rc6)[7] is called to find all justifications of a given entailment. All our experiments were conducted on a machine with 2.33GHz Intel Xeon E5410 CPU and 8GB RAM, running Windows Server 2003, where the maximum Java heap size was set to (max) 1252MB. The implementation, test sets and complete test results are available at `http://www.aifb.uni-karlsruhe.de/WBS/gqi/jp-module/`.

5.1 Experimental Setup

We conducted experiments on the GALEN Medical Knowledge Base[8], the Gene Ontology (GO)[9], the US National Cancer Institute Ontology (NCI)[10], as well as Lehigh University Benchmark (LUBM) [10] and University Ontology Benchmark (OWL Lite version) [14] (UOBM-Lite) ontologies. By LUBMn and UOBM-Lite$n$ we respectively denote the instances of LUBM and UOBM-Lite that contain axioms about n universities. We specifically tested on LUBM1, LUBM10, UOBM-Lite1 and UOBM-Lite10, where the former two were generated by the LUBM data generator[11], and the latter two were all downloaded from the UOBM Website[12].

Before testing our approach we stored ABoxes to MySQL databases. Table 1 lists the characteristics of the seven test ontologies.

[7] Pellet (`http://clarkparsia.com/pellet/`) employs a hitting set tree (HST) based algorithm [12] to find all justifications.

[8] `http://www.openclinical.org/prj_galen.html`

[9] `http://www.geneontology.org`

[10] `http://www.mindswap.org/2003/CancerOntology/nciOntology.owl`

[11] `http://swat.cse.lehigh.edu/projects/lubm/index.htm`

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

Table 1. The characteristics of test ontologies and the execution time in the offline phase

$\mathcal{O}$	$\lvert N_C\rvert$	$\lvert N_R\rvert$	$\lvert N_I\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{A}\rvert$	Offline time(sec)
GALEN	2,748	412	0	4,529	0	1,431
GO	20,465	1	0	28,897	0	7,519
NCI	27,652	70	0	46,940	0	10,901
LUBM1	59	16	50,253	94	100,543	9
LUBM10	59	16	629,568	94	1,272,575	116
UOBM-Lite1	51	43	95,010	130	245,864	62
UOBM-Lite10	51	43	820,208	130	2,096,973	679

Note: $\mathcal{O} = (\mathcal{T}, \mathcal{A})$ is a test ontology. N_C, N_R and N_I are respectively the sets of concept names, role names and individuals in $\mathcal{O}$.

Table 2. The test results on finding all justifications of a selected entailment

	Module Extr. by Our Method			Syn. Locality-based Module			# $\mathrm{SH_{LB}}$	# $\mathrm{Sz_{JP}}$
	#SH	$\mathrm{SHT_{avg}}$(sec)	$\mathrm{Size_{avg}}$	#SH	$\mathrm{SHT_{avg}}$(sec)	$\mathrm{Size_{avg}}$	$\setminus \mathrm{SH_{JP}}$	$< \mathrm{Sz_{LB}}$
GALEN	40	3.555	69.75	40	3.814	134.78	0	40
GO	40	7.314	9.55	40	11.985	32.25	0	40
NCI	40	4.065	7.23	40	7.518	70.95	0	40
LUBM1	40	69.061	22.15	20	201.481	100,596.00	0	40
LUBM10	40	95.721	20.48	0	$\mathrm{MO_1}$	1,272,615.00	0	40
UOBM-Lite1	16	24.813	897.80	11	155.220	245,966.00	0	40
UOBM-Lite10	15	32.278	799.83	0	$\mathrm{MO_2}$	2,097,047.00	0	40

Note: "#SH" is the number of selected entailments that are *successfully* handled, i.e., all justifications of the entailment can be computed over the extracted module without running out of time/memory. "$\mathrm{SHT_{avg}}$" is the average execution time for finding all justifications of each selected entailment that is successfully handled. "$\mathrm{Size_{avg}}$" is the average number of axioms in each extracted module (counting all 40 selected entailments). "$\#\mathrm{SH_{LB}} \setminus \mathrm{SH_{JP}}$" is the number of selected entailments that are successfully handled in syntactic locality-based modules but not successfully handled in modules extracted by our method. "$\#\mathrm{Sz_{JP}} < \mathrm{Sz_{LB}}$" is the number of selected entailments whose module extracted by our method is smaller than the corresponding syntactic locality-based module. "$\mathrm{MO_1}$" (resp. "$\mathrm{MO_2}$") means that all runs are out of memory when finding justifications (resp. when loading modules).

In our experiments, we randomly selected 40 subsumption entailments for each of GALEN, GO and NCI, and randomly selected 40 membership entailments over concept names for each of LUBM1, LUBM10, UOBM-Lite1 and UOBM-Lite10. For each selected subsumption, we extracted the just-preserving module introduced in this paper; for comparison, we also extracted the corresponding syntactic locality-based module. In particular, for each selected subsumption entailment $A \sqsubseteq B$, we extracted the syntactic locality-based module for A; for each selected membership entailment $A(a)$, we extracted the syntactic locality-based module for $\{a\}$. We set a time limit of 1000 seconds for Pellet to find all justifications of a selected entailment.

5.2 Experimental Results

Regarding the offline phase of our method, the execution time (in seconds) in this phase is shown in the last column of Table 1. Though the offline phase is rather costly, it

is reasonable due to the following reasons. First, the offline phase is independent of any given subsumption/membership entailment; i.e., once it is executed, the extraction of just-preserving modules for any given subsumption/membership entailment can be done without executing it again. This mechanism is suitable for the situation where the given ontology is stable, which can happen because users may want to know why an unwanted/surprising entailment holds when they get some implicit results from ontology reasoning. Second, the offline phase is theoretically tractable. More precisely, it works in time polynomial in the size of the given ontology under the assumption that numbers in qualifying number restrictions are bounded by some constant (see Lemma 6).

Regarding the online phase of our method, the test results are reported in Table 2. For comparison, Table 2 also shows the results for applying Pellet to find all justifications over syntactic locality-based modules. We need to point out that the execution time for finding all justifications over a module extracted by our method includes the module extraction time, but the execution time for finding all justifications over a syntactic locality-based module excludes the module extraction time as we assume that all syntactic locality-based modules are extracted offline.

The test results have some implications. First, a module extracted by our method is smaller than the corresponding syntactic locality-based module; esp. for membership entailments, by orders of magnitude smaller. Second, finding all justifications over modules extracted by our method is generally more efficient than finding all justifications over syntactic locality-based modules. The efficiency improvement is rather small for test ontologies with large TBoxes because most of the selected entailments have only a small justification (which results in a small syntactic locality-based module) and the module extraction time is only included in the former case. Third, finding all justifications over modules extracted by our method is much more scalable than over syntactic locality-based modules against increasing number of ABox axioms. In particular, a good number of membership entailments are handled efficiently in modules that are extracted from a test ontology with millions of ABox axioms using our method. It seems that the size of a module extracted by our method does not depend on the number of ABox axioms, but mainly depends on the TBox complexity.

6 Conclusion and Future Work

In this paper, we have proposed a goal-directed method for extracting a just-preserving module for a given entailment. The basic idea of the method is to first extract a relevant subset of a finite propositional program compiled from the given ontology, then identify a just-preserving module for the given entailment from the extracted subset. Experimental results on large ontologies show that a module extracted by our method is smaller than the corresponding syntactic locality-based module, improving the performance of the subsequent computation of all justifications. For future work, we plan to adapt the method to optimize finding all justifications of the inconsistency/incoherence of an OWL DL ontology, and upgrade the compilation method to an incremental one to cope with ontology changes.

Acknowledgments. Thanks all anonymous reviewers for their useful comments. Jianfeng Du is supported in part by NSFC grants 60673103 and 70801020. Guilin Qi is supported in part by the EU under the IST project NeOn (IST-2006-027595).

References

1. F. Baader, D. Calvanese, D. L. McGuinness, D. Nardi, and P. F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.
2. F. Baader and B. Suntisrivaraporn. Debugging SNOMED CT using axiom pinpointing in the description logic $\mathcal{EL}^+$. In *Proc. of KR-MED'08*, 2008.
3. A. Biere and C. Sinz. Decomposing SAT problems into connected components. *Journal on Satisfiability, Boolean Modeling and Computation*, 2:201–208, 2006.
4. P. Doran, V. A. M. Tamma, and L. Iannone. Ontology module extraction for ontology reuse: an ontology engineering perspective. In *Proc. of CIKM'07*, pages 61–70, 2007.
5. T. Eiter, N. Leone, C. Mateis, G. Pfeifer, and F. Scarcello. A deductive system for nonmonotonic reasoning. In *Proc. of LPNMR'97*, pages 364–375, 1997.
6. M. Fitting. *First-order Logic and Automated Theorem Proving (2nd ed.)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1996.
7. B. Cuenca Grau, C. Halaschek-Wiener, and Y. Kazakov. History matters: Incremental ontology reasoning using modules. In *Proc. of ISWC'07*, pages 183–196, 2007.
8. B. Cuenca Grau, I. Horrocks, Y. Kazakov, and U. Sattler. Just the right amount: Extracting modules from ontologies. In *Proc. of WWW'07*, pages 717–726, 2007.
9. B. Cuenca Grau, B. Parsia, E. Sirin, and A. Kalyanpur. Modularity and web ontologies. In *Proc. of KR'06*, pages 198–209, 2006.
10. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics*, 3(2–3):158–182, 2005.
11. U. Hustadt, B. Motik, and U. Sattler. Reasoning in description logics by a reduction to disjunctive datalog. *Journal of Automated Reasoning*, 39(3):351–384, 2007.
12. A. Kalyanpur, B. Parsia, M. Horridge, and E. Sirin. Finding all justifications of OWL DL entailments. In *Proc. of ISWC'07*, pages 267–280, 2007.
13. Y. Kazakov and B. Motik. A resolution-based decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning*, 40(2-3):89–116, 2008.
14. L. Ma, Y. Yang, Z. Qiu, G. Xie, Y. Pan, and S. Liu. Towards a complete OWL ontology benchmark. In *Proc. of ESWC'06*, pages 125–139, 2006.
15. B. Motik, U. Sattler, and R. Studer. Query answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.
16. N. F. Noy and M. A. Musen. Specifying ontology views by traversal. In *Proc. of ISWC'04*, pages 713–725, 2004.
17. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proc. of IJCAI'03*, pages 355–362, 2003.
18. J. Seidenberg and A. L. Rector. Web ontology segmentation: Analysis, classification and use. In *Proc. of WWW'06*, pages 13–22, 2006.
19. E. Sirin, B. Parsia, B. Cuenca Grau, A. Kalyanpur, and Y. Katz. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics*, 5(2):51–53, 2007.
20. H. Stuckenschmidt and M. Klein. Structure-based partitioning of large concept hierarchies. In *Proc. of ISWC'04*, pages 289–303, 2004.
21. B. Suntisrivaraporn. Module extraction and incremental classification: A pragmatic approach for $\mathcal{EL}^+$ ontologies. In *Proc. of ESWC'08*, pages 230–244, 2008.
22. B. Suntisrivaraporn, G. Qi, Q. Ji, and P. Haase. A modularization-based approach to finding all justifications for owl dl entailments. In *Proc. of ASWC'08*, pages 1–15, 2008.