A Decomposition-Based Approach to Optimizing Conjunctive Query Answering in OWL DL

Jianfeng Du[1,2], Guilin Qi[3,4], Jeff Z. Pan[5], and Yi-Dong Shen[2]

[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,ydshen}@ios.ac.cn
[3] AIFB, Universität Karlsruhe, D-76128 Karlsruhe, Germany
[4] School of Computer Science and Engineering, Southeast University, Nanjing, China
[5] Department of Computing Science, The University of Aberdeen, Aberdeen AB24 3UE, UK

Abstract. Scalable query answering over Description Logic (DL) based ontologies plays an important role for the success of the Semantic Web. Towards tackling the scalability problem, we propose a decomposition-based approach to optimizing existing OWL DL reasoners in evaluating conjunctive queries in OWL DL ontologies. The main idea is to decompose a given OWL DL ontology into a set of target ontologies without duplicated ABox axioms so that the evaluation of a given conjunctive query can be separately performed in every target ontology by applying existing OWL DL reasoners. This approach guarantees sound and complete results for the category of conjunctive queries that the applied OWL DL reasoner correctly evaluates. Experimental results on large benchmark ontologies and benchmark queries show that the proposed approach can significantly improve scalability and efficiency in evaluating general conjunctive queries.

1 Introduction

Scalable query answering over Description Logic (DL) based ontologies plays an important role for the success of the Semantic Web (SW). On the one hand, the W3C organization proposed the standard Web Ontology Language (OWL)[1] to represent ontologies in the SW, which is based on DLs and provides shared vocabularies for different domains. On the other hand, ontology query engines are expected to be scalable enough to handle the increasing semantic data that the Web provides.

OWL DL is the most expressive species in the OWL family that is decidable in terms of consistency checking. Though the decidability of conjunctive query answering in OWL DL is still an open problem, many OWL DL reasoners implement decision procedures for some categories of conjunctive queries (CQs) for which decidability is known, e.g., for CQs that have a kind of tree-shape or CQs that do not contain non-distinguished variables (i.e. existentially quantified variables). To name a few, Pellet [14] is a well-known OWL DL reasoner that supports general CQs that have a kind of tree-shape

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

A. Bernstein et al. (Eds.): ISWC 2009, LNCS 5823, pp. 146–162, 2009.

(i.e. do not contain cycles through non-distinguished variables); KAON2 [11] is another well-known OWL DL reasoner that supports CQs without non-distinguished variables. These reasoners still suffer from the scalability problem and call for optimizations to make them scale to larger ABoxes or more complex TBoxes.

To make existing OWL DL reasoners more scalable, we propose a decomposition-based approach to optimizing conjunctive query answering in OWL DL (see Section 4). Basically, the approach computes explicit answers (i.e. facts that satisfy the given CQ) first and then identifies candidate answers and target ontologies that are sufficient for checking whether candidate answers are indeed answers to the query. Different target ontologies have no common ABox axioms but may have common TBox axioms. The verification of whether a candidate answer is an answer is delegated to an existing OWL DL reasoner. This approach guarantees sound and complete results for the categories of CQs that the OWL DL reasoner correctly evaluates. For the categories of CQs that the OWL DL reasoner cannot handle, this approach still returns all candidates and results in an unsound but complete evaluation.

We implement the proposed approach and conduct experiments on LUBM [8] and UOBM [10] ontologies (see Section 5). Experimental results on all benchmark CQs given in [8,10] show that the proposed approach can significantly improve scalability and efficiency in evaluating general CQs.

Related Work. There are approaches to conjunctive query answering that have certain contributions to the scalability problem. Motik *et al.* [12] propose a resolution-based approach, implemented in KAON2 [11], to evaluating CQs without non-distinguished variables. This approach reduces the problem of conjunctive query answering to the problem of reasoning in disjunctive datalog programs; the latter problem has more scalable solutions for handling large ABoxes. Currently KAON2 does not support nominals, which are allowed in OWL DL. Dolby *et al.* [2] propose a summarization and refinement approach to instance retrieval, which is later adapted to evaluating CQs without non-distinguished variables by adding some optimizations for retrieving role instances [3]. This approach improves scalability because it works on a summarization of the ABox, but it does not support nominals either. Pan and Thomas [13] propose a semantic approximation approach to OWL DL. The approach converts an OWL DL ontology to a DL-Lite [1] ontology, which allows CQs to be evaluated in polynomial time. The above approaches, however, do not support or may not correctly evaluate CQs with non-distinguished variables.

The idea of decomposition has been exploited in managing large ontologies. The result of decomposing an ontology is usually a set of subsets of axioms in the ontology. Stuckenschmidt and Klein [15] propose a method for decomposing all concepts in an ontology to facilitate visualization of the ontology. This method does not concern ontology reasoning. Cuenca Grau *et al.* [6] propose a method for decomposing an ontology into a set of modules such that all inferences about the signature contained in a module can be made locally. The method focuses on TBoxes and does not concern conjunctive query answering. Guo and Heflin [7] propose a method for decomposing an ABox into possibly overlapped subsets. Only instance retrieval of atomic concepts/roles can be correctly performed in separate resulting subsets together with the whole TBox. Compared with the above methods, the primary distinction of our proposed approach is

that it yields target ontologies without duplicated ABox axioms and ensures conjunctive query answering to be correctly performed in separate target ontologies.

2 Preliminaries

OWL DL and Conjunctive Query Answering. OWL DL corresponds to DL $\mathcal{SHOIN}$. 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 $\mathcal{O} = (\mathcal{O}_\mathcal{T}, \mathcal{O}_\mathcal{A})$ 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* $\mathsf{Trans}(R)$ and *role inclusion axioms* $R \sqsubseteq S$, where C and D are OWL DL concepts, and R and S roles. The ABox $\mathcal{O}_\mathcal{A}$ consists of a finite set of *concept assertions* $C(a)$, *role assertions* $R(a, b)$, *equality assertions* $a \approx b$ and *inequality assertions* $a \not\approx b$, where C is an OWL DL concept, R a role, 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 *consistent* or *satisfiable* iff it has a model.

A conjunctive query (CQ) is of the form $q(\overrightarrow{x}) \leftarrow \exists \overrightarrow{y}.\mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ or simply $q(\overrightarrow{x}) \leftarrow \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$, where $q(\overrightarrow{x})$ is the *head* of q, $\mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ is the *body* of q, $\overrightarrow{x}$ are *distinguished variables*, $\overrightarrow{y}$ are *non-distinguished variables*, $\overrightarrow{c}$ are individuals, and $\mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ is a conjunction of atoms of the form $A(v)$ or $R(v_1, v_2)$ for A an atomic concept, R an atomic role, and v, v_1 and v_2 variables in $\overrightarrow{x}$ and $\overrightarrow{y}$ or individuals in $\overrightarrow{c}$. Here allowing only atomic concepts/roles is not a big issue in practice as querying against named relations is usual when people query over relational databases [13]. A CQ is called a *Boolean conjunctive query* (BCQ) if it has no distinguished variables.

A tuple $\overrightarrow{t}$ of individuals in an ontology $\mathcal{O}$ is called an *answer* of $q(\overrightarrow{x})$ in $\mathcal{O}$, denoted by $\mathcal{O} \models q[\overrightarrow{x} \mapsto \overrightarrow{t}]$, if every model of $\mathcal{O}$ satisfies $q[\overrightarrow{x} \mapsto \overrightarrow{t}]$, i.e. the body of q with every variable in $\overrightarrow{x}$ substituted by its corresponding individual in $\overrightarrow{t}$. A BCQ $q() \leftarrow \mathsf{conj}(\overrightarrow{y}, \overrightarrow{c})$ is said to have an answer $\langle\rangle$ in $\mathcal{O}$ if $\mathcal{O} \models q[\langle\rangle \mapsto \langle\rangle]$ (simply denoted by $\mathcal{O} \models q$). The problem of evaluating a CQ in $\mathcal{O}$, i.e. computing all answers of the CQ in $\mathcal{O}$, is called a problem of *conjunctive query answering*.

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 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* 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 $|cl^-| = 0$, and said to be *definite* if $|cl^+| = 1$.

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 *satisfiable* iff it has a model. Given a CQ $q(\overrightarrow{x}) \leftarrow \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$, a tuple $\overrightarrow{t}$ of constants is called an *answer* of $q(\overrightarrow{x})$ in P, denoted by $P \models q[\overrightarrow{x} \mapsto \overrightarrow{t}]$, if every model of P satisfies $q[\overrightarrow{x} \mapsto \overrightarrow{t}]$.

The first-order logic program P translated from a $\mathcal{SHOIN}$ 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 [5]. 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 $cl^- \subseteq M$, $cl^+ \cap M \neq \emptyset$.

3 The Proposed Decomposition-Based Approach

Throughout this section, by $\mathcal{O} = (\mathcal{O}_\mathcal{T}, \mathcal{O}_\mathcal{A})$ we denote a given OWL DL ontology. We assume that the given ontology $\mathcal{O}$ is consistent and treat $\mathcal{O}$ as a set of axioms.

3.1 The Basic Idea of the Proposed Approach

We use a BCQ $Q : q() \leftarrow p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \wedge \ldots \wedge p_n(\overrightarrow{y_n}, \overrightarrow{c_n})$ to show the basic idea of our approach. The approach to checking if $\mathcal{O} \models q$ consists of two phases.

In the first phase, we first translate $\mathcal{O}$ to a first-order logic program P such that $\mathcal{O} \models q$ iff $P \cup \{\neg p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \vee \ldots \vee \neg p_n(\overrightarrow{y_n}, \overrightarrow{c_n})\}$ is unsatisfiable (see Subsection 3.2), then consider transforming P to a proposition program Π such that $P \cup \{\neg p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \vee \ldots \vee \neg p_n(\overrightarrow{y_n}, \overrightarrow{c_n})\}$ is unsatisfiable iff $\Pi \cup \mathrm{Inst}_{\mathrm{BCQ}}(\Pi, Q) \cup \{\neg w_Q()\}$ is unsatisfiable, where w_Q a new predicate symbol corresponding to Q and not occurring in Π, and $\mathrm{Inst}_{\mathrm{BCQ}}(\Pi, Q)$ is a set of ground clauses instantiated from the clause $\neg p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \vee \ldots \vee \neg p_n(\overrightarrow{y_n}, \overrightarrow{c_n}) \vee w_Q()$ based on Π. We develop a basic method for extracting a target ontology $\mathcal{O}_{rel} \subseteq \mathcal{O}$ such that $\Pi \cup \mathrm{Inst}_{\mathrm{BCQ}}(\Pi, Q) \cup \{\neg w_Q()\}$ is unsatisfiable only if

$\mathcal{O}_{rel} \models q$ (note that this implies $\mathcal{O} \models q$ only if $\mathcal{O}_{rel} \models q$). Since Π may be infinite due to presence of function symbols in P, we instead transform P to a finite variant Π' of Π (see Subsection 3.3), such that the target ontology $\mathcal{O}'_{rel} \subseteq \mathcal{O}$ extracted from $\Pi' \cup \mathrm{Inst}^{\dagger}_{\mathrm{BCQ}}(\Pi', Q) \cup \{\neg w_Q()\}$ by using a similar method satisfies $\mathcal{O}_{rel} \subseteq \mathcal{O}'_{rel}$, where $\mathrm{Inst}^{\dagger}_{\mathrm{BCQ}}$ is a variant of $\mathrm{Inst}_{\mathrm{BCQ}}$. It should be noted that P and Π' are independent of any given query, so this phase (i.e., computing P and Π') can be performed offline.

In the second phase (see Subsection 3.4), we check if there exists a ground substitution σ such that $\{p_1(\overrightarrow{y_1}, \overrightarrow{c_1}), ..., p_n(\overrightarrow{y_n}, \overrightarrow{c_n})\}\sigma$ is a set of ground atoms occurring in definite ground facts in P. If such ground substitution exists, we conclude that $\mathcal{O} \models q$; otherwise, we extract the aforementioned target ontology $\mathcal{O}'_{rel}$ from $\Pi' \cup \mathrm{Inst}^{\dagger}_{\mathrm{BCQ}}(\Pi', Q) \cup \{\neg w_Q()\}$ and conclude that $\mathcal{O} \models q$ iff $\mathcal{O}'_{rel} \models q$.

From the above descriptions, we can see that our approach is correct, i.e., $\mathcal{O} \models q$ iff there is a ground substitution σ such that $\{p_1(\overrightarrow{y_1}, \overrightarrow{c_1}), ..., p_n(\overrightarrow{y_n}, \overrightarrow{c_n})\}\sigma$ is a set of ground atoms occurring in definite ground facts in P, or $\mathcal{O}'_{rel} \models q$.

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[2].

3.2 Translating to First-Order Logic

Since a direct translation from $\mathcal{SHOIN}$ to first-order clauses may incur exponential blowup [9], we apply the well-known *structural transformation* [11,9] to $\mathcal{O}$ before translating $\mathcal{O}$ to first-order clauses. By $\Theta(ax)$ we denote the result of applying structural transformation to an axiom ax, and by $\Theta(\mathcal{O})$ we denote $\bigcup_{ax \in \mathcal{O}} \Theta(ax)$. As structural transformation is well-known, we do not give its definition here but refer the reader to [11,9] or our technical report[2].

Throughout this section, we use $\mathcal{O}^{\dagger}$ to denote $\Theta(\mathcal{O})$ if not otherwise specified and treat $\mathcal{O}^{\dagger}$ as a set of axioms as well. By $\Xi(ax)$ we denote the result of translating an axiom ax in $\mathcal{O}^{\dagger}$ to a set of first-order clauses using the standard methods (see [9] or our technical report[2] for details). By $\Xi(\mathcal{O}^{\dagger})$ we denote $\bigcup_{ax \in \mathcal{O}^{\dagger}} \Xi(ax)$, and by $\Xi'(\mathcal{O}^{\dagger})$ we denote $\Xi(\mathcal{O}^{\dagger}) \cup \mathcal{E}(\Xi(\mathcal{O}^{\dagger}))$ if some equational atom $s \approx t$ occurs positively in $\Xi(\mathcal{O}^{\dagger})$, or $\Xi(\mathcal{O}^{\dagger})$ otherwise. Recall that $\mathcal{E}(\Xi(\mathcal{O}^{\dagger}))$ is used to axiomatize the equality predicate in $\Xi(\mathcal{O}^{\dagger})$ (see Section 2).

The following lemma shows that the problem of evaluating a BCQ in $\mathcal{O}$ can be reduced to a satisfiability problem about $\Xi'(\mathcal{O}^{\dagger})$. This lemma is similar to existing results given in [11,9].

Lemma 1. *For a BCQ* $q() \leftarrow p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \wedge ... \wedge p_n(\overrightarrow{y_n}, \overrightarrow{c_n})$ *in* $\mathcal{O}$, $\mathcal{O} \models q$ *iff* $\Xi'(\mathcal{O}^{\dagger}) \cup \{\neg p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \vee ... \vee \neg p_n(\overrightarrow{y_n}, \overrightarrow{c_n})\}$ *is unsatisfiable.*

Example 1. In our running example, we consider an ontology $\mathcal{O} = \{\mathsf{Man} \sqsubseteq \leq_1 \mathsf{hasFather} \sqcap \exists \mathsf{hasFather}.\mathsf{Man} \sqcap \mathsf{Human}, \mathsf{Man}(a_1), \mathsf{hasFather}(a_1, a_2)\}$. By applying the structural transformation, we obtain $\mathcal{O}^{\dagger} = \{\mathsf{Man} \sqsubseteq \leq_1 \mathsf{hasFather}, \mathsf{Man} \sqsubseteq \exists \mathsf{hasFather}.\mathsf{Man}, \mathsf{Man} \sqsubseteq \mathsf{Human}, \mathsf{Man}(a_1), \mathsf{hasFather}(a_1, a_2)\}$. By translating $\mathcal{O}^{\dagger}$ to

[2] `http://www.aifb.uni-karlsruhe.de/WBS/gqi/DecomBaR/decompose-long.pdf`

first-order clauses, we obtain $\Xi(\mathcal{O}^\dagger) = \{cl_1, ..., cl_6\}$. Since $y_1 \approx y_2$ occurs positively in $\Xi(\mathcal{O}^\dagger)$, we have $\Xi'(\mathcal{O}^\dagger) = \{cl_1, ..., cl_{13}\} \cup \{t \approx t \mid t \in \mathrm{HU}(\Xi(\mathcal{O}^\dagger))\}$.

$cl_1\colon\ \neg\mathsf{Man}(x) \vee \neg\mathsf{hasFather}(x, y_1) \vee \neg\mathsf{hasFather}(x, y_2) \vee y_1 \approx y_2$

$cl_2\colon\ \neg\mathsf{Man}(x) \vee \mathsf{hasFather}(x, f(x))$ $\quad cl_3\colon\ \neg\mathsf{Man}(x) \vee \mathsf{Man}(f(x))$

$cl_4\colon\ \neg\mathsf{Man}(x) \vee \mathsf{Human}(x)$ $\quad cl_5\colon\ \mathsf{Man}(a_1)$ $\quad cl_6\colon\ \mathsf{hasFather}(a_1, a_2)$

$cl_7\colon\ \neg(x \approx y) \vee y \approx x$ $\quad cl_8\colon\ \neg(x \approx y) \vee \neg(y \approx z) \vee x \approx z$ $\quad cl_9\colon\ \neg(x \approx y) \vee f(x) \approx f(y)$

$cl_{10}\colon\ \neg\mathsf{Man}(x) \vee \neg(x \approx y) \vee \mathsf{Man}(y)$ $\quad cl_{11}\colon\ \neg\mathsf{hasFather}(x, y) \vee \neg(x \approx z) \vee \mathsf{hasFather}(z, y)$

$cl_{12}\colon\ \neg\mathsf{hasFather}(x, y) \vee \neg(y \approx z) \vee \mathsf{hasFather}(x, z)$ $\quad cl_{13}\colon\ \neg\mathsf{Human}(x) \vee \neg(x \approx y) \vee \mathsf{Human}(y)$

3.3 Approximate Grounding of the First-Order Logic Program

According to Lemma 1, we need to address a satisfiability problem about $\Xi'(\mathcal{O}^\dagger)$. This can be done by considering a propositional program that is transformed from $\Xi'(\mathcal{O}^\dagger)$ and has the same set of minimal models as $\Xi'(\mathcal{O}^\dagger)$ has. We extend the well-known intelligent grounding (IG) technique [4] 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 a method for grounding a general first-order logic program, called *reduced grounding* and defined below.

Definition 1 (Reduced Grounding). For a first-order logic program P, the *reduced grounding* of P, denoted by $\mathcal{G}_r(P)$, is the union of two sets of ground clauses $\Pi_1 \cup \Pi_2$, where $\Pi_1 = \{cl \in P \mid cl \text{ is a definite ground fact}\}$, and Π_2 is the least fixpoint of $\Pi^{(n)}$ such that $\Pi^{(0)} = \emptyset$ and for $n > 0$, $\Pi^{(n)} = \{cl\,\sigma \mid cl \in P, \sigma \text{ is a ground substitution such that } cl^-\sigma \subseteq \mathsf{atoms}(\Pi^{(n-1)} \cup \Pi_1), cl^+\sigma \subseteq \mathrm{HB}(P) \text{ and } cl^+\sigma \cap \mathsf{atoms}(\Pi_1) = \emptyset\}$.

Lemma 2. *Let P be a first-order logic program in which the equality predicate $\approx$ has been axiomatized. Then $\mathcal{G}_r(P)$ is a subset of $\mathcal{G}(P)$ and has the same set of minimal models as P has.*

In the following theorem, we show a method that uses $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$ to check if $\mathcal{O} \models q$ for a BCQ $Q : q() \leftarrow p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \wedge ... \wedge p_n(\overrightarrow{y_n}, \overrightarrow{c_n})$. By $\mathrm{Inst}_{\mathrm{BCQ}}(\Pi, Q)$ we denote the result of instantiating the clause $cl : \neg p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \vee ... \vee \neg p_n(\overrightarrow{y_n}, \overrightarrow{c_n}) \vee w_Q()$ based on a propositional program Π, i.e., $\mathrm{Inst}_{\mathrm{BCQ}}(\Pi, Q) = \{cl\,\sigma \mid \sigma \text{ is a ground substitution such that } cl^-\sigma \subseteq \mathsf{atoms}(\Pi)\}$, where w_Q is a predicate symbol corresponding to Q and not occurring in Π. The introduction of $\mathrm{Inst}_{\mathrm{BCQ}}(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)), Q)$ enables the checking of $\mathcal{O} \models q$ to be performed in a propositional program, while the introduction of w_Q facilitates extracting a target ontology w.r.t. Q.

Theorem 1. *For a BCQ $Q : q() \leftarrow p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \wedge ... \wedge p_n(\overrightarrow{y_n}, \overrightarrow{c_n})$ in $\mathcal{O}$, $\mathcal{O} \models q$ iff $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)) \cup \mathrm{Inst}_{\mathrm{BCQ}}(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)), Q) \cup \{\neg w_Q()\}$ is unsatisfiable.*

Based on the above theorem, we develop a basic method that could improve the performance in evaluating a BCQ $Q : q() \leftarrow p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \wedge ... \wedge p_n(\overrightarrow{y_n}, \overrightarrow{c_n})$ in $\mathcal{O}$. This method first extracts a relevant subset Π_{rel} of $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)) \cup \mathrm{Inst}_{\mathrm{BCQ}}(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)), Q) \cup \{\neg w_Q()\}$ such that Π_{rel} is unsatisfiable iff $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)) \cup \mathrm{Inst}_{\mathrm{BCQ}}(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)), Q) \cup \{\neg w_Q()\}$ is unsatisfiable, then identifies a subset $\mathcal{O}_{rel}$ of axioms in $\mathcal{O}^\dagger$ from Π_{rel} such

that Π_{rel} is unsatisfiable only if $\mathcal{O}_{rel} \models q$, and finally checks if $\mathcal{O}_{rel} \models q$. By Theorem 1, we have $\mathcal{O} \models q$ only if $\mathcal{O}_{rel} \models q$. Since $\mathcal{O}_{rel} \subseteq \mathcal{O}^\dagger$ and $\mathcal{O} \models q$ iff $\mathcal{O}^\dagger \models q$, we also have $\mathcal{O} \models q$ if $\mathcal{O}_{rel} \models q$.

However, the basic method cannot be realized in general as $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$ can be infinite. We therefore consider a mapping function on ground terms occurring in a propositional program Π such that the range of this function is finite. 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, an ***e**quality-and-**f**unctional-**t**erm-collapsed mapping function* (simply *eft-mapping function*) for Π, if for every functional term $f_1(...f_n(a))$ (where $n > 1$) occurring in Π, $\lambda(f_1(...f_n(a))) = \lambda(f_n(a))$, and for every equational atom $s \approx t$ occurring positively in Π, $\lambda(s) = \lambda(t)$.

We naturally extend a mapping function λ on ground terms to other first-order objects, i.e., by $\lambda(\alpha)$, $\lambda(cl)$, $\lambda(\mathcal{A})$ and $\lambda(P)$ we respectively denote the results obtained from an atom α, a clause cl, a set $\mathcal{A}$ of atoms and a first-order logic program P by replacing every ground term t occurring in it with $\lambda(t)$.

It is clear that, when a propositional program Π is infinite but the number of constants, predicate symbols and function symbols occurring in Π is finite, $\lambda(\Pi)$ is finite for any eft-mapping function λ for Π. Even when Π is finite, $\lambda(\Pi)$ can be much smaller than Π because the subset of ground clauses in Π that form a congruence relation is collapsed in $\lambda(\Pi)$.

By $\mathrm{Inst}'_{\mathrm{BCQ}}(\Pi, Q, \lambda)$ we denote the result of instantiating the clause $cl : \neg p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \vee ... \vee \neg p_n(\overrightarrow{y_n}, \overrightarrow{c_n}) \vee w_Q()$ based on a propositional program Π and a mapping function λ, i.e., $\mathrm{Inst}'_{\mathrm{BCQ}}(\Pi, Q, \lambda) = \{cl\,\sigma \mid \sigma \text{ is a ground substitution such that } \lambda(cl^-)\sigma \subseteq \mathsf{atoms}(\Pi)\}$, where w_Q is a predicate symbol corresponding to Q and not occurring in Π. We revise the basic method by replacing $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$ with a finite superset Π_{sup} of $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)))$, where λ is an eft-mapping function for $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$. The revised method first extracts a relevant subset Π_{rel} of $\Pi_{sup} \cup \mathrm{Inst}'_{\mathrm{BCQ}}(\Pi_{sup}, Q, \lambda) \cup \{\neg w_Q()\}$, then computes the set $\mathcal{O}_{rel}$ of axioms ax in $\mathcal{O}^\dagger$ such that $\lambda(cl\,\sigma) \in \Pi_{rel}$ for some clause $cl \in \Xi(ax)$ and some ground substitution σ, and finally checks if $\mathcal{O}_{rel} \models q$.

Consider the extraction of a relevant subset Π_{rel} mentioned above. Our extraction method is based on the notion of *connected component* (see Definition 2 below). Simply speaking, a connected component of a propositional program Π is a subset of Π 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 because an unsatisfiable propositional program must have a maximal connected component that is unsatisfiable.

Definition 2 (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 for extracting Π_{rel} is that when $\Pi_{sup} \cup \mathrm{Inst}'_{\mathrm{BCQ}}(\Pi_{sup}, Q, \lambda) \cup \{\neg w_Q()\}$ is unsatisfiable, the maximal connected component of $\Pi_{sup} \cup \mathrm{Inst}'_{\mathrm{BCQ}}(\Pi_{sup}, Q, \lambda) \cup \{\neg w_Q()\}$ where $w_Q()$ occurs is also unsatisfiable. To obtain a smaller unsatisfiable

subset, we extend the basic idea by removing a subset Π_{ur} from Π_{sup} first and then extracting the maximal connected component Π_{rel} of $(\Pi_{sup} \cup \mathrm{Inst}'_{\mathrm{BCQ}}(\Pi_{sup}, Q, \lambda) \cup \{\neg w_Q()\}) \setminus \Pi_{ur}$ where $w_Q()$ occurs. The detailed description and the correctness of the method are shown in the following theorem.

Theorem 2. *Let $Q : q() \leftarrow p_1(\overrightarrow{y_1}, \overrightarrow{c_1}) \wedge ... \wedge p_n(\overrightarrow{y_n}, \overrightarrow{c_n})$ be a BCQ, λ be an eft-mapping function for $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$, Π_{sup} be a superset of $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)))$ and $\Pi' = \Pi_{sup} \cup \mathrm{Inst}'_{\mathrm{BCQ}}(\Pi_{sup}, Q, \lambda) \cup \{\neg w_Q()\}$. Let Π_{ur} be a subset of Π' such that for all clauses $cl \in \Pi_{ur}$, cl^+ contains at least one ground atom not occurring in $\Pi' \setminus \Pi_{ur}$ and Π_{rel} be the maximal connected component of $\Pi' \setminus \Pi_{ur}$ where $w_Q()$ occurs. Let $\mathcal{O}_{rel} = \{ax \in \mathcal{O}^\dagger \mid$ there exists a clause $cl \in \Xi(ax)$ and a ground substitution σ such that $\lambda(cl\,\sigma) \in \Pi_{rel}\}$. Then $\mathcal{O} \models q$ iff $\mathcal{O}_{rel} \models q$.*

Based on the above theorem, we develop an algorithm to compute a superset of $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)))$ for some eft-mapping function λ. This algorithm, denoted by $\mathrm{Approx\text{-}Ground}(\mathcal{O}^\dagger)$, accepts $\mathcal{O}^\dagger$ and returns a triple $(\Pi, \mathcal{S}_{df}, \mathcal{S})$, where $\mathcal{S}_{df}$ and $\mathcal{S}$ are two sets of sets of ground atoms and Π is a superset of $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)))$ for some eft-mapping function λ that can be constructed from $\mathcal{S}_{df}$ and $\mathcal{S}$. Since the algorithm is rather technical, we only explains the basic idea here and refer the interested reader to our technical report[2] for technical details.

The output $\mathcal{S}_{df}$ is actually the set of sets of constants such that for any constant a in any $\mathcal{C} \in \mathcal{S}_{df}$, there exists an equality assertion $a \approx b$ in $\mathcal{O}^\dagger$ for some constant $b \in \mathcal{C}$. The output $\mathcal{S}$ is actually the set of sets of ground terms whose functional depth is at most one, such that for any ground term s in any $\mathcal{C} \in \mathcal{S}$, there exists a ground term $t \in \mathcal{C}$ such that the equational atom $s \approx t$ appears in the execution of the algorithm. Both $\mathcal{S}_{df}$ and $\mathcal{S}$ are used to merge ground terms that may occur in the same equational atom occurring positively in $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$, making the output Π smaller. We call an element of $\mathcal{S}_{df}$ or $\mathcal{S}$ a *congruence class*.

The algorithm does not directly compute an eft-mapping function λ for $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$, because such mapping function can be constructed from $\mathcal{S}_{df}$ and $\mathcal{S}$. By $\mathsf{map}(t, \mathcal{S}_{df}, \mathcal{S})$ we denote a function $\mathrm{HU}(\Xi'(\mathcal{O}^\dagger)) \mapsto \mathrm{HU}(\Xi'(\mathcal{O}^\dagger))$ based on $\mathcal{S}_{df}$ and $\mathcal{S}$, recursively defined as follows, where a and b are constants, and s and t ground terms.

- $\mathsf{map}(f_1(...f_n(a)), \mathcal{S}_{df}, \mathcal{S}) = \mathsf{map}(f_n(a), \mathcal{S}_{df}, \mathcal{S})$, where $n > 1$;
- $\mathsf{map}(f(a), \mathcal{S}_{df}, \mathcal{S}) = \mathsf{map}(f(b), \emptyset, \mathcal{S})$, where $b = \min(\mathcal{C})$ if $a \in \mathcal{C}$ for some $\mathcal{C} \in \mathcal{S}_{df}$, or $b = a$ otherwise;
- $\mathsf{map}(a, \mathcal{S}_{df}, \mathcal{S}) = \mathsf{map}(b, \emptyset, \mathcal{S})$, where $b = \min(\mathcal{C})$ if $a \in \mathcal{C}$ for some $\mathcal{C} \in \mathcal{S}_{df}$, or $b = a$ otherwise;
- $\mathsf{map}(s, \emptyset, \mathcal{S}) = t$, where $t = \min(\mathcal{C})$ if $s \in \mathcal{C}$ for some $\mathcal{C} \in \mathcal{S}$, or $t = s$ otherwise.

We naturally extend the function map to other first-order objects, i.e., by $\mathsf{map}(\alpha, \mathcal{S}_{df}, \mathcal{S})$, $\mathsf{map}(cl, \mathcal{S}_{df}, \mathcal{S})$, $\mathsf{map}(\mathcal{A}, \mathcal{S}_{df}, \mathcal{S})$ and $\mathsf{map}(P, \mathcal{S}_{df}, \mathcal{S})$ we respectively denote the results obtained from an atom α, a clause cl, a set $\mathcal{A}$ of atoms and a first-order logic program P by replacing every ground term t occurring in it with $\mathsf{map}(t, \mathcal{S}_{df}, \mathcal{S})$.

We call a mapping function $\lambda : \mathrm{HU}(\Xi'(\mathcal{O}^\dagger)) \mapsto \mathrm{HU}'(\Xi(\mathcal{O}^\dagger))$ *induced from* the function map w.r.t. $\mathcal{S}_{df}$ and $\mathcal{S}$ if $\lambda(t) = \mathsf{map}(t, \mathcal{S}_{df}, \mathcal{S})$ for all ground terms $t \in \mathrm{HU}(\Xi'(\mathcal{O}^\dagger))$. The first goal of the algorithm is to ensure the mapping function λ induced from map w.r.t. $\mathcal{S}_{df}$ and $\mathcal{S}$ to be an eft-mapping function for $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$,

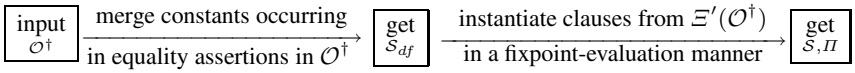

Fig. 1. The main steps for approximately grounding $\Xi'(\mathcal{O}^\dagger)$

i.e., ensure $\mathsf{map}(s, \mathcal{S}_{df}, \mathcal{S}) = \mathsf{map}(t, \mathcal{S}_{df}, \mathcal{S})$ for all equational atoms $s \approx t$ occurring positively in $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$. The second goal of the algorithm is to return a superset of $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)))$. To achieve the above two goals, the algorithm works in two main steps, as shown in Figure 1.

In the first step, the algorithm places any two constants a and b that occur in the same equality assertion in $\mathcal{O}^\dagger$ into the same congruence class $\mathcal{C}$ and adds $\mathcal{C}$ to $\mathcal{S}_{df}$. After this step, $\mathcal{S}_{df}$ will not be changed anymore.

In the second step, the algorithm instantiates clauses from $\Xi'(\mathcal{O}^\dagger)$ in a fixpoint-evaluation manner to generate a superset of $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)))$, where λ denotes a mapping function induced from map w.r.t. $\mathcal{S}_{df}$ and $\mathcal{S}$.

Before giving a fixpoint-like characterization of a superset of $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)))$, we need to introduce a restriction on $\mathcal{O}^\dagger$. We call $\mathcal{O}^\dagger$ *congruence-complete* if (1) $a \approx b \in O^\dagger$ implies $b \approx a \in O^\dagger$; (2) $a \approx b \in O^\dagger$ and $b \approx c \in O^\dagger$ imply $a \approx c \in O^\dagger$; (3) $a \approx b \in O^\dagger$ and $A(a) \in O^\dagger$ imply $A(b) \in \mathcal{O}^\dagger$; (4) $a \approx b \in O^\dagger$ and $R(a, c) \in O^\dagger$ imply $R(b, c) \in \mathcal{O}^\dagger$; and (5) $a \approx b \in O^\dagger$ and $R(c, a) \in O^\dagger$ imply $R(c, b) \in \mathcal{O}^\dagger$.

Let Π_1 denote the set of definite ground facts in $\Xi'(\mathcal{O}^\dagger)$. By induction on the level n of $\Pi^{(n)}$ given in Definition 1, we can show that, if Π is a subset of $\lambda(\mathcal{G}(\Xi'(\mathcal{O}^\dagger)))$ such that (*) $\lambda(cl\,\sigma) \in \Pi$ for any clause $cl \in \Xi'(\mathcal{O}^\dagger)$ and any ground substitution σ such that $\lambda(cl^-\sigma) \subseteq \mathsf{atoms}(\Pi \cup \lambda(\Pi_1))$ and every ground atom in $\lambda(cl^+\sigma) \cap \mathsf{atoms}(\lambda(\Pi_1))$ contains ground terms that are functional or occur in $\mathcal{S}$, then $\Pi \cup \lambda(\Pi_1)$ is a superset of $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger)))$ when $\mathcal{O}^\dagger$ is congruence-complete. We refer the interested reader to our technical report[2] to see the proof of the above conclusion, a counterexample on why the restriction on congruence-completeness is needed, as well as a simple method for making $\mathcal{O}^\dagger$ congruence-complete when $\Theta(\mathcal{O})$ is not congruence-complete.

We in what follows assume that $\mathcal{O}^\dagger$ is congruence-complete. Under this assumption, we refine the second goal of the algorithm to finding a subset Π of $\lambda(\mathcal{G}(\Xi'(\mathcal{O}^\dagger)))$ that satisfies the above condition (*). To achieve this goal, the second step of the algorithm adds $\lambda(cl\,\sigma)$ to Π for any clause $cl \in \Xi'(\mathcal{O}^\dagger)$ and ground substitution σ such that (i) $\lambda(cl^-\sigma) \subseteq \mathsf{atoms}(\Pi_P \cup \lambda(\Pi_1))$ and (ii) every ground atom in $\lambda(cl^+\sigma) \cap \mathsf{atoms}(\lambda(\Pi_1))$ contains ground terms that are functional or occur in $\mathcal{S}$. Meanwhile, any equational atom $s \approx t$ occurring positively in $cl\,\sigma$ is handled by placing s and t into the same congruence class $\mathcal{C}$ and by adding $\mathcal{C}$ to $\mathcal{S}$. In order to achieve the first goal of the algorithm, $f(s)$ and $f(t)$ for f a function symbol occurring in $\Xi(\mathcal{O}^\dagger)$ are merged similarly as s and t, because the clause $\neg(s \approx t) \vee f(s) \approx f(t)$, instantiated from the clause of the form (4) in Section 2, may belong to $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$.

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

Lemma 3. *Let* $(\Pi, \mathcal{S}_{df}, \mathcal{S})$ *be returned by* $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$ *and* λ *be a mapping function induced from the function* map *w.r.t.* $\mathcal{S}_{df}$ *and* $\mathcal{S}$*. Then: (1)* $\lambda(\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))) \subseteq \Pi$

and λ is an eft-mapping function for $\mathcal{G}_r(\Xi'(\mathcal{O}^\dagger))$; (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 number restrictions in $\mathcal{O}$ and s is the size of $\mathcal{O}$.*

Based on the above lemma, we can use the result $(\Pi, \mathcal{S}_{df}, \mathcal{S})$ of $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$ to compute a subset $\mathcal{O}_{rel}$ of $\mathcal{O}^\dagger$ such that $\mathcal{O} \models q$ iff $\mathcal{O}_{rel} \models q$ for a BCQ $q() \leftarrow \mathsf{conj}(\overrightarrow{y}, \overrightarrow{c})$, by applying the method given in Theorem 2.

Before ending this subsection, we show the result of $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$ for the ontology $\mathcal{O}^\dagger$ given in Example 1 (note that $\mathcal{O}^\dagger$ is congruence-complete).

Example 2 (Example 1 continued). $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$ returns a triple $(\Pi, \mathcal{S}_{df}, \mathcal{S})$, where $\Pi = \{cl'_1, ..., cl'_{30}\}$, $\mathcal{S}_{df} = \emptyset$ and $\mathcal{S} = \{\{a_2, f(a_1)\}\}$.

cl'_1: $\mathsf{Man}(a_1)$ cl'_2: $\mathsf{hasFather}(a_1, a_2)$
cl'_3: $\neg\mathsf{Man}(a_1) \vee \neg\mathsf{hasFather}(a_1, a_2) \vee \neg\mathsf{hasFather}(a_1, a_2) \vee a_2 \approx a_2$
cl'_4: $\neg\mathsf{Man}(a_2) \vee \neg\mathsf{hasFather}(a_2, f(a_2)) \vee \neg\mathsf{hasFather}(a_2, f(a_2)) \vee f(a_2) \approx f(a_2)$
cl'_5: $\neg\mathsf{Man}(f(a_2)) \vee \neg\mathsf{hasFather}(f(a_2), f(a_2)) \vee \neg\mathsf{hasFather}(f(a_2), f(a_2)) \vee f(a_2) \approx f(a_2)$
cl'_6: $\neg\mathsf{Man}(a_1) \vee \mathsf{hasFather}(a_1, a_2)$ cl'_7: $\neg\mathsf{Man}(a_2) \vee \mathsf{hasFather}(a_2, f(a_2))$
cl'_8: $\neg\mathsf{Man}(f(a_2)) \vee \mathsf{hasFather}(f(a_2), f(a_2))$ cl'_9: $\neg\mathsf{Man}(a_1) \vee \mathsf{Man}(a_2)$
cl'_{10}: $\neg\mathsf{Man}(a_2) \vee \mathsf{Man}(f(a_2))$ cl'_{11}: $\neg\mathsf{Man}(f(a_2)) \vee \mathsf{Man}(f(a_2))$
cl'_{12}: $\neg\mathsf{Man}(a_1) \vee \mathsf{Human}(a_1)$ cl'_{13}: $\neg\mathsf{Man}(a_2) \vee \mathsf{Human}(a_2)$ cl'_{14}: $\neg\mathsf{Man}(f(a_2)) \vee \mathsf{Human}(f(a_2))$
cl'_{15}: $a_1 \approx a_1$ cl'_{16}: $a_2 \approx a_2$ cl'_{17}: $f(a_2) \approx f(a_2)$
cl'_{18}: $\neg(a_2 \approx a_2) \vee a_2 \approx a_2$ cl'_{19}: $\neg(a_2 \approx a_2) \vee f(a_2) \approx f(a_2)$
cl'_{20}: $\neg(f(a_2) \approx f(a_2)) \vee f(a_2) \approx f(a_2)$ cl'_{21}: $\neg(a_2 \approx a_2) \vee \neg(a_2 \approx a_2) \vee a_2 \approx a_2$
cl'_{22}: $\neg(f(a_2) \approx f(a_2)) \vee \neg(f(a_2) \approx f(a_2)) \vee f(a_2) \approx f(a_2)$
cl'_{23}: $\neg\mathsf{Man}(a_2) \vee \neg(a_2 \approx a_2) \vee \mathsf{Man}(a_2)$ cl'_{24}: $\neg\mathsf{Man}(f(a_2)) \vee \neg(f(a_2) \approx f(a_2)) \vee \mathsf{Man}(f(a_2))$
cl'_{25}: $\neg\mathsf{hasFather}(a_1, a_2) \vee \neg(a_2 \approx a_2) \vee \mathsf{hasFather}(a_1, a_2)$
cl'_{26}: $\neg\mathsf{hasFather}(a_2, f(a_2)) \vee \neg(a_2 \approx a_2) \vee \mathsf{hasFather}(a_2, f(a_2))$
cl'_{27}: $\neg\mathsf{hasFather}(a_2, f(a_2)) \vee \neg(f(a_2) \approx f(a_2)) \vee \mathsf{hasFather}(a_2, f(a_2))$
cl'_{28}: $\neg\mathsf{hasFather}(f(a_2), f(a_2)) \vee \neg(f(a_2) \approx f(a_2)) \vee \mathsf{hasFather}(f(a_2), f(a_2))$
cl'_{29}: $\neg\mathsf{Human}(a_2) \vee \neg(a_2 \approx a_2) \vee \mathsf{Human}(a_2)$
cl'_{30}: $\neg\mathsf{Human}(f(a_2)) \vee \neg(f(a_2) \approx f(a_2)) \vee \mathsf{Human}(f(a_2))$

3.4 Computing All Answers with the Help of the Grounding

In this subsection, we present a method for computing all answers of a general CQ by using $\Xi(\mathcal{O}^\dagger)$ and the result of $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$.

For a propositional program Π, two sets $\mathcal{S}_{df}$ and $\mathcal{S}$ of sets of ground terms occurring in Π, and a CQ $Q : q(\overrightarrow{x}) \leftarrow p_1(\overrightarrow{x_1}, \overrightarrow{y_1}, \overrightarrow{c_1}) \wedge ... \wedge p_n(\overrightarrow{x_n}, \overrightarrow{y_n}, \overrightarrow{c_n})$, where $\overrightarrow{x}$ is the union of $\overrightarrow{x_1}, ..., \overrightarrow{x_n}$, by $\mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S})$ we denote the result of instantiating the clause $cl : \neg p_1(\overrightarrow{x_1}, \overrightarrow{y_1}, \overrightarrow{c_1}) \vee ... \vee \neg p_n(\overrightarrow{x_n}, \overrightarrow{y_n}, \overrightarrow{c_n}) \vee w_Q(\overrightarrow{x})$ based on Π, $\mathcal{S}_{df}$ and $\mathcal{S}$, where w_Q is a predicate symbol corresponding to Q and not occurring in Π, i.e., $\mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S}) = \{cl\,\sigma \mid \sigma$ is a ground substitution such that all ground atoms in $\mathsf{map}(cl^-, \mathcal{S}_{df}, \mathcal{S})\,\sigma$ occur in Π and $\overrightarrow{x}\sigma$ is a tuple of constants$\}$. For example, for $\Pi, \mathcal{S}_{df}$ and $\mathcal{S}$ given in Example 2 and a CQ $Q : q(x) \leftarrow \mathsf{Man}(x) \wedge \mathsf{hasFather}(x, y)$, $\mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S}) = \{\neg\mathsf{Man}(a_1) \vee \neg\mathsf{hasFather}(a_1, a_2) \vee w_Q(a_1)\} \cup \{\neg\mathsf{Man}(a_2) \vee \neg\mathsf{hasFather}(a_2, f(a_2)) \vee w_Q(a_2)\}$. With the above denotation, the following lemma gives a necessary condition that an answer of a CQ in $\mathcal{O}$ should satisfy.

Lemma 4. *Let $(\Pi, \mathcal{S}_{df}, \mathcal{S})$ be returned by* $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$. *For a CQ $Q : q(\overrightarrow{x}) \leftarrow \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ in $\mathcal{O}$, a tuple of constants $\overrightarrow{t}$ is an answer of Q in $\mathcal{O}$ only if $\mathsf{map}(w_Q(\overrightarrow{t}), \mathcal{S}_{df}, \mathcal{S})$ occurs in* $\mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S})$.

The following lemma gives a sufficient condition that ensures a tuple $\overrightarrow{t}$ of constants to be an answer of a CQ $Q : q(\overrightarrow{x}) \leftarrow \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ in $\mathcal{O}$, where $q[\overrightarrow{x} \mapsto \overrightarrow{t}, \overrightarrow{y} \mapsto \overrightarrow{s}]$ denotes the body of q with every variable in $\overrightarrow{x}$ and $\overrightarrow{y}$ respectively substituted by its corresponding ground terms in $\overrightarrow{t}$ and $\overrightarrow{s}$.

Lemma 5. *For a CQ $Q : q(\overrightarrow{x}) \leftarrow \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ in $\mathcal{O}$, a tuple $\overrightarrow{t}$ of constants is an answer of Q in $\mathcal{O}$ if there exists a tuple $\overrightarrow{s}$ of ground terms such that every ground atom occurring in $q[\overrightarrow{x} \mapsto \overrightarrow{t}, \overrightarrow{y} \mapsto \overrightarrow{s}]$ is satisfied by all models of $\Xi'(\mathcal{O}^\dagger)$.*

Based on the above two lemmas, we can identify a set of candidate answers of a CQ $Q : q(\overrightarrow{x}) \leftarrow \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ in $\mathcal{O}$. Afterwards, we may, for every candidate answer $\overrightarrow{t}$, compute a subset of axioms in $\mathcal{O}^\dagger$ to check if $\mathcal{O} \models q[\overrightarrow{x} \mapsto \overrightarrow{t}]$ by applying the method given in Theorem 2. However, handling candidate answers one by one is inefficient because it needs to extract, for each candidate answer $\overrightarrow{t}$, a relevant subset of $\Pi \cup \mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S}) \cup \{\mathsf{map}(\neg w_Q(\overrightarrow{t}), \mathcal{S}_{df}, \mathcal{S})\}$; such computation is quite costly. To improve the efficiency, we extract all relevant subsets from $\Pi \cup \mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S}) \cup \{\mathsf{map}(\neg w_Q(\overrightarrow{t}), \mathcal{S}_{df}, \mathcal{S}) \mid \overrightarrow{t}$ is a candidate answer$\}$ in one pass, then from each extracted subset, identify a subset $\mathcal{O}_{rel}$ of axioms in $\mathcal{O}^\dagger$ and evaluate Q over $\mathcal{O}_{rel}$ by applying an OWL DL reasoner.

The algorithm for query answering is given in Figure 2, where the input $\mathcal{A}$ can be the set of ground atoms occurring in definite ground facts in $\Xi(\mathcal{O}^\dagger)$. We explain how the algorithm works. Lines 1–2 respectively compute a set Ans of explicit answers and a set $Cands$ of candidate answers of Q in $\mathcal{O}$ based on Lemma 5 and Lemma 4. Line 3 decomposes $\Pi^\dagger = \Pi \cup \mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S}) \cup \{\mathsf{map}(\neg w_Q(\overrightarrow{t}), \mathcal{S}_{df}, \mathcal{S}) \mid \overrightarrow{t} \in Cands\}$ to a set of disjoint subsets from which target ontologies can be extracted. The subprocedure $\mathrm{Decompose}(\Pi^\dagger)$ first filters the largest subset Π_0 of $\Pi^\dagger$ such that for all clauses $cl \in \Pi_0$, cl^+ has at least one ground atom not occurring in $\Pi^\dagger \setminus \Pi_0$, then returns the set of maximal connected components of $\Pi^\dagger \setminus \Pi_0$. Basically, Π_0 is the greatest fixpoint of $\Pi_0^{(n)}$ such that $\Pi_0^{(0)} = \Pi^\dagger$ and for $n > 0$, $\Pi_0^{(n)} = \{cl \in \Pi_0^{(n-1)} \mid cl^+ \setminus \mathsf{atoms}(\Pi^\dagger \setminus \Pi_0^{(n-1)}) \neq \emptyset\}$ (see our technical report[2] for how to realize the decomposition process). Lines 4–6 handle every maximal connected component Π_{rel} of $\Pi^\dagger \setminus \Pi_0$: if any ground atom over w_Q does not occur in Π_{rel}, Π_{rel} is irrelevant to Q and thus neglected; otherwise, a subset $\mathcal{O}_{rel}$ of $\mathcal{O}^\dagger$ is extracted from Π_{rel}, and all answers of Q in $\mathcal{O}_{rel}$ are added to Ans (the evaluation of $\mathcal{O}_{rel}$ of $\mathcal{O}^\dagger$ is realized in $\mathrm{TraditionalEvaluate}$, which applies an OWL DL reasoner). Note that different extracted ontologies have no common ABox axioms because ABox axioms correspond to ground atoms in $\Pi^\dagger \setminus \Pi_0$ and different maximal connected components of $\Pi^\dagger \setminus \Pi_0$ have no common ground atoms.

Example 3 (Example 2 continued). Given a CQ $Q : q(x) \leftarrow \mathsf{Man}(x) \wedge \mathsf{hasFather}(x, y)$ in $\mathcal{O}$ given in Example 1, we show how $\mathrm{DecompBasedEvaluate}(Q, \mathcal{A}, \Pi, \mathcal{S}_{df}, \mathcal{S})$ works, where $\mathcal{A}$ is the set of ground atoms occurring in definite ground facts in $\Xi(\mathcal{O}^\dagger)$, i.e., $\mathcal{A} = \{\mathsf{Man}(a_1), \mathsf{hasFather}(a_1, a_2)\}$, and $\Pi, \mathcal{S}_{df}, \mathcal{S}$ are given in Example 2. Line 1 in Figure 2 sets Ans as $\{a_1\}$. Line 2 sets $Cands$ as $\{a_2\}$. Line 3 computes $\Pi^\dagger = \Pi \cup \mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S}) \cup \{\mathsf{map}(\neg w_Q(\overrightarrow{t}), \mathcal{S}_{df}, \mathcal{S}) \mid \overrightarrow{t} \in Cands\} = \{cl'_1, \ldots, cl'_{33}\}$ and calls $\mathrm{Decompose}(\Pi^\dagger)$, where $cl'_1, \ldots, cl'_{30}$ are given in Example 2, cl'_{31} is

Algorithm 1. $\mathrm{DecompBasedEvaluate}(Q, \mathcal{A}, \Pi, \mathcal{S}_{df}, \mathcal{S})$

In: A CQ $Q : q(\overrightarrow{x}) \leftarrow p_1(\overrightarrow{x_1}, \overrightarrow{y_1}, \overrightarrow{c_1}) \wedge \ldots \wedge p_n(\overrightarrow{x_n}, \overrightarrow{y_n}, \overrightarrow{c_n})$, a set $\mathcal{A}$ of ground atoms satisfied by all models of $\Xi'(\mathcal{O}^\dagger)$ and the results $\Pi, \mathcal{S}_{df}, \mathcal{S}$ returned by $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$.

Out: The set of answers of Q in $\mathcal{O}$.

1. $Ans := \{\overrightarrow{x}\sigma \mid \sigma$ is a ground substitution such that $\{p_1(\overrightarrow{x_1}, \overrightarrow{y_1}, \overrightarrow{c_1}), \ldots, p_n(\overrightarrow{x_n}, \overrightarrow{y_n}, \overrightarrow{c_n})\}\sigma \subseteq \mathcal{A}\}$ and $\overrightarrow{x}\sigma$ is a tuple of constants$\}$;
2. $Cands := \{\overrightarrow{x}\sigma \mid \sigma$ is a ground substitution such that $\mathsf{map}(w_Q(\overrightarrow{x}\sigma), \mathcal{S}_{df}, \mathcal{S})$ occurs in $\mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S})$ and $\overrightarrow{x}\sigma$ is a tuple of constants$\} \setminus Ans$;
3. $RSets := \mathrm{Decompose}(\Pi \cup \mathrm{Inst}_{\mathrm{CQ}}(\Pi, Q, \mathcal{S}_{df}, \mathcal{S}) \cup \{\mathsf{map}(\neg w_Q(\overrightarrow{t}), \mathcal{S}_{df}, \mathcal{S}) \mid \overrightarrow{t} \in Cands\})$;
4. **for** each $\Pi_{rel} \in RSets$ such that some ground atoms over w_Q occur in Π_{rel} **do**
5. $\quad \mathcal{O}_{rel} := \{ax \in \mathcal{O}^\dagger \mid$ there exists a clause $cl \in \Xi(ax)$ and a ground substitution σ such that $\mathsf{map}(cl\,\sigma, \mathcal{S}_{df}, \mathcal{S}) \in \Pi_{rel}\}$;
6. $\quad Ans := Ans \cup \mathrm{TraditionalEvaluate}(Q, \mathcal{O}_{rel})$;
7. **return** Ans;

Fig. 2. A decomposition-based algorithm for evaluating a CQ

$\neg\mathsf{Man}(a_1) \vee \neg\mathsf{hasFather}(a_1, a_2) \vee w_Q(a_1)$, cl'_{32} is $\neg\mathsf{Man}(a_2) \vee \neg\mathsf{hasFather}(a_2, f(a_2)) \vee w_Q(a_2)$ and cl'_{33} is $\neg w_Q(a_2)$. It can be checked that the filtered set Π_0 of $\Pi^\dagger$ is $\{cl'_{12}, cl'_{13}, cl'_{14}, cl'_{29}, cl'_{30}, cl'_{31}\}$ and the remaining set has a single maximal connected component. Hence $\mathrm{Decompose}(\Pi^\dagger)$ returns $\{\{\Pi_{rel}\}\}$, where $\Pi_{rel} = \Pi^\dagger \setminus \Pi_0$. Since $w_Q(a_2)$ occurs in Π_{rel}, line 5 in Figure 2 is executed, yielding $\mathcal{O}_{rel} = \{\mathsf{Man} \sqsubseteq \leq_1 \mathsf{hasFather}, \mathsf{Man} \sqsubseteq \exists\mathsf{hasFather}.\mathsf{Man}, \mathsf{Man}(a_1), \mathsf{hasFather}(a_1, a_2)\}$. Note that $\mathcal{O}_{rel}$ is a subset of $\mathcal{O}^+$ from which the axiom $\mathsf{Man} \sqsubseteq \mathsf{Human}$ is removed. By applying an OWL DL reasoner, we can check that a_2 is the unique answer of Q in $\mathcal{O}_{rel}$, so Ans is updated to $\{a_1, a_2\}$ in line 6 and finally returned by $\mathrm{DecompBasedEvaluate}(Q, \mathcal{A}, \Pi, \mathcal{S}_{df}, \mathcal{S})$.

In the remainder of this section, we assume that the OWL DL reasoner applied in our approach is sound and complete for the category of given CQs, i.e., the subprocedure $\mathrm{TraditionalEvaluate}$ is correct. The following theorem shows the correctness of $\mathrm{DecompBasedEvaluate}$.

Theorem 3. *Let* $Q : q(\overrightarrow{x}) \leftarrow \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ *be a CQ,* $\mathcal{A}$ *be a set of ground atoms satisfied by all models of* $\Xi'(\mathcal{O}^\dagger)$*,* $(\Pi, \mathcal{S}_{df}, \mathcal{S})$ *be returned by* $\mathrm{ApproxGround}(\mathcal{O}^\dagger)$*. Then* $\mathrm{DecompBasedEvaluate}(Q, \mathcal{A}, \Pi, \mathcal{S}_{df}, \mathcal{S})$ *returns the set of answers of* Q *in* $\mathcal{O}$*.*

3.5 Optimization by Computing More ABox Entailments

Based on Figure 2 (line 1), we can see that if we obtain more definite ground facts of $\Xi(\mathcal{O}^\dagger)$, we can compute more explicit answers of a given CQ; thus, we can further improve the performance of our approach. We therefore present an important optimization that computes more entailments of $\mathcal{O}^\dagger$ before calling $\mathrm{ApproxGrounding}(\mathcal{O}^\dagger)$.

Basically, the optimization computes a set $\mathcal{A}$ of ground atoms from the set of definite clauses in $\Xi(\Theta(\mathcal{O}))$ such that the functional depth of every ground term occurring in $\mathcal{A}$ is at most one. Recall that $\Theta(\mathcal{O})$ is the result of applying structural transformation to $\mathcal{O}$. We call such subset $\mathcal{A}$ the *bounded entailment set* of $\Theta(\mathcal{O})$, which is defined as the least fixpoint of $\mathcal{A}^{(n)}$ such that $\mathcal{A}^{(0)} = \emptyset$ and for $n > 0$, $\mathcal{A}^{(n)} = \bigcup\{cl^+\sigma \mid cl \in \mathrm{DS}(\Xi'(\Theta(\mathcal{O}))), \sigma$ is a ground substitution such that

$cl^{-}\sigma \subseteq \mathcal{A}^{(n-1)}$, $cl^{+}\sigma \subseteq \mathrm{HB}(\Xi(\Theta(\mathcal{O})))$ and $\mathsf{depth}(cl\,\sigma) \leq 1\}$, where $\mathsf{depth}(cl)$ denotes the maximum functional depth of all ground terms occurring in a ground clause cl, and $\mathrm{DS}(\Xi'(\Theta(\mathcal{O})))$ denotes the set of all definite clauses in $\Xi'(\Theta(\mathcal{O}))$.

Let $\mathcal{A}$ be the bounded entailment set of $\Theta(\mathcal{O})$. Since $\mathcal{A}$ is a set of ground atoms in the least model of $\mathrm{DS}(\Xi'(\Theta(\mathcal{O}))$ and the least model of $\mathrm{DS}(\Xi'(\Theta(\mathcal{O})))$ is a subset of every model of $\Xi'(\Theta(\mathcal{O}))$, every ground atom in $\mathcal{A}$ is satisfied by all models of $\Xi'(\Theta(\mathcal{O}))$. Let $\mathcal{A}_C$ be the subset of $\mathcal{A}$ that contains only constants. When $\Theta(\mathcal{O}) \cup \mathcal{A}_C$ has equality assertions, some equational atoms must occur positively in $\Xi(\Theta(\mathcal{O}))$, so $\mathcal{E}(\Xi(\Theta(\mathcal{O}))) \subseteq \mathrm{DS}(\Xi'(\Theta(\mathcal{O}))$. It follows that $\Theta(\mathcal{O}) \cup \mathcal{A}_C$ is congruence-complete. In the remainder of this section, we assume that $\mathcal{O}^{\dagger} = \Theta(\mathcal{O}) \cup \mathcal{A}_C$. Since every ground atom in $\mathcal{A}_C$ is satisfied by all models of $\Theta(\mathcal{O})$, Lemma 1 and whatever follows from Lemma 1 still hold when $\mathcal{O}^{\dagger} = \Theta(\mathcal{O}) \cup \mathcal{A}_C$.

To evaluate a CQ Q in $\mathcal{O}$, we now call $\mathrm{DecompBasedEvaluate}(Q, \mathcal{A}, \Pi, \mathcal{S}_{df}, \mathcal{S})$, where $\mathcal{A}$ is the bounded entailment set of $\Theta(\mathcal{O})$ and $(\Pi, \mathcal{S}_{df}, \mathcal{S})$ are returned by $\mathrm{ApproxGround}(\mathcal{O}^{\dagger})$. Theorem 4 shows that the optimization also guarantees sound and complete results. Example 4 illustrates that the optimization does take effect in our running example.

Theorem 4. *Let $Q : q(\overrightarrow{x}) \leftarrow \mathsf{conj}(\overrightarrow{x}, \overrightarrow{y}, \overrightarrow{c})$ be a CQ, $\mathcal{A}$ be the bounded entailment set of $\Theta(\mathcal{O})$, $(\Pi, \mathcal{S}_{df}, \mathcal{S})$ be returned by $\mathrm{ApproxGround}(\mathcal{O}^{\dagger})$ and Ans be returned by $\mathrm{DecompBasedEvaluate}(Q, \mathcal{A}, \Pi, \mathcal{S}_{df}, \mathcal{S})$. Then Ans is the set of answers of Q in $\mathcal{O}$.*

Example 4 (Example 3 continued). For the ontology $\mathcal{O}$ given in Example 1, since $\Xi'(\Theta(\mathcal{O})) = \{cl_1, ..., cl_{13}\} \cup \{t \approx t \mid t \in \mathrm{HU}(\Xi(\mathcal{O}^{\dagger}))\}$, where $cl_1, ..., cl_{13}$ are given in Example 1, we have $\mathrm{DS}(\Xi'(\Theta(\mathcal{O}))) = \Xi'(\Theta(\mathcal{O}))$. We can compute the bounded entailment set $\mathcal{A}$ of $\Theta(\mathcal{O})$ as $\{\mathsf{Man}(a_1), \mathsf{Man}(f(a_1)), \mathsf{Man}(a_2), \mathsf{Man}(f(a_2)), \mathsf{Human}(a_1), \mathsf{Human}(f(a_1)), \mathsf{Human}(a_2), \mathsf{Human}(f(a_2)), \mathsf{hasFather}(a_1, a_2), \mathsf{hasFather}(a_1, f(a_1)), \mathsf{hasFather}(a_2, f(a_2)), \mathsf{hasFather}(f(a_1), f(a_2)), a_2 \approx f(a_1), f(a_1) \approx a_2, a_1 \approx a_1, f(a_1) \approx f(a_1), a_2 \approx a_2, f(a_2) \approx f(a_2)\}$. By appending to $\Theta(\mathcal{O})$ the set $\mathcal{A}_C$ of ground atoms in $\mathcal{A}$ that contains only constants, we obtain $\mathcal{O}^{\dagger} = \{\mathsf{Man} \sqsubseteq \leq_1 \mathsf{hasFather}, \mathsf{Man} \sqsubseteq \exists \mathsf{hasFather}.\mathsf{Man}, \mathsf{Man} \sqsubseteq \mathsf{Human}, \mathsf{Man}(a_1), \mathsf{Man}(a_2), \mathsf{Human}(a_1), \mathsf{Human}(a_2), \mathsf{hasFather}(a_1, a_2), a_1 \approx a_1, a_2 \approx a_2\}$. $\mathrm{ApproxGround}(\mathcal{O}^{\dagger})$ returns $(\Pi, \mathcal{S}_{df}, \mathcal{S})$, where $\Pi = \{cl'_1, \ldots, cl'_{11}, cl'_{13}, \ldots, cl'_{30}\}$ (see $cl'_1, ..., cl'_{30}$ in Example 2), $\mathcal{S}_{df} = \emptyset$ and $\mathcal{S} = \{\{a_2, f(a_1)\}\}$. Consider again the CQ $Q : q(x) \leftarrow \mathsf{Man}(x) \wedge \mathsf{hasFather}(x, y)$ given in Example 3. By calling $\mathrm{DecompBasedEvaluate}(Q, \mathcal{A}, \Pi, \mathcal{S}_{df}, \mathcal{S})$, we get $Ans = \{a_1, a_2\}$ in line 1 and $Cands = \emptyset$ in line 2, thus we obtain the set of answers of Q in $\mathcal{O}$ without calling OWL DL reasoners.

To summarize, our decomposition-based approach to conjunctive query answering works as follows. In the offline phase, we compute the bounded entailment set $\mathcal{A}$ of $\Theta(\mathcal{O})$ and set $\mathcal{O}^{\dagger}$ as $\Theta(\mathcal{O}) \cup \{ax \in \mathcal{A} \mid ax \text{ contains only constants}\}$, then call $\mathrm{ApproxGround}(\mathcal{O}^{\dagger})$, obtaining $(\Pi, \mathcal{S}_{df}, \mathcal{S})$. In the online phase, for every coming CQ Q, we call $\mathrm{DecompBasedEvaluate}(Q, \mathcal{A}, \Pi, \mathcal{S}_{df}, \mathcal{S})$, obtaining all answers of Q.

4 Experimental Evaluation

We implemented the proposed approach in GNU C++, using MySQL as the back-end SQL engine. The implemented system[3] is called DecomBaR (abbr. for **Decom**position-**Ba**sed **R**easoner). It maintains ABox axioms and ground atoms in the approximate grounding in databases, maintains ground clauses in the approximate grounding in disk files, and calls Pellet [14] (version 2.0.0-rc6)[4] to evaluate CQs over extracted ontologies. All our experiments were conducted on a machine with 2GHz Intel Pentium Dual CPU and 2GB RAM, running Windows XP, where the maximum Java heap size was set to (max) 1312MB.

4.1 Experimental Setup

We conducted experiments on Lehigh University Benchmark (LUBM) [8] and University Ontology Benchmark (UOBM) [10] (including UOBM-Lite and UOBM-DL) ontologies. We used the above benchmark ontologies because they all come with benchmark CQs, which can provide a reasonable assessment on the effectiveness of our proposed approach. By LUBMn, UOBM-Lite$n$ and UOBM-DLn we respectively denote the instances of LUBM, UOBM-Lite and UOBM-DL that contain axioms about n universities. We specifically tested on LUBM1, LUBM10, UOBM-Lite1, UOBM-Lite10, UOBM-DL1 and UOBM-DL10, where the former two were generated by the LUBM data generator[5], and the latter four were all downloaded from the UOBM Website[6].

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

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

$\mathcal{O}$	$\lvert N_C\rvert$	$\lvert N_R\rvert$	$\lvert N_I\rvert$	$\lvert\mathcal{T}\rvert$	$\lvert\mathcal{A}\rvert$	$\lvert\mathcal{T}^\dagger\rvert$	$\lvert\mathcal{A}^\dagger\rvert$	Offline(sec)
LUBM1	59	16	50,253	94	100,543	105	100,543	17
LUBM10	59	16	629,568	94	1,272,575	105	1,272,575	219
UOBM-Lite1	51	43	95,010	130	245,864	151	245,864	101
UOBM-Lite10	51	43	820,208	130	2,096,973	151	2,096,973	1193
UOBM-DL1	112	44	96,081	151	260,540	210	260,587	242
UOBM-DL10	112	44	825,455	151	2,217,302	210	2,217,349	7103

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

4.2 Experimental Results

We compared DecomBaR with the original Pellet reasoner (simply Pellet) and the KAON2 reasoner (simply KAON2) on evaluating benchmark CQs given in [8,10]. We

[3] http://www.aifb.uni-karlsruhe.de/WBS/gqi/DecomBaR/
[4] http://clarkparsia.com/pellet/
[5] http://swat.cse.lehigh.edu/projects/lubm/index.htm
[6] http://www.alphaworks.ibm.com/tech/semanticstk/

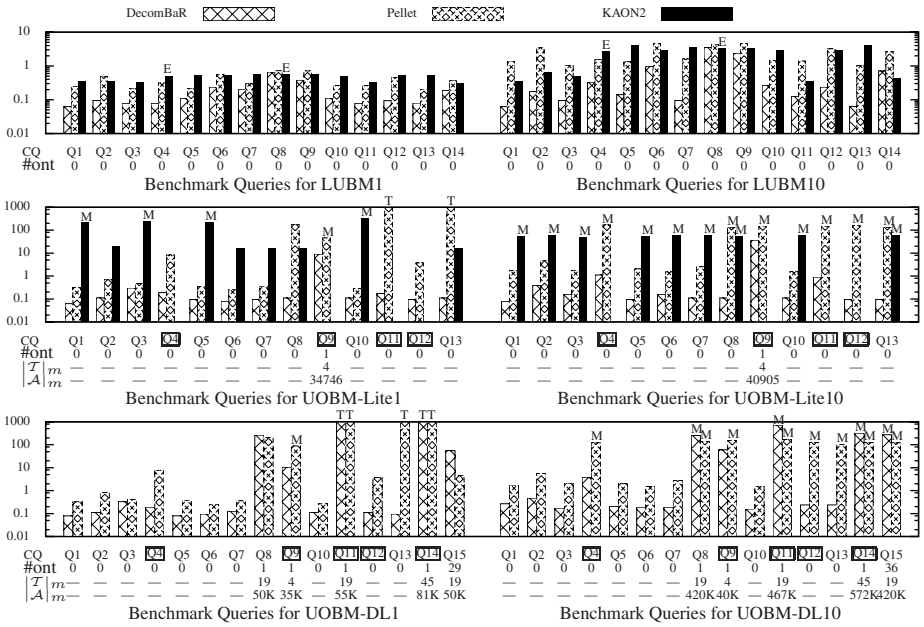

Fig. 3. The execution time (in seconds) for evaluating all benchmark CQs

implemented an interface to allow Pellet or KAON2 to read ABoxes from databases. We did not test KAON2 on UOBM-DLn ontologies nor CQs with non-distinguished variables as they are not supported by KAON2.

The execution time (in seconds) in the offline phase of DecomBaR is shown in the last column of Table 1. The results for evaluating every benchmark CQ are shown in Figure 3. The execution time about DecomBaR is the total evaluation time in the online phase, including the time for decomposing the propositional program compiled in the offline phase and the time for loading extracted ontologies to the called reasoner, waiting the called reasoner to return and handling the returned results. The execution time about Pellet or KAON2 is the time for query answering only, excluding the time for ontology loading and consistency checking (as we assume that the ontology is loaded and checked consistency offline) and the time for writing results.

Below the horizontal axis in Figure 3, "#ont" denotes the number of extracted ontologies over which Pellet is called to evaluate a test query, and "$|\mathcal{T}|_{max}$" (resp. "$|\mathcal{A}|_{max}$") denotes the maximum number of axioms in the TBox (resp. the ABox) of every extracted ontology. The name of a CQ is framed iff the CQ has non-distinguished variables. Above a bar, "M" means running out of memory after the displayed time, "T" means exceeding the time limit of 1000s, and "E" means that the set of computed answers is incorrect; we call any of these cases an unsuccessful evaluation. For every benchmark CQ that both DecomBaR and Pellet (resp. KAON2) successfully evaluate, the answers computed by DecomBaR and Pellet (resp. KAON2) coincide.

Comparing DecomBaR with Pellet, DecomBaR is more efficient than Pellet except for Q8 and Q15 on UOBM-DL1.[7] Such exception is due to that sometimes decomposition and interaction with the called reasoner introduce a significant overhead in the execution of DecomBaR. However, when DecomBaR does not generate any candidate answer (i.e. when #ont $= 0$), DecomBaR works very fast because it only needs to extract explicit and candidate answers by accessing the database through a SQL query. For example, DecomBaR spends about 0.1s for evaluating Q8 on both UOBM-Lite1 and UOBM-Lite10, while for evaluating the same CQ Pellet spends about 180s on UOBM-Lite1 and runs out of memory on UOBM-Lite10. Moreover, DecomBaR is more scalable than Pellet against increasing size of ABoxes. This is because accessing databases through SQL queries is relatively scalable (in case #ont $= 0$) and extracted ontologies could have similar sizes for different size of ABoxes (in case #ont > 0). For example, the UOBM-Lite benchmark query Q9 has an individual in the query body, which forces $\mathrm{Inst}_{\mathrm{CQ}}$ (defined in Subsection 3.4) to return similar ground clauses and then forces the extracted ontologies to have similar sizes for UOBM-Lite1 and UOBM-Lite10.

Comparing DecomBaR with KAON2, DecomBaR is generally more efficient (esp. for UOBM-Lite$n$ ontologies, by orders of magnitude more efficient) than KAON2. Moreover, the scalability of DecomBaR is comparable with that of KAON2 on LUBMn ontologies, and is much better than that of KAON2 on UOBM-Lite$n$ ontologies. This shows that DecomBaR is much more scalable than KAON2 against increasing complexity of TBoxes. It should also be mentioned that DecomBaR supports more expressive CQs than KAON2 does. In particular, KAON2 may not correctly evaluate CQs involving datatypes (e.g. the LUBM benchmark queries Q4 and Q8); this is a limitation of the resolution-based query mechanism [11] exploited in KAON2.

5 Conclusion and Future Work

In this paper, we have proposed a decomposition-based approach to optimize conjunctive query answering in OWL DL ontologies. The basic idea of the approach is to evaluate a CQ with the help of a precompiled propositional program: it computes explicit answers first and then computes other answers over separate ontologies that are extracted from the precompiled propositional program. Experimental results demonstrate the advantages of the proposed approach.

The proposed approach still has some limitations. First, it only works well on ontologies that rarely change as the offline phase is somewhat costly. We plan to upgrade the compilation method to an incremental one to copy with ontology changes. Second, the approach fails when some extracted ontologies are still too large to be handled by the called reasoner. This is the reason why our implemented system DecomBaR does not successfully evaluate six benchmark CQs in our experiments (see Figure 3). We will tackle this limitation by exploiting the idea of summarization [2,3].

[7] Here we do not compare DecomBaR and Pellet for those CQs that both DecomBaR and Pellet do not successfully evaluate.

Acknowledgments. Thanks all anonymous reviewers for their useful comments. Jianfeng Du is supported in part by NSFC grants 60673103 and 70801020. Yi-Dong Shen is supported in part by NSFC grants 60673103, 60721061, and by the 863 Program.

References

1. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: $\mathrm{DL-Lite}$: Tractable description logics for ontologies. In: Proc. of AAAI 2005, pp. 602–607 (2005)
2. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Ma, L.: Scalable semantic retrieval through summarization and refinement. In: Proc. of AAAI 2007, pp. 299–304 (2007)
3. Dolby, J., Fokoue, A., Kalyanpur, A., Ma, L., Schonberg, E., Srinivas, K., Sun, X.: Scalable grounded conjunctive query evaluation over large and expressive knowledge bases. In: Sheth, A.P., Staab, S., Dean, M., Paolucci, M., Maynard, D., Finin, T., Thirunarayan, K. (eds.) ISWC 2008. LNCS, vol. 5318, pp. 403–418. Springer, Heidelberg (2008)
4. Eiter, T., Leone, N., Mateis, C., Pfeifer, G., Scarcello, F.: A deductive system for nonmonotonic reasoning. In: Proc. of LPNMR 1997, pp. 364–375 (1997)
5. Fitting, M.: First-order Logic and Automated Theorem Proving, 2nd edn. Springer, New York (1996)
6. Cuenca Grau, B., Parsia, B., Sirin, E., Kalyanpur, A.: Modularity and web ontologies. In: Proc. of KR 2006, pp. 198–209 (2006)
7. Guo, Y., Heflin, J.: A scalable approach for partitioning OWL knowledge bases. In: Proc. of SSWS 2006, pp. 47–60 (2006)
8. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics 3(2–3), 158–182 (2005)
9. Kazakov, Y., Motik, B.: A resolution-based decision procedure for $\mathcal{SHOIQ}$. Journal of Automated Reasoning 40(2-3), 89–116 (2008)
10. 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)
11. Motik, B.: Reasoning in Description Logics using Resolution and Deductive Databases. PhD thesis, Univesität karlsruhe, Germany (January 2006)
12. Motik, B., Sattler, U., Studer, R.: Query answering for OWL-DL with rules. Journal of Web Semantics 3(1), 41–60 (2005)
13. Pan, J.Z., Thomas, E.: Approximating OWL-DL ontologies. In: Proc. of AAAI 2007, pp. 1434–1439 (2007)
14. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Journal of Web Semantics 5(2), 51–53 (2007)
15. Stuckenschmidt, H., Klein, M.: Structure-based partitioning of large concept hierarchies. In: McIlraith, S.A., Plexousakis, D., van Harmelen, F. (eds.) ISWC 2004. LNCS, vol. 3298, pp. 289–303. Springer, Heidelberg (2004)