Towards Scalable and Complete Query Explanation with OWL 2 EL Ontologies

Zhe Wang
School of Info. & Comm. Tech.
Griffith University, Australia
zhe.wang@griffith.edu.au

Mahsa Chitsaz
School of Info. & Comm. Tech.
Griffith University, Australia
mahsa.chitsaz@griffithuni.edu.au

Kewen Wang
School of Info. & Comm. Tech.
Griffith University, Australia
k.wang@griffith.edu.au

Jianfeng Du
Cisco School of Informatics
Guangdong University of Foreign Studies, China
jfdu@mail.gdufs.edu.cn

ABSTRACT

Ontology-mediated data access and management systems are rapidly emerging. Besides standard query answering, there is also a need for such systems to be coupled with explanation facilities, in particular to explain missing query answers (i.e. desired answers of a query which are not derivable from the given ontology and data). This support is highly demanded for debugging and maintenance of big data, and both theoretical results and algorithms proposed. However, existing query explanation algorithms either cannot scale over relative large data sets or are not guaranteed to compute all desired explanations. To the best of our knowledge, no existing algorithm can efficiently and completely explain conjunctive queries (CQs) w.r.t. $\mathcal{ELH}_\bot$ ontologies. In this paper, we present a hybrid approach to achieve this. An implementation of the proposed query explanation algorithm has been developed using an off-the-shelf Prolog engine and a datalog engine. Finally, the system is evaluated over practical ontologies. Experimental results show that our system scales over large data sets.

Categories and Subject Descriptors

I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*Representations*; H.4.0 [**Information Systems Applications**]: General

Keywords

Abductive reasoning; description logics; conjunctive query.

P

1. INTRODUCTION

One promising solution to the effective access and management of the ever-growing sea of digital information has been to use computers in setting up databases of information, completed with ontologies that assist in organization and access. In such an ontology-mediated data access framework, ontologies are used as virtual schemas over data sets to enrich query answering with ontological knowledge through logical reasoning, while making use of practical database management systems to scale over very large data sets. The standardised web ontology language OWL 2 and its three profiles, EL, QL and RL[1], are based on description logics (DLs), and conjunctive queries (CQs) are used as a vital querying tool.

Besides standard query answering, explanation services are often required by end-users to understand why certain answers are derived or missing. For instance, suppose ontology $\mathcal{O}$ specifies that a PhD student is a student, i.e., $\mathsf{PhD} \sqsubseteq \mathsf{Student}$, each PhD student has some supervisor, i.e., $\mathsf{PhD} \sqsubseteq \exists \mathsf{hasSupervisor}.\mathsf{Person}$, Tom is a PhD student and Mary is a student, i.e., $\mathsf{PhD}(\mathsf{Tom})$ and $\mathsf{Student}(\mathsf{Mary})$. For the query to retrieve those individuals who have supervisors, which can be expressed as $\mathcal{Q}(x) = \exists y.\mathsf{hasSupervisor}(x, y)$, "Tom" is an answer whereas "Mary" is not. One explanation for "Mary" being missing is that $\mathsf{PhD}(\mathsf{Mary})$ is absent. Such explanation services are critical for knowledge management to identify specific pieces of the knowledge that need to be revised. For example, the observation that Mary should have a supervisor indicates that the current knowledge is incomplete and $\mathsf{PhD}(\mathsf{Mary})$ could be added.

On the other hand, however, the implementation of existing OWL reasoners does not support such an explanation mechanism even through tracing the execution of query answering. For this reason, extensive efforts have been devoted to equip ontology-based systems with various explanation facilities [22, 23, 4, 15, 24, 6]. While early research was mainly on explaining derived answers (a.k.a. positive answers, like "Tom" in the above example), explaining missing answers (a.k.a. negative answers, like "Mary") has attracted much attention lately [3, 6]. Such an explanation facility is essential to understand why the ontology-based system fails

[1] http://www.w3.org/TR/owl2-profiles/

to derive certain desired answers, and to enrich incomplete data according to user observations. Such a query explanation problem is usually formalised as a problem of abduction in AI, called *query abduction*.

Existing research efforts showed that query abduction is challenging. First, while ontology-based reasoning empowers query answering, it also complicates query abduction. The computational complexity of query abduction is indeed higher than query answering [6]. Also, efficiency of abduction implementation has not been a central concern in AI but it is essential for ontology-based systems, and it is highly non-trivial to develop a scalable algorithm for query abduction. Furthermore, existing query abduction approaches are either incapable to scale over large data sets or incomplete in explaining CQs (i.e., they may fail to compute certain explanations, see a detailed discussion in Section 2). In this paper, we demonstrate that *it is possible to achieve both scalability and completeness in one framework*, by presenting a complete and scalable query abduction method for $\mathcal{ELH}_\bot$ ontologies. We choose $\mathcal{ELH}_\bot$ because it underlies the OWL 2 EL profile, is expressive enough for many practical ontologies such as SNOMED CT, and is a core fragment of most expressive DLs. CQ answering for $\mathcal{ELH}_\bot$ ontologies has attracted intensive interest lately [19, 26].

The main contributions of the paper include: (1) We reduce a query abduction problem in $\mathcal{ELH}_\bot$ to an equivalent one in datalog, and prove the soundness and completeness. (2) We introduce a method to extract (often very small) modules of the data through data summarisation, and we introduce an advanced pruning strategy to couple the basic backward chaining procedure in the search for solutions. (3) We implemented a prototype system that integrates highly optimised Prolog and datalog engines, and our evaluation shows that the algorithm is capable of handling query abduction involving around 10 million facts.

2. RELATED WORKS

Several approaches to query abduction have been proposed, which vary according to their underlying DLs and types of observations. Some of these focused on expressive DLs and observations formulated by simple atomic queries or ground CQs [17, 14, 21, 9]. These approaches typically use tableau-based methods, which however, could rarely scale over relative large data sets. One exception is the approach by Du et al. [9], which we will discuss in detail later.

Query abduction for CQs with existentially quantified variables, which will be referred to as *general CQs*, has been investigated only for light-weight DLs underlying OWL 2 RL and QL profiles. Borgida et al. [3] are among the first who advocated to study the explanation of negative answers, and they focused on OWL 2 QL. Du et al. [10] have proposed a practical approach in OWL 2 RL, and an OWL 2 RL ontology can be directly translated into a datalog program. Calvanese et al. [6] investigated the computational complexity of query abduction in OWL 2 QL. To the best of our knowledge, we are the first to study query abduction for general CQs in OWL 2 EL.

Du et al. [9, 8] were among the first to tackle the scalability issue, yet those approaches still suffer from some shortcomings. The method proposed in [9] is incomplete for general CQs, that is, their method is unable to compute all minimal explanations for some general CQs. This is mainly due to the ontology transformation adopted in that approach, which discard rules with existentially quantified variables in the head. Such rules will be referred to as *genuine existential rules*. For example, consider the ontology $\mathcal{O}$ in Section 1, the algorithm developed in [9] discards the axiom $\mathsf{PhD} \sqsubseteq \exists \mathsf{hasSupervisor.Person}$ and thus cannot obtain the explanation $\mathsf{PhD(Mary)}$. Also, the system developed in [9] still has difficulty in processing large data sets (see Section 5 for details). In [8], a notion of a representative explanation is introduced, which is informally a schema of multiple explanations. While a scalable algorithm is developed, the approach works only for first-order (FO) rewritable ontologies, and ontologies expressed in popular ontology languages, such as OWL 2 EL which we look into, may not be FO rewritable. It is challenging to efficiently compute representative explanations for ontologies that may not be FO rewritable. To see this, consider an ontology consisting of one axiom $\exists R.A \sqsubseteq A$. For the observation $A(a)$, there would an infinite number of representative explanations of the form $\{R(a, u_1), R(u_1, u_2), \ldots, R(u_{n-1}, u_n), A(u_n)\}$.

In summary, compared to existing query abduction approaches, our approach is the only one that enjoys all the following three properties: (1) In contrast to [17, 14, 21], our approach scales over large data sets. (2) In contrast to [9], our approach is complete for general CQs in a sense that all (minimal) explanations can be computed. (3) In contrast to [3, 10, 6, 8], our approach applies to $\mathcal{ELH}_\bot$ ontologies that are not necessarily FO rewritable.

3. PRELIMINARIES

We use standard notions of first-order (FO) constants, variables, terms, substitutions, predicates, atoms, (ground) formulae, and sentences. All atoms and formulae considered in the paper are function free. A *fact* is a ground atom. For a formula ϕ, with $\phi(\vec{x})$ we denote that $\vec{x}$ are the free variables in ϕ. For a (set of) formula(e) ϕ, $pred(\phi)$, $term(\phi)$, $const(\phi)$ denote the set of predicates, terms, and constants occurring in ϕ. First-order entailment are defined as usual. For convenience, we sometimes treat a conjunction of atoms as a set consisting of the atoms.

3.1 Description Logic and Query Answering

Consider countably infinite and mutually disjoint sets N_C, N_R, N_I, and N_V of concept names (i.e., unary predicates), role names (i.e. binary predicates), individuals (i.e. constants), and variables respectively. In $\mathcal{ELH}_\bot$, a *concept description* (or simply *concept*) C is defined inductively using the following constructors:

$$C := \top \mid \bot \mid A \mid C_1 \sqcap C_2 \mid \exists R.C,$$

where $A \in N_C$, $R \in N_R$, and C_1 and C_2 are concepts. A *TBox* is a finite set of axioms of the forms $C \sqsubseteq D$, and $R \sqsubseteq S$, where C, D are both concepts and $R, S \in N_R$. An *ABox* is a finite set of facts of the forms $A(a)$ and $R(a, b)$, where $A \in N_C$, $R \in N_R$ and $a, b \in N_I$. A *knowledge base* (KB) $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ consists of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. The semantics of $\mathcal{ELH}_\bot$ is defined as usual in terms of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, and we refer to [1] for details.

An $\mathcal{ELH}_\bot$ TBox is *normalized* if it consists of only axioms of the forms $A \sqsubseteq B$, $A_1 \sqcap \cdots \sqcap A_n \sqsubseteq A$, $A_1 \sqsubseteq \exists R.A$, $\exists R.A_1 \sqsubseteq A$, and $R \sqsubseteq S$, where $A_{(i)} \in N_C \cup \{\top\}$, $B \in N_C \cup \{\top, \bot\}$, and $R, S \in N_R$. An $\mathcal{ELH}_\bot$ TBox can be transformed in polynomial time to a normalized TBox that is equivalent to

the TBox [1]. Without loss of generality, we consider only TBoxes in normal form.

An *FO query* $\mathcal{Q}(\vec{x})$ is a first-order formula (with free variables $\vec{x}$, we will sometimes write $\mathcal{Q}$ for simplicity). It is called *boolean* if $\vec{x}$ is empty. For a tuple of constants $\vec{a}$ with the same arity as $\vec{x}$, we write $\mathcal{Q}(\vec{a})$ to denote the Boolean FO query obtained from $\mathcal{Q}(\vec{x})$ by substituting the variables $\vec{x}$ with constants $\vec{a}$. For a TBox $\mathcal{T}$, an ABox $\mathcal{A}$, and a FO query $\mathcal{Q}(\vec{x})$, a tuple of constants $\vec{a}$ is a *certain answer* (or simply an *answer*) to $\mathcal{Q}(\vec{x})$ over $\mathcal{T}$ and $\mathcal{A}$, if the arity of $\vec{a}$ agrees with that of $\vec{x}$ and $\mathcal{T} \cup \mathcal{A} \models \mathcal{Q}(\vec{a})$. Otherwise, if $\mathcal{T} \cup \mathcal{A} \not\models \mathcal{Q}(\vec{a})$ then $\vec{a}$ is a *negative answer*.

A conjunctive query (CQ) $\mathcal{Q}(\vec{x})$ is of the form $\exists \vec{y}.\phi(\vec{x}, \vec{y})$, where ϕ is a conjunction of atoms with predicates and terms from $N_C \cup N_R$ and $N_I \cup N_V$, respectively. Variables $\vec{x}$ are *answer variables* and $\vec{y}$ are *quantified variables*. A *ground CQ* is a CQ where $\vec{y}$ is empty, and an *atomic query* is a ground CQ where $\phi(\vec{x}, \vec{y})$ consists of a single atom.

3.2 Query Abduction

We consider the *query abduction problem (QAP)* where a CQ and an answer tuple are given as the *observation* and a solution to the problem (or equivalently an explanation to the query answer) is a set of facts [6]. Formally, given a background ontology consisting of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, taking an instantiated CQ (or a BCQ) $\mathcal{Q}(\vec{a})$ as the *observation*, a solution to the query abduction problem is a (possibly empty) set $\mathcal{E}$ of facts such that $\mathcal{E}$ is consistent with $\mathcal{T} \cup \mathcal{A}$ and $\vec{a}$ is an answer to $\mathcal{Q}(\vec{x})$ over $\mathcal{T} \cup \mathcal{A}$ (only) together with $\mathcal{E}$. In classical abduction setting, to reduce the search space and retrieve only relevant solutions, the predicates allowed in the solutions are often pre-specified, which are called *abducibles*. Furthermore, for the ease of understanding, solutions should be as succinct as possible, and the minimal solutions (w.r.t. set containment) are desired. The formal definition of QAP and a solution to the QAP is given as follows.

Definition 1 (QAP). *An instance of a* query abduction problem (QAP) *is a five tuple* $\mathcal{P} = \langle \mathcal{T}, \mathcal{A}, \mathcal{Q}(\vec{a}), \Sigma, \Delta \rangle$, *where* $\mathcal{T}$ *is an* $\mathcal{ELH}_\bot$ *TBox,* $\mathcal{A}$ *is an* $\mathcal{ELH}_\bot$ *ABox,* $\mathcal{Q}(\vec{x})$ *is a CQ,* $\vec{a}$ *a tuple of constants with the same arity as* $\vec{x}$, $\Sigma \subseteq N_C \cup N_R$ *is a finite set of predicates, and* Δ *is a finite set of constants. BCQ* $\mathcal{Q}(\vec{a})$ *is the* observation, Σ *is the set of* abducibles, *and* Δ *is the* domain.

A solution $\mathcal{E}$ *to the QAP* $\mathcal{P}$ *is a set of facts such that*

1. $pred(\mathcal{E}) \subseteq \Sigma$ *and* $const(\mathcal{E}) \subseteq \Delta$;
2. $\mathcal{T} \cup \mathcal{A} \cup \mathcal{E} \models Q(\vec{a})$;
3. $\mathcal{T} \cup \mathcal{A} \cup \mathcal{E} \not\models \bot$; *and*
4. $\mathcal{E} \not\models \mathcal{Q}(\vec{a})$;

Conditions 3 and 4 are called non-triviality *conditions for a solution. Let* $\mathsf{sol}(\mathcal{P})$ *be the set of solutions to* $\mathcal{P}$. *Moreover,* $\mathcal{E}$ *is* minimal *if there is no solution* $\mathcal{E}' \in \mathsf{sol}(\mathcal{P})$ *s.t.* $\mathcal{E}' \subset \mathcal{E}$.

In contrast to the definitions in [6, 8], the above definition has a finite domain Δ as an additional attribute, and Δ contains all the constants occurring in a solution. In [6, 8], a solution can contain any constants, including arbitrarily many fresh constants not present in the initial ABox. In this case infinitely many (minimal) solutions often exist for a QAP, and hence it is impossible to compute all the (minimal) solutions. Even if *representative solutions* are considered [8], which ignore renaming of fresh constants and are minimal up to substitution, still infinitely many such representative solutions may exist in general, as shown in Section 2. On the other hand, allowing uncontrolled introduction of fresh constants to QAP solutions is arguably of mere theoretical interest. In query explanation, a finite number of constants are often sufficient. Hence, we assume a finite domain Δ in a QAP, which can either be specified by the user, or by default set to be the set of all the constants in the initial ABox $\mathcal{A}$ and/or the constants in the observation Q. Given that the predicates and constants allowed in a solution are both finite and pre-specified, the total number of solutions to a QAP is always finite (even for ontologies that are not FO rewritable).

3.3 Existential Rules

Since our new approach translates a QAP for $\mathcal{ELH}_\bot$ into a problem of abduction in programs of existential rules, we recall some basics of existential rules in this subsection. Formally, an *existential rule* [5, 2] is a sentence of the form

$$\forall \vec{x}.\forall \vec{y}.[\phi(\vec{x}, \vec{y}) \rightarrow \exists \vec{z}.\psi(\vec{x}, \vec{z})]$$

where $\phi(\vec{x}, \vec{y})$ and $\psi(\vec{x}, \vec{z})$ are conjunctions of function-free atoms and $\vec{x}$, $\vec{y}$ and $\vec{z}$ are pairwise disjoint. For convenience of presentation, universal quantifiers are often omitted. Formula ϕ is the *body* and formula ψ is the *head* of the rule. It is shown in [5] that $\mathcal{ELH}_\bot$ can be captured by existential rules. In particular, given an $\mathcal{ELH}_\bot$ TBox $\mathcal{T}$, $\mathcal{T}$ can be transformed into an existential program $\mathcal{R}_\mathcal{T}$ that is semantically equivalent to $\mathcal{T}$ in first-order logic. Query answering and query abduction in programs of existential rules are defined in the same way as in $\mathcal{ELH}_\bot$, and for query abduction of existential rules, we extend the definition QAP to allow $\mathcal{Q}$ in observations to be a FO query.

A *datalog* rule is an existential rule whose head $\psi(\vec{x}, \vec{z})$ consists of a single atom and $\vec{z}$ is empty. In logic programming, the semantics of a datalog program $\mathcal{D}$ is given by the least Herbrand model. For a datalog program $\mathcal{D}$, let $\mathcal{M}$ be the least Herbrand model of $\mathcal{D} \cup \mathcal{F}$, it is well known that for a tuple $\vec{a}$, $\mathcal{D} \cup \mathcal{F} \models \mathcal{Q}(\vec{a})$ if and only if $\mathcal{M}$ satisfies $\mathcal{Q}(\vec{a})$.

Answering CQs w.r.t. a datalog program can also be achieved through a backward chaining procedure based on SLD-resolution (with *tabling*, a technique to control cycles and to guarantee termination), which we formally present as follows. A *goal* is a conjunction of atoms $\alpha_1 \wedge \ldots \wedge \alpha_m$. The *SLD-resolution rule* takes as premises a goal and a datalog rule, and it produces a new goal as follows

$$\frac{\alpha_1 \wedge \ldots \wedge \alpha_m, \;\; \phi \rightarrow \beta}{\sigma(\alpha_2) \wedge \ldots \wedge \sigma(\alpha_m) \wedge \sigma(\phi)}$$

where σ is the most general unifier of α_1 and β. An *SLD-proof* of a goal G_0 w.r.t. a datalog program $\mathcal{D}$ and a set $\mathcal{F}$ of facts is a sequence of goals $G_0, \ldots, G_n$, where $G_n \subseteq \mathcal{F}$ and each G_{i+1} is obtained from G_i by a single SLD-resolution. SLD-resolution is sound and complete for datalog: for a datalog program $\mathcal{D}$, a set $\mathcal{F}$ of fact, a CQ $\mathcal{Q}(\vec{x}) = \exists \vec{y}.\phi(\vec{x}, \vec{y})$ and a tuple $\vec{a}$ of constants, $\mathcal{D} \cup \mathcal{F} \models Q(\vec{a})$ iff there exists an SLD-proof of the goal $\phi(\vec{a}, \vec{y})$ w.r.t. $\mathcal{D}$ and $\mathcal{F}$. In this case, we call it an SLD-proof of $\mathcal{Q}(\vec{a})$ *w.r.t.* $\mathcal{D}$ and $\mathcal{F}$. For a rule $r \in \mathcal{D}$, denote $r \in SLD(\mathcal{Q}, \mathcal{D}, \mathcal{F})$ if r is involved in an SLD-proof of $\mathcal{Q}$ *w.r.t.* $\mathcal{D}$ and $\mathcal{F}$; and for a fact α, denote

$\alpha \in SLD(\mathcal{Q}, \mathcal{D}, \mathcal{F})$ if there exists an SLD-proof $G_0, \ldots, G_n$ of $\mathcal{Q}(\vec{a})$ such that $\alpha \in G_n$.

4. OUR APPROACH

In our query abduction setting, observations may contain existentially quantified variables and ontologies may have genuine existential rules. It is well observed in query answering and abduction literature that the sophisticated interaction between general CQs and genuine existential rules often post great challenges to computation. Our approach to address the scalability and completeness issues in query abduction combines several novel techniques for ontology transformation, query rewriting, data summarisation and module extraction, and our system integrates a datalog engine, a Prolog engine and an RDBMS. The key feature of our approach is that it reduces ontological query abduction in $\mathcal{ELH}_\bot$ to rule-based reasoning and thus can make use of off-the-shelf rule-based systems (instead of less scalable OWL reasoners). Also, it is sound and complete in computing all the minimal solutions, and at the same time scales over large data sets.

4.1 Overview

In a nutshell, given a QAP $\mathcal{P} = \langle \mathcal{T}, \mathcal{A}, \mathcal{Q}, \Sigma, \Delta \rangle$ in $\mathcal{ELH}_\bot$ our approach computes all its minimal solutions in the following steps.

Ontology transformation and observation rewriting: Our first step is to compute a datalog approximation $\mathcal{D}_\mathcal{T}$ of the initial TBox $\mathcal{T}$ and a rewriting $\mathcal{Q}^*$ of the initial observation. In the construction of $\mathcal{D}_\mathcal{T}$, instead of eliminating all the axioms in $\mathcal{T}$ that cannot be equivalently translated into datalog, we replace the existentially quantified variables with fresh constants. For approximated reasoning, we could replace $\mathcal{T}$ with $\mathcal{D}_\mathcal{T}$ and compute solutions for the new QAP using a resolution-based procedure. However, the approximation may introduce incorrect solutions, and hence $\mathcal{Q}^*$ is constructed by adding filtering conditions to $\mathcal{Q}$, which will invalidate certain resolution proofs and hence filter out the corresponding (spurious) solutions.

Module Extraction: In the second step, we extract (small) modules of each component (except for the observation) of the QAP that are used for query abduction, and we achieve this through data summarisation. First, we obtain a compact summary $\delta(A)$ of the initial ABox $\mathcal{A}$ by grouping and merging individuals in $\mathcal{A}$. Then, using $\delta(A)$ together with $\mathcal{D}_\mathcal{T}$ and $\mathcal{Q}^*$, we extract modules of $\mathcal{D}_\mathcal{T}$, $\mathcal{A}$, Σ and Δ that are relevant to the query abduction. For example, two (small) modules $\mathcal{A}_\mathcal{Q}$ and $\mathcal{A}_\bot$ of $\mathcal{A}$ are extracted, one for solution generation and one for solution verification. The module $\mathcal{A}_\mathcal{Q}$ is obtained through a resolution-based procedure and by tracking the facts in $\delta(A)$ that participate in some resolution proof of $\mathcal{Q}$.

Solution generation and verification with pruning: In the final step, we use the modules extracted in the previous step to generate and verify solution candidates. We employ a Prolog engine in this step and the previous step to perform resolution, by encoding the corresponding QAPs in Prolog. Besides a systematic search for solutions via resolution, an advanced pruning strategy is also applied. That is, once a solution $\mathcal{E}$ is computed, we exploit it further to generate other candidates $\mathcal{E}'$, by applying hierarchical knowledge from the TBox $\mathcal{T}$. In this the resolution proof for constructing $\mathcal{E}'$ is often pruned. Verifications of solution candidates are performed with a datalog engine. Hence, our approach consists of a combination of backward chaining (for solution generation) and forward chaining (for verification).

4.2 Ontology Transformation

For an $\mathcal{ELH}_\bot$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ with a normalized TBox, we will first transform $\mathcal{T}$ into a datalog program $\mathcal{D}_\mathcal{T}$ in a way similar to [26]. In particular, $\mathcal{D}_\mathcal{T}$ consists of the rules in Table 1, plus one rule $A(x) \to \top(x)$ for each concept name A occurring in $\mathcal{T}$ and two rules $R(x,y) \to \top(x)$ and $R(x,y) \to \top(y)$ for each role name R occurring in $\mathcal{T}$, where we use $\top$ as a special unary predicate in datalog. These datalog rules capture the semantics of $\top$ in $\mathcal{ELH}_\bot$.

$\mathcal{ELH}_\bot$ axiom		datalog rule
$A \sqsubseteq \bot$	$\rightsquigarrow$	$A(x) \to \bot(x)$ $\bot$ is a unary predicate.
$A_1 \sqsubseteq A$	$\rightsquigarrow$	$A_1(x) \to A(x)$
$A_1 \sqcap A_2 \sqsubseteq A$	$\rightsquigarrow$	$A_1(x) \wedge A_2(x) \to A(x)$
$A_1 \sqsubseteq \exists R.A$	$\rightsquigarrow$	$A_1(x) \to R(x, c_{R,A})$ $A_1(x) \to A(c_{R,A})$ $c_{R,A}$ is a fresh constant.
$\exists R.A_1 \sqsubseteq A$	$\rightsquigarrow$	$R(x,y) \wedge A_1(y) \to A(x)$
$R \sqsubseteq S$	$\rightsquigarrow$	$R(x,y) \to S(x,y)$

Table 1: Transformation from $\mathcal{ELH}_\bot$ to datalog.

The following example illustrates the transformations.

EXAMPLE 1. *Let TBox $\mathcal{T}$ consist of the following three axioms:*

T1) $\mathsf{RA} \sqsubseteq \mathsf{Person} \sqcap \exists \mathsf{works.RG}$
T2) $\mathsf{Student} \sqsubseteq \mathsf{Person} \sqcap \exists \mathsf{takes.Course}$
T3) $\mathsf{Person} \sqcap \exists \mathsf{takes.Course} \sqsubseteq \mathsf{Student}$

*Axiom T1 says that a research assistant (*RA*) is a person and she works in some research group (*RG*), and axioms T2 and T3 say that a student is a person who takes some courses.*

The corresponding existential program $\mathcal{R}_\mathcal{T}$ has five rules R1–R5:

R1) $\mathsf{RA(x)} \to \mathsf{Person(x)}$
R2) $\mathsf{RA(x)} \to \exists \mathsf{y.(works(x,y)} \wedge \mathsf{RG(y))}$
R3) $\mathsf{Student(x)} \to \mathsf{Person(x)}$
R4) $\mathsf{Student(x)} \to \exists \mathsf{y.(takes(x,y)} \wedge \mathsf{Course(y))}$
R5) $\mathsf{Person(x)} \wedge \mathsf{takes(x,y)} \wedge \mathsf{Course(y)} \to \mathsf{Student(x)}$

where R5 is an abbreviation of two rules and the auxiliary concept introduced via normalization is omitted for brevity.

Then, $\mathcal{R}_\mathcal{T}$ is transformed into the datalog program $\mathcal{D}_\mathcal{T}$ containing R1, R3, R5, and the following datalog rules S1–S4:

S1) $\mathsf{RA(x)} \to \mathsf{works(x, c_1)}$ *S2)* $\mathsf{RA(x)} \to \mathsf{RG(c_1)}$
S3) $\mathsf{Student(x)} \to \mathsf{takes(x, c_2)}$
S4) $\mathsf{Student(x)} \to \mathsf{Course(c_2)}$

where $\mathsf{c_1}, \mathsf{c_2}$ are fresh constants, S1 and S2 are transformed from R2, and S3 and S4 from R4.

The following result states that the datalog approximation of each KB is complete for query abduction. That is, each solution to the QAP *w.r.t.* KB $\mathcal{T} \cup \mathcal{A}$ is a solution to the corresponding QAP *w.r.t.* datalog program $\mathcal{D}_\mathcal{T} \cup \mathcal{A}$.

PROPOSITION 1. *Let $\mathcal{P} = \langle \mathcal{T}, \mathcal{A}, \mathcal{Q}(\vec{a}), \Sigma, \Delta \rangle$ be a QAP in $\mathcal{ELH}_\bot$, and $\mathcal{P}'$ be the QAP obtained from $\mathcal{P}$ by replacing $\mathcal{T}$ with $\mathcal{D}_\mathcal{T}$. Then, $\mathsf{sol}(\mathcal{P}) \subseteq \{\mathcal{E} \in \mathsf{sol}(\mathcal{P}') \mid \mathcal{D}_\mathcal{T} \cup \mathcal{A} \cup \mathcal{E} \not\models \exists x.\bot(x)\}$.*

However, the converse of Proposition 1 may not hold, that is, the transformation is not necessarily sound, and the following example illustrates it.

EXAMPLE 2. *Let TBox $\mathcal{T} = \{A \sqsubseteq \exists R.A\}$. Then, the datalog program $\mathcal{D}_\mathcal{T}$ contains the following datalog rules $A(x) \rightarrow R(x, c)$ and $A(x) \rightarrow A(c)$. Take $\Sigma = \{A\}$ and $\Delta = \{a, b\}$.*

1. *Consider CQ $\mathcal{Q}_1(x, y) = \exists z.[R(x, z) \wedge R(y, z)]$ and QAP $\mathcal{P}_1 = \langle \mathcal{T}, \emptyset, \mathcal{Q}_1(a, b), \Sigma, \Delta \rangle$. Then, the QAP has no solution, i.e., $\mathsf{sol}(\mathcal{P}_1)$ is empty. However, let $\mathcal{P}_1'$ be the QAP obtained from $\mathcal{P}_1$ by replacing $\mathcal{T}$ with $\mathcal{D}_\mathcal{T}$. Then, $\{A(a), A(b)\}$ is a solution to $\mathcal{P}_1'$.*
2. *Consider CQ $\mathcal{Q}_2(x) = \exists y.[R(x, y) \wedge R(y, y)]$ and QAP $\mathcal{P}_2 = \langle \mathcal{T}, \emptyset, \mathcal{Q}_2(a), \Sigma, \Delta \rangle$. Again, $\mathsf{sol}(\mathcal{P}_2)$ is empty. Yet for $\mathcal{P}_2'$ the QAP obtained from $\mathcal{P}_2$ by replacing $\mathcal{T}$ with $\mathcal{D}_\mathcal{T}$, $\{A(a)\}$ is a solution to $\mathcal{P}_2'$.*

The cause of incorrect solutions is that the above transformation from a TBox to a datalog program enforces fork-shaped and cyclic structures in the models by unifying the instantiation of existentially quantified variables in each rule.

4.3 Observation Rewriting

To retain the soundness of our TBox transformation, we rewrite observations to filter out spurious solutions. In particular, we rewrite a CQ $\mathcal{Q}(\vec{x})$ into a FO query $\mathcal{Q}^*(\vec{x})$ such that for any observation $\mathcal{Q}(\vec{a})$, the solutions to a QAP $\mathcal{P}$ are exactly those to the QAP $\mathcal{P}'$ obtained by replacing the TBox $\mathcal{T}$ in $\mathcal{P}$ with $\mathcal{D}_\mathcal{T}$. Our rewriting approach is adapted from [20], where a rewriting method is used to filter out spurious query matching. While the rewriting in [20] is shown to be sufficient for query answering, it was unknown whether it works for abduction. Also, it is worth noting that our rewriting is simpler than that of [20], as we use a different program transformation from theirs. In what follows, we show that the rewriting technique can be used in abduction and in particular, our transformation-based procedure and the adapted observation rewriting form a sound and complete algorithm for query abduction in $\mathcal{ELH}_\bot$.

We first introduce unary predicates Ind and Aux to assert respectively the individuals occurring in the initial ABox and the auxiliary constants (i.e., those fresh constants generated in the approximation phase shown in Table 1). Then, to prevent spurious solutions introduced by fork-shaped or cyclic structures in the least Herbrand model of $\mathcal{D}_\mathcal{T}$ (but not in models of $\mathcal{T}$), for a CQ $\mathcal{Q}$, we define the equivalence relation $\sim$ over the terms (i.e., variables and constants) in $\mathcal{Q}$ as in [20]. Formally, $\sim \subseteq term(\mathcal{Q}) \times term(\mathcal{Q})$ is the smallest reflexive, symmetric and transitive relation that satisfies the following condition:

(*) if $R_1(s, t), R_2(s', t') \in \mathcal{Q}$ with $t \sim t'$ then $s \sim s'$.

Intuitively, the relation $\sim$ will be used to guarantee that if an instantiation of the query is satisfied by the least Herbrand model of $\mathcal{D}_\mathcal{T}$, then the instantiation does not contain a fork or cycle introduced by auxiliary constants (e.g., as in Example 2). For each equivalent class ξ of $\sim$, a representative $t_\xi \in \xi$ is chosen.

For a CQ $\mathcal{Q}$, the following notions help us to identify substructures in $\mathcal{Q}$ that can possibly lead to spurious solutions, as their instantiations may contain forks and cycles:

- **Fork** is the set of pairs $(pre(\xi), \xi)$ with ξ being an equivalence class of $\sim$, $pre(\xi) = \{t \in term(\mathcal{Q}) \mid \exists R \in N_R, \exists t' \in \xi$ such that $R(t, t') \in \mathcal{Q}\}$, and $|pre(\xi)| \geq 2$.
- **Cyc** is the set of quantified variables v such that there are $R_0(t_0, t_0'), \cdots, R_m(t_m, t_m'), \cdots, R_n(t_n, t_n') \in \mathcal{Q}$ $(n, m \geq 0)$, with $v \sim t_i$ for some $0 \leq i \leq n$, $t_i' \sim t_{i+1}$ for all $i < n$, and $t_n' \sim t_m$.

Fork and **Cyc** can be also computed in time polynomial to the size of $\mathcal{Q}$ [20].

Now, we introduce the rewriting of observations. Let CQ $\mathcal{Q}(\vec{x}) = \exists \vec{y}.\phi(\vec{x}, \vec{y})$, its *rewriting* $\mathcal{Q}^*(\vec{x})$ is the FO query $\exists \vec{y}.[\phi \wedge \phi_1 \wedge \phi_2]$ where

$$\phi_1 := \bigwedge_{(\{t_1, \cdots, t_k\}, \xi) \in \mathbf{Fork}} \big(Aux(t_\xi) \rightarrow \bigwedge_{1 \leq i < k} t_i = t_{i+1}\big)$$

$$\phi_2 := \bigwedge_{v \in \mathbf{Cyc}} Ind(v).$$

Intuitively, filter ϕ_1 says that fork-shaped substructures in $\mathcal{Q}$ should be instantiated in a way that all the legs merge into one, and ϕ_2 says that a cyclic substructure in $\mathcal{Q}$ can only be instantiated with individuals (not auxiliary constants).

EXAMPLE 3 (CONT'D EXAMPLE 1). *There are two equivalence classes $\{x, y\}$ and $\{z\}$ with respect to the relation $\sim$ in $\mathcal{Q}_1(x, y)$, and the rewriting of CQ $\mathcal{Q}_1(x, y)$ is*

$$\mathcal{Q}_1^*(x, y) = \exists z.[R(x, z) \wedge R(y, z) \wedge (Aux(z) \rightarrow x = y)].$$

Let $\mathcal{P}_1''$ be obtained from $\mathcal{P}_1'$ by replacing $\mathcal{Q}_1(a, b)$ with $\mathcal{Q}_1^*(a, b)$, then $\{A(a), A(b)\}$ is not a solution to $\mathcal{P}_1''$. Indeed, $\mathcal{P}_1''$ has no solution just as $\mathcal{P}_1$.

There is only one equivalence classes $\{x, y\}$ with respect to the relation $\sim$ in $\mathcal{Q}_2(x)$, and the rewriting of CQ $\mathcal{Q}_2(x)$ is

$$\mathcal{Q}_2^*(x) = \exists y.[R(x, y) \wedge R(y, y) \wedge (Aux(y) \rightarrow x = y) \wedge Ind(y)].$$

Let $\mathcal{P}_2''$ be obtained from $\mathcal{P}_2'$ by replacing $\mathcal{Q}_2(a)$ with $\mathcal{Q}_2^*(a)$, then $\{A(a)\}$ is not a solution to $\mathcal{P}_2''$, and no solution exists to $\mathcal{P}_2''$.

Note that our rewriting is simpler than that of [20], largely due to the datalog transformation we adopt in the paper, which provides a tighter approximation on the models. For example, given an axiom $A \sqsubseteq \exists R.B \sqcap \exists S.B$, it is transformed into four datalog rules $A(x) \rightarrow R(x, c_{R,B})$, $A(x) \rightarrow B(c_{R,B})$, $A(x) \rightarrow S(x, c_{S,B})$, and $A(x) \rightarrow B(c_{S,B})$. In an approximated model in [20], however informally, $c_{R,B}$ and $c_{S,B}$ are not distinguished. Thus, it is not hard to see that we do not need the filter queries related to $\mathbf{Fork}_{\neq}$ and $\mathbf{Fork}_{\mathcal{H}}$ in [20].

The following theorem states that the ontology transformation combined with observation rewriting is sound and complete in $\mathcal{ELH}_\bot$.

THEOREM 1. *Let $\mathcal{P} = \langle \mathcal{T}, \mathcal{A}, \mathcal{Q}(\vec{a}), \Sigma, \Delta \rangle$ be a QAP in $\mathcal{ELH}_\bot$ and $\mathcal{P}' = \langle \mathcal{D}_\mathcal{T}, \mathcal{A}, \mathcal{Q}^*(\vec{a}), \Sigma, \Delta \rangle$. Then, $\mathsf{sol}(\mathcal{P}) = \{\mathcal{E} \in \mathsf{sol}(\mathcal{P}') \mid \mathcal{D}_\mathcal{T} \cup \mathcal{A} \cup \mathcal{E} \not\models \exists x.\bot(x)\}$.*

Reducing a QAP in $\mathcal{ELH}_\bot$ to an equivalent one in datalog allows us to use a resolution-based procedure, like SLD-resolution, to generate solution candidates $\mathcal{E}$. Let Λ be the

set of all facts constructed from the predicates in Σ and the constants in Δ. Then, all the candidates for minimal solutions can be generated by enumerating the SLD-proofs of goal $\mathcal{Q}(\vec{a})$ *w.r.t.* $\mathcal{D}_\mathcal{T}$ and $\mathcal{A} \cup \Lambda$. We will speak informally about (SLD-)resolution proofs of $\mathcal{Q}^*(\vec{a})$, which are SLD-proofs of $\mathcal{Q}(\vec{a})$ validated against the additional filtering conditions in $\mathcal{Q}^*(\vec{a})$. Practically, $\mathcal{Q}^*(\vec{a})$ can be encoded in Prolog, and hence it makes sense to talk about resolution proofs of $\mathcal{Q}^*(\vec{a})$ and the solution candidates generated from them. We will omit $\vec{a}$ in the (rewritten) observation for simplicity.

To verify the non-triviality of a solution $\mathcal{E}$, existing query abduction algorithms [10] often use an OWL reasoner like HermiT or Pellet. Theorem 1 shows that we can use a datalog engine for such verifications, which is often more efficient than an OWL reasoner.

4.4 Module Extraction

Although reducing a QAP in $\mathcal{ELH}_\bot$ to a QAP in datalog allows us to apply rule-based reasoning and highly efficient rule-based systems to compute solutions, a resolution procedure can still be computationally very expensive, especially when handling a large number of facts. In such cases, even if there are a small number of datalog rules, a resolution procedure may suffer from a significant computational overhead when it attempts to find out valid substitutions for a large number of constants. This is illustrated with the following example.

Example 4. *Consider the BCQ* $\mathcal{Q} = \exists x.[A(x) \wedge B(x) \wedge C(x)]$*, datalog program* $\mathcal{D} = \{R(x,y) \wedge D(y) \rightarrow B(x), E(x) \rightarrow F(x)\}$*, and ABox* $\mathcal{A} = \{A(a), C(a), D(b), A(c_1), \ldots, A(c_m), E(d_1), \ldots, E(d_n)\}$*. Let* $\Sigma = \{R\}$ *and* Δ *consist of all the individuals in* $\mathcal{A}$*. Then the QAP* $\mathcal{P} = \langle \mathcal{D}, \mathcal{A}, Q, \Sigma, \Delta \rangle$ *has a single solution* $\{R(a,b)\}$*. Yet in the computation of (all minimal) solutions, a resolution procedure will attempt to resolve* $A(x)$ *with each* $A(c_i)$ *(*$1 \leq i \leq m$*). Then, after resolving* $B(c_i)$ *to* $R(c_i, y) \wedge D(y)$ *via the rule* $R(x,y) \wedge D(y) \rightarrow B(x)$*, it will attempt to resolve* y *in* $R(c_i, y)$ *with all the individuals including all the* d_j*'s (*$1 \leq j \leq n$*). Such an effort only fails eventually when trying to resolve say* $D(d_j)$ *and* $C(c_i)$*.*[2]

When the ABox is large, a huge number of similar unfortunate attempts may occur, and as a result the computation suffers.

Hence, we want to reduce the number of input facts by extracting modules of the initial QAP components that are sufficient for the query abduction. Using ABox modules can effectively reduce the search spaces for resolution. We achieve this through data summarisation, a technique used in query answering [7, 11, 27]. Yet instead of using the result of summarisation directly for reasoning, we apply a novel method to extract modules of the initial QAP components. A data summarisation method groups the individuals in the ABox according to their occurrences in the ABox and merges individuals within the same group. In the result of summarisation, only one representative individual remains for each group. Individuals are grouped according to their *types* in the ABox. For an ABox $\mathcal{A}$ and an individual a in $\mathcal{A}$, the *type* of a in $\mathcal{A}$ is $\tau = \{A \mid A(a) \in \mathcal{A}, A \in N_C\}$, i.e., a type is a finite set of concept names an individual is asserted to be a member in the ABox. Then, an ABox summary is defined as follows.

Definition 2. *Given an ABox* $\mathcal{A}$*, for each type* τ *of an individual in* $\mathcal{A}$*, assign a distinct fresh individual* c_τ *to* τ*. Let* δ *map each individual* a *in* $\mathcal{A}$ *to* c_τ *where* τ *is the type of* a *in* $\mathcal{A}$*. Then,* $\delta(\mathcal{A})$ *is the ABox obtained by replacing each individual* a *in* $\mathcal{A}$ *with* $\delta(a)$*, and is referred to as the* summary *of* $\mathcal{A}$*.*

In practice, the number of types of individuals in a dataset is often much smaller than the number of individuals. As individuals with the same type tend to behave in a similar way in reasoning, merging them in data summarisation often leads to a compact and tight approximation of the initial ABox.

For a QAP $\mathcal{P}' = \langle \mathcal{D}_\mathcal{T}, \mathcal{A}, \mathcal{Q}^*, \Sigma, \Delta \rangle$, we first show how to use the ABox summary to extract two modules of the ABox $\mathcal{A}$, one for generating QAP solutions and the other for verifying non-triviality. After that, we will show how to extract modules of other components of the QAP. For convenience, we regard δ as a mapping defined on the set of individuals in Δ by defining $\delta(a) = a$ for each a not occurring $\mathcal{A}$. Given an ABox summary, we are able to define two modules as follows. Recall that Λ is the set of all facts over the predicates in Σ and the constants in Δ.

$$\mathcal{A}_\mathcal{Q} = \{\alpha \in \mathcal{A} \mid \delta(\alpha) \in SLD(\delta(\mathcal{Q}^*), \mathcal{D}_\mathcal{T}, \delta(\mathcal{A} \cup \Lambda)\},$$
$$\mathcal{A}_\bot = \{\alpha \in \mathcal{A} \mid \delta(\alpha) \in SLD(\exists x.\bot(x), \mathcal{D}_\mathcal{T}, \delta(\mathcal{A} \cup \Lambda)\}.$$

Module $\mathcal{A}_\mathcal{Q}$ is for generating candidate solutions, whereas module $\mathcal{A}_\bot$ is for verifying non-triviality. To extract $\mathcal{A}_\mathcal{Q}$, we start with QAP $\mathcal{P}'' = \langle \mathcal{D}_\mathcal{T}, \delta(\mathcal{A}), \delta(\mathcal{Q}^*), \Sigma, \delta(\Delta) \rangle$, and generate solution candidates to $\mathcal{P}''$ with a resolution-based procedure. Note that using ABox summary $\delta(\mathcal{A})$ instead of $\mathcal{A}$ can largely reduce the number of input facts. In the process of generating solution candidates, we keep track of the facts in $\delta(\mathcal{A})$ that are involved in the resolution. Finally, the corresponding facts in $\mathcal{A}$ are extracted to form the module $\mathcal{A}_\mathcal{Q}$. Module $\mathcal{A}_\bot$ is extracted analogously.

In the same resolution and solution generation processes, we can also extract modules for other components of the QAP. In particular, define

$$\mathcal{D}_\mathcal{Q} = \{r \in \mathcal{D}_\mathcal{T} \mid r \in SLD\big(\delta(\mathcal{Q}^*), \mathcal{D}_\mathcal{T}, \delta(\mathcal{A} \cup \Lambda)\big)\},$$
$$\mathcal{D}_\bot = \{r \in \mathcal{D}_\mathcal{T} \mid r \in SLD\big(\exists x.\bot(x), \mathcal{D}_\mathcal{T}, \delta(\mathcal{A} \cup \Lambda)\big)\},$$
$$\Sigma_\mathcal{Q} = \{P \in \Sigma \mid \text{a fact } \alpha \text{ containing } P \text{ exists s.t. } \alpha \in SLD\big(\delta(\mathcal{Q}^*), \mathcal{D}_\mathcal{T}, \delta(\mathcal{A} \cup \Lambda)\big)\},$$
$$\Delta_\mathcal{Q} = \{a \in \Delta \mid \text{a fact } \alpha \text{ containing } \delta(a) \text{ exists s.t. } \alpha \in SLD\big(\delta(\mathcal{Q}^*), \mathcal{D}_\mathcal{T}, \delta(\mathcal{A} \cup \Lambda)\big)\}.$$

Note that $\mathcal{A}_\mathcal{Q}$, $\mathcal{D}_\mathcal{Q}$, $\Sigma_\mathcal{Q}$ and $\Delta_\mathcal{Q}$ can be extracted in one go via the resolution of $\delta(\mathcal{Q}^*)$, and $\mathcal{A}_\bot$ and $\mathcal{D}_\bot$ can be extracted via the resolution of $\exists x.\bot(x)$.

Example 5 (Cont'd Example 4). *There are four types in* $\mathcal{A}$*:* $\{A, C\}$ *for* a*,* $\{D\}$ *for* b*,* $\{C\}$ *for each* c_i *(*$1 \leq i \leq m$*), and* $\{E\}$ *for each* d_j *(*$1 \leq j \leq n$*). Using* a *and* b *themselves as representatives, and* c *and* d *as representatives for* c_i*'s and* d_j*'s, respectively,* $\delta(\mathcal{A}) = \{A(a), C(a), D(b), A(c), E(d)\}$*. Then,* $\mathcal{A}_\mathcal{Q} = \{A(a), C(a), D(b)\}$*,* $\mathcal{A}_\bot = \emptyset$*,* $\mathcal{D}_\mathcal{Q} = \{R(x,y) \wedge D(y) \rightarrow B(x)\}$*,* $\mathcal{D}_\bot = \emptyset$*,* $\Sigma_\mathcal{Q} = \{R\}$*,* $\Delta_\mathcal{Q} = \{a, b\}$*.*

[2] We assume the resolution proceeds from left to right as implemented in most Prolog engines, which may not be the case for every resolution procedure. Indeed, certain optimisation with ordering could help in this specific example, yet the example is to demonstrate a general issue not necessarily tied to ordering.

The following theorem shows that the above modules are sufficient for query abduction.

THEOREM 2. *Let $\mathcal{P} = \langle \mathcal{T}, \mathcal{A}, \mathcal{Q}, \Sigma, \Delta \rangle$ be a QAP in $\mathcal{ELH}_\bot$. Then, $\mathcal{E}$ is a minimal solution in $\mathsf{sol}(\mathcal{P})$ iff the following conditions hold:*

1. *$pred(\mathcal{E}) \subseteq \Sigma_\mathcal{Q}$ and $const(\mathcal{E}) \subseteq \Delta_Q$;*
2. *$\mathcal{D}_\mathcal{Q} \cup \mathcal{A}_\mathcal{Q} \cup \mathcal{E} \models Q^*$;*
3. *$\mathcal{D}_\bot \cup \mathcal{A}_\bot \cup \mathcal{E} \not\models \exists x.\bot(x)$;*
4. *$\mathcal{E} \not\models \mathcal{Q}$; and*
5. *no $\mathcal{E}' \subset \mathcal{E}$ exists satisfying conditions 1–4.*

In Exampe 5, $\mathcal{E} = \{R(a, b)\}$, which is the only ABox that satisfies the conditions in Theorem 2. $\mathcal{E}$ can be obtained via a SLD-proof of $\mathcal{Q}$ (the same as $\mathcal{Q}^*$) *w.r.t.* $\mathcal{D}_\mathcal{Q}$ and $\mathcal{A}_\mathcal{Q} \cup \Lambda_\mathcal{Q}$, where $\Lambda_\mathcal{Q}$ is the set of all facts over $\Sigma_\mathcal{Q}$ and $\Delta_\mathcal{Q}$. Then, $\mathcal{E}$ is verified against non-triviality conditions $\mathcal{D}_\bot \cup \mathcal{A}_\bot \cup \mathcal{E} \not\models \exists x.\bot(x)$ and $\mathcal{E} \not\models \mathcal{Q}$, and is returned as a solution.

4.5 Solution Search with Pruning

After preprocessing of the initial QAP, we obtain a datalog QAP of relatively small size. Our basic approach systematically searches for solution candidates using a resolution-based procedure (with tabling to guarantee termination), verifies each candidate using a datalog engine, and keeps only the minimal solutions. Besides systematically enumerating the resolution proofs, the search for solutions can largely benefit from advanced pruning methods. In particular, when a solution $\mathcal{E}$ is computed, instead of discarding $\mathcal{E}$ and constructing a new resolution proof, a pruning method further exploits $\mathcal{E}$ to generate another candidate solution $\mathcal{E}'$. When $\mathcal{E}'$ is verified to be a solution, the resolution proofs for $\mathcal{E}'$ and those for its supersets are pruned. Various heuristics can be applied in pruning. In [9], a pruning strategy that exploits the subsets $\mathcal{E}'$ of $\mathcal{E}$ is proposed, which allows the minimality of a solution to be checked immediately after its generation. Although such a pruning strategy for query abduction may require a larger number of verifications than the basic approach, such verifications are often less expensive than resolution. Given that the candidates $\mathcal{E}'$ obtained from $\mathcal{E}$ are close enough to valid solutions and can be enumerated efficiently, a search with pruning can largely outperform a basic one.

Once a minimal solution $\mathcal{E}$ is computed, we propose a new pruning strategy for solution search that exploits $\mathcal{E}$ and the hierarchical knowledge from the TBox. A *hierarchical axiom* in $\mathcal{ELH}_\bot$ is of the form $A \sqsubseteq B$ or $R \sqsubseteq S$. Such hierarchical axioms often constitute a large portion of practical ontology axioms, and can cause computation difficulty in resolution. This is illustrated in the following example.

EXAMPLE 6. *Consider the BCQ $Q = A_1(a_1) \wedge \ldots \wedge A_1(a_m)$ and the datalog program $\mathcal{D} = \{A_{i+1}(x) \rightarrow A_i(x) \mid 1 \leq i \leq n\}$. Let $\mathcal{A} = \emptyset$, $\Sigma = \{A_1, \ldots, A_n\}$, and $\Delta = \{a_1, \ldots, a_m\}$. The QAP $\mathcal{P} = \langle \mathcal{D}, \mathcal{A}, Q, \Sigma, \Delta \rangle$ has n^m minimal solutions of the form $\{A_{j_1}(a_1), \ldots, A_{j_m}(a_m) \mid 1 \leq j_1, \ldots, j_m \leq n\}$.*

Constructing a resolution proof for each of the minimal solution is a tedious job and involves a large amount of repetition. However, from one minimal solution $\{\ldots, A_j(a_i), \ldots\}$, it is convenient to directly construct another minimal solution $\{\ldots, A_{j+1}(a_i), \ldots\}$ using hierarchical rule $A_{j+1}(x) \rightarrow A_j(x)$.

Based on this observation, we propose the following pruning technique that makes use of the hierarchical knowledge on concept names and role names in the TBox. Consider the QAP $\mathcal{P}' = \langle \mathcal{D}_\mathcal{T}, \mathcal{A}, \mathcal{Q}^*, \Sigma, \Delta \rangle$ and the modules of its components $\mathcal{D}_\mathcal{Q}$, $\mathcal{A}_\mathcal{Q}$, $\Sigma_\mathcal{Q}$, and $\Delta_\mathcal{Q}$. For a solution $\mathcal{E} \in \mathsf{sol}(\mathcal{P})$, a *weakening* of $\mathcal{E}$ is obtained by replacing some $A(a) \in \mathcal{E}$ with $B(a)$ such that $B \in \Sigma_\mathcal{Q}$ and $\mathcal{T} \models A \sqsubseteq B$, or by replacing some $R(a, b) \in E$ with $S(a, b)$ such that $S \in \Sigma_\mathcal{Q}$ and $\mathcal{T} \models R \sqsubseteq S$; a *strengthening* of $\mathcal{E}$ is obtained by replacing some $A(a) \in \mathcal{E}$ with $B(a)$ such that $B \in \Sigma_\mathcal{Q}$ and $\mathcal{T} \models B \sqsubseteq A$, or by replacing some $R(a, b) \in E$ with $S(a, b)$ such that $S \in \Sigma_\mathcal{Q}$ and $\mathcal{T} \models S \sqsubseteq R$. $\mathcal{E}_\uparrow$ and $\mathcal{E}_\downarrow$ denote a weakening and a strengthening of $\mathcal{E}$, respectively.

PROPOSITION 2. *Let $\mathcal{P} = \langle \mathcal{T}, \mathcal{A}, \mathcal{Q}, \Sigma, \Delta \rangle$ be a QAP in $\mathcal{ELH}_\bot$. For a solution $\mathcal{E} \in \mathsf{sol}(\mathcal{P})$, the following conditions hold:*

- *$\mathcal{E}_\uparrow \in \mathsf{sol}(\mathcal{P})$ iff $\mathcal{D}_\mathcal{Q} \cup \mathcal{A}_\mathcal{Q} \cup \mathcal{E}_\uparrow \models \mathcal{Q}^*$ and $\mathcal{E}_\uparrow \not\models \mathcal{Q}$; and $\mathcal{E}_\uparrow$ is minimal if $\mathcal{E}$ is minimal.*
- *$\mathcal{E}_\downarrow \in \mathsf{sol}(\mathcal{P})$ iff $\mathcal{D}_\bot \cup \mathcal{A}_\bot \cup \mathcal{E}_\downarrow \not\models \exists x.\bot(x)$ and $\mathcal{E}_\downarrow \not\models \mathcal{Q}$; and $\mathcal{E}_\downarrow$ is minimal only if $\mathcal{E}$ is minimal.*

Note that in the second condition, the minimality of $\mathcal{E}_\downarrow$ is not guaranteed by the second half of the condition. For example, let $\mathcal{Q} = D(a)$, $\mathcal{D}_\mathcal{Q} = \{B(x) \rightarrow A(x), A(x) \wedge C(x) \rightarrow D(x), B(x) \rightarrow D(x)\}$, $\Sigma_\mathcal{Q} = \{A, B, C\}$, $\Delta_\mathcal{Q} = \{a\}$, and $\mathcal{D}_\bot$ and $\mathcal{A}_\mathcal{Q} \cup \mathcal{A}_\bot$ are both empty. Then, a minimal solution is $\mathcal{E} = \{A(a), C(a)\}$, whereas $\mathcal{E}_\downarrow = \{B(a), C(a)\}$ is a solution but not minimal.

Once a solution $\mathcal{E}$ is computed, our pruning method checks the minimality of $\mathcal{E}$ as in [9]. If $\mathcal{E}$ is minimal, the method enumerates and verifies whether each $\mathcal{E}_\uparrow$ and each $\mathcal{E}_\downarrow$ is a minimal solution. $\mathcal{E}_\uparrow$ and $\mathcal{E}_\downarrow$ are enumerated using the concept and role hierarchies pre-computed from $\mathcal{T}$ using an OWL reasoner. Verifications of $\mathcal{E}_\uparrow$ and $\mathcal{E}_\downarrow$ are performed efficiently using a datalog engine.

5. EXPERIMENTS

Based on the approach proposed in Section 4, we have implemented a prototype system called ABEL (ABduction for EL). Our experiments show that ABEL is efficient and can be used to compute minimal explanations for large ontologies and data sets in $\mathcal{ELH}_\bot$.

ABEL is implemented in Java. In our system, the KARMA system [26] is used for ontology transformation, Prolog engine XSB 3.4 [25] is used for generating solution candidates, and OWL reasoner Pellet[3] as well as datalog engine RDFox[4] is used for verifying solution candidates. We have evaluated our system on three suites of ontology benchmarks and under different settings. All the experiments were performed on a PC with an Intel Xenon 2.8 GHz processor and 16 GB RAM. The system and testing data are available at `http://www.ict.griffith.edu.au/~kewen/AbductionEL/`.

5.1 Test Cases

In our experiments, we have employed ontologies that are widely used for benchmarking ontology-based CQ answering and for abduction [19, 26, 9]. The ontologies, corresponding data sets and queries are described as follows. To generate observations, we instantiated each query by substituting

[3] `clarkparsia.com/pellet`

[4] `www.cs.ox.ac.uk/isg/tools/RDFox`

their query variables with randomly chosen constants from the corresponding data set and verified that each instantiation indeed formed a negative answer to the query.

VICODI: The VICODI ontology[5] describes European history and contains no genuine existential rules. This benchmark has a moderate-sized data set, has been widely used for evaluating atomic queries, and hence is useful for comparing our system with DuQAP, a system developed by Du et al. [9], since it allows only atomic queries. We generated 40 atomic queries for this benchmark with a random set of atomic concepts and roles.

SEMINTEC: The SEMINTEC ontology[6] for modelling financial processes contains genuine existential rules and is coupled with a moderate-sized data set. We used it in comparing our system with DuQAP, and we generated 40 atomic queries for this benchmark as above.

LSTW: The LSTW ontology [19] extends the Lehigh University benchmark LUBM [13] with many genuine existential rules. Corresponding data sets can be automatically generated in a range of sizes using EUGen [19]. LSTW(n) denotes the variant with a data set of n universities. Besides using this benchmark for comparison, with 40 atomic queries generated as above, we also used it in the scalability evaluation, with 120 CQs generated using the query generator SyGENiA [16][7].

Table 2 shows some statistics on the aforementioned ontologies: the numbers of their concept names (#C), role names (#R), TBox axioms ($|\mathcal{T}|$), genuine existential rules (#E), ABox facts ($|\mathcal{A}|$), and individuals (#I).

Table 2: Benchmark statistics.

Ontology	#C	#R	$\lvert\mathcal{T}\rvert$	#E	$\lvert\mathcal{A}\rvert$	#I
VICODI	194	10	214	0	116,181	32,238
SEMINTEC	60	16	207	8	65,244	17,945
LSTW(n)	132	32	223	29	$\approx 10^5 n$	$\approx 10^4 n$

We evaluated ABEL in three different configurations.

ABEL-N is the naive version of ABEL without pruning nor module extraction, and was used for comparison with DuQAP. In order to have a fair comparison, database techniques were not employed in ABEL-N. Also, Pellet was used instead of a datalog engine for reasoning tasks such as solution verification.

ABEL-P uses MySQL database system to store data and enables the pruning optimisation. Also, datalog engine RDFox was employed for solution verification. However, module extraction was disabled in order to study its effect on performance.

ABEL-M is the optimised version of ABEL-P using the module extraction technique. It is tested in the scalability evaluation with large data sets.

5.2 Comparison

In the first set of experiments, we compared our system with DuQAP. DuQAP accepts as input expressive ontologies in $\mathcal{SHIOQ}$ but only allows observations formulated in atomic queries. For each of our testing ontologies, we generated 40 observations by instantiating the corresponding 40 atomic queries. We included all concepts in the ontology as abducibles, and the domain includes all constants in the corresponding data set. Roles were not included in the abducibles, since DuQAP ran out of memory in those cases.

Table 3 shows the comparison results: the average numbers of minimal solutions (#$\mathcal{E}$, averaged over 40 observations)[8], and the average times (in seconds) for computing all the minimal solutions. DuQAP could not handle LSTW(10) or beyond due to running out of memory. It is worth noting that ABEL-N already outperformed DuQAP, and this superiority is largely enhanced with other optimisations like pruning (with the time reduced by at least 90%).

Table 3: Comparison of DuQAP and ABEL.

Ontology	#$\mathcal{E}$	DuQAP	ABEL-N	ABEL-P
VICODI	12.4	9.5	5.8	0.1
SEMINTEC	0.4	3.4	1.0	0.1
LSTW(1)	8.6	7.8	5.1	0.4
LSTW(5)	8.6	46.9	24.8	1.1
LSTW(10)	8.6	N/A	85.9	5.2

5.3 Scalability Evaluation

To evaluate the scalability of our system with observations formulated in general CQs that can interact with genuine existential rules in a complex manner, we evaluated ABEL-P and ABEL-M with the 120 generated CQs, a large portion of which are general CQs. One observation is generated for each query. Since LSTW contains no disjointness axiom, to enforce non-triviality checks, we added 5 disjointness axioms to the LSTW ontology. We increased the scope of abducibles so that all the concepts and all the roles in the ontology are included, and each domain is of size 5 (with constants in the observation plus possibly some from the data set). We set a 10 minutes' time bound for each QAP instance, and a QAP instance is considered to be successfully processed only if all its minimal solutions are computed in the time bound.

ABEL-P was able to handle all the 120 observations for moderate-sized LSTW(1), and the average computation time was 19.2 seconds. Table 4 shows the results for large-sized LSTW(10), LSTW(50) and LSTW(100): the number of observations successfully processed (#), the times for computing all the minimal solutions and for extracting modules (t_c and t_m, in seconds, both averaged over the successfully processed observations), and the sizes of modules divided by the sizes of the initial data set $\mathcal{A}$ ($|\mathcal{A}_\mathcal{Q}|$ and $|\mathcal{A}_\perp|$, in percentage, both averaged over the successfully processed observations).

ABEL-P was able to successfully process most of the observations (105, that is 88%) for LSTW(10), and nearly half of them (55, that is 46%) for each of LSTW(50) and LSTW(100). ABEL-M could handle all the observations successfully processed by ABEL-P, and the advantage of ABEL-M (in particular, the module extraction optimisation) becomes more obvious when the data sets get larger. In

[5] `http://www.vicodi.org`

[6] `http://www.cs.put.poznan.pl/alawrynowicz/semintec.htm`

[7] As SyGENiA failed to terminate in some cases on LSTW, we set a bound on the chase algorithm of SyGENiA to generate CQs with up to three atoms.

[8] The numbers were the same for both ABEL(-N and -P) and DuQAP.

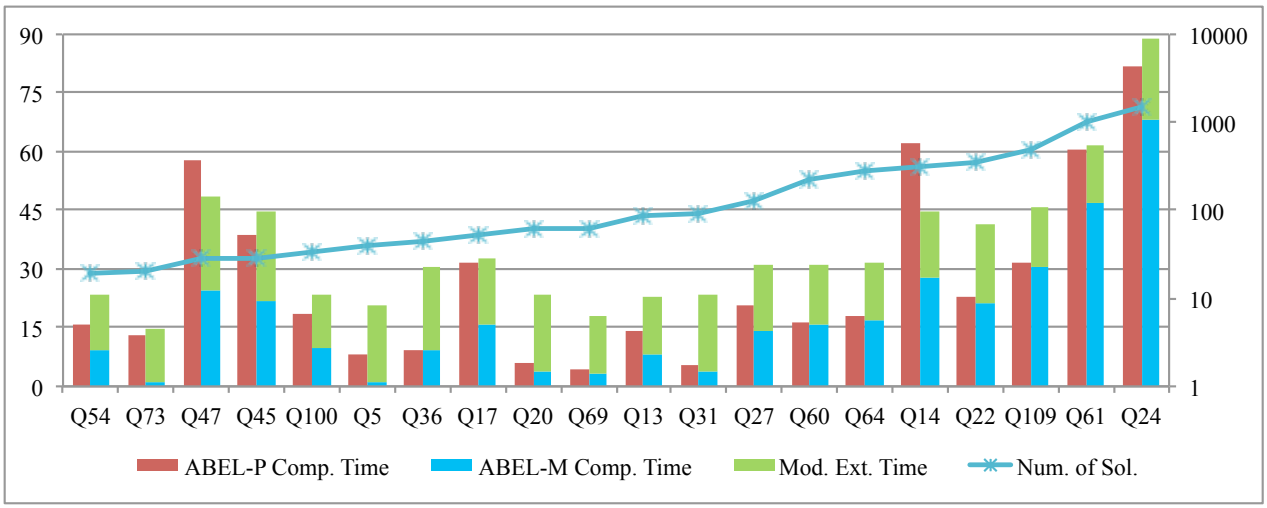

Figure 1: Scalability evaluation on LSTW(50).

Table 4: Scalability evaluation on generated queries.

LSTW (n)	ABEL-P		ABEL-M				
	#	t_c	#	t_c	t_m	$\lvert\mathcal{A}_\mathcal{Q}\rvert$	$\lvert\mathcal{A}_\perp\rvert$
10	105	39.2	106	25.6	15.4	3.4%	10.7%
50	55	23.6	83	41.1	26.2	3.7%	8.5%
100	55	43.7	82	31.5	31.4	0.3%	1.8%

particular, for LSTW(50) and LSTW(100), ABEL-M could process 28 and 27 observations respectively, on which ABEL-P failed. Moreover, for all the observations over LSTW(50), ABEL-M could compute at least some minimal solutions, and failed to output any solutions for only two observations in the case of LSTW(100). These two difficult observations are both associated with recursive rules in the ontology.

Figure 1 shows the reasoning times and the numbers of solutions computed by ABEL-P and ABEL-M, on 20 selected observations over LSTW(50). These 20 observations are the most challenging ones among those successfully processed by both ABEL-P and ABEL-M. The column chart shows the computation and module extraction times of ABEL-P and ABEL-M (the left vertical axis, in seconds). The line chart shows the total numbers of minimal solutions (the right vertical axis, in logarithm scale).

Figure 1 demonstrates the efficiency gain as well as the overhead for module extraction in ABEL-M (compared to ABEL-P). On the observations that could be easily processed by ABEL-P, the efficiency gain from module extraction was often not significant enough to override its overhead. The efficient gain was significant however, on observations that were hard for ABEL-P, such as Q14, Q17, Q45, and Q47, where the computation times were reduced by half because of module extraction. On hard observations where ABEL-P failed to process, the advantage of module extraction became very significant, as shown in Table 4. Hence, in general, for query explanation over large data sets, it would be a good strategy to first attempt to generate explanations with ABEL-P, and to employ module extraction (ABEL-M) only when ABEL-P fails in the time bound.

6. CONCLUSION

We have developed a scalable and complete system for query abduction w.r.t. observations formulated in BCQs and background ontologies in $\mathcal{ELH}_\perp$. Our approach involves transforming an $\mathcal{ELH}_\perp$ ontologies to a datalog program and rewriting the observation. While the ontology transformations and rewriting techniques have been intensively studied for query answering, our work initiates to adapt such techniques from standard query answering to query abduction. Indeed, by adapting these techniques, we provide, to the best of our knowledge, the first sound and complete algorithm for query abduction that supports both general CQs and ontologies with genuine existential rules. Moreover, we introduced a novel method for module extraction in query abduction, by applying data summarisation techniques, and we developed a pruning strategy that largely enhances the efficiency of solution generation. Finally, the potential of using our algorithm to handle practical ontologies and large data sets in practice is corroborated through our evaluation.

Interesting future work includes, firstly, extending the current approach to other ontology languages, including Horn DLs and existential rules. The approach in [18] can be a starting point. For more expressive Horn DLs and existential rules, an interesting research problem is to decide which fragments or which ontologies/queries are *combined rewritable*, that is, an ontology transformation and an observation rewriting exist that are sound and complete for query abduction. The study on combined rewritability of existential rules regarding query answering in presented in [12], and would shed light on this research direction. Moreover, although the observation rewriting explores the structures of the CQs to some extent, we believe a more fine-grained analysis of query structures and possibly specific structures of some ontology axioms will allow us to apply advanced heuristics in solution construction.

7. ACKNOWLEDGMENTS

This work was partially supported by Australian Research Council (ARC) under grants DP110101042 and DP130102302. Jianfeng Du is partially supported by NSFC (61375056) and Guangdong Natural Science Foundation (S2013010012928).

8. REFERENCES

[1] F. Baader, S. Brandt, and C. Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, pages 364–369, 2005.

[2] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat. On rules with existential variables: Walking the decidability line. *Artificial Intelligence*, 175(9-10):1620–1654, 2011.

[3] A. Borgida, D. Calvanese, and M. Rodriguez-Muro. Explanation in the DL-Lite family of description logics. In *On the Move to Meaningful Internet Systems: OTM*, pages 1440–1457. 2008.

[4] A. Borgida, E. Franconi, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider. Explaining $\mathcal{ALC}$ subsumption. In *Proceedings of the 14th European Conference on Artificial Intelligence*, pages 209–213, 2000.

[5] A. Calì, G. Gottlob, T. Lukasiewicz, and A. Pieris. Datalog+/-: A family of languages for ontology querying. In *Datalog Reloaded - First International Workshop*, pages 351–368, 2010.

[6] D. Calvanese, M. Ortiz, M. Šimkus, and G. Stefanoni. Reasoning about explanations for negative query answers in DL-Lite. *Journal of Artificial Intelligence Research*, 48:635–669, 2013.

[7] J. Dolby, A. Fokoue, A. Kalyanpur, E. Schonberg, and K. Srinivas. Scalable highly expressive reasoner SHER. *Journal of Web Semantics*, 7(4):357–361, 2009.

[8] J. Du and Y.-D. S. Kewen Wang. Tractable approach to ABox abduction over description logic ontologies. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, 2014.

[9] J. Du, G. Qi, Y. Shen, and J. Z. Pan. Towards practical ABox abduction in large description logic ontologies. *International Journal on Semantic Web and Information Systems*, 8(2):1–33, 2012.

[10] J. Du, S. Wang, G. Qi, J. Z. Pan, and Y. Hu. A new matchmaking approach based on abductive conjunctive query answering. In *Proceedings of the Joint International Semantic Technology Conference*, pages 144–159, 2011.

[11] B. Glimm, Y. Kazakov, T. Liebig, T. Tran, and V. Vialard. Abstraction refinement for ontology materialization. In *Proceedings of the 13th International Semantic Web Conference* , pages 180–195, 2014.

[12] G. Gottlob, M. Manna, and A. Pieris. Polynomial combined rewritings for existential rules. In *Proceedings of the 14th International Conference on Principles of Knowledge Representation and Reasoning*, 2014.

[13] Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2-3):158 – 182, 2005.

[14] K. Halland and K. Britz. ABox abduction in $\mathcal{ALC}$ using a DL tableau. In *Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference*, pages 51–58, 2012.

[15] M. Horridge, B. Parsia, and U. Sattler. Laconic and precise justifications in OWL. In *Proceedings of the 7th International Semantic Web Conference*, pages 323–338, 2008.

[16] M. Imprialou, G. Stoilos, and B. C. Grau. Benchmarking ontology-based query rewriting systems. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence*, 2012.

[17] S. Klarman, U. Endriss, and S. Schlobach. ABox abduction in the description logic $\mathcal{ALC}$. *Journal of Automated Reasoning*, 46(1):43–80, 2011.

[18] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The combined approach to query answering in DL-Lite. In *Proceedings of the 12th International Conference on Principles of Knowledge Representation and Reasoning*, 2010.

[19] C. Lutz, n. Seylan, D. Toman, and F. Wolter. The combined approach to OBDA: Taming role hierarchies using filters. In *Proceedings of the 12th International Semantic Web Conference*, pages 314–330. 2013.

[20] C. Lutz, D. Toman, and F. Wolter. Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 2009.

[21] Y. Ma, T. Gu, B. Xu, and L. Chang. An ABox abduction algorithm for the description logic $\mathcal{ALCI}$. In *Intelligent Information Processing*, pages 125–130, 2012.

[22] D. L. McGuinness and A. T. Borgida. Explaining subsumption in description logics. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 816–821, 1995.

[23] D. L. McGuinness and P. F. Patel-Schneider. Usability issues in knowledge representation systems. In *Proceedings of the 15th AAAI Conference on Artificial Intelligence*, pages 608–614, 1998.

[24] R. Peñaloza and B. Sertkaya. Complexity of axiom pinpointing in the DL-Lite family of description logics. In *Proceedings of the 19th European Conference on Artificial Intelligence*, pages 29–34, 2010.

[25] P. Rao, K. Sagonas, T. Swift, D. Warren, and J. Freire. XSB: A system for efficiently computing well-founded semantics. In *Logic Programming And Nonmonotonic Reasoning*, volume 1265 of *Lecture Notes in Computer Science*, pages 430–440. Springer Berlin Heidelberg, 1997.

[26] G. Stefanoni, B. Motik, and I. Horrocks. Introducing nominals to the combined query answering approaches for $\mathcal{EL}$. In *Proceedings of the 27th AAAI Conference on Artificial Intelligence*, 2013.

[27] Y. Zhou, Y. Nenov, B. C. Grau, and I. Horrocks. Pay-as-you-go OWL query answering using a triple store. In *Proceedings of the 28th AAAI Conference on Artificial Intelligence*, pages 1142–1148, 2014.