Artificial Intelligence 239 (2016) 7–53

Contents lists available at ScienceDirect

Artificial Intelligence

www.elsevier.com/locate/artint

Data repair of inconsistent nonmonotonic description logic programs ☆

Thomas Eiter, Michael Fink, Daria Stepanova *

Institute of Information Systems, Vienna University of Technology, Vienna, Austria

ARTICLE INFO

Article history:
Received 14 June 2015
Received in revised form 10 June 2016
Accepted 12 June 2016
Available online 15 June 2016

Keywords:
Rules and ontologies
Answer set programming
Description logics
Inconsistency management
Nonmonotonic reasoning

ABSTRACT

Combining Description Logic (DL) ontologies and nonmonotonic rules has gained increasing attention in the past decade, due to the growing range of applications of DLs. A well-known proposal for such a combination are non-monotonic DL-programs, which support rule-based reasoning on top of DL ontologies in a loose coupling, using a well-defined query interface. However, inconsistency may easily arise as a result of the interaction of the rules and the ontology, such that no answer set (i.e., model) of a DL-program exists; this makes the program useless. To overcome this problem, we present a framework for repairing inconsistencies in DL-programs by exchanging formulas of an ontology formulated in *DL-Lite*$_{\mathcal{A}}$, which is a prominent DL that allows for tractable reasoning. Viewing the data part of the ontology as a source of inconsistency, we define program repairs and repair answer sets based on them. We analyze the complexity of the notion, and we extend an algorithm for evaluating DL-programs to compute repair answer sets, under optional selection of preferred repairs that satisfy additional constraints. The algorithm induces a generalized ontology repair problem, in which the entailment respectively non-entailment of queries to the ontology, subject to possible updates, must be achieved by a data change. While this problem is intractable in general, we identify several tractable classes of preferred repairs that are useful in practice. For the class of deletion repairs among them, we optimize the algorithm by reducing query evaluation to constraint matching, based on the novel concept of support set, which roughly speaking is a portion of the data from which entailment of an ontology query follows. Our repair approach is implemented within an answer set program system, using a declarative method for repair computation. An experimental evaluation on a suite of benchmark problems shows the effectiveness of our approach and promising results, both regarding performance and quality of the obtained repairs. While we concentrate on *DL-Lite*$_{\mathcal{A}}$ ontologies, our notions extend to other DLs, for which more general computation approaches may be used.

1. Introduction

Description Logics (DLs) [4], which emerged from semantic networks with the goal to equip respective formalisms with a clear formal semantics based on logic, nowadays play a dominant role among formalisms for Knowledge Representation

☆ Some of the results were presented in preliminary form at IJCAI 2013 [33] and ECAI 2014 [35]. This work has been supported by the Austrian Science Fund (FWF) project P24090.

* Corresponding author.
E-mail addresses: eiter@kr.tuwien.ac.at (T. Eiter), fink@kr.tuwien.ac.at (M. Fink), dasha@kr.tuwien.ac.at (D. Stepanova).

http://dx.doi.org/10.1016/j.artint.2016.06.003
0004-3702/

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

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

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

and Reasoning (KRR). As such, DLs are geared towards describing domains in terms of concepts that map to sets of domain objects and their relations, as well as roles that capture relationships among domain objects. This makes DLs well-suited for representing ontologies formally and to reason about them, which has a central role in the Semantic Web vision [9]; indeed, DLs provide the formal underpinning of the Web Ontology Language (OWL), a recommended standard for expressing ontological knowledge on the web. Fueled by the success in this area, DLs have been successfully deployed to many other contexts and applications, among them reasoning about actions [6], data integration and ontology based data access [20,19], spatial reasoning [69], runtime verification and program analysis [2,53], and many others.

Most DL ontologies are fragments of classical first-order logic, and as such lack sufficient expressiveness for the requirements of certain problems; for instance, they cannot model closed-world reasoning, nor can they express nonmonotonicity; these features are often essential in practical application scenarios. Furthermore, DLs do not offer rules, which are popular in practical knowledge representation and serve a complementary aspect: while DLs are focused on specifying and reasoning about conceptual knowledge, logic rules serve well for reasoning about individuals; furthermore they target issues associated with nonmonotonic inference as well as non-determinism. To overcome these shortcomings, several extensions of DLs have been developed, e.g. [80,5,26,27,65,15,23,52,47,14] and various notions of *hybrid knowledge bases* (KBs) have been proposed to get the best out of the DL and rules worlds by combining them (see [66] and references therein). Among them, *Nonmonotonic Description Logic (DL-)programs* [37] are the most prominent approach for a loose coupling between the rules and the ontology via so-called DL-atoms, which serve as query interfaces to the ontology that support information hiding and the use of legacy software (i.e., ontology reasoners). The possibility to add information from the rules part prior to query evaluation allows for adaptive combinations.

Example 1. Consider the DL-program Π in Fig. 1, which captures information about children of a primary school and their parents in simplistic form. It is given as a pair $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ of an ontology $\mathcal{O}$ and a set of rules $\mathcal{P}$. The ontology $\mathcal{O}$ contains a taxonomy $\mathcal{T}$ of concepts (i.e., classes) in (1)–(3) and factual data (i.e., assertions) $\mathcal{A}$ about some individuals in (4)–(6). Intuitively, $\mathcal{T}$ states that every child has a parent, adopted child is a child, and male and female are disjoint. The rules $\mathcal{P}$ contain some 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*) the constraint that a child has for sure at most one father, unless it is adopted (where $\bot$ stands for falsity); finally (11)–(12) single out contact persons for children, which by default are the parents; for adopted children, fathers from the ontology are omitted if some other contact exists. The rules and the ontology interact via DL-atoms, which are the expressions starting with "DL"; e.g., $\mathrm{DL}[\mathit{Male} \uplus \mathit{boy};\ \mathit{Male}](X)$ informally selects all individuals c, such that $\mathit{Male}(c)$ is provable from $\mathcal{O}$ after temporarily adding for boys the assertions that they are male in the ontology.

The semantics of DL-programs was given in the seminal paper [37] in terms of answer sets, as a generalization of the answer set semantics of nonmonotonic logic programs [46]. In this way, DL-programs are an extension of answer set programming (ASP) [18] in which the user can evaluate in the rules queries over an ontology via DL-atoms. Notably, DL-atoms enable a bidirectional information flow between the rules and the ontology, which may even be cyclic; this makes DL-programs quite expressive, and allows one to formulate advanced reasoning applications on ontologies, such as extended closed-world or terminological default reasoning [37].

On the other hand, the information flow can lead to inconsistency, i.e., that no answer set of the DL-program exists, even if the ontology and rules are perfectly consistent when considered separately; this happens in the example above, where the DL-program has no answer set. An inconsistent DL-program yields no information and is of no use for constructive problem solving; it may be viewed as broken and in need of an appropriate management of this situation. Systems for evaluating DL-programs, among them dlvhex[1] and DReW,[2] however can not resolve inconsistencies easily; this is clearly a drawback for their deployment to applications.

1 www.kr.tuwien.ac.at/research/systems/dlvhex.
2 www.kr.tuwien.ac.at/research/systems/drew.

Adequate treatment of inconsistent information is a ubiquitous challenge faced by many KR formalisms in various settings. The issue has been extensively studied in various fields, e.g. diagnosis [73], nonmonotonic reasoning [17,76], belief revision [1,42], knowledge base updates [28], databases (see [10] for an overview) and many others (e.g., [11,67,62,25]). Although a large number of "inconsistency-tolerant" approaches exist (see Section 7 for a discussion), most of them are applicable only to formalisms that are based on a single underlying logic. DL-programs in turn constitute a hybrid formalism, and existing approaches can not be readily applied for such a setting; thus, suitable methods for inconsistency handling in DL-programs are needed.

In this work, we address this need and develop techniques for repairing inconsistent DL-programs. Our main contributions can be summarized as follows.

(1) We formalize repairing DL-programs and introduce the notions of repair and repair answer set. They are based on changes of the assertions in the ontology that enable answer sets. As it turns out, repair answer sets do not have higher complexity than ordinary answer sets (more precisely, weak and *flp* answer sets) if queries in DL-atoms are evaluable in polynomial time; to ensure this, we concentrate on the prominent Description Logic $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ from the DL-Lite family [21]. Furthermore, we model repair preference by functions σ that select preferred repairs from a set of candidate repairs. As selecting most preferred repairs in a repair ordering may be a source of complexity, [54], we focus on selections σ that allow to filter preferred repairs independent of other repairs (which is relevant in practice).
(2) The task of repair computation involves a generalized ontology repair problem (ORP), which arises from a candidate answer set and the DL-atoms of the program. It consists of two sets D_1 and D_2 containing entailment and non-entailment queries to the ontology, respectively, under temporary assertions induced by the answer set candidate, and asks for an ABox satisfying these sets. Importantly, if a selection function σ is independent, the σ-selected ABoxes also yield, modulo a conditional check on the rules part, the σ-selected repairs of the program. Unsurprisingly, the ORP problem is intractable (NP-complete) for $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ in general, and NP-hard even in elementary ontology settings, due to the temporary assertions. However, we identify several tractable cases of σ-selections that are useful in practice. The ORP problem is of independent interest, as it can arise in a general context where multiple ontologies are integrated which share the taxonomy and some defeasible data, where queries serve as constraints.
(3) To optimize repair answer set computation, we introduce support sets as means to shortcut the ontology access for query evaluation. Informally, a support set of a DL-atom is a portion of the data in the ontology and the answer set from which the entailment of the query in the DL-atom follows; by a simple ontology enhancement, this data can be described entirely in terms of data in the ontology. Furthermore, support sets lift faithfully to the nonground level, i.e., can be schematically described, and the latter can for $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ ontologies not only be efficiently computed, but are also small; this provides the basis for scalability in exploitation. Using support sets likewise proved to be effective for evaluating DL-programs, as was shown in [32]; they can be seen as non-ground justifications why a query to the ontology evaluates to true and informally generalize explanations of positive query answers [16] to a setting with further ad-hoc input data.
(4) Utilizing support sets, we devise an algorithm for the effective computation of deletion repairs of DL-programs under weak and *flp*-answer set semantics, and we discuss potential generalizations. The algorithm is implemented within the dlvhex answer set solving framework, using a declarative approach for support set evaluation. Furthermore, we report results of an extensive experimental evaluation of the implementation on a suite of benchmarks that gather scenarios of different characteristics. The results provide evidence for the effectiveness of the method and scalability with respect to intuitively increasing inconsistency in the data.

Organization. The remainder of this article is organized as follows. Section 2 provides necessary preliminaries on DL-programs. In Section 3, the notions of repair and repair answer sets are introduced and a detailed analysis of their computational complexity is presented. Section 4 elaborates on support sets as optimization means and algorithms for deletion repair computation of DL-programs over $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ ontologies based on them. In Section 5 the structure of the prototype and the implementation details are given, and in Section 6 the evaluation results are presented and analyzed. A comprehensive discussion of further and related work is given in Section 7, followed by concluding remarks and an outlook in Section 8. In order not to distract from the flow of reading, longer proofs have been moved to the Appendix.

This article significantly extends the preliminary work in [33,35].

2. Preliminaries

In this section, we recall basic notions of Description Logics, where we focus on $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ [21,70], and DL-programs [37]; for more background on Description Logics, see [4].

2.1. Description logic knowledge bases

We consider Description Logic (DL) knowledge bases (KBs) over a signature $\Sigma_{\mathcal{O}} = \langle \mathbf{I}, \mathbf{C}, \mathbf{R} \rangle$ with a set $\mathbf{I}$ of individuals (constants), a set $\mathbf{C}$ of concept names (unary predicates), and a set $\mathbf{R}$ of role names (binary predicates) as usual.

A *DL knowledge base* (or *ontology*) is a pair $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ of a *TBox* $\mathcal{T}$ and an *ABox* $\mathcal{A}$, which are finite sets of formulas capturing taxonomic resp. factual knowledge, whose form depends on the underlying DL. In abuse of notation, we also write $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ viewing $\mathcal{O}$ as a set of formulas.

Syntax. In *DL-Lite*$_\mathcal{A}$, concepts C, denoting sets of objects, and roles R, denoting binary relations between objects, obey the following syntax, where $A \in \mathbf{C}$ is an atomic concept and $P \in \mathbf{R}$ an atomic role:

$$C \rightarrow A \mid \exists R, \qquad D \rightarrow C \mid \neg C, \qquad R \rightarrow P \mid P^-, \qquad S \rightarrow R \mid \neg R.$$

DL-Lite$_\mathcal{A}$ TBox axioms are then of the form:

$$C \sqsubseteq D, \qquad R \sqsubseteq S, \qquad (\mathit{funct}\ R)$$

Axioms where $D = C$ resp. $S = R$ are *positive inclusion axioms* and where $D = \neg C$ resp. $S = \neg R$ are *disjointness axioms*; $(\mathit{funct}\ R)$ is a *functionality axiom*. As a further constraint, roughly speaking in *DL-Lite*$_\mathcal{A}$ (inverse) functional roles can not be specialized, i.e., they can not appear on the right-hand side of positive inclusion axioms; for formal details, see [70] An *assertion* is a formula $A(c)$ or $P(c,d)$, where $A \in \mathbf{C}$, $P \in \mathbf{R}$, and $c, d \in \mathbf{I}$ (called *positive*) or its negation, i.e., $\neg A(c)$ resp. $\neg P(c,d)$ (*negative*).[3] An example of a *DL-Lite*$_\mathcal{A}$ ontology is given in Fig. 1.

Semantics. The semantics of DL ontologies $\mathcal{O}$ is based on first-order interpretations [21].

Definition 2 *(Interpretation).* An *interpretation* is a pair $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$ of a non-empty domain $\Delta^\mathcal{I}$ and an interpretation function $\cdot^\mathcal{I}$ that assigns to each individual $c \in \mathbf{I}$ an object $c^\mathcal{I} \in \Delta^\mathcal{I}$, to each concept name C a subset $C^\mathcal{I}$ of $\Delta^\mathcal{I}$, and to each role name R a binary relation $R^\mathcal{I}$ over $\Delta^\mathcal{I}$.

An interpretation I extends inductively to non-atomic concepts C and roles R according to the concept resp. role constructors; as for *DL-Lite*$_\mathcal{A}$, $(\exists R)^\mathcal{I} = \{o_1 \mid \langle o_1, o_2 \rangle \in R^\mathcal{I}\}$ and $(\neg C)^\mathcal{I} = \Delta^\mathcal{I} \backslash C^\mathcal{I}$, and $R^{-\mathcal{I}} = \{\langle o_1, o_2 \rangle \mid \langle o_2, o_1 \rangle \in R^\mathcal{I}\}$ and $(\neg R)^\mathcal{I} = \Delta^\mathcal{I} \times \Delta^\mathcal{I} \setminus R^\mathcal{I}$. Based on this satisfaction of formulas in $\mathcal{I} \models \omega$ is defined as follows.

Definition 3 *(Satisfaction). Satisfaction* of an axiom respectively assertion w.r.t. an interpretation $\mathcal{I}$ is as follows:

- $\mathcal{I} \models C \sqsubseteq D$, if $C^\mathcal{I} \subseteq D^\mathcal{I}$;
- $\mathcal{I} \models R \sqsubseteq S$, if $R^\mathcal{I} \subseteq S^\mathcal{I}$;
- $\mathcal{I} \models \mathit{funct}(R)$, if $(o_1, o_2) \in R^\mathcal{I}$ and $(o_1, o_3) \in R^\mathcal{I}$ implies $o_2 = o_3$ for all $o_1, o_2, o_3 \in \Delta^\mathcal{I}$;
- $\mathcal{I} \models C(a)$, if $a^\mathcal{I} \in C^\mathcal{I}$ and $\mathcal{I} \models \neg C(a)$, if $a^\mathcal{I} \in \Delta^\mathcal{I} \backslash C^\mathcal{I}$;
- $\mathcal{I} \models P(a,b)$, if $(a^\mathcal{I}, b^\mathcal{I}) \in P^\mathcal{I}$ and $\mathcal{I} \models \neg P(a,b)$, if $(a^\mathcal{I}, b^\mathcal{I}) \in \Delta^\mathcal{I} \times \Delta^\mathcal{I} \backslash P^\mathcal{I}$.

Furthermore, $\mathcal{I}$ satisfies a set of formulas Γ, denoted $\mathcal{I} \models \Gamma$, if $\mathcal{I} \models \alpha$ for each $\alpha \in \Gamma$.

A TBox $\mathcal{T}$, ABox $\mathcal{A}$ respectively ontology $\mathcal{O}$ is *satisfiable* (or *consistent*), if some interpretation $\mathcal{I}$ satisfies it. We call $\mathcal{A}$ *consistent with* $\mathcal{T}$, if $\mathcal{T} \cup \mathcal{A}$ is consistent.

Example 4 *(cont'd).* The ontology $\mathcal{O}$ in Fig. 1 is consistent, since there exists a satisfying interpretation $\mathcal{I} = \langle \Delta^\mathcal{I}, \cdot^\mathcal{I} \rangle$, defined by setting $\Delta^\mathcal{I} = \{\mathit{john}, \mathit{pat}\}$, $\mathit{Male}^\mathcal{I} = \{\mathit{john}, \mathit{pat}\}$, $\mathit{hasParent}^\mathcal{I} = \{(\mathit{john}, \mathit{pat})\}$ and $\mathit{Child}^\mathcal{I} = \mathit{Female}^\mathcal{I} = \emptyset$. The ontology $\mathcal{O}' = \mathcal{O} \cup \{\mathit{Female}(\mathit{pat})\}$ does not have any model, and thus is inconsistent.

It has been shown that in *DL-Lite*$_\mathcal{A}$ inconsistency arises by few assertions [21].

Proposition 5 *(cf. [21]). In DL-Lite$_\mathcal{A}$, for a given TBox $\mathcal{T}$ every $\subseteq$-minimal ABox $\mathcal{A}$ such that $\mathcal{T} \cup \mathcal{A}$ is inconsistent fulfills $|\mathcal{A}| \leq 2$.*

Throughout the paper, we consider ontologies in *DL-Lite*$_\mathcal{A}$ under the unique names assumption, i.e., ${o_1}^\mathcal{I} \neq {o_2}^\mathcal{I}$ whenever $o_1 \neq o_2$ holds in any interpretation.

2.2. DL-programs

A DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is given as a pair of a DL ontology $\mathcal{O}$ and a set $\mathcal{P}$ of DL-rules, which extend rules in non-monotonic logic programs with special DL-atoms. They are formed over a signature $\Sigma_\Pi = \langle \mathcal{C}, \mathbf{P}, \mathbf{I}, \mathbf{C}, \mathbf{R} \rangle$, where $\Sigma_\mathcal{P} = \langle \mathcal{C}, \mathbf{P} \rangle$ is a signature of the rule part $\mathcal{P}$ with $\mathcal{C}$ being a finite set of constant symbols, and $\mathbf{P}$ a finite set of predicate symbols (called

[3] Negative assertions $\neg F(\vec{t})$ are easily compiled to positive ones using a fresh concept resp. role name $F_\neg$ and $F_\neg(\vec{t})$, $F_\neg \sqsubseteq \neg F$.

lp predicates) of arities ≥ 0, and $\Sigma_{\mathcal{O}} = \langle \mathbf{I}, \mathbf{C}, \mathbf{R} \rangle$ is a DL signature. The set $\mathbf{P}$ is disjoint with $\mathbf{C}, \mathbf{R}$. For simplicity, we assume here $\mathcal{C} = \mathbf{I}$.

Syntax. A *(disjunctive) DL-program* $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ consists of a DL ontology $\mathcal{O}$ and a finite set $\mathcal{P}$ of *DL-rules* r of the form

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

where *not* is *negation as failure (NAF)*[4] and each a_i, $0 \leq i \leq n$, is a first-order atom $p(\vec{t})$ with predicate $p \in \mathbf{P}$ (called *ordinary* or *lp-atom*) and each b_i, $1 \leq i \leq m$, is either an lp-atom or a *DL-atom*. If $n = 0$, the rule is a *constraint*, and if $n \leq 1$, it is *normal*. The notions of a head and a body of a rule are naturally inherited from normal logic programs, i.e. for a DL-rule r of the form (1), $H(r) = \{a_1, \ldots, a_n\}$ is called the *head* of r, and $B(r) = \{b_1, \ldots, b_k, \textit{not}\ b_{k+1}, \ldots, \textit{not}\ b_m\}$ is called the *body* of r.

A *DL-atom* $a(\vec{t})$ is of the form

$$\mathrm{DL}[\lambda;\ Q\,](\vec{t}), \tag{2}$$

where

(a) $\lambda = S_1\, op_1\, p_1, \ldots, S_m\, op_m\, p_m$, $m \geq 0$ is the *input list* and for each i, $1 \leq i \leq m$, $S_i \in \mathbf{C} \cup \mathbf{R}$, $op_i \in \{\uplus, \cup\!\!\!-, \cap\!\!\!-\}$ is an *update operator*, 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, while $op_i = \cap\!\!\!-$ constrains S_i to p_i;
(b) $Q\,(\vec{t})$ is a *DL-query*, which has one of the forms (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; (iii) Q is an inclusion axiom and $\vec{t} = \epsilon$; (iv) Q is a disjointness axiom and $\vec{t} = \epsilon$; or (v) $\neg Q'(\vec{t})$ where $Q'(\vec{t})$ is from (i)–(iv). We omit $(\vec{t})$ for $\vec{t} = \epsilon$.

Example 6 *(cont'd).* Consider a ground version DL[*Male* $\uplus$ *boy*; *Male*](*pat*) of the DL-atom in the rule (9) of Π in Fig. 1. It has a DL-query *Male*(*pat*); its list $\lambda = \textit{Male} \uplus \textit{boy}$ contains an input predicate *boy* which extends the ontology predicate *Male* via an update operator $\uplus$.

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 Herbrand base w.r.t. $\mathcal{C}$ and $\mathbf{P}$ (i.e. all ground atoms over $\mathcal{C}$ and $\mathbf{P}$); I satisfies an lp- or DL-atom a, if

(i) $a \in I$, if a is an lp-atom, and
(ii) $\langle \mathcal{O} \cup \lambda^I(a) \rangle \models Q\,(\vec{t})$ where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, if a is a DL-atom of form (2), where

$$\lambda^I(a) = \bigcup_{i=1}^{m} A_i(I) \tag{3}$$

and

- $A_i(I) = \{S_i(\vec{c}) \mid p_i(\vec{c}) \in I\}$, for $op_i = \uplus$;
- $A_i(I) = \{\neg S_i(\vec{c}) \mid p_i(\vec{c}) \in I\}$, for $op_i = \cup\!\!\!-$;
- $A_i(I) = \{\neg S_i(\vec{c}) \mid p_i(\vec{c}) \in HB_\Pi \setminus I\}$, for $op_i = \cap\!\!\!-$.

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 ω (atom, rule, etc.) with $I \models^{\mathcal{O}} \omega$. A model I of ω is *minimal*, if no model I' of ω exists such that $I' \subset I$.

Example 7 *(cont'd).* The interpretation $I = \{\textit{ischildof}(\textit{john}, \textit{alex}), \textit{boy}(\textit{john})\}$ satisfies the DL-atom $o = $ DL[*Child* $\uplus$ *boy*; *Male*](*john*), as $\mathcal{O} \cup \lambda^I(o) \models \textit{Male}(\textit{john})$. Furthermore, $I \not\models^{\mathcal{O}}$ DL[; *Adopted*](*john*), since the input list of DL[; *Adopted*](*john*) is empty and $\mathcal{O} \not\models \textit{Adopted}(\textit{john})$.

Answer sets. Various semantics for DL-programs extend the answer sets semantics of (disjunctive) logic programs [46] to DL-programs, e.g. [37,59,82,77]. We concentrate here on *weak answer sets* [37], in which DL-atoms are treated like atoms under NAF, and *flp answer sets* [38], which obey a stronger foundedness condition. Both are like answers sets of ordinary logic programs defined as interpretations that are minimal models of a program reduct, which intuitively captures that assumption-based application of the rules on an interpretation can reconstruct the latter.

Definition 8 *(Weak answer sets).* Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program. The *weak reduct* of $\mathcal{P}$ relative to $\mathcal{O}$ and to an interpretation $I \subseteq HB_\Pi$, denoted by $\mathcal{P}^{I,\mathcal{O}}_{weak}$ is the ordinary positive program obtained from $gr(\mathcal{P})$ by deleting

[4] Strong negation $\neg a$ can be added resp. emulated as usual [37].

- all DL-rules r such that either $I \not\models^{\mathcal{O}} a$ for some DL-atom $a \in B^+(r)$, or $I \models^{\mathcal{O}} l$ for some $l \in B^-(r)$; and
- from every remaining DL-rule r all the DL-atoms in $B^+(r)$ and all the literals in $B^-(r)$.

A *weak answer set* of Π is any interpretation $I \subseteq HB_\Pi$ that is a minimal model of $\mathcal{P}^{I,\mathcal{O}}_{weak}$. By $AS_{weak}(\Pi)$ we denote the set of all weak answer sets of Π.

Note that $\mathcal{P}^{I,\mathcal{O}}_{weak}$ is an ordinary ground positive program without DL-atoms and default-negated literals, which has the least (unique minimal) model if each rule in $\mathcal{P}$ is definite (i.e., $n = 1$ in (1)).

Example 9. Let $\mathcal{O}$ be as in Fig. 1 and let the rule set $\mathcal{P}$ be as follows:

$$\mathcal{P} = \left\{ \begin{array}{ll} (7) & \mathit{ischildof}(\mathit{john}, \mathit{alex});\quad (8)\ \mathit{boy}(\mathit{john}); \\ (9) & \mathit{hasfather}(\mathit{john}, \mathit{pat}) \leftarrow \mathrm{DL}[\mathit{Male} \uplus \mathit{boy};\ \mathit{Male}](\mathit{pat}), \mathrm{DL}[;\ \mathit{hasParent}](\mathit{john}, \mathit{pat}); \\ (10) & \mathit{contact}(\mathit{john}, \mathit{pat}) \leftarrow \mathrm{DL}[;\ \mathit{hasParent}](\mathit{john}, \mathit{pat}), \mathit{not}\ \mathit{omit}(\mathit{john}, \mathit{pat}); \\ (11) & \mathit{omit}(\mathit{john}, \mathit{pat}) \leftarrow \mathrm{DL}[;\ \mathit{Adopted}](\mathit{john}), \mathit{hasfather}(\mathit{john}, \mathit{pat}), \mathit{contact}(\mathit{john}, \mathit{alex}) \end{array} \right\}$$

Consider $I = \{\mathit{ischild}(\mathit{john}, \mathit{alex}), \mathit{boy}(\mathit{john}), \mathit{contact}(\mathit{john}, \mathit{pat}), \mathit{hasfather}(\mathit{john}, \mathit{pat})\}$. The *weak*-reduct $\mathcal{P}^{I,\mathcal{O}}_{weak}$ contains the following rules:

$$\mathcal{P}^{I,\mathcal{O}}_{weak} = \left\{ \begin{array}{ll} (7) & \mathit{ischildof}(\mathit{john}, \mathit{alex});\quad (8)\ \mathit{boy}(\mathit{john}); \\ (9') & \mathit{hasfather}(\mathit{john}, \mathit{pat}); \\ (10') & \mathit{contact}(\mathit{john}, \mathit{pat}) \end{array} \right\}.$$

The interpretation I is a *weak*-answer set of Π, since I is a minimal model of $\mathcal{P}^{I,\mathcal{O}}_{weak}$. In fact, $AS_{weak}(\Pi) = \{I\}$ holds.

The *flp*-answer set semantics is defined as follows.

Definition 10 *(flp answer sets).* Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program. The *flp*-reduct of $\mathcal{P}$ relative to $\mathcal{O}$ and an interpretation $I \subseteq HB_\Pi$ is the set of rules $\mathcal{P}^{I,\mathcal{O}}_{flp} = \{r^{I,\mathcal{O}}_{flp} \mid r \in \mathcal{P}\}$ where $r^{I,\mathcal{O}}_{flp} = r$, if the body of r is satisfied, i.e., $I \models^{\mathcal{O}} b_i$, for all b_i, $1 \le i \le k$ and $I \not\models^{\mathcal{O}} b_j$, for all $k < j \le m$; otherwise, $r^{I,\mathcal{O}}_{flp}$ is empty.

An *flp-answer set* of Π is any interpretation $I \subseteq HB_\Pi$ that is a minimal model of $\mathcal{P}^{I,\mathcal{O}}_{flp}$. By $AS_{flp}(\Pi)$ we denote the set of all *flp* answer sets of Π.

Example 11. Reconsider $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ and I from Example 9. The reduct $\mathcal{P}^{I,\mathcal{O}}_{flp}$ contains all rules of $\mathcal{P}$ apart from (11). It is not difficult to verify that I is a minimal model of $\mathcal{P}^{I,\mathcal{O}}_{flp}$, and hence an *flp*-answer set of Π; in fact $AS_{flp}(\Pi) = \{I\}$. □

In general, the set of all *flp* answer sets of a DL-program is contained in the set of its strong answer sets [37], which in turn is contained in the set of weak answer sets. Strong answer sets coincide with *flp* ones in some cases, in particular, if the constraint operator $\cap\!\!\!\!-$ does not occur in Π. For more information, see [37,82].

When dealing with evaluation of DL-atoms w.r.t. a given interpretation I, it is often convenient to consider input assertions defined as follows.

Definition 12 *(Input assertion).* Given a DL-atom $d = \mathrm{DL}[\lambda;\ Q](\vec{t})$ and $P \circ p \in \lambda$, $\circ \in \{\uplus, \cup\!\!\!\!-\}$, we call $P_p(\vec{c})$ an *input assertion for* d, where P_p is a fresh ontology predicate and $\vec{c} \subseteq \mathcal{C}$. By $\mathcal{A}_d$ we denote the set of all such assertions.

For a TBox $\mathcal{T}$ and a DL-atom d, we let

$$\mathcal{T}_d = \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\} \cup \{P_p \sqsubseteq \neg P \mid P \cup\!\!\!\!- p \in \lambda\},$$

and for an interpretation I, we let

$$\mathcal{O}^I_d = \mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{c}) \in \mathcal{A}_d \mid p(\vec{c}) \in I\}.$$

We then have:

Lemma 13. *For every $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, DL-atom $d = \mathrm{DL}[\lambda;\ Q](\vec{t})$ and interpretation I, it holds that $I \models^{\mathcal{O}} d$ iff $I \models^{\mathcal{O}^I_d} \mathrm{DL}[\epsilon;\ Q](\vec{t})$ iff $\mathcal{O}^I_d \models Q(\vec{t})$.*

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

Table 1
Complexity of deciding weak and *flp*answer set existence for ground DL-programs over *DL-Lite*$_{\mathcal{A}}$ (completeness results).

$AS_x(\Pi)$ | $RAS_x(\Pi) \neq \emptyset$?	Normal Π	Disjunctive Π
$x = weak$	NP | NP	$\Sigma_2^P \mid \Sigma_2^P$
$x = flp$	$\Sigma_2^P \mid \Sigma_2^P$	$\Sigma_2^P \mid \Sigma_2^P$

3. Repair semantics

The powerful formalism of DL-programs permits a bidirectional information flow between the rule part and the ontology, which makes it attractive for various application scenarios. This information flow, however, can have unforeseen effects and cause that a DL-program has no answer set; we call such DL-programs *inconsistent*.

Example 14 *(cont'd).* The DL-program Π from Fig. 1 does not have any weak nor *flp* answer set, and thus is inconsistent. The inconsistency arises in this program as *john*, who is not provably adopted, has *pat* as father by the ontology, and by the local information possibly also *alex*; this causes the constraint (10) to be violated. □

Absence of answer sets makes a DL-program unusable, which calls for a remedy to this problem. As mentioned earlier, there are two principled approaches: to tolerate inconsistency, in the sense that reasoning does not trivialize, or to *repair* the program, i.e., change formulas in it to obtain consistency. As regards DL-programs (and likewise similar hybrid formalisms), previous works [72,40] focused on inconsistency tolerance, by suppressing or weakening information that leads to inconsistency in model building.

In this section, we consider DL-program repair from a theoretical perspective by introducing a repair semantics and analyzing its computational complexity. In our setting, we assume that the rule part $\mathcal{P}$, which is on top of the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, is reliable and that the cause for inconsistency is in the latter. Thus when searching for a repair, modifications should only be applied to $\mathcal{O}$. In principle, the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ of the ontology could be subject to change; however, as usually the TBox is well-developed and a suitable TBox change is less clear in general (the more by an external user), we confine to change only the ABox. For example, in the DL-program Π in Example 14 it would be sufficient to delete the assertion $hasParent(john, pat)$ from the ABox to obtain a (weak respectively *flp*) answer set.

From a general perspective, our goal is, given a possibly inconsistent DL-program, to find an ABox $\mathcal{A}'$ such that replacing the ABox $\mathcal{A}$ by $\mathcal{A}'$ makes the DL-program consistent. The answer sets of such a "repaired" DL-program are then referred to as *repair answer sets* of the program.

Formally, they are defined as follows.

Definition 15 *(x-repairs and x-repair answer sets).* Given a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, an ABox $\mathcal{A}'$ is an *x-repair* of Π, where $x \in \{flp, weak\}$, if

(i) $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ is consistent, and
(ii) $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$ has some x-answer set.

By $rep_x(\Pi)$ we denote the set of all x-repairs of Π. An interpretation I is an *x-repair answer set* of Π, if $I \in AS_x(\Pi')$, where $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$, $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$, and $\mathcal{A}' \in rep_x(\Pi)$. By $RAS_x(\Pi)$ we denote the set of all x-repair answer sets of Π.

Furthermore, by $rep_x^I(\Pi) = \{\mathcal{A}' \in rep_x(\Pi) \mid I \in AS_x(\Pi'), \Pi' = \langle \mathcal{O}', \mathcal{P} \rangle, \mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle\}$ we denote the set of all ABoxes $\mathcal{A}'$ under which I becomes an x-answer set of Π.

Example 16 *(cont'd).* Reconsider Π in Example 1. The interpretation $I_1 = \{boy(john), ischildof(john, alex)\}$ is an *flp*-repair answer set with *flp*-repair $\mathcal{A}'_1 = \{Male(john), Male(pat)\}$. Another *flp*-repair for I_1 is $\mathcal{A}'_2 = \{hasParent(john, pat), Female(pat), Male(john)\}$. The interpretation I_1 is also a *weak*-repair answer set with the *weak*-repairs $\mathcal{A}'_1$ and $\mathcal{A}'_2$.

3.1. Complexity of RAS existence for DL-programs over DL-Lite$_{\mathcal{A}}$ DL

We now look at the problem of deciding whether a given DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ has an x-(repair) answer set for $x \in \{flp, weak\}$. Table 1 compactly summarizes our complexity results for this problem for $\mathcal{O}$ in *DL-Lite*$_{\mathcal{A}}$.

Before formally addressing the complexity of repair answer sets, we first state the following proposition:

Proposition 17. *Given any $I \subseteq HB_\Pi$, $\mathcal{O}$ in DL-Lite$_{\mathcal{A}}$, and a DL-atom $a = \mathrm{DL}[\lambda;\ Q](\vec{t})$, deciding $I \models^{\mathcal{O}} a$ is feasible in polynomial time.*

Proof. Deciding whether $I \models^{\mathcal{O}} a$ is equivalent to checking $\mathcal{O} \cup \lambda^I(a) \models Q(\vec{t})$. As instance checking is known to be polynomial [21] in *DL-Lite*$_{\mathcal{A}}$, the result immediately follows. □

We are now ready to formally prove basic complexity results for checking the existence of repair answer sets for a DL-program.

Theorem 18. *Given a ground DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ with $\mathcal{O}$ in DL-Lite$_{\mathcal{A}}$ deciding whether $RAS_x(\Pi) \neq \emptyset$ is*

(*i*) NP*-complete for normal Π and $x = weak$;*
(*ii*) Σ_2^P*-complete for arbitrary Π and $x \in \{weak, flp\}$;*
(*iii*) Σ_2^P*-complete for normal Π and $x = flp$.*

We remark that the problem in (i) remains NP-hard even if Π consists of a stratified DL-program in the sense of [37] that has additional constraints, cf. [79]. In (ii), the Σ_2^P-hardness is inherited from the complexity of answer sets of ordinary disjunctive logic programs. In (iii), the complexity drops to NP-completeness if the update operator $\cap\!\!\!-$ is excluded, as then the *flp*- and the strong answer sets of such DL-programs are guaranteed to coincide and deciding strong answer set existence is co-NP-complete [36]. Furthermore, all results extend to the setting where independent selection functions for determining preferred solutions, which are introduced in the next section, of polynomial time complexity are available.

3.2. Selection functions

Clearly, not all repairs are equally useful or interesting for a certain scenario. For instance, repairs that have no common assertions with the original ABox might be unwanted; repairs that introduce assertions that are not in the initial ABox; repairs that would cause non-minimal change etc. Formally, we model preferred repairs using a *selection* function:

Definition 19 *(Selection function).* A *selection function* is a mapping $\sigma : 2^{\mathcal{AB}} \times \mathcal{AB} \to 2^{\mathcal{AB}}$, where $\mathcal{AB}$ is the set of all ABoxes, that assigns every pair $(S, \mathcal{A})$ of a set S of ABoxes and an ABox $\mathcal{A}$ a set $\sigma(S, \mathcal{A}) \subseteq S$ of *preferred* (or *selected*) ABoxes.

This notion captures a variety of selection principles, including minimal repairs according to some preference relation, or some global selection property. We then define:

Definition 20 *((σ, x)-repairs and (σ, x)-repair answer sets).* Given $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, and a selection σ, we call $rep_{(\sigma,x)}(\Pi) = \sigma(rep_x(\Pi), \mathcal{A})$ the *(σ, x)-repairs* of Π. An interpretation $I \subseteq HB_\Pi$ is a *(σ, x)-repair answer set* of Π, if $rep^I_{(\sigma,x)}(\Pi) \neq \emptyset$, where $rep^I_{(\sigma,x)}(\Pi) = rep_{(\sigma,x)}(\Pi) \cap rep^I_x(\Pi)$; by $RAS_{(\sigma,x)}(\Pi)$ we denote the set of all such repair answer sets.

Example 21. Consider a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O} = \langle \emptyset, \mathcal{A} \rangle = \{Child(john)\}$ and $\mathcal{P}$ is as follows:

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ male(john); \\ (2)\ pupil(john) \leftarrow \mathrm{DL}[;\ studiesAt](john, sch80); \\ (3)\ boy(john) \leftarrow \mathrm{DL}[Child \uplus boy;\ Child](john), male(john); \\ (4)\ \bot \leftarrow boy(john), not\ pupil(john) \end{array} \right\}.$$

The interpretation $I = \{male(john), pupil(john), boy(john)\}$ is a $(\sigma, weak)$-repair answer set of Π with a possible $(\sigma, weak)$-repair $\mathcal{A}' = \{studiesAt(john, sch80)\}$, i.e. $I \in RAS_{(\sigma,weak)}(\Pi)$ and $\mathcal{A}' \in rep^I_{(\sigma,weak)}(\Pi)$, where σ chooses repairs $\mathcal{A}'$, such that the set difference between $\mathcal{A}$ and $\mathcal{A}'$ contains at most 2 assertions. Indeed, we have that $\mathcal{P}^{I,\mathcal{O}'}_{weak} = \{male(john);\ pupil(john);\ boy(john)\}$, and clearly I is its minimal model.

Moreover, $I \in RAS_{(\sigma,flp)}(\Pi)$, and $\mathcal{A}'' = \{studiesAt(john, sch80), Child(john)\} \in rep^I_{\sigma,flp}(\Pi)$ is an (σ, flp)-repair of Π. To verify this, observe that the reduct $\mathcal{P}^{I,\mathcal{O}''}_{flp}$ contains the rules (1)–(3), and I is a minimal model of $\langle \mathcal{A}'', \mathcal{P}^{I,\mathcal{O}''}_{flp} \rangle$, where $\mathcal{O}'' = \langle \emptyset, \mathcal{A}'' \rangle$. Note that while $\mathcal{A}'' \in rep^I_{(\sigma,weak)}(\Pi)$, we have that $\mathcal{A}' \notin rep^I_{(\sigma,flp)}(\Pi)$. More specifically, I is not a minimal model of $\langle \mathcal{O}', \mathcal{P}^{I,\mathcal{O}'}_{flp} \rangle$, where $\mathcal{P}^{I,\mathcal{O}'}_{flp} = \mathcal{P}^{I,\mathcal{O}''}_{flp}$ and $\mathcal{O}' = \langle \emptyset, \mathcal{A}' \rangle$, since there is a smaller model $I' = I \setminus \{boy(john)\}$, which satisfies all rules of $\mathcal{P}^{I,\mathcal{O}'}_{flp}$.

The repair $\mathcal{A}'_1 = \{Male(john), Male(pat)\}$ from Example 16 is in $rep^{I_1}_{\sigma_1,x}(\Pi)$ for $I_1 = \{ischildof(john, alex),\ boy(john)\}$, where $x \in \{weak, flp\}$ and σ_1 selects deletion repairs, i.e. subsets of $\mathcal{A}$. Furthermore, the ABox $\mathcal{A}'_2 = \{hasParent(john, pat), Male(john), Female(pat)\}$ is in $rep^{I_1}_{\sigma_2,x}(\Pi)$, where $x \in \{weak, flp\}$, and σ_2 selects repairs $\mathcal{A}'$, which differ from $\mathcal{A}$ only on assertions over gender predicates *Male*, *Female*, and $|\mathcal{A}| = |\mathcal{A}'|$. Consequently, $I_1 \in RAS_{(\sigma_1,x)}(\Pi)$ and $I_2 \in RAS_{(\sigma_2,x)}(\Pi)$ for $x \in \{weak, flp\}$. $\Box$

In general, even polynomially computable selections σ may incur intractability, e.g., selecting ABoxes $\mathcal{A}'$ with set-minimal change to $\mathcal{A}$, or with smallest Dalal (Hamming) distance (see e.g. [54]). Naturally, we aim at selections that are useful in practice and have benign computational properties, which are pragmatic specifically for our problem.

Definition 22 *(Independent selection).* A selection $\sigma : 2^{\mathcal{AB}} \times \mathcal{AB} \to 2^{\mathcal{AB}}$ is *independent*, if $\sigma(S, \mathcal{A}) = \sigma(S', \mathcal{A}) \cup \sigma(S \setminus S', \mathcal{A})$ whenever $S' \subseteq S$.

Example 23. All selection functions considered in Example 21 are independent. The selection function σ, which seeks repairs $\mathcal{A}'$ that contain a minimal number of changes in assertions over predicate *Adopted* w.r.t. $\mathcal{A}$ in Example 1 is not independent, since to find the preferred σ-repair one needs to compute all repair candidates first, and then choose the best one among them. □

Independence allows us to decide whether a given repair $\mathcal{A}' \in S$ is selected by σ without looking at other repairs, and composition works here easily. This makes the introduced property valuable, since independent selection functions of different kind can be conveniently combined without a major increase in the complexity. Formally,

Proposition 24. *If selection functions σ_1 and σ_2 are independent, then their composition $\sigma_1 \circ \sigma_2$ is also independent.*

Proof. We show that whenever $S' \subseteq S$ it holds that $\sigma_1(\sigma_2(S, \mathcal{A}), \mathcal{A}) = \sigma_1(\sigma_2(S', \mathcal{A}), \mathcal{A}) \cup \sigma_1(\sigma_2(S \setminus S', \mathcal{A}), \mathcal{A})$. By independence of σ_2 we have $\sigma_2(S, \mathcal{A}) = \sigma_2(S', \mathcal{A}) \cup \sigma_2(S \setminus S', \mathcal{A})$. Hence, $\sigma_2(S', \mathcal{A}) \subseteq \sigma_2(S, \mathcal{A})$, and thus by independence of σ_1 we get $\sigma_1(\sigma_2(S, A), A) = \sigma_1(\sigma_2(S', \mathcal{A}), \mathcal{A}) \cup \sigma_1(\sigma_2(S, \mathcal{A}) \setminus \sigma_2(S', \mathcal{A}), \mathcal{A})$. As $\sigma_2(S, \mathcal{A}) \setminus \sigma_2(S', \mathcal{A}) = \sigma_2(S \setminus S', \mathcal{A})$, the result follows. □

Clearly, set-minimal change and smallest Dalal distance are not independent, as to decide whether $\mathcal{A}' \in \sigma(S, \mathcal{A})$ one has to compare $\mathcal{A}'$ with all other ABoxes from S. On the other hand, selecting all ABoxes such that $\mathcal{A}' \subseteq \mathcal{A}$, is obviously independent. The latter, and several other independent selections that are useful in practice, will be considered in the next section.

Independence leads to the following beneficial property.

Proposition 25. *For every Π and selection σ, if σ is independent, then $rep^I_{(\sigma,x)}(\Pi) \subseteq rep_{(\sigma,x)}(\Pi)$, for every $I \subseteq HB_\Pi$.*

Proof. By definition $rep_{(\sigma,x)}(\Pi) = \sigma(rep_x(\Pi), \mathcal{A})$ and $rep^I_{(\sigma,x)}(\Pi) = \sigma(rep^I_x(\Pi), \mathcal{A})$. Now as $rep^I_x(\Pi) \subseteq rep_x(\Pi)$ and σ is independent, we obtain $\sigma(rep_x(\Pi), \mathcal{A}) = \sigma(rep^I_x(\Pi), \mathcal{A}) \cup \sigma(rep_x(\Pi) \backslash rep^I_x(\Pi), \mathcal{A})$, from which the result is obtained. □

Proposition 25 implies that if we can turn an interpretation I into an answer set of Π by a σ-selected repair from the repairs which achieve this for I, then I is a σ-repair answer set of Π; that is, *local selection* is enough for a *global* σ-repair answer set. This will be exploited later in this section.

3.3. Ontology repair problem

In this section we introduce the Ontology Repair Problem (ORP), which is an important subtask of repair answer set computation. Intuitively, an ORP is the problem of identifying an ABox under which a simultaneous entailment and non-entailment of sets of queries, where further individual additions for each query are possible, is guaranteed. In our setting updates and queries are obtained from a candidate interpretation and values of DL-atoms, under which this interpretation is an answer set of a DL-program at hand (see Section 4 for details).

Let us now provide a formal definition for this repair problem.

Definition 26 *(Ontology repair problem (ORP)).* An *ontology repair problem (ORP)* is a triple $\mathcal{R} = \langle \mathcal{O}, D_1, D_2 \rangle$ where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is an ontology and $D_i = \{\langle U^i_j, Q^i_j \rangle \mid 1 \leq j \leq m_i\}$, $i = 1, 2$, are sets of pairs where each U^i_j is an ABox and each Q^i_j is a DL-query. A *repair* (*solution*) for $\mathcal{R}$ is any ABox $\mathcal{A}'$ such that

(i) the ontology $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ is consistent;
(ii) $\langle \mathcal{T}, \mathcal{A}' \cup U^1_j \rangle \models Q^1_j$ holds for $1 \leq j \leq m_1$;
(iii) $\langle \mathcal{T}, \mathcal{A}' \cup U^2_j \rangle \not\models Q^2_j$ holds for $1 \leq j \leq m_2$.

For an illustration of ORPs, we resort to the ontology from Fig. 1.

Example 27. Consider $\mathcal{R} = \langle \mathcal{O}, D_1, D_2 \rangle$ with $\mathcal{O}$ as in Fig. 1, and the following sets D_1 and D_2:

- $D_1 = \{\langle U^1_1, Q^1_1 \rangle, \langle U^1_2, Q^1_2 \rangle, \langle U^1_3, Q^1_3 \rangle\}$, where
  - $U^1_1 = \{Male(john)\}$, $Q^1_1 = Male(pat)$;
  - $U^1_2 = \emptyset$, $Q^1_2 = hasParent(john, pat)$;

  - $U_3^1 = \{Child(john)\}$, $Q_2^1 = Male(alex)$;
- $D_2 = \{\langle U_1^2, Q_1^2 \rangle\}$, where
  - $U_1^2 = \emptyset$, $Q_1^2 = Adopted(john)$.

One of the possible solutions to the described ORP is the ABox $\mathcal{A}' = \{Male(alex), hasParent(john, pat), Male(pat)\}$. Indeed, it is easy to verify that

- $\langle \mathcal{T}, \mathcal{A}' \cup Male(john) \rangle \models Male(pat)$, $\langle \mathcal{T}, \mathcal{A}' \rangle \models hasParent(john, pat)$, $\langle \mathcal{T}, \mathcal{A}' \cup Child(john) \rangle \models Male(alex)$;
- $\langle \mathcal{T}, \mathcal{A}' \rangle \not\models Adopted(john)$. □

We now analyze the complexity of the ORP problem in the general setting.

3.4. Complexity results for ORP

Unsurprisingly, the Ontology Repair Problem is intractable in general. However, this holds already for very simple ontologies, which we show in the next proposition.

Proposition 28. *Deciding whether an ORP $\mathcal{R} = \langle\langle \mathcal{T}, \mathcal{A}\rangle, D_1, D_2\rangle$ has some repair $\mathcal{A}'$ is* NP*-complete, and* NP*-hard even if $\mathcal{T}$ contains only positive concept inclusions and $\mathcal{A} = \emptyset$.*

In fact, even if both TBox and ABox are empty, the problem stays intractable, which is formally stated in the following proposition.

Theorem 29. *Deciding whether an ORP $\mathcal{R}=\langle\langle \mathcal{T}, \mathcal{A}\rangle, D_1, D_2\rangle$ has some repair is* NP*-hard even if $\mathcal{O} = \emptyset$.*

We note that ORP has two sources of NP-hardness, viz. the data part (as in the proof above) and the taxonomy, which under σ-repairs may derive further assertions. Furthermore, each ORP can be encountered in some DL-program setting; we show this on an example.

Example 30. Consider the ORP $\mathcal{R} = \langle \mathcal{O}, D_1, D_2 \rangle$, where $D_1 = \{\delta_1\}$, $D_2 = \{\delta_2\}$, such that $\delta_1 = \langle \{C(c), \neg D(c)\}, \neg E(c) \rangle$, and $\delta_2 = \langle \{D(d), \neg S(d)\}, C(d) \rangle$. We introduce predicates $p_C^{\delta_1}, p_D^{\delta_1}$ for δ_1 and $p_D^{\delta_2}, p_S^{\delta_2}$ for δ_2 and construct $\Pi = \langle \mathcal{O}, \mathcal{P}_I \cup \mathcal{P}_{DL} \rangle$, where

$$\mathcal{P}_I = \{p_C^{\delta_1}(c);\ p_D^{\delta_1}(c);\ p_D^{\delta_2}(d);\ p_S^{\delta_2}(d)\},$$

$$\mathcal{P}_{DL} = \left\{ \bot \leftarrow not\ \underbrace{\mathrm{DL}[C \uplus p_C^{\delta_1}, D \cup\!\!\!\!- p_D^{\delta_1}; \neg E](c)}_{a_1} \quad (1) \qquad \bot \leftarrow \underbrace{\mathrm{DL}[D \uplus p_D^{\delta_2}, S \cup\!\!\!\!- p_S^{\delta_2}; C](d)}_{a_2}\} \quad (2) \right\}.$$

Then Π has a single repair answer set candidate, in which a_1 must evaluate to true and a_2 to false. This gives rise to $\mathcal{R}$; the rule (1) effects the pair δ_1 in D_1 and the rule (2) the pair δ_2 in D_2. □

Generalizing the above example, for each $\mathcal{R} = \langle \mathcal{O}, D_1, D_2 \rangle$ one can construct a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, such that the solutions of $\mathcal{R}$ correspond to the repairs of Π as follows. A DL-atom a_i^j is created for every pair $\langle U_i^j, Q_i^j \rangle \in D_j$, such that the DL-query of a_i^j is Q_i^j, and the input signature λ_i^j encodes the update U_i^j: for every $C(\vec{e}) \in U_i^j$ (resp. $\neg C(\vec{e})$) the signature λ_i^j contains $C \uplus p_C^{i,j}$ (resp. $C \cup\!\!\!\!- p_C^{i,j}$). Furthermore, for each such update the fact $p_C^{i,j}(\vec{e})$ is added to $\mathcal{P}$. The rules of $\mathcal{P}$ ensure that all DL-atoms a_i^j are true for $j = 1$ and false for $j = 2$. That is, the logic program part $\mathcal{P}$ of Π contains

- a constraint $\bot \leftarrow not\ a_{i_1}^1$, for every $a_{i_1}^1$, and
- a constraint $\bot \leftarrow a_{i_2}^2$, for every $a_{i_2}^2$.

As there are no predicates in $\mathcal{P}$ apart from those occurring in facts, the only possible repair answer set I of Π contains all facts of $\mathcal{P}$. Therefore, the update $\lambda^I(a_i^j)$ of every a_i^j corresponds exactly to U_i^j, and the constraints of $\mathcal{P}$ guarantee the simultaneous entailment and non-entailment of sets of queries under possible temporary updates encoded by the given $\mathcal{R}$.

3.5. Tractable ORP cases

As Theorem 29 demonstrates, we obtain intractability results for ORP even if the ontology is empty. In what follows we aim at finding tractable cases for the ORP problem given that $\mathcal{O}$ is in DL *DL-Lite*$_{\mathcal{A}}$.

If there are few DL-atoms in the ground DL-program Π, then the ORP becomes tractable. However, in application settings Π is obtained by grounding a DL-program that has variables, which will lead to many DL-atoms in Π. Therefore, the pairs D_1 and D_2 are hard to control in practice, and to gain tractability for ORP, we consider restrictions on repairs and the ontology. We present four tractable cases of σ-repairs with independent selection function σ, which are arguably useful in practice. In what follows, let $\mathcal{R} = \langle \mathcal{O}, D_1, D_2 \rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$.

3.5.1. Bounded $\delta^{\pm}$-change

A natural restriction that one could exploit is to bound the distance from the original ABox, i.e.

$$\sigma_{\delta^{\pm},k}(S, \mathcal{A}) = \{\mathcal{A}' \mid |\mathcal{A}' \triangle \mathcal{A}| \leq k\}, \quad k \geq 0, \tag{4}$$

where $\mathcal{A}' \triangle \mathcal{A} = (\mathcal{A}' \backslash \mathcal{A}) \cup (\mathcal{A} \backslash \mathcal{A}')$ is the symmetric difference of sets. Our tractability result for this setting is as follows.

Proposition 31. *Deciding whether an ORP $\mathcal{R} = \langle \mathcal{O}, D_1, D_2 \rangle$ has a $\delta^{\pm}$, k-change repair, is feasible in polynomial time for fixed k.*

Proof. As the number m of possible ABox assertions is polynomial in the size of $\mathcal{T}$ and $\mathcal{A}$, traversing all $O(\binom{m}{k})$ possible $\mathcal{A}'$ and checking the repair condition can be done in polynomial time. □

We illustrate this repair type by the following example.

Example 32. For the DL-program from Fig. 1 the ABox $\mathcal{A}' = (\mathcal{A} \backslash \{Male(pat)\}) \cup \{Female(pat)\}$ is a possible $\delta^{\pm}$, k-change repair for $k = 2$. Another repair candidate is $\mathcal{A}' = (\mathcal{A} \backslash \{Male(pat)\}) \cup \{Male(mat)\}$ provided that *mat* is a constant from the ontology signature. □

The $\delta^{\pm}$-change repairs are arguably useful in practice. The repairs that restore consistency by getting rid of such deficiencies as typos and syntactical inaccuracies fall into this repair category. For instance, in Example 32 the fact $Male(pat)$ was in the ontology instead of $Male(mat)$, as the letters p and m were confused during the data engineering process. In such scenarios one can search for repairs by applying selective changes to certain ontology assertions. These selective changes include modifications of the predicate or constants occurring in the assertion, i.e. $P(\vec{t})$ could be changed to $P(\vec{t'})$ or $P'(\vec{t})$.

To ensure tractability, the number of constants or predicates with which the initial facts can be modified is bounded by n. Under this restriction, an ABox $\mathcal{A}$ with at most k assertions allowed for modification has $O(k^{2n})$ repair candidates; thus if both n and k are bounded by a constant, deciding whether a $\delta^{\pm}$-solution for ORP exists is polynomial. The alternatives (i.e. constants and predicates) used for fixing initial facts can be created by partitioning the elements of the ontology signature into subsets based on their syntactical similarity (measured by some string distance, cf. [24], such as Hamming or Levenshtein distance [57]). For example, the constants *mat* and *pat* differ just by a single letter and thus will be put to the same partition. This way one naturally limits the number of possibilities for changing a certain fact.

Swapping constants in role assertions is another special setting with obvious practical applications.

Example 33. $\mathcal{A}' = \mathcal{A} \backslash \{hasParent(john, pat)\} \cup \{hasParent(pat, john)\}$ would be a plausible repair for the DL-program Π from Fig. 1. □

3.5.2. Deletion repair

Another important restriction is to allow only to delete assertions from the original ABox i.e., use

$$\sigma_{del}(S, \mathcal{A}) = \{\mathcal{A}' \mid \mathcal{A}' \subseteq \mathcal{A}\}. \tag{5}$$

Example 34. For Π in Fig. 1, each $\mathcal{A}' \subset \mathcal{A}$ except for $\{Male(pat), hasParent(john, pat)\}$ is a deletion repair. □

Before formally stating the complexity results for this repair we establish the following lemma.

Lemma 35. *If $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent, then $\langle \mathcal{T}, \mathcal{A} \cup U_j^i \rangle \models Q_j^i$ iff $\langle \mathcal{T}, \mathcal{A}_0 \cup U_j^i \rangle \models Q_j^i$ for some $\mathcal{A}_0 \subseteq \mathcal{A}$ with $|\mathcal{A}_0| \leq 1$.*

To achieve tractability, we exclude non-containment ($\not\sqsubseteq$) DL-queries, i.e., of the form $\neg Q$ where Q is an inclusion or a disjointness axiom, from $\mathcal{P}$; let us call any ORP $\not\sqsubseteq$-*free*, if no DL-query of this form occurs in it. Under the reasonable (and necessary) assumption that the original ontology is consistent, we then obtain.

Theorem 36. *Deciding whether a $\not\sqsubseteq$-free ORP $\mathcal{R} = \langle \mathcal{O}, D_1, D_2 \rangle$ with consistent $\mathcal{O}$ has a σ_{del}-repair is feasible in polynomial time.*

If non-containment queries are allowed in DL-atoms, computing deletion repairs remains NP-hard.

Theorem 37. *Deciding whether an ORP $\mathcal{R} = \langle D_1, D_2, \mathcal{O}\rangle$ with a consistent $\mathcal{O}$ has some σ_{del} repair is* NP*-complete, and* NP*-hardness holds even if each $\langle U_j^2, Q_j^2\rangle \in D_2$ has $U_j^2 = \emptyset$ and either (i) each $\langle U_i^1, Q_i^1\rangle \in D_1$ has $U_i^1 = \emptyset$ (thus, $\mathcal{R}$ has only empty updates), or (ii) $\mathcal{T} = \emptyset$.*

3.5.3. Deletion δ^+2 repair

This selection combines deletion and small change in a prioritized way. First one deletes assertions from $\mathcal{A}$ (assumed to be consistent) according to some polynomial method μ (using domain knowledge etc.) until some $\mathcal{A}_0 = \mu(\mathcal{O}) \subseteq \mathcal{A}$ results that satisfies Definition 26 (iii). If $\mathcal{A}_0$ is a repair, it is the result; otherwise, one looks for a close repair with bounded δ^+ change. That is

$$\sigma_{del,\delta^+}(S, \mathcal{A}) = \begin{cases} \{\mu(\mathcal{O})\}, & \text{if } \mu(\mathcal{O}) \in S \\ \sigma_{\delta^+}(S, \mu(\mathcal{O})), & \text{if } \mu(\mathcal{O}) \notin S. \end{cases} \tag{6}$$

Example 38. If $\mu(\mathcal{O})$ drops unreliable information about the gender of certain persons in Example 1 (e.g. *pat*), $\mathcal{A}_0$={*Male*(*john*), *hasParent*(*john*, *pat*)} is a deletion repair. If the constraint

$$\perp \leftarrow \text{DL}[;\ hasParent](X, Y), \mathit{not}\ \text{DL}[;\ Male](Y), \mathit{not}\ \text{DL}[;\ Female](Y)$$

(the gender of parents must be known) would be in $\mathcal{P}$, then one would have to add *Female*(*pat*) to $\mathcal{A}_0$ to obtain a deletion-δ^+ repair. □

Then one can try all possible combinations of k assertions that can be added to the ABox $\mathcal{A}'$ such that along with condition (iii), also (ii) and (i) of the repair definition hold. Observe that $\mu(\mathcal{O})$ is selected by an independent selection function σ_{del}, which chooses subsets of $\mathcal{A}$. Furthermore, $\sigma_{\delta+}$ is applied to $\mu(\mathcal{O})$, the selection $\sigma_{\delta+}$ chooses an ABox $\mathcal{A}' \supseteq \mu(\mathcal{O})$, such that $\mathcal{A}' \setminus \mu(\mathcal{O})$ contains not more then k assertions. The selection $\sigma_{\delta+}$ is independent by Proposition 24, as it is a composition of σ_{del} and $\sigma_{\delta+}$ both of which are independent. As both σ_{del} and $\sigma_{\delta+}$ are realizable in polynomial time, the overall problem is tractable.

3.5.4. Addition under bounded opposite polarity

Repairs by unbounded additions become tractable, if few of them are positive resp. negative, i.e., the number of assertions with opposite polarity is bounded (which by Theorem 29 is necessary). That is, if $\mathcal{A}^+$ (resp., $\mathcal{A}^-$) is the positive (negative) part of an ABox $\mathcal{A}$, then

$$\sigma_{bop,k}(S, \mathcal{A}) = \{\mathcal{A}' \supseteq \mathcal{A} \mid |\mathcal{A}'^{+} \setminus \mathcal{A}| \leq k \text{ or } |\mathcal{A}'^{-} \setminus \mathcal{A}| \leq k\}, \quad k \geq 0. \tag{7}$$

The following result is instrumental.

Theorem 39. *For a $\not\sqsubseteq$-free ORP $\mathcal{R} = \langle \mathcal{O}, D_1, D_2\rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$ and $\mathcal{T}$ has no disjointness axioms*[5] *deciding whether some σ_{bop}-repair exists is polynomial.*

3.5.5. Applicability of independent selections

Like for relational databases, our tractable cases fit real applications, e.g. in case of deletion repairs (observing that non-subsumption queries are insignificant for practical DL-programs) and scenarios akin to key-constraint violations in databases. Restoring consistency by removing conflicting pieces of data is a common approach in data management.

Composability of independent selections adds to their applicability. Moreover, they may be combined with DB-style factorization and localization techniques (see [10] and references therein) and with local search to compute closest repairs.

Bounding the number of changes, especially additions, is also compliant with practice, where too many potential repairs suggest human intervention (cf. [10]). Finally, one may increase the bound in iterative deepening (assuming that not many changes are needed).

3.6. Domain-based restrictions on repairs

In previous sections we have proposed several technical means for treating inconsistencies in DL-programs. We have presented some repair forms that are practically usable and computationally effective, but until now no domain knowledge has been incorporated into the DL-program repair process. It is natural, however, to believe that the end users of DL-programs will wish to contribute to the repair by sharing their subject expertise.

Qualitative and domain-dependent aspects of repairs are of crucial importance for their practicability. These qualitative aspects formulated in terms of additional local restrictions put on repairs help to effectively filter out the irrelevant repair

[5] Disregarding axioms $F_\neg \sqsubseteq \neg F$ to compile negative assertions.

candidates. For example, availability of meta information about the trustfulness of certain ontology pieces may allow to adjust the repair process.

Example 40. Being aware of the *unreliability* of ontology facts about the individual *john* in Example 1 motivates one to consider the repair $\mathcal{A}' = \mathcal{A} \backslash \{hasParent(john, pat)\}$ for the DL-program Π in the first instance.

Knowing additionally that the set of *Adopted* children is very likely to be incomplete naturally adds $\mathcal{A}'' = \mathcal{A} \cup \{Adopted(john)\}$ to the set of repair possibilities. $\square$

Similarly the user might be willing to keep some information bits in the ontology unchanged.

Example 41. If in Example 40 one wants to avoid dropping the data about individuals belonging to the concept *Child* but not known to be *Adopted*; then the repair $\mathcal{A}'$ is no longer among the preferred options. $\square$

The guidelines on the operations that are allowed to be applied to the ontology could clearly influence the repair process further.

Example 42. If in Example 40 additions to the ontology are strongly prohibited, then the repair $\mathcal{A}''$ is automatically dropped from the set of leading candidates. $\square$

In some scenarios various dependencies among the data parts stored in the ontology might influence the repair process. Deletion (resp. addition) of a certain fact might force further ontology changes to be incorporated.

Example 43. Consider a variant of Example 1, in which each *Adopted* child stored in the ontology is desired to have a certain identification number (*ID*) assigned to it through the predicate *hasID*. This additional constraint could be expressed by the TBox axiom $Adopted \sqsubseteq \exists hasID$. However, this restriction might not be a formal requirement, but rather a wish of the user, for whom it is more convenient to track adopted children by their IDs. Thus the TBox axiom might not be in the ontology explicitly. In such a setting the repair $\mathcal{A}''$ from Example 40 in which information about *john*'s adoption is added, is not among the best repair candidates any longer, as together with this new information, the additional knowledge about the *ID* of *john* should be available.

Similarly, if not only adopted children, but all persons are required to have an ID, and the latter is indeed given in the original ontology, the repair $\mathcal{A}' = \mathcal{A} \backslash \{Male(pat)\} \cup \{Male(mat)\}$ from Example 32 forces one to delete the *ID* of *pat* and add the ID of *mat*; in case the latter is not known, the repair $\mathcal{A}'$ becomes undesired. $\square$

Integration of domain restrictions into the repair computation process. The wide spectrum of potential restrictions that could be applied to the repair candidates motivates one to consider various possible ways of integrating additional domain knowledge into the repair computation process. Three global modes of repairing inconsistent DL-programs seem reasonable in this context:

1) The first mode suggests the computation of repair candidates with some σ-selection function, followed by a post-filtering of the candidates taking into account the domain knowledge. If some of the protected ontology elements are no longer present in the repair candidate, and their reintroduction violates the repair conditions, then one proceeds with the analysis of a next repair candidate. Otherwise, the desired repair is computed, and the computation process terminates.
2) The second mode assumes that the domain knowledge is encoded in the selection function and consequently all identified repairs *a priori* satisfy the introduced domain-based requirements.

   Example 44. Suppose we want to compute the $\delta^{\pm}$ repairs with the desired property expressed in Example 43, i.e. in the repairs for all *Adopted* children their *ID* should be known. Then our problem amounts to the problem of computing $\delta^{\pm}$ repairs of the original DL-program extended by the following rules that conveniently encode the additional requirement:

   $$(1)\quad assigned(X) \leftarrow \mathrm{DL}[;\, Adopted](X),\ \mathrm{DL}[;\, ID](Y),\ \mathrm{DL}[;\, hasID](X,Y);$$
   $$(2)\quad \bot \leftarrow \mathrm{DL}[;\, Adopted](X),\ not\ assigned(X).$$

   The repairs of the extended program correspond to the repairs of the original program post-filtered by the respective domain-specific condition. $\square$

3) The third mode is the combination of the first two, where some domain conditions are incorporated into the repair search process, but further post-filtering conditions can be checked.

4) The mode (2) can be extended to support prioritized repair computation. That is, first one aims at finding the best repairs that fully satisfy the domain specific requirements, and then if such search does not bring any results, the requirements are weakened accordingly or even dropped altogether.

Example 45. Recall the setting from Example 44. We first aim at repairs such that *IDs* of all adopted children are known. Once some repair answer set of Π with rules (1) and (2) is found, the computation terminates and the result is output. If no such I was identified, then one might be willing to relax the repair condition by allowing at most k adopted children to lack *IDs*. For that the constraint (2) can be changed to a rule (2′) having $not\_assigned(X)$ in the head. Repair answer sets I of the resulting program with at most k ground predicates over *not_assigned* will satisfy the above requirement, and consequently any repair $\mathcal{A}' \in rep^{I}_{\sigma,x}(\Pi)$ is guaranteed to be preferred, where σ is a $\delta^{\pm}$-change selection function. $\square$

All of the discussed domain-specific repair preferences can be combined and ordered in various ways. The techniques for their computation heavily depend on the application scenario, and in different concrete settings could be adapted and extended.

4. Computation

In this section, we first recall the essentials of the evaluation algorithm for DL-programs as a special class of so-called HEX-programs as in [30], and we then provide a naive and an optimized extension of that algorithm for computing repairs.

4.1. DL-program evaluation

The evaluation of a DL-program Π builds on a program rewriting $\hat{\Pi}$, where DL-atoms a are replaced by ordinary atoms (called *replacement atoms*) e_a, and a guess on the truth value of the latter by 'choice' rules $e_a \vee ne_a$ is added.

Example 46. Consider the following grounding of some rules from Fig. 1:

$$\mathcal{P}' = \left\{ \begin{array}{l} (7)\, ischildof(john, alex); \quad (8)\, boy(john); \\ (9)\, hasfather(john, pat) \leftarrow \mathrm{DL}[Male \uplus boy;\ Male](pat), \mathrm{DL}[;\ hasParent](john, pat); \\ (10)\, \bot \leftarrow not\ \mathrm{DL}[;\ Adopted](john), hasfather(john, pat), \\ \qquad\qquad ischildof(john, alex), not\ \mathrm{DL}[Child \uplus boy;\ \neg Male](alex) \end{array} \right\}$$

The replacement program $\hat{\Pi}'$ for $\Pi' = \langle \mathcal{O}, \mathcal{P}' \rangle$ comprises the following rules:

$$\hat{\Pi}' = \left\{ \begin{array}{ll} (7)\, ischildof(john, alex); & (11)\, e_{a_1(pat)} \vee ne_{a_1(pat)}; \\ (8)\, boy(john) & (12)\, e_{a_2(john,pat)} \vee ne_{a_2(john,pat)}; \\ (9)\, hasfather(john, pat) \leftarrow e_{a_1(pat)}, e_{a_2(john,pat)}; & (13)\, e_{a_3(john)} \vee ne_{a_3(john)}; \\ (10)\, \bot \leftarrow not\ e_{a_3(john)}, hasfather(john, pat), & (14)\, e_{a_4(alex)} \vee ne_{a_4(alex)} \\ \qquad\qquad ischildof(john, alex), not\ e_{a_4(alex)}; & \end{array} \right\}$$

where

$$\begin{array}{ll} a_1 = \mathrm{DL}[Male \uplus boy;\ Male](pat), & a_2 = \mathrm{DL}[;\ hasParent](john, pat), \\ a_3 = \mathrm{DL}[;\ Adopted](john), & a_4 = \mathrm{DL}[Child \uplus boy;\ \neg Male](alex). \end{array} \quad \square$$

Given an interpretation $\hat{I}$ of the replacement program $\hat{\Pi}$, we use $I|_{\Pi}$ to denote its restriction to the original language of Π. A crucial notion is that of *compatible set*.

Definition 47 *(Compatible set).* A *compatible set* of a (ground) DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is an interpretation $\hat{I}$, such that (i) $\hat{I}$ is an answer set of $\hat{\Pi}$, and (ii) $e_a \in \hat{I}$ iff $I|_{\Pi} \models^{\mathcal{O}} a$, for every $a = \mathrm{DL}[\lambda;\ Q](c)$ occurring in Π.

Example 48. Consider an interpretation $\hat{I} = \{ischildof(john, alex), boy(john), hasfather(john, pat), e_{a_1}, e_{a_2}, e_{a_3}, ne_{a_4}\}$ of $\hat{\Pi}'$ from Example 46. This interpretation is not compatible for $\Pi' = \langle \mathcal{O}, \mathcal{P}' \rangle$, since $e_{a_3(john)} \in \hat{I}$, but it holds that $I \not\models^{\mathcal{O}}$ DL[; *Adopted*](*john*), and thus (ii) of Definition 47 is not satisfied. However, the interpretation $\hat{I}$ is a compatible set for $\Pi'' = \langle \mathcal{O}', \mathcal{P}' \rangle$ where $\mathcal{O}' = \mathcal{O} \cup \{Adopted(john)\}$. Furthermore, the restriction of $\hat{I}$ to the language of Π'' is $\hat{I}|_{\Pi''} = \{ischildof(john, alex), hasfather(john, pat), boy(john)\}$.

Conversely, given an interpretation I of Π, we denote by I_c the interpretation of $\hat{\Pi}$ such that I_c coincides with I on normal atoms, and each replacement atom e_a is in I_c (i.e. true) iff $I \models^{\mathcal{O}} a$ for the respective DL-atom a.

With these concepts in place, we are ready to describe the basic algorithm (cf. Algorithm 1) for evaluating a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ adopted from [30]. First, $\hat{\Pi}$ is evaluated by an ordinary ASP solver; for every answer set $\hat{I}$ of $\hat{\Pi}$ in (b), the function *CMP* checks for compatibility, while *xFND* tests foundedness, i.e., whether $\hat{I}|_{\Pi}$ is a $\subseteq$-minimal model of the reduct $\mathcal{P}_x^{I|_{\Pi},\mathcal{O}}$. In case of $x = \mathit{weak}$, *xFND* just returns true, otherwise ($x = \mathit{flp}$) it checks for disjointness with unfounded sets as defined in [30]. If both tests succeed, then $\hat{I}|_{\Pi}$ is output as an answer set.

Algorithm 1: *AnsSet*: Compute $AS_x(\Pi)$.

```
    Input: A DL-program Π, x ∈ {weak, flp}
    Output: AS_x(Π)
(a) for Î ∈ AS(Π̂) do
(b)     if CMP(Î, Π) ∧ xFND(Î, Π) then
            output Î|_Π
        end
    end
```

Example 49. Suppose we are interested in computing *flp*-answer sets of Π'' from Example 48. In (a) among $AS(\hat{\Pi}'')$ the interpretation $\hat{I} = \{ischild(john, alex), boy(john), hasfather(john, pat), e_{a_1(pat)}, e_{a_3(john)}, ne_{a_4(alex)}, e_{a_2(john,pat)}\}$ is identified. Both the compatibility and the foundedness check in (b) for $\hat{I}$ succeed, and thus $\hat{I}|_{\Pi''}$ is output as an *flp*-answer set of Π''.

An important link between the answer sets of Π and $\hat{\Pi}$ is the following property.

Proposition 50. *If* $I \in AS_x(\Pi)$ *then* $I_c \in AS_x(\hat{\Pi})$.

While *AnsSet* is clearly sound, from this result its completeness follows, i.e. restricting the search to $AS_x(\hat{\Pi})$ does not yield any loss of answer sets.

4.2. Naive algorithm for repair computation

We next present a naive algorithm for computing deletion repair answer sets which extends the above DL-program evaluation algorithm. First we aim at a procedure for computing (σ, x)-repairs given an independent selection function σ. Then, we describe how its main subroutine can be used for an extension of *AnsSet* that computes answer sets if they exist, and (σ, x)-repair answer sets otherwise.

A first key observation is that Proposition 50 generalizes to repair answer sets. More precisely:

Proposition 51. *If* $I \in RAS_x(\Pi)$ *then* $I_c \in AS(\hat{\Pi})$.

Proof. By definition of $RAS_x(\Pi)$, we get that $I \in AS(\Pi')$, where $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$, $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ and $\mathcal{A}' \in rep_x(\Pi)$. Since by Proposition 50 $I_c \in AS(\hat{\Pi}')$ and $\hat{\Pi} = \hat{\Pi}'$, the result immediately follows. □

Thus, our approach is to traverse $AS(\hat{\Pi})$ and check for each $\hat{I} \in AS(\hat{\Pi})$ whether $\hat{I}|_{\Pi}$ is a (σ, x)-repair answer set of Π. The latter proceeds in two steps, where the first step is to search for potential σ-repairs of the ontology such that Definition 47 (ii) holds for $\hat{I}$, that is to find solutions of the corresponding ontology repair problem.

Algorithm 2: *RepAns*: Compute (σ, x)-repairs $rep^{\hat{I}|_{\Pi}}_{(\sigma,x)}(\Pi)$ of Π for $x \in \{\mathit{weak}, \mathit{flp}\}$.

```
    Input: Π = ⟨O, P⟩, O = ⟨T, A⟩, Î∈AS(Π̂), σ
    Output: rep^{Î|_Π}_{(σ,x)}(Π)
(a) for A' ∈ ORP(Î, Π, σ) do
(b)     if CMP(Î, ⟨T, A', P⟩) ∧ xFND(Î, ⟨T, A', P⟩) then
            output A'
        end
    end
```

The procedure *RepAns* (cf. Algorithm 2) calls the subroutine $ORP(\hat{I}, \Pi, \sigma)$ in (a) to compute σ-repairs $\mathcal{A}'$ of the corresponding ORP, constructed from the DL-atoms with their guessed values and the ontology. Further on, *RepAns* re-uses the functions *CMP* and *xFND* in (b) to check whether $\hat{I}$ is a compatible set of Π' and that it is founded w.r.t. $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$, $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$. It thus computes the set of all ABoxes under which $\hat{I}$ becomes a (σ, x)-repair answer set. We demonstrate *RepAns* on an example.

Algorithm 3: *RepAnsSet*: Compute a set $RAS_{(\sigma,x)}(\Pi)$ of (σ,x)-repair AS of Π for $x\in\{weak, flp\}$.

```
Input: Π = ⟨O, P⟩, O = ⟨T, A⟩, σ
Output: I ∈ RAS_(σ,x)(Π)
for Î ∈ AS(Π̂) do
    if RepAns(Π, O, Î, σ) ≠ ∅ then
        output Î|Π
    end
end
```

Example 52. Suppose that *RepAns* gets as input Π', $\hat{I}$ from Examples 46 and 48, and $\sigma_{\delta\pm,1}$ selection function computing $\delta\pm$ repairs. The corresponding ORP $\mathcal{R}$ is given by $\mathcal{R} = \langle \mathcal{O}, D_1, D_2\rangle$, where $D_1 = \{\langle\{Male(john)\}, Male(pat)\rangle, \langle\emptyset, hasParent(john, pat)\rangle, \langle\emptyset, Adopted\rangle\}$ and $D_2 = \{\langle\emptyset, \neg Male(alex)\rangle\}$. The ABox $\mathcal{A}' = \{Male(john), Male(pat), hasParent(john, pat), Adopted(john)\}$ is computed in (a) as the $\sigma_{\delta\pm}$-repair for $\mathcal{R}$. The checks in (b) succeed for the ABox $\mathcal{A}'$, and it is output to the user.

Example 53. Let $\Pi = \langle \mathcal{O}, \mathcal{P}\rangle$ be a DL-program, where

$$\mathcal{O} = \left\{ \begin{array}{lll} A \sqsubseteq \neg C; & A(c); & \neg E(c); \\ A \sqsubseteq D; & D(c); & C(c) \end{array} \right\} \quad \mathcal{P} = \left\{ \begin{array}{l} p(c);\;\; r(c);\;\; q(c) \leftarrow \mathrm{DL}[C \uplus r;\; D](c); \\ \bot \leftarrow \mathrm{DL}[D \uplus p, E \uplus r;\; \neg C](c) \end{array} \right\}.$$

We denote by a_1 and a_2 the DL-atoms $\mathrm{DL}[C \uplus r;\; D](c)$ and $\mathrm{DL}[D \uplus p, E \uplus r;\; \neg C](c)$ respectively. Consider the interpretation $\hat{I} = \{p(c), r(c), q(c), e_{a_1}, ne_{a_2}\}$, in which a_1 is guessed true and a_2 guessed false. The corresponding ORP is given by $\mathcal{R} = \langle \mathcal{O}, D_1, D_2\rangle$, where $D_1 = \{\langle\{\neg C(c)\}; D(c)\rangle\}$ and $D_2 = \{\langle\{D(c), \neg E(c)\}; \neg C(c)\rangle\}$. Let σ select the deletion repairs, than we get $\mathcal{A}' = \{D(c), C(c)\}$ as a possible output of the procedure $ORP(\hat{I}, \Pi, \sigma)$, for which the compatibility check verified by the call $CMP(\hat{I}, \langle \mathcal{T}, \mathcal{A}', \mathcal{P}\rangle)$ is passed. If we are interested in $(\sigma, weak)$ repairs then $\mathcal{A}'$ is output by the algorithm *RepAns*. Yet another foundedness test is needed to check whether $\mathcal{A}'$ is a (σ, flp) repair. This test is done in $flp\text{-}FND(\hat{I}, \langle \mathcal{T}, \mathcal{A}', \mathcal{P}\rangle)$, which checks whether I is a minimal model of the *flp*-reduct $\mathcal{P}_{flp}^{I,\mathcal{O}'} = \{p(c);\;\; q(c);\;\; q(c) \leftarrow \mathrm{DL}[C \uplus r;\; D](c)\}$. As the latter test succeeds, the ABox $\mathcal{A}'$ is an *flp*repair and thus it is in the output of $RepAns(\Pi, \hat{I}, \sigma_{del})$. $\square$

Let now *RepAnsSet* (Algorithm 3) be the algorithm that iteratively calls *RepAns* for every $\hat{I} \in AS(\hat{\Pi})$, and outputs any $\hat{I}$, where the result of *RepAns* is nonempty, i.e. some repair $\mathcal{A}'$ was computed. We then have:

Theorem 54. *RepAns and RepAnsSet are both sound and complete for $rep_{(\sigma,x)}(\Pi)$ and $RAS_{(\sigma,x)}(\Pi)$, respectively, for every independent selection function σ.*

A natural question is whether computing repair answer sets via compatible sets $\hat{I}$ of Π makes repair answer set checking for $\hat{I}|_\Pi$ easier than for arbitrary interpretations I. Unfortunately, this is not the case; we thus obtain a strengthening of the results of Theorem 18.

Theorem 55. *For ground $\Pi = \langle \mathcal{O}, \mathcal{P}\rangle$ and $I \subseteq HB_\Pi$, deciding whether $I \in RAS_x(\Pi)$ is (i) NP-complete for $x = weak$ and (ii) Σ_2^p-complete for $x = flp$; hardness holds even if $I = \hat{I}|_\Pi$ for an answer set $\hat{I} \in \mathrm{AS}(\hat{\Pi})$ (and, moreover, I is unique).*

Intuitively, even if we know $\hat{I}$, we still need to guess a repair $\mathcal{A}'$ that witnesses $\hat{I}|_\Pi$. The verification of the guess involves a foundedness test, which is co-NP-hard in case of $x = flp$; this results in Σ_2^p-completeness.

Note that, for illustration, we kept the algorithms simple; several optimizations apply, some of which we discuss below. For instance, to compute just some (σ, x)-repair answer set, we can modify *RepAns* to a version that merely computes a first witnessing ABox $\mathcal{A}'$. Moreover, caching ABoxes $\mathcal{A}'$ and/or all answer sets of the respective Π' (which can be straight output as (σ, x)-repair answer sets of Π) further reduces the search space.

4.3. Support sets

The algorithms *RepAns* and *RepAnsSet* represent natural realizations of repair computation. However, they turn out as too naive and do not scale for practical applications; each ORP derived from an answer set $\hat{I}$ of the replacement program $\hat{\Pi}$ is solved from scratch, as no information about past ORPs is exploited.

We thus develop an alternative approach for computing repair answer sets based on the notion of *support set*. Intuitively, a support set for a DL-atom $d = \mathrm{DL}[\lambda;\; Q](\vec{t})$ is a portion of its input that, together with ABox assertions, is sufficient to conclude that the query $Q(\vec{t})$ evaluates to true; i.e., 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 $I \models^{\mathcal{O}} Q(\vec{t})$. Basically, our method precomputes support sets for each DL-atom at a nonground level. During DL-program evaluation, for each candidate interpretation the ground instantiations of the support sets are effectively obtained. The latter help to prune the answer set search space and also allow one to solve ORPs by constraint matching.

Exploiting Lemma 13 we define support sets using only ontology predicates as follows:

Definition 56 *(Ground support sets).* Given a ground DL-atom $d = \mathrm{DL}[\lambda;\ Q](\vec{t})$, a set S of assertions from $\mathcal{A} \cup \mathcal{A}_d$ is a *support set* for d w.r.t. an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, if $\mathcal{T}_d \cup S \models Q(\vec{t})$. By $Supp_{\mathcal{O}}(d)$ we denote the set of all support sets S for d w.r.t. $\mathcal{O}$.

Support sets can be grouped into families of support sets or simply *support families*. More formally,

Definition 57 *(Support family).* Any collection $\mathcal{S} \subseteq Supp_{\mathcal{O}}(d)$ of support sets for a DL-atom d w.r.t. an ontology $\mathcal{O}$ is a *support family* of d w.r.t. $\mathcal{O}$.

Clearly, support sets as defined above may be subsumed by other support sets (e.g., $\{A(c), R(c,d)\}$ by $\{A(c)\}$) and removed. We concentrate on $\subseteq$-minimal support sets S for a DL-atom d, i.e. for every $S' \subset S$ it holds that $S' \notin Supp_{\mathcal{O}}(d)$. In general even $\subseteq$-minimal support sets can be arbitrarily large and there can be infinitely many (exponentially many for acyclic $\mathcal{T}$) support sets. However, fortunately it turns out that for *DL-Lite*$_{\mathcal{A}}$ support sets are of a particular structure. In view of the property that in *DL-Lite*$_{\mathcal{A}}$ a single assertion is sufficient to derive a query [21] from a consistent ontology, we obtain that for *DL-Lite*$_{\mathcal{A}}$ support sets are at most of size 2. More formally,

Proposition 58. *Every $\subseteq$-minimal support set S for a DL-atom $d = \mathrm{DL}[\lambda;\ Q](\vec{t})$ w.r.t. an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ in DL-Lite$_{\mathcal{A}}$ has either the form (i) $S = \{P(\vec{c})\}$, such that $\mathcal{T}_d \cup S \models Q(\vec{t})$, or (ii) $S = \{P(\vec{c}), P'(\vec{d})\}$ such that $\mathcal{T}_d \cup S$ is inconsistent.*

Support sets are linked to interpretations by the following notion.

Definition 59 *(Coherence).* A support set S of a DL-atom d is *coherent with an interpretation I*, if for each $P_p(\vec{c}) \in S$ it holds that $p(c) \in I$.

We illustrate the notion of coherence by the following example.

Example 60. The set $\{hasParent(john, pat)\}$ is a support set for the DL-atom 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, $\{Male_{boy}(pat)\}$ 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 61. *Let d be a ground DL-atom, let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ 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 62. *Given a ground DL-atom d and an ontology $\mathcal{O}$, some interpretation I exists such that $I \models^{\mathcal{O}} d$ iff $Supp_{\mathcal{O}}(d) \neq \emptyset$.*

Apart from the maximal number of assertions that participate in support sets for DL-atoms accessing *DL-Lite*$_{\mathcal{A}}$ ontologies, there is also a limit on the number of constants that can occur in such support sets. In fact, in Definition 58 $\vec{c} \cup \vec{d}$ can involve at most 3 constants, which is formally stated in the following proposition.

Proposition 63. *Let S be a $\subseteq$-minimal support set of a ground DL-atom d w.r.t. a DL-Lite$_{\mathcal{A}}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. Then S involves at most 3 constants.*

When working with support sets for DL-atoms that access an *DL-Lite*$_{\mathcal{A}}$ ontology, we can exploit the above proposition and limit ourselves only to support sets of size 2 involving at most 3 constants.

4.4. 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 at 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 64 *(Nonground support sets).* Let $\mathcal{T}$ be a TBox, and let $d(\vec{X}) = \mathrm{DL}[\lambda;\ Q\,](\vec{X})$ be a nonground DL-atom. Suppose that V is a set of distinct variables such that $\vec{X} \subseteq V$, and that $\mathcal{C}$ is a set of constants. A *nonground support set* for d w.r.t. $\mathcal{T}$ is a set $S = \{P_1(\vec{Y}_1), \dots, P_k(\vec{Y}_k)\}$ such that

(i) $\vec{Y}_1, \dots, \vec{Y}_k \subseteq V$ and
(ii) for each substitution $\theta : V \to \mathcal{C}$, the instance $S\theta = \{P_1(\vec{Y}_1\theta), \dots, P_k(\vec{Y}_k\theta)\}$ is a support set for $d(\vec{X}\theta)$ w.r.t. $\mathcal{O}_{\mathcal{C}} = \langle \mathcal{T}, \mathcal{A}_{\mathcal{C}} \rangle$, where $\mathcal{A}_{\mathcal{C}}$ is the set of all possible ABox assertions over $\mathcal{C}$.

By $\mathbf{Supp}_{\mathcal{O}}(d)$ we denote the set of all nonground support sets for d.

Here $\mathcal{A}_{\mathcal{C}}$ takes care of any possible ABox, by considering the maximal ABox (since $\mathcal{O} \subseteq \mathcal{O}'$ implies that $Supp_{\mathcal{O}}(d) \subseteq Supp_{\mathcal{O}'}(d)$). Now generalizing Propositions 58 and 63 we obtain the following characterization for nonground support sets accessing *DL-Lite*$_{\mathcal{A}}$ ontologies:

Proposition 65. *Every $\subseteq$-minimal nonground support set S for a DL-atom d w.r.t. an ontology $\mathcal{O}$ in DL-Lite$_{\mathcal{A}}$ has either the form (i) $S = \{P(\vec{Y})\}$ or (ii) $S = \{P(\vec{Y}), P'(\vec{Y}')\}$, where $\vec{Y} \cup \vec{Y}'$ contains at most 3 distinct variables.*

Example 66 *(cont'd).* Certainly $\{hasParent(X,Y)\}$ is a nonground support set for $\mathrm{DL}[;\ hasParent](X,Y)$, and so are $\{Male(X)\}$ and $\{Male_{boy}(X)\}$ for the DL-atom $d(X) = \mathrm{DL}[Male \uplus boy;\ Male](X)$, but $d(X)$ has also $\{Male_{boy}(Y), Female(Y)\}$ as a nonground support set.

Nonground support sets S for *DL-Lite*$_{\mathcal{A}}$ are sound in the sense that each instance $S\theta$ matching with $\mathcal{A} \cup \mathcal{A}_d$ is a support set of the ground DL-atom $d\theta$ w.r.t. $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. They are also complete, i.e., every support set S of a ground DL-atom d w.r.t. $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ results as such an instance, and thus can be determined by syntactic matching.

If a sufficient portion of support sets is precomputed, then the ontology access can be fully avoided. We call such a portion a *complete* support family.

Definition 67 *(Completeness).* A family $\mathbf{S} \subseteq \mathbf{Supp}_{\mathcal{O}}(d)$ of nonground support sets for a (non-ground) DL-atom $d(\vec{X})$ w.r.t. a *DL-Lite*$_{\mathcal{A}}$ ontology $\mathcal{O}$ is *complete*, if for every θ: $\vec{X} \to \mathcal{C}$ and $S \in Supp_{\mathcal{O}}(d(\vec{X}\theta))$, some $S' \in \mathbf{S}$ exists such that $S = S'\theta'$, for some extension $\theta' : V \to \mathcal{C}$ of θ to V, where V is a set of distinct variables, such that $\vec{X} \subseteq V$.

Example 68. Consider the DL-atom $d = \mathrm{DL}[Male \uplus boy;\ Male](X)$ from Fig. 1. For computing a complete family $\mathbf{S}$ of nonground support sets for d w.r.t.$\mathcal{O}$, we may refer to $\mathcal{T}_d = \mathcal{T} \cup \{Male_{boy} \sqsubseteq Male\}$. The support family $\mathbf{S} = \{S_1, S_2, S_3, S_4\}$ is complete for d, where $S_1 = \{Male(X)\}$, $S_2 = \{Male_{boy}(X)\}$, $S_3 = \{Male_{boy}(Y), \neg Male(Y)\}$, $S_4 = \{Male_{boy}(Y), Female(Y)\}$. $\square$

4.5. Determining nonground support sets

Our technique for computing the nonground support sets for DL-atoms over *DL-Lite*$_{\mathcal{A}}$ ontologies is based on TBox classification, which is an important problem in Description Logics [4]: 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. For example, [56] studies it for the OWL 2 QL profile and [51] discusses it for $\mathcal{EL}$. Respective algorithms are thus suitable and also easily adapted for the computation of (a complete family of) nonground support sets for a DL-atom $d(\vec{X})$ w.r.t. an ontology $\mathcal{O}$ in *DL-Lite*$_{\mathcal{A}}$.

In principle, one can exploit Proposition 13 and resort to $\mathcal{T}_d$, i.e., compute the classification $Clf(\mathcal{T}_d)$, and determine nonground support sets of $d(\vec{X})$ minimal conflict sets [74]. To determine inconsistent support sets, perfect rewriting [21] 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 can nonetheless be worthwhile to compute $Clf(\mathcal{T})$ first, as it is reusable for all DL-atoms. The additional axioms in $\mathcal{T}_d$, i.e., those of form $P_p \sqsubseteq (\neg)P$ (induced by update operators), are handled when determining the nonground support sets for a particular DL-atom from $Clf(\mathcal{T})$.

Example 69. Consider the DL-atom $d = \mathrm{DL}[Male \uplus boy; Male](X)$ from Example 1. For computing a complete family $\mathbf{S}$ of nonground support sets for d w.r.t. $\mathcal{O}$, we may refer to $\mathcal{T}_d = \mathcal{T} \cup \{Male_{boy} \sqsubseteq Male\}$ and its classification $Clf(\mathcal{T}_d)$. Here, $S_1 = \{Male(X)\}$ and $S_2 = \{Male_{boy}(X)\}$ are the only unary nonground support sets of d. Further nonground support sets are obtained by computing minimal conflict sets, yielding $\{P(\vec{Y}), \neg P(\vec{Y})\}$ for each $P \in \mathbf{C} \cup \mathbf{R}$, as well as $S_3 = \{Male_{boy}(Y), \neg Male(Y)\}$, $S_4 = \{Male_{boy}(Y), Female(Y)\}$, and $S_5 = \{Male(Y), Female(Y)\}$. 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 $\mathcal{A}$). Hence, $\mathbf{S} = \{S_1, S_2, S_3, S_4\}$ is a complete support family for d w.r.t. $\mathcal{O}$. □

Algorithm 4: *SupRAnsSet*: all deletion repair answer sets.

```
    Input: Π=⟨T, A, P⟩
    Output: flpRAS(Π)
(a) compute a complete set S of nongr. supp. sets for the DL-atoms in Π
(b) for Î ∈ AS(Π̂) do
        D_p ← {d | e_d ∈ Î}; D_n ∈ {d | ne_d ∈ Î}; S^Î_gr ← Gr(S, Î, A);
(c)     if S^Î_gr(d) ≠ ∅ for d ∈ D_p and every S ∈ S^Î_gr(d) for d ∈ D_n fulfills S ∩ A ≠ ∅ then
(d)         for all d ∈ D_p do
(e)             if some S ∈ S^Î_gr(d) exists s.t. S ∩ A = ∅ then pick next d
                else remove each S from S^Î_gr(d) s.t. S ∩ A ∩ ⋃_{d'∈D_n} S^Î_gr(d') ≠ ∅
(f)             if S^Î_gr(d) = ∅ then pick next Î
            end
(g)         A' ← A \ ⋃_{d'∈D_n} S^Î_gr(d');
(h)         if flpFND(Î, ⟨T, A', P⟩) then output Î|_Π
        end
    end
```

4.6. Optimized algorithm for repair computation

We are now ready to describe our optimized algorithm *SupRAnsSet* (see Algorithm 4), which avoids 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 $\mathcal{A} \cup \mathcal{A}_d$. 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 $d' \in D_n$ (hence α needs to be deleted; note that due to consistency of $\mathcal{A}$, even inconsistent support of d' 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 $d' \in D_n$. An eventual check (h) for foundedness (minimality) w.r.t. $\mathcal{A}'$ determines whether a deletion repair answer set has been found.

Example 70. Consider the DL-atoms $a = \mathrm{DL}[; hasParent](john, pat)$ and $b = \mathrm{DL}[Male \uplus boy; Male](pat)$ from Example 1, and assume that $\{e_a, ne_b\} \subseteq \hat{I}$. Then, we get $\mathbf{S}^{\hat{I}}_{gr}(a) = \{\{hasParent(john, pat)\}\}$, and we reach 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 we show, 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 71. *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 $\Pi = \langle \mathcal{T}, \mathcal{A}, \mathcal{P} \rangle$. If $\hat{I}$ is a compatible set for $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$ where $\mathcal{A}' \supseteq \mathcal{A}$, then I is an answer set for $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$.*

Armed with this result, we establish the correctness result.

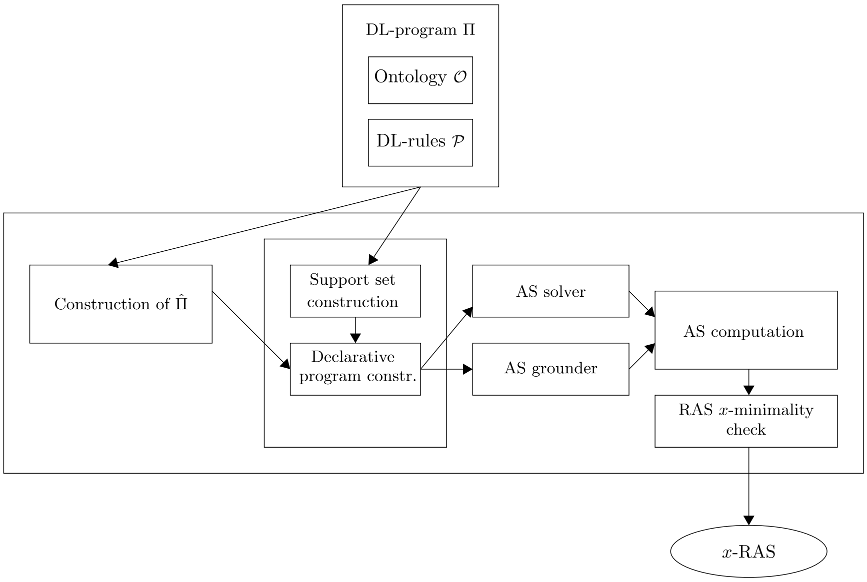

Fig. 2. System architecture of the dlliteplugin for Repair Answer Set Computation.

Theorem 72. *SupRAnsSet is sound and complete w.r.t. computing deletion repair answer sets, i.e., given a DL-program* $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ *with DL-Lite*$_{\mathcal{A}}$ *ontology* $\mathcal{O}$*, SupRAnsSet*(Π) *correctly outputs all deletion repair answer sets of* Π*.*

In the next section, we turn to an implementation of Algorithm *SupRAnsSet*, where we discuss the key implementation issues and present a declarative realization of support set handling in the steps (c)–(g).

5. Implementation

We have implemented the repair answer set computation algorithms as a part of the dlliteplugin plugin of the dlvhex framework, thus providing a means to effectively compute deletion repair answer sets for DL-programs over *DL-Lite*$_{\mathcal{A}}$ ontologies.

The dlvhex framework is a system for evaluating Answer Set Programs with external computations. The system is written in C++ and open source available.[6] Implementations of external source functions can be conveniently provided as plugins, which distinguishes the dlvhex system from other ASP solvers. A wide range of such plugins are already available, ranging from string manipulation functions to complex plugins implementing Equilibrium-semantics of Multi-Context Systems.

The source code of the new plugin is available at https://github.com/hexhex/dlliteplugin. The dlliteplugin uses the owlcpp[7] [58] library for ontology parsing and invokes the fact++[8] system as a back-end for ontology reasoning tasks. In the sequel, we present an overview of the dlliteplugin architecture and give some implementation details.

5.1. Architecture overview

The architecture of the dlliteplugin is shown in Fig. 2, where arcs model both control and data flow of the system. The DL-program at hand is described by the user in the files, storing the ontology part $\mathcal{O}$ and the DL-rules part $\mathcal{P}$ of the DL-program Π respectively. After creating the replacement program $\hat{\Pi}$, complete nonground support families for DL-atoms in Π are determined within the dlliteplugin. The support sets in these families are processed declaratively using rules Π_{supp} (explained in detail below). These rules are then extended with the facts encoding ontology ABox and the program $\hat{\Pi}$. The models of the obtained program encode the repair answer sets and repairs of the original DL-program Π. For evaluating the declarative program, the backend grounder and the solver of the dlvhex system are invoked. Finally, the repair answer set candidates I of Π and their respective repairs $\mathcal{A}'$ are extracted from the computed models. Each such I is already a weak repair answer set of Π. For *flp*-repair answer sets, an additional *flp*-minimality check is made.

6 https://github.com/hexhex.
7 http://owl-cpp.sourceforge.net.
8 https://code.google.com/p/factplusplus.

$$Prog_{\mathcal{T}_{class}} = \left\{ \begin{array}{ll} (1) & sub(X,Z) \leftarrow sub(X,Y), sub(Y,Z); \\ (2) & sub(Y',X') \leftarrow sub(X,Y), op(X,X'), op(Y,Y'); \\ (3) & sub(X',Y') \leftarrow sub(X,Y), inv(X,X'), inv(Y,Y'); \\ (4) & conf(X,Y') \leftarrow sub(X,Y), op(Y,Y'); \\ (5) & inv(X',X) \leftarrow inv(X,X'); \\ (6) & op(X,Y) \leftarrow op(Y,X); \\ (7) & confref(X) \leftarrow conf(X,Y), op(Y,Z), inv(X,Z); \end{array} \right\}$$

Fig. 3. Program $Prog_{\mathcal{T}_{class}}$ for computing classification of $\mathcal{T}$.

$$\begin{array}{llll} (1) & \bot \leftarrow e_a(\vec{X}), not\ Sup_a(\vec{X}); & (3) & Sup_a(\vec{X}) \leftarrow r(S_a(\vec{Y})), not\ \bar{S}_a^{\mathcal{A}}(\vec{Y}); \\ (2) & \bot \leftarrow ne_a(\vec{X}), Sup_a(\vec{X}); & (4) & \bar{S}_a^{\mathcal{A}}(\vec{Y}) \leftarrow ne_a(\vec{X}), r(S_a(\vec{Y})) \end{array}$$

Fig. 4. Rules of the program Π_{supp}.

5.2. Implementation details

In order to take advantage of existing dlvhex data structures (e.g. for parsing) and optimization methods (such as nogood learning, etc.), a declarative ASP approach was pursued to realize both construction of complete support families and computation of repair answer sets and repairs over *DL-Lite*$_{\mathcal{A}}$ ontologies.

First we describe our approach to computing the support families. The routine for computing support families gets a *DL-Lite*$_{\mathcal{A}}$ ontology and a nonground DL-atom as input. After parsing the ontology $\mathcal{O}$ using the owlcpp library, its TBox classification is computed. The latter is done declaratively using the program $Prog_{\mathcal{T}_{class}}$ shown in Fig. 3, which exploits a construction for computing unary and binary conflict sets expressed in [74].

The program $Prog_{\mathcal{T}_{class}}$ reifies concepts (roles, existential restrictions on roles), as well as positive replacements of their negations. Facts express subsumptions in $Pos(\mathcal{T})$ using predicate *sub*, role inverses using *inv*, role functionalities with *funct*, and the duality of concepts (roles, etc.) and their opposites with *op*. The rule (1) of $Prog_{\mathcal{T}_{class}}$ transitively closes the subsumption relation, while (2) expresses contraposition for subsumption and (3) derives subsumption relations for roles whose inverses are subsumed. The rules (4)–(7) mimic the construction of binary and unary conflict sets (based on the theoretical results from [74]) that are then stored in the predicates *conf* and *confref* respectively. Since the program $Prog_{\mathcal{T}_{class}}$ is positive, it has a single answer set $M_{\mathcal{T}_{class}}$, from which the support family $\mathcal{S}$ for a DL-atom $d = \mathrm{DL}[\lambda;\ Q](X)$ is conveniently extracted in the following way:

- for every $sub(P,Q) \in M_{\mathcal{T}_{class}}$, where P is a positive ontology predicate, we add $S = \{P(\vec{X})\}$ to $\mathcal{S}$;
- for every $sub(P,Q) \in M_{\mathcal{T}_{class}}$, where P is a replacement for an existential restriction $\exists R$, we add $S = \{R(X,Y)\}$ to $\mathcal{S}$;
- for every $conf(P,P') \in M_{\mathcal{T}_{class}}$, we add $S = \{P_p(\vec{Y}), P'(\vec{Y})\}$ to $\mathcal{S}$, if $P_p(\vec{c}) \in \mathcal{A}_d$ for some $c \in \mathcal{C}$, and there is no $S' \subset S$ such that $S' \in \mathcal{S}$;
- for every $conf(P,P') \in M_{\mathcal{T}_{class}}$, we add $S = \{P_p(\vec{Y}), P'_p(\vec{Y})\}$ to $\mathcal{S}$, if $P_p(\vec{c}), P'_p(\vec{d}) \in \mathcal{A}_d$ for some $\vec{c}, \vec{d} \in \mathcal{C}$, and there is no $S' \subset S$ such that $S' \in \mathcal{S}$;
- for every $confref(P) \in M_{\mathcal{T}_{class}}$, we add $S = \{P_p(Y,Y)\}$ to $\mathcal{S}$, if $P_p(c,d) \in \mathcal{A}_d$ for some $c, d \in \mathcal{C}$;
- for every $funct(P) \in M_{\mathcal{T}_{class}}$, we add $S = \{P_p(Y,Z), P_p(Y,Z'), Z \neq Z'\}$ to $\mathcal{S}$, if $P_p(c,d), P_p(c,e) \in \mathcal{A}_d$ for some $c, d, e \in \mathcal{C}$;
- for every $funct(P) \in M_{\mathcal{T}_{class}}$, we add $S = \{P_p(Y,Z), P(Y,Z'), Z \neq Z'\}$ to $\mathcal{S}$, if $P_p(c,d) \in \mathcal{A}_d$ and $P(c,e) \in \mathcal{A}$ for some $c, d, e \in \mathcal{C}$.

From Proposition 58, the definition of complete support families and the results in [74], we obtain:

Proposition 73. *The support family constructed from the model $M_{\mathcal{T}_{class}}$ of $Prog_{\mathcal{T}_{class}}$ is complete.*

We now turn to determining the repair answer sets, for which we use also a declarative approach. More specifically, for every nonground DL-atom $a(\vec{X})$ and its nonground support set $S_a(\vec{Y})$ with $\vec{Y} = \vec{X}\vec{X}'$, the rules from Fig. 4 are constructed. These form a program Π_{supp}, which is added to the replacement program $\hat{\Pi}$ to filter candidate deletion repair answer sets as done by the algorithm *SupRAnsSet*. Here $r(S_a(\vec{Y}))$ is a suitable representation of a support set $S_a(\vec{Y})$ for a DL-atom $a(\vec{X})$ using predicates $p(\vec{X})$ for input assertions $P_p(\vec{X})$, resp. $p_P(\vec{X})$ ($np_P(\vec{X})$) for ABox assertions $P(\vec{X})$ ($\neg P(\vec{X})$). $\bar{S}_a^{\mathcal{A}}$ states that the ABox part of S_a is marked for deletion if $S_a \cap \mathcal{A} \neq \emptyset$, otherwise it is void. Furthermore, Sup_a is a fresh predicate not occurring in $\mathcal{P}$, that says a has an applicable support set, i.e. its ABox part is either empty or not marked for deletion. The resulting program intuitively prunes candidates $\hat{I}$ (resp. encodes deletion repair answer sets) according to the algorithm *SupRAnsSet*.

(1) $man(X) \leftarrow e_{a_1}(X);$

(2) $\bot \leftarrow e_{a_2}(X);$

(3) $e_{a_1}(X) \vee ne_{a_1}(X);$

(4) $e_{a_2}(X) \vee ne_{a_2}(X);$

(5) $\bot \leftarrow e_{a_1}(X), not\ Sup_{a_1}(X);$

(6) $\bot \leftarrow ne_{a_1}(X), Sup_{a_1}(X);$

(7) $Sup_{a_1}(X) \leftarrow r(S_{2a_1}(X)), not\ \bar{S}_{2a_1}^{\mathcal{A}}(X);$

(8) $Sup_{a_1}(X) \leftarrow r(S_{1a_1}(X)), not\ \bar{S}_{1a_1}^{\mathcal{A}}(X);$

(9) $\bar{S}_{1a_1}^{\mathcal{A}}(X) \leftarrow ne_{a_1}(X), r(S_{1a_1}(X));$

(10) $\bar{S}_{2a_1}^{\mathcal{A}}(X) \leftarrow ne_{a_1}(X), r(S_{2a_1}(X));$

(11) $\bot \leftarrow e_{a_2}(X), not\ Sup_{a_2}(X);$

(12) $\bot \leftarrow ne_{a_2}(X), Sup_{a_2}(X);$

(13) $Sup_{a_2}(X) \leftarrow r(S_{1a_2}(X)), not\ \bar{S}_{1a_1}^{\mathcal{A}}(X)$

(14) $\bar{S}_{1a_2}^{\mathcal{A}}(X) \leftarrow ne_{a_2}(X), r(S_{1a_2}(X));$

Fig. 5. Rules of the program Π_{supp}.

Example 74. Consider a simple program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where

$$\mathcal{O} = \{Student(pat)\},$$

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ man(X) \leftarrow \mathrm{DL}[Male \uplus man;\ Male](X); \\ (2)\ \bot \leftarrow \mathrm{DL}[;\ Student](X) \end{array} \right\}.$$

The DL-atom $a_1(X) = \mathrm{DL}[Male \uplus man;\ Male](X)$ has $S_{1a_1}(X) = \{Male(X)\}$ and $S_{2a_1}(X) = \{Male_{man}(X)\}$ as its support sets, while the DL-atom $a_2(X) = \mathrm{DL}[;\ Student](X)$ has the support set $S_{1a_2}(X) = \{Student(X)\}$. The declarative program $\hat{\Pi} \cup \Pi_{supp} \cup$ *facts*$(\mathcal{A})$ contains the data part *facts*$(\mathcal{A}) = \{p_{Student}(pat)\}$ encoding the ABox assertion $Student(pat)$ using a fresh predicate $p_{Student}$, and the rules $\hat{\Pi} \cup \Pi_{supp}$ shown in Fig. 5, where

- $r(S_{1a_1}(X)) = p_{Male}(X);\ \bar{S}_{1a_1}^{\mathcal{A}}(X) = \bar{p}_{Male}(X);\ r(S_{2a_1}(X)) = man(X);$
- $\bar{S}_{2a_1}^{\mathcal{A}}(X) = \emptyset;\ r(S_{1a_2}(X)) = p_{Student}(X);\ \bar{S}_{1a_2}^{\mathcal{A}}(X) = \bar{p}_{Student}(X).$

The rules (1)–(4) correspond to $\hat{\Pi}$, the other rules form the program Π_{supp} encoding the support information for the DL-atoms $a_1(X)$ in (5)–(10) and $a_2(X)$ in (11)–(14) respectively. □

The correctness of the described declarative implementation is now formally stated.

Proposition 75. *Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a ground DL-program, where $\mathcal{O}$ is a DL-Lite$_{\mathcal{A}}$ ontology, and let $a_1, \ldots, a_n$ be the DL-atoms of Π. Let, moreover, $\mathcal{S}_1, \ldots, \mathcal{S}_n$ be complete nonground support families for $a_1, \ldots, a_n$ w.r.t. $\mathcal{O}$, and let Π_{supp} be the set of rules of the forms (r_1)–(r_4) constructed for every support set from $\mathcal{S}_i$ covering a_i, $1 \le i \le n$. Then*

$$AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))|_{\Pi} = RAS_{weak}(\Pi),$$

where $facts(\mathcal{A}) = \{p_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A}\} \cup \{np_P(\vec{c}) \mid \neg P(\vec{c}) \in \mathcal{A}\}$ is the set of facts corresponding to the assertions from $\mathcal{A}$ and $AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))|_{\Pi} = \{I|_{\Pi} \mid I \in AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))\}$.

Observe that our declarative implementation computes exactly the weak repair answer sets. Thus, in some cases rarely met in practice [30] an additional minimality check is needed to ensure that the identified weak repair answer set is also an *flp*-repair answer set. This happens in case of cyclic support, i.e. recursion through a DL-atom that makes an atom true [37]. We illustrate this by the following example:

Example 76. Reconsider $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ from Example 74. For the interpretation $\hat{I} = \{man(pat), e_{a_1}, ne_{a_2}, Sup_{a_1}\ \ p_{Student(pat)}, \bar{p}_{Student(pat)}\}$ we have that $(\hat{\Pi} \cup \Pi_{supp})^{\hat{I}}_{gl}$ contains the rules (1)–(5), (7), (9), (10'), (11) and (12), where (10') is the rule $\bot \leftarrow e_{a_2}$. It holds that $\hat{I}$ is a minimal model of this reduct, thus an answer set of $\hat{\Pi} \cup \Pi_{supp}$. As $\bar{p}_{Student}(pat) \in I$ the repair $\mathcal{A}' = \mathcal{A} \setminus \{Student(pat)\}$ is extracted from $\hat{I}$. Let us now look at $I|_{\Pi} = \{man(pat)\}$. Certainly, $I|_{\Pi}$ is a minimal model of the weak reduct

$$\mathcal{P}_{weak}^{I|_{\Pi}, \mathcal{O}'} = \{man(pat)\},$$

where $\mathcal{O}' = \emptyset$, and therefore $I|_{\Pi}$ is a weak repair answer set of Π. However, $I|_{\Pi}$ is not an *flp*-repair answer set of Π, since $I' \subset I|_{\Pi}$ exists, namely $I' = \emptyset$, which is a smaller model of the *flp* reduct

$$\mathcal{P}_{flp}^{I|_{\Pi}, \mathcal{O}'} = \{man(pat) \leftarrow \mathrm{DL}[Male \uplus man;\ Male](pat)\}. \qquad \square$$

6. Evaluation

For evaluating the developed deletion repair answer set computation algorithms based on complete support families, we have built a benchmark suite, consisting of DL-programs over ontologies in $DL\text{-}Lite_{\mathcal{A}}$. The assessment of our algorithms concerned the following aspects:

- *Performance.* We evaluated the performance of deletion repair answer set computation in comparison to the standard answer set computation on various benchmarks including Family, Network, Taxi and LUBM. For the Family benchmark we additionally varied the following parameters:
  - size of the DL-program data part;
  - size of the ontology TBox;
  - number of rules in the DL-program.
- *Exploiting DL-programs expressive power.* We analyzed how various advanced expressive features allowed in DL-programs like defaults, guesses and recursiveness, influence the repair answer set computation running time (Network, LUBM benchmarks).
- *Repair quality.* The σ-selection functions that we introduced allow one to restrict the repair search space to application-relevant repair candidates, thus ensuring a certain level of quality of the results. We evaluated how the independent σ-selection functions, like bound on the number/type of assertions eligible for deletion influence the overall algorithm runtime.
- *Real world data.* To demonstrate the applicability of the developed algorithms to the real world scenarios, we conducted experiments on the DL-programs from the taxi-driver assignment problem over the MyITS ontology[9] designed for personalized route planning.

6.1. Platform description

We have evaluated the repair answer set computation approach on a Linux server with two 12-core AMD 6176 SE CPUs with 128 GB RAM running the HTCondor load distribution system,[10] which is a specialized workload management system for compute-intensive tasks. We used the version 2.3.0 of the dlvhex system. For each run the system usage was limited to two cores and 8 GB RAM. The timeout was set to 300 seconds for each instance. The experimental data are online available.[11]

Since to the best of our knowledge no other algorithms for repairing DL-programs are available, we had to proceed with comparison of our approach to the standard answer set computation.

The list of systems for DL-programs evaluation includes the following:

- The DReW system[12] [83] is designed for evaluating DL-programs by means of a rewriting to datalog. A straightforward implementation of the repair computation was realized within the DReW system with the naive guess of the repair ABox candidate, followed by a check of its suitability. However, this implementation turned out to be ineffective even on small instances, since in general the search space of the repairs is too big for its full exploitation, and guided search is vital to ensure scalability. We have not performed a full comparison of our implementation with the DReW system, since in its current version negative queries and the negative updates (operator $\uplus$) are not supported.
- The dlplugin of dlvhex,[13] which uses the RacerPro reasoner as a back-end for evaluation of the calls to the ontology, is another candidate for comparison. However, since the dlliteplugin used for standard answer set computation for DL-programs over lightweight ontologies scales better than the former [32], we use the latter for comparison in our experiments.

6.2. Evaluation workflow

We now describe the general workflow of the experimental evaluation.

In the first step of the evaluation process we **constructed benchmarks**. This was nontrivial, since first very few benchmarks already exist [83] and second it is difficult to synthesize random test instances whose conflict space would effectively reflect realistic scenarios. We exploited the existing ontologies and aimed at building rules and constraints on top of them in such a way that for some data parts the constructed programs become inconsistent.

9 http://www.kr.tuwien.ac.at/research/projects/myits/.
10 http://research.cs.wisc.edu/htcondor.
11 http://www.kr.tuwien.ac.at/staff/dasha/thesis/experimental_data.zip.
12 http://www.kr.tuwien.ac.at/research/systems/drew/.
13 https://github.com/hexhex/dlplugin.

Table 2
Family benchmark: data size variations, fixed $\mathcal{P}$ and $\mathcal{T}$.

p	$\mathcal{A}_{50}$			$\mathcal{A}_{1000}$		
	AS	*RAS*		*AS*	*RAS*	
		no_restr	$lim = 10$		*no_restr*	$lim = 20$
10 (20)	0.14 (0)[0]	0.22 (0)[20]	1.73 (0)[20]	63.12 (0)[0]	37.03 (0)[20]	60.21 (0)[20]
20 (20)	0.14 (0)[0]	0.23 (0)[20]	2.10 (0)[19]	62.56 (0)[0]	38.56 (0)[20]	62.19 (0)[20]
30 (20)	0.14 (0)[0]	0.24 (0)[20]	2.33 (0)[18]	62.83 (0)[0]	40.03 (0)[20]	64.27 (0)[20]
40 (20)	0.14 (0)[0]	0.25 (0)[20]	2.88 (0)[11]	63.23 (0)[0]	41.81 (0)[20]	66.81 (0)[20]
50 (20)	0.14 (0)[0]	0.25 (0)[20]	3.93 (0) [1]	63.42 (0)[0]	43.86 (0)[20]	68.87 (0)[20]
60 (20)	0.15 (0)[0]	0.26 (0)[20]	3.93 (0) [2]	63.42 (0)[0]	45.87 (0)[20]	71.63 (0)[20]
70 (20)	0.14 (0)[0]	0.27 (0)[20]	4.00 (0) [0]	63.18 (0)[0]	47.83 (0)[20]	74.14 (0)[20]
80 (20)	0.15 (0)[0]	0.28 (0)[20]	4.08 (0) [0]	63.38 (0)[0]	49.71 (0)[20]	76.35 (0)[20]
90 (20)	0.15 (0)[0]	0.29 (0)[20]	4.48 (0) [0]	63.59 (0)[0]	52.18 (0)[20]	79.14 (0)[20]
100 (20)	0.14 (0)[0]	0.30 (0)[20]	4.42 (0) [0]	63.08 (0)[0]	54.14 (0)[20]	81.81 (0)[20]

When the scenario was defined, we created shell scripts for **instance generation** with certain varying parameters (e.g. data size, rules size, TBox size), specific for each benchmark.[14] We then **ran the benchmarks** using the HTCondor system and finally **extracted** the results from the log files of the runs.

For each benchmark we **present** our experimental results in tables. The first column, p, in the tables specifies the size of the instance (varied according to certain parameters specific for each benchmark), and the number of generated instances in round brackets. For example, the value 10(20) in the first column states that 20 instances with the size of the parameter equal to 10 were evaluated. The rest of the columns vary from benchmark to benchmark. They represent configurations, in which the system was tested: *AS* (*RAS*) stands for normal (repair) answer set computation. Restrictions on repairs are applied in some cases the meaning of which is separately clarified where tables are presented. The cells contain combinations of numbers of the form $t(m)[n]$, where t is the total average running time in seconds, m is the number of timeouts and n is the number of (repair) answer sets computed.

6.3. Benchmarks

For the evaluation of the algorithms, we considered the following benchmarks.

(1.1) The Family benchmark describes a scenario that is built from a version of Example 1 with ABoxes $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$ containing 50 and 1000 children and information about their families;
(1.2) The Network benchmark comprises rules with recursion and guessing features over an ontology containing data about availability of nodes and edges of a network. We considered a fragment of the Vienna transport system with 161 nodes, and its part with 67 nodes, covering central area;
(1.3) The Taxi benchmark represents a driver-customer assignment problem over an ontology with ABoxes $\mathcal{A}_{50}$ and $\mathcal{A}_{500}$ containing information about 50 and 500 customers respectively. Based on certain conditions about the drivers, customers, their positions and intentions, the customers are assigned to drivers for serving needs of customers;
(1.4) The LUBM benchmark is a set of rules with various expressiveness features built over the famous LUBM ontology[15] in its *DL-Lite*$_{\mathcal{A}}$ form containing information about one university. The original version of LUBM is in $\mathcal{ALEHI}(D)$ form. For creating the *DL-Lite*$_{\mathcal{A}}$ version of LUBM we rewrote if possible and removed otherwise the TBox axioms that do not fall into the DL *DL-Lite*$_{\mathcal{A}}$. For ABox generation we used the dedicated Combo tool.[16]

6.3.1. Family benchmark

The first benchmark is derived from Example 1. For our evaluation we have constructed different scenarios, varying the size of the TBox, the data part as well as the rule part of the DL-program.

1. Size of the data part. In the first setting, we fixed two ABoxes $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$, where $\mathcal{A}_{50}$ contains 50 children (7 adopted), 20 female and 32 male adults; and for $\mathcal{A}_{1000}$ twenty times as many. 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 $\mathcal{P}$. Instances are varied in terms of facts over **I** included in $\mathcal{P}$. The parameter reflecting the instance size is p, which ranges from 10 to 100. A benchmark instance has size p if for every child c, additional facts $boy(c)$ and $isChildOf(c,d)$ appear in $\mathcal{P}$ with a probability $p/100$, where d is a random male adult non-parent. As the number of facts in $\mathcal{P}$ varies, the size of the actual conflict part in the program can be controlled.

[14] The scripts are available at https://github.com/hexhex/dlplugin/benchmarks.
[15] http://swat.cse.lehigh.edu/projects/lubm/.
[16] http://code.google.com/p/combo-obda/.

Table 3
Family benchmark: TBox size variations, fixed $\mathcal{P}$ and $\mathcal{A}_{50}$.

p	$\mathcal{T}_{500}$			$\mathcal{T}_{5000}$		
	AS	*RAS*		*AS*	*RAS*	
		no_restr	$lim = 10$		*no_restr*	$lim = 10$
10 (20)	0.15 (0)[0]	0.32 (0)[20]	1.95 (0)[20]	0.28 (0)[0]	3.58 (0)[20]	6.03 (0)[20]
20 (20)	0.16 (0)[0]	0.47 (0)[20]	2.17 (0)[20]	0.48 (0)[0]	12.89 (0)[20]	15.96 (0)[20]
30 (20)	0.17 (0)[0]	0.68 (0)[20]	2.47 (0)[20]	0.75 (0)[0]	27.76 (0)[20]	31.42 (0)[20]
40 (20)	0.19 (0)[0]	0.93 (0)[20]	2.78 (0)[20]	1.10 (0)[0]	48.46 (0)[20]	53.24 (0)[20]
50 (20)	0.20 (0)[0]	1.25 (0)[20]	3.19 (0)[20]	1.51 (0)[0]	76.39 (0)[20]	81.54 (0)[20]
60 (20)	0.21 (0)[0]	1.58 (0)[20]	3.56 (0)[20]	1.99 (0)[0]	108.33 (0)[20]	114.71 (0)[20]
70 (20)	0.23 (0)[0]	2.09 (0)[20]	4.18 (0)[20]	2.56 (0)[0]	146.62 (0)[20]	152.91 (0)[20]
80 (20)	0.24 (0)[0]	2.54 (0)[20]	4.68 (0)[20]	3.17 (0)[0]	191.37 (0)[20]	198.72 (0)[20]
90 (20)	0.26 (0)[0]	3.06 (0)[20]	5.28 (0)[20]	3.91 (0)[0]	241.51 (0)[20]	248.19 (0)[20]

Table 4
Family benchmark: rule size variations, fixed $\mathcal{T}$ and $\mathcal{A}_{50}$.

p	$Rules_{50}$		$Rules_{500}$		$Rules_{5000}$	
	RAS	$RAS_{lim=10}$	*RAS*	$RAS_{lim=10}$	*RAS*	$RAS_{lim=20}$
10 (20)	0.55 (0)[20]	2.09 (0)[20]	2.56 (0)[20]	23.23 (0)[0]	64.65 (0)[20]	110.92 (0)[20]
20 (20)	0.69 (0)[20]	2.35 (0)[20]	5.22 (0)[20]	77.30 (0)[0]	257.35 (11)[9]	300.00 (20)[0]
30 (20)	0.90 (0)[20]	2.67 (0)[20]	8.50 (0)[20]	158.23 (0)[0]	300.00 (20)[0]	300.00 (20)[0]
40 (20)	0.97 (0)[20]	2.86 (0)[20]	11.86 (0)[20]	128.87 (1)[0]	300.00 (20)[0]	300.00 (20)[0]
50 (20)	1.18 (0)[20]	3.11 (0)[20]	14.91 (0)[20]	144.71 (0)[0]	300.00 (20)[0]	300.00 (20)[0]
60 (20)	1.29 (0)[20]	3.28 (0)[20]	17.68 (0)[20]	164.70 (0)[0]	300.00 (20)[0]	300.00 (20)[0]
70 (20)	1.42 (0)[20]	3.19 (0)[20]	20.11 (0)[20]	186.38 (3)[0]	300.00 (20)[0]	300.00 (20)[0]

The results for this benchmark are provided in Table 2. For each probability p we generated 20 random instances with the fixed $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$ ABoxes, and evaluated the running time for the standard answer set (column *AS*) and the repair answer set computation (column *RAS*) with no restrictions on the repairs (column *no_restr*) as well as limiting the number of allowed assertions for deletion to 10 for $\mathcal{A}_{50}$ and 20 for $\mathcal{A}_{1000}$ (columns $lim = 10$ and $lim = 20$ resp.).

The numbers in the second column reveal that all considered instances are inconsistent, which is recognized by the AS solver within 2 milliseconds. In most cases all the repairs are found for both $\mathcal{A}_{50}$ and $\mathcal{A}_{1000}$ except for $lim = 10$ of $\mathcal{A}_{50}$, where the repairs are computed only for some of the instances up to $p = 60$.

2. Ontology TBox size. In the second setting, we built instances based on the size of the TBox, leaving the ontology ABox fixed to $\mathcal{A}_{50}$ and the rule part same as in the previous benchmark setting. The TBox axioms from Example 1 are extended by the inclusions $P \sqsubseteq Person$ for all concepts P, informally stating that every individual known to be either child, adopted, male or female is a person. Moreover, for each concept P from the ontology signature and $1 \leq i \leq \mathcal{T}_{max}$, we added the following inclusions with probability $p/100$ (p ranges from 10, 20 to 90).

$$(1)\ PMemberOfSocGroup_i \sqsubseteq P \quad (2)\ \exists hasIDOfSocGroup_i \sqsubseteq Person.$$

Intuitively, (1) reflects that a P-member of a social group i is in the class P, while (2) states that each individual having ID of a certain social group i is a person.

The evaluation results for this setting are presented in Table 3. One can see that the repair computation is slower then the standard answer set computation, which is more obvious for $\mathcal{T}_{5000}$; This is due to the construction of support sets and their exploitation in the declarative approach for repair answer set computation. In the standard setting, we do not exploit the TBox extensively, and therefore its growing size does not affect the running time. As expected, bounding a number of eliminated facts to k slows down the repair computation process.

3. Size of the rule part. The third setting evaluates the influence of the rule part size. Apart from the rules (11) and (12) from Example 1 that were excluded in the previous settings, we also added for $1 \leq i \leq R_{max}$ and for $1 \leq j \leq i$ with probability $p/100$ ($10 \leq p \leq 70$) the following rules:

$$(1)\ contact_i(X, Y) \leftarrow contact(X, Y), not\ omit(X, Y) \qquad (2)\ omit_i(X, Y) \leftarrow omit(X, Y)$$
$$(3)\ contact_j(X, Y) \leftarrow contact_i(X, Y), not\ omit_j(X, Y) \qquad (4)\ omit_j(X, Y) \leftarrow omit_i(X, Y).$$

The fresh predicates $contact_i(c, d)$ informally mean that d is a contact representative for a child c within a social group i. The rules (1)–(4) state that if a contact for a child was identified, then this contact can be propagated to randomly chosen social groups i and j.

Table 5
Network-connectivity benchmark results: $\mathcal{A}_{67}$.

p	*AS*	*RAS*			
		no_restr	$lim = 3$	$lim = 20$	$Broken, forbid$
2 (20)	0.10 (0)[12]	0.46 (0)[20]	0.84 (0)[20]	0.66 (0)[20]	0.46 (0)[20]
6 (20)	0.10 (0) [5]	0.45 (0)[16]	0.79 (0)[16]	0.61 (0)[16]	0.44 (0)[16]
10 (20)	0.09 (0) [3]	0.43 (0)[14]	0.76 (0)[14]	0.59 (0)[14]	0.43 (0)[14]
14 (20)	0.09 (0) [2]	0.41 (0)[10]	0.71 (0)[10]	0.54 (0)[10]	0.41 (0)[10]
18 (20)	0.09 (0) [0]	0.40 (0) [7]	0.67 (0) [7]	0.51 (0) [7]	0.40 (0) [7]
22 (20)	0.09 (0) [0]	0.41 (0) [9]	0.70 (0) [9]	0.54 (0) [9]	0.41 (0) [9]
26 (20)	0.09 (0) [0]	0.38 (0) [3]	0.63 (0) [3]	0.47 (0) [3]	0.38 (0) [3]
30 (20)	0.09 (0) [0]	0.37 (0) [2]	0.62 (0) [2]	0.46 (0) [2]	0.37 (0) [2]

The results are presented in Table 4. Standard answer set computation times out even for smaller instances. Intuitively, this is due to the large number of rules in the programs. For a fair comparison, standard answer set optimization techniques that evaluate independent components of a DL-program separately were not considered. We used a monolithic evaluation heuristics instead. The repair model generator does not support module-based heuristics at the moment and the extensions are nontrivial.

The maximal number of rules that were added is specified in the column "names". Each such rule is present in the test instance with probability $p/100$. We can see that the growing number of rules makes an impact on the running time of the algorithm, which is not surprising, as the added rules introduce conflicts due to a cycle through negation. Restricting the elimination to 10 facts slows down the computation for $Rule_{50}$ compared to the unrestricted scenario. For larger instance size, i.e. $Rules_{500}$, this restriction turns out to be too strict, thus no repairs are actually found. Weakening the restriction for larger instance size ($Rules_{5000}$) produces again some repair answer sets, though only for smaller p. For larger p timeouts result, which is natural as even for a standard ASP solver and consistent DL-programs with thousands of rules, their evaluation is time-consuming.

6.3.2. Network benchmark

In the next scenario, the properties of the nodes and edges in a network are described by a fixed ontology $\mathcal{O}$ using predicates $Blocked$, $Broken$, $Avail$ for nodes and $forbid$ for edges. The TBox encodes that if an edge is forbidden, then its starting point 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 considered two networks, N_1 and N_2, that are fragments of the Vienna public transportation net. Network N_1 corresponds to the central area of the metro lines and has 67 nodes and 117 edges; N_2 covers all metro lines and part of the urban railways, and has 161 nodes and 335 edges. In each network we randomly made 30% of the nodes broken and 20% of the edges forbidden; network N_2 has in addition 47 blocked nodes. This information is stored in the data part of $\mathcal{O}$.

The experiments were run on two DL-programs $\mathcal{P}_{conn}$ and $\mathcal{P}_{guess}$ over $\mathcal{O}$. Both programs contain as facts edges and nodes of the graph, as well as randomly generated facts determining the portion of the nodes on which a condition expressed by the rules of the program is checked. For creating the data part of the $\mathcal{P}_{conn}$ program, we partitioned the set of nodes randomly into two sets, i.e. the set of in nodes and the set of out nodes. For each node n from the in set, the fact $in(n)$ is added with probability $p/100$. For each node n' from the set of out nodes, the fact $out(n')$ is added with probability p' computed in the following way: if $0 \leq p \leq 20$, then $p' = p * 4/100$, if $20 \leq p \leq 30$, then $p' = p * 3/100$. $\mathcal{P}_{conn}$ contains, moreover, the following rules:

$$\mathcal{P}_{conn} = \left\{ \begin{array}{l} (1)\ go(X,Y) \leftarrow open(X), open(Y), edge(X,Y); \\ (2)\ route(X,Z) \leftarrow route(X,Y), route(Y,Z); \\ (3)\ route(X,Y) \leftarrow go(X,Y), not\ \mathrm{DL}[;\ forbid](X,Y); \\ (4)\ open(X) \leftarrow node(X), not\ \mathrm{DL}[;\ \neg Avail](X); \\ (5)\ ok(X) \leftarrow in(X), out(Y), route(X,Y); \\ (6)\ fail \leftarrow in(X), not\ ok(X); \\ (7)\ \perp \leftarrow fail. \end{array} \right\}$$

Intuitively, (1)–(4) 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. Rules (5)–(7) encode the requirement that each node from the in set must be connected to at least one node from the out set via a route, which amounts to a variation of a generalized connectivity problem.

The running times and repair results for the benchmark with N_1 are given in Table 5. The same number of repairs is computed for all of the *RAS* settings, but the running times for these settings slightly vary as expected. The last column,

Table 6
Network-connectivity benchmark results: $\mathcal{A}_{161}$.

p	RAS				
	no_restr	$lim = 3$	$lim = 20$	$lim = 100$	$Broken, forbid$
2 (20)	179.49 (1)[19]	280.73 (16)[0]	288.64 (17)[3]	176.06 (1)[19]	125.47 (0)[0]
4 (20)	218.80 (8)[12]	291.80 (18)[0]	295.48 (19)[1]	226.25 (8)[12]	127.68 (0)[0]
8 (20)	230.79 (9)[11]	298.39 (19)[0]	300.00 (20)[0]	232.65 (9)[11]	126.97 (0)[0]
10 (20)	258.08 (14)[5]	300.00 (20)[0]	300.00 (17)[0]	259.69 (14)[6]	125.63 (0)[0]

Table 7
Network-guessing benchmark results: $\mathcal{A}_{67}$.

p	AS	RAS				
		no_restr	$lim = 3$	$lim = 10$	$limc = 10$	$Broken$
2 (20)	180.06 (12)[0]	0.51 (0)[20]	0.91 (0)[19]	0.90 (0)[20]	0.69 (0)[20]	0.50 (0)[20]
10 (20)	15.17 (1) [0]	1.33 (0)[16]	0.89 (0) [2]	1.61 (0)[16]	0.85 (0)[16]	1.31 (0)[16]
18 (20)	0.18 (0) [0]	1.68 (0) [8]	0.90 (0) [0]	1.40 (0) [8]	0.81 (0) [8]	1.68 (0) [8]
26 (20)	0.19 (0) [0]	0.62 (0) [1]	0.97 (0) [0]	0.95 (0) [1]	0.60 (0) [1]	0.62 (0) [1]
34 (20)	0.20 (0) [0]	0.79 (0) [1]	1.04 (0) [0]	1.02 (0) [1]	0.62 (0) [1]	0.78 (0) [1]

Table 8
Network-guessing benchmark results: $\mathcal{A}_{161}$.

p	RAS			
	no_restr	$lim = 10$	$limc = 100$	$Broken$
2 (20)	178.52 (3)[15]	187.65 (2)[16]	175.64 (2)[16]	179.57 (3)[15]
4 (20)	201.89 (6)[10]	211.10 (7) [9]	213.66 (9) [7]	178.55 (3)[13]
8 (20)	212.18 (10) [2]	215.44 (10) [2]	205.77 (9) [3]	191.97 (7) [5]
10 (20)	190.58 (9) [0]	184.80 (8) [1]	191.54 (9) [0]	191.06 (9) [0]

where only broken nodes and forbidden edges are allowed for removal, has similar running times as the unrestricted setting. This is also the case for network N_2 (Table 6), where this restriction does not yield repairs. Here one also needs to remove blocked/unavailable nodes from the ontology in order to obtain repairs.

Another setting that we considered is a benchmark over the program $\mathcal{P}_{guess}$, which has the same rules (1) and (2) as $\mathcal{P}_{conn}$, while the rest of the rules are as follows:

(3*) $route(X,Y) \leftarrow go(X,Y), not\ \mathrm{DL}[Block \uplus block; forbid](X,Y);$

(4*) $open(X) \vee block(X) \leftarrow domain(X), not\ \mathrm{DL}[;\ \neg Avail](X);$

(5*) $open(X) \leftarrow node(X), not\ \mathrm{DL}[;\ Broken](X), not\ block(X);$

(6*) $negis(X) \leftarrow domain(X), route(X,Y), X \neq Y;$

(7*) $\perp \leftarrow domain(X), not\ negis(X)$

The rule (3*) has an update in the DL-atom; the rule (4*) amounts to guessing for all selected nodes (predicate *domain*) not known to be unavailable, whether they are blocked or not, i.e. it contains nondeterminism, which makes rule processing challenging. Other nodes are open by default, unless they are known to be broken, which is encoded in the rule (5*). Rules (6*) and (7*) check whether none of the *domain* nodes is isolated, i.e. does not have a connection to any other node via a route.

The results for $\mathcal{P}_{guess}$ with the two networks are in Tables 7 and 8, respectively. The facts $domain(n)$ are added for each node n with probability $p/100$. For the smaller network N_1 one can observe a strict increase in the running time for $p = 2$ and $p = 10$ in the standard answer set computation mode. As many of the instances for smaller p are consistent, due to the guessing rules the standard answer set solver can not compute the answer sets within the time frame of 300 seconds. For bigger p the instances are inconsistent and the conflict is quickly determined by the solver. The results for network N_2 in Table 8 show that the guided search (last column) increases the number of found repairs quite a bit, and less timeouts are hit for $p = 4$ and $p = 8$.

6.3.3. Taxi benchmark

The third experimental setting represents a taxi-driver assignment problem. Imagine a system that assigns potential customers to taxi drivers under constraints, using (in a simplistic form) the DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ presented in Fig. 6. The (external) ontology $\mathcal{O}$ has a taxonomy $\mathcal{T}$ in (1)–(3). The logic program $\mathcal{P}$ has the following rules: (5) and (6) single out customers resp. taxi drivers; (7) assigns taxi drivers to customers in the same region; and (8) forbids drivers of electro-cars to serve needs going outside their working region. Finally, the rules (9), (10) and a constraint (11) make sure that each customer is assigned to at least one driver.

Table 9
Taxi-basic benchmark results: $\mathcal{A}_{50}$.

p	AS	RAS					
		no_restr	$lim = 3$	$lim = 10$	$limp = 2$	$limc = 10$	$EDriver$
10 (20)	0.69 (0)[0]	0.14 (0)[13]	0.75 (0)[11]	0.75 (0)[13]	0.31 (0)[13]	0.26 (0)[13]	0.14 (0)[13]
20 (20)	0.37 (0)[0]	0.15 (0) [8]	0.89 (0) [4]	0.87 (0) [8]	0.32 (0) [8]	0.25 (0) [8]	0.15 (0) [8]
30 (20)	0.22 (0)[0]	0.16 (0) [7]	0.92 (0) [2]	0.89 (0) [7]	0.32 (0) [7]	0.26 (0) [7]	0.16 (0) [7]
40 (20)	0.58 (0)[0]	0.18 (0) [8]	1.06 (0) [1]	1.04 (0) [8]	0.36 (0) [8]	0.28 (0) [8]	0.18 (0) [8]
50 (20)	0.46 (0)[0]	0.18 (0) [7]	1.01 (0) [2]	0.98 (0) [7]	0.36 (0) [7]	0.29 (0) [7]	0.18 (0) [7]
60 (20)	0.22 (0)[0]	0.19 (0)[11]	1.02 (0) [1]	0.99 (0)[11]	0.38 (0)[11]	0.31 (0)[11]	0.19 (0)[11]
70 (20)	0.22 (0)[0]	0.21 (0) [4]	1.00 (0) [0]	0.99 (0) [4]	0.37 (0) [4]	0.29 (0) [4]	0.20 (0) [4]
80 (20)	1.02 (0)[0]	0.22 (0) [9]	1.10 (0) [1]	1.10 (0) [9]	0.40 (0) [9]	0.33 (0) [9]	0.22 (0) [9]
90 (20)	1.30 (0)[0]	0.23 (0)[12]	1.26 (0) [0]	1.20 (0)[12]	0.44 (0)[12]	0.36 (0)[12]	0.24 (0)[12]
100 (20)	1.47 (0)[0]	0.24 (0)[13]	1.20 (0) [0]	1.15 (0)[13]	0.45 (0)[13]	0.37 (0)[13]	0.26 (0)[13]

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \mathit{Driver} \sqsubseteq \neg \mathit{Cust} & (3)\ \exists \mathit{worksIn} \sqsubseteq \mathit{Driver} \\ (2)\ \mathit{EDriver} \sqsubseteq \mathit{Driver} & (4)\ \mathit{worksIn} \sqsubseteq \neg \mathit{notworksIn} \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (5)\ \mathit{cust}(X) \leftarrow \mathit{isIn}(X, Y), \mathit{not}\ \mathrm{DL}[;\ \neg \mathit{Cust}](X); \\ (6)\ \mathit{driver}(X) \leftarrow \mathit{not}\ \mathit{cust}(X), \mathit{isIn}(X, Y); \\ (7)\ \mathit{drives}(X,Y) \leftarrow \mathit{cust}(Y), \mathit{isIn}(Y,Z), \mathit{isIn}(X,Z), \\ \qquad\qquad\qquad\qquad \mathit{driver}(X), \mathit{not}\ \mathit{omit}(X, Y); \\ (8)\ \mathit{omit}(X,Y) \leftarrow \mathit{needsTo}(Y, Z),\ \mathrm{DL}[;\ \mathit{notworksIn}](X, Z), \\ \qquad\qquad\qquad\qquad \mathrm{DL}[\mathit{Driver} \uplus \mathit{driver};\ \mathit{EDriver}](X); \\ (9)\ \mathit{ok}(Y) \leftarrow \mathit{customer}(Y), \mathit{drives}(X, Y); \\ (10)\ \mathit{fail} \leftarrow \mathit{customer}(Y), \mathit{not}\ \mathit{ok}(Y); \\ (11)\ \bot \leftarrow \mathit{fail}. \end{array} \right\}$$

Fig. 6. DL-program from Taxi-basic benchmark.

1. Guided repair search. One might argue that in case of inconsistency there are not many possibilities for repairing the given system. Indeed, for instance, removing information about the drivers seems absurd at first glance, as some individuals are no longer known to be drivers, and thus assumed to be customers by default (5). Observe that a complete removal of driver information will not make the system consistent, but on the contrary will create even more customers, who will then possibly need to be assigned to the drivers. Therefore, it is obvious that the guided repair search is often crucial and it should not only improve the repair quality but also reduce the computation runtime.

In this setting we considered the evaluation time of the repair computation under various independent selection functions. The latter include restrictions to a certain set of predicates for deletion (in our case $EDriver$ assertions) and limiting the number of removed facts, predicates and constants. While the latter is natural and can be easily justified, one might wonder when removal of e-car driver is of practical use. We can imagine that e-cars are hybrid and can run on petrol, which for environmental reasons is undesired, and the government wants to reduce petrol usage. However, in case it is vital and some customers are left without drivers, they still can switch back to the petrol energy supply.

For the DL-program Π the ABox $\mathcal{A}_{50}$ contains 50 customers, 20 drivers (among them 19 driving electro-cars), and 5 regions; every driver works in 2–4 regions. In the program $\mathcal{P}$ from above, facts $\mathit{isIn}(c, r)$, $\mathit{needsTo}(c, r)$, $\mathit{goTo}(d, r)$ for appropriate constants c, d, r from $\mathcal{A}$ are randomly added with probability $p/100$ under the following constraints: persons are in at most one region; customers need to go to at most one region, and their position is known in that case. Furthermore, driver positions are added as facts $\mathit{isIn}(d, r)$ with fixed probabilities of 0.3, 0.7 and 1 growing discretely in accordance with p.

The results for $\mathcal{A}_{50}$ are given in Table 9, where the first column shows in parentheses the number of instances generated per value p. The second and third column state results for standard and repair answer set computation, respectively, while the rest of the columns present the running times for repair computation under various selection functions, i.e. in the fourth and fifth column we restricted repairs by allowing removal of only a limited number of assertions (3 and 10) and in the sixth and seventh column we computed repairs where only facts containing 2 predicates and 10 constants are eliminated. Finally in the last column the results for removing only $EDriver$ facts are shown.

One can see that bounding the number of removed assertions makes the computation slower. For $\mathit{repdel} = EDriver$, the guided repair computation effectively reduces the search space, and it helps the solver to find repairs quicker. In fact, the analysis of the program reveals that most of the valid repairs exclude certain $EDriver$ concept memberships, since they often cause the omission of driver-customer assignments and thus violate constraint (11).

Table 10
Taxi-districts benchmark results: $\mathcal{A}_{50}$.

p	*AS*	*RAS*					
		no_restr	$lim = 3$	$lim = 10$	$limp = 2$	$limc = 10$	$EDriver$
2 (20)	0.25 (0) [5]	4.12 (0) [5]	5.27 (0) [5]	5.32 (0) [5]	5.01 (0) [5]	4.98 (0) [5]	4.10 (0) [5]
10 (20)	0.25 (0) [0]	4.18 (0)[11]	6.19 (0) [7]	6.18 (0)[11]	5.22 (0)[11]	5.15 (0)[11]	4.13 (0) [3]
18 (20)	0.25 (0) [1]	4.24 (0)[14]	6.71 (0)[10]	6.74 (0)[14]	5.34 (0)[14]	5.19 (0)[14]	4.15 (0) [3]
26 (20)	0.25 (0) [1]	4.28 (0)[14]	7.26 (0) [9]	7.42 (0)[14]	5.50 (0)[14]	5.24 (0)[14]	4.22 (0) [5]
34 (20)	0.26 (0) [3]	4.39 (0)[19]	8.54 (0)[16]	8.52 (0)[19]	5.74 (0)[19]	5.40 (0)[19]	4.35 (0) [9]
42 (20)	0.27 (0) [5]	4.42 (0)[18]	9.35 (0)[18]	9.31 (0)[18]	5.86 (0)[18]	5.49 (0)[18]	4.51 (0)[16]
50 (20)	0.29 (0)[10]	4.49 (0)[19]	10.42 (0)[19]	10.29 (0)[19]	6.05 (0)[19]	5.54 (0)[19]	4.63 (0)[19]
58 (20)	0.32 (0)[14]	4.62 (0)[20]	11.48 (0)[20]	11.50 (0)[20]	6.33 (0)[20]	5.63 (0)[20]	4.76 (0)[20]
66 (20)	0.31 (0)[11]	4.61 (0)[20]	11.59 (0)[20]	13.42 (0)[20]	6.27 (0)[20]	5.71 (0)[20]	4.76 (0)[18]

Table 11
Taxi-districts benchmark results: $\mathcal{A}_{500}$.

p	*AS*	*RAS*					
		no_restr	$lim = 3$	$lim = 10$	$limp = 2$	$limc = 10$	$EDriver$
2 (20)	2.11 (0) [0]	9.22 (0) [7]	25.05 (0) [6]	24.91 (0) [7]	12.32 (0) [7]	10.24 (0) [6]	7.56 (0) [0]
10 (20)	2.23 (0) [0]	14.17 (0)[20]	46.37 (0)[20]	46.52 (0)[20]	20.54 (0)[20]	15.75 (0)[15]	12.16 (0) [4]
18 (20)	5.58 (0) [5]	15.96 (0)[20]	51.89 (0)[20]	52.44 (0)[20]	23.11 (0)[20]	17.93 (0)[20]	28.00 (0)[20]
26 (20)	17.95 (0)[12]	18.28 (0)[20]	55.30 (0)[20]	55.84 (0)[20]	25.57 (0)[20]	20.27 (0)[20]	31.76 (0)[20]
34 (20)	37.87 (0)[17]	20.81 (0)[20]	58.71 (0)[20]	58.51 (0)[20]	28.35 (0)[20]	22.93 (0)[20]	36.00 (0)[20]

2. Real world data. For another benchmark, we considered rules on top of the ontology developed in the MyITS project, which enhanced personalized route planning with semantic information [39].[17] That ontology is augmented with axioms (1)–(4) in Fig. 6 and the axiom (5') $adjoint \sqsubseteq \neg disjoint$, stating that adjoint regions are not disjoint; the resulting ontology has 389 TBox axioms on 339 concepts and 41 roles. This scenario is modeled to demonstrate the applicability of the repair answer set computation approach for TBoxes from a real world domain. We considered DL-programs over two ABoxes $\mathcal{A}_{50}$ and $\mathcal{A}_{500}$ (containing 10 times as many customers, drivers and e-car drivers as $\mathcal{A}_{50}$). Along with customer and driver information from above, the ABoxes also contain data about mutual spatial relations among the districts of Vienna. These relations are stored using the predicates *adjoint* and *disjoint*. The rule part of the DL-program has the same rules (5)–(6) and (9)–(11) as in Fig. 6, while the rules (7) and (8) are as follows:

$$
\begin{aligned}
(7^*)\quad & drives(X,Y) \leftarrow driver(X), cust(Y), needsTo(X,Z1),\\
& \qquad goTo(X,Z2), \mathrm{DL}[;\ adjoint](Z1,Z2), not\ omit(X,Y)\\
(8^*)\quad & omit(X,Y) \leftarrow \mathrm{DL}[;\ EDriver](X), needsTo(Y,Z), \mathrm{DL}[;\ notworksIn](X,Z)
\end{aligned}
$$

Intuitively, the rule (7*) states that a driver can be assigned to a customer only if the driver is going to a region adjoint to the destination region of the customer. Similar as in the previous scenario, some of the assignments are dropped if they involve drivers of e-cars aiming at the regions they are not assigned to. The rule (8*) is the same as the rule (8), with the only difference that the DL-atom involved in it does not have any updates.

The benchmark results for this setting and $\mathcal{A}_{50}$ are presented in Table 10. Unsurprisingly, the restriction on the number of assertions allowed for deletion slows down the repair computation again. With the increase of this limit the running time slightly improves. As in the previous setting the restriction of the set of predicates allowed for deletion to $EDriver$ does not yield much of the computation overhead; however, in contrast to the previous setting the number of repairs found decreases. Since the number of districts increased compared to the previous setting, apart from the information about drivers of e-cars, one needs to expand the working area of the drivers too; thus removal of *notworksIn* facts should again increase the number of obtained repairs.

Table 11 presents the results for the ABox $\mathcal{A}_{500}$. Despite a natural increase in running times compared to the smaller ABox, repairs are found in many cases for this setting. While the number of regions stays the same as for $\mathcal{A}_{50}$, proportionally there are more available drivers per district, and more customers can be served.

6.3.4. LUBM benchmark

We have evaluated also DL-programs over the famous LUBM ontology[18] in its $DL\text{-}Lite_{\mathcal{A}}$ form. For ABox generation we used the dedicated Combo tool.[19] We considered an extended assignment problem in combination with multiple mutually

[17] http://www.kr.tuwien.ac.at/research/projects/myits/.
[18] http://swat.cse.lehigh.edu/projects/lubm/.
[19] http://code.google.com/p/combo-obda/.

$$\mathcal{O} = \left\{ \begin{array}{ll} (1)\ \ \mathit{Student} \sqsubseteq \neg \mathit{NonStudent} & (4)\ \ \mathit{VisitPostDoc} \sqsubseteq \mathit{PostDoc} \\ (2)\ \ \mathit{VisitPostDoc} \sqsubseteq \mathit{ResearchAssistant} & (5)\ \ \mathit{Student} \sqsubseteq \mathit{OrgHelp} \\ (3)\ \ \mathit{VisitPostDoc} \sqsubseteq \mathit{NonStudent} & \end{array} \right\}$$

$$\mathcal{P} = \left\{ \begin{array}{l} (6)\ \ \mathit{resas}(X) \leftarrow \mathrm{DL}[;\ \mathit{ResearchAssistant}](X); \\ (7)\ \ \mathit{student}(X) \leftarrow \mathit{resas}(X), \mathit{not}\ \mathit{negStudent}(X); \\ (8)\ \ \mathit{negStudent}(X) \leftarrow\ \mathit{resas}(X), \mathrm{DL}[\mathit{Student} \uplus \mathit{student};\ \neg \mathit{Student}](X); \\ (9)\ \ \mathit{helps}(Y, X) \leftarrow \mathit{resas}(X), \mathrm{DL}[;\ \mathit{PostDoc}](Y), \mathit{not}\ \mathit{omit}(Y, X); \\ (10)\ \ \mathit{omit}(Y, X) \leftarrow \mathit{helps}(Y, X), \mathrm{DL}[;\ \mathit{PostDoc}](X); \\ (11)\ \ \mathit{visitPostDoc}(Z) \leftarrow \mathrm{DL}[;\ \mathit{VisitPostDoc}](Z); \\ (12)\ \ \mathit{orghelp}(Z1) \leftarrow \mathrm{DL}[;\ \mathit{OrgHelp}](Z1); \\ (13)\ \ \mathit{supports}(Z1, Z) \leftarrow \mathit{orghelp}(Z1), \mathit{visitPostDoc}(Z), \mathit{not}\ \mathit{drop}(Z1, Z); \\ (14)\ \ \mathit{negStudent}(Z1) \leftarrow \mathit{orghelp}(Z1), \mathit{not}\ \mathit{student}(Z1); \\ (15)\ \ \mathit{student}(Z1) \leftarrow \mathit{orghelp}(Z1), \mathrm{DL}[\mathit{NonStudent} \uplus \mathit{negStudent};\ \neg \mathit{NonStudent}](Z1); \\ (16)\ \ \mathit{drop}(Z1, Z) \leftarrow \mathit{supports}(Z1, Z), \mathrm{DL}[;\ \mathit{InternationalStudent}](Z1). \end{array} \right\}$$

Fig. 7. DL-program from LUBM benchmark.

Table 12
LUBM benchmark results.

p	*AS*	*RAS*				
		RAS	*lim* = 20	*limp* = 2	*limc* = 20	*IS*
2 (20)	3.97 (0)[0]	13.98 (0)[20]	38.90 (0)[20]	16.01 (0)[20]	15.24 (0)[20]	15.20 (0)[6]
6 (20)	4.25 (0)[0]	16.16 (0)[20]	115.62 (0)[19]	18.08 (0)[20]	18.63 (0)[19]	11.16 (0)[2]
10 (20)	4.64 (0)[0]	18.95 (0)[20]	245.40 (0)[7]	20.85 (0)[20]	20.79 (0)[4]	9.12 (0)[0]
14 (20)	4.86 (0)[0]	21.50 (0)[20]	236.40 (1)[3]	23.73 (0)[20]	23.50 (0)[1]	9.53 (0)[0]
18 (20)	5.33 (0)[0]	24.86 (0)[20]	230.21 (0)[1]	27.11 (0)[20]	26.86 (0)[0]	10.15 (0)[0]
22 (20)	5.54 (0)[0]	28.21 (0)[20]	228.12 (0)[0]	30.19 (0)[20]	29.93 (0)[0]	10.36 (0)[0]
26 (20)	5.71 (0)[0]	31.50 (0)[20]	222.78 (0)[0]	33.84 (0)[20]	33.26 (0)[0]	10.75 (0)[0]
30 (20)	6.07 (0)[0]	36.88 (0)[20]	225.18 (0)[0]	38.82 (0)[20]	38.47 (0)[0]	11.45 (0)[0]
34 (20)	6.36 (0)[0]	42.18 (0)[20]	241.30 (0)[0]	44.29 (0)[20]	44.01 (0)[0]	12.22 (0)[0]
38 (20)	6.55 (0)[0]	46.07 (0)[20]	245.77 (0)[0]	47.87 (0)[20]	47.64 (0)[0]	12.41 (0)[0]
42 (20)	6.93 (0)[0]	52.50 (0)[20]	255.74 (0)[0]	54.17 (0)[20]	56.91 (0)[0]	12.94 (0)[0]
46 (20)	7.15 (0)[0]	56.98 (0)[20]	276.52 (5)[0]	58.96 (0)[20]	58.47 (0)[0]	13.35 (0)[0]
50 (20)	7.53 (0)[0]	63.96 (0)[20]	276.07 (5)[0]	65.79 (0)[20]	65.50 (0)[0]	14.18 (0)[0]

related defaults (see Fig. 7). Informally, the goal of this program is to construct candidate assignments by identifying postdocs helping students with their research work and organizational staff supporting visiting postdocs with language related issues. From every model of the program a set of candidate assignments satisfying additional side constraints expressed by the rules of the program is extracted.

- The rules (6)–(8) encode the default that research assistants are students unless the contrary is derived.
- The rule (9) assigns postdocs to every research assistant (who is a student by default). In case the "supposed" student has problems, there is always a person to contact, viz. some assigned postdoc; the possible assignments are collected in the *helps*. However, a research assistant may happen to be a visiting postdoc and thus a postdoc (axiom (4) in $\mathcal{O}$); then, no help from another postdoc is needed (rule (10)).
- Visiting postdocs do not need help with their work-related problems, but they need language support, as (being foreigners) they will not know the local language. Hence, a person who can provide organizational help ought to be found for each postdoc. Rule (11) collects all visiting postdocs into a respective predicate, and rule (12) similarly persons capable of providing organizational help. Rule (13) assigns any such person to a visiting postdoc using the *supports* predicate.
- However, not all people who can provide organizational help are equally good in rule (13), and some may be exempted; in particular, rule (16) exempts international students from organizational help.
- As for organizational help, persons are assumed not to be students by default (rules (14)–(15)).

The absence of answer sets for the program is caused by the cyclic dependencies of a literal from its default negation, which manifests in the rules (9)–(10) and (13)–(16). The results of the experiments are given in Table 12. Standard answer set computation outperforms repair answer set computation; thus in this benchmark inconsistency is found faster than the first repair. There are many DL-atoms without input predicates, so called *outer* DL-atoms. In the standard answer set mode, for these atoms all relevant constants are retrieved at an early stage, which speeds up the computation. The restricted repairs are found in this benchmark too, and the results are as expected: the stricter the limit, the less repairs are found and the more time is needed. The last column of Table 12 shows the results for removal restricted to *InternationalStudents*. As one can see, this guided search speeds up the computation but significantly decreases the number of found repairs. Note that allowing deletion of at most 20 facts leads to higher running time than the other restrictions; this is explained by the structure of repairs, which do not involve many different predicates, but the number of facts in each repair is very likely to exceed 20.

7. Discussion

In this section, we discuss extensions of our work for DL-programs on ontologies beyond *DL-Lite*$_{\mathcal{A}}$ and consider related work on inconsistency management in more detail.

7.1. Further work

Our notions of repair and repair answer set naturally generalize to DL-programs over ontologies in other DLs, and similar techniques as above can be employed to compute repair answer sets. Algorithms 2 and 3 are general and work for arbitrary DLs, and similarly Algorithm 4 works for DLs that admit complete (non-ground) support families. In particular, the approach was extended in [34,79] to DL-programs over $\mathcal{EL}$ ontologies; the $\mathcal{EL}$ family [3] includes like the *DL-Lite* family prominent DLs that are tractable and despite limited expressiveness still useful for many application domains. The general complexity results in Sections 3.1 and 3.3 carry over to the core DL $\mathcal{EL}$, assuming that negative concepts assertions are admissible (which do not affect tractability of standard reasoning tasks). In absence of such assertions, and thus of the update operators $\uplus$ and $\Cap$, deciding existence of *flp*-repair answer sets for normal DL-programs drops to NP. However, like for *DL-Lite*$_{\mathcal{A}}$ deciding *weak*- or *flp*-repair answer set existence is NP-hard for DL-programs with simple structure and input-free DL-atoms where deciding answer set existence is tractable [79]. Furthermore, deletion repairs for ORPs over $\mathcal{EL}$ ontologies are intractable, even in absence of negative assertions, and so are the other repair notions in Section 3.5 except bounded $\delta^{\pm}$-change repairs. Intuitively, support sets for DL-atoms over $\mathcal{EL}$ ontologies can involve arbitrarily many assertions, and disabling a support set leads to choosing one of them; thus, hypergraph 2-colorability, which is well-known to be NP-complete [44], can be easily expressed. Complete support families for DL-atoms over $\mathcal{EL}$ can get very large (exponential size) or even infinite in case of cyclic TBoxes (which are though less frequent in practice [43]).

To address these issues, a version of the algorithm for repair answer set computation was given in [34] that operates on incomplete (partial) support families; this algorithm and the underlying framework can be applied to repair DL-programs over ontologies in other DLs as well. It uses hitting sets to disable known support sets of negative DL-atoms and performs evaluation postchecks–if needed–to compensate incompleteness of support families. Moreover, it trades answer completeness for scalability by using minimal hitting sets. A declarative implementation for ontologies in $\mathcal{EL}$ is available on top of dlvhex, where partial support families for DL-atoms are computed by unfolding datalog rewritings of queries over an $\mathcal{EL}$ ontology; for more details, see [34,79]. Finally, we remark that the notion of support set has been fruitfully generalized to HEX programs [38], in which instead of an ontology arbitrary external information sources of computation can be accessed from an answer set program [32] (see also Appendix A).

7.2. Related work

Handling inconsistencies in DL-programs is a rather recent issue, which has been targeted in few works, including [72, 40], and these works focused on inconsistency tolerance. Pührer et al. [72] aimed to avoid answer sets that are non-intuitive due to inconsistency in DL-atoms, by dynamically disabling rules that possibly involve spoiled information. Here the underlying assumption is that the ontology can or should not be changed; for the case where changes are possible, ontology repair was posed as an important open issue. Fink [40] addressed inconsistency of DL-programs due to the lack of stability in models by resorting to semi-stable models based on [31], and combined the resulting paracoherent semantics with paraconsistency techniques for handling classical conflicts (i.e., truth of a formula and its negation) similar as in Description Logics [60]. Semi-stable models repair in a sense the DL-program by changing the data part, but are quite different from repair answer sets: indeed, only addition of data is possible, but no deletion; additions are not restricted to ontology assertions; and noticeably, the additions are treated as unjustified beliefs rather than as facts that are true. Finally, additions must be smallest possible (w.r.t. set inclusion), which leads to a complexity increase that makes reasoning from semi-stable models harder than from repair answer sets.

Like for DL-programs, for other hybrid formalisms inconsistency management has so far concentrated on inconsistency tolerance rather than repair. For instance, Huang et al. [49] presented a four-valued paraconsistent semantics, based on Belnap's logic [8], for hybrid MKNF knowledge bases [66], which are the most prominent tightly coupled combination of rules and ontologies. Inspired by the paracoherent stable semantics from [75], the work [49] was extended in [48] to handle also incoherent MKNF KBs, i.e. programs in which inconsistency arises as a result of dependency of an atom on its default negation in analogy to [40]. Another direction of inconsistency handling for hybrid MKNF KBs is using the three-valued (well-founded) semantics of Knorr et al. [52], which avoids incoherence for disjunction-free stratified programs. Most recently, this has been extended in [50] with additional truth values to evaluate contradictory pieces of knowledge, such that inconsistency can be modeled with a new truth value and non-contradictory knowledge that is only derivable from the inconsistent part of a KB is still considered to be true in the classical sense, or in another view truth which depends on the inconsistent part of a KB is distinguished from truth derivable without involving any contradictory knowledge (also known as suspicious reasoning). However, these works aim at inconsistency tolerance rather than repair, and are geared in spirit to query answering that is inherent to well-founded semantics.

In the context of Description Logics, repairing ontologies has been studied intensively, foremost to handle inconsistency. In particular, Lembo et al. [54] and Bienvenu [12] studied consistent query answering over DL-Lite ontologies based on the repair technique from databases (see [10]). A framework for explaining (negative) query answers under the inconsistency tolerant semantics. In the spirit of minimal change, an inconsistent ontology (with a consistent TBox) is repaired by identifying and eliminating minimal conflict sets causing (i.e., explaining) the inconsistency; this results in maximal deletion repairs. Note that our algorithm *SupRAnsSet* constructs in its search all maximal deletion repairs; in that it is similar to ABox cleaning [64,74] (though in general non-maximal repairs are also computed by our method). However, our setting differs also in other respects fundamentally from those in [54,12]: (i) the ontology is consistent and inconsistency arises only through the interface of a DL-atom; (ii) several DL-atom queries, where each is either an entailment or a non-entailment query, have to be considered en bloc; and (iii) in addition, individual ABox updates are possible.

Calvanese et al. [22] considered explaining negative answers to instance queries and unions of conjunctive queries in *DL-Lite*$_{\mathcal{A}}$, i.e., to give reasons for tuples missing from the output, complementing [16] which considered explanations for positive query answers in *DL-Lite*. They proposed abductive explanations that correspond to repairs by increasing the ABox, and they characterized the computational complexity of deciding explanation existence and other reasoning problems around explanations, for arbitrary and preferred explanations that amount to non-independent σ-selections. In absence of preferences and with empty ABox, this problem can be seen as a special ORP with a single query and empty update sets, and thus contributes a tractable case. On the other hand, the issues (ii) and (iii) in the previous paragraph apply also here and turn ORPs into multi-abduction problems of positive and negative queries with individual ABox additions; it remains unclear how one could readily exploit the existing abduction algorithms to solve such ORPs efficiently.

Repairing inconsistent non-monotonic logic programs is less developed. Sakama and Inoue [76] used extended abduction to delete minimal sets of rules; however, notably also adding rules can remove inconsistency from such a program. This was exploited by Balduccini and Gelfond [7], who proposed consistency-restoring rules that may be added, under Occam's razor, in order to remove inconsistency. Syränen [81] aimed at finding reasons for the absence of answer sets and addressed debugging logic programs based on model-based diagnosis [73], which in a generalized setting was considered by Gebser et al. [45], who provided explanations why interpretations are not answer sets of a program. Repairing rules in a DL-program subsumes repair of ordinary nonmonotonic logic programs, and thus represents a challenge as such, especially if repair goes beyond merely dropping rules. Inconsistent DL-programs can be seen as programs with bugs that need appropriate debugging techniques for fixing. These were studied in [68], where an approach building on [81,45,71] was developed. The idea is to proceed in a user-interactive way by stepping through the rules of the DL-program, and to distinguish at each step a set of active rules along with an intermediate interpretation. Faulty rules are identified if a conflict is reached in the stepping process. It would be interesting to see if stepwise debugging and data repair can be fruitfully combined, which remains for future work.

Our ideas on domain-dependent restrictions on repairs are related to the inconsistency policy for databases discussed e.g. in [78,63], where the authors presented preference-based techniques for repairing databases. In the context of DL-programs, this has not been considered before. The complexity of consistent query answering based on preferred repairs over lightweight ontologies (in particular, in *DL-Lite*$_{\mathcal{R}}$) has been recently studied in [13], where for a number of preferences that amount to non-independent σ-selections intractability was shown, which in most cases is beyond NP.

8. Conclusion

We have considered the issue of repairing DL-programs, which are a well-known loose-coupling combination of nonmonotonic logic rules and Description Logic ontologies, in case of inconsistency, i.e., when no answer set (model) of a DL-program exists. To this end, we have introduced *repair answer sets* based on *repairs*, which change the data part (ABox) of the ontology to gain consistency. We have characterized the computational complexity of repair answer sets, showing that they do not add to the complexity of answer sets (more specifically, to weak and *flp* answer sets) for ontologies in *DL-Lite*$_{\mathcal{A}}$, which is a prominent Description Logic featuring tractable reasoning. This similarly holds for other tractable Description Logics. Indeed, while we concentrated on *DL-Lite*$_{\mathcal{A}}$, our general methodology for restoring consistency can be applied to DL-programs over ontologies in a range of Description Logics; we refer the reader to [79] for further discussion. We have provided selection functions to single out preferred repairs from a candidate set, and we have discussed the benign property of independence which allows for local preferred repair selection (filtering).

We have then extended an in-use algorithm for DL-program evaluation for computing repair answer sets. At the heart of this extension is a generalized *Ontology Repair Problem (ORP)*, which asks for a modified ABox that simultaneously entails respectively non-entails sets of queries, possibly under individual ABox updates. While intractable in general, we have presented several non-trivial tractable cases, among them deletion repairs, which are often applied in practice.

As a naive extension lacks scalability, we developed a new evaluation approach that is based on the novel notion of support set for DL-atoms, and we showed that for DL-programs over *DL-Lite*$_{\mathcal{A}}$ ontologies, a complete support family of such supports sets that allows to completely avoid ontology reasoning during the repair computation can be efficiently constructed. For the experimental evaluation of the approach, we have built a set of benchmarks in different scenarios that involve different ontologies. The experimental results are promising. In particular, for inconsistent DL-programs the repair answer set computation is often faster than standard answer set computation. Furthermore, use-case guided restrictions on repairs often did not introduce much overhead. Overall, the empirical evaluation has revealed a great potential of the novel repair methodology.

8.1. Outlook

We can see several directions for future work. One is to consider repair semantics and computation for DL-programs over Description Logics other than *DL-Lite*$_{\mathcal{A}}$. As mentioned above, for $\mathcal{EL}$ this was done in [34,79], but more expressive DLs can be considered, e.g. the DLs $\mathcal{SHIQ}$, $\mathcal{SHOIN}$, and $\mathcal{SROIQ}$ that are important in the Semantic Web context. Orthogonal to other DLs, additional repair possibilities may be considered besides ABox changes. For repairing DL-rules the works on ASP debugging [41,45,81] may serve as starting point, but the problem is challenging as the search space of possible changes is large. The latter applies to changes of DL-atoms as well, and in both cases restrictions and/or user interaction will be necessary. Another direction would be to consider other formalisms for hybrid knowledge bases, or more general formalisms than DL-programs for combining knowledge bases such as HEX programs [38]. Heterogeneity of external sources in HEX-programs makes both repair and inconsistency-tolerant reasoning very challenging.

Regarding optimizations, learning techniques may be exploited for repair computation, e.g. caching of intermediate repairs/repair answer sets, considering correlation patters between them, and identifying mutual dependence of DL-atoms might be worthwhile. Furthermore, program and repair decomposition can be considered, where a DL-program is split into modular components that can be handled separately, and local repairs for them are combined into a global repair. It remains to be seen, however, to what extent and for which program classes the repair methods can be adapted for a modular setting. As regards ABox change, localization and decomposition methods from databases may be exploited [29].

Another direction are alternative evaluation approaches. Instead of turning answer sets of the replacement program into repair answer sets by suitable changes of the ontology ABox, one could aim at finding repair answer sets incrementally, e.g., by exploiting debugging based on stepping techniques [68]. In a user-interactive mode, one traverses the rules of a DL-program until a conflict is identified. If the latter occurred due to a DL-atom, the ontology ABox is repaired and then the stepping process is continued. While this strategy may not work in general, it can be of interest in restricted settings, e.g. for stratified DL-programs.

On the practical side, providing other independent selection functions apart from deletions (see Section 3) is an important issue, along with means to incorporate domain specific information in the repair process (e.g., protected ontology parts). This calls for convenient representation and effective exploitation of such information with the dlliteplugin.

Appendix A. Supplement to Section 2

This section introduces HEX-programs [38] and explains their correspondence with DL-programs (which are a proper instance of HEX programs). The material is included for the convenience of the reader, as a supplement to ease deeper understanding of the evaluation algorithm for DL-programs, which is in terms of the more general class of HEX programs [30]. However, this appendix is not strictly needed and can be omitted.

A.1. HEX-programs

Apart from the interaction with the DL ontology through a logic program there are other ways of accessing information from different external sources. An important generalization of DL-programs are *HEX-programs* [38], which accommodate a universal bidirectional interface for arbitrary sources of external computation. This is achieved by means of the notion of an external atom. Using such external atoms, whose semantics is abstractly modeled by an input-output relationship, one can access different kinds of information and reasoning in a single program. HEX-programs have been successfully used in various kinds of applications. Some examples include multi-agent systems, rule-based policy specification, distributed SPARQL processing, to mention a few.

We assume that for a given HEX-program the vocabulary consists of mutually disjoint sets $\mathcal{C}$ of constants, $\mathbf{V}$ of variables, $\mathbf{P}$ of predicates, $\mathbf{X}$ of external predicates. Next we recall syntax and semantics of HEX-programs.

Syntax. HEX-programs generalize (disjunctive) extended logic programs under the answer set semantics described earlier with *external atoms*, allowed in the bodies of the rules. External atoms have a list of input parameters (constants or predicate names) and a list of output parameters.

Definition 77 *(External atom).* An *external atom* $a(\vec{Z})$ is of the form

$$\&g[\vec{Y}](\vec{X}), \tag{A.1}$$

where $\&g \in \mathbf{X}$, $\vec{Y} = Y_1, \ldots, Y_\ell$, and $\vec{X} = X_1, \ldots, X_m$, such that $Y_i, X_j \in \mathbf{P} \cup \mathcal{C} \cup \mathbf{V}$, for $1 \le i \le \ell$ and $1 \le j \le m$, and $\vec{Z}$ is the restriction of $\vec{Y}$ and $\vec{X}$ to elements from $\mathbf{V}$.

An external atom is *ground* if $Y_i \in \mathcal{C} \cup \mathbf{P}$ for all $1 \le i \le l$ and $X_j \in \mathcal{C}$ for all $1 \le j \le m$.

Example 78. Consider the external atom $a(\vec{X}) = \&diff[p,q](\vec{X})$, where p and q are predicates. The atom $a(\vec{X})$ computes the set of all elements X, which are in the extension of p but not in the extension of q. □

HEX-programs are defined as follows:

Definition 79 *(HEX-program).* A *HEX-program* consists of rules r of form

$$a_1 \vee \cdots \vee a_n \leftarrow b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m \ , \tag{A.2}$$

where each a_i is an (ordinary) atom, each b_j is either an ordinary atom or an external atom, and $n + m > 0$.

Like for ordinary logic programs, we refer to $H(r) = \{a_1, \ldots, a_n\}$ as the *head* of r, and to $B(r) = \{b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_n\}$ as the *body* of r.

Example 80. Consider the program Π

$$\begin{aligned} d(c) \leftarrow; \qquad & q(c) \ \leftarrow d(c), \&diff[d,p](c); \\ & p(c) \ \leftarrow d(c), \&diff[d,q](c) \end{aligned}$$

Informally, this program implements a choice from $p(c)$ and $q(c)$ □

A program is *ground*, no variables occur in it. For non-ground HEX-programs, a suitable safety conditions allows to use a *grounding procedure* that transforms the program to a ground program with the same answer sets.

Semantics. The semantics of a HEX-program is defined via interpretations I over the Herbrand base, which is naturally generalized from ordinary logic programs as follows:

Definition 81 *(Herbrand base).* The *Herbrand Base* of a HEX-program Π, denoted $HB(\Pi)$ is the set of all atoms constructable from the predicates occurring in Π and the constants from $\mathcal{C}$.

Given a HEX-program Π, satisfaction of (sets of) literals, rules, etc. O w.r.t. an interpretation I over $HB(\Pi)$, denoted $I \models O$, extends naturally from ordinary [46] to HEX-programs, and the satisfaction of a ground external atom $\&g[\vec{y}](\vec{x})$ is more involved. It is given by the value of a $1+|\vec{y}|+|\vec{x}|$-ary Boolean function $f_{\&g}$. Formally,

Definition 82 *(Satisfaction).* Let Π be a HEX-program and $I \subseteq HB(\Pi)$ an interpretation. The satisfaction relation is defined as follows:

- for an ordinary atom b, $I \models b$, if $b \in I$, and $I \not\models b$, if $b \notin I$;
- for a ground external atom $\&g[\vec{y}](\vec{x})$, $I \models \&g[\vec{y}](\vec{x})$, if $f_{\&g}(I, \vec{y}, \vec{x}) = 1$, and $I \not\models \&g[\vec{y}](\vec{x})$, if $f_{\&g}(I, \vec{y}, \vec{x}) = 0$;
- I satisfies an (ordinary or external) literal *not* b, if $I \not\models b$;
- I satisfies a rule of form (A.2), if $I \models a_i$ for some $1 \le i \le k$ or $I \not\models b_i$ for some $1 \le i \le m$ or $I \models b_i$ for some $m < i \le n$;
- I satisfies a ground HEX-program Π (I is a model of Π), if $I \models r$ for all rules r of Π.

The answer sets of HEX-programs are defined in terms of the *flp*-reduct.

Definition 83 *(flp reduct).* Let Π be a HEX-program and let I be an assignment. An *flp-reduct* of Π w.r.t. I is a program $\Pi^I_{flp} = \{r \in \Pi \mid I \models B(r)\}$.

Definition 84 *(flp-answer set).* Given a HEX-program Π, an assignment I is an *flp-answer set* of Π, if I is a $\subseteq$-minimal model of Π^I_{flp}. $AS_{flp}(\Pi)$ denotes the set of all *flp*-answer sets of a HEX-program Π.

Example 85. Recall the HEX-program from Example 80 and consider an assignment $I_1 = \{d(c)\}$. The reduct $\Pi^{I_1}_{flp}$ of Π relative to I_1 is as follows:

$$\Pi^{I_1}_{flp} = \{d(c);\ \ p(c) \leftarrow \&diff[d,q](c)\}.$$

Observe, that I_1 is a minimal model of $\Pi^{I_1}_{flp}$, therefore $I_1 \in AS_{flp}(\Pi)$.

The assignment $I_2 = \{d(c), q(c)\}$ is another *flp*-answer set of Π. Indeed, the *flp*-reduct comprises

$$\Pi^{I_2}_{flp} = \{d(c);\ \ q(c) \leftarrow \&diff[d,p](c)\}.$$

As I_2 is the minimal model of $\Pi^{I_2}_{flp}$, we get that $I_2 \in AS_{flp}(\Pi)$. □

A.2. From HEX-programs to DL-programs

We now provide a correlation between DL-programs and HEX-programs.

Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a DL-program, where $\mathcal{O}$ is a consistent ontology fixed as an external source and $\mathcal{P}$ is a set of DL-rules. DL-atoms are encoded as external atoms of the form $\&DL[c^+, c^-, r^+, r^-, Q](\vec{x})$, where c^+, c^- (r^+, r^-) are binary (resp. ternary) predicates and Q is a string which encodes an ontology query. The query Q is a possibly negated ontology concept or a role name, concept or role subsumption or its negation.

The oracle function of $\&DL$ is defined by

$$f_{\&DL}(I, c^+, c^-, r^+, r^-, \vec{x}) = 1 \iff \mathcal{O} \cup U^I(c^+, c^-, r^+, r^-) \models Q(\vec{x}),$$

where $U^I(c^+, c^-, r^+, r^-)$ is an update to $\mathcal{O}$, specified by the (extension of the) predicates c^+, c^-, r^+, r^-. More specifically, it contains for each $c^+(C, a) \in I$ (resp. $c^-(C, a) \in I$), a concept assertion $C(a)$ (resp. $\neg C(a)$). Updates of roles, generated by the predicates r^+ and r^- are analogous.

Example 86. DL[$Male \uplus boy$; $Male$](X) from Fig. 1 is translated to $\&DL[c^+, c^-, r^+, r^-, Male](X)$, s.t. $\mathcal{P}$ is extended by the rule $c^+(Male, X) \leftarrow boy(X)$, and the predicate c^+ does not occur elsewhere $\mathcal{P}$.

The rule (9) of $\mathcal{P}$ in Fig. 1 corresponds to the following rules in the HEX-program:

$$\mathcal{P} = \left\{ \begin{array}{ll} (9) & hasfather(X, Y) \leftarrow \&DL[c'^+, c'^-, r'^+, r'^-, hasParent](X, Y), \\ & \qquad \&DL[c^+, c^-, r^+, r^-, Male](Y); \\ (9') & c^+(Male, X) \leftarrow boy(X) \end{array} \right\} \qquad \square$$

Appendix B. Proofs of Section 3

Proof of Theorem 18. (i) NP-completeness result for normal Π and $x = weak$.
(Membership) Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a normal DL-program, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. The algorithm of deciding whether $RAS(\Pi) \neq \emptyset$ proceeds as follows: we guess an interpretation I, the values of the DL-atoms and the repair ABox $\mathcal{A}'$. We then check whether I is a repair answer set of Π as follows:

(1) evaluate all DL-atoms over $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ and compare their values with the guessed values;
(2) check whether I is a minimal model of the reduct $\mathcal{P}^{I,\mathcal{O}'}_{weak}$.

The check (1) is feasible in polynomial time, which follows from the Proposition 17. As for the check (2), observe that the reduct $\mathcal{P}^{I,\mathcal{O}'}_{weak}$ is constructable in polynomial time, and it is a normal positive ASP program, which has a single model. Therefore, the check (2) is also polynomial. The above algorithm solves the target problem, which proves its membership in NP.

(Hardness) The NP-hardness is inherited from ordinary normal logic programs, whose repair answer sets coincide with their answer sets; as deciding answer set existence for normal logic programs is NP-hard [61], the result follows.

(ii) Σ_2^P-completeness result for arbitrary Π and $x = weak$.
(Membership) The overall algorithm of deciding the existence of a weak repair answer set proceeds as follows: we guess an interpretation I, values of DL-atoms and an ABox $\mathcal{A}'$ and then check whether:

(1) the real values of DL-atoms over I and $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ coincide with the guessed values;
(2) I is a minimal model of $\mathcal{P}^{I,\mathcal{O}'}_{weak}$.

Like in (i) the check (1) is polynomial. $\mathcal{P}^{I,\mathcal{O}}_{weak}$ is a propositional disjunctive program. Deciding whether I is its minimal model can be verified with a call to an NP oracle, from which the membership in Σ_2^P follows.

(Hardness) Similar like in (i), the hardness results for arbitrary DL-programs and *weakRAS*-existence are inherited from the answer set existence for ordinary disjunctive logic programs.

(iii) Σ_2^P-completeness result for normal Π and $x = flp$.
(Membership) We can guess a repair $\mathcal{A}'$ together with an interpretation I and then check whether I is an *flp*-repair answer set of $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$, where $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$. Constructing the reduct $\mathcal{P}^{I,\mathcal{O}'}_{flp}$ is polynomial, as we only need to pick those rules of Π whose body is satisfied by I, and all DL-atoms can be evaluated in polynomial time. With the reduct $\mathcal{P}^{I,\mathcal{O}'}_{flp}$ at hand we then need to check whether

(1) all values of DL-atoms over $\mathcal{O}'$ coincide with the guessed ones;
(2) I is a minimal model of $\mathcal{P}_{flp}^{I,\mathcal{O}'}$.

The check (1) can be done in polynomial time. For (2) we have that the interpretation I is not a minimal model of $\mathcal{P}_{flp}^{I,\mathcal{O}'}$ iff there exists an interpretation $I' \subset I$ such that $I' \models \mathcal{P}_{flp}^{I,\mathcal{O}'}$. A guess for I' is verifiable in polynomial time, thus deciding whether I is not an answer set of $\mathcal{P}_{flp}^{I,\mathcal{O}'}$ is in NP. From this we get that deciding whether I is an answer set of $\mathcal{P}_{flp}^{I,\mathcal{O}'}$ is in co-NP. Hence for the check (ii) we need to make a call to a co-NP oracle. Since having an oracle for co-NP is equivalent to having an oracle for NP, we get that the overall problem can be solved in $\mathsf{NP}^{\mathsf{NP}} = \Sigma_2^p$.

(Hardness) We prove the Σ_2^p-hardness result by a reduction from deciding validity of a QBF formula

$$\phi = \exists x_1 \dots x_n \forall y_1 \dots y_m E, \quad n, m \geq 1, \tag{B.1}$$

where $E = \chi_1 \vee \dots \vee \chi_r$ is a DNF formula, and each $\chi_k = l_{k_1} \wedge l_{k_2} \wedge l_{k_3}$ is a conjunction of literals over atoms $x_1, \dots, x_n, y_1, \dots, y_m$.

For each atom x_i we introduce a fresh concept X_i, and for each atom y_j we introduce a fresh concept Y_j and a fresh logic program predicate y_j. Furthermore, we introduce an additional fresh predicate w. Given ϕ, we construct $\Pi = \langle \emptyset, \mathcal{A}, \mathcal{P} \rangle$ with $\mathcal{A} = \{X_1(b), \dots, X_n(b)\}$ and $\mathcal{P}$ as follows:

$$\mathcal{P} = \left\{ \begin{array}{ll} (1) & \bot \leftarrow \mathit{not}\ \mathrm{DL}[;\ X_i](b), \mathit{not}\ \mathrm{DL}[;\ \neg X_i](b); \\ (2) & \bot \leftarrow \mathrm{DL}[;\ Y_j](b); \\ (3) & \bot \leftarrow \mathrm{DL}[;\ \neg Y_j](b); \\ (4) & w(b) \leftarrow \mathit{not}\ w(b); \\ (5) & y_j(b) \leftarrow w(b); \\ (6) & w(b) \leftarrow f(l_{k_1}), f(l_{k_2}), f(l_{k_3}) \end{array} \right\},$$

where $f(x_i) = \mathrm{DL}[X_i \uplus w;\ X_i](b)$, $\quad f(y_j) = \mathrm{DL}[Y_j \uplus y_j, Y_j \uplus w;\ Y_j](b)$,
$f(\neg x_i) = \mathrm{DL}[X_i \cup w;\ \neg X_i](b)$, $\quad f(\neg y_j) = \mathrm{DL}[Y_j \cap y_j, Y_j \cup w;\ \neg Y_j](b)$.

Intuitively, the rules of the form (1) of $\mathcal{P}$ ensure that for each x_i at least one of $X_i(b)$ and $\neg X_i(b)$ is present in the repair ABox $\mathcal{A}'$, while the rules (2) and (3) forbid that $Y_j(b)$ resp. $\neg Y_j(b)$ is in $\mathcal{A}'$. The rule (4) forces each consistent *flp*-repair answer set of Π to contain $w(b)$. The rule (5) ensures that the ground atoms of the form $y_j(b)$ are also contained in each repair answer set. Finally, the rules of the form (6) are present in $\mathcal{P}$ for each clause χ_k of ϕ. For each literal l_{k_h} in χ_k these rules have a DL-atom $f(l_{k_h})$ in the body, which poses to the ontology under some updates an instance query corresponding to the literal l_{k_h}.

We now formally show that ϕ is valid iff $\mathit{RAS}_{flp}(\Pi) \neq \emptyset$.

($\Rightarrow$) Let ϕ be valid and let $\nu(\phi)$ be a satisfying assignment, i.e. for all extensions of ν to variables $y_1, \dots, y_m$ it holds that $\nu(\phi)$ is true. From this we construct a repair ABox $\mathcal{A}'$ as follows. If $\nu(x_i) = \mathit{true}$, then $X_i(b) \in \mathcal{A}'$, otherwise $\neg X_i(b) \in \mathcal{A}'$. By construction the repair $\mathcal{A}'$ represents a maximal consistent subset of $\{X_i(b), \neg X_i(b) \mid 1 \leq i \leq n\}$. Therefore, the constraints (1)–(3) are not violated under $\mathcal{A}'$.

We now show that for any interpretation I the body of at least one rule of the form (6) of $\Pi' = \langle \emptyset, \mathcal{A}', \mathcal{P} \rangle$ must be satisfied by I. Let us consider various possibilities for an interpretation I of Π'.

- $I \cap \{y_1(b), \dots, y_m(b)\} = \emptyset$. Let us look at an extension ν' of ν, under which all variables y_j of ϕ are false. Since $\nu'(\phi) = \mathit{true}$, there must exist a clause χ_k, such that $\nu'(\chi_k) = \mathit{true}$. Consider the rule r_k of the form (6) that corresponds to χ_k. The clause χ_k is a conjunction of literals, thus all of its conjuncts over y_j must be negative. We have that each $\neg y_j$ occurring in χ_k corresponds to a DL-atom of the form $f(\neg y_j) = \mathrm{DL}[Y_j \cap y_j, Y_j \cup w;\ \neg Y_j](b)$. As $y_j(b) \notin I$, it holds that $\lambda^I(f(y_j)) = \{\neg Y_j(b)\}$, leading to $I \models^{\mathcal{O}'} f(\neg y_j)$. All DL-atoms of the forms $f(x_i)$ and $f(\neg x_i)$ are satisfied by the construction of $\mathcal{A}'$.
- $I \cap \{y_1(b), \dots y_m(b)\} \neq \emptyset$. Let us look at an extension ν' of ν such that

$$\nu'(y_j) = \begin{cases} \mathit{true}, & \text{if } y_j(b) \in I \\ \mathit{false}, & \text{if } y_j(b) \notin I. \end{cases}$$

  Since $\nu(\phi)$ is a satisfying assignment of ϕ, there must exist a clause χ_k in ϕ such that $\nu'(\chi_k) = \mathit{true}$. Let us look at the rule r_k of the form (6) corresponding to χ_k. For all literals l_{k_h} we have that $I \models f(l_{k_h})$. Indeed, if l_{k_h} is a literal over x_i, then the corresponding DL-atom is true by construction of $\mathcal{A}'$. If $l_{k_h} = y_j$ then as $\nu'(y_j) = \mathit{true}$ we have that $y_j(b) \in I$ and thus $\lambda^I(f(y_j)) = \{Y_j(b)\}$. Similarly, if $l_{k_h} = \neg y_j$, then $\lambda^I(f(\neg y_j)) = \{\neg Y_j(b)\}$. Therefore, $I \models f(l_{k_h})$ for all l_{k_h} occurring in χ_k.

So we have that for any I the body of at least one rule r_k of the form (6) must be satisfied, and hence the rule r_k must be present in the reduct $\mathcal{P}_{flp}^{I,\mathcal{O}'}$. Moreover, if Π' has some *flp*-answer set I, then it must contain $w(b)$ (this follows from $w(b) \leftarrow not\ \ w(b)$), and thus the rule of the form (4) is not in $\mathcal{P}_{flp}^{I,\mathcal{O}'}$. Finally, according to the rules (5) the answer set I should also contain all $y_j(b)$ for $1 \leq j \leq m$.

As there are no other atoms which could be in the answer set, we now show that $I = \{w(b), y_1(b), \ldots, y_m(b)\}$ is a minimal model of $\mathcal{P}_{flp}^{I,\mathcal{O}}$. First, obviously I satisfies all rules of the reduct; we only need to show its minimality. Towards a contradiction, assume that there is an interpretation $I' \subset I$, such that $I' \models \mathcal{P}_{flp}^{I,\mathcal{O}'}$. There are two possibilities: either $w(b) \in I'$ or $w(b) \notin I'$. The former can not be true, as then there is some $y_j(b)$, such that $y_j(b) \notin I'$, and hence for some rule r of the form (5) we have that r is not satisfied by I'. If the latter holds, then we know that there are no rules of the form (6), whose body is satisfied by I'. Consider an extension ν'' of the assignment ν to the atoms y_j, such that $\nu''(y_j) = true$, if $y_j(b) \in I'$, and $\nu''(y_j) = false$ otherwise. We know that $\nu''(\phi) = true$, i.e. there is a disjunct χ_k in ϕ, such that $\nu''(\chi_k) = true$. Let us look at the rule r_k corresponding to the disjunct χ_k. All DL-atoms $f(x_i)$ are satisfied by I', due to the construction of the ABox $\mathcal{A}'$. The DL-atoms of the forms $f(y_j)$ are satisfied by I', because $y_j(b) \in I'$, and thus $\lambda^I(f(y_j)) \models Y_j(b)$. Similarly, the DL-atoms of the form $f(\neg y_j)$ are satisfied, as for them we have that $y_j(b) \notin I'$, and thus $\lambda^I(f(\neg y_j)) \models \neg Y_j(b)$. Hence I' must satisfy $B(r_k)$; but since $w(b) \notin I'$, we have that $I' \not\models r_k$, leading to a contradiction. Therefore, I is indeed an *flp*-repair answer set of Π.

($\Leftarrow$) Let $I \in RAS_{flp}(\Pi)$ be some *flp*-repair answer set of Π with a repair ABox $\mathcal{A}'$, i.e. $I \in AS_{flp}(\langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle)$. Since I is a repair answer set, the repair ABox $\mathcal{A}'$ must contain a nonempty consistent subset of $\{X_i(b), \neg X_i(b)\}$, $1 \leq i \leq n$ because of constraints of the form (1). We construct an assignment ν of ϕ from $\mathcal{A}'$ as follows:

$$\nu(x_i) = \begin{cases} true, & \text{if } X_i(b) \in \mathcal{A}' \\ false, & \text{if } \neg X_i(b) \in \mathcal{A}'. \end{cases}$$

We now show that ν is a satisfying assignment of ϕ, i.e. for any extension ν' of ν to the values of y_j, we have that $\nu'(\phi) = true$. Towards a contradiction, assume that this is not the case, i.e. there exists an extension ν' of ν to the values of y_j, such that $\nu'(\phi) = false$, that is $\nu'(\chi_k) = false$ for all clauses χ_k of ϕ.

Let us now look at the interpretation I' of Π', such that $y_j(b) \in I'$, if $\nu'(y_j) = true$ and $y_j(b) \notin I'$, if $\nu'(y_j) = false$. We know that $I \supset I'$ is a minimal model of $\mathcal{P}_{flp}^{I,\mathcal{O}'}$. Therefore, it must hold that $I' \not\models r$ for some rule r of $\mathcal{P}_{flp}^{I,\mathcal{O}'}$, i.e. $I' \models B(r)$, but $I' \not\models H(r)$. Observe that the reduct $\mathcal{P}_{flp}^{I,\mathcal{O}'}$ contains only the rules (5) and (6). Since $w(b) \notin I'$ by construction, the rule r that I' does not satisfy can not be of the form (5), hence it must be of the form (6). Let us look at the corresponding clause χ_k in ϕ. By our assumption $\nu'(\chi_k) = false$, i.e. there is a conjunct l_{k_h} in χ_k, such that $\nu'(l_{k_h}) = false$. We distinguish the following cases:

- l_{k_h} is a literal over x_i. We know that $\lambda^{I'}(f(l_{k_h})) = \emptyset$, because $w(b) \notin I$. Thus it must be true that $\mathcal{A}' \models f(l_{k_h})$. Since $\mathcal{A}'$ is a repair, by Definition 26 it must be consistent. Thus the query of $f(l_{k_h})$ must be explicitly present in $\mathcal{A}'$, i.e. $X_i(b) \in \mathcal{A}'$, if $l_{k_h} = x_i$; $\neg X_i(b) \in \mathcal{A}'$, if $l_{k_h} = \neg x_i$. However, then by construction of ν' we have that $\nu(l_{k_h}) = true$, which leads to a contradiction.
- l_{k_h} is a literal over y_j. There are two possibilities: either $l_{k_h} = y_j$ or $l_{k_h} = \neg y_j$.
  - First suppose that $l_{k_h} = y_j$. The corresponding DL-atom $f(y_j) = \mathrm{DL}[Y_j \uplus y_j, Y_j \uplus w;\ Y_j](b)$ is true under I' by our assumption. Since the repair ABox $\mathcal{A}'$ is consistent and does not contain any concepts of the form $Y_j(b)$, it must hold that $\lambda^{I'}(f(y_j)) \models Y_j(b)$. Observe that $w(b) \notin I'$, thus it must be true that $y_j(b) \in I'$; however, then $\nu'(l_{k_h}) = true$, leading to a contradiction.
  - Now assume that $l_{k_h} = \neg y_j$. We have that $I' \models f(\neg y_j)$, where $f(\neg y_j) = \mathrm{DL}[Y_j \cap\!\!\!\!- \, y_j, Y_j \uplus w;\ \neg Y_j](b)$. It must hold that $\lambda^{I'}(f(\neg y_j)) \models \neg Y_j(b)$, and hence $y_j(b) \notin I'$ since $w(b) \notin I'$. Therefore, $\nu'(y_j) = false$, i.e. $\nu'(l_{k_h}) = true$, contradicting our assumption.

We have shown that ν' is a satisfying assignment for ϕ for each extension of ν to variables y_j, from which the validity of ϕ follows. $\square$

Proof of Proposition 28. A guess for $\mathcal{A}'$ is verifiable in polynomial time, as deciding all $\langle \mathcal{T}, \mathcal{A}' \cup U_i \rangle \models Q_i$ is polynomial in *DL-Lite*$_{\mathcal{A}}$ [21]. NP-hardness is shown by a reduction from SAT instances $\phi = \chi_1 \wedge \cdots \wedge \chi_m$ over atoms $x_1, \ldots, x_n$. We construct the ORP $\mathcal{R} = \langle \langle \mathcal{T}, \emptyset \rangle, D_1, D_2 \rangle$, using concepts $X_i, \bar{X}_i$ for the x_i, C_j for the χ_j, and a fresh concept ν as follows:

- $\mathcal{T} = \{X_j \sqsubseteq C_i, \bar{X}_{j'} \sqsubseteq C_i \mid 1 \leq i \leq m, x_j \in \chi_i, \neg x_{j'} \in \chi_i\}$
- $D_1 = \{\langle \emptyset, C_i(b) \rangle, \langle U_i, \neg C_i(b) \rangle, \langle V_j, A(b) \rangle \mid 1 \leq i \leq m,\ 1 \leq j \leq n\}$, where $U_i = \{\bar{X}_k(b), X_{k'}(b) \mid x_k \in \chi_i, \neg x_{k'} \in \chi_i\}$, $1 \leq i \leq m$, and $V_j = \{\neg X_j(b), \neg \bar{X}_j(b)\}$, $1 \leq j \leq n$; and
- $D_2 = \{\langle \emptyset, \neg C_i(b) \rangle \mid 1 \leq i \leq m\} \cup \{\langle \emptyset, A(b) \rangle\}$.

Intuitively, by D_2 a repair $\mathcal{A}'$ must not contain $\neg C_i(b)$ nor $A(b)$, and must be consistent. By D_1 the repair must entail $C_i(b)$. Therefore, for each i, the ABox $\mathcal{A}'$ must contain some X_k (resp. $\bar{X}_k$), such that $X_k \sqsubseteq C_i$ (resp. $\bar{X}_k \sqsubseteq C_i$). Moreover, adding either U_i or V_j to $\mathcal{A}'$ causes inconsistency. The former implies that $\mathcal{A}'$ contains some $\neg\bar{X}_k(b)$ (resp. $\neg X_k(b)$) such that $x_k \in \chi_k$ ($\neg x_k \in \chi_k$), and the latter implies that at least one of $\neg X_k(b)$ and $\neg\bar{X}_k(b)$ must be in $\mathcal{A}'$ for all $1 \le k \le n$. Since in addition the ABox is consistent as argued above, it can not contain both $\neg\bar{X}_k$ and $\neg X_k$ thus $\mathcal{A}'$ represents a consistent choice of literals that satisfies ϕ.

We formally show that ϕ is satisfiable iff $\mathcal{R}$ has a repair.

($\Rightarrow$) Let ν be a satisfying assignment for ϕ. We construct a repair ABox $\mathcal{A}'$ for $\mathcal{R}$ as follows: if a variable x_k is set to true in the satisfying assignment of ϕ, then we add $X_k(b)$ and $\neg\bar{X}_k(b)$ to the ABox $\mathcal{A}'$, otherwise, i.e. if x_k is set to false in ν, we add $\bar{X}_k(b)$ and $\neg X_k(b)$ to $\mathcal{A}'$. We now verify whether the constructed ABox is indeed a repair for $\mathcal{R}$ by checking whether it satisfies the conditions (i) to (iii) of Definition 26.

(i) $\mathcal{T} \cup \mathcal{A}'$ is consistent, since ν is a consistent set of literals (not both $X_k(b)$ and $\neg X_k(b)$ (resp. $\bar{X}_k(b), \neg\bar{X}_k(b)$) can be present in $\mathcal{A}'$).
(ii) We check whether for all $\langle U_i^1, Q_i^1\rangle \in D_1$ it holds that $\mathcal{T} \cup \mathcal{A}' \cup U_i^1 \models Q_i^1$. Let us first consider $\langle \emptyset, C_i\rangle$, $1 \le i \le m$. Observe that ν is a satisfying assignment of ϕ, therefore each clause of ϕ is satisfied under ν. Thus, for each clause either there exists a variable x_j occurring as a disjunct in the clause C_i positively and being set to *true* in the satisfying assignment ν or occurring negatively as a disjunct in C_i and being set to false in ν. By construction of $\mathcal{A}'$, we have that $\mathcal{T} \cup \mathcal{A}' \models C_i(b)$ for all $1 \le i \le m$ due to the inclusion $X_j \sqsubseteq C_i$ (resp. $\bar{X}_j \sqsubseteq C_i$). Similarly, we have that for all U_i, $\mathcal{A}' \cup U_i$ is inconsistent and, therefore trivially entails $\neg C_i(b)$. Finally, since the assignment ν is full, each x_i has a truth value. Hence, due to the form of updates V_j, we have that $\mathcal{A}' \cup V_j$ is inconsistent for all j, and thus the queries $A(b)$ are also entailed.
(iii) It is left to show that for all $\langle U_i^2, Q_i^2\rangle \in D_2$ we have that $\mathcal{T} \cup \mathcal{A}' \cup U_i^2 \not\models Q_i^2$. The latter holds since the ontology $\langle \mathcal{T}, \mathcal{A}'\rangle$ is consistent, and there is no way to derive either $\neg C(b)$ or $A(b)$ by means of the TBox axioms and the facts in $\mathcal{A}'$.

The above shows that the ABox $\mathcal{A}'$ is indeed a solution to the $\mathcal{R}$.

($\Leftarrow$) Now assume that there exists an ABox $\mathcal{A}'$ that is a solution to $\mathcal{R}$. We show that then the formula ϕ is satisfiable. First since $\mathcal{T} \cup \mathcal{A}' \cup V_j \models A(b)$, $\mathcal{T} \cup \mathcal{A}' \not\models A(b)$ and $\mathcal{T} \cup V_j \not\models A(b)$, we have that $\mathcal{T} \cup \mathcal{A}' \cup V_j$ must be inconsistent. Moreover, as $\mathcal{T} \cup \mathcal{A}' \not\models \neg C_i(b)$, we know that the inconsistency must occur due to the facts $X_j(b)$, $\bar{X}_j(b)$. Therefore, for each j either $\neg X_j(b)$ or $\neg\bar{X}_j(b)$ must be in $\mathcal{A}'$. Observe now, that due to $\langle \emptyset, C_i(b)\rangle, \langle U_i, \neg C_i(b)\rangle \in D_1$, the ABox $\mathcal{A}'$ must contain such $X_j(b)$ (resp. $\bar{X}_j(b)$), that x_j (resp. $\neg x_j$) is a disjunct in the clause χ_i. Moreover, due to $\langle U_i, \neg C_i(b)\rangle$ for some k such that $x_k \in \chi_i$ (or $\neg x_k \in \chi_i$), it must hold that $\neg\bar{X}_k(b)$ (resp. $\neg X_k(b)$) is in $\mathcal{A}'$. The above argument shows that the ABox $\mathcal{A}'$ encodes a satisfying assignment ν for ϕ: if $X_i(b) \in \mathcal{A}'$, then $\nu(x_i) = true$; if $\bar{X}_i(b) \in \mathcal{A}'$, then $\nu(x_i) = false$. □

Proof of Theorem 29. NP-hardness of an ORP holds by a reduction from SAT. Given $\phi = \chi_1 \wedge \cdots \wedge \chi_m$ on atoms $x_1, \ldots, x_n$, we construct $\mathcal{R} = \langle\langle \emptyset, \emptyset\rangle, D_1, D_2\rangle$, with concepts $X_j, \bar{X}_j$ for the x_j and a fresh concept A, such that

- $D_1 = \{\langle U_i, A(b)\rangle, \langle V_j, A(b)\rangle \mid 1 \le i \le m, 1 \le j \le n\}$, where $U_i = \{\bar{X}_j(b),\ X_{j'}(b) \mid x_j \in \chi_i, \neg x_{j'} \in \chi_i\}$, $1 \le i \le m$ and $V_j = \{\neg X_j(b), \neg\bar{X}_j(b)\}$, $1 \le j \le n$, and
- $D_2 = \{\langle \emptyset, A(b)\rangle\}$.

Intuitively, by D_2 a repair $\mathcal{A}'$ must not contain $A(b)$, and by D_1 adding either (i) U_i or (ii) V_j to $\mathcal{A}'$ causes inconsistency. By (i) $\mathcal{A}'$ must contain at least one $\neg\bar{X}_j(b)$ (resp. $\neg X_j(b)$) such that $x_j \in \chi_i$ ($\neg x_j \in \chi_i$), and by (ii) at least one of $X_j(b), \bar{X}_j(b)$ must be in $\mathcal{A}'$. Furthermore, D_2 forbids both $X_j(b), \neg X_j(b)$ (resp. $\bar{X}_j(b), \neg\bar{X}_j(b)$) to be in $\mathcal{A}'$. Thus $\mathcal{A}'$ encodes a consistent choice of literals that satisfies ϕ. □

Proof of Theorem 37. The guess of the repair $\mathcal{A}' \subseteq \mathcal{A}$ out of 2^n candidates, where $n = |\mathcal{A}|$, is verifiable in polynomial time. The NP-hardness for the cases (i) and (ii) is proved separately.

(i) NP-hardness for (i) is shown by a reduction from SAT instances $\phi = \chi_1 \wedge \cdots \wedge \chi_m$ over atoms $x_1, \ldots, x_n$. We construct the ORP $\mathcal{R} = \langle\langle \mathcal{T}, \emptyset\rangle, D_1, D_2\rangle$, where all U_i^k are empty for $k \in \{1, 2\}$. We use concepts $X_j, \bar{X}_j, X'_j$ for the x_j, C_i for the χ_i as follows:
   - $\mathcal{T} = \{X_j \sqsubseteq C_i,\ \bar{X}_{j'} \sqsubseteq C_i, \mid x_j \in \chi_i, \neg x_{j'} \in \chi_i, 1 \le i \le m\} \cup \{\bar{X}_k \sqsubseteq \neg X'_k \mid 1 \le k \le n\}$
   - $\mathcal{A} = \{X_j(b),\ \bar{X}_j(b) \mid 1 \le j \le n\}$
   - $D_1 = \{\langle \emptyset, C_i(b)\rangle \mid 1 \le i \le m\}$,
   - $D_2 = \{\langle \emptyset, \neg(X_j \sqsubseteq X'_j)\rangle \mid 1 \le j \le n\}$.

   Intuitively, the queries in D_1 ensure that at least one $X_j(b)$ (resp. $\bar{X}_j(b)$) is present in the ontology ABox, such that x_j (resp. $\neg x_j$) is a disjunct in χ_i. The queries in D_2 forbid both $X_j(b)$ and $\bar{X}_j(b)$ to be in the ABox, which is expressed by the non-containment query $\neg(X_j \sqsubseteq X'_j)$ and TBox axioms of the form $\bar{X}_j \sqsubseteq \neg X'_j$. Therefore, the solution to $\mathcal{R}$ encodes a satisfying assignment of ϕ.

   We now formally prove that ϕ is satisfied iff σ_{del} solution for $\mathcal{R}$ exists.

(⇒) Let ϕ be satisfiable, and let ν be a satisfying assignment of ϕ. From this we construct a solution $\mathcal{A}'$ to $\mathcal{R}$ as follows: if $\nu(x_j) = true$, then $X_j(b) \in \mathcal{A}'$, otherwise, $\bar{X}_j(b) \in \mathcal{A}'$. The ontology $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ is clearly consistent. Assume towards a contradiction that $\mathcal{A}'$ is not a solution to $\mathcal{R}$. That is, either (1) D_1 contains a tuple $\langle \emptyset, Q_i^1 \rangle$, such that $\mathcal{O}' \not\models Q_i^1$ or (2) some $\langle \emptyset, Q_j^2 \rangle$ exists in D_2, such that $\mathcal{O}' \models Q_j^2$. If (1) holds then $C_i(b)$ is not entailed for some i from $\mathcal{O}'$. That means that there is a conjunct $\chi_i \in \phi$, such that for none of its disjuncts x_j (resp. $\neg x_j$) we have the corresponding assertion $X_j(b)$ (resp. $\bar{X}_j(b)$) in $\mathcal{A}'$. Hence by construction none of the literals in χ_i is true under ν, meaning that $\nu(\chi_i) = false$ and thus $\nu(\phi) = false$, i.e. contradiction. If (2) holds then for some j we have that $X_j(b)$ and $\neg X_j'(b)$ are entailed by $\mathcal{O}'$. As by construction of $\mathcal{A}'$ it holds that $\mathcal{A}' \subseteq \mathcal{A}$, both $X_j(b)$ and $\neg X_j'(b)$ are entailed only if $X_j(b), \bar{X}_j(b) \in \mathcal{A}'$. This can not happen, as $\mathcal{A}'$ is built from a satisfying assignment ν of ϕ, and thus it represents a consistent set of values for x_j. Hence we arrived at a contradiction.

(⇐) Let $\mathcal{A}'$ be a σ_{del} solution to $\mathcal{R}$. From this we construct a satisfying assignment ν for ϕ as follows:

$$\nu(x_j) = \begin{cases} true, & \text{if } X_j(b) \in \mathcal{A}' \text{ or } X_j(b), \bar{X}_j(b) \notin \mathcal{A}' \\ false, & \text{if } \bar{X}_j(b) \in \mathcal{A}'. \end{cases}$$

We show that $\nu(\phi) = true$. Observe that for every C_i there must exist X_j (resp. $\bar{X}_j$), such that $X_j \sqsubseteq C_i$ (resp. $\bar{X}_j \sqsubseteq C_i$) due to the tuples $\langle \emptyset, C_i(b) \rangle$ in D_1 and the fact that $\mathcal{A}' \subseteq \mathcal{A}$. Thus by construction of ν in each clause χ_i some disjunct is true. It is left to show that ν is well defined, i.e. it is not the case that (i) either $\nu(x_j) = true$ or $\nu(x_j) = true$ is defined for every j, and (ii) it is not the case that $\nu(x_j) = true$ and $\nu(x_j) = false$ for some j. In other words we need to show that $\nu(x_j) \neq \nu(\neg x_j)$. Towards a contradiction suppose that this is not the case. Then for some i it holds that $\nu(x_j)$ and $\nu(\neg x_j)$ have the same value. Then $X_j(b)$ and $\bar{X}_j(b)$ are entailed from $\mathcal{O}$ for some j, and therefore $X_j'(b)$ is also entailed from $\mathcal{O}$ due to $\bar{X}_j \sqsubseteq X_j' \in \mathcal{T}$. However, this means that $\mathcal{O}' \models \neg(X_j \sqsubseteq X_j')$, which is forbidden by the respective tuple $\langle , \neg(X_j \sqsubseteq X_j') \rangle$ in D_2. The latter means that $\mathcal{A}'$ is not a solution to $\mathcal{R}$, leading to a contradiction. Thus ν is a satisfying assignment of ϕ.

(ii) NP-hardness for (ii) is shown by a reduction from monotone not-all-equal SAT (NAE-SAT) instances $\phi = \chi_1 \wedge \cdots \wedge \chi_m$ over atoms $x_1, \ldots, x_n$ [44]. In monotone NAE-SAT, all occurrences of literals in clauses are positive, but a formula is "satisfied" only if there is an assignment under which both a literal assigned to true and a literal assigned to false occur in each clause. We construct ORP $\mathcal{R} = \langle \langle \emptyset, \emptyset \rangle, D_1, D_2 \rangle$, using concepts $X_j, \bar{X}_j$ for the x_j, C_i for the χ_i as follows:

– $\mathcal{A} = \{X_j(b), \bar{X}_j(b) \mid 1 \leq j \leq n\}$,
– $D_1 = \{\langle \{\neg X_j(b) \mid x_j \in \chi_i\}, C_i(b) \rangle, \langle \{\neg \bar{X}_{j'}(b) \mid x_{j'} \in \chi_i\}, C_i(b) \rangle \mid 1 \leq i \leq m\}$,
– $D_2 = \{\langle \emptyset, \neg(X_j \sqsubseteq \neg \bar{X}_j) \rangle, \langle \emptyset, C_i(b) \rangle \mid 1 \leq j \leq n, 1 \leq i \leq m\}$.

Intuitively, the queries Q_i^1 can only be satisfied if the repair ABox $\mathcal{A}'$ is inconsistent with the updates U_i^1, as $\mathcal{T} = \emptyset$ and explicit presence of $C_i(b)$ in $\mathcal{A}'$ is forbidden by tuples $\langle \emptyset, C_i(b) \rangle \in D_2$. Therefore, for every χ_i some $X_j(b) \in \mathcal{A}'$ must exist such that x_j is a conjunct in χ_i, which is ensured by $\langle \{X_j \mid x_j \in \chi_i\}, C_i \rangle \in D_1$. However, also some $\bar{X}_{j'}(b)$ must be in $\mathcal{A}'$, such that $x_{j'} \in \chi_i$, which is ensured by $\langle \{X_{j'} \mid x_{j'} \in \chi_i\}, C_i \rangle \in D_1$. By $\langle \emptyset, \neg(X_j \sqsubseteq \bar{X}_j) \rangle$, the indices j and j' must be different, thus the repair ABox encodes a consistent choice of truth values for variables in ϕ, corresponding to a satisfying assignment of ϕ.

We now formally show that ϕ is a positive instance of monotone NAE-SAT iff the $\mathcal{R}$ has some solution.

(⇒) Let ϕ be a positive instance of monotone NAE-SAT, and let ν be the witnessing assignment. From this we construct the solution $\mathcal{A}'$ to $\mathcal{R}$ as follows. $X_i(b) \in \mathcal{A}'$, if $\nu(x_j) = true$, and $\bar{X}_j(b) \in \mathcal{A}'$, if $\nu(x_j) = false$. Since for every clause χ_i some $x_j \in \chi_i$ must be set to true, we have that some $X_j(b) \in \mathcal{A}'$, and hence the query of $\langle \{\neg X_j(b) \mid x_j \in \chi_i\}, C_i(b) \rangle \in D_1$ is satisfied by inconsistency. Similarly, queries of tuples $\langle \{\neg \bar{X}_j(b) \mid x_j \in \chi_i\}, C_i(b) \rangle \in D_1$ are satisfied, as at least one x_j in χ_i is set to false, and by construction the respective $\bar{X}_j(b)$ is in $\mathcal{A}'$. The queries in D_2 are satisfied, since ν represents a consistent choice of values for x_j, and thus both $X_j(b)$ and $\bar{X}_j(b)$ can not be present in $\mathcal{A}'$.

(⇐) Let $\mathcal{A}'$ be a solution to the $\mathcal{R}$. From this we construct the assignment of ϕ as follows.

$$\nu(x_j) = \begin{cases} true, & \text{if } X_j(b) \in \mathcal{A}', \\ false, & \text{if } \bar{X}_j(b) \in \mathcal{A}'. \end{cases}$$

Since C_i can not be in $\mathcal{A}'$ by $\langle \emptyset, C_i \rangle \in D_2$ and $\mathcal{T} = \emptyset$, we have that all queries in D_1 are entailed by inconsistency introduced by the updates, and hence in every clause at least one of x_j must be true and at least one $x_{j'}$ must be false. Furthermore, the assignment ν represents a consistent set of values for x_j by construction, since for all j not both $X_j(b)$ and $\bar{X}_j(b)$ can be in $\mathcal{A}'$ due to $\langle \emptyset, \neg(X_j \sqsubseteq \bar{X}_j) \rangle \in D_2$. □

Proof of Lemma 35. By Proposition 5 inconsistency is introduced in a *DL-Lite*$_{\mathcal{A}}$ ontology with at most 2 ABox assertions, i.e. for every inconsistent $\mathcal{A} \cup \mathcal{T}$, an ABox $\mathcal{A}' \subseteq \mathcal{A}$ exists, such that $|\mathcal{A}'| \leq 2$, $\mathcal{A}' \cup \mathcal{T} \models \alpha$ and $\mathcal{A}' \cup \mathcal{T} \models \neg\alpha$ for some assertion α. We have, either $|\mathcal{A}'| = 1$, in which case the result is obtained, or there are consistent ABoxes $\mathcal{A}'' \subset \mathcal{A}$ and $\mathcal{A}''' \subset \mathcal{A}$, such that $\mathcal{A}'' \cup \mathcal{T} \models \alpha$ and $\mathcal{A}''' \cup \mathcal{T} \models \neg\alpha$. Again we have $|\mathcal{A}''| = |\mathcal{A}'''| = 1$. □

Proof of Theorem 36. The proof exploits the property of $\not\sqsubseteq$ DL-queries established in Lemma 35, which states that at most one assertion α from $\mathcal{A}$ is sufficient to derive the query.

Now if $\mathcal{T} \cup U_j^i \models Q_j^i$, we can drop $\langle U_j^i, Q_j^i \rangle$ from $\mathcal{R}$ if i=1, and stop if $i = 2$ as no repair exists. Otherwise, we let the set $Supp_j^i$ of Q_j^i contain all assertions α such that $\mathcal{T} \cup \{\alpha\} \cup U_j^i \models Q_j^i$. Then, any repair $\mathcal{A}'$ must fulfill $\mathcal{A}' \cap Supp_j^1 \neq \emptyset$ for each j (i.e., be a *hitting set*), and must be disjoint with each $Supp_{j'}^2$. Let then $\mathcal{S}_j := (Supp_j^1 \cap \mathcal{A}) \setminus \bigcup_{j'} Supp_{j'}^2$. A σ_{del}-repair $\mathcal{A}'$ exists iff each $\mathcal{S}_j$ is nonempty; the hitting sets of the $\mathcal{S}_j$ are all the σ_{del}-repairs. The construction of the $\mathcal{S}_j$ and the check can be done in polynomial time, thus the overall problem is tractable. Note that, furthermore, the (possibly exponentially many) σ_{del}-repairs can be output in total polynomial time. □

Proof of Theorem 39. We prove the statement for the case when few negative assertions are added to the ABox, i.e. $|\mathcal{A}'^- \setminus \mathcal{A}| \leq k$. The case when few positive assertion are added to the ABox, i.e. $|\mathcal{A}'^+ \setminus \mathcal{A}| \leq k$ is completely symmetric, and our proof can be easily adapted to treat is it as well.

We provide an extension of the method for deletion repairs. Assuming that $\langle \mathcal{T}, \mathcal{A} \rangle$ is consistent (otherwise no σ_{bop}-repair exists), we proceed as follows:

1. Like for deletion repairs, we compute the sets $Supp_j^i$. We simplify $\mathcal{O}\Pi_{supp}\mathcal{P}$, resp. quit if no repair can exist, checking also whether $Supp_j^i \cap \mathcal{A} \neq \emptyset$ (as then Q_j^i is entailed). More specifically, whenever $U_j^i \cup \mathcal{A} \cup \mathcal{T} \models Q_j^i$ or $Supp_j^i \cap \mathcal{A} \neq \emptyset$,
   - we drop $\langle U_j^i, Q_j^i \rangle$ from D_i, if $i = 1$, and
   - we quit, if $i = 2$.
2. We then let $\mathcal{S}_j = Supp_j^1 \setminus (\mathcal{A} \cup \bigcup_{j'} Supp_{j'}^2)$. Similar as in the proof of Theorem 36, the σ_{bop}-repairs are then of the form $\mathcal{A}' = \mathcal{A} \cup \mathcal{H}$ where $\mathcal{H}$ is a hitting set of the $\mathcal{S}_j$, but we must ensure that $\langle \mathcal{T}, \mathcal{A}' \rangle$ is consistent as $\mathcal{H}$ consists of new assertions.
3. We choose a set $\mathcal{H}^- \subseteq \bigcup_j \mathcal{S}_j$ of at most k negative assertions, which is a partial hitting set, and check that $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{H}^- \rangle$ is consistent. If yes, we remove $\mathcal{S}_j$ if it intersects with $\mathcal{H}^-$ and remove otherwise from $\mathcal{S}_j$ each positive assertion α such that $\neg\alpha$ is entailed by $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{H}^- \rangle$, and all negative assertions.
4. Then, for every hitting set $\mathcal{H}^+$ of $\mathcal{S}_j'$, the ABox $\mathcal{A}' = \mathcal{A} \cup \mathcal{H}^- \cup \mathcal{H}^+$ is a σ_{bop}-repair. On the other hand, some σ_{bop}-repair with few negative additions exists only if some choice for $\mathcal{H}^-$ succeeds.

The crucial point for the correctness of this method is that, if $\mathcal{T}$ has no disjointness axioms, by adding to $\mathcal{A} \cup \mathcal{H}^-$ positive assertions $\mathcal{H}^+$ we can not infer new negative assertions, unless inconsistency emerges; this is exploited in Step 3, which limits the candidate space for positive hitting sets *a priori*.

We now show the correctness of the proposed algorithm formally. Suppose that given the Ontology Repair Problem $\mathcal{R} = \langle \mathcal{O}, D_1, D_2 \rangle$ as an input to the algorithm from above, the ABox $\mathcal{A}'$ was produced as the output after execution of the Steps 1-3. We prove that the ABox $\mathcal{A}'$ is indeed a σ_{bop}-repair for $\mathcal{R}$, i.e. we prove that the conditions that a σ_{bop}-repair needs to satisfy are indeed satisfied by $\mathcal{A}'$.

(i) $\mathcal{T} \cup \mathcal{A}' \cup U_j^1 \models Q_j^1$ for all $\langle U_j^1, Q_j^1 \rangle \in D_1$. Towards a contradiction, suppose that there is some $\langle U_{j_i}^1, Q_{j_i}^1 \rangle \in D_1$, such that $\mathcal{A}' \cup \mathcal{T} \cup U_{j_i}^1 \not\models Q_{j_i}^1$. We know that by construction, it either holds that (1) $U_{j_i}^1 \cup \mathcal{T} \models Q_{j_i}^1$; (2) $\mathcal{A} \cap Supp_{j_i}^1$, i.e. $\mathcal{A} \cup \mathcal{T} \models Q_{j_i}^1$; (3) $\mathcal{H}^- \subseteq \mathcal{A}'$ hits $\mathcal{S}_{j_i}$ or (4) $\mathcal{H}^+ \subseteq \mathcal{A}'$ hits $\mathcal{S}_{j_i}$. For (1) and (2) we immediately get a contradiction. For (3) it holds that $\mathcal{H}^- \cap Supp_{j_i}^1 \neq \emptyset$. Therefore, there is $\alpha \in \mathcal{A}'$, such that $\{\alpha\} \cup U_j^1 \cup \mathcal{T} \models Q_{j_i}^1$.

(ii) $\mathcal{T} \cup \mathcal{A}' \cup U_{j'}^2 \not\models Q_j'^2$ for all $\langle U_{j'}^2, Q_{j'}^2 \rangle \in D_2$. To the contrary, assume that there exists some j_i', such that $\mathcal{T} \cup \mathcal{A}' \cup U_{j_i'}^2 \models Q_{j_i'}^2$. There are several possibilities: (1) $U_{j_i'}^2 \cup \mathcal{T} \models Q_{j_i'}^2$; (2) there is $\alpha \in \mathcal{A}$, such that $\{\alpha\} \cup U_{j_i'}^2 \cup \mathcal{T} \models Q_{j_i'}^2$; (3) there is $\alpha \in \mathcal{H}^-$, such that $\{\alpha\} \cup \mathcal{T} \cup U_{j_i'}^2 \models Q_{j_i'}^2$; (4) there is $\alpha \in \mathcal{H}^+$, such that $\{\alpha\} \cup \mathcal{T} \cup U_{j_i'}^2 \models Q_{j_i'}^2$. Observe that if (1) or (2) were the case, then the algorithm would terminate at Step 1, and no repair $\mathcal{A}'$ would be in the output. For the case (3) we have that $\mathcal{H}^- \cap Supp_{j_i'}^2 \neq \emptyset$. However, according to the Step 3 of our algorithm, it holds that $\mathcal{H}^- \subseteq \bigcup_j (Supp_j^1 \setminus (\mathcal{A} \cup \bigcup_{j'} Supp_{j'}^2))$, meaning that $\mathcal{H}^- \cap Supp_{j_i'}^2 = \emptyset$, which leads to a contradiction.

(iii) $\mathcal{A}' \supseteq \mathcal{A} \mid |\mathcal{A}'^- \setminus \mathcal{A}| \leq k$, i.e. there are at most k negative assertions in the ABox $\mathcal{A}'$.
Finally, we show that the number of negative assertions in $\mathcal{A}' \setminus \mathcal{A}$ is indeed bounded by k. Towards a contradiction, suppose that there are more then k negative assertions in $\mathcal{A}' \setminus \mathcal{A} = \mathcal{H}^+ \cup \mathcal{H}^-$. According to the Step 3 of our algorithm, it holds that $\mathcal{H}^-$ contains at most k negative assertions. Therefore, the rest of the negative assertions must be in $\mathcal{H}^+$. The set $\mathcal{H}^+$ is constructed at Step 4 as a hitting set of sets $\mathcal{S}_j$, which due to the Step 3 contain only positive assertions. Therefore, there are no negative assertions in the set $\mathcal{H}^+$, moreover $\mathcal{T} \cup \mathcal{H}^+ \cup \mathcal{H}^-$ infers only at most k negative assertions, since $\mathcal{T}$ contains only positive inclusions and $\mathcal{A} \cup \mathcal{H}^+ \cup \mathcal{H}^- \cup \mathcal{T}$ is guaranteed to be consistent at Step 3.

This shows that the output $\mathcal{A}'$ is indeed a σ_{bop}-repair for the $\mathcal{R}$ with at most k negative assertions. The case when few positive assertions are allowed for addition is symmetric.

Finally, we show that if a given $\mathcal{R}$ has σ_{bop} repairs, then after executing the Steps 1–3 some σ_{bop} repair is found, i.e. $\mathcal{A}' = \mathcal{A} \cup \mathcal{H}^+ \cup \mathcal{H}^-$, such that $|\mathcal{H}^-| \leq k$. Assume towards a contradiction that this is not the case. We distinguish the cases based on stages of the algorithm at which the computation could have terminated.

- Suppose that the computation terminated at (1). Then there is some $\langle U_j^2, Q_j^2 \rangle \in D_2$, such that either (i) $U_j^2 \cup \mathcal{T} \models Q_j^2$ or (ii) $\mathcal{A} \cap Supp_j^2 \neq \emptyset$. If (i) holds then by monotonicity we have that for any $\mathcal{A}'$ the condition (iii) of Definition 26 is not satisfied, i.e. $\mathcal{R}$ does not have any solutions, which contradicts our assumption. If (ii) is the case, then there is some $\alpha \in \mathcal{A}$, such that $\alpha \cup \mathcal{T} \models Q_j^2$. Again due to monotonicity, for any ABox $\mathcal{A}' \supseteq \mathcal{A}$ it is true that $\mathcal{A}' \models Q_j^2$. Thus all repairs $\mathcal{A}'$ for $\mathcal{R}$ are such that $\mathcal{A}' \not\supseteq \mathcal{A}$. Therefore, no σ_{bop} repair exists for $\mathcal{R}$, contradicting our assumption.
- Assume that we have reached (2), and constructed the sets $\mathcal{S}_j$. Suppose that the computation stopped at (2), i.e. no hitting set $\mathcal{H}$ of $\mathcal{S}_j$ was found. This means that some j_1 exists, such that $\mathcal{S}_{j_1} = \emptyset$. Therefore, by construction of $\mathcal{S}_{j_1}$ it holds that $Supp_{j_1} \setminus (\mathcal{A} \cup \bigcup_{j'} Supp_{j'}^2) = \emptyset$. Since all $\langle U_j^1, Q_j^1 \rangle$, such that $Supp_j^1 \cap \mathcal{A} \neq \emptyset$ were removed from D_1 at (1), we have that for all $\alpha \in Supp_{j_1}^1$, it holds that $\alpha \in Supp_k^2$ for some k. Hence, for all ABoxes $\mathcal{A}' = \mathcal{A} \cup \alpha$ some $\langle U_k^2, Q_k^2 \rangle \in D_2$ exists, such that $\langle \mathcal{T}, \mathcal{A}' \rangle \models Q_k^2$, meaning that $\mathcal{R}$ does not have any solutions, which leads to a contradiction.
- Suppose that the state (3) has been reached, i.e. some repair candidate $\mathcal{A}' = \mathcal{A} \cup \mathcal{H}$ was identified at (2), where $\mathcal{H}$ is a hitting set of $\mathcal{S}_j$. At (3) we picked some set $\mathcal{H}^-$ and updated every $\mathcal{S}_j$ by removing appropriate assertions from $\mathcal{S}_j$. Computation could not have stopped at (3), therefore, we are guaranteed to reach (4). Assume that the algorithm terminated at (4). Then it must be the case that no hitting set $\mathcal{H}^+$ of updated $\mathcal{S}_j$ has been found at (4); that is for all choices of $\mathcal{H}^-$ at (3) some j_1 exists, such that $\mathcal{S}_{j_1} = \emptyset$ at (4). Consider some particular $\mathcal{H}^- \subseteq \bigcup_j \mathcal{S}_j$ of at most k assertions computed at (3), such that $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{H}^- \rangle$ is consistent. We have that $\mathcal{S}_{j_1} \cap \mathcal{H}^- = \emptyset$ at (3), since otherwise $\mathcal{S}_{j_1}$ would have been removed and would have not been considered in the computation of a hitting set $\mathcal{H}^+$ at (4). We have that for all positive $\alpha \in \mathcal{S}_{j_1}$, the ontology $\langle \mathcal{T}, \mathcal{A} \cup \mathcal{H}^- \cup \{\alpha\} \rangle$ is inconsistent. As $\langle U_{j_1}^1, Q_{j_1}^1 \rangle$ was not dropped at (1), we have that $\langle \mathcal{T}, U_{j_1}^1 \cup \mathcal{A} \rangle \not\models Q_{j_1}^1$. Therefore, it follows by Lemma 35 that no σ_{bop}-repair exists, such that $|\mathcal{A}'^- \setminus \mathcal{A}| \leq k$, leading to a contradiction.

We have shown that if $\mathcal{R}$ has solutions with at most k negative assertions, then some such solution will be found by our algorithm. The argument can be accordingly adjusted to prove the statement for few positive assertions are allowed for addition. □

Appendix C. Proofs for Section 4

Proof of Lemma 13. We prove each "if" direction of the statement separately.

- We first show that if $I \models^{\mathcal{O}} d$ then $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon;\ Q](\vec{t})$. Let $I \models^{\mathcal{O}} d$. That means that $\mathcal{O} \cup \lambda^I(d) \models Q(\vec{t})$. By definition, we have that $\lambda^I(d) = \{P(\vec{t}) \mid p(\vec{t}) \in I \text{ and } P \uplus p \in \lambda\} \cup \{\neg P(\vec{t}) \mid p(\vec{t}) \in I \text{ and } P \cup\!\!\!\!- p \in \lambda\}$. Therefore, $\mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\} \cup \{P_p \sqsubseteq \neg P \mid P \cup\!\!\!\!- p \in \lambda\} \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\} \models \lambda^I(d)$, i.e. $\mathcal{O}_d^I \models \lambda^I(d)$, and hence $\mathcal{O}_d^I \models Q(\vec{t})$. Therefore, we get that $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon;\ Q](\vec{t})$.
- We now prove the opposite direction, i.e. if $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon;\ Q](\vec{t})$ then $I \models^{\mathcal{O}} d$, where $d = \mathrm{DL}[\lambda;\ Q](\vec{t})$. Let $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon;\ Q](\vec{t})$. Then we have that $\mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\} \models Q(\vec{t})$. By construction of $\mathcal{O}_d^I$, λ must be as follows: $P \uplus p \in \lambda$ iff $P_p \sqsubseteq P \in \mathcal{T}_d$; $P \cup\!\!\!\!- p \in \lambda$ iff $P_p \sqsubseteq \neg P \in \mathcal{T}_d$. Therefore, for all $P' \in \mathsf{sig}(\mathcal{A} \cap (\mathcal{A}_d \backslash \mathcal{A}))$, we have that if $\mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{t}) \mid p(\vec{t}) \in I\} \models P'(\vec{t}')$ then $\mathcal{T} \cup \mathcal{A} \cup \lambda^I(d) \models P'(\vec{t}')$. As $Q \notin \mathsf{sig}(\mathcal{A}_d \backslash \mathcal{A})$, we obtain that $\mathcal{O} \cup \lambda^I(d) \models Q(\vec{t})$, and hence $I \models^{\mathcal{O}} d$.
- The last implications, i.e. $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon;\ Q](\vec{t})$ iff $I \models^{\mathcal{O}_d^I} Q(\vec{t})$ are immediate from the definition of a DL-atom's satisfaction by an interpretation. □

Proof of Theorem 54. *(Soundness of RepAns)* Let $\mathcal{A}'$ be an output of *RepAns*. Towards a contradiction, suppose $\mathcal{A}' \notin rep_{(\sigma,x)}^{\hat{I}|_\Pi}(\Pi)$. Then $\hat{I}|_\Pi \notin AS(\Pi')$, where $\Pi' = \langle \mathcal{T}, \mathcal{A}', P \rangle$ and $\mathcal{A}'$ is σ-selected. Clearly, $\mathcal{A}'$ is σ-selected, since otherwise $\mathcal{A}' \notin ORP(\hat{I}, \Pi, \sigma)$ and $\mathcal{A}'$ is not in the output. As it holds that $\hat{I} \in AS(\hat{\Pi})$, it must hold that either $\hat{I}$ is not a compatible set of Π' or it is not x-founded. If either of these cases is true, then the corresponding procedure *CMP* or *xFND* returns false and $\mathcal{A}'$ is not in the output, which leads to contradiction.

(Completeness of RepAns) Let $rep_{(\sigma,x)}^{\hat{I}|_\Pi}(\Pi)$ be the set of all σ-selected repairs for Π that turn $\hat{I}|_\Pi$ into an x-repair answer set. Towards a contradiction, assume that there exists some $\mathcal{A}' \in rep_{(\sigma,x)}^{\hat{I}|_\Pi}(\Pi)$ which is not an output of the algorithm *RepAns*. Then either (1) $\mathcal{A}' \notin ORP(\hat{I}, \Pi, \sigma)$; (2) $CMP(\hat{I}, \langle \mathcal{T}, \mathcal{A}', P \rangle) = false$ or (3) $xFND(\hat{I}, \langle \mathcal{T}, \mathcal{A}', P \rangle) = false$. If (1) holds, then $\mathcal{A}'$ is not a solution of the ORP instance. Thus either $\langle \mathcal{T}, \mathcal{A}' \rangle$ is unsatisfiable (contradiction to $\mathcal{A}' \in rep_{(\sigma,x)}^{\hat{I}|_\Pi}(\Pi)$ by the definition of repair) or the actual values of the DL-atoms do not coincide with the replacement atoms in $\hat{\Pi}$ (contradiction due to the

failure of the compatibility check). Finally, if either (2) or (3) holds then we obtain a contradiction, since $\mathcal{A}' \in rep^{\hat{I}|_\Pi}_{(\sigma,x)}(\Pi)$ implies $\hat{I}|_\Pi$ should be compatible and x-founded.

Soundness and Completeness of *RepAnsSet* follow immediately from the soundness and completeness of *RepAns*, respectively, and Proposition 51. □

Proof of Theorem 55. (i) NP-completeness result for $x = weak$.

(Membership) Given a candidate interpretation $I = \hat{I}|_\Pi$ for some $\hat{I} \in \mathrm{AS}(\hat{\Pi})$, we guess a repair $\mathcal{A}'$ and then check whether I satisfies all rules of the reduct $\mathcal{P}^{I,\mathcal{O}}_{weak}$. Both the construction of the reduct $\mathcal{P}^{I,\mathcal{O}}_{weak}$ and the check whether I satisfies all rules of $\mathcal{P}^{I,\mathcal{O}}_{weak}$ is polynomial, from which the membership in NP is obtained.

(Hardness) To prove NP-hardness, we reduce 3SAT to deciding whether a given interpretation I obtained from answer set $\hat{I} \in \mathrm{AS}(\hat{\Pi})$ is a weak repair answer set of Π as follows.

Let $\phi = C_1 \wedge \cdots \wedge C_m$ be 3SAT instance, where each C_j, $1 \le j \le m$, is a disjunction of three atoms over the variables $x_1, \ldots, x_n$. From this we construct a DL-program $\Pi = \langle \mathcal{T}, \mathcal{A}, \mathcal{P} \rangle$.

- As for the TBox $\mathcal{T}$, we introduce concept names X_i and $\bar{X}_i$, for each variable x_i occurring in ϕ. Moreover, we introduce a concept name C_j for each clause C_j in ϕ. Then $\mathcal{T}$ contains the following axioms:
  - $X_i \sqsubseteq C_j$ iff x_i is a disjunct in C_j;
  - $\bar{X}_i \sqsubseteq C_j$ iff $\neg x_i$ is a disjunct in C_j;
  - $X_i \sqsubseteq \neg \bar{X}_i$ and $\bar{X}_i \sqsubseteq \neg X_i$ for all pairs X_i, $\bar{X}_i$;
- the ABox is $\mathcal{A} = \{D(b)\}$, where D and b are a fresh concept and a fresh constant respectively.
- As for $\mathcal{P}$, we introduce fresh ground atoms $p_i(b)$ (resp. $\bar{p}_i(b)$) for each x_i occurring positively (resp. negatively) in ϕ. The rules of $\mathcal{P}$ are as follows:

$$\mathcal{P} = \left\{ \begin{array}{l} (1)\ \bot \leftarrow \mathrm{DL}[;\ D](b); \\ (2)\ \bot \leftarrow not\ \mathrm{DL}[;\ C_j](b), \quad 1 \le j \le m; \\ (3)\ \bot \leftarrow not\ \mathrm{DL}[\lambda_j;\ \neg C_j](b), \quad 1 \le j \le m; \\ (4)\ p_i(b).\ \mid p_i \text{ occurs in } \lambda_j, 1 \le i \le n, 1 \le j \le m; \\ (5)\ \bar{p}_i(b),\ \mid \bar{p}_i \text{ occurs in } \lambda_j, 1 \le i \le n, 1 \le j \le m \end{array} \right\},$$

  where for each x_i, we have that $X_i \uplus p_i$ (resp. $\bar{X}_i \uplus \bar{p}_i$) occurs in λ_j if $\neg x_i$ (resp. x_i) is a disjunct in C_j of ϕ. In addition, $\mathcal{P}$ contains the facts $p_i(b)$ (resp. $\bar{p}_i(b)$) iff x_i (resp. $\bar{x}_i$) occurs in some λ_j.
- I consists of the atoms that occur as facts in $\mathcal{P}$.

For illustration, let $\phi = x_1 \vee \neg x_2 \vee x_3$ with $n = 3$ and $m = 1$. Then the DL-program $\Pi = \langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$ is such that $\mathcal{T} = \{X_1 \sqsubseteq C_1;\ \bar{X}_2 \sqsubseteq C_1;\ X_3 \sqsubseteq C_1\}$, $\mathcal{A} = \{D(b)\}$ and

$$\mathcal{P} = \left\{ \begin{array}{l} \bot \leftarrow \mathrm{DL}[;\ D](b); \\ \bot \leftarrow not\ \mathrm{DL}[;\ C_1](b); \\ \bot \leftarrow not\ \mathrm{DL}[\bar{X}_1 \uplus \bar{p}_1,\ X_2 \uplus p_2, \bar{X}_3 \uplus \bar{p}_3;\ \neg C_1](b); \\ \bar{p}_1(b);\quad p_2(b);\quad \bar{p}_3(b) \end{array} \right\}.$$

The interpretation I contains the facts of $\mathcal{P}$, i.e. $I = \{\bar{p}_1(b), p_2(b), \bar{p}_3(b)\}$. Note that $I = \hat{I}|_\Pi$ for some answer set $\hat{I}$ of Π. The assignment $\nu(\phi)$ such that $\nu(x_1) = \nu(x_2) = true$, and $\nu(x_3) = false$ satisfies of ϕ; according to our construction, from $\nu(\phi)$ the repair $\mathcal{A}' = \{X_1(b), X_2(b), \bar{X}_3(b)\}$ of Π is obtained.

Note that for every Π constructed, all answer sets of $\hat{\Pi}$ coincide on the predicates of Π, i.e., $\hat{I}|_\Pi = \hat{J}|_\Pi$ for every $\hat{I}, \hat{J} \in AS(\hat{\Pi})$.

We claim that ϕ is satisfiable iff $I \in RAS_{weak}(\Pi)$, i.e. there exists an ABox $\mathcal{A}'$ such that I is a weak answer set of $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$.

($\Rightarrow$) Suppose that ϕ is satisfiable and $\nu(\phi)$ is a satisfying assignment. From this we construct a repair ABox $\mathcal{A}'$, such that $X_i(b) \in \mathcal{A}'$ (resp. $\bar{X}_i(b) \in \mathcal{A}'$), if x_i is true (resp. false) under the assignment $\nu(\phi)$.

Now we show that I is a weak answer set of $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$, and thus a weak repair answer set of Π. Observe that the body of the rule (1) is not satisfied, as $D(b) \notin \mathcal{A}'$. Furthermore, the DL-atoms $\mathrm{DL}[;\ C_1](b), \ldots, \mathrm{DL}[;\ C_m](b)$ evaluate to true under $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$, since $\mathcal{O}' \models C_j(b)$ for all $1 \le j \le m$ by construction. Moreover, each $d_j = \mathrm{DL}[\lambda_j;\ \neg C_j](b)$ evaluates to true under I, because the ontology $\mathcal{O}' \cup \lambda^I(d_j)$ is unsatisfiable (by construction $X_i(b) \in \mathcal{A}'$ or $\bar{X}_i(b) \in \mathcal{A}'$ for some $X_i \sqsubseteq C_j$ resp. $\bar{X}_i \sqsubseteq C_j$), and thus each $\neg C_j(b)$ is trivially entailed. Therefore, none of the constraints of $\mathcal{P}$ is present in the program reduct $\mathcal{P}^{I,\mathcal{O}'}_{weak}$. The reduct $\mathcal{P}^{I,\mathcal{O}}_{weak}$ contains only facts of the program, from which we get that I is a weak repair answer set of Π.

($\Leftarrow$) Let I be a weak repair answer set of Π and let $\mathcal{A}'$ be its respective repair. Then all DL-atoms of Π apart from DL[; D](b) are true. This means that for all C_j it holds that $\mathcal{O}' \models C_j(b)$. The ontology $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ is satisfiable, therefore $X_i(b)$ and $\bar{X}_i(b)$ simultaneously can not be in $\mathcal{A}'$. Therefore, either

(i) $C_j(b) \in \mathcal{A}'$ or
(ii) $X(b) \in \mathcal{A}'$, such that $X \sqsubseteq C_j$ is in $\mathcal{T}$.

If (i) was true, then the bodies of the constraints (4) would be satisfied, which contradicts I being a repair answer set. Thus, it holds that some $X(b) \in \mathcal{A}'$ such that $X \sqsubseteq C_j \in \mathcal{T}$. Hence, from the repair ABox $\mathcal{A}'$ a satisfying assignment $\nu(\phi)$ can be constructed as follows: $\nu(\phi)$ such that $\nu(x_i) = \mathit{true}$ (resp. $\nu(a_i) = \mathit{false}$) if $X_i(b) \in \mathcal{A}'$ (resp. $\bar{X}_i(b) \in \mathcal{A}'$). The assignment $\nu(\phi)$ witnesses satisfiability of ϕ.

(ii) Σ_2^P-completeness result for $x = \mathit{flp}$.

(Membership) We can guess a repair $\mathcal{A}'$ and then check whether I is an *flp*-repair answer set of $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$, where $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$. Constructing the reduct $\mathcal{P}_{\mathit{flp}}^{I,\mathcal{O}'}$ is polynomial, as we only need to pick those rules of Π whose body is satisfied by I, and all DL-atoms can be evaluated in polynomial time. As shown in the proof of Theorem 18 the check (i) is polynomial and the check (ii) is in co-NP, from which membership in Σ_2^P follows.

(Hardness) The hardness is shown by the construction in the proof of Theorem 18 (iii). We set $I = \{w(a), y_1(a), \ldots, y_m(a)\}$ and consider deciding whether $I \in \mathit{RAS}(\Pi)$, i.e. whether some ABox $\mathcal{A}'$ exists such that $I \in \mathit{AS}(\Pi')$, where $\Pi' = \langle \mathcal{T}, \mathcal{A}'\mathcal{P} \rangle$. Note that every answer set of $\hat{\Pi}$ resp. repair answer set of Π must contain $w(a)$, and that $I = \hat{I}|_{\Pi}$ for some $\hat{I} \in \mathrm{AS}(\hat{\Pi})$ and $\Pi_{\mathit{flp}}^{I,\mathcal{O}'} = \{(5), (6)\}$ for every $\mathcal{O}' = \mathcal{T} \cup \mathcal{A}'$. Furthermore, $\hat{I}|_{\Pi} = \hat{J}|_{\Pi}$ for every answer sets $\hat{I}, \hat{J} \in \mathrm{AS}(\hat{\Pi})$.

Due to (1)–(3), a repair $\mathcal{A}'$ must be a maximal consistent subset of $\{X_i(a), \neg X_i(a) \mid 1 \le i \le n\}$ and thus encode a truth assignment ν to $x_1, \ldots, x_n$. Now $I \in \mathit{RAS}_{\mathit{flp}}(\Pi)$ implies that some $\mathcal{A}'$ exists s.t. by minimality of I, for each $I' \subseteq I \backslash \{w(a)\}$ some index k exists such that all $f(l_{k_1})$, $f(l_{k_2})$, $f(l_{k_3})$ are true, hence χ_k is true; therefore, ϕ is true. Conversely, every assignment ν to $x_1, \ldots, x_n$ witnessing that ϕ is true induces some maximal consistent subset $\mathcal{A}' \subseteq \{X_i(a), \neg X_i(a) \mid 1 \le i \le n\}$. By a slight adaptation of the argument in the proof of Theorem 18, it can be shown that $\mathcal{A}' \in \mathit{rep}^I_{\mathit{flp}}(\Pi)$; this proves Σ_2^P-hardness under the asserted restriction. $\Box$

Proof of Proposition 61. ($\Rightarrow$) Suppose $d = DL[\lambda; Q](\vec{t})$ evaluates w.r.t. $\mathcal{O}$ and I to true, i.e., $\lambda^I(d) \cup \mathcal{O} \models Q(\vec{t})$. Towards a contradiction, assume no $S \in \mathit{Supp}_{\mathcal{O}}(d)$ is coherent with I. There are two cases:

(1) $\lambda^I(d) \cup \mathcal{O}$ is consistent. Proposition 5 implies that an assertion $\alpha \in \lambda^I(d) \cup \mathcal{A}$ must exist such that $\mathcal{T} \cup \{\alpha\} \models Q(t)$. If $\alpha \in \mathcal{A}$ then $\mathit{Supp}_{\mathcal{O}}(d)$ contains $\{\alpha\}$ by (i) of Proposition 58, which trivially is coherent with I and thus contradicts the assumption. If $\alpha \in \lambda^I(d)$, then α is an input assertion for d. For $\alpha_d \in \mathcal{A}_d$, we then obtain that $\{\alpha_d\} \in \mathit{Supp}_{\mathcal{O}}(d)$ according to (i) of Proposition 58, again a contradiction due to coherence with I.

(2) $\lambda^I(d) \cup \mathcal{O}$ is inconsistent. From Proposition 5 and consistency of $\mathcal{O}$, it follows that some $\delta \in \lambda^I(d)$ exists such that either (a) $\mathcal{T} \cup \{\delta\}$ is inconsistent, or (b) some $\gamma \in \mathcal{A} \cup \lambda^I(d)$ exists such that $\mathcal{T} \cup \{\delta, \gamma\}$ is inconsistent. In case a), we obtain $\{\delta_d\} \in \mathit{Supp}_{\mathcal{O}}(d)$, for the corresponding input assertion $\delta_d \in \mathcal{A}_d$. by (i) of Proposition 58; this is a contradiction, as $\{\delta_d\}$ is coherent with I. In case b), we similarly conclude that either $\{\delta_d, \gamma\} \in \mathit{Supp}_{\mathcal{O}}(d)$ or $\{\delta_d, \gamma_d\} \in \mathit{Supp}_{\mathcal{O}}(d)$, depending on whether $\gamma \in \lambda^I(d)$, according to (ii) of Proposition 58. Again this is a support set coherent with I, contradiction.

($\Leftarrow$) Suppose some $S \in \mathit{Supp}_{\mathcal{O}}(d)$ is coherent with I. Assume towards a contradiction that $I \not\models^{\mathcal{O}} d$. Again we consider two cases:

(1) $\mathcal{T}_d \cup S$ is consistent. Then, $\mathcal{T}_d \cup S \models Q(\vec{t})$ by item (i) of Proposition 58. Since S is coherent with I, we conclude that $\mathcal{O}_d^I \models Q(\vec{t})$ which implies $I \models^{\mathcal{O}} d$ by Proposition 13. Contradiction.

(2) $\mathcal{T}_d \cup S$ is inconsistent. Then, due to coherence with I, so is $\mathcal{O}_d^I$, and trivially $\mathcal{O}_d^I \models Q(\vec{t})$; again we arrive at a contradiction by concluding that $I \models^{\mathcal{O}} d$ from Proposition 13. $\Box$

Proof of Proposition 63. By Proposition 58 there are two possibilities: either (i) S is unary or (ii) it is binary. In case of (i) we have that S is either a concept or a role assertion, and therefore, it involves at most two constants. For the case (ii) it holds that $S \cup \mathcal{T}_d$ is inconsistent. Hence S represents a binary conflict set. It has been shown in [55] that S can be a binary conflict set only due to one of the following reasons:

- $\mathcal{T} \models C \sqsubseteq \neg D$, and $S = \{C(a), D(a)\}$, in which case S involves only 1 constant;
- $\mathcal{T} \models R \sqsubseteq \neg R'$, and $S = \{R(a,b), R'(a,b)\}$, thus S involves at most 2 constants;
- $\mathcal{T} \models C \sqsubseteq \neg \exists R$ or $C \sqsubseteq \neg \exists R^-$, and $S = \{C(a), R(a,b)\}$ resp. $S = \{C(a), R(b,a)\}$, i.e. S involves at most 2 constants;
- $\mathcal{T} \models \exists R \sqsubseteq \neg C$ or $\exists R^- \sqsubseteq \neg C$, and $S = \{R(a,b), C(a)\}$ resp. $S = \{R(b,a), C(a)\}$, in which case S involves 2 constants maximum;
- $\mathcal{T} \models \exists R \sqsubseteq \neg \exists R'$, and $S = \{R(a,b), R'(a,c)\}$, thus there are 3 constants occurring in S (similarly for the cases $\mathcal{T} \models \exists R^- \sqsubseteq \neg \exists R'^-$, $\mathcal{T} \models \exists R \sqsubseteq \neg \exists R'^-$, $\mathcal{T} \models \exists R^- \sqsubseteq \neg \exists R'^-$);

- $funct(R) \in \mathcal{T}$, and $S = \{R(a,b), R(a,c)\}$ with $b \neq c$, in which case again at most 3 constants appear in S (the case when $funct(R^-) \in \mathcal{T}$ is analogous).

As we have considered all possibilities for binary conflict sets, the statement is proved. □

Proof of Proposition 71. Assume that I is an answer set of $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$, where $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ and that $\hat{I}$ is a compatible set for $\Pi' = \langle \mathcal{O}', \mathcal{P} \rangle$ where $\mathcal{O}' = \langle \mathcal{T}, \mathcal{A}' \rangle$ and $\mathcal{A}' \supset \mathcal{A}$. Towards contradiction, suppose I is not an answer set of Π. Hence, $I = \hat{I}|_\Pi$ is not a minimal model of $\Pi'^{I,\mathcal{O}}_{flp} = \langle \mathcal{T}, \mathcal{A}', \mathcal{P}^{I,\mathcal{O}}_{flp} \rangle$. That is, some $I' \subset I$ exists such that $I' \models^{\mathcal{O}'} \mathcal{P}^{I,\mathcal{O}}_{flp}$. We then obtain that also $I' \models^{\mathcal{O}} \mathcal{P}^{I,\mathcal{O}}_{flp}$; this contradicts $I \in AS(\Pi)$. Indeed, suppose that $I' \not\models^{\mathcal{O}} \mathcal{P}^{I,\mathcal{O}}_{flp}$. Then some rule $r \in \mathcal{P}^{I,\mathcal{O}}_{flp}$ of form (1) is violated wrt. I' and $\mathcal{O}$, i.e., (i) $I' \models^{\mathcal{O}} b_i$ for each $1 \leq i \leq k$, (ii) $I' \not\models^{\mathcal{O}} b_j$ for each $k < j \leq m$, and (iii) $I' \not\models^{\mathcal{O}} a_h$ for each $1 \leq h \leq n$. By monotonicity of $I \models^{\mathcal{O}} a$ w.r.t. I and $\mathcal{O}$, we conclude $I' \models^{\mathcal{O}'} b_i$, $I' \not\models^{\mathcal{O}'} b_j$ (as $\hat{I}$ is a compatible set for both $\hat{\Pi}$ and $\hat{\Pi}'$), and $I \not\models^{\mathcal{O}} b_j$, and $I' \not\models^{\mathcal{O}'} a_h$. But then $I' \not\models^{\mathcal{O}'} \mathcal{P}^{I,\mathcal{O}}_{flp}$, which is a contradiction. Hence, I' does not exist and I is an answer set of Π'. □

Proof of Theorem 72.
(Soundness) Suppose *SupRAnsSet* outputs $I = \hat{I}|_\Pi$. We can get to (h) only if $\hat{I}$ is an answer set of $\hat{\Pi}$; furthermore, by setting $\mathbf{S}^{\hat{I}}_{gr}$ to $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$ in (b) and by the further modifications, it is ensured at (h) that each DL-atom $a \in D_p$ has some coherent support set that matches with $\mathcal{A}'$ (i.e., $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) \neq \emptyset$), while no DL-atom $a' \in D_n$ has such a support set. Thus from Proposition 61, it follows that $\hat{I}$ is a compatible set for $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', P \rangle$; hence $I \models \Pi'$. Furthermore, as *flpFND*$(\hat{I}, \mathcal{T} \cup \mathcal{A}', P)$ succeeds, I is a minimal model of $\Pi'^{I,\mathcal{O}}_{flp}$. Hence I is an answer set of Π', and thus a deletion repair answer set of Π.

(Completeness) Suppose I is a deletion repair answer set. That is, for some $\mathcal{A}' \subseteq \mathcal{A}$, we have that I is an answer set of $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', P \rangle$. This implies Proposition 50 that $\hat{I}$ is an answer set of $\hat{\Pi}$ and thus will be considered in (b), with D_p and D_n reflecting the (correct) guess for $I \models^{\mathcal{O}'} a$ for each DL-atom a, where $\mathcal{O}' = \mathcal{T} \cup \mathcal{A}'$. From Proposition 61 and completeness of $\mathbf{S}$, we obtain that each $a \in D_p$ has $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) \neq \emptyset$ and each $a \in D_n$ has $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) = \emptyset$. The initial $\mathbf{S}^{\hat{I}}_{gr}$ is such that $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a) \subseteq \mathbf{S}^{\hat{I}}_{gr} = Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a)$ holds for each DL-atom a; in further steps, the algorithm removes all support sets $S \in Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a)$ for $a \in D_p$ from $\mathbf{S}^{\hat{I}}_{gr}(a)$ such that such that $S \cap S' \cap \mathcal{A} \neq \emptyset$ for some support set $S' \in Gr(\mathbf{S}, \hat{I}, \mathcal{A})(a')$ and $a' \in D_n$, and removes all assertions in $S' \cap \mathcal{A}$ from $\mathcal{A}$. Importantly no removed S is in $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(a)$, since by the assertion that $\mathcal{T} \cup \mathcal{A}$ is consistent, $|S' \cap \mathcal{A}| = 1$ must hold. Thus step (g) will be reached, and the variable $\mathcal{A}'$ is assigned an ABox $\mathcal{A}''$ such that $\mathcal{A}' \subseteq \mathcal{A}'' \subseteq \mathcal{A}$. Since $\hat{I}$ is a compatible set for $\Pi'' = \langle \mathcal{T} \cup \mathcal{A}'', \mathcal{P} \rangle$ and I is an answer set of Π', by Proposition 71 I is also an answer set of Π'', and thus I is a minimal model of $\Pi''^{I,\mathcal{O}}_{flp} = \langle \mathcal{T} \cup \mathcal{A}'', \mathcal{P}^{I,\mathcal{O}}_{flp} \rangle$. Hence, the test *flpFND*$(\hat{I}, \mathcal{T} \cup \mathcal{A}', \mathcal{P})$ in step (h) (where $\mathcal{A}'$ has value $\mathcal{A}''$) succeeds, and $\hat{I}_\Pi$, i.e, I is output. □

Appendix D. Proofs for Section 5

Proof (sketch) of Proposition 73. For *DL-Lite*$_\mathcal{A}$ ontologies classification can be modeled declaratively as a reachability problem, what is exactly reflected in the rules (1), (2) and (3). The conflict sets in turn are found by means of the rules (4)–(7) and analysis of functional roles. As a result the program $Prog_{\mathcal{T}_{class}}$ computes all concept and role inclusions that follow from the TBox as well as all unary and binary conflict sets (whose construction is sound and complete based on the results in [74]). All support sets of type (i) of Proposition 58 are extracted from subsumptions and unary conflict sets, while the support sets of type (ii) correspond to binary conflict sets. As according to Proposition 58 there are no other types of support sets from the model $M_{\mathcal{T}_{class}}$ of $Prog_{\mathcal{T}_{class}}$, a complete support family for a given DL-atom can be extracted. □

Proof of Proposition 75. We separately prove $AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))|_\Pi \subseteq RAS_{weak}(\Pi)$ and $AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))|_\Pi \supseteq RAS_{weak}(\Pi)$, i.e., correctness and completeness of the provided implementation.

($\subseteq$) Assume towards a contradiction that $AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))|_\Pi \not\subseteq RAS_{weak}(\Pi)$. Then there exists an element $I \in AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))$, such that $I|_\Pi \notin RAS_{weak}(\Pi)$. This means that for all $\mathcal{A}' \subseteq \mathcal{A}$, it holds that $I|_\Pi \notin AS_{weak}(\Pi')$ with $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. Consider the ABox $\mathcal{A}'' = \{P(\vec{c}) \mid p_P(\vec{c}) \in I|_{facts(\mathcal{A})},\ \bar{p}_P(\vec{c}) \notin I\}$,[20] which is a particular subset of $\mathcal{A}$. We have that $I|_\Pi \notin AS_{weak}(\Pi'')$ with $\Pi'' = \langle \mathcal{T}, \mathcal{A}'', \mathcal{P} \rangle$. Thus one of the following must be true: (i) no extension of $I|_\Pi$ with guessed values of replacement atoms is a model of $\hat{\Pi}''$, (ii) no model of $\hat{\Pi}''$ is a compatible set for Π'' or (iii) there exists $I' \subset I|_\Pi$, which is a model of $\mathcal{P}^{I|_\Pi, \mathcal{O}''}_{weak}$.

The case (i) is irrelevant, as $I|_{\hat{\Pi}}$ satisfies all rules of $\hat{\Pi}$ due to $I \in AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))$ and $\hat{\Pi}'' = \hat{\Pi}$. We next show that (ii) can not hold by deriving a contradiction. Indeed, assume that (ii) holds, then as $I|_{\hat{\Pi}}$ is a model of $\hat{\Pi}$, it is not a compatible set for Π. Therefore there exists a DL-atom a_i in Π'', such that its real value is different from the guessed value

[20] $\bar{p}_P$ corresponds to the respective $\bar{S}^{\mathcal{A}}_a$.

in $I|_{\hat{\Pi}}$. Suppose first that $I|_{\Pi} \models a_i$, but $ne_{a_i} \in I|_{\hat{\Pi}}$. By Proposition 61 there must exist a support set $S \in \mathcal{S}_i$, such that S is coherent with $I|_{\Pi}$ and its ABox part $S^{\mathcal{A}}$ is in $\mathcal{A}''$. If $S^{\mathcal{A}}$ is nonempty, then due to the rule of the form (r_4) of Π_{supp} we get that $\bar{S}^{\mathcal{A}}$ must be in I, but then $S_{\mathcal{A}}$ is not present in $\mathcal{A}''$. Therefore, $S^{\mathcal{A}}$ must be empty, i.e. S must contain only input assertions. However, then the body of the constraint (r_2) of Π_{supp} is satisfied, contradicting $I \in AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))$. In conclusion, this shows that (ii) does not hold, and in particular that $I|_{\hat{\Pi}}$ is a compatible set for Π.

Finally, the last possibility is that (iii) holds, meaning that there is an interpretation $I' \subset I|_{\Pi}$ which is a model of $\mathcal{P}^{I|_{\Pi},\mathcal{O}''}_{weak}$. The interpretations $I|_{\Pi}$ and I' differ on the set $M = I|_{\Pi} \backslash I'$, containing only ground atoms from the language of Π. Let us now look at the interpretation $I'' = I \backslash M$. We know that I is an answer set of $\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A})$, i.e. it is a minimal model of $\hat{\Pi}^{I}_{gl} \cup {\Pi_{supp}}^{I}_{gl} \cup facts(\mathcal{A})$. Therefore, there must exist some rule r^{I}_{gl} either in (1) $\hat{\Pi}^{I}_{gl}$ or in (2) ${\Pi_{supp}}^{I}_{gl}$, which I'' does not satisfy, i.e. $I'' \models B(r^{I}_{gl})$ and $I'' \not\models H(r^{I}_{gl})$.

Assume that (1) holds. Then the rule r^{I}_{gl} must involve some replacement atoms e_a occurring positively. Otherwise $I' \not\models r^{I}_{gl}$, and since this rule is also in $\mathcal{P}^{I|_{\Pi},\mathcal{O}''}_{weak}$, we have that I' is not a model of $\mathcal{P}^{I|_{\Pi},\mathcal{O}''}_{weak}$, leading to a contradiction. Furthermore, we know that $I|_{\hat{\Pi}}$ is a compatible set. Therefore, $r^{I,\mathcal{O}''}_{weak}$ is the rule r^{I}_{gl} without replacement atoms in its body; but then $I' \not\models r^{I,\mathcal{O}''}_{weak}$, and hence I' is not a model of $\mathcal{P}^{I|_{\Pi},\mathcal{O}''}_{weak}$.

Now assume that (2) holds, i.e. there is a rule $r^{I}_{gl} \in {\Pi_{supp}}^{I}_{gl}$, such that $I'' \models B(r^{I}_{gl})$, but $I'' \not\models H(r^{I}_{gl})$. The rule r^{I}_{gl} can not be a constraint of the forms r_1, r_2, since then $I \supset I''$ is not an answer set of $\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A})$, leading to a contradiction. Therefore, r must be of the form r_3 or r_4. However, the latter is not possible either, since the set of atoms M on which I and I'' differ contains only atoms from the signature of Π, and $H(r^{I}_{gl})$ does not fall into this set, meaning that $I \not\models r^{I}_{gl}$, which contradicts to $I \in AS(\hat{\Pi} \cup facts(\mathcal{A}) \cup \Pi_{supp})$.

($\supseteq$) Suppose that $I \in RAS_{weak}(\Pi)$, but there is no $I' \supseteq I$, such that $I' \in AS(\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))$. By definition of repair answer sets, some $\mathcal{A}' \subset \mathcal{A}$ exists, such that $I \in AS_{weak}(\Pi')$, where $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. We construct the interpretation I' by extending I with

- $\{e_a \mid I \models^{\mathcal{O}'} a\} \cup \{ne_a \mid I \not\models^{\mathcal{O}'} a\}$, i.e. facts stating the values of the replacement atoms under I and $\mathcal{A}'$;
- $facts(\mathcal{A})$;
- $\{\bar{p}_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A} \backslash \mathcal{A}'\}$;
- $Sup_{a_i}(\vec{c})$ encoding information about support sets of $a_i(\vec{c})$ coherent with I.

We now show that I' is an answer set of $\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A})$, i.e. it is a minimal model of $(\hat{\Pi} \cup facts(\mathcal{A}') \cup \Pi_{supp})^{I'}_{gl}$. Assume towards a contradiction that this is not the case. Then either (i) I' does not satisfy some rules of the reduct, or (ii) some smaller model of the reduct exist.

First consider (i). I' immediately satisfies all facts as well as all rules in $\hat{\Pi}^{I'}_{gl}$. This means that there must be some rule $r^{I'}_{gl}$ in ${\Pi_{supp}}^{I'}_{gl}$ that is not satisfied, i.e. $I' \models B(r^{I'}_{gl})$, but $I' \not\models H(r^{I'}_{gl})$. By construction of I' and Proposition 61, if $e_a \in I'$ (resp. $ne_a \in I'$) then $Sup_a \in I'$ (resp. $Sup_a \notin I'$), therefore r can not be of the form (r_1) or (r_2). Suppose that r is of the form (r_3). We have that some DL-atom a has a support set whose ABox part is in $\mathcal{A}'$ or empty. Then by construction of I' the head of the rule $r^{I'}_{gl}$ has to be satisfied. Therefore, the rule r must be of the form (r_4). Then $I \not\models^{\mathcal{O}'} a$ for some DL-atom a, such that there is a support set for a which is coherent with I and its ABox part is either empty or present in $\mathcal{A}$. In both cases by Proposition 61 we get that $I \models^{\mathcal{O}'} a$, which leads to a contradiction.

Let us now look at (ii), i.e. some interpretation $I'' \subset I'$ exists such that $I'' \models (\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))^{I'}_{gl}$. Note that I'' and I' can not differ only on replacement atoms, since for each DL-atom a, either e_a or ne_a must be in I''. As I' already contains the corresponding replacement atoms, removal of any such atom will violate the satisfaction of some guessing rule $e_a \vee ne_a$ in $\hat{\Pi}^{I'}_{gl}$. Suppose that $I'' \backslash I'$ contains some atoms from Π. Consider $I''|_{\Pi}$, which is a subset of I. Observe that $I''|_{\Pi}$ can not be a model of $\mathcal{P}^{I,\mathcal{O}'}_{weak}$, because $I \supset I''|_{\Pi}$ is its minimal model. Therefore, some rule $r^{I,\mathcal{O}'}_{weak}$ must exist in the reduct $\mathcal{P}^{I,\mathcal{O}'}_{weak}$ which is not satisfied by $I''|_{\Pi}$, i.e. $I''|_{\Pi} \models B(r^{I,\mathcal{O}'}_{weak})$ but $I''|_{\Pi} \not\models H(r^{I,\mathcal{O}'}_{weak})$. By construction of the weak reduct this rule does not contain any DL-atoms. Let us look at the corresponding rule in the reduct $\hat{\Pi}^{I''}_{gl}$. The rule $r^{I''}_{gl}$ either does not contain any replacement atoms or contains only positive atoms e_a such that $e_a \in I''$ (by construction of the GL-reduct). Therefore $I'' \models B(r^{I'}_{gl})$, but $I'' \not\models H(r^{I'}_{gl})$, contradicting $I'' \models \hat{\Pi}^{I'}_{gl}$.

Suppose that the interpretations I' and I'' differ only on the facts over predicates in Π_{supp}. We know that the rule $r^{I'}_{gl}$, where r is of the form (r_1) is not present in ${\Pi_{supp}}^{I'}_{gl}$, moreover, $I'' \not\models r'^{I'}_{gl}$ for r' of the form (r_2). If the difference $I' \backslash I''$ contains Sup_a, then it must contain some atoms from $r(S_a)$ too. Moreover, these atoms must be related to the ABox facts, which are present in I'. This, however, means that some fact in $facts(\mathcal{A})$ is not satisfied, contradicting $I'' \models (\hat{\Pi} \cup \Pi_{supp} \cup facts(\mathcal{A}))^{I'}_{gl}$. Finally, $I'' \setminus I'$ can not contain elements $\bar{S}^{\mathcal{A}}_a$, as then the rule $r^{I'}_{gl}$ for r of the form (r_4) is not satisfied by I''. $\square$

References

[1] C.E. Alchourrón, P. Gärdenfors, D. Makinson, On the logic of theory change: partial meet contraction and revision functions, J. Symb. Log. 50 (2) (1985) 510–530.

[2] F. Baader, A. Bauer, M. Lippmann, Runtime verification using a temporal description logic, in: S. Ghilardi, R. Sebastiani (Eds.), Proc. 7th International Symp. Frontiers of Combining Systems (FroCoS 2009), in: LNCS, vol. 5749, Springer, 2009, pp. 149–164.

[3] F. Baader, S. Brandt, C. Lutz, Pushing the $\mathcal{EL}$ envelope, in: Proc. 19th Joint Conf. Artificial Intelligence (IJCAI 2005), AAAI Press, Scotland, 2005, pp. 364–369.

[4] F. Baader, D. Calvanese, D. McGuinness, D. Nardi, P.F. Patel-Schneider (Eds.), The Description Logic Handbook: Theory, Implementation and Applications, Cambridge Univ. Press, 2003.

[5] F. Baader, B. Hollunder, Embedding defaults into terminological knowledge representation formalisms, J. Autom. Reason. 14 (1) (1995) 149–180.

[6] F. Baader, C. Lutz, M. Milicic, U. Sattler, F. Wolter, Integrating description logics and action formalisms: first results, in: M.M. Veloso, S. Kambhampati (Eds.), Proc. 20th National Conf. Artificial Intelligence (AAAI 2005), AAAI Press/The MIT Press, 2005, pp. 572–577.

[7] M. Balduccini, M. Gelfond, Logic programs with consistency-restoring rules, in: Proc. International Symposium on Logical Formalization of Commonsense Reasoning, in: AAAI 2003 Spring Symposium Series, California, USA, 2003, pp. 9–18.

[8] N. Belnap, A useful four-valued logic, in: J.M. Dunn, G. Epstein (Eds.), Modern Uses of Multiple-Valued Logic, Reidel Publishing Company, Dordrecht, 1977, pp. 7–37.

[9] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Sci. Amer. 2841 (5) (2001).

[10] L.E. Bertossi, Database Repairing and Consistent Query Answering, Morgan & Claypool Publishers, Ottawa, Canada, 2011.

[11] L.E. Bertossi, A. Hunter, T. Schaub, Introduction to inconsistency tolerance, in: L.E. Bertossi, et al. (Eds.), Inconsistency Tolerance [Result from a Dagstuhl Seminar], in: LNCS, vol. 3300, Springer, 2005, pp. 1–14.

[12] M. Bienvenu, On the complexity of consistent query answering in the presence of simple ontologies, in: Proc. 26th Conf. Artificial Intelligence (AAAI 2012), AAAI Press, 2012, pp. 705–711.

[13] M. Bienvenu, C. Bourgaux, F. Goasdoué, Querying inconsistent description logic knowledge bases under preferred repair semantics, in: C.E. Brodley, P. Stone (Eds.), Proc. 28th Conf. Artificial Intelligence (AAAI 2014), AAAI Press, Canada, 2014, pp. 996–1002.

[14] P.A. Bonatti, M. Faella, I. Petrova, L. Sauro, A new semantics for overriding in description logics, Artif. Intell. 222 (2015) 1–48.

[15] P.A. Bonatti, C. Lutz, F. Wolter, The complexity of circumscription in DLs, J. Artif. Intell. Res. 35 (2009) 717–773.

[16] A. Borgida, D. Calvanese, M. Rodriguez-Muro, Explanation in *DL-Lite*, in: Proc. 21st International Workshop on Description Logics (DL 2008), Germany, 2008.

[17] G. Brewka, Preferred subtheories: an extended logical framework for default reasoning, in: Proc. 11th International Joint Conf. Artificial Intelligence (IJCAI 1989), Morgan Kaufmann, USA, 1989, pp. 1043–1048.

[18] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Commun. ACM 54 (12) (2011) 92–103.

[19] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, R. Rosati, Ontology-based database access, in: M. Ceci, D. Malerba, L. Tanca (Eds.), Proc. 15th Italian Symposium on Advanced Database Systems (SEBD 2007), Italy, 2007, pp. 324–331.

[20] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Lembo, A. Poggi, R. Rosati, MASTRO-I: efficient integration of relational data through DL ontologies, in: D. Calvanese, et al. (Eds.), Proc. 20th International Workshop on Description Logics (DL 2007), Italy, in: CEUR Workshop Proc., vol. 250, 2007, http://CEUR-WS.org.

[21] D. Calvanese, D. Lembo, M. Lenzerini, R. Rosati, Tractable reasoning and efficient query answering in description logics: the DL-Lite family, J. Autom. Reason. 39 (3) (2007) 385–429.

[22] D. Calvanese, M. Ortiz, M. Simkus, G. Stefanoni, Reasoning about explanations for negative query answers in *DL-Lite*, J. Artif. Intell. Res. 48 (2013) 635–669.

[23] G. Casini, U. Straccia, Rational closure for defeasible description logics, in: Proc. 12th European Conf. Logic in Artificial Intelligence (JELIA 2010), Finland, 2010, pp. 77–90.

[24] W.W. Cohen, P.D. Ravikumar, S.E. Fienberg, A comparison of string distance metrics for name-matching tasks, in: Proc. IJCAI 2003 Workshop on Information Integration on the Web (IIWeb-03), Mexico, 2003, pp. 73–78.

[25] N.C.A. da Costa, F. Holik, A formal framework for the study of the notion of undefined particle number in quantum mechanics, Synthese 192 (2) (2015) 505–523.

[26] F.M. Donini, M. Lenzerini, D. Nardi, W. Nutt, A. Schaerf, An epistemic operator for description logics, Artif. Intell. 100 (1–2) (1998) 225–274.

[27] F.M. Donini, D. Nardi, R. Rosati, Description logics of minimal knowledge and negation as failure, ACM Trans. Comput. Log. 3 (2) (2002) 177–225.

[28] T. Eiter, E. Erdem, M. Fink, J. Senko, Updating action domain descriptions, in: Proc. 19th International Joint Conf. Artificial Intelligence (IJCAI 2005), Morgan Kaufmann, Scotland, 2005, pp. 418–423.

[29] T. Eiter, M. Fink, G. Greco, D. Lembo, Repair localization for query answering from inconsistent databases, ACM Trans. Database Syst. 33 (2) (2008) 10.

[30] T. Eiter, M. Fink, T. Krennwallner, C. Redl, P. Schüller, Efficient HEX-program evaluation based on unfounded sets, J. Artif. Intell. Res. 49 (2014) 269–321.

[31] T. Eiter, M. Fink, J. Moura, Paracoherent answer set programming, in: Proc. 12th International Conf. Principles of Knowledge Representation and Reasoning (KR 2010), AAAI Press, 2010, pp. 486–496.

[32] T. Eiter, M. Fink, C. Redl, D. Stepanova, Exploiting support sets for answer set programs with external evaluations, in: Proc. 28th Conf. Artificial Intelligence (AAAI 2014), AAAI Press, Canada, 2014, pp. 1041–1048.

[33] T. Eiter, M. Fink, D. Stepanova, Data repair of inconsistent dl-programs, in: F. Rossi (Ed.), Proc. 23rd International Joint Conf. Artificial Intelligence (IJCAI 2013), AAAI Press/IJCAI, China, 2013, pp. 869–876.

[34] T. Eiter, M. Fink, D. Stepanova, Computing repairs for inconsistent dl-programs over $\mathcal{EL}$ ontologies, in: Proc. 14th Joint European Conf. Logics in Artificial Intelligence (JELIA 2014), Portugal, 2014, pp. 426–441.

[35] T. Eiter, M. Fink, D. Stepanova, Towards practical deletion repair of inconsistent dl-programs, in: T. Schaub, et al. (Eds.), Proc. 21st European Conf. Artificial Intelligence (ECAI 2014), Czeck Republic, IOS Press, 2014, pp. 285–290.

[36] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, Well-founded semantics for description logic programs in the Semantic Web, ACM Trans. Comput. Log. 12 (2) (2011) 11.

[37] T. Eiter, G. Ianni, T. Lukasiewicz, R. Schindlauer, H. Tompits, Combining answer set programming with description logics for the Semantic Web, Artif. Intell. 172 (12–13) (2008) 1495–1539.

[38] T. Eiter, G. Ianni, R. Schindlauer, H. Tompits, A uniform integration of higher-order reasoning and external evaluations in answer-set programming, in: Proc. 19th International Joint Conf. Artificial Intelligence (IJCAI 2005), Morgan Kaufmann, Scotland, 2005, pp. 90–96.

[39] T. Eiter, T. Krennwallner, M. Prandtstetter, C. Rudloff, P. Schneider, M. Straub, Semantically enriched multi-modal routing, Int. J. Intell. Transp. Syst. Res. (2014), http://dx.doi.org/10.1007/s13177-014-0098-8.

[40] M. Fink, Paraconsistent hybrid theories, in: Proc. 13th International Conf. Principles of Knowledge Representation and Reasoning (KR 2012), AAAI Press, Italy, 2012, pp. 141–151.

[41] M. Frühstück, J. Pührer, G. Friedrich, Debugging answer-set programs with ouroboros – extending the SeaLion plugin, in: Proceedings of the 12th International Conf. Logic Programming and Nonmonotonic Reasoning, Spain, 2013, pp. 323–328.

[42] P. Gärdenfors, H. Rott, Belief revision, in: Dov M. Gabbay, C.J. Hogger, J.A. Robinson (Eds.), Handbook of Logic in Artificial Intelligence and Logic Programming, vol. 4, Oxford Univ. Press, 1995, pp. 35–132.
[43] T. Gardiner, D. Tsarkov, I. Horrocks, Framework for an automated comparison of description logic reasoners, in: Proc. International Semantic Web Conference (ISWC 2006), USA, 2006, pp. 654–667.
[44] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, W.H. Freeman, 1979.
[45] M. Gebser, J. Pührer, T. Schaub, H. Tompits, A meta-programming technique for debugging answer-set programs, in: Proc. 23rd AAAI Conf. Artificial Intelligence, AAAI, Chicago, Illinois, USA, 2008, pp. 448–453.
[46] M. Gelfond, V. Lifschitz, Classical negation in logic programs and disjunctive databases, New Gener. Comput. 9 (1991) 365–385.
[47] L. Giordano, V. Gliozzi, N. Olivetti, G.L. Pozzato, A non-monotonic description logic for reasoning about typicality, Artif. Intell. 195 (2013) 165–202.
[48] S. Huang, J. Hao, D. Luo, Incoherency problems in a combination of description logics and rules, J. Appl. Math. (2014) 604753.
[49] S. Huang, Q. Li, P. Hitzler, Reasoning with inconsistencies in hybrid MKNF knowledge bases, Log. J. IGPL 21 (2) (2013) 263–290.
[50] T. Kaminski, M. Knorr, J. Leite, Efficient paraconsistent reasoning with ontologies and rules, in: Q. Yang (Ed.), Proc. 24th International Joint Conf. Artificial Intelligence (IJCAI 2015), AAAI Press/IJCAI, Argentina, 2015, pp. 3098–3105.
[51] Y. Kazakov, M. Krötzsch, F. Simancik, The incredible ELK – from polynomial procedures to efficient reasoning with $\mathcal{EL}$ ontologies, J. Autom. Reason. 53 (1) (2013) 1–61.
[52] M. Knorr, J.J. Alferes, P. Hitzler, Local closed world reasoning with description logics under the well-founded semantics, Artif. Intell. 175 (9–10) (2011) 1528–1554.
[53] T. Kotek, M. Simkus, H. Veith, F. Zuleger, Towards a description logic for program analysis: extending $\mathcal{ALCQIO}$ with reachability, in: Informal Proc. 27th International Workshop on Description Logics (DL 2014), Vienna, Austria, 2014, pp. 591–594.
[54] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D.F. Savo, Inconsistency-tolerant semantics for description logics, in: Proc. 4th Conf. Rules and Web Reasoning (RR 2010), Italy, 2010, pp. 103–117.
[55] D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, D.F. Savo, Query rewriting for inconsistent *DL-Lite* ontologies, in: Proc. 5th International Conf. Web Reasoning and Rule Systems (RR 2011), Ireland, 2011, pp. 155–169.
[56] D. Lembo, V. Santarelli, D.F. Savo, A graph-based approach for classifying OWL 2 QL ontologies, in: Proc. 26th International Workshop on Description Logics (DL 2013), Germany, 2013, pp. 747–759.
[57] V. Levenshtein, Binary codes capable of correcting deletions, insertions and reversals, Sov. Phys. Dokl. 10 (1966) 707.
[58] M.K. Levin, A. Ruttenberg, A.M. Masci, L.G. Cowell, owl_cpp, a C++ library for working with OWL ontologies, in: Proc. 2nd International Conf. Biomedical Ontology, USA, 2011, pp. 255–257.
[59] T. Lukasiewicz, A novel combination of answer set programming with description logics for the semantic web, IEEE Trans. Knowl. Data Eng. 22 (11) (2010) 1577–1592.
[60] Y. Ma, P. Hitzler, Z. Lin, Paraconsistent reasoning for expressive and tractable description logics, in: Proc. 21st International Workshop on Description Logics (DL 2008), Germany, 2008.
[61] V.W. Marek, M. Truszczyński, Autoepistemic logic, J. ACM 38 (3) (1991) 588–619.
[62] M.V. Martinez, C. Molinaro, V.S. Subrahmanian, L. Amgoud, A General Framework for Reasoning on Inconsistency, Springer Briefs in Computer Science, Springer, 2013.
[63] M.V. Martinez, F. Parisi, A. Pugliese, G.I. Simari, V.S. Subrahmanian, Policy-based inconsistency management in relational databases, Int. J. Approx. Reason. 55 (2) (2014) 501–528.
[64] G. Masotti, R. Rosati, M. Ruzzi, Practical Abox cleaning in *DL-Lite* (progress report), in: Proc. 24th International Workshop on Description Logics (DL 2011), Spain, 2011.
[65] B. Motik, I. Horrocks, U. Sattler, Adding integrity constraints to OWL, in: Proc. Workshop on OWL: Experiences and Directions (OWLED 2007), Austria, 2007.
[66] B. Motik, R. Rosati, Reconciling description logics and rules, J. ACM 57 (5) (2010) 1–62.
[67] N.T. Nguyen, Advanced Methods for Inconsistent Knowledge Management, Advanced Information and Knowledge Processing, Springer, 2008.
[68] J. Oetsch, J. Pührer, H. Tompits, Stepwise debugging of description-logic programs, in: Correct Reasoning – Essays on Logic-Based AI in Honour of Vladimir Lifschitz, 2012, pp. 492–508.
[69] Ö.L. Özçep, R. Möller, Combining DL-Lite with spatial calculi for feasible geo-thematic query answering, in: Proc. 25th International Workshop on Description Logics, (DL 2012), Rome, in: CEUR Workshop Proc., vol. 846, 2012, http://CEUR-WS.org.
[70] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, J. Data Semant. 10 (2008) 133–173.
[71] J. Pührer, Stepwise debugging in answer-set programming: theoretical foundations and practical realisation, PhD thesis, Vienna University of Technology, Vienna, Austria, 2014.
[72] J. Pührer, S. Heymans, T. Eiter, Dealing with inconsistency when combining ontologies and rules using DL-programs, in: Proc. 7th Extended Semantic Web Conference (ESWC 2010), Part I, Springer, Greece, 2010, pp. 183–197.
[73] R. Reiter, A theory of diagnosis from first principles, Artif. Intell. 32 (1) (1987) 57–95.
[74] R. Rosati, M. Ruzzi, M. Graziosi, G. Masotti, Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies, in: Proc. 11th International Semantic Web Conference (ISWC 2012), USA, 2012, pp. 337–349.
[75] C. Sakama, K. Inoue, Paraconsistent stable semantics for extended disjunctive programs, J. Log. Comput. 5 (3) (1995) 265–285.
[76] C. Sakama, K. Inoue, An abductive framework for computing knowledge base updates, Theory Pract. Log. Program. 3 (6) (2003) 671–713.
[77] Y.-D. Shen, Well-supported semantics for description logic programs, in: Proc. 22nd International Joint Conf. Artificial Intelligence (IJCAI 2011), IJCAI/AAAI Press, Spain, 2011, pp. 1081–1086.
[78] S. Staworko, J. Chomicki, J. Marcinkowski, Prioritized repairing and consistent query answering in relational databases, Ann. Math. Artif. Intell. 64 (2–3) (2012) 209–246.
[79] D. Stepanova, Inconsistencies in hybrid knowledge bases, PhD thesis, Vienna University of Technology, Vienna, Austria, 2015.
[80] U. Straccia, Default inheritance reasoning in hybrid KL-ONE-style logics, in: Proc. 13th International Joint Conf. Artificial Intelligence (IJCAI 1993), France, 1993, pp. 676–681.
[81] T. Syrjänen, Debugging inconsistent answer-set programs, in: Proceedings of the 11th International Workshop on Nonmonotonic Reasoning (NMR 2006), University of Clausthal, Department of Informatics, 2006, pp. 77–83, Technical Report IfI-06-04.
[82] Y. Wang, J.-H. You, L.-Y. Yuan, Y.-D. Shen, Loop formulas for description logic programs, Theory Pract. Log. Program. 10 (4–6) (2010) 531–545.
[83] G. Xiao, Inline evaluation of hybrid knowledge bases, PhD thesis, Vienna University of Technology, Vienna, Austria, 2014.