Towards Practical Deletion Repair of Inconsistent DL-programs

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. Nonmonotonic Description Logic (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 but a straightforward computation method lacks practicality. We present a novel evaluation algorithm for deletion repair answer sets based on support sets, which reduces evaluation of $DL\text{-}Lite_{\mathcal{A}}$ ontology queries to constraint matching. This leads to significant performance gains towards inconsistency management in practice.

1 Introduction

The need for combining rules with ontologies has led to different approaches (see [12] and references therein). Among them Nonmonotonic Description Logic (DL-) programs [8] embody a loose coupling of rules and a DL-knowledge base (i.e., ontology) which the rules can query through special *DL-atoms*. As the ontology may be enriched before query evaluation, a bidirectional flow of information enables one to solve advanced reasoning tasks on top of ontologies. On the other hand, the information flow may lead to inconsistency, that is, cause a DL-program to lack answer sets (i.e., models).

Example 1. Consider the DL-program Π in Figure 1, which represents information about children of a primary school and their parents in simplistic form [7]. It has an ontology $\mathcal{O}$ consisting of a concept taxonomy (TBox) $\mathcal{T}$ in (1)-(3) and a set $\mathcal{A}$ of assertions (ABox), i.e. facts, about individuals in (4)-(6). The rules P contain further facts (7), (8) and proper rules: (9) determines fathers from the ontology, upon feeding information to it; (10) checks, informally, against them for local parent information ($ischildof$) that a child has surely not two fathers, unless it is adopted; finally (11)-(12) single out contact persons for children (by default, the parents); for adopted children, fathers in $\mathcal{O}$ are omitted if some other contact exists. Inconsistency arises as $john$, who is not provably adopted, has pat as father by the ontology, and by the local information possibly also $alex$.

An inconsistent program may be unusable since it yields no answer set. To cope with this, notions of DL-program repair (under different semantics) have been formalized recently [7]. They give rise to so-called repair answer sets under the supposition of

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

$$\mathcal{O} = \left\{ \begin{array}{lll} (1)\ \mathit{Child} \sqsubseteq \exists.\mathit{hasParent} & (2)\ \mathit{Adopted} \sqsubseteq \mathit{Child} & (3)\ \mathit{Female} \sqsubseteq \neg \mathit{Male} \\ (4)\ \mathit{Male}(\mathit{pat}) & (5)\ \mathit{Male}(\mathit{john}) & (6)\ \mathit{hasParent}(\mathit{john}, \mathit{pat}) \end{array} \right\}$$

$$P = \left\{ \begin{array}{l} (7)\ \mathit{ischildof}(\mathit{john}, \mathit{alex}); \quad (8)\ \mathit{boy}(\mathit{john}); \\ (9)\ \mathit{hasfather}(X, Y) \leftarrow \mathrm{DL}[\mathit{Male} \uplus \mathit{boy};\ \mathit{Male}](Y), \mathrm{DL}[;\ \mathit{hasParent}](X, Y); \\ (10)\ \bot \leftarrow \mathit{not}\ \mathrm{DL}[;\ \mathit{Adopted}](X), Y_1 \neq Y_2, \mathit{hasfather}(X, Y_1), \mathit{ischildof}(X, Y_2), \\ \qquad\qquad \mathit{not}\ \mathrm{DL}[\mathit{Child} \uplus \mathit{boy};\ \neg \mathit{Male}](Y_2); \\ (11)\ \mathit{contact}(X, Y) \leftarrow \mathrm{DL}[;\ \mathit{hasParent}](X, Y), \mathit{not}\ \mathit{omit}(X, Y); \\ (12)\ \mathit{omit}(X, Y) \leftarrow \mathrm{DL}[;\ \mathit{Adopted}](X), Z \neq Y, \mathit{hasfather}(X, Y), \mathit{contact}(X, Z). \end{array} \right\}$$

suitable changes to the ABox (data) of the ontology. As for repair computation, however, a natural realization of corresponding algorithms turns out to be too naive and does not scale for practical applications due to large number of ABoxes to be checked in general.

We thus propose an alternative approach for repair answer set computation which is based on the notion of support sets. Intuitively, a support set for a ground DL-atom $a = \mathrm{DL}[\lambda;\ Q](\boldsymbol{t})$ is a set of assertions which together with the original TBox is sufficient for the given DL-query $Q(\boldsymbol{t})$ to be derived.

Our basic method is to precompute small support sets for each DL-atom on a non-ground level. Then during the DL-program evaluation, for each candidate interpretation the ground instantiations of the support sets are effectively obtained. These help to prune the answer set search space and they are also used for the ABox repair construction.

This approach is particularly attractive for $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ ontologies, for which it scales much better then the naive approach [7], as the number of necessary support sets is small and they are easy to compute.

Our contributions on DL-program repair computation are:

(1) We formally establish support sets as sound and complete structures for DL-atom evaluation avoiding ontology access. Moreover, this characterization is faithfully lifted to the nonground level, enabling scalability of exploitation.

(2) Nonground support sets for DL-atoms over $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ ontologies can not only be efficiently computed, but also turn out to be small. We utilize this fact to devise an algorithm for the effective computation of deletion repairs of DL-programs under so-called *flp* semantics and discuss potential generalizations.

(3) We report experimental results obtained implementing our approach and evaluating its scalability on a set of benchmark scenarios; they provide evidence for the effectiveness of the method.

2 Preliminaries

We start with briefly recalling DL-programs and repair answer sets; see [8; 7] for details.

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

$\mathcal{O}$ is a *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 here throughout that $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ is a *consistent* $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ KB [3] with a *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:

$$C \rightarrow A \mid \exists R \quad B \rightarrow C \mid \neg C \quad R \rightarrow U \mid U^{-} \quad S \rightarrow R \mid \neg R$$

TBox axioms are of the form $C \sqsubseteq B$ and $R \sqsubseteq S$ (inclusion axiom), or $(\mathit{func}\ R)$ (functionality axiom). ABox contains positive and negative fact assertions.

When the distinction between concepts and roles is immaterial, P denotes a possibly negated predicate from $\mathbf{C} \cup \mathbf{R}$ and $\overline{P}$ is the opposite of P, i.e. $\overline{P} = \neg P$ and $\overline{\neg P} = P$.

The logic program part P of $\Pi = \langle \mathcal{O}, P \rangle$ consists of the 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 each a_i, $0 \leq i \leq n$, is an lp-atom and each b_i $1 \leq i \leq m$, 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 $\mathcal{P}$ of predicate names disjoint with $\mathbf{C}$ and $\mathbf{R}$, and constants from the set $\mathcal{C} = \mathbf{I}$;
- a *DL-atom* $a(\mathbf{t})$ is of form $\mathrm{DL}[\lambda;\, Q](\boldsymbol{t})$, where

$$\lambda = S_1\ op_1\, p_1, \ldots, S_m\ op_m\, p_m, \quad m \geq 0, \tag{2}$$

is such that for $1 \leq i \leq m$, $S_i \in \mathbf{C} \cup \mathbf{R}$, $op_i \in \{\uplus, \uplus\}$ is an *update operator*, and $p_i \in \mathcal{P}$ is an *input predicate* of the same arity as S_i—intuitively, $op_i = \uplus$ (resp., $op_i = \uplus$) 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). We skip $(\boldsymbol{t})$ for $\boldsymbol{t} = \epsilon$[1] .

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

Example 2 (cont'd). A DL-atom $\mathrm{DL}[\mathit{Male} \uplus \mathit{boy};\ \mathit{Male}](Y)$ contained in the rule (9) of Example 1 first enriches the concept Male in $\mathcal{O}$ by the extension of the predicate boy in P via $\uplus$, and then queries the concept Male over the modified ontology.

Semantics. The semantics of a DL-program $\Pi = \langle \mathcal{O}, P \rangle$ is defined in terms of its grounding $gr(\Pi) = \langle \mathcal{O}, gr(P) \rangle$ over $\mathcal{C}$, i.e., $gr(P)$ contains all ground instances of rules r in 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 $\mathcal{P}$; 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

- $A_i(I) = \{S_i(\boldsymbol{t}) \mid p_i(\boldsymbol{t}) \in I\}$, for $op_i = \uplus$;
- $A_i(I) = \{\neg S_i(\boldsymbol{t}) \mid p_i(\boldsymbol{t}) \in I\}$, for $op_i = \uplus$.

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

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

Example 3 (cont'd). In Example 1, $I = \{ischildof(john,\ alex),\ boy(john)\}$ satisfies $a = \mathrm{DL}[Child \uplus boy;\ \neg Male](alex)$, as $\mathcal{O} \cup \lambda^I(a) \models \neg Male(alex)$, while $I \not\models^{\mathcal{O}} \mathrm{DL}[;\ Adopted](john)$, as the input list is empty and $\mathcal{O} \not\models Adopted(john)$.

An *(flp-)answer set* of $\Pi = \langle \mathcal{T} \cup \mathcal{O}, \mathcal{A} \rangle$ is any interpretation I that is a $\subseteq$-minimal model of the *flp*-reduct Π^I_{FLP}, which maps any set P of rules and $I \subseteq HB_\Pi$ to the rule set $P^I_{FLP} = \{r^I_{FLP} \mid r \in gr(P)\}$, 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 $\Pi = \langle \mathcal{T} \cup \mathcal{O}, \mathcal{A} \rangle$, if it is an *flp*-answer set of some $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', P \rangle$ where $\mathcal{A}' \subseteq \mathcal{A}$; any such $\mathcal{A}'$ is called a *deletion repair* of Π.

Example 4 (cont'd). As mentioned, Π is inconsistent; if we drop (4) from $\mathcal{A}$, then $I = \{boy(john), ischildof(john, alex),\ contact(john, pat)\}$ is an answer set. Along with the facts (7) and (8) the *flp*-reduct P^I_{FLP} contains the ground rule (11), where X and Y are substituted by $john$ and pat respectively. The interpretation $I_1 = \{boy(john),\ ischildof(john, alex)\}$ is a deletion repair answer set with a deletion repair $\mathcal{A}'_1 = \{Male(pat), Male(john)\}$.

In $DL\text{-}Lite_{\mathcal{A}}$ ontologies, inconsistency arises by few assertions.

Proposition 1 (cf. [3]). *In $DL\text{-}Lite_{\mathcal{A}}$ given a TBox $\mathcal{T}$ every $\subseteq$-minimal ABox $\mathcal{A}$ such that $\mathcal{T} \cup \mathcal{A}$ is inconsistent fulfills $|\mathcal{A}| \leq 2$.*

From this result it is not hard to establish that $\leq$3 distinct constants occur in such an $\mathcal{A}$.

3 Support Sets

In this section, we introduce support sets for DL-atoms. Intuitively, a *support set* for a DL-atom $d = \mathrm{DL}[\lambda;\ Q](\boldsymbol{t})$ is a portion of its input that, together with ABox assertions, is sufficient to conclude that the query $Q(\boldsymbol{t})$ will evaluate to true, i.e, that given a subset $I' \subseteq I$ of an interpretation I and a set $\mathcal{A}' \subseteq \mathcal{A}$ of ABox assertions from the ontology, we can conclude that $I \models^{\mathcal{O}} Q(\boldsymbol{t})$. The evaluation of d w.r.t. I reduces then to test whether some support set $S = I' \cup \mathcal{A}'$ exists; to this end, a sufficient collection of such sets S can be precomputed and stored. Fortunately, for $DL\text{-}Lite_{\mathcal{A}}$ this is efficiently possible.

To simplify matters and avoid dealing with I' separately, it is convenient to introduce *input assertions* as follows.

Definition 1. *Given a DL-atom $d = \mathrm{DL}[\lambda;\ Q](\boldsymbol{t})$, we call $P_p(c)$ an* input assertion for *d, if $P \circ p \in \lambda$, $\circ \in \{\uplus, \sqcup\}$ and $c \in \mathcal{C}$, where P_p is a fresh ontology predicate; $\mathcal{A}_d$ is the set of all such assertions.*

For a TBox $\mathcal{T}$ and DL-atom d, let then $\mathcal{T}_d = \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\} \cup \{P_p \sqsubseteq \neg P \mid P \sqcup p \in \lambda\}$, and for an interpretation I, let $\mathcal{O}^I_d = \mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\boldsymbol{t}) \in \mathcal{A}_d \mid p(\boldsymbol{t}) \in I\}$. Then we have:

Proposition 2. *For every $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$, DL-atom $d = \mathrm{DL}[\lambda;\, Q](\boldsymbol{t})$ and interpretation I, $I \models^{\mathcal{O}} d$ iff $I \models^{\mathcal{O}^I_d} \mathrm{DL}[\epsilon;\, Q](\boldsymbol{t})$ iff $\mathcal{O}^I_d \models Q(\boldsymbol{t})$.*

Unlike (3), in $\mathcal{O}^I_d$ there is a clear distinction between native assertions and assertions induced by d w.r.t. I (via facts P_p and axioms $P_p \sqsubseteq (\neg)P$), mirroring the lp-input of d.

Now, in view of the property that in $DL\text{-}Lite_{\mathcal{A}}$ a single assertion is sufficient to derive a query [3] from a consistent ontology, we obtain support sets as follows.

Definition 2 (Ground Support Sets). *Given a ground DL-atom $d = DL[\lambda; Q](\boldsymbol{t})$, a set S of assertions from $\mathcal{A} \cup \mathcal{A}_d$ is a* support set *for d w.r.t. an ontology $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$, if either (i) $S = \{P(\boldsymbol{c})\}$ and $\mathcal{T}_d \cup S \models Q(\boldsymbol{t})$, or (ii) $S = \{P(\boldsymbol{c}), P'(\boldsymbol{d})\}$ such that $\mathcal{T}_d \cup S$ is inconsistent, where $\boldsymbol{c} \cup \boldsymbol{d}$ has at most three distinct constants; by $Supp_{\mathcal{O}}(d)$ we denote the set of all such S.*

Support sets are linked to interpretations by the following notion.

Definition 3. *A support set S of a DL-atom d is* coherent with an interpretation I*, if for each $P_p(\boldsymbol{c}) \in S$ it holds that $p(c) \in I$.*

Example 5 (cont'd). The set $\{hasParent(john, pat)\}$ is a support set for the DL-atom $\mathrm{DL}[;\, hasParent](john, pat)$ w.r.t. $\mathcal{O}$, and so is $\{Male(pat)\}$ for the DL-atom $a = \mathrm{DL}[Male \uplus boy;\, Male](pat)$. Moreover, $\{P_{boy}(pat)\}$ is in $Supp_{\mathcal{O}}(a)$ but incoherent with minimal models of Π.

The evaluation of d w.r.t. I then reduces to the search for coherent support sets.

Proposition 3. *Let d be a ground DL-atom, let $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ be an ontology, and let I be an interpretation. Then, $I \models^{\mathcal{O}} d$ iff some $S \in Supp_{\mathcal{O}}(d)$ exists s.t. S is coherent with I.*

As a simple consequence, we get:

Corollary 1. *Given a ground DL-atom d and an ontology $\mathcal{O}$, there exists an interpretation I such that $I \models^{\mathcal{O}} d$ iff $Supp_{\mathcal{O}}(d) \neq \emptyset$.*

3.1 Nonground Support Sets

Using support sets, we can completely eliminate the ontology access for the evaluation of DL-atoms. In a naive approach, one precomputes all support sets for all ground DL-atoms with respect to relevant ABoxes, and then uses them during the repair answer set computation. This does not scale in practice, since support sets may be computed that are incoherent with all candidate repair answer sets.

An alternative is to fully interleave the support set computation with the search for repair answer sets. Here we construct coherent ground support sets for each DL-atom and interpretation on the fly. As the input to a DL-atom may change in different interpretations, its support sets must be recomputed, however, since reuse may not be possible; effective optimizations are not immediate.

A better solution is to precompute support sets on a nonground level, that is, schematic support sets, prior to repair computation. Furthermore, in that we may leave the concrete ABox open; the support sets for a DL-atom instance are then easily obtained by syntactic matching. This leads to the following definition.

Definition 4 (Support Set for Nonground DL-atom). *Let $d(\boldsymbol{X}) = \mathrm{DL}[\lambda; Q](\boldsymbol{X})$ be a DL-atom and $\mathcal{T}$ be a TBox. Let $V = \{X, Y, Z\}$ be distinct variables such that $\boldsymbol{X} \subseteq V$ and let $\mathcal{C} = \{a, b, c\}$. A* nonground support set *for d w.r.t. $\mathcal{T}$ is a set $S = \{P(\boldsymbol{Y})\}$ resp. $S = \{P(\boldsymbol{Y}), P'(\boldsymbol{Y'})\}$ such that (i) $\boldsymbol{Y}, \boldsymbol{Y}' \subseteq V$ and (ii) for each substitution $\theta : V \to \mathcal{C}$, the instance $S\theta = \{P(\boldsymbol{Y}\theta)\}$ (resp. $S\theta = \{P(\boldsymbol{Y}\theta), P'(\boldsymbol{Y}'\theta)\}$) is a support set for $d(\boldsymbol{X}\theta)$ w.r.t. $\mathcal{O}_\mathcal{C} = \mathcal{T} \cup \mathcal{A}_\mathcal{C}$, where $\mathcal{A}_\mathcal{C}$ is the set of all possible assertions over $\mathcal{C}$.*

Here $\mathcal{A}_\mathcal{C}$ takes care for any possible ABox, by considering the maximal ABox (as $\mathcal{O} \subseteq \mathcal{O}'$ implies $Supp_\mathcal{O}(d) \subseteq Supp_{\mathcal{O}'}(d)$); three variables suffice as at most three different constants are involved.

Example 6 (cont'd). Certainly $\{hasParent(X, Y)\}$ is a nonground support set for the DL-atom $\mathrm{DL}[; hasParent](X, Y)$, and so are $\{Male(X)\}$ and $\{Male_{boy}(X)\}$ for the DL-atom $a(X) = \mathrm{DL}[Male \uplus boy; Male](X)$. But also, $\{Male_{boy}(X), Female(X)\}$ is a nonground support set for $a(X)$.

Nonground support sets S are sound, i.e. each instance $S\theta$ that matches with $\mathcal{A} \cup \mathcal{A}_d$ is a support set of the ground DL-atom $d\theta$ w.r.t. $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$; they are also complete, i.e., every support set S of a ground DL-atom d w.r.t. $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ results as such an instance, and thus can be determined by syntactic matching. Clearly, support sets as defined above may be subsumed by other support sets (e.g., $\{A(X), R(X, Y)\}$ by $\{A(X)\}$ and removed; for space reasons, we omit further discussion here. In the next section, we discuss how to compute a sufficient set of nonground support sets.

3.2 Determining Nonground Support Sets

Our technique for computing the nonground support sets is based on TBox classification, which is one of the main ontology reasoning tasks. This reasoning service computes complete information about the TBox constraints specified at the conceptual level.

More formally, given a TBox $\mathcal{T}$ over a signature Σ_o, the TBox classification $Clf(\mathcal{T})$ determines all subsumption relations $P \sqsubseteq (\neg)P'$ between concepts and roles P, P' in Σ_o that are entailed by $\mathcal{T}$. This can be exploited for our goal to compute nonground support sets, more precisely a complete family $\mathbf{S}$ of such sets. Here *completeness* of $\mathbf{S}$ for a (non-ground) DL-atom $d(\boldsymbol{X})$ w.r.t. $\mathcal{O}$ means that for every θ: $\boldsymbol{X} \to \mathbf{I}$ and $S \in Supp_\mathcal{O}(d(\boldsymbol{X}\theta))$, some $S' \in \mathbf{S}$ exists s.t. $S = S'\theta'$ for some extension θ' of θ to V.

TBox classification is well studied in Description Logics [1]. For example, [9] discusses it for $\mathcal{EL}$ w.r.t. concept hierarchy, and [10] studies it for the OWL 2 QL profile, which is based on $DL\text{-}Lite_\mathcal{R}$.

Respective algorithms are thus suitable and also easily adapted for the computation of (a complete family of) nonground support sets for a DL-atom $d(\boldsymbol{X})$ w.r.t. $\mathcal{O}$. In principle, one can exploit Proposition 2 and resort to $\mathcal{T}_d$, i.e., compute the classification $Clf(\mathcal{T}_d)$, and determine nonground support sets of $d(\boldsymbol{X})$ proceeding similar as for computing minimal conflict sets [10]. To determine inconsistent support sets, perfect rewriting [3] can be done over $Pos(\mathcal{T})$, i.e., the TBox obtained from $\mathcal{T}$ by substituting all negated concepts (roles) $\neg C$ ($\neg R$, $\neg\exists.R$, $\neg\exists.R^-$) with positive replacements $\overline{C}$ ($\overline{R}$, $\overline{\exists.R}$, $\overline{\exists.R^-}$).

In practice (and as in our implementation), it nonetheless can be worthwhile to compute $Clf(\mathcal{T})$ first, as it is reusable for all DL-atoms. The additional axioms in $\mathcal{T}_d$,

Algorithm 1: $SupRAnsSet$: all deletion repair answer sets

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

(a) compute a complete set $\mathbf{S}$ of nongr. supp. sets for the DL-atoms in Π
(b) **for** $\hat{I} \in AS(\hat{\Pi})$ **do**
  $D_p \leftarrow \{a \mid e_a \in \hat{I}\}$; $D_n \in \{a \mid ne_a \in \hat{I}\}$; $\mathbf{S}^{\hat{I}}_{gr} \leftarrow Gr(\mathbf{S}, \hat{I}, \mathcal{A})$;
(c) **if** $\mathbf{S}^{\hat{I}}_{gr}(a) \neq \emptyset$ *for* $a \in D_p$ *and every* $S \in \mathbf{S}^{\hat{I}}_{gr}(a)$ *for* $a \in D_n$ *fulfills* $S \cap \mathcal{A} \neq \emptyset$ **then**
(d) **for** *all* $a \in D_p$ **do**
(e) **if** *some* $S \in \mathbf{S}^{\hat{I}}_{gr}(a)$ *exists s.t.* $S \cap \mathcal{A} = \emptyset$ **then** pick next a
  else remove each S from $\mathbf{S}^{\hat{I}}_{gr}(a)$ s.t. $S \cap \mathcal{A} \cap \bigcup_{a' \in D_n} \mathbf{S}^{\hat{I}}_{gr}(a') \neq \emptyset$
(f) **if** $\mathbf{S}^{\hat{I}}_{gr}(a) = \emptyset$ **then** pick next $\hat{I}$
  end
(g) $\mathcal{A}' \leftarrow \mathcal{A} \setminus \bigcup_{a' \in D_n} \mathbf{S}^{\hat{I}}_{gr}(a')$;
(h) **if** $flpFND(\hat{I}, \langle \mathcal{T} \cup \mathcal{A}', P \rangle)$ **then** output $\hat{I}|_{\Pi}$
  end
end

i.e., those of form $P_p \sqsubseteq (\neg)P$ (according to the update operators), are handled when determining the nonground support sets for a particular DL-atom from $Clf(\mathcal{T})$.

Example 7 (cont'd). Consider the DL-atom $a = \mathrm{DL}[Male \uplus boy;\ Male](X)$ in our running example. For computing a complete family $\mathbf{S}$ of nonground support sets for a w.r.t. $\mathcal{O}$, we may refer to $\mathcal{T}_a = \mathcal{T} \cup \{Male_{boy} \sqsubseteq Male\}$ and its classification $Clf(\mathcal{T}_a)$. Hence, $S_1 = \{Male(X)\}$ and $S_2 = \{Male_{boy}(X)\}$ are the only unary nonground support sets of a. Further nonground support sets are obtained computing minimal conflict sets, yielding $\{P(\boldsymbol{X}), \neg P(\boldsymbol{X})\}$ for each $P \in \mathbf{C} \cup \mathbf{R}$, as well as $\{Male_{boy}(X), \neg Male(X)\}$, $\{Male(X), Female(X)\}$, and $\{Male_{boy}(X), Female(X)\}$. However, since we are interested in completeness w.r.t. $\mathcal{O}$ and $\mathcal{O}$ is consistent, pairs not involving input assertions can be dropped (as they will not have a match in $\mathcal{A}$). The remaining two sets are supersets of S_2, therefore $\mathbf{S} = \{S_1, S_2\}$ is a complete family of support sets for a w.r.t. $\mathcal{O}$.

4 Repair Answer Set Computation

We are now ready to describe our algorithm for computing deletion repair answer sets with the use of support sets. DL-programs are evaluated usually (as in DLVHEX) by means of 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 $e_a v ne_a$, where ne_a stands for negation of e_a. We denote interpretations of $\hat{\Pi}$ by $\hat{I}$, and use $\hat{I}|_{\Pi}$ to denote their restriction to the original language of Π.

A basic algorithm for computing repair answer sets was given in [7]. It checks whether candidates $\hat{I}$ (i.e., answer sets of $\hat{\Pi}$) yield repair answer sets by solving a corresponding ontology repair problem (ORP). Intuitively, solutions to an ORP correspond to (potentially) modified ABoxes $\mathcal{A}'$ (not necessarily obtained by deletion, i.e. subsets of

$\mathcal{A}$) such that all DL-atoms evaluate as guessed in $\hat{I}$ (formally $\hat{I}$ is a compatible set of $\langle \mathcal{T} \cup \mathcal{A}', P \rangle$, cf. [7]); they are computed through multiple calls to the external ontology via a query interface to evaluate DL-atoms under varying ABoxes. A final foundedness check (subset minimality of $\hat{I}|_{\Pi}$ w.r.t. $\Pi'^{\hat{I}|_{\Pi}}_{FLP} = \langle \mathcal{T} \cup \mathcal{A}', P^{\hat{I}|_{\Pi}}_{FLP} \rangle$) establishes repair answer sets.

Our new algorithm $SupRAnsSet$ (see Algorithm 1), avoids instead multiple interface calls and merely needs to access the ontology once. Given a (ground) DL-program Π for input, $SupRAnsSet$ proceeds as follows. We start (a) by computing a complete family $\mathbf{S}$ of nonground support sets for each DL-atom. Afterwards the replacement program $\hat{\Pi}$ is created and its answer sets are computed one by one. Once an answer set $\hat{I}$ of $\hat{\Pi}$ is found (b), we first determine the sets of DL-atoms D_p (resp. D_n) that are guessed true (resp. false) in $\hat{I}$. Next, for all ground DL-atoms in $D_p \cup D_n$, the function $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$ instantiates $\mathbf{S}$ to relevant ground support sets, i.e., that are coherent with $\hat{I}$ and match with $\mathcal{A} \cup \mathcal{A}_a$. We then check in (c) for atoms in D_p (resp. D_n) without support (resp. input only support). If either is the case, we skip to (b), the next model candidate, since no repair exists for the current one. Otherwise, in a loop (d) over atoms in D_p—except for those supported input only (e)—we remove support sets S that are conflicting w.r.t. D_n. Intuitively, this is the case if S hinges on an assertion $\alpha \in \mathcal{A}$ that also supports some atom $a' \in D_n$ (hence α needs to be deleted; note that due to consistency of $\mathcal{A}$, even inconsistent support of a' leaves no choice). If this operation leaves the atom from D_p under consideration without support (check at (f)), then no repair exists and the next model candidate is considered. Otherwise (exiting the loop at (g)), a potential deletion repair $\mathcal{A}'$ is obtained from $\mathcal{A}$ by removing assertions that occur in any support set for some atom $a' \in D_n$. An eventual check (h) for foundedness (minimality) w.r.t. $\mathcal{A}'$ determines whether a deletion repair answer set has been found.

Example 8 (cont'd). Suppose $\{e_a, ne_b\} \subseteq \hat{I}$ for $a = \mathrm{DL}[; hasParent](john, pat)$ and $b = \mathrm{DL}[Male \uplus boy; Male](pat)$. Then, $\mathbf{S}^{\hat{I}}_{gr}(a) = \{\{hasParent(john, pat)\}\}$ we get to the else part of Step (e) where nothing is removed from $\mathbf{S}^{\hat{I}}_{gr}(a)$, since $\mathbf{S}^{\hat{I}}_{gr}(b) = \{\{Male(pat)\}\}$ and $\mathbf{S}^{\hat{I}}_{gr}(a) \cap \mathbf{S}^{\hat{I}}_{gr}(b) = \emptyset$. Hence, at Step (g) we must drop $Male(pat)$ from $\mathcal{A}$ to make $\hat{I}$ a deletion repair answer set.

As can be shown, algorithm $SupRAnsSet$ correctly computes the deletion repair answer sets of the input DL-program. For the completeness part, i.e., that all deletion repair answer sets are indeed produced, the following proposition is crucial.

Proposition 4. *Given a DL-program Π, let $\hat{I}$ be an answer set of $\hat{\Pi}$ such that $I = \hat{I}|_{\Pi}$ is an answer set of $\Pi = \langle \mathcal{T} \cup \mathcal{A}, P \rangle$. If $\hat{I}$ is a compatible set for $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', P \rangle$ where $\mathcal{A}' \supseteq \mathcal{A}$, then I is an answer set for $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', P \rangle$.*

Armed with this result, we establish the correctness result.

Theorem 1 *$SupRAnsSet$ is sound and complete w.r.t. deletion repair answer sets.*

5 Implementation and Experiments

We implemented Algorithm $SupRAnsSet$ within the DLVHEX evaluation framework[2], which allows us to effectively compute deletion repair answer sets for $DL\text{-}Lite_{\mathcal{A}}$. In order to profit from existing DLVHEX data structures (e.g. for parsing) and optimization methods (such as nogood learning, etc.), we pursued a declarative ASP approach to realize $SupRAnsSet$. This applies to (a) computing complete nonground support families and to (b) searching candidate deletion repair answer sets and deletion repairs.

As for (a), we use a (stratified) logic program that intuitively mimics perfect rewriting to compute classifications relevant for support set computation on top. The logic program reifies concepts (roles, etc.), as well as positive replacements. Facts express subsumptions in $Pos(\mathcal{T})$, their contra-positives, and the duality of concepts (roles, etc.) and positive replacements of their negation. A simple rule transitively closes the subsumption relation; support sets are naturally expressed (and thus computed), using unary and binary predicates to represent respective support for a DL-atom, the query and relevant concepts (roles) as from the input signature.

Concerning (b), non-ground rules (see below) are added to $\hat{\Pi}$ for the purpose of filtering candidate deletion repair answer sets as done by $SupRAnsSet$. Therefore, the language of $\hat{\Pi}$ is extended to include support set information. For any nonground support set S that covers a ground DL-atom a, the additional rules are of the form:

$$\bar{S}_{\mathcal{A}} \leftarrow r(S), ne_a; \quad sup_a \leftarrow r(S), e_a, not\ \bar{S}_{\mathcal{A}}; \quad \leftarrow e_a, not\ sup_a;$$

where $r(S)$ is a suitable representation of S, i.e., using predicates $p(\boldsymbol{X})$ for input assertions $P_p(\boldsymbol{X})$, resp. $p_P(\boldsymbol{X})$ ($np_P(\boldsymbol{X})$) for ABox assertions $P(\boldsymbol{X})$ ($\neg P(\boldsymbol{X})$). Furthermore, p_P, np_P, and sup_a are fresh predicates not in $\mathcal{P}$, and $\bar{S}_{\mathcal{A}}$ refers to $np_P(\boldsymbol{X})$ (resp. $p_P(\boldsymbol{X})$) if $S \cap \mathcal{A} \neq \emptyset$ and the assertion is positive (resp. negative), otherwise it is void. With further constraints $\leftarrow p_P(\boldsymbol{X}), np_P(\boldsymbol{X})$, the resulting program intuitively prunes candidates $\hat{I}$, resp. encodes deletion repair candidates, according to $SupRAnsSet$.

Experimental Setup. We have evaluated the approach by comparing it with ordinary answer set computation on two scenarios. They were run on a Linux server with two 12-core AMD 6176 SE CPUs/128GB RAM using DLVHEX 2.3.0; a timeout of 120 secs was set for each run.

Family Benchmark. The first benchmark is derived from our running example. We fixed two ABoxes with $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$, of different size, where $\mathcal{A}_{50}$ contains 50 children (7 adopted), 20 female and 32 male adults; and twenty times that many for $\mathcal{A}_{1000}$. Every child has at most two parents of different sex and the number of children per parent varies from 1 to 3. Rules (11) and (12), not involved in conflicts, have been dropped from P. Instances are varied in terms of facts over $\mathbf{I}$ included in P by increasing the probability $p/100$ (p ranges from 5 to 35): for every child c, one additional fact $isChildOf(c, p)$ is added with probability $p/100$ for a random male adult non-parent; as well $boy(c)$ is included with probability $p/100$.

Network Benchmark. In a second scenario a network is described by a fixed ontology $\mathcal{O}$ using a relation $edge$. Some nodes might be broken or blocked. There are overall 70

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

nodes, (among them 23 broken and rest available). There are 68 edges (among them 5 forbidden). The TBox encodes that if an edge is forbidden, then its endpoint must be blocked, and if a node is known to be broken, then it is automatically blocked, moreover blocked nodes are not available:

$$\mathcal{O} = \{\exists.forbid \sqsubseteq Block, \quad Broken \sqsubseteq Block, \quad Block \sqsubseteq \neg Avail\}.$$

We consider two DL-programs, P_{guess} and P_{conn}, over $\mathcal{O}$ that are randomly generated as follows. P_{guess} contains for any node n the fact $node(n)$ with probability $p/100$, and the following rules:

(1) $go(X,Y) \leftarrow open(X), open(Y), \mathrm{DL}[; edge](X,Y).$
(2) $route(X,Z) \leftarrow route(X,Y), route(Y,Z).$
(3) $route(X,Y) \leftarrow go(X,Y), not\ \mathrm{DL}[Block \uplus block; forbid](X,Y).$
(4) $open(X) \vee block(Y) \leftarrow node(X), not\ \mathrm{DL}[; \neg Avail](X).$
(5) $negIs(X) \leftarrow node(X), route(X,Y), X \neq Y.$
(6) $\bot \leftarrow node(X), not\ negIs(X).$

Intuitively, (1), (2) and (3) recursively determine routes over non-blocked ($open$) nodes; where (3) expresses that by default a route is recommended unless it is known to be forbidden. Rule (4) amounts to guessing for each node not known to be unavailable, whether it is blocked or not, i.e. it contains nondeterminism, which makes rule processing challenging. Rules (5) and (6) encode the requirement that no input node is isolated w.r.t. the resulting routes.

The program P_{conn} contains the same random node facts as P_{guess} and facts $in(n)$ and $out(n)$, either for each node n with equal probability (i.e., in with $p = 1/2$, out, otherwise). It contains the rules (1) and (2) above, and variants of (3), (4) and (6):

(3′) $route(X,Y) \leftarrow go(X,Y), not\ \mathrm{DL}[; forbid](X,Y).$
(4′) $open(X) \leftarrow node(X), not\ \mathrm{DL}[; \neg Avail](X).$
(6′) $\bot \leftarrow in(X), out(Y), not\ route(X,Y).$

Intuitively, rather than guessing, (4′) expresses that each node is open by default unless known to be unavailable, and by (6′) the program checks whether each in-node is connected to all out-nodes (thus ensuring they are available, i.e. not blocked).

Results. The results that we obtained for these settings are given in Table 1. Instances are grouped by the probability (the value p) used for generating them. For each p, 25 instances were generated and their average running time (in seconds) to compute a first answer set and deletion repair answer set is reported. The naive approach [7] has not been implemented (but expectedly would time out for even the smallest problem instances).

Despite a small overhead for computation of deletion repair answer sets compared to ordinary ones the results for the family problem with $\mathcal{A}_{50}$ scale well. For $\mathcal{A}_{1000}$ the first repair answer set is found quicker then the inconsistency is identified during ordinary answer set computation. Liberal safety [5] exploited in the experiments improves greatly the running time both for repair and ordinary answer set computation modes.

In the Network domain, as expected, running times grow exponentially with the instance size for P_{guess} due to the guessing rule (4). Employing a default in P_{conn} rather than guessing scales linearly, however. For P_{conn} we also considered different splits (rather than $p = 1/2$) between in and out, although without significant change.

p	$\mathcal{A}_{50}$		$\mathcal{A}_{1000}$		p	P_{conn}		P_{guess}	
	AS	rep	AS	rep		AS	rep	AS	rep
5 (25)	0.12 (0)	0.19 (0)	63.93 (0)	36.70 (0)	10 (25)	0.19 (0)	0.13 (0)	0.48 (0)	0.16 (0)
10 (25)	0.12 (0)	0.19 (0)	63.77 (0)	37.59 (0)	20 (25)	0.19 (0)	0.44 (0)	0.48 (0)	0.37 (0)
15 (25)	0.12 (0)	0.20 (0)	63.98 (0)	38.29 (0)	30 (25)	0.19 (0)	1.28 (0)	0.51 (0)	0.86 (0)
20 (25)	0.12 (0)	0.20 (0)	63.90 (0)	39.17 (0)	40 (25)	0.19 (0)	2.36 (0)	0.53 (0)	1.31 (0)
25 (25)	0.12 (0)	0.20 (0)	63.82 (0)	39.97 (0)	50 (25)	0.19 (0)	5.60 (0)	0.57 (0)	3.00 (0)
30 (25)	0.12 (0)	0.21 (0)	63.97 (0)	40.77 (0)	60 (25)	0.19 (0)	8.83 (0)	0.60 (0)	5.03 (0)
35 (25)	0.12 (0)	0.21 (0)	63.18 (0)	41.41 (0)	70 (25)	0.19 (0)	15.92 (0)	0.64 (0)	7.27 (0)
					80 (25)	0.20 (0)	25.89 (0)	0.69 (0)	11.45 (0)
time in seconds for first ordinary and deletion repair answer set					90 (25)	0.20 (0)	37.33 (0)	0.75 (0)	16.95 (0)
					100 (25)	0.20 (0)	55.43 (0)	0.87 (0)	16.38 (0)

Table 1. Family and Network benchmark results

Additionally, our approach has been evaluated on inconsistent programs over the LUBM[3] ontology; more precisely, over a $DL\text{-}Lite_{\mathcal{A}}$ version together with a fixed, generated[4] ABox (1 university with over 500 students, 50 courses and 29 teaching assistants). Rules encode default reasoning under constraints: teaching assistants (TAs) are normally students, and TAs must not take the exam of a course they teach. Facts $takesExam(c_1, c_2)$ have been randomly added to the program with a probability between 0.05 to 0.2. A first repair answer set is found for all instances within 4 seconds, while detecting inconsistency with ordinary answer set computation takes about 1 minute.

These results indicate the effectiveness of our approach for practical settings.

6 Conclusion

Support sets reduce the evaluation of DL-atoms over $DL\text{-}Lite_{\mathcal{A}}$ ontologies to constraint matching, providing a base for effective deletion repair answer set computation under *flp*-semantics. Our results, and in particular algorithm $SupRAnsSet$, easily extend to other semantics, e.g., weak repair answer sets [7], and to other notions of repairs. Furthermore, support sets are used in a companion work to compute answer sets of HEX-programs [6]. As external atoms lack an explicit data part (ABox), support sets ought to be more abstract, which makes direct efficient usage less clear; repair is an open issue.

Note that algorithm $SupRAnsSet$ constructs in its search all maximal deletion repairs, i.e., $\subseteq$-maximal repairs $\mathcal{A}' \subseteq \mathcal{A}$ that admit an answer set (however, it also may construct non-maximal repairs). In this regard it is similar to work on ABox cleaning [11; 13]. There, an inconsistent ontology (with consistent TBox) is repaired by identifying and eliminating minimal conflict sets causing, i.e., explaining, the inconsistency, thus resulting in maximal deletion repairs. However, our setting and work differ fundamentally: (i) the ontology is consistent and inconsistency arises only through the interface of DL-atoms, and (ii) several DL-atom queries have to be considered (entailment or non-entailment) under different potential ABox updates.

[3] http://swat.cse.lehigh.edu/projects/lubm/

[4] http://code.google.com/p/combo-obda/

More remotely related to our work are approaches for explaining positive and negative answers to conjunctive queries in $DL\text{-}Lite$ [4; 2], as they apply the perfect reformulation algorithm [3] and then extract explanations in a nontrivial way.

In ongoing work, we consider restricted repairs [7] and an extension of our approach to other DLs like $\mathcal{EL}$. While no small complete support families exist for $\mathcal{EL}$ in general, this might still apply for practically relevant fragments. Furthermore incomplete support families can be used for optimization (caching) to reduce access to an $\mathcal{EL}$ reasoner.

References

1. Baader, F., Calvanese, D., McGuinness, D.L., Nardi, D., Patel-Schneider, P.F.: The Description Logic Handbook: Theory, Implementation, and Applications. Cambridge University Press, New York, NY, 2nd edn. (2007)
2. Borgida, A., Calvanese, D., Rodriguez-Muro, M.: Explanation in $DL\text{-}Lite$. In: Baader, F., Lutz, C., Motik, B. (eds.) Description Logics. CEUR Workshop Proc., vol. 353. CEUR-WS.org (2008)
3. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The $DL\text{-}Lite$ family. J. Autom. Reasoning 39(3), 385–429 (2007)
4. Calvanese, D., Ortiz, M., Simkus, M., Stefanoni, G.: The complexity of explaining negative query answers in $DL\text{-}Lite$. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) Principles of Knowledge Representation and Reasoning: Proc. of 13th International Conference, KR 2012, Rome, Italy, June 10-14, 2012. pp. 583–587. AAAI Press (2012)
5. Eiter, T., Fink, M., Krennwallner, T., Redl, C.: Liberal safety for answer set programs with external sources. In: desJardins, M., Littman, M.L. (eds.) Proc. of 27th Conf. on Artificial Intelligence (AAAI '13), pp. 267–275. AAAI Press (2013)
6. Eiter, T., Fink, M., Redl, C., Stepanova, D.: Exploiting support sets for answer set programs with external evaluations. In: Brodley, C., Stone, P. (eds.) Proc. 28th Conf. on Artificial Intelligence (AAAI '14), AAAI Press (2014), To appear.
7. Eiter, T., Fink, M., Stepanova, D.: Data repair of inconsistent dl-programs. In: Rossi, F. (ed.) Proc. 23rd International Joint Conf. on Artificial Intelligence (IJCAI-13), pp. 869–876. AAAI Press/IJCAI (2013)
8. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the semantic web. Artificial Intelligence 172(12-13), 1495–1539 (2008)
9. Kazakov, Y., Krötzsch, M., Simancik, F.: The incredible ELK - from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies. J. Autom. Reasoning 53(1), 1–61 (2014)
10. Lembo, D., Santarelli, V., Savo, D.F.: A graph-based approach for classifying OWL 2 QL ontologies. In: Eiter, T., Glimm, B., Kazakov, Y., Krötzsch, M. (eds.) Description Logics. CEUR Workshop Proc., vol. 1014, pp. 747–759. CEUR-WS.org (2013)
11. Masotti, G., Rosati, R., Ruzzi, M.: Practical abox cleaning in $DL\text{-}Lite$ (progress report). In: Rosati, R., Rudolph, S., Zakharyaschev, M. (eds.) Description Logics. CEUR Workshop Proce., vol. 745. CEUR-WS.org (2011)
12. Motik, B., Rosati, R.: Reconciling Description Logics and Rules. Journal of the ACM 57(5), 1–62 (June 2010)
13. Rosati, R., Ruzzi, M., Graziosi, M., Masotti, G.: Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies. In: Cudré-Mauroux, et al. (eds.) International Semantic Web Conference (2). Lecture Notes in Computer Science, vol. 7650, pp. 337–349. Springer (2012)