Computing Repairs for Inconsistent DL-programs over $\mathcal{EL}$ Ontologies

Thomas Eiter, Michael Fink, and Daria Stepanova

Institute of Information Systems
Vienna University of Technology
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,fink,dasha}@kr.tuwien.ac.at

Abstract. DL-programs couple nonmonotonic logic programs with DL-ontologies through queries in a loose way which may lead to inconsistency, i.e., lack of an answer set. Recently defined repair answer sets remedy this. In particular, for $DL\text{-}Lite_{\mathcal{A}}$ ontologies, the computation of deletion repair answer sets can effectively be reduced to constraint matching based on so-called support sets. Here we consider the problem for DL-programs over $\mathcal{EL}$ ontologies. This is more challenging than adopting a suitable notion of support sets and their computation. Compared to $DL\text{-}Lite_{\mathcal{A}}$, support sets may neither be small nor few, and completeness may need to be given up in favor of sound repair computation on incomplete support information. We provide such an algorithm and discuss partial support set computation, as well as a declarative implementation. Preliminary experiments show a very promising potential of the partial support set approach.

1 Introduction

Nonmonotonic Description Logic (DL-) programs [29] are a prominent proposal to combine rules and ontologies, following a loose coupling approach (see [21] for an overview of approaches). Due to a bidirectional information flow between rules and the ontology via special *DL-atoms*, they provide a powerful framework for expressing many advanced reasoning applications. However, the loose interaction between rules and the ontology can easily lead to inconsistency (lack of answer sets, i.e. models).

Example 1. Consider the DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ in Figure 1 formalizing an access policy over an ontology $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ [4], whose taxonomy (TBox) $\mathcal{T}$ is given by (1)-(3), while (4)-(9) is a sample data part (ABox) $\mathcal{A}$. Besides facts (10), (11) and a simple rule (12), the rule part $\mathcal{P}$ contains defaults (13), (14) expressing that staff members are granted access to project files unless they are blacklisted, and a constraint (15), which forbids that owners of project information lack access to it. Both parts, $\mathcal{P}$ and $\mathcal{O}$, interact via DL-atoms, such as $\mathrm{DL}[Project \uplus projfile;\ StaffRequest](X)$. The latter specifies an update of $\mathcal{O}$, via operator $\uplus$, prior to querying it: i.e. additional assertions $Project(c)$ are considered for each individual c, such that $projfile(c)$ is true in an interpretation of $\mathcal{P}$, before all instances X of $StaffRequest$ are retrieved from $\mathcal{O}$. Inconsistency arises as $john$, the chief of project $p1$ and owner of its files, has no access to them.

As an inconsistent DL-program yields no information, a relevant issue is how to change it in order to gain consistency. In [9], different repair options were discussed

Fig. 1. DL-program Π over a policy ontology

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{Blacklisted} \sqsubseteq \mathit{Staff} \\ (2)\ \mathit{StaffRequest} \equiv \exists \mathit{hasAction}.\mathit{Action} \sqcap \exists \mathit{hasSubject}.\mathit{Staff} \sqcap \exists \mathit{hasTarget}.\mathit{Project} \\ (3)\ \mathit{BlacklistedStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubject}.\mathit{Blacklisted} \\ (4)\ \mathit{StaffRequest}(r1) \quad (5)\ \mathit{hasSubject}(r1, \mathit{john}) \quad (6)\ \mathit{Blacklisted}(\mathit{john}) \\ (7)\ \mathit{hasTarget}(r1, p1) \quad (8)\ \mathit{hasAction}(r1, \mathit{read}) \quad (9)\ \mathit{Action}(\mathit{read}) \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (10)\ \mathit{projfile}(p1); \quad (11)\ \mathit{hasowner}(p1, \mathit{john}); \\ (12)\ \mathit{chief}(Y) \leftarrow \mathit{hasowner}(X, Y), \mathit{projfile}(X); \\ (13)\ \mathit{grant}(X) \leftarrow \mathrm{DL}[\mathit{Project} \uplus \mathit{projfile}; \mathit{StaffRequest}](X), \mathit{not}\ \mathit{deny}(X); \\ (14)\ \mathit{deny}(X) \leftarrow \mathrm{DL}[\mathit{Staff} \uplus \mathit{chief}; \mathit{BlacklistedStaffRequest}](X), \\ (15)\ \bot \leftarrow \mathit{hasowner}(Y, Z), \mathit{not}\ \mathit{grant}(X), \\ \qquad \mathrm{DL}[; \mathit{hasTarget}](X, Y), \mathrm{DL}[; \mathit{hasSubject}](X, Z). \end{array} \right\}$$

and a theoretical framework for repairing inconsistent DL-programs was proposed, in which the ontology ABox (a likely source of errors) is changed such that the modified DL-program has answer sets, called *repair answer sets*. An algorithm to compute the latter was given in [9] as well, which however lacks practicality.

For $DL\text{-}Lite_{\mathcal{A}}$ ontologies, a more effective repair algorithm was given in [10]. It is based on support sets [8] for a DL-atom, which are portions of its input that together with the ABox determine the value of the DL-atom. The algorithm uses complete support families, i.e. stocks of support sets such that the value of each DL-atom under every interpretation can be decided without ontology access. Fortunately, for $DL\text{-}Lite_{\mathcal{A}}$ ontologies complete support families are small and easy to compute.

In this paper, we consider a similar approach for ontologies in $\mathcal{EL}$, which like $DL\text{-}Lite_{\mathcal{A}}$ is another prominent Description Logic that offers tractable reasoning. Despite limited expressivity, $\mathcal{EL}$ ontologies are still useful for many application domains, including biology, medicine, chemistry, policy, etc. Due to range restrictions and concept conjunctions on the left-hand side of inclusion axioms in $\mathcal{EL}$, a DL-atom accessing an $\mathcal{EL}$ ontology can have arbitrarily large and infinitely many support sets in general. While for acyclic TBoxes (which is a property often met in practice [13]) the latter is excluded, complete support set families can be still very large, and constructing as well as managing them might be impractical. This obstructs to a deployment of the approach in [10] to $\mathcal{EL}$ ontologies.

For this reason, we introduce here a more general algorithm for repair answer set computation that operates on incomplete (partial) support families. More specifically, our contributions and advances over previous works [8; 10] are summarized as follows:

- We generalize repair answer set computation to deal with partial support families, such that $\mathcal{EL}$ ontologies can be handled.
- Following [8], we formally define both ground and nonground support sets for $\mathcal{EL}$ ontologies and present techniques for their computation. In contrast to [8; 10], we take advantage of datalog rewritings of queries over an $\mathcal{EL}$ ontology (see also [15]).
- We provide a declarative realization of an algorithm dealing with partial support families for repair answer set computation within the DLVHEX system. For that, we present some experimental results showing very promising potential of the approach.

Fig. 2. Normalized TBox

$$\mathcal{T}_{norm} = \left\{ \begin{array}{l} (1*)\ \mathit{StaffRequest} \sqsubseteq \exists \mathit{hasAction}.\mathit{Action} \\ (2*)\ \mathit{StaffRequest} \sqsubseteq \exists \mathit{hasSubject}.\mathit{Staff} \\ (3*)\ \mathit{StaffRequest} \sqsubseteq \exists \mathit{hasTarget}.\mathit{Project} \\ (4*)\ \exists \mathit{hasAction}.\mathit{Action} \sqsubseteq C_{\exists \mathit{hasA}.A} \\ (5*)\ \exists \mathit{hasSubject}.\mathit{Staff} \sqsubseteq C_{\exists \mathit{hasS}.St} \\ (6*)\ \exists \mathit{hasTarget}.\mathit{Project} \sqsubseteq C_{\exists \mathit{hasT}.P} \\ (7*)\ C_{\exists \mathit{hasA}.A} \sqcap C_{\exists \mathit{hasS}.St} \sqsubseteq C_{\exists \mathit{hasA}.A \sqcap \exists \mathit{hasS}.St} \\ (8*)\ C_{\exists \mathit{hasA}.A \sqcap \exists \mathit{hasS}.St} \sqcap C_{\exists \mathit{hasT}.P} \sqsubseteq \mathit{StaffRequest} \end{array} \right\}$$

As a practical result of this work, we have an implementation of inconsistency tolerant DL-programs over $\mathcal{EL}$-ontologies, which is the first of its kind.

2 Preliminaries

We first briefly recall DL-programs and repair answer sets; see [29; 9] for details.

Syntax. A DL-program is a pair $\mathit{\Pi} = \langle \mathcal{O}, \mathcal{P} \rangle$ of a finite ontology $\mathcal{O}$ and a finite set of rules $\mathcal{P}$ defined as follows.

• $\mathcal{O}$ is an *DL-knowledge base* (or *ontology*) over a signature $\Sigma_o = \langle \mathbf{I}, \mathbf{C}, \mathbf{R} \rangle$ with a set $\mathbf{I}$ of individuals, a set $\mathbf{C}$ of concept names and a set $\mathbf{R}$ of role names. We assume that $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ is a *consistent* $\mathcal{EL}$ KB [26] with *TBox* $\mathcal{T}$ and *ABox* $\mathcal{A}$, which are sets of axioms capturing taxonomic resp. factual knowledge. Concepts C and roles R obey the following syntax, where $A \in \mathbf{C}$ is an atomic concept and $U \in \mathbf{R}$ is an atomic role:

$$C \to A \quad B \to C \mid C \sqcap D \mid \exists R.C \quad R \to U$$

TBox axioms are of the form $B_1 \sqsubseteq B_2$ (inclusion axiom); ABox assertions are of the form $A(a)$ and $U(a, b)$, where $A \in \mathbf{C}$, $U \in \mathbf{R}$ and $a, b \in \mathbf{I}$; A TBox is *normalized*, if all of its axioms have one of the following forms:

$$A_1 \sqsubseteq A_2 \quad A_1 \sqcap A_2 \sqsubseteq A_3 \quad \exists R.A_1 \sqsubseteq A_2 \quad A_1 \sqsubseteq \exists R.A_2,$$

where A_1, A_2, A_3 are atomic concepts. E.g., the axioms (1) and (2) in Example 1 are in normal form, while axiom (3) is not. For any $\mathcal{EL}$ TBox, an equivalent TBox in normal form is constructable in linear time [26] (over an extended signature); Figure 2 shows a normalized form of the TBox in Example 1.

In the sequel, we use P as a generic predicate from $\mathbf{C} \cup \mathbf{R}$ (if distinction is immaterial).

• $\mathcal{P}$ consists of logic program rules r of the form

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, \mathit{not}\ b_{k+1}, \ldots, \mathit{not}\ b_m\,, \tag{1}$$

where $n + m > 0$, all a_i are lp-atoms, and each b_i is either an lp-atom or a DL-atom; here

- an *lp-atom* is a first-order atom $p(\boldsymbol{t})$ with predicate p from a set $\mathbf{P}$ of predicate names disjoint with $\mathbf{C}$ and $\mathbf{R}$, and constants from a set $\mathcal{C}$; we adopt $\mathcal{C} = \mathbf{I}$.
- a *DL-atom* $a(\mathbf{t})$ is of form $\mathrm{DL}[\lambda;\, Q](\boldsymbol{t})$, where

$$\lambda = S_1\ \mathit{op}_1\, p_1, \ldots, S_m\ \mathit{op}_m\, p_m, \quad m \geq 0, \tag{2}$$

s.t. for $1 \leq i \leq m$, $S_i \in \mathbf{C} \cup \mathbf{R}$, $op_i \in \{\uplus, \cup\}$ is an *update operator*, [1] and $p_i \in \mathbf{P}$ is an *input predicate* of the same arity as S_i—intuitively, $op_i = \uplus$ (resp., $op_i = \cup$) increases S_i (resp., $\neg S_i$) by the extension of p_i; $Q(\boldsymbol{t})$ is a *DL-query*, which is either of the form (i) $C(t)$, where C is a concept and t is a term; (ii) $R(t_1, t_2)$, where R is a role and t_1, t_2 are terms; or (iii) $\neg Q'(\boldsymbol{t})$ where $Q'(\boldsymbol{t})$ is from (i)-(ii).

If $n = 0$, the rule r is a *constraint*.

Example 2 (cont'd). The DL-atom $\mathrm{DL}[Project \uplus projfile;\ StaffRequest](X)$ contained in rule (12) of Example 1 first enriches the concept $Project$ in $\mathcal{O}$ by the extension of the predicate $projfile$ in $\mathcal{P}$ via $\uplus$, and then queries the concept $StaffRequest$.

Semantics. The semantics of a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is in terms of its grounding $gr(\Pi) = \langle \mathcal{O}, gr(\mathcal{P}) \rangle$ over $\mathcal{C}$, i.e., $gr(\mathcal{P})$ contains all ground instances of rules r in $\mathcal{P}$ over $\mathcal{C}$. In the remainder, by default we assume that Π is ground.

A (Herbrand) *interpretation* of Π is a set $I \subseteq HB_\Pi$ of ground atoms, where HB_Π is the usual Herbrand base w.r.t. $\mathcal{C}$ and $\mathbf{P}$; I satisfies an lp-atom a, if $a \in I$ and a DL-atom a of the form (2) if

$$\mathcal{O} \cup \tau^I(a) \models Q(\mathbf{c}) \tag{3}$$

where $\tau^I(a) = \bigcup_{i=1}^m A_i(I)$, and $A_i(I) = \begin{cases} \{S_i(\boldsymbol{t}) \mid p_i(\boldsymbol{t}) \in I\}, & \text{for } op_i = \uplus; \\ \{\neg S_i(\boldsymbol{t}) \mid p_i(\boldsymbol{t}) \in I\}, & \text{for } op_i = \cup. \end{cases}$

Satisfaction of a DL-rule r resp. set $\mathcal{P}$ of rules by I is then as usual, where I satisfies $not\ b_j$ if I does not satisfy b_j; I satisfies Π, if it satisfies each $r \in \mathcal{P}$. We denote that I satisfies (is a *model* of) an object o (atom, rule, etc.) by $I \models^{\mathcal{O}} o$.

Example 3 (cont'd). Consider $I = \{projfile(p1),\ hasowner(p1, john),\ chief(john)\}$, which satisfies dl-atom $d = \mathrm{DL}[Project \uplus projfile;\ StaffRequest](r_1)$ of Example 1, as $\mathcal{O} \cup \tau^I(d) \models StaffRequest(r_1)$. However for $\mathcal{O}'$, given by $\mathcal{O}$ without (4) and (5), $\mathcal{O}' \cup \tau^I(d) \not\models StaffRequest(r_1)$ and thus I does not satisfy d under $\mathcal{O}'$.

An *(flp-)answer set* of $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is any interpretation I that is a $\subseteq$-minimal model of the *flp*-reduct Π^I_{FLP}, which maps $\mathcal{P}$ and $I \subseteq HB_\Pi$ to the rule set $\mathcal{P}^I_{FLP} = \{r^I_{FLP} \mid r \in gr(\mathcal{P})\}$, where $r^I_{FLP} = r$ if the body of r is satisfied, i.e., $I \models^{\mathcal{O}} b_i$, for all b_i, $1 \leq i \leq k$ and $I \not\models^{\mathcal{O}} b_j$, for all $k < j \leq m$; otherwise, r^I_{FLP} is void.

A DL-program Π is *inconsistent*, if it has no answer set. An interpretation I is an *(flp-)deletion repair answer set* of $\Pi = \langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$, if it is an *flp*-answer set of some $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', \mathcal{P} \rangle$ where $\mathcal{A}' \subseteq \mathcal{A}$; any such $\mathcal{A}'$ is called a *deletion repair* of Π. Note that we consider arbitrary deletion repairs. One might resort to more refined notions of repair [9], e.g., $\subseteq$-maximal deletion repairs, however resulting in a complexity increase.

Example 4 (cont'd). Program Π is inconsistent; if we remove (6) from $\mathcal{A}$, then $I = \{projfile(p1), hasowner(p1, john), chief(john), grant(r1)\}$, becomes an answer set. Along with the facts (8) and (9) the *flp*-reduct $\mathcal{P}^I_{FLP}$ contains the ground rule (10), where X is substituted by $r1$. Then I is a deletion repair answer set with respect to the repair $\mathcal{A}' = \{Action(read), hasAction(r1, read), StaffRequest(r1), hasSubject(r1, john), hasTarget(r1, p1)\}$.

[1] We disregard here for simplicity the less used constrains-operator $\Cap$ and subsumption queries.

Shifting Lemma. To simplify matters and avoid dealing with the logic program predicates separately, we shall shift as in [10] the lp-input of DL-atoms to the ontology. Given a DL-atom $d = \mathrm{DL}[\lambda;\, Q](\boldsymbol{t})$ and $P \circ p \in \lambda, \circ \in \{\uplus, \uplus\}$, we call $P_p(c)$ an *input assertion for* d, where P_p is a fresh ontology predicate and $c \in \mathcal{C}$; $\mathcal{A}_d$ is the set of all such assertions. For a TBox $\mathcal{T}$ and a DL-atom d, we let $\mathcal{T}_d = \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\}$, and for an interpretation I, let $\mathcal{O}_d^I = \mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\boldsymbol{t}) \in \mathcal{A}_d \mid p(\boldsymbol{t}) \in I\}$. We then have:

Proposition 1 ([10]). *For every $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$, DL-atom $d = \mathrm{DL}[\lambda;\, Q](\boldsymbol{t})$ and interpretation I, it holds that $I \models^{\mathcal{O}} d$ iff $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon;\, Q](\boldsymbol{t})$ iff $\mathcal{O}_d^I \models Q(\boldsymbol{t})$.*

Unlike $\mathcal{O} \cup \tau^I(d)$, in $\mathcal{O}_d^I$ there is a clear distinction between native assertions and input assertions for d w.r.t. I (via facts P_p and axioms $P_p \sqsubseteq P$), mirroring its lp-input.

3 Support Sets for DL-atoms

In this section, we provide a definition of support sets using the framework given in [8]. Intuitively, a *support set* for a DL-atom d is a portion of its input that determines the output values of d.

Definition 1 (Ground Support Sets). *Let $d(\boldsymbol{c}) = \mathrm{DL}[\lambda;\, Q](\boldsymbol{c})$ be a ground DL-atom of a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$. Then a* support set *for d is a subset of the Herbrand Base $S = \{p_i(\boldsymbol{t}) \in HB_\Pi, P_i \uplus p_i \in \lambda\}$, s.t. for all interpretations $I, I' \supseteq S$, it holds that $I \models^{\mathcal{O}} d$ iff $I' \models^{\mathcal{O}} d$. Moreover, S is positive (resp. negative), if for every interpretation $I \supseteq S$ it holds that $I \models^{\mathcal{O}} d$ (resp. $I \not\models^{\mathcal{O}} d$).*

In this work we exploit only positive support sets, i.e. portions of the ontology input, which ensure that the DL-atom will be true.

Example 5. Recall Π and $d(r1) = \mathrm{DL}[\mathit{Project} \uplus \mathit{projfile};\, \mathit{StaffRequest}](r1)$ from Example 1. A positive ground support set for $d(r1)$ is $S = \{\mathit{projfile}(p1)\}$. Indeed, for all interpretations $I \supseteq \{\mathit{projfile}(p1)\}$, it holds that $\mathcal{A} \cup \mathcal{T} \cup \lambda^I(d) \models \mathit{StaffRequest}(r1)$.

Intuitively, support sets reflect the relevant part of an external source (ontology in our case). Thus different ground support sets can be similar with respect to their structure. With this motivation in mind in [8] support sets were lifted to the nonground level. The definition of a nonground support set exploits source information in the form of so-called *conditional guards* (γ); we now adapt it to DL-programs.

Definition 2. *Let Π be a DL-program and let $d(\boldsymbol{X}) = \mathrm{DL}[\lambda;\, Q](\boldsymbol{X})$ be a DL-atom of Π. A positive nonground support set S for $d(\boldsymbol{X})$ is a pair $\langle N, \gamma \rangle$, where*

- *$N \subseteq \{p_i(\boldsymbol{Y}) \mid P_i \circ p_i \in \lambda, \circ \in \{\uplus, \uplus\}\}$ is a set of nonground atoms over the input signature λ of d;*
- *$\gamma : \mathcal{C}^{|\boldsymbol{X}|} \times grnd_{\mathcal{C}}(N) \to \{0, 1\}$ is a Boolean function (called the guard), s.t. for all $\boldsymbol{c} \in \mathcal{C}^{|\boldsymbol{X}|}$ and $N_{gr} \in grnd_{\mathcal{C}}(N)$ it holds that $\gamma(\boldsymbol{c}, N_{gr}) = 1$ only if N_{gr} is a ground support set for $d(\boldsymbol{c})$.*

In this definition, $grnd_{\mathcal{C}}(N)$ is the support family, i.e. a set of support sets, constructed from N by replacing all variables with constants from $\mathcal{C}$ in all possible ways. Intuitively, the guard γ is an abstract function that checks a condition under which the ground atoms for predicates in N form a ground support set. A family $\mathbf{S}$ of support sets

is said to be *complete* for a (non-ground) DL-atom $d(\boldsymbol{X})$ iff for every $\boldsymbol{c} \in \mathcal{C}^{|\boldsymbol{X}|}$ and ground support set S of $d(\boldsymbol{c})$, there exists $S' = \langle N, \gamma \rangle \in \mathbf{S}$, such that $S \in grnd_{\mathcal{C}}(N)$ and $\gamma(\boldsymbol{c}, S) = 1$.

Example 6. The DL-atom $d(X) = \mathrm{DL}[Project \uplus projfile;\ StaffRequest](X)$ has $S_1 = \langle projfile(Y), \gamma \rangle$ as a nonground support set, where $\gamma : \mathcal{C} \times grnd_{\mathcal{C}}(projfile(Y)) \rightarrow \{0, 1\}$ is such that $\gamma(c, projfile(c')) = 1$ only if the ABox $\mathcal{A}$ contains the assertions $hasAction(c, c_1)$, $Action(c_1)$, $hasSubject(c, c_2)$, $Staff(c_2)$, and $hasTarget(c, c')$, where c_1, c_2 are arbitrary constants from $\mathcal{C}$.

Another nonground support set for $d(X)$ is $S_2 = \langle \emptyset, \gamma' \rangle$, where $\gamma' : \mathcal{C} \times \emptyset \rightarrow \{0, 1\}$ is such that $\gamma'(c, \emptyset) = 1$ only if $StaffRequest(c) \in \mathcal{A}$.

The abstract definition of nonground support sets leaves room for flexible realization of the conditional guard γ. A natural one is by (unions of) conjunctive queries (UCQs) over the ontology ABox viewed as a database. In Example 6, the guard γ of S_1 takes as input a constant $c \in \mathcal{C}$ and a ground instance of form $projfile(c')$, and returns 1 if the Boolean CQ $q(c) \leftarrow \exists X, X'\ \phi(X, X')$ evaluates to true, where $\phi(X, X') = hasAction(c, X) \wedge Action(X) \wedge hasSubject(c, X') \wedge Staff(X') \wedge hasTarget(c, c')$. The UCQ $q(c) \leftarrow \exists X, X'\ \phi(X, X') \vee \psi(X, X')$, where $\psi(X, X') = hasAction(c, X) \wedge Action(X) \wedge hasSubject(c, X') \wedge Blacklisted(X') \wedge hasTarget(c, X')$, is more general; even more general guards are possible (e.g. nonrecursive datalog programs).

3.1 Computing Support Sets for DL-atoms over $\mathcal{EL}$ Ontology

We now provide a method for support set construction that allows us to just work with ontology predicates when constructing nonground support sets.

As negation is not available nor expressible in $\mathcal{EL}$ ($\bot$ is unavailable), from now on we restrict our attention to DL-atoms $\mathrm{DL}[\lambda;\ Q](\boldsymbol{c})$ with positive updates, i.e. $\circ \in \{\uplus\}$ for all $P \circ p \in \lambda$.

The discussion above reveals that for support set construction, it is natural to exploit (conjunctive) query answering methods in $\mathcal{EL}$ (e.g., [23; 19; 18; 25]). Most of them are based on rewriting the query and the TBox into a datalog program over the ABox; to construct guard functions that use a datalog rewriting of the TBox seems thus suggestive.

Suppose we are given a DL-program $\mathit{\Pi} = \langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is an $\mathcal{EL}$ ontology and a DL-atom $d(\boldsymbol{X}) = \mathrm{DL}[\lambda;\ Q](\boldsymbol{X})$. Our method for constructing nonground support sets for $d(\boldsymbol{X})$ consists of the following three steps.

Step 1. DL-query Rewriting over the TBox. The first step exploits the rewriting of the DL-query Q of $d(\boldsymbol{X})$ over the TBox $\mathcal{T}_d = \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\}$ into a set of datalog rules, see e.g. Figure 3. At the preprocessing stage, the normalization technique is first applied to the TBox $\mathcal{T}_d$. This technique restricts the syntactic form of TBoxes by decomposing complex axioms into syntactically simpler ones. For this purpose, a minimal required set of fresh concept symbols is introduced. Given a TBox $\mathcal{T}_d$, its normalized form $\mathcal{T}_{d_{norm}}$ is computed in linear time [1]. We then rewrite the part of the TBox, relevant for the query at hand, into a datalog program $Prog_{Q, \mathcal{T}_{d_{norm}}}$ using the translation given in Table 1, which is a variant of [22; 28]. When rewriting axioms of the form $A_1 \sqsubseteq \exists R.A_2$ (fourth axiom in Table 1) we introduce fresh constants (o_{A_2}) to represent "unknown" objects. A similar rewriting is exploited in the REQUIEM system (where function symbols are used instead of fresh constants). As a result we obtain:

Axiom	Datalog rule
$A_1 \sqsubseteq A_2$	$A_2(X) \leftarrow A_1(X)$
$A_1 \sqcap A_2 \sqsubseteq A_3$	$A_3(X) \leftarrow A_1(X), A_2(X)$
$\exists R.A_2 \sqsubseteq A_1$	$A_2(X) \leftarrow R(X,Y), A_3(Y)$
$A_1 \sqsubseteq \exists R.A_2$	$R(X, o_{A_2}) \leftarrow A_1(X)$
	$A_2(o_{A_2}) \leftarrow A_1(X)$

Table 1. $\mathcal{EL}$ TBox Rewriting

Lemma 1. *For any data part, i.e., ABox $\mathcal{A}$, and any ground assertion $Q(\boldsymbol{c})$, deciding $Prog_{Q,\mathcal{T}_{d_{norm}}} \cup \mathcal{A} \models Q(\boldsymbol{c})$ is equivalent to checking $\mathcal{T}_{d_{norm}} \cup \mathcal{A} \models Q(\boldsymbol{c})$.*

Step 2. Query Unfolding. The second step proceeds with the standard unfolding of the rules of $Prog_{Q,\mathcal{T}_{d_{norm}}}$ w.r.t. the target DL-query Q. We start with the rule that has Q in the head and expand its body using other rules of the program $Prog_{Q,\mathcal{T}_{d_{norm}}}$. By applying this procedure exhaustively, we get a number of rules which correspond to the rewritings of the query Q over $\mathcal{T}_{d_{norm}}$. Note that it is not always possible to obtain all of the rewritings effectively, since in general there might be infinitely many of them (exponentially many for acyclic $\mathcal{T}$). We discuss possible restrictions in the next section.

Step 3. Support Set Extraction. The last step is devoted to the extraction of nonground support sets from the rewritings computed in Step 2. We select those that contain only predicates from $\mathcal{T}_d$ and obtain a set of rules r of the form

$$Q(\boldsymbol{X}) \leftarrow P_1(\boldsymbol{Y_1}), \dots P_k(\boldsymbol{Y_k}), P_{k+1_{p_{k+1}}}(\boldsymbol{Y_{k+1}}), \dots, P_{n_{p_n}}(\boldsymbol{Y_n}), \tag{4}$$

where each P_i is a native ontology predicate if $1 \leq i \leq k$, and a predicate mirroring lp-input of d otherwise. From such rules r we construct pairs $S = \langle N, \gamma \rangle$, where

- $N = \{p_i(\boldsymbol{Y_i}) \mid P_{ip_i}(\boldsymbol{Y_i}) \in B(r), k+1 \leq i \leq n\}$;
- $\gamma : \mathcal{C}^{|\boldsymbol{X}|} \times grnd_{\mathcal{C}}(N) \rightarrow \{0,1\}$ is such that $\gamma(\boldsymbol{c}, N_{gr}) = 1$ only if $Q(\boldsymbol{c})$ follows from $r \cup \mathcal{A}_d$, where $\mathcal{A}_d = \mathcal{A} \cup \{P_{ip_i}(\boldsymbol{t}) \mid p_i(\boldsymbol{t}) \in N_{gr}\}$.

Then the following holds.

Proposition 2. *Let $d(\boldsymbol{X}) = \mathrm{DL}[\lambda; Q](\boldsymbol{X})$ be a DL-atom of a program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, is an $\mathcal{EL}$ ontology. A set S, constructed using Step 1-Step 3 is a nonground support set for $d(\boldsymbol{X})$.*

Proof. Towards a contradiction assume that S is not a nonground support set for $d(\boldsymbol{X})$. This means that either (1) N is not a set of nonground predicates from λ or (2) the function γ of Definition 2 is not correct.

The predicates of the form P_P in the TBox $\mathcal{T}_d$ are obtained from λ of $d(\boldsymbol{X})$ by construction and clearly so are the predicates $P_{j_{p_j}}$ of each rule r. Thus predicates in N are indeed nonground predicates from the input signature of $d(\boldsymbol{X})$.

Hence the function γ must be incorrect, i.e. some $\boldsymbol{c} \in \mathcal{C}^{|X|}$, and $N_{gr} \in grnd_{\mathcal{C}}(N)$ must exist, s.t. $\gamma(\boldsymbol{c}, N_{gr}) = 1$ but N_{gr} is not a positive ground support set for $d(\boldsymbol{c})$. The latter means that some interpretation $I' \supseteq N_{gr}$ exists s.t. $I' \not\models^{\mathcal{O}} d(\boldsymbol{c})$. By Proposition 1 we have that $\mathcal{O}_d^{I'} \not\models Q(\boldsymbol{c})$, i.e. $\mathcal{T}_d \cup \mathcal{A}_d \cup \{P_{ip_i}(\boldsymbol{t}) \mid p_i(\boldsymbol{t}) \in I'\} \not\models Q(\boldsymbol{c})$. On the other hand, we know that $Q(\boldsymbol{c})$ follows from r and $\mathcal{A}_d \cup \{P_{ip_i}(\boldsymbol{t}) \mid p_i(\boldsymbol{t}) \in I'\}$. Since r is obtained by unfolding of the rules in $Prog_{Q,\mathcal{T}_{d_{norm}}}$, we know that $Q(\boldsymbol{c})$ also follows

Fig. 3. DL-query Rewriting for $\mathrm{DL}[\mathit{Project} \uplus \mathit{projfile};\ \mathit{StaffRequest}](X)$ over $\mathcal{T}_{d_{norm}}$

$$Prog_{Q,\mathcal{T}_{d_{norm}}} = \left\{ \begin{array}{l} (4')\ C_{\exists hasA.A}(X) \leftarrow \mathit{hasAction}(X, Y), \mathit{Action}(Y). \\ (5')\ C_{\exists hasS.St}(X) \leftarrow \mathit{hasSubject}(X, Y), \mathit{Staff}(Y). \\ (6')\ C_{\exists hasT.P}(X) \leftarrow \mathit{hasTarget}(X, Y), \mathit{Project}(Y). \\ (7')\ C_{\exists hasA.A \sqcap \exists hasS.St}(X) \leftarrow C_{\exists hasA.A}(X), C_{\exists hasS.St}(X). \\ (8')\ \mathit{StaffRequest}(X) \leftarrow C_{\exists hasA.A \sqcap \exists hasS.St}(X), C_{\exists hasT.P}(X). \\ (9)\ \ \mathit{Project}(X) \leftarrow \mathit{Project}_{\mathit{projfile}}(X). \end{array} \right\}$$

from $Prog_{Q,\mathcal{T}_{d_{norm}}} \cup \mathcal{A}_d$ and hence from $\mathcal{T}_{dnorm} \cup \mathcal{A}$ by construction of $Prog_{Q,\mathcal{T}_{d_{norm}}}$. Consequently, $\mathcal{T} \cup \mathcal{A}_d \models Q(\boldsymbol{c})$ must hold. □

As shown above, when working with support sets we can restrict ourselves to the ontology predicates and operate only on them. More specifically, rules of the form (4) fully reflect nonground support sets as of Definition 2, and ground instantiations of such a rule over constants from $\mathcal{C}$ implicitly correspond to ground support sets.

According to novel results [15], complete support families can be computed for large classes of ontologies. However, in general there might be exponentially many unfoldings produced at Step 2. Thus, to cope with exponentiality, one might often want to apply reasonable restrictions on the support families.

4 Partial Support Family Computation

In this section we discuss restrictions on the size, structure and number of support sets, which is of interest for practical applications, and we analyze conditions under which all support sets from the restricted category form a complete support family.

In general, unlike for the $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ case, due to possible cyclic dependencies of the form $C \sqsubseteq \exists R.C$ allowed in $\mathcal{EL}$, the explanations of an instance query can be of infinite size and so are the support sets for DL-atoms accessing an $\mathcal{EL}$ ontology.

An analysis of a vast number of ontologies has revealed that in many realistic cases they do not contain (or imply) cyclic axioms [13]; we thus assume that the TBox of the ontology in a given DL-program is acyclic (i.e., does not entail inclusion axioms of form $C \sqsubseteq \exists R.C$). However, even under this restriction support sets can be large in general.

Example 7. If $\mathcal{T}$ implies the following chain of inclusions $\exists R_1.A_1 \sqsubseteq Q, \exists R_2.A_2 \sqsubseteq A_1, \exists R_3.A_3 \sqsubseteq A_2, \ldots, \exists R_n.A_n \sqsubseteq A_{n-1}$, then the set of ground support sets for

$$\mathrm{DL}[R_1 \uplus p_1, R_2 \uplus p_2, \ldots, R_n \uplus p_n, A_n \uplus q;\ Q](c_1)$$

contains $\{p_1(c_1, c_2), p_2(c_2, c_3), p_3(c_3, c_4) \ldots p_n(c_{n-1}, c_n), q(c_n)\}$. Replacing A_i with nested range restrictions and conjunctions would yield support sets of exponential size.

This raises the question of reasonable restrictions on the form and size of support sets, and under which conditions such restrictions still yield complete support families.

Support set size. A natural approach for computing a partial support family is the restriction of the target support set size. We may put a certain bound on the size of support sets that we want to compute and proceed with unfolding of the rules of the datalog program. When a certain unfolding branch reaches the size limit, we stop its further expansion and choose a different branch.

Suppose the size is bounded by n. Under the following conditions on the TBox the set of all support sets of size at most n is complete:

– inclusions do not contain any existential restrictions on the left hand side and the number of conjuncts on the left hand side of all inclusions in the TBox is bounded by n.

– all existential restrictions of form $\exists R.A$ occurring on the left hand side of inclusions are such that A occurs in the TBox elsewhere only in simple atomic concept inclusions.

Number of Support Sets. Another restriction relevant in practice regards the number of support sets. In general, determining the exact number of support sets that is needed to form a complete family for a DL-atom is a hard problem. It is tightly related to counting minimal explanations for an abduction problem, which was analyzed in [16] for propositional theories under various restrictions; there it was shown that counting all smallest solutions (explanations) for an abduction problem over a Horn theory is $\#Opt_P[log_n]$-complete. Moreover, meaningful conditions such that a fixed number n of support sets suffices to obtain a complete family are non-obvious (bounded tree-width [14] might be useful, as for efficient datalog abduction); a careful analysis of real world ontologies is needed to ensure practical relevance. This remains for future research.

5 Algorithm for Repair Answer Set Computation

In this section, we present our algorithm $SoundRAnsSet$ for computing deletion repair answer sets by exploiting support families for DL-atoms accessing an $\mathcal{EL}$ ontology.

Exploiting DLVHEX, DL-programs are evaluated via a rewriting $\hat{\Pi}$ of $gr(\Pi)$, where DL-atoms a are replaced by ordinary atoms e_a (*replacement atoms*), together with a guess on their truth by additional "choice" rules $e_a \vee ne_a$, where ne_a stands for the negation of e_a. We denote interpretations of $\hat{\Pi}$ by $\hat{I}$, and use $\hat{I}|_\Pi$ when referring to their restriction to the original language of Π.

The naive algorithm for repair answer set computation [9] cycles through all ABox candidates $\mathcal{A}'$ and checks whether under $\mathcal{A}'$ the guess for the replacement atoms coincides with their actual values. If $\mathcal{A}'$ fulfills this, an *unfoundedness check* is performed for this repair candidate. An alternative approach [10], specifically targeted at $DL\text{-}Lite_\mathcal{A}$ ontologies, aims at finding repairs using complete support families for DL-atoms. In our algorithm $SoundRAnsSet$ for $\mathcal{EL}$ ontologies (see Algorithm 1) we also exploit support families, but do not require that they are complete. If the families are complete (which may be known), then $SoundRAnsSet$ is guaranteed to be complete; otherwise, it may miss repair answer sets (an easy extension ensures completeness though).

We start (a) by computing a family $\mathbf{S}$ of nonground support sets for each DL-atom. Next the replacement program $\hat{\Pi}$ is created, whose answer sets $\hat{I}$ are computed one by one in (b). For $\hat{I}$, we first determine the sets D_p (resp. D_n) of DL-atoms that are guessed true (resp. false) in it and then use the function $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$ which instantiates $\mathbf{S}$ for the DL-atoms in $D_p \cup D_n$ to relevant ground support sets, i.e., those compatible with $\hat{I}$.

In (d) we check whether some DL-atom in D_n has a support set S consisting just of input assertions; if so we move to the next answer set $\hat{I}$ of $\hat{\Pi}$. Otherwise, we (e) loop over all minimal hitting sets $H \subseteq \mathcal{A}$ of the support sets for DL-atoms in D_n, formed by ABox assertions only. For each H we check whether every atom in D_p has at least one support set disjoint from H. If yes (f), i.e. removing H from $\mathcal{A}$ does not affect the values

of DL-atoms in D_p, then we evaluate in a *postcheck* the atoms from D_n over $\mathcal{T} \cup \mathcal{A} \backslash H$ w.r.t. $\hat{I}$. Otherwise, we evaluate the DL-atoms from D_n and D_p. A Boolean flag rep stores the evaluation result of a function $eval_n$(resp. $eval_p$). More specifically, given D_n (resp. D_p), $\hat{I}$ and $\mathcal{T} \cup \mathcal{A} \backslash H$, the function $eval_n$ (resp. $eval_p$) returns $true$, if all atoms in D_n (resp. D_p) evaluate to false (resp. true). If rep is $true$ and the foundedness check $\mathit{fltpFND}(\hat{I}, \mathcal{A} \setminus H, \mathcal{P})$ succeeds, then in (g) $\hat{I}|_{\Pi}$ is output as repair answer set.

We remark that in many cases, the foundedness check might be trivial [7]; if we would consider weak FLP-answer sets [9], it can be skipped.

Example 8 (cont'd). Consider Π from Example 1 with equivalence ($\equiv$) in axioms (2) and (3) substituted by $\sqsupseteq$. Let $\hat{I} = \{\mathit{projfile}(p1), \mathit{hasowner}(p1, john), \mathit{chief}(john), e_a, ne_b\}$ be returned at (b), where $a = \mathrm{DL}[\mathit{Project} \uplus \mathit{projfile};\ \mathit{Staffrequest}](r1)$ and $b = \mathrm{DL}[\mathit{Staff} \uplus \mathit{chief};\ \mathit{BlacklistedStaffRequest}](r1)$. At (c) we obtained

- $\mathbf{S}^{\hat{I}}_{gr}(a) = \{S_1, S_2\}$, where $S_1 = \{\mathit{hasAction}(r1, read), \mathit{hasSubject}(r1, john), \mathit{Action}(read), \mathit{Staff}(john), \mathit{hasTarget}(r1, p1), \mathit{Project}_{\mathit{projfile}}(p1)\}$ and $S_2 = \{\mathit{StaffRequest}(r1)\}$;
- $\mathbf{S}^{\hat{I}}_{gr}(b) = \{S'_1, S'_2\}$ with $S'_1 = \{\mathit{StaffRequest}(r1), \mathit{hasSubject}(r1, john), \mathit{Blacklisted}(john)\}$ and $S'_2 = \{\mathit{BlacklistedStaffRequest}(r1)\}$.

At (e) we got a hitting set $H = \{\mathit{StaffRequest}(r1), \mathit{BlacklistedStaffRequest}(r1)\}$, which is disjoint with S_1. Thus we get to the if branch of (f) and check whether b is false under $\mathcal{A} \backslash \{\mathit{StaffRequest}(r1)\}$. This is not true, hence $rep = false$ and we pick a different hitting set H', e.g $\{\mathit{Blacklisted}(john), \mathit{BlacklistedStaffRequest}(r1)\}$. Proceeding with H', we get to (g), since at (f) $eval_n(b, \hat{I}, \mathcal{T} \cup \mathcal{A} \cap H) = true$.

Proposition 3. *$SoundRAnsSet$ is sound, it outputs only deletion repair answer sets.*

If the support families are complete, then the postchecks at (f) are redundant. In case the if-condition of (f) is satisfied, we set $rep = true$, otherwise $rep = false$.

Proposition 4. *If for all DL-atoms in Π the support families in $\mathbf{S}$ are complete, then $SoundRAnsSet$ is complete, i.e., it outputs every deletion repair answer set.*

We easily can turn $SoundRAnsSet$ into a complete algorithm, by modifying (e) to consider all hitting sets and not only minimal ones. In the worst case, this means a fallback to almost the naive algorithm (note that all hitting sets can be enumerated efficiently relative to their number).

6 Implementation and Experiments

We have implemented the algorithm within the DLVHEX evaluation framework,[2] thus providing a means to effectively compute some deletion repair answer sets for $\mathcal{EL}$. For support set computation we exploit the REQUIEM tool [22], which produces the rewritings of the target query using datalog rewriting techniques.

More specifically, we proceed as follows: first for each DL-atom we compute query rewritings of a certain size using REQUIEM. We then use a declarative approach

[2] http://www.kr.tuwien.ac.at/research/systems/dlvhex

Algorithm 1: $SoundRAnsSet$: compute deletion repair answer sets

Input: $\Pi{=}\langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$

Output: a set of deletion repair answer sets of Π

(a) compute a set $\mathbf{S}$ of nongr. supp. sets for the DL-atoms in Π

(b) **for** $\hat{I} \in AS(\hat{\Pi})$ **do**

(c) $\quad D_p \leftarrow \{a \mid e_a \in \hat{I}\}$; $D_n \in \{a \mid ne_a \in \hat{I}\}$; $\mathbf{S}^{\hat{I}}_{gr} \leftarrow Gr(\mathbf{S}, \hat{I}, \mathcal{A})$;

(d) $\quad$ **if** *every* $S \in \mathbf{S}^{\hat{I}}_{gr}(a')$ *for* $a' \in D_n$ *fulfills* $S \cap \mathcal{A} \neq \emptyset$ **then**

(e) $\quad\quad$ **for** *all min. hitting sets* $H \subseteq \mathcal{A}$ *of* $\bigcup_{a' \in D_n} \mathbf{S}^{\hat{I}}_{gr}(a')$ **do**

(f) $\quad\quad\quad$ **if** *for every* $a \in D_p$ *some* $S \in \mathbf{S}^{\hat{I}}_{gr}(a)$ *exists s.t.* $S \cap H = \emptyset$*;*

$\quad\quad\quad$ **then** $rep \leftarrow eval_n(D_n, \hat{I}, \mathcal{T} \cup \mathcal{A}\backslash H)$;

$\quad\quad\quad$ **else** $rep \leftarrow eval_n(D_n, \hat{I}, \mathcal{T} \cup \mathcal{A}\backslash H) \wedge eval_p(D_p, \hat{I}, \mathcal{T} \cup \mathcal{A}\backslash H)$;

(g) $\quad\quad\quad$ **if** rep *and* $flpFND(\hat{I}, \langle \mathcal{T} \cup \mathcal{A}\backslash H, \mathcal{P}\rangle)$ **then** output $\hat{I}|_{\Pi}$;

$\quad\quad$ **end**

$\quad$ **end**

end

for computing repair answer sets, in which support detection and minimal hitting set computation are accomplished by rules. To this end, for each DL-atom $a(\boldsymbol{X})$ fresh predicates $S_a(\boldsymbol{X})$, $S^{\mathcal{P}}_a(\boldsymbol{Y})$ and $S^{\mathcal{A},\mathcal{P}}_a(\boldsymbol{Y})$ are introduced, where $\boldsymbol{Y} = \boldsymbol{X}\boldsymbol{X'}$, which intuitively say that $a(\boldsymbol{X})$ has some support set, some support set with only logic program predicates, and some mixed support set, respectively (for simplicity we superficially use uniform variables). Furthermore, rules of the following form are added:

(1) $S_a(\boldsymbol{X}) \leftarrow S^{\mathcal{P}}_a(\boldsymbol{Y})$

(2) $S_a(\boldsymbol{X}) \leftarrow S^{\mathcal{A},\mathcal{P}}_a(\boldsymbol{Y})$

(3) $S^{\mathcal{P}}_a(\boldsymbol{Y}) \leftarrow rb(S^{p}_a(\boldsymbol{Y}))$

(4) $S^{\mathcal{A},\mathcal{P}}_a(\boldsymbol{Y}) \leftarrow rb(S^{\mathcal{A},\mathcal{P}}_a(\boldsymbol{Y})), nd(S^{\mathcal{A},\mathcal{P}}(\boldsymbol{Y}))$

(5) $\bot \leftarrow ne_a(\boldsymbol{X}), S^{\mathcal{P}}_a(\boldsymbol{Y})$

(6) $\bar{P}_{1a}(\boldsymbol{Y}) \vee \ldots \vee \bar{P}_{na}(\boldsymbol{Y}) \leftarrow ne_a(\boldsymbol{X}), S^{\mathcal{A},\mathcal{P}}_a(\boldsymbol{Y})$

(7) $eval_a(\boldsymbol{X}) \leftarrow e_a(\boldsymbol{X}), not\ C_a(\boldsymbol{X}), not\ S_a(\boldsymbol{X})$

(8) $eval_a(\boldsymbol{X}) \leftarrow ne_a(\boldsymbol{X}), not\ C_a(\boldsymbol{X})$

Here $C_a(\boldsymbol{X})$ states that the support family for $a(\boldsymbol{X})$ is known to be complete; such information can be added by facts. The rules (1)-(4) derive information about support sets of $a(\boldsymbol{X})$ under a potential repair; $rb(S)$ stands for a rule body rendering of a support set S, and $nd(S) = not\ \bar{P}_{1a}(\boldsymbol{Y}), \ldots, not\ \bar{P}_{na}(\boldsymbol{Y})$, where $\{P_{1a}(\boldsymbol{Y}), \ldots, P_{na}(\boldsymbol{Y})\}$ is the ontology part of S and $\bar{P}_{ia}(\boldsymbol{Y})$ states that the assertion $P_{ia}(\boldsymbol{Y})$ is marked for deletion. The constraint (5) forbids $a(\boldsymbol{X})$, if guessed false, to have a matching support set with only input assertions; (6) means that if $a(\boldsymbol{X})$ has instead a matching mixed support set, then some assertion from its ontology part must be eliminated. The rule (7) says that if $a(\boldsymbol{X})$ is guessed true and completeness of its support family is not known, then an evaluation postcheck must be performed ($eval_a(\boldsymbol{X})$) if no matching support set is available; rule (8) is similar for $a(\boldsymbol{X})$ guessed false (mind rule (5)).

6.1 Experiments

We have conducted a preliminary experimental evaluation of our approach by considering inconsistent DL-programs over acyclic OWL 2 EL ontologies.

Experimental Setup. We have computed repair answer sets with complete and incomplete support families. The experiments were run on a Linux server with two 12-core AMD 6176 SE CPUs/128GB RAM using DLVHEX 2.3.0; a timeout of 300 secs was set for each run. As benchmarks, we used the following problems.

Access Policy Control. The first benchmark is a slight modification of Example 1 with an additional TBox axiom $Blacklisted \sqsubseteq Unauthorized$. We have run experiments in two settings: (a) with complete support families and (b) with support families obtained under different restrictions, viz. bounded size and cardinality. We considered three ABoxes with 40, 100 and 1000 staff members, respectively, and generated facts of the form $hasowner(p_i, s_i)$, and such that $Staff(s_i), Project(p_i) \in \mathcal{A}$. For the setting where complete support families were computed, we used ABoxes with 100 and 1000 staff members, respectively. For the incomplete scenario we used an ABox with 40 staff members. In each data set, 30% of staff members are unauthorized and 20% are blacklisted. Instances vary on facts $hasowner(p_i, s_i)$. For each s_i, p_i s.t. $Staff(s_i), Project(p_i) \in \mathcal{A}$, a fact $hasowner(p_i, s_i)$ is added to the program with probability $p/100$, where p ranges from 20 to 90 for the complete setting and from 5 to 35 for the incomplete one.

The total average running times (including support set computation and timeouts) for computing the first repair answer set for these settings are shown in Table 2. The number of timeouts per each run is reported in brackets. The columns for the incomplete case show the restriction on support sets we used in their generation, viz. size (resp. number) of support sets bounded by 2 resp. unlimited; the latter means that in fact all support sets were computed, but the system is not aware of the completeness.

We exploit partial completeness for the number restriction case, i.e. if no more support sets for an atom are computed and the number limit is not yet reached, then the support family for the considered atom is complete.

Open Street Map. For the second benchmark, we added rules on top of the ontology developed in the MyITS project,[3] which enhanced personalized route planning with semantic information. The ontology contains 4601 axioms, where 406 axioms are in the TBox and 4195 are in the ABox. The fragment $\mathcal{O}$ relevant for our scenario and the rules P are shown in Figure 4. Intuitively, $\mathcal{O}$ states that building features located inside private areas are not publicly accessible and a covered bus stop is a bus stop with a roof. The rules P check that public stations do not lack public access, using CWA on private areas.

We used the method in [12] to extract data from the OpenStreetMap,[4] and we constructed an ABox $\mathcal{A}$ by extracting the sets of all bus stops (285) and leisure areas (682) of the Irish city Cork, as well as $isLocatedInside$ relations between them (9) (i.e., bus stops located in leisure areas). As the data has been gathered by many volunteers, chances of inaccuracies may be high (e.g. imprecise gps data). As data about roofed bus stops and private areas is not available yet, we randomly made 80% of the bus stops roofed and 60% of leisure areas private. Finally, we added for each bs_i s.t. $isLocatedInside(bs_i, la_j) \in \mathcal{A}$

[3] http://www.kr.tuwien.ac.at/research/projects/myits/

[4] http://www.openstreetmap.org/

p	Complete supp. family		p	Support set size restricted		Support set number restricted	
				sizelim=2	sizelim=∞	numlim=2	numlim=∞
	$\mathcal{A}_{100}$	$\mathcal{A}_{1000}$		$\mathcal{A}_{40}$			
20 (30)	2.28 (0)	13.89 (0)	5 (30)	21.09 (0)	4.35 (2)	2.70 (2)	4.30 (2)
30 (30)	2.27 (0)	13.93 (0)	10 (30)	26.62 (5)	5.50 (3)	6.64 (3)	5.49 (3)
40 (30)	2.28 (0)	14.02 (0)	15 (30)	30.20 (12)	7.53 (5)	3.25 (10)	7.56 (5)
50 (30)	2.29 (0)	14.33 (0)	20 (30)	48.99 (5)	5.21 (4)	3.07 (4)	5.38 (4)
60 (30)	2.28 (0)	14.59 (0)	25 (30)	37.37 (16)	26.41 (6)	4.48 (14)	26.39 (6)
70 (30)	2.29 (0)	15.08 (0)	30 (30)	19.33 (22)	38.75 (6)	6.74 (12)	40.41 (6)
80 (30)	2.30 (0)	15.59 (0)	35 (30)	16.32 (26)	49.41 (10)	5.23 (17)	51.47 (10)
90 (30)	2.30 (0)	16.23 (0)					

Table 2. Policy benchmark results (30 runs per p; time in sec. (#timeouts) for 1st rep. AS)

p	Complete supp. family	Support set size restricted			Support set number restricted		
		sizelim=1	sizelim=2	sizelim=∞	numlim=1	numlim=2	numlim=∞
10 (30)	10.08 (0)	13.87 (0)	13.22 (0)	14.19 (0)	13.82 (0)	13.98 (0)	13.89 (0)
20 (30)	9.36 (0)	23.38 (0)	22.82 (0)	20.32 (1)	20.32 (1)	20.19 (1)	20.29 (1)
30 (30)	9.13 (0)	27.92 (1)	27.48 (1)	20.36 (3)	20.39 (3)	20.18 (3)	20.22 (3)
40 (30)	9.53 (0)	54.63 (3)	54.36 (3)	23.34 (10)	23.31 (10)	23.42 (10)	23.51 (10)
50 (30)	9.62 (0)	76.08 (1)	76.18 (1)	19.61 (13)	19.48 (13)	19.48 (13)	19.64 (13)

Table 3. OpenStreetMap benchmark results (30 runs per p; time in sec. (#timeouts) for 1st rep. AS)

Fig. 4. DL-program Π over OpenStreetMap ontology

$$\mathcal{O} = \left\{ \begin{array}{l} (1)\ \mathit{BuildingFeature} \sqcap \exists \mathit{isLocatedInside}.\mathit{Private} \sqsubseteq \mathit{NoPublicAccess} \\ (2)\ \mathit{BusStop} \sqcap \mathit{Roofed} \sqsubseteq \mathit{CoveredBusStop} \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (9)\ \mathit{publicstation}(X) \leftarrow \mathrm{DL}[\mathit{BusStop} \uplus \mathit{busstop};\ \mathit{CoveredBusStop}](X); \\ \qquad\qquad\qquad\qquad \mathit{not}\ \mathrm{DL}[;\ \mathit{Private}](X); \\ (10)\ \bot \leftarrow \mathrm{DL}[\mathit{BuildingFeature} \uplus \mathit{publicstation};\ \mathit{NoPublicAccess}](X), \\ \qquad\qquad \mathit{publicstation}(X). \end{array} \right\}$$

the fact $\mathit{busstop}(\mathit{bs}_i)$ to $\mathcal{P}$ with probability $p/100$. Some instances are inconsistent since in our data set there are roofed bus stops, located inside private areas.

The results for both complete and incomplete support families are shown in Table 3.

Discussion of Results. As expected, using complete support families works for both settings well in practice. For the policy benchmark, allowing up to 2 support sets is more effective than bounding the size by 2. This is due to exploitation of partial completeness for the case when the number of support sets is limited. Moreover, there are just few support sets for each DL-atom in this scenario; however almost all support sets have size larger than 2. Thus many random guesses on potential repair candidates need to be done, which is witnessed by jumps in the runtime for $p = 40$ to $p = 80$. If both size and number of support sets are unlimited, the obtained results are practically the same.

A similar behavior is observed for the OpenStreetMap scenario. Even if the ontology is big, runtimes do not differ significantly from the Policy example. This is due to liberal safety [6], which effectively restricts the reasoning only to relevant individuals. We can again see that bounding the number of support sets works better; however, there are no jumps for bounded support set size. This is because a considerable number of support sets has size at most 2, and they guide the repair search effectively.

7 Related Work and Conclusion

We considered computing repair answer sets of DL-programs over $\mathcal{EL}$ ontologies, for which we generalized the support set approach [10; 8] for $DL\text{-}Lite_{\mathcal{A}}$ to work with incomplete families of supports sets; this advance is needed since in $\mathcal{EL}$ complete support families can be large or even infinite. We discussed how to generate support families, by exploiting query rewriting over ontologies to datalog [19; 23; 25], which is in contrast to [10; 8] where TBox classification is invoked. We presented an algorithm to compute deletion repair answer sets which trades answer completeness for scalability (a variant is complete); a declarative implementation shows very promising results.

As for related work, our DL-program repair is related to ABox cleaning [20; 24]. However, the latter differs in various aspects: it aims at restoring consistency of an *inconsistent* ontology by deleting $\subseteq$-minimal sets of assertions (i.e., computing $\subseteq$-maximal deletion repairs); we deal with inconsistency incurred on top of a consistent ontology, by arbitrary (non-monotonic) rules which access it with an interface. Furthermore, we must consider multiple ABoxes at once (via updates), and use $\mathcal{EL}$ instead of $DL\text{-}Lite$. Refining our algorithm to compute $\subseteq$-maximal deletion repairs is possible.

Our support sets are related to solutions of abduction problems for $\mathcal{EL}$ [2], and correspond in the ground case to support sets for query answering over first-order rewritable ontologies [3]; nonground computation naturally links to TBox classification [17]. Abduction had been studied for $DL\text{-}Lite$ in [5] and for datalog e.g. in [14; 16]. The use of incomplete support families for DL-atoms is related in spirit to approximate inconsistency-tolerant reasoning on DLs using restricted support sets [3]; however, we target repair computation while [3] targets inference from all repairs.

As for implementation, no comparable system exists. The DReW system [27] can evaluate DL-programs over $\mathcal{EL}$ ontologies after transforming the input to datalog, where DL-atoms are replaced by datalog rewritings; the latter amount to succinct representations of support sets. However, DReW can not handle inconsistencies and how to inject repairs efficiently is non-obvious (naive attempts fail).

It remains an issue for further research to identify classes of $\mathcal{EL}$ ontologies for which support sets have a benign structure and can be effectively computed, and on the other side to extend the work to other members of the $\mathcal{EL}$ family. To increase usability in practice, real world ontologies need to be analyzed to develop good heuristics and strategies for computing incomplete support families. Another possible research direction is computing specific types of repairs, e.g. by bounded deletion or addition [9].

References

1. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: Kaelbling, L.P., Saffiotti, A. (eds.) IJCAI, pp. 364–369. Prof. Book Center (2005)
2. Bienvenu, M.: Complexity of abduction in the $\mathcal{EL}$ family of lightweight description logics. In: KR Proc., pp. 220–230. AAAI Press (2008)
3. Bienvenu, M., Rosati, R.: New inconsistency-tolerant semantics for robust ontology-based data access. In: DL CEUR Workshop Proc., vol. 1014, pp. 53–64. CEUR-WS.org (2013)
4. Bonatti, P.A., Faella, M., Sauro, L.: $\mathcal{EL}$ with default attributes and overriding. In: ISC (1). Lecture Notes in CS, vol. 6496, pp. 64–79. Springer (2010)

5. Borgida, A., Calvanese, D., Rodriguez-Muro, M.: Explanation in $DL\text{-}Lite$. In: DL CEUR Workshop Proc., vol. 353. CEUR-WS.org (2008)
6. Eiter, T., Fink, M., Krennwallner, T., Redl, C.: Liberal safety for answer set programs with external sources. In: AAAI, pp. 267-275. AAAI Press (2013)
7. Eiter, T., Fink, M., Krennwallner, T., Redl, C., Schüller, P.: Efficient HEX-program evaluation based on unfounded sets. J. of Artif. Intell. Res. 49, 269–321 (2014)
8. Eiter, T., Fink, M., Redl, C., Stepanova, D.: Exploiting support sets for answer set programs with external computations. In: AAAI 2014 (to appear)
9. Eiter, T., Fink, M., Stepanova, D.: Data repair of inconsistent DL-programs. In: IJCAI, pp. 869-876. IJCAI/AAAI (2013)
10. Eiter, T., Fink, M., Stepanova, D.: Towards practical deletion repair of inconsistent DL-programs. In: ECAI 2014 (to appear)
11. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artif. Intell. 172(12-13), 1495–1539 (2008)
12. Eiter, T., Schneider, P., Simkus, M., Xiao, G.: Using *openstreetmap* data to create benchmarks for ontology-based query answering systems. In: ORE 2014 (to appear)
13. Gardiner, T., Tsarkov, D., Horrocks, I.: Framework for an automated comparison of description logic reasoners. In: ISWC Lecture Notes in Computer Science, vol. 4273, pp. 654–667. Springer (2006)
14. Gottlob, G., Pichler, R., Wei, F.: Efficient datalog abduction through bounded treewidth. In: AAAI, pp. 1626–1631. AAAI Press (2007)
15. Hansen, P., Lutz, C., Seylan, I., Wolter, F.: Query rewriting under $\mathcal{EL}$-TBoxes: efficient algorithms In: DL 2014 (to appear)
16. Hermann, M., Pichler, R.: Counting complexity of minimal cardinality and minimal weight abduction. In: JELIA. Lecture Notes in CS, vol. 5293, pp. 206–218. Springer (2008)
17. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK. J. of Autom. Reason. pp. 1–61 (2013).
18. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in $DL\text{-}Lite$. In: KR proc., pp. 247–257. AAAI Press (2010)
19. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in $\mathcal{EL}$ using a database system. In: OWLED. CEUR Workshop Proc., vol. 432. CEUR-WS.org (2008)
20. Masotti, G., Rosati, R., Ruzzi, M.: Practical abox cleaning in $DL\text{-}Lite$ (progress report). In: DL. CEUR Workshop Proc., vol. 745. CEUR-WS.org (2011)
21. Motik, B., Rosati, R.: Reconciling Description Logics and Rules. J. of the ACM 57(5), pp. 1–62 (2010)
22. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. of Applied Logic 8(2), 186–209 (2010).
23. Rosati, R.: On conjunctive query answering in $\mathcal{EL}$. In: DL CEUR Workshop Proc., vol. 250. CEUR-WS.org (2007)
24. Rosati, R., Ruzzi, M., Graziosi, M., Masotti, G.: Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies. In: ISWC. Lecture Notes in Computer Science, vol. 7650, pp. 337–349. Springer (2012)
25. Stefanoni, G., Motik, B., Horrocks, I.: Small datalog query rewritings for $\mathcal{EL}$. In: DL CEUR Workshop Proc., vol. 846. CEUR-WS.org (2012)
26. Stuckenschmidt, H., Parent, C., Spaccapietra, S. (eds.): Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization, Lecture Notes in Computer Science, vol. 5445. Springer (2009)
27. Xiao, G., Eiter, T., Heymans, S.: The DReW system for nonmonotonic DL-programs. In: CSWS2012 and CWSC2012, pp. 383–389. Springer, New York (2013).

28. Zhao, Y., Pan, J.Z., Ren, Y.: Implementing and evaluating a rule-based approach to querying regular $\mathcal{EL}+$ ontologies. In: HIS (3), pp. 493–498. IEEE Computer Society (2009)
29. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web, Artif. Intell. 172 (12-13), pp. 1495–1539 (2008)