Computing Explanations for Negative Answers to Conjunctive Queries in $\mathcal{ELH}$

Zhe Wang, Mahsa Chitsaz, and Kewen Wang

School of Information and Communication Technology,
Griffith University, Australia

Abstract. Ontology-based data access and query answering are emerging to be a crucial reasoning support for many data-intensive applications. Besides standard query answering, there is also a need for query engines to provide the users with meaningful explanations to so-called *negative query answers* (i.e. desired answers of a query which are not derivable from the given ontology). However, a practical algorithm to explain negative answers of conjunctive queries (CQs) over an ontology with existential rules is still missing. In this paper, an efficient algorithm is developed to achieve this for $\mathcal{ELH}$ ontologies, using abductive reasoning. In particular, we apply a tight datalog approximation to a given ontology so that a resolution-based datalog abduction approach can be used. Also, we adapt a query rewriting technique to filter out spurious solutions due to the approximation. We show that our algorithm is sound and complete. Furthermore, we encode the abduction problem in Prolog and use an off-the-shelf Prolog engine XSB to generate solutions. Finally, we evaluate the new algorithm over practical ontologies, and our evaluation shows that the algorithm is capable of handling reasonably large data sets efficiently.

1 Introduction

Ontologies have been applied in a wide range of practical domains such as e-Science, e-Commerce, medical informatics, bio-informatics, and the Semantic Web [22]. The standardised web ontology language OWL and its latest version OWL 2 are based on description logics (DLs), where an ontology is formalized as a DL knowledge base or just a TBox. Efficient DL reasoners have been developed for reasoning tasks such as classification and query answering. In order to meet usability requirements set by domain users, efforts have been made to equip DL-based ontology systems with explanation algorithms for reasoning services [20,19,3,21,12,5]. One important explanation problem for DL ontologies, referred to as ABox abduction, can be formalised as *abductive reasoning* (or, *abduction*) investigated in AI [8]. In particular, given an ontology and an observation (i.e., a query with an answer), a solution to the ABox abduction problem (a.k.a. an explanation for the query answer) is an ABox that together with the ontology can entail the query answer.

Existing approaches to ABox abduction in DLs vary according to their underlying DLs and types of queries. For instance, [20,19] are based on $\mathcal{ALCN}$, [14,11,18] are based on $\mathcal{ALC}$, [2,5] are based on DL-Lite, while [7] is based on $\mathcal{SHIOQ}$. Regarding queries used for observations, both atomic queries and *conjunctive queries* (CQs)

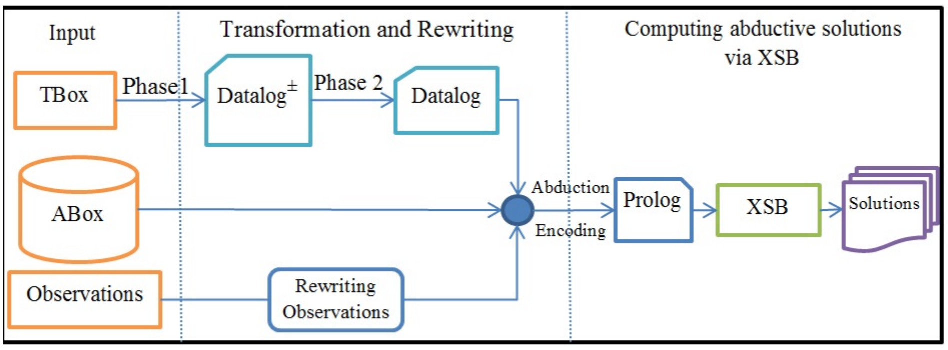

Fig. 1. The overview of the developed ABox abduction system

are considered in the literature. Moreover, a query answer can be either *positive* or *negative*. A query answer is positive *w.r.t.* a given ontology and a query if it occurs in the answer set of the query over the ontology; otherwise, it is negative (a user expects some tuple to be among the answers but it turns out to be not). Previous research focused on ABox abduction for positive atomic queries until the problem of explaining negative answers for CQs over DL-Lite ontologies is studied by Borgida, Calvanese and Rodriguez-Muro [2]. They advocated that an ontology-based system should also be equipped with the reasoning service of explaining *negative query answers* as it is essential to understand why a desired answer is missing from the query answers.

Some researchers have investigated tableau-based algorithms for ABox abduction [14,11,18] as well as computational complexity of ABox abduction [5]. However, algorithms such as the tableau-based ones, are not efficient enough to deal with ABox abduction for realistic ontologies. *The problem of developing practical algorithms for ABox abduction supporting CQs is rarely studied except for [7]*, in which an algorithm is developed for ABox abduction with CQs and it is shown that the algorithm is able to handle some large ontologies such as LUMB ontologies [10]. Besides several optimization techniques, a major idea in their algorithm is to encode an ABox abduction problem as a Prolog program. However, in their transformation, rules containing existential restrictions in the head, which we refer to as *existential rules*, are simply discarded. While this brute force approach to handle existential rules is simple, the resulting Prolog program is too weak to find some solutions for ABox abduction with DL ontologies containing (essentially) existential rules. For example, consider the ontology $\mathcal{K}$ consisting of only one TBox axiom $A \sqsubseteq \exists r.B$ and one ABox assertion $B(a)$, and the query $q(x) = \exists y.r(x, y)$. Note that a is a negative answer to the query as $q(a)$ is not derivable from the ontology. Intuitively, the new ABox $\{A(a)\}$ is an explanation for a as $\mathcal{K} \cup \{A(a)\} \models q(a)$. However, the algorithm developed in [6] discards the axiom $A \sqsubseteq \exists r.B$ and thus the resulting Prolog program is insufficient for obtaining the explanation $\{A(a)\}$.

In this paper, we develop a new practical algorithm for ABox abduction for CQs in $\mathcal{ELH}$ [1] as shown in Figure 1. This is achieved through two steps of ontology transformations and one step of observation rewriting. First, we equivalently translate a given $\mathcal{ELH}$ ontology into a datalog$^{\pm}$ program using a translation introduced in [4]. Then, in order to employ some off-the-shelf Prolog engines, such as XSB[1], the resulting datalog$^{\pm}$ program is further transformed into a plain datalog program based on a technique introduced for query answering in [23]. Specifically, we rewrite each existential rule into two datalog rules by instantiating existential variables with fresh constants, instead of discarding the existential rule as in [6] (cf. Table 2). For instance, the axiom $A \sqsubseteq \exists r.B$ in the above example is transformed into two datalog rules $A(x) \to r(x, c)$ and $A(x) \to B(c)$ with c a fresh constant. Our approach is more fine-grained than that in [6], while it is still unsound. To ensure the soundness of our approach, we adapt a query rewriting technique to rewrite the observation so as to filter out incorrect solutions. As a result, the ABox abduction problem is reduced to an abduction problem in datalog, and we show that our approach is sound and complete for ABox abduction. Finally, the abduction problem is encoded as a Prolog program with an extended encoding of [6], and thus, by executing the Prolog program on XSB, all the minimal solutions for ABox abduction are computed. For the above example in particular, our algorithm will return $\{A(a)\}$ as a solution. We evaluate our algorithm over some realistic ontologies with reasonably large data sets such as LUMB [10]. Our experimental results show that the algorithm is capable of handling ABox abduction problems for large ontologies efficiently.

We choose $\mathcal{ELH}$ as the OWL 2 EL profile is based on it, and many practical ontologies can be expressed in $\mathcal{ELH}$. Furthermore, $\mathcal{ELH}$ is a core fragment of many expressive DLs, and our results on $\mathcal{ELH}$ provide a foundation for studying ABox abduction for other DLs.

Due to space limitation, proofs are omitted from the paper but can be found at `http://www.ict.griffith.edu.au/~kewen/AbductionEL/eswc2014.pdf`.

2 Preliminaries

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

2.1 Description Logic $\mathcal{ELH}$

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

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

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 $r, s \in N_R$.

[1] http://xsb.sourceforge.net/

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 $\mathcal{ELH}$ is defined as usual in terms of an interpretation $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, and we refer to [1] for details. As $\mathcal{ELH}$ is a fragment of first-order logic, an $\mathcal{ELH}$ KB $\mathcal{K}$ can be seen as a first-order theory. A *model* of $\mathcal{K}$ and entailment can also be defined as first-order ones. $\mathcal{K}$ is *consistent* if it has at least one model.

An $\mathcal{ELH}$ TBox is *normalized* if it consists axioms only the forms of $A \sqsubseteq B$, $A_1 \sqcap A_2 \sqsubseteq B$, $A \sqsubseteq \exists r.B$, $\exists r.A \sqsubseteq B$, and $r \sqsubseteq s$. An $\mathcal{ELH}$ TBox can be transformed in polynomial time to a normalized TBox that is equivalent to the TBox [1].

2.2 Rule Languages

Datalog$^{\pm}$ [4] is developed as a family of rule languages that extend datalog with existential rules and are sufficient to capture several light DLs including $\mathcal{ELH}$. A *datalog*$^{\pm}$ *rule* is a sentence of the form

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

where $\phi(\boldsymbol{x},\boldsymbol{y})$ and $\psi(\boldsymbol{x},\boldsymbol{z})$ are conjunctions of function-free atoms and $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ 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. Note that a fact is a special datalog$^{\pm}$ rule whose head is a ground atom and whose body is empty. A datalog$^{\pm}$ program is a finite set of datalog$^{\pm}$ rules.

A *datalog* rule is a datalog$^{\pm}$ rule whose head $\psi(\boldsymbol{x},\boldsymbol{z})$ consists of a single atom and $\boldsymbol{z}$ is empty.

2.3 Conjunctive Queries

A *conjunctive query* (CQ) is a formula $q(\boldsymbol{x}) = \exists y.\phi(\boldsymbol{x},\boldsymbol{y})$ with $\phi(\boldsymbol{x},\boldsymbol{y})$ a conjunction of function-free atoms over $N_C \cup N_R$. Variables $\boldsymbol{x}$ are *answer variables*, denoted $avar(q)$, and $\boldsymbol{y}$ are *quantified variables*, denoted $qvar(q)$. A CQ can also be denoted as a datalog rule $\phi(\boldsymbol{x},\boldsymbol{y}) \to q(\boldsymbol{x})$. An *atomic query* is a CQ where $\phi(\boldsymbol{x},\boldsymbol{y})$ consists of a single atom and $\boldsymbol{y}$ is empty.

A tuple of constants $\boldsymbol{a}$ is a *positive answer* (or simply an *answer*) of a CQ $q(\boldsymbol{x})$ over a KB $\mathcal{K}$, if the arity of $\boldsymbol{a}$ agrees with that of $\boldsymbol{x}$ and $\mathcal{K} \models q(\boldsymbol{a})$. Otherwise, if $\mathcal{K} \not\models q(\boldsymbol{a})$ then $\boldsymbol{a}$ is a *negative answer* of $q(\boldsymbol{x})$ over $\mathcal{K}$.

2.4 ABox Abduction

Given a background KB $\mathcal{K}$ and a ground CQ $q(\boldsymbol{a})$ as an *observation*, a solution to the ABox abduction problem is an (possibly empty) ABox $\mathcal{E}$ such that $\mathcal{E}$ does not entail $q(\boldsymbol{a})$ but $\mathcal{K}$ together with $\mathcal{E}$ entails $q(\boldsymbol{a})$. Also, 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, a solution should be as succinct as possible. The formal definition is given as follows.

Definition 1 (ABox Abduction Problem). *An* ABox abduction problem *is a triple* $\langle \mathcal{K}, q(\boldsymbol{a}), S\rangle$*, where* $\mathcal{K}$ *is a KB,* $S \subseteq N_C \cup N_R$ *is a finite set of* abducibles*, and* $q(\boldsymbol{a})$ *is an* observation *with* q *a CQ and* $\boldsymbol{a}$ *a tuple of constants.*

An ABox $\mathcal{E}$ *is a* solution *to the ABox abduction problem iff (1)* $pred(\mathcal{E}) \subseteq S$*, (2)* $\mathcal{K} \cup \mathcal{E} \models q(\boldsymbol{a})$*, (3)* $\mathcal{E} \not\models q(\boldsymbol{a})$*, and (4)* $\mathcal{K} \cup \mathcal{E}$ *is consistent. Let* $sol(\mathcal{K}, q(\boldsymbol{a}), S)$ *be the set of solutions to the ABox abduction problem. Moreover, we call* $\mathcal{E}$*:*

- set minimal *iff there is no solution* $\mathcal{E}' \in sol(\mathcal{K}, q(\boldsymbol{a}), S)$ *s.t.* $\mathcal{E}' \subset \mathcal{E}$*.*
- cardinality minimal *iff there is no solution* $\mathcal{E}' \in sol(\mathcal{K}, q(\boldsymbol{a}), S)$ *s.t.* $|\mathcal{E}'| < |\mathcal{E}|$*.*

Note that an $\mathcal{ELH}$ KB is always consistent, and thus $\mathcal{K} \cup \mathcal{E}$ is always consistent. However, condition (4) is necessary for other logics when inconsistency may occur. In the above definition, we can replace $\mathcal{K}$ with a datalog$^{\pm}$ program or a datalog program. Thus, an abduction problem in datalog$^{\pm}$ and datalog can be defined accordingly.

3 Transformation-Based Algorithms for ABox Abduction

While we consider a light weight DL $\mathcal{ELH}$, the problem of ABox abduction addressed in this paper is more general than some previous approaches in the sense that an observation is a ground CQ not just an atomic query. In addition, we do not simply discard existential rules. These two factors make the goal of efficiently computing solutions for ABox abduction very challenging. In this section, we develop an algorithm for ABox abduction using techniques of program transformations and query rewriting. Specifically, the algorithm for ABox abduction consists of two phases: The first phase, consisting of two program transformations, transforms the given TBox into a Prolog program and then, the second phase rewrites the observation in order to guarantee that the soundness of the first phase in our algorithm.

3.1 Ontology Transformations for ABox Abduction

For an $\mathcal{ELH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$ with a normalized TBox, we will first transform $\mathcal{T}$ into a datalog$^{\pm}$ program and then further transform it into a datalog program. The primary goal of using these transformations here is to reduce the problem of ABox abduction into the problem of abduction in datalog.

An $\mathcal{ELH}$ TBox $\mathcal{T}$ can be transformed into a datalog$^{\pm}$ program $\Sigma_{\mathcal{T}}$ as shown in Table 1 [4]. Since $\mathcal{T}$ and $\Sigma_{\mathcal{T}}$ are equivalent as theories in first-order logic, two corresponding abductive problems $\langle(\mathcal{T}, \mathcal{A}), q(\boldsymbol{a}), S\rangle$ and $\langle \Sigma_{\mathcal{T}} \cup \mathcal{A}, q(\boldsymbol{a}), S\rangle$ are equivalent too, for any ABox $\mathcal{A}$ and abducibles S.

The above transformation reduces the problem of ABox abduction into that of abduction in datalog$^{\pm}$ but there is no efficient datalog$^{\pm}$ abduction engine available. For this reason, we use another transformation to reduce the problem of datalog$^{\pm}$ abduction into that of datalog abduction. Formally, the resulting datalog$^{\pm}$ program $\Sigma_{\mathcal{T}}$ is transformed into a datalog program $\Xi_{\mathcal{T}}$ as specified in Table 2 [23]. That is, $\Xi_{\mathcal{T}}$ is obtained by replacing each existential rule in datalog$^{\pm}$ program $\Sigma_{\mathcal{T}}$ with two datalog rules. As mentioned in [23], $\Xi_{\mathcal{K}}$ can be computed in linear time to the size of $\mathcal{K}$.

This following example illustrates the two steps of transformations.

$\mathcal{ELH}$ axiom		datalog$^\pm$rule
$A \sqsubseteq B$	$\rightsquigarrow$	$A(x) \to B(x)$
$A_1 \sqcap A_2 \sqsubseteq B$	$\rightsquigarrow$	$A_1(x) \land A_2(x) \to B(x)$
$A \sqsubseteq \exists r.B$	$\rightsquigarrow$	$A(x) \to \exists y.(r(x,y) \land B(y))$
$\exists r.A \sqsubseteq B$	$\rightsquigarrow$	$r(x,y) \land A(y) \to B(x)$
$r \sqsubseteq s$	$\rightsquigarrow$	$r(x,y) \to s(x,y)$

Table 1. Transformation from $\mathcal{ELH}$ to datalog$^\pm$.

datalog$^\pm$rule		datalog rules
$A(x) \to \exists y.(r(x,y) \land B(y))$	$\rightsquigarrow$	$A(x) \to r(x, c_{r,B})$ $A(x) \to B(c_{r,B})$ $c_{r,B}$ is a fresh constant.

Table 2. Transformation from datalog$^\pm$ to datalog.

Example 1. Let TBox $\mathcal{T}$ consist of the following two axioms T1 and T2:

T1) RA $\sqsubseteq$ Person $\sqcap$ $\exists$works.RG
T2) Student $\equiv$ Person $\sqcap$ $\exists$takes.Course

The axiom T1 says that a research assistant (RA) is a person and (s)he works in some research group (RG) and T2 expresses that a student is a person who takes some courses.

Then the corresponding datalog$^\pm$ program $\Sigma_\mathcal{T}$ has five rules R1–R5, where R5 is an abbreviation of two rules and the auxiliary concept introduced via normalization is omitted for brevity.

R1) RA$(x) \to \exists y$ works$(x,y) \land$ RG(y)
R2) RA$(x) \to$ Person(x)
R3) Student$(x) \to \exists y$ takes$(x,y) \land$ Course(y)
R4) Student$(x) \to$ Person(x)
R5) Person$(x) \land$ takes$(x,y) \land$ Course$(y) \to$ Student(x)

Then $\Sigma_\mathcal{T}$ is transformed into the datalog program $\Xi_\mathcal{T}$ containing R2, R4, R5, and the following datalog rules S1–S4:

S1) RA$(x) \to$ works(x, sc_1) S2) RA$(x) \to$ RG(sc_1)
S3) Student$(x) \to$ takes(x, sc_2) S4) Student$(x) \to$ Course(sc_2)

For an $\mathcal{ELH}$ KB $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, the program $\Xi_\mathcal{K} = \Xi_\mathcal{T} \cup \mathcal{A}$ is referred to as its *datalog approximation.* The following theorem states that the datalog approximation of each ontology is complete for ABox abduction. That is, each abductive solution for ontology $\mathcal{K}$ is an abductive solution for datalog program $\Xi_\mathcal{K}$.

Proposition 1. *Let $\mathcal{K}$ be an $\mathcal{ELH}$ KB, $q(\boldsymbol{a})$ be an observation, and S be a set of abducibles. Then, $sol(\mathcal{K}, q(\boldsymbol{a}), S) \subseteq sol(\Xi_\mathcal{K}, q(\boldsymbol{a}), S)$.*

However, the converse of Proposition 1 may not hold, that is, the approximation is not necessarily sound. We demonstrate this in the following two examples, respectively.

Example 2. Consider the ABox abduction problem $\langle \mathcal{K}, O, S \rangle$, where $\mathcal{K} = (\mathcal{T}, \mathcal{A})$, $\mathcal{T}$ is the TBox in Example 1, $\mathcal{A} = \{\text{Student}(john), \text{Student}(julie), \text{RG}(rg_1)\}$, $S = \{\text{RA}\}$, the observation $O = q(john, julie)$, and the conjunctive query $q(x, z)$ is as follows:

$$q(x, z) = \exists y.(\text{works}(x, y) \wedge \text{works}(z, y) \wedge \text{RG}(y) \wedge x \neq z).$$

Then, a solution to the datalog abduction problem $\langle \Xi_{\mathcal{K}}, O, \{\text{RA}\} \rangle$ is $\mathcal{E} = \{\text{RA}(john), \text{RA}(julie)\}$, which is not a solution to the ABox abduction problem $\langle \mathcal{K}, O, \{\text{RA}\} \rangle$.

The cause of the incorrect solution is illustrated by Figure 2. In particular, in each model (A) of $\Xi_{\mathcal{K}}$, all the research assistants work in the same research group sc_1, whereas in a model (B) of $\mathcal{K}$, two research assistants can work in two distinct research groups.

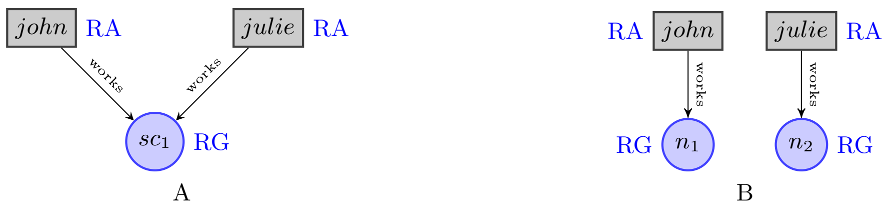

Fig. 2. Fork-shaped model introduced by the datalog approximation.

Example 3. Consider a KB $\mathcal{K}$ whose TBox $\mathcal{T}$ consists of axioms T1 and T2.The datalog approximation for $\mathcal{T}$ consists of three rules R1–R3 as follows:

$$\begin{array}{lcl} \text{T1) A} \sqsubseteq \text{B} & \rightsquigarrow & \text{R1) } \text{A}(x) \rightarrow \text{B}(x). \\ \text{T2) B} \sqsubseteq \exists\text{r.B} & \rightsquigarrow & \text{R2) } \text{B}(x) \rightarrow \text{r}(x, sc_1). \\ & & \text{R3) } \text{B}(x) \rightarrow \text{B}(sc_1). \end{array}$$

Let the ABox consists of exactly one fact $C(a)$. Consider the CQ

$$q(x) = \exists y.(\text{r}(x, y) \wedge \text{r}(y, y)),$$

and observation $q(a)$. Then, a solution to the datalog abduction $\langle \Xi_{\mathcal{K}}, q(a), \{\text{A}\} \rangle$ is $\mathcal{E} = \{\text{A}(a)\}$, which is not a solution to the initial ABox abduction $\langle \mathcal{K}, q(a), \{\text{A}\} \rangle$. The cause of the problem is shown in Figure 3, where $\Xi_{\mathcal{K}}$ enforces an r-cycle in each of its model.

3.2 Observation Rewriting for ABox Abduction

In the last subsection, we have been able to reduce an ABox abduction instance into an abduction instance in datalog. However, the procedure may bring in some unwanted solutions. To retain the soundness of our ontology transformation, we propose to rewrite observations in ABox abduction. Our rewriting approach is adapted

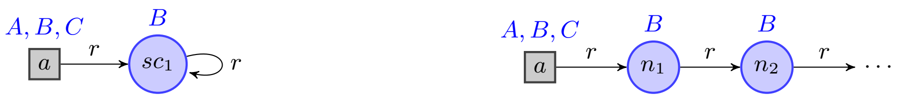

Fig. 3. Cycle model enforced by the datalog approximation.

from [17], which extends the initial CQ with filter queries to detect spurious query matching. We note that, while the rewriting is shown to be sufficient for query answering in [17], it was unknown whether it works for abduction, and in particular for the datalog approximation adopted in our paper. In this section, we show that the rewriting technique can be used in ABox abduction and in particular, our transformation-based procedure and the adapted observation rewriting form a sound and complete algorithm for ABox abduction in $\mathcal{ELH}$.

We first introduce a unary predicate Aux to mark the auxiliary constants which are those fresh constants generated in approximation phase shown in Table 2. All auxiliary constants and only these constants are asserted to be members of Aux in the ABox. Then, we use an equivalence relation $\sim_q$ for each CQ q from [17]. This is the smallest reflexive, symmetric and transitive relation on $term(q)$ that satisfies the following condition:

(*) if $r(s,t), r(s',t') \in q$ with $t \sim_q t'$, then $s \sim_q s'$.

This equivalence relation can be computed in polynomial time to the size of q [17]. For each equivalent class ν of $\sim_q$, a representative $t_\nu \in \nu$ will be chosen. The following two problematic patterns in CQs are responsible for the violation of soundness, corresponding respectively to the cases shown in Examples 2 and 3:

– Let **Fork** be the set of pairs (T, ν) with ν an equivalence class of $\sim_q$, $T = \{t \in term(q) | \exists r \in N_R, \exists t' \in \nu : r(t,t') \in q\}$, and $|T| \geq 2$.

– Let **Cyc** be the set of variables $v \in qvar(q)$ such that there are $r_0(t_0, t_0'), \cdots, r_m(t_m, t_m'), \cdots, r_n(t_n, t_n') \in q$, $n, m \geq 0$, with $v \sim_q t_i$ for some $i \leq n$, $t_i' \sim_q t_{i+1}$ for all $i < n$ and $t_n' \sim_q t_m$.

Fork and **Cyc** can be also computed in time polynomial to the size of q [17].

Given an observation $q(\boldsymbol{a})$, it is rewritten to $q^*(\boldsymbol{a})$, where $q^*(\boldsymbol{x})$ is the first-order query $q(\boldsymbol{x}) \wedge q_1 \wedge q_2$ where

$$q_1 := \bigwedge_{(\{t_1,\cdots,t_k\},\nu)\in\textbf{Fork}} (Aux(t_\nu) \rightarrow \bigwedge_{1\leq i<k} t_i = t_{i+1})$$
$$q_2 := \bigwedge_{v\in\textbf{Cyc}} \neg Aux(v)$$

Intuitively, filter query q_1 says that if a fork-shaped pattern occurs in q involving an existential variable then both edge should merge, and q_2 says that an existential variable cannot lead to a circle.

Example 4 (Cont'd Example 2). For observation $q(john, julie)$ with

$$q(x,z) = \exists y.(\text{works}(x,y) \wedge \text{works}(z,y) \wedge \text{RG}(y) \wedge x \neq z),$$

the $\sim_q$ includes equivalence classes $\{x, z\}$ and $\{y\}$. Thus, the rewritten observation is:

$$q^*(john, julie) = q(john, julie) \wedge (Aux(y) \rightarrow (john = julie))$$

In this case, there is no solution to $\langle \Xi_{\mathcal{K}}, q^*(john, julie), \{\text{RA}\}\rangle$, and one solution to $\langle \Xi_{\mathcal{K}}, q^*(john, julie), \{\text{works}\}\rangle$ is $\mathcal{E} = \{\text{works}(john, rg_1), \text{works}(julie, rg_1)\}$.

Example 5 (Cont'd Example 3). For observation $q(a)$ with $q(x) = \exists y.(\text{r}(x, y) \wedge \text{r}(y, y))$, the rewritten observation is $q^*(a) = q(a) \wedge \neg Aux(y)$. There is no solution to $\langle \Xi_{\mathcal{K}}, q^*(a), \{\text{A}\}\rangle$, and the minimal solution to $\langle \Xi_{\mathcal{K}}, q^*(a), \{\text{r}\}\rangle$ is $\mathcal{E} = \{r(a, a)\}$.

Note that our rewriting is simpler than that of [17], 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 [17], 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 [17].

The following result shows the new rewriting is sufficient for our approximation, which is unknown before.

Proposition 2. *Let $\mathcal{T}$ be an $\mathcal{ELH}$ TBox, q be a CQ, and $\boldsymbol{a}$ be a tuple of constants. Then for each ABox $\mathcal{A}$, $(\mathcal{T}, \mathcal{A}) \models q(\boldsymbol{a})$ iff $\Xi_{\mathcal{T}} \cup \mathcal{A} \models q^*(\boldsymbol{a})$.*

Now, we can show that our ontology transformation and observation rewriting together grant sound and complete ABox abduction computing.

Theorem 1. *Let $\mathcal{K}$ be an $\mathcal{ELH}$ KB, $q(\boldsymbol{a})$ be an observation, and S be a set of abducibles. Then, $sol(\mathcal{K}, q(\boldsymbol{a}), S) = sol(\Xi_{\mathcal{K}}, q^*(\boldsymbol{a}), S)$.*

4 Encoding ABox Abduction in Prolog

To compute solutions to an ABox abduction problem, we encode the datalog translation of ABox abduction problem in Prolog and make use of highly optimized Prolog engine XSB. While datalog is a sub-language of Prolog, as we will show later, such an encoding is far from straightforward. Our encoding is adapted from and extended that of [6,7]. Following the syntax of Prolog, variables start with upper case letters, and predicates, constants and functions start with lower case letters. Also, $\wedge$ is replaced by comma (`,`). In what follows, we present the Prolog program $\Pi(\mathcal{K}, q, \boldsymbol{a}, S)$ that encodes the datalog abduction problem $\langle \Xi_{\mathcal{K}}, q^*(\boldsymbol{a}), S\rangle$.

As in [7], we assign a binary predicate *p_A* for each $A \in N_C$ and a ternary predicate *p_r* for each $r \in N_R$. The increase of arity is for an extra parameter to store solutions. In particular, the solutions are stored and manipulated in an inbuilt list structure in Prolog. Hence, we have atoms of the forms `p_A(X,L)` and `p_r(X,Y,L)` where `X`, `Y` are variables and `L` is a list. To encode $\Xi_{\mathcal{K}}$, we encode each datalog rule as a Prolog rule. For example, a datalog rule $r(x, y) \wedge B(y) \rightarrow A(x)$ is encoded as

```
p_A(X,L) :- p_r(X,Y,L1), p_B(Y,L2), merge(L1,L2,L).        (1)
```

where `merge(L1, L2, L)` merges lists `L1` and `L2` to `L`.

In contrast to the encoding in [6,7], we need to handle datalog rules of the forms $A(x) \to r(x, c)$ and $A(x) \to B(c)$ as a pair, where c is an auxiliary constant with $Aux(c)$ asserted. This is non-trivial due to the following two issues. Firstly, Prolog does not support constants in a rule. To handle constants, we apply some standard handling in logic programming to "move" the constants into the facts. In particular, we assign a fresh unary predicate `p_c` to each auxiliary constant c, and assert c and only c to be a member of `p_c`. The above rules and facts can be encoded as follows:

```
p_r(X,Y,L):- p_A(X,L), p_c(Y).       p_c(c).       p_Aux(c).    (2)
  p_B(Y,L):- p_A(X,L), p_c(Y).                                  (3)
```

Secondly, while the above Prolog rules faithfully capture the semantics of the corresponding datalog rules, they can dramatically affect the efficiency of the resolution. More specifically, when the resolution introduces both `p_r(a,c,L)` and `p_B(c,L)`, rule (3) will generate a large number of redundant facts `p_A(b,L)`, which consumes a large amount of time and memory. To address this issue, we replace rule (3) with

```
p_B(Y,L) :- p_r(X,Y,L), p_c(Y).        (4)
```

Such a replacement does not affect computation of abductive solutions, as for any Prolog program P containing neither constant `c` nor predicate `p_c`, $P \cup \{(2), (4)\}$ behaves exactly as $P \cup \{(2), (3)\}$ when it comes to query answering and abduction.

Facts like $r(a, b)$ and $A(a)$ are encoded as `p_r(a,b,[]).` and `p_A(a,[]).` respectively, where `[]` is an empty list. Moreover, a fact `ind(a).` is added to the program for each individual a occurring in the ABox. And for the lists to work properly, that is, they store solutions consisting of only facts constructed from abducibles in S and constants from the ABox, the following Prolog rules need to be included:

- for each concept name $A \in S$, `p_A(X,[p_A(X)]) :- ind(X).`
- for each role name $r \in S$, `p_r(X,Y,[p_r(X,Y)]) :- ind(X), ind(Y).`

To encode observation $q^*(\boldsymbol{a})$, we first need to encode CQ $q^*(\boldsymbol{x})$. A CQ of the form $\bigwedge_{1 \le i \le n} B_i(\boldsymbol{x}_i) \to q(\boldsymbol{x})$ where each $B_i(\boldsymbol{x}_i)$ is a body atom and $\boldsymbol{x} \subseteq \bigcup_{1 \le i \le n} \boldsymbol{x}_i$, is naturally encoded as a Prolog rule

$$\texttt{q}(\mathbf{X}\texttt{,L) :- p\_B}_1\texttt{(}\mathbf{X}_1\text{,L1}), \ldots, \texttt{p\_B}_n\texttt{(}\mathbf{X}_n\text{,Ln}), \text{merge(L1,} \ldots \text{,Ln,L}\texttt{).} \qquad (5)$$

where $\mathbf{X}$ and $\mathbf{X}_i$ correspond to $\boldsymbol{x}$ and $\boldsymbol{x}_i$, respectively, and `merge(L1,...,Ln,L)` is an abbreviation. For example, `merge(L1,L2,L3,L)` abbreviates `merge(L1,L2,L12), merge(L12,L3,L)`. For the filter queries $q_1(\boldsymbol{x})$ and $q_2(\boldsymbol{x})$, the sets **Fork** and **Cyc** are pre-computed via a Java program. Then the encoding of $q^*(\boldsymbol{x})$ is obtained by adding the following atoms to the body of (5):

- for each pair $(\{t_1, \ldots, t_k\}, \nu)$ in **Fork**, where $t_i \in term(q)$ and ν is an equivalent class of $\sim_q$ with representative t_ν, add

$$\texttt{p\_Aux(T}_\nu\texttt{) ->(T}_1\texttt{== T}_2, \ldots, \texttt{T}_i\texttt{== T}_{i+1}\texttt{); true).}$$

where $\mathtt{T}_\nu$ and $\mathtt{T}_i$ correspond to t_ν and t_i, respectively, and if t_ν or t_i is a constant, it is moved into the facts as above. The symbol `->` in Prolog is a *If-Then-Else* command, that is, the sentence before `->` is the *If* part; the sentence between `->` and the semicolon (`;`) is the *Then* part; and reminder is the *Else* part. If the *Else* part is empty, the rule fails, therefore the `true` sentence is used;
– for each variable v in **Cyc**, add `not p_Aux(V)`, where `not` is *negation as failure* (instead of classical negation) but it behaves as required.

Finally, the following rule encodes the observation and starts the execution.

$$\texttt{go :- q*(a,L), add\_answer(L), fail.} \tag{6}$$

where `q*(a,L)` is an atom that encodes the observation. The `add_answer(L)` directive verifies `L` is minimal, and we have implemented two versions of it regarding the cardinality and set minimality. A solution is kept if and only if it is minimal. The `fail` directive is necessary for the execution.

Example 6 (Cont'd Examples 1 and 2). In this example, Prolog rules P1–P5 encode datalog rules R2,R4–R5,and S1–S4; F1–F3 encode the facts in $\mathcal{A}$. Consider the abducibles $S = \{\text{works,RA}\}$, and I1–I2 encode the initialization of solutions. Finally, recall the CQ $q(x,z) = \exists y.(\text{works}(x,y) \wedge \text{works}(z,y) \wedge \text{RG}(y) \wedge x \neq z)$ and consider observation $q(\mathit{john},\mathit{julie})$, which are encoded as O.

Then XSB outputs a single list `[p_works(john,rg1),p_works(julie,rg1)]`, which corresponds to the minimal solution to $\langle \mathcal{K}, q(\mathit{john},\mathit{julie}), \{\text{works,RA}\}\rangle$.

```
P1) p_works(X,Y,L) :- p_RA(X,L), p_sc1(Y).         p_Aux(sc1).
    p_RG(Y,L) :- p_works(X,Y,L), p_sc1(Y).         p_sc1(sc1).
P2) p_Person(X,L) :- p_RA(X,L).
P3) p_takes(X,Y,L) :- p_Student(X,L), p_sc2(Y).    p_Aux(sc2).
    p_Course(Y,L) :- p_takes(X,Y,L), p_sc2(Y).     p_sc2(sc2).
P4) p_Person(X,L) :- p_Student(X,L).
P5) p_Student(X,L) :- p_Person(X,L1), p_takes(X,Y,L2),
                merge(L1,L2,L12), p_Course(Y,L3), merge(L12,L3,L).
F1) p_Student(john,[]). ind(john).              F3) p_RG(rg1,[]).
F2) p_Student(julie,[]). ind(julie).                ind(rg1).
I1) p_works(X,Y,[p_works(X,Y)]) :- ind(X), ind(Y).
I2) p_RA(X,[p_RA(X)]) :- ind(X).
O) go :- q(john,julie,L), add_answer(L), fail.
q(X,Z,L) :- p_works(X,Y,L1), p_works(Z,Y,L2), p_RG(Y,L3), not(X==Z),
   (p_Aux(Y) -> (X==Z); true), merge(L1,L2,L12), merge(L12,L3,L).
```

For the Prolog program Π obtained above, let $\mathit{list}(\Pi)$ be the set of lists returned by Π when `go` is called. As the minimality check is implemented for cardinality and set minimality, we distinguish the outputs of the respective implementations as $\mathit{list}_{\leq}(\Pi)$ and $\mathit{list}_{\subseteq}(\Pi)$ when necessary. For a list $L = [s_1, \ldots, s_n]$ where each s_i $(1 \leq i \leq n)$ is a string of the forms `p_A(a)` or `p_r(a,b)`, let $\mathcal{A}_L$ denotes the ABox corresponding to L, i.e., $\mathcal{A}_L = \{\alpha_1, \ldots, \alpha_n\}$ where $\alpha_i = A(a)$ if s_i is `p_A(a)` and $\alpha_i = r(a,b)$ if s_i is `p_r(a,b)`. The following result shows that the Prolog encoding is sufficient to retrieve all ABox abductive solutions.

Proposition 3. *Given an ABox abduction problem $\langle \mathcal{K}, q(\boldsymbol{a}), S\rangle$, let $\Pi = \Pi(\mathcal{K}, q, \boldsymbol{a}, S)$. Then, for each $L \in list(\Pi)$, (1) $pred(\mathcal{A}_L) \subseteq S$ and (2) $\mathcal{K} \cup \mathcal{A}_L \models q(\boldsymbol{a})$. Conversely, let $\preceq \in \{\leq, \subseteq\}$, then for each $\preceq$-minimal ABox $\mathcal{E}$ satisfying (1) and (2), there exists a list $L \in list_{\preceq}(\Pi)$ such that $\mathcal{A}_L = \mathcal{E}$.*

Note that the definition of an abductive solution requires the observation to be not derivable from the solution alone and the solution to be consistent with the background KB, i.e., conditions (3) and (4) in Definition 1. For condition (3), we have implemented a simple checker to verify this. And condition (4) is trivial in $\mathcal{ELH}$ since $\mathcal{K} \cup \mathcal{E}$ is always consistent. However, for more expressive DLs where inconsistency is an issue, we can use an existing DL reasoner (such as HermiT) to verify it.

Theorem 2. *Given an ABox abduction problem $\langle \mathcal{K}, q(\boldsymbol{a}), S\rangle$, let $\preceq \in \{\leq, \subseteq\}$ and $\Pi = \Pi(\mathcal{K}, q, \boldsymbol{a}, S)$. Then,*

$$\min_{\preceq}(sol(\mathcal{K}, q(\boldsymbol{a}), S)) = \{\mathcal{A}_L \mid L \in list_{\preceq}(\Pi), \mathcal{A}_L \not\models q(\boldsymbol{a}), \mathcal{K} \cup \mathcal{A}_L \textit{ is consistent}\}.$$

5 Experimental Results

Our algorithm for ABox abduction in $\mathcal{ELH}$ has been implemented in Java, adopting the KARMA system [23] for datalog transformation and XSB 3.4 for compiling and executing the Prolog programs. We have evaluated the new prototype system using two sets of experiments on two ontology benchmarks. The experimental results demonstrate that our algorithm is efficient and able to find explanations for negative answers to CQs in large and realistic ontologies. Our system and testing data are available at `http://www.ict.griffith.edu.au/~kewen/AbductionEL/`.

The first benchmark used in our experiments is the SEMINTEC ontology, a financial ontology developed at the University of Poznan. The second benchmark, LSTW [16], is an extended version of LUBM [10] with more axioms containing existential restrictions on the right-hand side (i. e. existential rules). Both ontologies are widely used for benchmarking ontology-based CQ answering and abduction [23,16,7]. We tested on LSTW with one (LSTW1), five (LSTW5), and ten (LSTW10) universities, respectively. Table 3 shows statistics of the ontologies: the numbers of their concept names (#C), role names (#R), TBox axioms ($|\mathcal{T}|$), existential rules (#E), ABox assertions ($|\mathcal{A}|$), and individuals (#I).

Ontology	#C	#R	$\lvert\mathcal{T}\rvert$	#E	$\lvert\mathcal{A}\rvert$	#I
SEMINTEC	60	16	207	8	65,244	17,945
LSTW(n)	132	32	223	29	$\approx 10^5$n	$\approx 1.7 \times 10^4$n

Table 3. Ontology statistics.

All the experiments were performed on a Windows 7 PC with an Intel Xenon 2.8 GHz processor and 16 GB RAM.

For the first set of experiments, we used SyGENiA [13], a query generator for benchmarking CQ answering, to generate related CQs for each ontology. As SyGENiA failed to terminate on some instances of SEMINTEC, we set a bound on the chase algorithm in SyGENiA. We generated 32 and 27 CQs for SEMINTEC and LSTW ontologies, additionally to 18 and 13 atomic queries, respectively. Then, to generate observations, each CQ is grounded with one tuple of ABox individuals that are verified to be a negative answer of the query. All abducibles are concept names because all existential rules have only an atomic concept in the body of the rule. In addition, we randomly generated 15 and 25 abducibles for each of the above ontologies and observations. Thus, one ontology, one observation, and one set of abducibles form a *test case*. Then our implementation was used to compute abductive solutions for these test cases. To avoid generating trivial abduction instances, we require the set of abducibles to be disjoint with the set of predicates in queries. We set a timeout of 5, 6, 10, and 20 minutes for SEMINTEC, LSTW1, LSTW5, and LSTW10, respectively. Table 4 shows the success rates (i.e. the number of terminated test cases within the time and memory limit divided by the total number of test cases), the compilation times (Compilation), the loading and computation times (Computation), and the numbers of solutions generated (#S) by XSB. We present average (Avg.) and maximal (Max.) times over the relevant test cases, and all times are in seconds.

Ontologies	Success Rate	Compilation		Computation		#S	
		Avg.	Max.	Avg.	Max.	Avg.	Max.
15 Abducibles							
SEMINTEC	90%	17.56	21.82	5.6	238.68	0.8	5
LSTW1	85%	11.19	11.42	17.33	188.06	2.24	6
LSTW5	82%	74.69	76.27	71.32	390.69	2.27	6
LSTW10	82%	167.33	171.51	142.89	1223.14	2.27	8
25 Abducibles							
SEMINTEC	86%	19.95	22.39	0.49	1.82	1.44	7
LSTW1	82%	11.22	11.51	41.18	314.0	3.94	15
LSTW5	68%	74.36	76.38	79.05	525.25	3.67	15
LSTW10	68%	169.12	172.77	173.0	1483.93	3.56	13

Table 4. Abductive solutionss w.r.t. generated queries.

Since SyGENiA does not generate a fork-shaped or circle query, we conducted the second set of experiments to evaluate our algorithm for fork-shaped and circle queries. We manually created 4 CQs for SEMINTEC and 4 CQs for LSTW ontologies based on the existential rules in the TBox, including 7 fork-shaped queries and 1 circle query with one instantiation of the answer variables of each query. The circle query (i.e. Q6) is created for LSTW as it contains a recursive axiom Professor $\sqsubseteq$ $\exists$advisor.Professor, while no such axiom exists in SEMINTEC. 15 atomic concepts were automatically generated as abducibles for each ontology and each query. We set a timeout of 30 minutes for the execution of each query. Table 5 shows the computation times (Time) and the numbers of solutions generated (#S) by XSB. Times are in seconds.

Query	SEMINTEC		Query	LSTW1		LSTW5		LSTW10	
	Time	#S		Time	#S	Time	#S	Time	#S
Q1	1.17	0	Q5	6.786	0	41.855	0	84.381	0
Q2	0.889	0	Q6	0.39	0	2.605	0	5.46	0
Q3	0.718	0	Q7	968.61	0	Timeout	-	Timeout	-
Q4	0.78	0	Q8	11.856	0	85.239	0	145.95	0

Table 5. Abductive solutions w.r.t. manually created queries.

Considering the high complexity of the problem, the relatively high success rates and reasonable amount of time taken for computing abductive solutions suggest that our algorithm can be used to handle problems of finding explanations for practical ontologies and queries in $\mathcal{ELH}$.

6 Conclusion

We have developed a practical algorithm for ABox abduction w.r.t. observations in the form of conjunctive queries (CQs) and background ontologies in $\mathcal{ELH}$. The algorithm involves transforming an $\mathcal{ELH}$ ontologies to a datalog program and rewriting the observation using adapted query rewriting techniques. While the ontology transformations and rewriting techniques have been intensively studied for query answering, our work initiates to apply techniques from query answering in ABox abduction. Indeed, by adapting these techniques, we provide, to the best of our knowledge, the first sound and complete algorithm for ABox abduction that supports both CQs and ontologies with existential rules. Moreover, the potential of using our algorithm to handle large ontologies in practice is justified through our evaluation.

Interesting future work includes, firstly, extending the current approach to DL-Lite and more expressive Horn DLs. The former is, as far as we can see, relatively feasible, yet it is non-trivial to develop a well-behaved ontology transformation and an observation rewriting approach. The approach in [15] can be a starting point. Secondly, 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. Finally, as our current implementation uses XSB as a black box, certain interventions with the Prolog system may allow for significant improvements to the efficient of our prototype implementation.

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ Envelope. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence. pp. 364–369 (2005)
2. Borgida, A., Calvanese, D., Rodriguez-Muro, M.: Explanation in the DL-Lite Family of Description Logics. In: On the Move to Meaningful Internet Systems: OTM, pp. 1440–1457 (2008)
3. Borgida, A., Franconi, E., Horrocks, I., McGuinness, D.L., Patel-Schneider, P.F.: Explaining ALC Subsumption. In: Proc. of the 14th ECAI. pp. 209–213 (2000)
4. Calì, A., Gottlob, G., Lukasiewicz, T., Pieris, A.: Datalog+/-: A Family of Languages for Ontology Querying. In: Datalog Reloaded Workshop. pp. 351–368 (2010)
5. Calvanese, D., Ortiz, M., Šimkus, M., Stefanoni, G.: Reasoning about Explanations for Negative Query Answers in DL-Lite. JAIR 48, 635–669 (2013)
6. Du, J., Qi, G., Shen, Y.D., Pan, J.Z.: Towards Practical ABox Abduction in Large OWL DL Ontologies. In: Proc. of the 25th AAAI Conference. pp. 1160–1165 (2011)
7. Du, J., Wang, S., Qi, G., Pan, J., Hu, Y.: A New Matchmaking Approach Based on Abductive Conjunctive Query Answering. In: The Semantic Web - Joint International Semantic Technology Conference, pp. 144–159 (2012)
8. Eiter, T., Gottlob, G.: The Complexity of Logic-Based Abduction. Journal of the ACM 42(1), 3–42 (1995)
9. Eiter, T., Ortiz, M., Simkus, M.: Conjunctive query answering in the description logic sh using knots. J. Comput. Syst. Sci. 78(1), 47–85 (2012)
10. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for {OWL} knowledge base systems. Web Semantics 3(23), 158 – 182 (2005)
11. Halland, K., Britz, K.: ABox Abduction in ALC Using a DL Tableau. In: Proceedings of the South African Institute for Computer Scientists and Information Technologists Conference. pp. 51–58 (2012)
12. Horridge, M., Parsia, B., Sattler, U.: Laconic and Precise Justifications in OWL. In: Proceedings of the 7th ISWC. pp. 323–338 (2008)
13. Imprialou, M., Stoilos, G., Grau, B.C.: Benchmarking Ontology-Based Query Rewriting Systems. In: Proceedings of the 26th AAAI Conference (2012)
14. Klarman, S., Endriss, U., Schlobach, S.: ABox Abduction in the Description Logic **ALC**. Journal of Automated Reasoning 46(1), 43–80 (2011)
15. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in DL-Lite. In: Proceedings of KR-10 (2010)
16. Lutz, C., Seylan, n., Toman, D., Wolter, F.: The Combined Approach to OBDA: Taming Role Hierarchies Using Filters. In: Proc. of the 12th ISWC, pp. 314–330 (2013)
17. Lutz, C., Toman, D., Wolter, F.: Conjunctive Query Answering in the Description Logic EL using a Relational Database System. In: Proc. of the 20th IJCAI (2009)
18. Ma, Y., Gu, T., Xu, B., Chang, L.: An abox abduction algorithm for the description logic alci. In: Intelligent Information Processing. pp. 125–130 (2012)
19. McGuinness, D.L., Borgida, A.T.: Explaining Subsumption in Description Logics. In: Proceedings of IJCAI. pp. 816–821 (1995)
20. McGuinness, D.L., Patel-Schneider, P.F.: Usability issues in knowledge representation systems. In: Proc. of the 15th AAAI Conference. pp. 608–614 (1998)
21. Peñaloza, R., Sertkaya, B.: Complexity of Axiom Pinpointing in the DL-Lite Family of Description Logics. In: Proc. of the 19th ECAI. pp. 29–34 (2010)
22. Staab, S., Studer, R.: Handbook on ontologies. Springer (2009)
23. Stefanoni, G., Motik, B., Horrocks, I.: Introducing nominals to the combined query answering approaches for EL. In: Proc. of the 27th AAAI Conference (2013)