Towards Practical Deletion Repair of Inconsistent DL-programs

Thomas Eiter and Michael Fink and Daria Stepanova

Abstract. Nonmonotonic Description Logic (DL-) programs couple nonmonotonic 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-Lite_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, a DL-program to lack answer sets (i.e., models).

Example 1. Consider the DL-program II 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) \( T \) in (1)-(3) and a set \( \mathcal{A} \) of assertions (ABox), i.e., facts, about individuals in (4)-(6). The rules \( \mathcal{P} \) 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 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 since the number of ABoxes to be checked might be large 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 \( a = DL[A]; Q(t) \) is a set of assertions which together with the original TBox is sufficient for the given DL-query \( Q(t) \) to be derived.

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

This approach is particularly attractive for DL-Lite_A ontologies, for which it scales much better than 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 DL-Lite_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.

1 Supported by the Austrian Science Fund (FWF) project P24090.

2 Institute of Information Systems, Vienna University of Technology, Austria. email: {eiter,fink,dasha}@kr.tuwien.ac.at
2 PRELIMINARIES

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

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

- \( \mathcal{O} \) is a DL-knowledge base (or ontology) over a signature \( \Sigma = (\mathcal{C}, \mathcal{R}) \) with a set \( \mathcal{I} \) of individuals, a set \( \mathcal{C} \) of concept names and a set \( \mathcal{R} \) of role names. We assume here throughout that \( \mathcal{O} = T \cup A \) is a consistent DL-Lite\(_A\) KB [3] with a TBox \( T \) and ABox \( A \), which are sets of axioms capturing taxonomic resp. factual knowledge. Concepts \( C \) and roles \( R \) obey the following syntax, where \( A \in \mathcal{C} \) is an atomic concept and \( U \in \mathcal{R} \) is an atomic role:

\[
\begin{align*}
C & \rightarrow A \mid \exists R \quad \forall R \rightarrow U \mid U^- \\
B & \rightarrow C \mid \neg C \quad S \rightarrow R \mid \neg R
\end{align*}
\]

TBox axioms are of the form \( C \subseteq B \) and \( R \subseteq S \) (inclusion axiom), or \((func \ R)\) (functionality axiom).

An (ABox) assertion is a formula \( A(c) \) or \( U(c, d) \) where \( A \in \mathcal{C}, \ U \in \mathcal{R}, \ c, d \in I \) (called positive) or its negation, i.e., \( \neg A(c) \) resp. \( \neg (U(c, d)) \).

Whenever the distinction between concepts and roles is immaterial, \( P \) denotes a possibly negated predicate from \( \mathcal{C} \cup \mathcal{R} \). We denote by \( P^\top \) the opposite of \( P \), i.e., \( P^\top = \neg P \) and \( \neg P^\top = P \).

- \( \mathcal{P} \) consists of rules of the form

\[
a_1 \lor \ldots \lor a_n \leftarrow b_1, \ldots, b_k, \lnot b_{k+1}, \ldots, \lnot b_m, \tag{1}
\]

where each \( a_i, 0 \leq i \leq n, \) is anlp-atom and each \( b_j 1 \leq j \leq m, \) is either anlp-atom or a DL-atom; here

- an lp-atom is a first-order atom \( p(t) \) with predicate \( p \) from a set \( \mathcal{P} \) of predicate names disjoint with \( \mathcal{C} \) and \( \mathcal{R} \), and constants from the set \( I \) is

- a DL-atom \( a(t) \) is of form \( DL[\lambda; Q](t) \), where \( \lambda \)

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

is such that, for \( 1 \leq i \leq m, S_i \in \mathcal{C} \cup \mathcal{R}, \ op_i \in \{\lnot, \lor, \land\} \) is an update operator, and \( p_i \in \mathcal{P} \) is an input predicate of the same arity as \( S_i \)—intuitively, \( op_1 = \lor \) (resp., \( op_m = \lnot \)) increases \( S_i \) (resp., \( \neg S_i \)) by the extension of \( p_i \); (b) \( Q(t) \) is a DL-query, which is either of the form \( (c)(t) \), \( C \) 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) \( \neg Q(t) \) where \( Q(t) \) is from (i)-(ii). We skip \( (t) \) for \( t = \varepsilon \).

If \( n = 0 \), the rule is a constraint.

Example 2 (cont’d). In Example 1, the rule (9) contains a DL-atom \( DL[\lambda; bo\{Y\}] \), which first enriches the concept \( Male \) in \( O \) by the extension of the predicate \( bo\{Y\} \) in \( P \) via \( \lor \), and then queries the concept \( Male \) over the modified ontology.

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

A (Herbrand) interpretation of \( \Pi \) is a set \( I \subseteq HB_{\Pi} \) of ground atoms, where \( HB_{\Pi} \) is the usual Herbrand base w.r.t. \( \mathcal{C} \) and \( \mathcal{P} \); \( I \) satisfies anlp-atom \( a, \) if \( a \in I \) and a DL-atom \( a \) of the form \( (2) \) if \( \mathcal{O} \cup \lambda(a) \models Q(c) \).

\[
I \models \lambda(a) \tag{3}
\]

We disregarded here for simplicity the less used constraints-operator \( \oplus \) and subsumption queries.

3 SUPPORT SETS

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

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

Definition 6. Given a DL-atom \( d = DL[\lambda; Q](t) \), we call \( \Pi_p(c) \) an input assertion for \( d \), if \( p \in \lambda, \) and \( c \in \{\lnot, \lor, \land\} \) where \( p \) is a fresh ontology predicate; \( \mathcal{A}_d \) is the set of all such assertions.

For a TBox \( T \) and DL-atom \( d \), let then \( T_d = T \cup \mathcal{A}_d \) and \( P \cup p \in \lambda \} \cup \{P \cup \neg p \in \lambda \}, \) and for an interpretation \( I \), let \( O_d \models T_d \cup A \cup \{P(t) \in A \cup \{p(t) \in I \} \}. \) Then we have:

Proposition 7. For every \( O = T \cup A \), DL-atom \( d = DL[\lambda; Q](t) \) and interpretation \( I \models \lambda \) iff \( I \models O_d \models Q(t \models O_d \models Q(t). \)

\[\text{Note that additionally considering program facts can be emulated.}\]
Unlike (3), in \( O \) there is a clear distinction between native assertions and input assertions of \( d \) w.r.t. \( I \) (via facts \( P_r \) and axioms \( P_r \subseteq \neg P_t \)), mirroring the Ip-input of \( d \).

Now, in view of the property that in DL-Lite, a single assertion is sufficient to derive a query [3] from a consistent ontology, we obtain support sets as follows.

**Definition 8 (Ground Support Sets).** Given a ground DL-atom \( d = DL[I; Q](t) \), a set \( S \) of assertions from \( A \cup A_c \) is a support set for \( d \) w.r.t. an ontology \( Q \) if \( T \cup A \), if either

1. \( S = \{ P(e) \} \) and \( T_d \cup S \models Q(t) \), or
2. \( S = \{ P(e), P'(d) \} \) such that \( T_d \cup S \) is inconsistent, by \( \text{Supp}_Q(d) \) we denote the set of all such \( S \).

Support sets are linked to interpretations by the following notion.

**Definition 9.** A support set \( S \) of a DL-atom \( d \) is coherent with an interpretation \( I \), if for each \( P_y(e) \in S \) it holds that \( p(c) \in I \).

**Example 10 (cont’d).** The set \( \{ \text{hasParent}(john, pat) \} \) is a support set for the DL-atom \( DL[I; \text{hasParent}(john, pat)] \) w.r.t. \( O \), and so is \( \{ \text{Male}(pat) \} \) for the DL-atom \( a = DL[\text{Male} \equiv \text{boy}; \text{Male}(pat)] \). Moreover, \( \{ \text{Male}(pat) \} \) is in \( \text{Supp}_Q(a) \) but inconsistent with minimal models of \( \Pi \).

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

**Proposition 11.** Let \( d \) be a ground DL-atom, let \( O = T \cup A \) be an ontology, and let \( I \) be an interpretation. Then, \( I \models d \) iff some \( S \in \text{Supp}_Q(d) \) exists such that \( S \) is coherent with \( I \).

**Proof (Sketch).** \( \Rightarrow \) Suppose \( d = DL[I; Q](t) \) evaluates w.r.t. \( O \) and \( I \) to true, i.e., \( \lambda(d) \cup O \models Q(t) \). Towards a contradiction, assume no \( S \in \text{Supp}_Q(d) \) is coherent with \( I \). There are two cases:

1. \( \lambda(d) \cup O \) is consistent. Proposition 5 implies that an assertion \( \alpha \in \lambda(d) \cup A \) must exist s.t. \( T \cup \{ \alpha \} \models Q(t) \). If \( \alpha \in A \) then \( \text{Supp}_Q(d) \) contains \( \{ \alpha \} \) by (i) of Definition 8, which trivially is coherent with \( I \) and thus contradicts the assumption.
2. \( \lambda(d) \cup O \) is inconsistent. From Proposition 5 and consistency of \( O \), it follows that some \( \delta \in \lambda(d) \) exists such that either (a) \( T \cup \{ \delta \} \) is inconsistent, or (b) some \( \gamma \in A \cup \lambda(d) \) exists s.t. \( T \cup \{ \delta, \gamma \} \) is inconsistent. In case (a), we obtain \( \{ \delta \} \in \text{Supp}_Q(d) \), for the corresponding input assertion \( \delta \in A_c \). By (i) of Definition 8, this is a contradiction, as \( \{ \delta \} \) is coherent with \( I \). In case (b), we similarly conclude that either \( \{ \delta, \gamma \} \in \text{Supp}_Q(d) \) or \( \{ \delta, \gamma \} \in \text{Supp}_Q(d) \), depending on whether \( \gamma \in \lambda(d) \), according to (ii) of Definition 8.

Again this is a support set coherent with \( I \), contradiction.

\( \Leftarrow \) Suppose some \( S \in \text{Supp}_Q(d) \) is coherent with \( I \). Assume towards a contradiction that \( I \not\models d \).

Again we consider two cases:

1. \( T_d \cup S \) is consistent. Then, \( T_d \cup S \models Q(t) \) by item (i) of Definition 8. Since \( S \) is coherent with \( I \), we conclude that \( O_d' \models Q(t) \) which implies \( I \models d \) by Proposition 7. Contradiction.
2. \( T_d \cup S \) is inconsistent. Then, due to coherence with \( I \), so is \( O_d' \), and trivially \( O_d' \models Q(t) \); again we arrive at a contradiction by concluding that \( I \models d \) from Proposition 7.

As a simple consequence, we get:

**Corollary 12.** Given a ground DL-atom \( d \) and an ontology \( O \), there exists an interpretation \( I \) such that \( I \models d \) iff \( \text{Supp}_Q(d) \not= \emptyset \).

### 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 inconsistent 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 13 (Support Set for Nonground DL-Atom).** Let \( d(X) = DL[I; Q](X) \) be a DL-atom and \( T \) be a TBox. Let \( V = \{ X, Y, Z \} \) be distinct variables such that \( X \subseteq V \) and let \( C = \{ a, b, c \} \).

A nonground support set for \( d \) w.r.t. \( T \) is a set \( S = \{ P(Y) \} \) resp. \( S = \{ P(Y), P'(Y') \} \) such that (i) \( Y, Y' \subseteq V \) and (ii) for each substitution \( \theta : V \rightarrow C \), the instance \( S\theta \models \{ P(Y) \} \) (resp. \( S\theta \models \{ P(Y), P'(Y') \} \)) is a support set for \( d(X) \) w.r.t. \( O_c = T \cup A_c \), where \( A_c \) is the set of all possible assertions over \( C \).

Here \( A_c \) takes care of any possible ABox, by considering the maximal ABox (as \( O \subseteq O' \) implies \( \text{Supp}_{O'}(d) \subseteq \text{Supp}_O(d) \); three variables suffice as at most three different constants are involved.

**Example 14 (cont’d).** Certainly \( \{ \text{hasParent}(X, Y) \} \) is a nonground support set for the DL-atom \( DL[I; \text{hasParent}](X, Y) \), and so are \( \{ \text{Male}(X) \} \) and \( \{ \text{Male}(X) \} \) for the DL-atom \( a(X) = DL[\text{Male} \equiv \text{boy}; \text{Male}(X)] \). Moreover, \( \{ \text{Male}(X), \text{Female}(X) \} \) is a nonground support set for \( a(X) \).

Nonground support sets \( S \) are sound, i.e. each instance \( S\theta \) that matches with \( A \cup A_c \) is a support set of the ground DL-atom \( d \) w.r.t. \( O = T \cup A \); they are also complete, i.e., every support set \( S \) of a ground DL-atom \( d \) w.r.t. \( O = T \cup A \) 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 \( T \) over a signature \( \Sigma \), the TBox classification \( \text{Cl}(T) \) determines all subsumption relations \( P \subseteq \neg P' \) between concepts and roles \( P, P' \) in \( \Sigma \) that are entailed by \( T \). This can be exploited for our goal to compute nonground support sets, more precisely a complete family \( S \) of such sets. Here completeness of \( S \) for a (non-ground) DL-atom \( d(X) \) w.r.t. \( O \) means that for every
θ: X → I and S ∈ Supp(_d(Xθ)), some S′ ∈ S exists such that S = S′θ′, for some extension θ′ of θ to V.

TBox classification is well studied in Description Logics [1]. For example, [9] discusses it for EŁ w.r.t. concept hierarchy, and [10] studies it for the OWL 2 DL profile, which is based on DL-LiteR.

Respective algorithms are thus suitable and also easily adapted for the computation of a complete family of noground support sets for a DL-alternative repair answer set. In principle, one can exploit Proposition 7 and resort to TO, i.e., compute the classification Clf(T), and determine noground support sets of d(X) proceeding similarly as for computing minimal conflict sets [10]. To determine inconsistent support sets, perfect rewriting [3] can be done over Pos(T), i.e., the TBox obtained from T by substituting all negated concepts (roles) ¬C (¬R, ¬∃R, ¬∀R) with positive replacements C (R, ∃R, ∀R).

(Practice and in our implementation, it nonetheless can be worthwhile to compute Clf(T) first, as it is reusable for all DL-atoms.)

The additional axioms in TΩ, i.e., those of form Pn ⊆ ¬(¬P) (according to the update operators), are handled when determining the noground support sets for a particular DL-alternative from Clf(T).

Let us now illustrate the computation of noground support sets.

Algorithm 1: SupRASet: all deletion repair answer sets

Input: Π = (T ∪ A, P)
Output: flP[RAS](Π)

(a) compute a complete set S of noground support sets for the DL-atoms in Π
(b) for I ∈ AS(Π) do
   Dp ← {a ∈ I ∪ A| eI,a ∈ I}; Dn ∈ {a | neI,a ∈ I}; Sflp,fln ← Gr(S, I, A);
   (c) if Sflp,fln(a) ̸= ∅ for all a ∈ Dp and every S ∈ Sfln(a) for a ∈ Dn fulfills S ∩ A ̸= ∅ then
   (d) if some S ∈ Sflp,fln(a) exists s.t. S ∩ A = ∅ then pick next a
   (e) else remove each S from Sflp,fln(a) s.t. S ∩ A = ∅ and that S fln(a) ̸= ∅,
   (f) if Sflp,fln(a) ̸= ∅ then pick next I
   (g) if flpFND(I, ⟨T ∪ A, P⟩) then output I
   end
end

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 Π of gr(Π), where DL-atoms are replaced by ordinary atoms eI,a (replacement atoms), together with a guess on their truth by additional ‘choice’ rules eI,a ∨ neI,a, where neI,a stands for negation of eI,a. We denote interpretations of Π by I, and use I|Π to denote their restriction to the original language of Π.

A basic algorithm for computing repairing answer sets was given in [7]. It checks whether candidates I (i.e., answer sets of Π) yield repair answer sets by solving a corresponding ontology repair problem (ORP). Intuitively, solutions to an ORP correspond to (potentially) modified ABoxes A′ (not necessarily obtained by deletion, i.e., subsets of A) such that all DL-atoms evaluate as guessed in I (formally I is a compatible set of (T ∪ A′, P), cf. [7]); they are computed through multiple calls to the external interface via a query interface to evaluate DL-atoms under varying ABoxes. A final foundedness check (subset minimality of IΠ w.r.t. fΠIΠ = (T ∪ A′, fΠIΠ)) establishes repair answer sets.

Our new algorithm SupRASet (see Algorithm 1), avoids instead multiple interface calls and merely needs to access the ontology once. Given a (ground) DL-program Π for input, SupRASet proceeds as follows. We start (a) by computing a complete family S of noground support sets for each DL-atom. Afterwards the replacement program Π is created and its answer sets are computed one by one. Once an answer set I of Π is found (b), we first determine the sets of DL-atoms Dp (resp. Dn) that are guessed true (resp. false) in I. Next, for all ground DL-atoms in Dp ∪ Dn, the function Gr(S(Π, I, A) instantiates S to relevant ground support sets, i.e., that are coherent with I and match with A ∪ A′. We then check in (c) for atoms in Dp (resp. Dn) 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 Dp (and that are supported input only) we remove support sets S that are conflicting w.r.t. Dp. Intuitively, this is the case if S hinges on an assertion α ∈ A that also supports some atom a′ ∈ Dn (hence α needs to be deleted; note that due to consistency of A, even inconsistent support of a′ leaves no choice). If this operation leaves the atom from Dp under consideration without support (check at (f)), then no repair exists and the next model candidate is considered. Otherwise (exit the loop at (g)), a potential deletion repair A′ is obtained from A by removing assertions that occur in any support set for some atom a′ ∈ Dn. An eventual check (b) for foundedness (minimality) w.r.t. A′ determines whether a deletion repair answer set has been found (output at (i), or not).

Example 16 (cont’d). Suppose a = DL [hasParent](john, pat), b = DL [Male ⊔ boy] [Male](X) and {eI,a, neI,a} ⊆ I. Then, Sflp(a) = {{hasParent(john, pat)}} and we get to the else part of Step (e) where nothing is removed from Sfln(a), since Sfln(b) = {{Male(pat)}} and Sfln(a) ∩ Sfln(b) = ∅. Hence, at Step (g) we must drop Male(pat) from A to make I a deletion repair answer set.

As can be shown, algorithm SupRASet 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 17. Given a DL-program Π, let I be an answer set of Π such that I = IΠΠ is an answer set of Π = (T ∪ A, P). If I is a compatible set for Π′ = (T ∪ A′, P′) where A′ ⊆ A, then I is an answer set of Π′ = (T ∪ A′, P′).

Proof (Sketch). Assume that I is an answer set of Π = (⟨O, P⟩), where O = T ∪ A and that I is a compatible set for Π′ = (⟨O′, P′⟩) where O′ = T ∪ A′ and A′ ⊆ A. Towards contradiction, suppose I is not an answer set of Π. Hence, I = IΠΠ is not a minimal model of fΠΠ = (T ∪ A′, fΠΠ). That is, some I′ ⊆ I exists such that I′ |= O′ fΠΠ. Then we obtain that also I′ |= O fΠΠ; this contradicts that I is an answer set of Π. Indeed, suppose that I′ |= O fΠΠ. Then
some rule $r \in T^{P}$ of form (1) is violated wrt. $I'$ and $\Omega$, i.e., (i) $I' \models \Omega \, b_i$ for each $1 \leq i \leq k$, (ii) $I' \not\models \hat{\Omega} \, b_j$ for each $k < j \leq m$, and (iii) $I' \not\models \hat{\Omega} \, a_h$ for each $1 \leq h \leq n$. By monotonicity of $I \models \hat{\Omega} \, a \, r$ w.r.t. $I$ and $\Omega$, we conclude $I' \models \hat{\Omega} \, b_i, I' \models \hat{\Omega} \, b_j$ (as $I'$ is a compatible set for both $\hat{I}$ and $\hat{I}'$), and $I' \not\models \hat{\Omega} \, a_h$. But then $I' \not\models \Omega \, f^{P}$, which is a contradiction. Hence, $I'$ does not exist and $I$ is an answer set of $\hat{I}$.

Armed with this result, we establish the correctness result.

**Theorem 18.** Algorithm SupRAnsSet is sound and complete w.r.t. deletion repair answer sets.

**Proof (Sketch).** Soundness. Suppose SupRAnsSet outputs $I = \hat{I}[[1]]$. We can get to (i) only if $I$ is an answer set of $\hat{I}$; furthermore, by setting $S'_{P\hat{I}}$ to $Gr(S, \hat{I}, A)$ in (b) and by the further modifications, it is ensured at (i) that each DL-atom $a \in D_n$ has some coherent support set that matches with $A'$ i.e., $Gr(S, \hat{I}, A')(a) \neq \emptyset$, while no DL-atom $a' \in D_n$ has such a support set. Thus from Proposition 11, it follows that $I$ is a compatible set for $\hat{\Pi} = (T \cup A', P)$; hence $I \models \hat{\Pi}$. Furthermore, as $flipND(I, T \cup A', P)$ succeeds, $I$ is a minimal model of $\hat{f}^{P}$. Hence $I$ is an answer set of $\hat{I}$, and thus a deletion repair answer set of $\hat{I}$.

**Completeness.** Suppose $I$ is a deletion repair answer set. That is, for some $A' \subseteq A$, we have that $I$ is an answer set of $\hat{\Pi} = (T \cup A', P)$. This implies that $I$ is an answer set of $\Pi$ and thus will be considered in (b), with $D_{\Pi}$ and $D_{\hat{\Pi}}$ reflecting the (correct) guess for $I \models \Omega \, a$ for each DL-atom $a$, where $O' = T \cup A'$. From Proposition 11 and completeness of $S$, we obtain that each $a \in D_n$ has $Gr(S, I, A')(a) \neq \emptyset$ and each $a \in D_n$ has $Gr(S, I, A)(a) = \emptyset$. The initial $S'_{P\hat{I}}$ is such that $Gr(S, I, A')(a) \subseteq S'_{P\hat{I}} = Gr(S, I, A)(a)$ holds for each DL-atom $a$; in further steps, the algorithm removes all support sets $S \in Gr(S, I, A)(a)$ for $a \in D_n$ from $S'_{P\hat{I}}$ (such that $S \cap S' \cap A \neq \emptyset$ for some support set $S' \in Gr(S, I, A)(a')$ and $a' \in D_n$, and removes all assertions in $S' \cap A$ from $A$. Importantly no removed $S$ is in $Gr(S, I, A')(a)$, since by the assertion that $T \cup A'$ is consistent, $|S' \cap A| = 1$ must hold. Thus step (g) will be reached, and the variable $A'$ is assigned an ABox $A'$ such that $A' \subseteq A' \subseteq A$. Since $I$ is a compatible set for $\hat{\Pi}' = (T \cup A', P)$ and $I$ is an answer set of $\hat{\Pi}'$, by Proposition 11 $I'$ is also an answer set of $\hat{\Pi}'$, and thus $I'$ is a minimal model of $\hat{\Pi}' = (T \cup A', f^{P})$. Hence, the test $flipND(I, T \cup A', P)$ in step (h) (where $A'$ has value $A'$) succeeds, and $I_{\hat{\Pi}}$, i.e. $I$ is output.

5 IMPLEMENTATION AND EXPERIMENTS

We implemented the Algorithm SupRAnsSet within the DLVHEX evaluation framework, which allows us to effectively compute deletion repair answer sets for DL-LiteA. 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 $Pos(T)$, their contra-positives, and the duality of concepts (roles, etc.) and positive replacements of their negation. A simple rule transitivity 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 $\hat{I}$ for the purpose of filtering candidate deletion repair answer sets as done by SupRAnsSet. Therefore, the language of $\hat{I}$ 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:

$$S_A \leftarrow r(S), a_e \leftarrow sup_a(r(S), a_e), not S_A \leftarrow e, not sup_a$$

where $r(S)$ is a suitable representation of $S$, i.e., using predicates $p(X)$ for input assertions $p(X)$, resp. $p_r(X)$ (resp. $p_r(X)$) for ABox assertions $P(X)$ (resp. $P(X)$). Furthermore, $pr$, $npp$, and $sup_a$ are fresh predicates not in $P$, and $S_A$ refers to $npp_r(X)$ (resp. $p_r(X)$) if $S \cap A \neq \emptyset$ and the assertion is positive (resp. negative), otherwise it is void. With further constraints $pr_r(X)$, the resulting program intuitively prunes candidates $I$, 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 $A_{50}$ and $A_{1000}$, of different size, where $A_{50}$ contains 50 children (7 adopted), 20 female and 32 male adults; and twenty times that many for $A_{1000}$. Every child has at most two parents of different sex and the number of children per parent varies from 1 to 3. Rules (1) and (12), not involved in conflicts, have been dropped from $P$. Instances are varied in terms of facts over $I$ included in $P$ by increasing the probability $p/100$ ($p$ ranges from 5 to 35); for every child $c$, one additional fact $\text{isChildOf}(c, p)$ is added with probability $p/100$ for a random male adult non-parent; as well $\text{boy}(c)$ is included with probability $p/100$.

**Network Benchmark.** In the second scenario a network is described by a fixed ontology $O$ using a relation edge. Some nodes might be broken or blocked. There are overall 70 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: $O = \{3\text{forbid} \subseteq \text{Block}, \text{Broken} \subseteq \text{Block}, \text{Block} \subseteq -\text{Avail}\}$.

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

1. $go(X, Y) ← open(X), open(Y), DL[\text{edge}]X, Y$.
2. $route(X, Z) ← route(X, Y), route(Y, Z)$.
3. $route(X, Y) ← go(X, Y), not DL[\text{Block} \cup \text{block}](X, Y)$.
4. $open(X) \lor \text{block}(Y) ← \text{node}(X), not DL[\text{~Avail}](X)$.
5. $\neg\text{block}(X) ← \text{node}(X), route(X, Y), X \neq Y$.

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 $P_{\text{cons}}$ contains the same random node facts as $P_{\text{guess}}$ and additional 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) ← got(X,Y), not DL: forbid(X,Y).
(4') open(X) ← node(X), not DL: ¬Available[X].
(6') ⊥ ← 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).

<table>
<thead>
<tr>
<th>p</th>
<th>AN</th>
<th>rep</th>
<th>AN</th>
<th>rep</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 (25)</td>
<td>0.12 (0)</td>
<td>0.19 (0)</td>
<td>63.95 (0)</td>
<td>36.40 (0)</td>
</tr>
<tr>
<td>10 (25)</td>
<td>0.12 (0)</td>
<td>0.19 (0)</td>
<td>63.77 (0)</td>
<td>37.59 (0)</td>
</tr>
<tr>
<td>15 (25)</td>
<td>0.12 (0)</td>
<td>0.20 (0)</td>
<td>63.98 (0)</td>
<td>38.29 (0)</td>
</tr>
<tr>
<td>20 (25)</td>
<td>0.12 (0)</td>
<td>0.20 (0)</td>
<td>63.90 (0)</td>
<td>39.17 (0)</td>
</tr>
<tr>
<td>25 (25)</td>
<td>0.12 (0)</td>
<td>0.20 (0)</td>
<td>63.82 (0)</td>
<td>39.97 (0)</td>
</tr>
<tr>
<td>30 (25)</td>
<td>0.12 (0)</td>
<td>0.21 (0)</td>
<td>63.97 (0)</td>
<td>40.77 (0)</td>
</tr>
<tr>
<td>35 (25)</td>
<td>0.12 (0)</td>
<td>0.21 (0)</td>
<td>63.18 (0)</td>
<td>41.41 (0)</td>
</tr>
</tbody>
</table>

Table 1. Family benchmark results

<table>
<thead>
<tr>
<th>p</th>
<th>Pconn</th>
<th>Pguess</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 (25)</td>
<td>0.19 (0)</td>
<td>0.13 (0)</td>
</tr>
<tr>
<td>20 (25)</td>
<td>0.19 (0)</td>
<td>0.44 (0)</td>
</tr>
<tr>
<td>30 (25)</td>
<td>0.19 (0)</td>
<td>1.28 (0)</td>
</tr>
<tr>
<td>40 (25)</td>
<td>0.19 (0)</td>
<td>2.36 (0)</td>
</tr>
<tr>
<td>50 (25)</td>
<td>0.19 (0)</td>
<td>5.60 (0)</td>
</tr>
<tr>
<td>60 (25)</td>
<td>0.19 (0)</td>
<td>8.83 (0)</td>
</tr>
<tr>
<td>70 (25)</td>
<td>0.19 (0)</td>
<td>15.92 (0)</td>
</tr>
<tr>
<td>80 (25)</td>
<td>0.20 (0)</td>
<td>25.89 (0)</td>
</tr>
<tr>
<td>90 (25)</td>
<td>0.20 (0)</td>
<td>37.33 (0)</td>
</tr>
<tr>
<td>100 (25)</td>
<td>0.20 (0)</td>
<td>55.43 (0)</td>
</tr>
</tbody>
</table>

Table 2. Network benchmark results

Results. The results that we obtained for these settings are given in Table 1 and Table 2. 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 (using DLVHEX) 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 A50 scale well. For A1000 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 Pguess due to the guessing rule (4). Employing a default in Pconn, rather than guessing scales linearly however. For Pconn we also considered different splits (rather than p = 1/2) between in and out, although without significant change. These results indicate the effectiveness of our approach for practical settings.

6 CONCLUSION

Support sets reduce the evaluation of DL-atoms over DL-LiteA 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 matching in its search all maximal deletion repairs, i.e., C-maximal repairs A′ ⊆ A that admit an answer set (however, it may also 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.

More remotely related to our work are approaches for explaining positive and negative answers to conjunctive queries in DL-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 DL. While no small complete support families exist for DL 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 DL reasoner.

REFERENCES