Journal of Artificial Intelligence Research 56 (2016) 463-515 Submitted 12/15; published 07/16

Computing Repairs of Inconsistent DL-Programs over $\mathcal{EL}$ Ontologies

Thomas Eiter EITER@KR.TUWIEN.AC.AT
Michael Fink FINK@KR.TUWIEN.AC.AT
Daria Stepanova DASHA@KR.TUWIEN.AC.AT
Institut für Informationssysteme, TU Wien,
Favoritenstraße 9-11, 1040 Vienna, Austria

Abstract

Description Logic (DL) ontologies and non-monotonic rules are two prominent Knowledge Representation (KR) formalisms with complementary features that are essential for various applications. Nonmonotonic Description Logic (DL) programs combine these formalisms thus providing support for rule-based reasoning on top of DL ontologies using a well-defined query interface represented by so-called DL-atoms. Unfortunately, interaction of the rules and the ontology may incur inconsistencies such that a DL-program lacks answer sets (i.e., models), and thus yields no information. This issue is addressed by recently defined repair answer sets, for computing which an effective practical algorithm was proposed for $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ ontologies that reduces a repair computation to constraint matching based on so-called support sets. However, the algorithm exploits particular features of $\mathit{DL\text{-}Lite}_{\mathcal{A}}$ and can not be readily applied to repairing DL-programs over other prominent DLs like $\mathcal{EL}$. Compared to $\mathit{DL\text{-}Lite}_{\mathcal{A}}$, in $\mathcal{EL}$ support sets may neither be small nor only few support sets might exist, and completeness of the algorithm may need to be given up when the support information is bounded. We thus provide an approach for computing repairs for DL-programs over $\mathcal{EL}$ ontologies based on partial (incomplete) support families. The latter are constructed using datalog query rewriting techniques as well as ontology approximation based on logical difference between $\mathcal{EL}$-terminologies. We show how the maximal size and number of support sets for a given DL-atom can be estimated by analyzing the properties of a support hypergraph, which characterizes a relevant set of TBox axioms needed for query derivation. We present a declarative implementation of the repair approach and experimentally evaluate it on a set of benchmark problems; the promising results witness practical feasibility of our repair approach.

1. Introduction

Description Logics (DLs) are a powerful formalism for Knowledge Representation (KR) that is used to formalize domains of interest by describing the meaning of terms and relationships between them. They are well-suited for terminological modelling in contexts such as, the Semantic Web, data integration and ontology-based data access (Calvanese, De Giacomo, Lenzerini, Lembo, Poggi, & Rosati, 2007b; Calvanese, De Giacomo, Lembo, Lenzerini, Poggi, & Rosati, 2007a), reasoning about actions (Baader, Lutz, Milicic, Sattler, & Wolter, 2005), spatial reasoning (Özccep & Möller, 2012), or runtime verification and program analysis (Baader, Bauer, & Lippmann, 2009; Kotek, Simkus, Veith, & Zuleger, 2014), to mention a few.

As most DLs are fragments of classical first-order logic, they have some shortcomings for modelling application settings, where nonmonotonicity or closed-world reasoning needs to be expressed. Rules as in nonmonotonic logic programming offer these features. In addition, they serve well as a tool for declaring knowledge and reasoning about individuals, and for modelling nondeterminism in model generation as possible in Answer Set Programming. To get the best out of the two

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

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

Figure 1: DL-program Π over a policy ontology

worlds of DLs and nonmonotonic rules, the natural idea of combining them led to a number of approaches for such a combination, which are often called hybrid knowledge bases; see the work of Motik and Rosati (2010) and references therein. Among them, *Nonmonotonic Description Logic (DL-)programs* (Eiter, Ianni, Lukasiewicz, Schindlauer, & Tompits, 2008) are a prominent approach in which so-called DL-atoms serve as query interfaces to the ontology in a loose coupling and enable a bidirectional information flow between the rules and the ontology. The possibility to add information from the rules part prior to query evaluation allows for adaptive combinations. However, the loose interaction between rules and ontology can easily lead to inconsistency, that is to a lack of models or answer sets.

Example 1 *Consider the DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ in Figure 1 formalizing an access policy over an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ (Bonatti, Faella, & Sauro, 2010), whose taxonomy (TBox) $\mathcal{T}$ is given by (1)-(3), while (4)-(9) is a sample data part (ABox) $\mathcal{A}$. Besides facts (10), (11) and a simple rule (12), the rule part $\mathcal{P}$ contains defaults (13), (14) expressing that staff members are granted access to project files unless they are blacklisted, and a constraint (15), which forbids that owners of project information lack access to it. Both parts, $\mathcal{P}$ and $\mathcal{O}$, interact via DL-atoms such as $\mathrm{DL}[\mathit{Project} \uplus \mathit{projfile};\ \mathit{StaffRequest}](X)$. The latter specifies a temporary update of $\mathcal{O}$ via the operator $\uplus$, prior to querying it; i.e. additional assertions $\mathit{Project}(c)$ are considered for each individual c, such that $\mathit{projfile}(c)$ is true in an interpretation of $\mathcal{P}$, before all instances X of StaffRequest are retrieved from $\mathcal{O}$. Inconsistency arises as john, the chief of project $\mathit{p1}$ and owner of its files, has no access to them.*

Inconsistency is a well-known problem in logic-based and data intensive systems, and the problem of treating logically contradicting information has been studied in various fields, e.g. belief revision (Alchourrón, Gärdenfors, & Makinson, 1985; Gärdenfors & Rott, 1995), knowledge base updates (Eiter, Erdem, Fink, & Senko, 2005), diagnosis (Reiter, 1987), ontology based data access (Lembo, Lanzerini, Rosati, Ruzzi, & Savo, 2015), nonmonotonic reasoning (Brewka, 1989; Sakama & Inoue, 2003), and many others; (cf. Bertossi, Hunter, & Schaub, 2005; Nguyen, 2008; Martinez, Molinaro, Subrahmanian, & Amgoud, 2013; Bertossi, 2011). In hybrid formalisms so far inconsistency management has concentrated mostly on inconsistency tolerance. For instance, for MKNF knowledge bases paraconsistent semantics was developed by Knorr, Alferes, and Hitzler (2008), Huang, Li, and Hitzler (2013) and Kaminski, Knorr, and Leite (2015). For DL-programs inconsistency tolerance issues were targeted by Fink (2012), where a paraconsistent semantics based on the Logic of Here and There was introduced. Furthermore, Pührer, Heymans, and Eiter (2010) considered suppressing certain problematic DL-atoms. These approaches aimed at reasoning in an inconsistent system rather then making required changes to the system to arrive at a consistent state. This is in contrast to repair techniques that have been recently developed by Eiter, Fink, and Stepanova (2013, 2014d).

In the theoretical framework for repairing inconsistent DL-programs proposed by Eiter et al. (2013), the ontology ABox (a likely source of errors) is changed such that the modified DL-program has answer sets, called *repair answer sets*. Different repair options including deletion of ABox formulas and various restricted forms of addition have been considered together with a naive algorithm for computing repair answer sets which lacked practicality.

An effective repair algorithm in which all DL-atoms can be decided without dynamic ontology access was presented by Eiter, Fink, and Stepanova (2015). It is based on support sets (Eiter, Fink, Redl, & Stepanova, 2014b) for DL-atoms, the portions of the input that together with the ABox determine the truth value of the DL-atom. The algorithm exploits complete support families, i.e. stocks of support sets from which the value of a DL-atom under every interpretation can be determined, such that an (repeated) ontology access can be avoided. The approach works well for $DL\text{-}Lite_{\mathcal{A}}$, which is a prominent tractable DL, since complete support families are small and easy to compute.

However, unfortunately, for other DLs this approach is not readily usable, because in general there can be large or infinite support families. This applies even for $\mathcal{EL}$, which is another well-known important DL that offers tractable reasoning and is widely applied in many domains, including biology, (cf. e.g., Schulz, Cornet, & Spackman, 2011; Aranguren, Bechhofer, Lord, Sattler, & Stevens, 2007), medicine (Steve, Gangemi, & Mori, 1995), chemistry, policy management, etc. Due to the features of $\mathcal{EL}$ that include range restrictions and concept conjunctions on the left-hand side of inclusion axioms, a DL-atom accessing an $\mathcal{EL}$ ontology can have arbitrarily large and infinitely many support sets in general. While the latter is excluded for acyclic TBoxes, often occurring in practice (Gardiner, Tsarkov, & Horrocks, 2006), complete support families can be still very large, and constructing as well as managing them might be impractical. This obstructs the deployment of the approach proposed by Eiter et al. (2014d) to $\mathcal{EL}$ ontologies. In this paper we tackle this issue and develop repair computation techniques for DL-programs over ontologies in $\mathcal{EL}$. We focus on $\mathcal{EL}$, since apart from being simple and widely used, this DL is well-researched, and available effective algorithms for query rewriting and other important reasoning can be readily used.

More specifically, we introduce here a more general algorithm for repair answer set computation that operates on partial (incomplete) support families along with techniques how such families can be effectively computed. The problem of computing repair answer sets for DL-programs over $\mathcal{EL}$ ontologies is Σ_2^P-complete (in its formulation as a decision problem; we refer to the work of Stepanova (2015) for details on the complexity).

Our contributions and advances over previous works by Eiter et al. (2014b, 2014d, 2015) are summarized as follows:

- For effective computation of repair answer sets we exploit the support sets of Eiter et al. (2014d). In contrast to the approaches of Eiter et al. (2014d, 2015), however, where TBox classification is invoked, we use datalog rewritings of queries for computing support sets (see also Hansen, Lutz, Seylan, & Wolter, 2014). We introduce the notion of partial support families, with which ontology reasoning access can be completely eliminated.

- As in general constructing complete support families is not always feasible for $\mathcal{EL}$ ontologies, we provide novel methods for computing partial support families by exploiting ontology approximation techniques based on the logical difference between $\mathcal{EL}$-terminologies as considered by Konev, Ludwig, Walther, and Wolter (2012) and Ludwig and Walther (2014).

- To capture restricted classes of TBoxes, for which complete support families can still be effectively computed, we consider a *support hypergraph* for DL-atoms, which is inspired by ontology hypergraphs (Nortje, Britz, & Meyer, 2013; Ecke, Ludwig, & Walther, 2013). The support hypergraph serves to characterize the TBox parts that are relevant for deriving the query. The analysis of support hypergraphs allows us to estimate the maximal size and number of support sets that is needed to form a complete support family.

- We generalize the algorithm for repair answer set computation proposed by Eiter et al. (2014d) such that $\mathcal{EL}$ ontologies can be handled. The novel algorithm operates on partial support families, and in principle can be applied to the ontologies in any DLs beyond $\mathcal{EL}$. 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; however completeness may be ensured by a simple extension.

- We provide a system prototype with a declarative realization of the novel algorithm for repair answer set computation. Our repair approach has been evaluated using some novel benchmarks; the results show very promising potential of the proposed approach.

Organization. The rest of the paper is organized as follows. In Section 2, we recall basic notions and preliminary results. Section 3 deals with support sets and their computation, while Section 4 discusses partial support family construction based on TBox approximation techniques. In Section 5 we analyze properties of a support hypergraph for estimating the maximal size and number of support sets in a complete support family for a DL-atom. In Section 6, the algorithm for repair answer set computation and its declarative implementation are presented. Experiments are presented in Section 7, followed by a discussion of related work in Section 8 and concluding remarks in Section 9.

2. Preliminaries

In this section, we recall basic notions of Description Logics, where we focus on $\mathcal{EL}$ (Baader, Brandt, & Lutz, 2005), and DL-programs (Eiter et al., 2008). For more background on Description Logics, (see Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2003).

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

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

Figure 2: Normalized TBox

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 $\mathcal{EL}$, 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 $R \in \mathbf{R}$ an atomic role:

$$C \rightarrow A \mid \top \mid C \sqcap C \mid \exists R.C$$

In $\mathcal{EL}$, TBox axioms are of the form $C_1 \sqsubseteq C_2$ (also called generalized concept inclusion axioms, *GCIs*), where C_1, C_2 are $\mathcal{EL}$-concepts. ABox formulas are of the form $A(c)$ or $R(c,d)$, where $A \in \mathbf{C}$, $R \in \mathbf{R}$, and $c, d \in \mathbf{I}$. In the sequel, we use P as a generic predicate from $\mathbf{C} \cup \mathbf{R}$ (if the distinction is immaterial).

An example of an $\mathcal{EL}$ ontology is given in Figure 1.

Definition 2 (normalized TBox) *A TBox is* normalized*, if all of its axioms have one of the following forms:*

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

where A_1, A_2, A_3 are atomic concepts.

E.g., the axiom (1) in Example 1 is in normal form, while the axioms (2) and (3) are not. For any $\mathcal{EL}$ TBox, an equivalent TBox in normal form is constructible in linear time (Stuckenschmidt, Parent, & Spaccapietra, 2009) (over an extended signature)[1] (Baader et al., 2005).

A special class of TBoxes widely studied in literature are $\mathcal{EL}$*-terminologies*, defined as follows:

Definition 3 ($\mathcal{EL}$-terminology) *An $\mathcal{EL}$-*terminology *is an $\mathcal{EL}$ TBox $\mathcal{T}$, satisfying the following conditions:*

(1) $\mathcal{T}$ consists of axioms of the forms $A \equiv C$ and $A \sqsubseteq C$, where A is atomic and C is an arbitrary $\mathcal{EL}$ concept;

(2) no concept name occurs more then once on the left hand side of axioms in $\mathcal{T}$.

For example, the TBox of the ontology in Figure 1 is an $\mathcal{EL}$-terminology.

Semantics. The semantics of DL ontologies is based on first-order interpretations (Baader et al., 2005). 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

1. Linear complexity results are obtained under the standard assumption in DLs that each of the atomic concepts is of constant size, i.e., the length of a binary string representing an atomic concept does not depend on the particular knowledge base.

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 $\Delta^{\mathcal{I}}$. The interpretation $\mathcal{I}$ extends inductively to non-atomic concepts C and roles R according to the concept resp. role constructors; as for $\mathcal{EL}$, $(\exists R.C)^{\mathcal{I}} = \{o_1 \mid \langle o_1, o_2 \rangle \in R^{\mathcal{I}}, o_2 \in C^{\mathcal{I}}\}$ and $(C \sqcap D)^{\mathcal{I}} = \{o_1 \mid o_1 \in C^{\mathcal{I}}, o_1 \in D^{\mathcal{I}}\}$.

Satisfaction of an axiom resp. assertion ω w.r.t. an interpretation $\mathcal{I}$, i.e. $\mathcal{I} \models \omega$, is as follows: (i) $\mathcal{I} \models C \sqsubseteq D$, if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$; (ii) $\mathcal{I} \models C(a)$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$; (iii) $\mathcal{I} \models R(a, b)$, if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$. 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 $\mathcal{T}$ (respectively an ABox $\mathcal{A}$, an ontology $\mathcal{O}$) is *satisfiable* (or *consistent*), if some interpretation $\mathcal{I}$ satisfies it. We call an ABox $\mathcal{A}$ *consistent with* a TBox $\mathcal{T}$, if $\mathcal{T} \cup \mathcal{A}$ is consistent.

Since negation is neither available nor expressible in $\mathcal{EL}$, all $\mathcal{EL}$ ontologies are consistent.

Example 4 *The ontology $\mathcal{O}$ in Figure 1 is consistent; a satisfying interpretation $\mathcal{I} = \langle \Delta^{\mathcal{I}}, \cdot^{\mathcal{I}} \rangle$ exists, where $\Delta^{\mathcal{I}} = \{john, read, p1, r1\}$, $Action^{\mathcal{I}} = \{read\}$, $Blacklisted^{\mathcal{I}} = Staff^{\mathcal{I}} = \{john\}$, $hasSubject^{\mathcal{I}} = \{r1, john\}$, $StaffRequest^{\mathcal{I}} = BlacklistedStaffRequest^{\mathcal{I}} = \{r1\}$, $hasAction^{\mathcal{I}} = \{r1, read\}$, $hasTarget^{\mathcal{I}} = \{r1, p1\}$.*

Throughout the paper, we consider ontologies in $\mathcal{EL}$ under the unique name assumption (UNA), i.e., ${o_1}^{\mathcal{I}} \neq {o_2}^{\mathcal{I}}$ whenever $o_1 \neq o_2$ holds in any interpretation. However, our results carry over to ontologies without UNA, as it is not hard to see that the UNA has for $\mathcal{EL}$ no effect on query answering, (cf. Lutz, Toman, & Wolter, 2009).

2.2 DL-Programs

A DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is 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 a set $\mathcal{C}$ of constant symbols and a (finite) set $\mathbf{P}$ of predicate symbols (called *lp predicates*) of non-negative arities, 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 $\mathcal{C} = \mathbf{I}$.

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

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

where not is *negation as failure (NAF)*[2] 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*. The rule is a *constraint*, if $n = 0$, and *normal*, if $n \leq 1$. We call $H(r) = \{a_1, \ldots, a_n\}$ the *head* of r, and $B(r) = \{b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m\}$ the *body* of r. $B^+(r) = \{b_1, \ldots, b_k\}$ and $B^-(r) = \{b_{k+1}, \ldots, b_m\}$ denote the positive and the negative parts of $B(r)$ respectively.

A *DL-atom* $d(\vec{t})$ 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, $1 \leq i \leq m$, $S_i \in \mathbf{C} \cup \mathbf{R}$, $op_i \in \{\uplus\}$ is an *update operator*, and $p_i \in \mathbf{P}$ is an *input predicate* of the same arity as S_i; intuitively, $op_i = \uplus$ increases S_i by the extension of p_i;

2. Strong negation $\neg a$ can be added resp. emulated as usual (Eiter et al., 2008).

(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) $C_1 \sqsubseteq C_2$ and $\vec{t} = \epsilon$.

Note that inclusion DL-queries of the form $C_1 \sqsubseteq C_2$ can be easily reduced to instance queries.[3] Thus for simplicity, we consider in this work only instance DL-queries.

Example 5 *Consider a DL-atom* $\mathrm{DL}[\mathit{Project} \uplus \mathit{projfile};\, \mathit{StaffRequest}](X)$ *in the rule (13) of* Π *in Figure 1 for* $X = r1$*. It has a DL-query* $\mathit{StaffRequest}(r1)$*; its list* $\lambda = \mathit{Project} \uplus \mathit{projfile}$ *contains an input predicate* $\mathit{projfile}$ *which extends the ontology predicate* $\mathit{Project}$ *via an update operator* $\uplus$*.*

Semantics. The semantics of a DL-program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ is given in terms of its grounding $gr(\Pi) = \langle \mathcal{O}, gr(\mathcal{P}) \rangle$ over $\mathcal{C}$, i.e., $gr(\mathcal{P}) = \bigcup_{r \in \mathcal{P}} gr(r)$ contains all possible ground instances of all 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 for $\Sigma_\mathcal{P}{=}\langle \mathcal{C}, \mathbf{P} \rangle$, i.e. the set of all ground atoms over $\Sigma_\mathcal{P}$; I satisfies an lp- or DL-atom a, if

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

$$\lambda^I(d) = \bigcup_{i=1}^{m} A_i(I) \text{ and } A_i(I) = \{S_i(\vec{t}) \mid p_i(\vec{t}) \in I\}, 1 \leq i \leq m. \tag{3}$$

Satisfaction of a DL-rule r (resp. a set $\mathcal{P}$ of rules) by a Herbrand interpretation I of $\Pi = \langle \mathcal{P}, \mathcal{O} \rangle$ is then as usual, where I satisfies $\mathit{not}\ b_j$, if I does not satisfy b_j; I satisfies Π, if it satisfies each $r \in \mathcal{P}$. By $I \models^\mathcal{O} \omega$ we denote that I satisfies (is a model of) an object ω, where ω can be an (DL-)atom, a rule or a set of rules; the superscript $\mathcal{O}$ of $\models$ specifies the ontology on which DL-atoms are evaluated. A model I of ω is *minimal*, if no model I' of ω exists such that $I' \subset I$.

Example 6 *The DL-atom* $d = \mathrm{DL}[\mathit{Project} \uplus \mathit{projfile};\, \mathit{StaffRequest}](r1)$ *is satisfied by the interpretation* $I = \{\mathit{projfile}(p1), \mathit{hasowner}(p1, \mathit{john})\}$*, since* $\mathcal{O} \models \mathit{StaffRequest}(r1)$*. For* $\mathcal{O}' = \mathcal{O} \backslash \{\mathit{StaffReqeust}(r1)\}$ *it still holds that* $I \models^{\mathcal{O}'} d$*, as* $\mathcal{O}' \cup \lambda^I(d) \models \mathit{StaffRequest}(r1)$*.*

Repair Answer Sets. Various semantics for DL-programs extend the answer set semantics of logic programs (Gelfond & Lifschitz, 1991) to DL-programs, (e.g., Eiter et al., 2008; Lukasiewicz, 2010; Wang, You, Yuan, & Shen, 2010; Shen, 2011). We concentrate here on *weak answer sets* (Eiter et al., 2008), which treat DL-atoms like atoms under NAF, and *flp-answer sets* (Eiter, Ianni, Schindlauer, & Tompits, 2005), which obey a stronger foundedness condition. Both are like answer sets of an ordinary logic program interpretations that are minimal models of a program reduct, which intuitively captures that assumption-based application of the rules can reconstruct the interpretation.

The *weak-reduct* $\mathcal{P}^{I,\mathcal{O}}_{weak}$ of $\mathcal{P}$ relative to $\mathcal{O}$ and to $I \subseteq HB_\Pi$ results from $gr(\mathcal{P})$ by deleting (i) all rules r such that either $I \not\models^\mathcal{O} d$ for some DL-atom $d \in B^+(r)$, or $I \models^\mathcal{O} l$ for some $l \in B^-(r)$; (ii) all DL-atoms in $B^+(r)$ and all literals in $B^-(r)$.

3. Evaluating $d = \mathrm{DL}[\lambda;\, C_1 \sqsubseteq C_2]()$ over $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ reduces to evaluating $d' = \mathrm{DL}[\lambda;\, A_{C_2}](a)$ over $\mathcal{O}' = \mathcal{T} \cup \{A_{C_1} \sqsubseteq C_1, C_2 \sqsubseteq A_{C_2}\} \cup \mathcal{A} \cup \{A_{C_1}(a)\}$, where a is a fresh constant and A_{C_1}, A_{C_2} are fresh concepts (similar as in TBox normalization).

The *flp-reduct* $\mathcal{P}_{\mathit{flp}}^{I,\mathcal{O}}$ of $\mathcal{P}$ results from $gr(\mathcal{P})$ by deleting all rules r, whose bodies are not satisfied by I, i.e. $I \not\models^{\mathcal{O}} b_i$, for some b_i, $1 \leq i \leq k$ or $I \models^{\mathcal{O}} b_j$, for some b_j, $k < j \leq m$. We illustrate the notions on an example.

Example 7 *Let $\mathcal{O}$ be as in Figure 1, and let the rule set $\mathcal{P}$ contain the facts (10), (11) and the rules (12), (13) with X, Y, Z instantiated to $r1, john, p1$ respectively. Consider the interpretation $I = \{projfile(p1), hasowner(p1, john), chief(john), grant(r1)\}$. While the flp-reduct $\mathcal{P}_{\mathit{flp}}^{I,\mathcal{O}}$ contains all rules of $\mathcal{P}$, in the weak-reduct $\mathcal{P}_{weak}^{I,\mathcal{O}}$ the rule (13) is replaced by the fact $grant(r1)$.*

Definition 8 (x-deletion repair answer set) *An interpretation I is an x-*deletion repair answer set *of $\Pi = \langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$ for $x \in \{\mathit{flp}, weak\}$, if it is a minimal model of $\mathcal{P}_x^{I,\mathcal{T}\cup\mathcal{A}'}$, where $\mathcal{A}' \subseteq \mathcal{A}$; any such $\mathcal{A}'$ is called an x-*deletion repair *of Π. If $\mathcal{A}' = \mathcal{A}$, then I is a standard x-answer set.*

Example 9 *$I = \{projfile(p1), chief(john), hasowner(p1, john), grant(john)\}$ is both a weak and flp-repair answer set of Π in Example 1 with a repair $\mathcal{A}' = \mathcal{A} \backslash \{Blacklisted(john)\}$.*

Notation. We denote for any normal logic program $\mathcal{P}$ by $AS(\mathcal{P})$ the set of all answer sets of $\mathcal{P}$, and for any DL-program Π by $AS_x(\Pi)$ (resp. $RAS_x(\Pi)$) the set of all x-answer sets (resp. x-repair answer sets) of Π.

In general an *flp*-answer set is a $weak$-answer set, but not vice versa, i.e. *flp*-answer sets are a more restrictive notion; however, in many cases $weak$ and *flp* answer sets coincide. For more information on the reducts, see works by Eiter et al. (2008) and Wang et al. (2010).

Shifting Lemma. To simplify matters and avoid dealing with the logic program predicates separately, we shall shift as Eiter et al. (2014d) the lp-input of DL-atoms to the ontology. Given a DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$ and $P \uplus p \in \lambda$, we call $P_p(c)$ an *input assertion for* d, where P_p is a fresh ontology predicate and $c \in \mathcal{C}$; $\mathcal{A}_d$ is the set of all such assertions. For a TBox $\mathcal{T}$ and a DL-atom d, we let $\mathcal{T}_d = \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\}$, and for an interpretation I, we let $\mathcal{O}_d^I = \mathcal{T}_d \cup \mathcal{A} \cup \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\}$. We then have:

Proposition 10 (Eiter et al., 2014d) *For every $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$, DL-atom $d = \mathrm{DL}[\lambda; Q](\vec{t})$ and interpretation I, it holds that $I \models^{\mathcal{O}} d$ iff $I \models^{\mathcal{O}_d^I} \mathrm{DL}[\epsilon; Q](\vec{t})$ iff $\mathcal{O}_d^I \models Q(\vec{t})$.*

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

3. Support Sets for DL-Atoms

In this section, we recall support sets for DL-atoms by Eiter et al. (2014b), which are an effective optimization means for (repair) answer set computation (Eiter et al., 2014d). 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 $\mathcal{O}$, we can conclude that $I \models^{\mathcal{O}} Q(\vec{t})$. Basically, our method suggests precomputing support sets for each DL-atom at the nonground level. During DL-program evaluation, for each candidate interpretation ground instances of support sets are computed, which help to prune the search space for (repair) answer sets.

Exploiting Proposition 10 we have the following definition of support sets using only ontology predicates.

Definition 11 (ground support sets) *Given a ground DL-atom $d = DL[\lambda; Q](\vec{t})$, a set $S \subseteq \mathcal{A} \cup \mathcal{A}_d$ 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 are grouped into families of support sets or simply *support families*. More formally,

Definition 12 (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. $\mathcal{O}$.*

In general and for $\mathcal{EL}$ in particular, even $\subseteq$-minimal support sets can be arbitrarily large and infinitely many support sets may exist (not for acyclic TBoxes $\mathcal{T}$, where still exponentially many support sets are possible). However, we nonetheless can exploit them for the repair answer set computation algorithms in Section 6.

Support sets are linked to interpretations by the following notion.

Definition 13 (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 $p(c) \in I$.*

Example 14 *The DL-atom $d = \mathrm{DL}[Project \uplus Projfile;\ StaffRequest](r1)$ from Figure 1 has two support sets: $S_1 = \{StaffRequest(r1)\}$ and $S_2 = \{hasSubject(r1, john), Project_{projfile}(p1), Staff(john), hasAction(r1, read), Action(read)\}$. S_1 is coherent with any interpretation, while S_2 is coherent only with interpretations $I \supseteq \{projfile(p1)\}$.*

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

Proposition 15 *Let $d = DL[\lambda; Q](\vec{t})$ 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.*

Using a sufficient portion of 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 the 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.

Definition 16 (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 $V \supseteq \vec{X}$ is a set of distinct variables and $\mathcal{C}$ is a set of constants. A* nonground support set *for d w.r.t. $\mathcal{T}$ is a set $S = \{P_1(\vec{Y_1}), \dots, P_k(\vec{Y_k})\}$ of atoms such that* *(i)* $\vec{Y}_1, \ldots, \vec{Y}_k \subseteq V$ *and*

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

For any ontology $\mathcal{O} = \mathcal{T} \cup \mathcal{A}_\mathcal{C}$*, we denote by* $\mathbf{Supp}_\mathcal{O}(d)$ *the set of all nonground support sets for* d *w.r.t.* $\mathcal{T}$*.*

Here $\mathcal{A}_\mathcal{C}$ takes care of any possible ABox, by considering the largest ABox (since $\mathcal{O} \subseteq \mathcal{O}'$ implies that $Supp_\mathcal{O}(d) \subseteq Supp_{\mathcal{O}'}(d)$).

Example 17 *For* $d = \mathrm{DL}[Project \uplus projfile;\ StaffRequest](X)$ *the set* $S_1 = \{StaffRequest(X)\}$ *is a nonground support set, and likewise the set* $S_2 = \{Action(W), Staff(Y), hasSubject(X, Y),$ $hasTarget(X, Z),$ $Project_{projfile}(Z), hasAction(X, W)\}$*.*

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

Definition 18 (complete support family) *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. an ontology* $\mathcal{O}$ *is* complete*, if for every support set* $S \in Supp_\mathcal{O}(d(\vec{X}\theta))$*, where* θ*:* $\vec{X} \to \mathcal{C}$*, some* $S' \in \mathbf{S}$ *and an extension* $\theta' : V \to \mathcal{C}$ *of* θ *to* $V \supseteq \vec{X}$ *exist such that* $S = S'\theta'$*.*

Example 19 *Consider the DL-atom* $d(X) = \mathrm{DL}[Project \uplus projfile;\ StaffRequest](X)$ *from Figure 1. The family* $\mathbf{S} = \{S_1, S_2, S_3, S_4, S_5, S_6\}$ *is complete for* d *w.r.t.* $\mathcal{O}$*, where* $hT = hasTarget$*,* $hS = hasSubject$ *and* $hA = hasAction$*:*

- $S_1 = \{StaffRequest(X)\}$*;*
- $S_2 = \{Project(Y), hT(X, Y), hS(X, Z), Staff(Z), hA(X, Z'), Action(Z')\}$*;*
- $S_3 = \{Project_{projfile}(Y), hT(X, Y), hS(X, Z), Staff(Z), hA(X, Z'), Action(Z')\}$*;*
- $S_4 = \{Project(Y), hT(X, Y), hS(X, Z), Blacklisted(Z), hA(X, Z'), Action(Z')\}$*;*
- $S_5 = \{Project_{projfile}(Y), hT(X, Y), hS(X, Z), Blacklisted(Z), hA(X, Z'), Action(Z')\}$*;*
- $S_6 = \{BlacklistedStaffRequest(X)\}$*.* $\square$

We say that two nonground support sets (resp. support families) are *ground-identical*, if their groundings coincide. E.g., the support sets $S_1 = \{P(X), r(X, Y)\}$ and $S_2 = \{P(X), r(X, Z)\}$ are ground-identical for a DL-atom $d(X) = \mathrm{DL}[\lambda;\ Q](X)$, and so are the respective support families $\{S_1\}$ and $\{S_2\}$.

Definition 20 (subsumption) *A nonground support set* S *is* subsumed *by* S'*, denoted by* $S' \subseteq_\theta S$*, if for every ground instance* $S\theta$ *of* S *some ground instance* $S'\theta'$ *of* S' *exists such that* $S'\theta' \subseteq S\theta$*. For nonground support families, we say that* $\mathbf{S}_1$ *is* subsumed *by* $\mathbf{S}_2$*, denoted* $\mathbf{S}_2 \subseteq_\theta \mathbf{S}_1$*, if for each instance* $S\theta$ *of* $S \in \mathbf{S}_1$ *some instance* $S'\theta'$ *of* S' *in* $\mathbf{S}_2$ *exists such that* $S'\theta' \subseteq S\theta$ *holds.*

Example 21 $S = \{BlacklistedStaffRequest(X), hasSubject(X, Y), Blacklisted(Y)\}$ *is a support set for the DL-atom* $d(X) = \mathrm{DL}[Staff \uplus chief;\ BlacklistedStaffRequest](X)$ *w.r.t.* $\mathcal{T}$ *from Figure 1, which is subsumed by* $S' = \{BlacklistedStaffRequest(X)\}$*, i.e.* $S' \subseteq_\theta S$*. Moreover,* $\mathbf{S}' \subseteq_\theta \mathbf{S}$*, where* $\mathbf{S}' = \{S'\}$ *and* $\mathbf{S} = \{S\}$*, while the support families* $\mathbf{S}'' = \{S, S'\}$ *and* $\mathbf{S}''' = \{S,$ $\{BlacklistedStaffRequest(X), hasSubject(X, Z), Blacklisted(Z)\}\}$ *mutually subsume each other.*

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

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

Definition 22 (maximal support set size, $maxsup$) *The* maximal support set size *of a DL-atom d w.r.t. $\mathcal{T}$, denoted by $maxsup(d)$, is the smallest integer $n \geq 0$ such that for every complete non-ground support family* $\mathbf{S}$ *for d w.r.t. $\mathcal{T}$ and support set $S \in \mathbf{S}$ with $|S| > n$, a support set $S' \subseteq_\theta S$ exists for d w.r.t. $\mathcal{T}$ in $\mathbf{Supp}_d(\mathcal{O})$ with $|S'| \leq n$.*

For instance, for the DL-atom d and the TBox $\mathcal{T}$ from Example 19, the maximal support set size is 6, i.e., $maxsup(d) = 6$.

3.1 Computing Support Sets

In this section, we provide methods for constructing nonground support sets. A natural approach for the computation of nonground support sets is to exploit (conjunctive) query answering methods in $\mathcal{EL}$ (e.g., Rosati, 2007; Lutz et al., 2009; Kontchakov, Lutz, Toman, Wolter, & Zakharyaschev, 2010; Stefanoni, Motik, & Horrocks, 2012).

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

Step 1. DL-query Rewriting over the TBox. The first step exploits the rewriting of the DL-query Q of $d(\vec{X})$ over the TBox $\mathcal{T}_d = \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\}$ into a set of datalog rules. At a preprocessing stage, the TBox $\mathcal{T}_d$ is normalized. This technique restricts the syntactic form of TBoxes by decomposing complex into simpler axioms. For this purpose, a set of fresh concept symbols is introduced. Once the normalized form $\mathcal{T}_{dnorm}$ of $\mathcal{T}_d$ is computed, we rewrite the part of the TBox that is relevant for the query Q into a datalog program $Prog_{Q,\mathcal{T}_{dnorm}}$ using the translation given in Table 1, which is a variant of a translation by Pérez-Urbina, Motik, and Horrocks (2010) and Zhao, Pan, and Ren (2009). When rewriting axioms of the form $A_1 \sqsubseteq \exists R.A_2$ (fourth axiom in Table 1), we introduce fresh constants (o_{A_2}) to represent "unknown" objects. A similar rewriting is exploited in the REQUIEM system (Pérez-Urbina et al., 2010), where function symbols are used instead of fresh constants. As a result we obtain:

Lemma 23 *For every data part, i.e., ABox $\mathcal{A}$, and every ground assertion $Q(\vec{c})$, deciding whether $Prog_{Q,\mathcal{T}_{dnorm}} \cup \mathcal{A} \models Q(\vec{c})$ is equivalent to checking $\mathcal{T}_{dnorm} \cup \mathcal{A} \models Q(\vec{c})$.*

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

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

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

be infinitely many of them if $\mathcal{T}$ is cyclic, and still exponentially many for acyclic $\mathcal{T}$; we discuss techniques for computing partial support families in the next section.

Step 3. Support Set Extraction. The last step extracts nonground support sets from the rewritings of Step 2. We select those containing only predicates from $\mathcal{T}_d$ and obtain rules r of the form

$$Q(\vec{X}) \leftarrow P_1(\vec{Y_1}), \ldots, P_k(\vec{Y_k}), P_{k+1_{p_{k+1}}}(\vec{Y}_{k+1}), \ldots, P_{n p_n}(\vec{Y_n}), \tag{4}$$

where each P_i is a native ontology predicate if $1 \leq i \leq k$, and a predicate mirroring lp-input of d otherwise. The bodies of such rules correspond to the support sets for a given DL-atom, i.e.

$$S = \{P_1(\vec{Y_1}), \ldots, P_k(\vec{Y_k}), P_{k+1_{p_{k+1}}}(\vec{Y}_{k+1}), \ldots, P_{n p_n}(\vec{Y_n})\} \tag{5}$$

Now the following holds.

Proposition 24 *Let $d(\vec{X}) = \mathrm{DL}[\lambda; Q](\vec{X})$ be a DL-atom of a program $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ with an $\mathcal{EL}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$. Every set S constructed in Steps 1-3 is a nonground support set for $d(\vec{X})$.*

By the Shifting Lemma, when working with support sets we can focus on the ontology predicates and operate only on them. More specifically, rules of the form (4) for $k \leq n$ fully reflect nonground support sets as of Definition 16, and ground instantiations of such a rule over constants from $\mathcal{C}$ implicitly correspond to ground support sets.

We now illustrate the computation of nonground support sets for DL-atoms over $\mathcal{EL}$ ontologies.

Example 25 *Consider a DL-atom $\mathrm{DL}[Project \uplus projfile;\ StaffRequest](X)$ accessing an $\mathcal{EL}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ from Figure 1. The datalog rewriting for d computed at Step 1 is given in Figure 3. In Step 2 we obtain the following query unfoldings for StaffRequest:*

(1) $StaffRequest(X) \leftarrow StaffRequest(X)$;

(2) $StaffRequest(X) \leftarrow hasAction(X,Y), Action(Y), hasSubject(X,Y'),$
$Staff(Y'), hasTarget(X,Y''), Project_{projfile}(Y'')$;

(3) $StaffRequest(X) \leftarrow hasAction(X,Y), Action(Y), hasSubject(X,Y'),$
$Staff(Y'), hasTarget(X,Y''), Project(Y'')$;

(4) $StaffRequest(X) \leftarrow hasAction(X,Y), Action(Y), hasSubject(X,Y'),$
$Blacklisted(Y'), hasTarget(X,Y''), Project(Y'')$;

(5) $StaffRequest(X) \leftarrow hasAction(X,Y), Action(Y), hasSubject(X,Y'),$
$Blacklisted(Y'), hasTarget(X,Y''), Project_{projfile}(Y'')$.

In Step 3 we thus get from the rule (2) $S_2 = \{hasAction(X,Y), Action(Y), Staff(Y'), hasSubject(X,Y'), hasTarget(X,Y''), Project_{projfile}(Y'')\}$ and from rule (3) $S_3 = \{Action(Y), hasAction(X,Y), Staff(Y'), hasSubject(X,Y'), Project(Y''), hasTarget(X,Y'')\}$. From (1), (4) and (5) the remaining support sets are similarly obtained. □

3.2 Partial Support Families

Finding all support sets for a DL-atom is tightly related to computing all solutions to a logic-based abduction problem. Abduction is an important mode of reasoning widely applied in different areas of AI including planning, diagnosis, natural language understanding and many others (Console, Sapino, & Dupré, 1995). Various variants of this problem were actively studied, e.g. by Eiter, Gottlob, and Leone (1997) and Bienvenu (2008). Unfortunately, most of the practically important problems in the context of abduction are intractable even for restricted propositional theories (Eiter & Makino, 2007). The abduction problem for $\mathcal{EL}$ TBoxes has been considered by Bienvenu (2008), represented by a tuple $\langle \mathcal{T}, \mathcal{H}, O \rangle$, with a TBox $\mathcal{T}$, a set of atomic concepts $\mathcal{H}$ and an atomic concept O. An explanation is a set $\{A_1, \dots, A_n\} \subseteq \mathcal{H}$, such that $\mathcal{T} \models A_1 \sqcap \dots \sqcap A_n \sqsubseteq O$. If the ABox $\mathcal{A} \cup \mathcal{A}_d$ contains only atomic concepts, then computing all nonground support sets for $d = \mathrm{DL}[\lambda;\, Q](X)$ accessing $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ corresponds to an abduction problem $\langle \mathcal{T}_d, \mathsf{sig}(\mathcal{A} \cup \mathcal{A}_d), Q \rangle$. If roles occur in $\mathcal{A} \cup \mathcal{A}_d$, then one has to introduce new fresh concepts to construct the complex concepts as hypothesis, e.g., for $\exists R.A$ an inclusion $C_{\exists R.A} \sqsubseteq \exists R.A$ can be added to $\mathcal{T}_d$, and $C_{\exists R.A}$ to $\mathcal{H}$, where $C_{\exists R.A}$ is a fresh concept.

Unlike for $DL\text{-}Lite_{\mathcal{A}}$, support families for DL-atoms over $\mathcal{EL}$ ontologies have no particular structure; they can be large, and maximal support set size can be exponential in the size of $\mathcal{T}$.

Example 26 *Consider the following acyclic TBox $\mathcal{T}$, which contains the axioms:*

$$\begin{aligned}
&\textit{(1)}\ \exists r.B_0 \sqcap \exists s.B_0 \sqsubseteq B_1 \\
&\textit{(2)}\ \exists r.B_1 \sqcap \exists s.B_1 \sqsubseteq B_2 \\
&\quad \dots \\
&\textit{(n)}\ \exists r.B_{n-1} \sqcap \exists s.B_{n-1} \sqsubseteq B_n
\end{aligned}$$

For $d_1 = \mathrm{DL}[\lambda;\, B_1](X_1)$, the maximal support set size is 4, which is witnessed by

$$S_1 = \{r(X_1, X_2), B_0(X_2), s(X_1, X_3), B_0(X_3)\}.$$

For the DL-atom $d_2 = \mathrm{DL}[\lambda;\, B_2](X_1)$, we have $maxsup(d_2) = 10$, due to $S_2 = \{r(X_1, X_2), r(X_2, X_3), B_0(X_3), s(X_2, X_4), B_0(X_4), s(X_1, X_5), r(X_5, X_6), B_0(X_6), s(X_5, X_7), B_0(X_7)\}$. Moreover, for $d_i = \mathrm{DL}[\lambda;\, B_i](X)$, we have $maxsup(d_i) = maxsup(d_{i-1}) \times 2 + 2$, $1 \leq i \leq n$.

Note that the maximal support set for d_n involves $n + 3$ predicates. Therefore, if the TBox is of the above form, and $|\mathsf{sig}(\mathcal{T})| = k$, a lower bound for the worst case support set size for d is $2^{k-1} + 2 = \Omega(2^k)$, which is single exponential in the size of $\mathcal{T}$. □

While in general many unfoldings can be produced at Step 2, according to recent results of Hansen et al. (2014), complete support families for $\mathcal{EL}$ can be computed for large classes of ontologies. Therefore, we still exploit support families, but unlike Eiter et al. (2014d) we do not require them to be complete, and develop techniques for computing *partial* (i.e. incomplete) support families for DL-atoms. A natural approach in this context is to aim at finding support sets of bounded size. In general, due to cyclic dependencies such as $\exists r.C \sqsubseteq C$, which are possible in $\mathcal{EL}$ but not in $DL\text{-}Lite_{\mathcal{A}}$, support sets can be arbitrary large. An analysis of a vast number of ontologies has revealed that in many realistic cases ontologies do not contain (nor imply) cyclic axioms (Gardiner et al., 2006); we thus assume for practical considerations that the TBox of the ontology in a given DL-program is *acyclic*, i.e., it does not entail inclusion axioms of form $\exists r.C \sqsubseteq C$. However, even under this restriction support sets can be large as Example 26 shows.

If computing complete support families is computationally too expensive, a natural approach is to produce only support sets of a certain size k using e.g. limited program unfolding. When an unfolding branch reaches the depth k, we stop and expand a different branch. Similarly, we can compute a limited number k of support sets by stopping the rule unfolding of the program $Prog_{Q,\mathcal{T}_{d_{norm}}}$ once the k-th support set is produced. An alternative approach, based on TBox approximation techniques, is pursued in the next section.

4. Partial Support Family Construction via TBox Approximation

We now provide practical methods to construct partial support families using TBox approximation.

4.1 TBox Approximation

The approximation of DL ontologies over a source language $\mathcal{L}$ in a different target language $\mathcal{L}'$ is a well-known and important technique in ontology management. Existing approaches for such approximation are roughly divided into *syntactic approaches* and *semantic approaches*. The former, e.g. those by Tserendorj, Rudolph, Krötzsch, and Hitzler (2008) and Wache, Groot, and Stuckenschmidt (2005), focus on the syntactic form of the axioms of the original ontology and appropriately rewrite the axioms that do not comply with the syntax of the target language. They are rather effective in general but can produce unsound answers (Pan & Thomas, 2007). Semantic approaches focus on the model-based entailment from the original ontology, rather than on its syntactic structure. They aim at preserving these entailments as much as possible while transforming the ontology into the target language; in general they are sound, but they might be computationally more expensive (Console, Mora, Rosati, Santarelli, & Savo, 2014).

For our task of computing partial support families, sound ontology approximation techniques are relevant. We choose $DL\text{-}Lite_{core}$ as the target approximation language, as it lies in the intersection of $\mathcal{EL}$ and $DL\text{-}Lite_{\mathcal{A}}$, for which complete support families can be effectively identified (Eiter et al., 2014d). Our approach for approximating a TBox in $\mathcal{EL}$ to $DL\text{-}Lite_{core}$ exploits the *logical difference* between $\mathcal{EL}$ TBoxes considered by Konev et al. (2012). The idea behind it is to decide whether two ontologies give the same answers to queries over a given vocabulary (called signature) Σ, and compute a succinct representation of the difference if it is not empty. Typical queries include subsumption between concepts, instance queries and conjunctive queries. In our setting subsumption queries are of particular interest, as based on them nonground support families are constructed.

Our approach is as follows. Given a DL-atom $d = \mathrm{DL}[\lambda;\, Q](\vec{X})$ and an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, we eliminate from the TBox $\mathcal{T}_d$ axioms outside the $DL\text{-}Lite_{core}$ language, and obtain a simplified TBox $\mathcal{T}_d'$. We then compute a succinct representation of the logical difference between $\mathcal{T}_d$ and $\mathcal{T}_d'$ w.r.t. $\Sigma = \{\mathsf{sig}(\mathcal{A}_d \cup \mathcal{A}) \cup Q\}$; the axioms in the logical difference that fall into $DL\text{-}Lite_{core}$ are then added to $\mathcal{T}_d'$. By restricting Σ to predicates that can potentially appear in support sets we avoid redundant computations and we approximate only the relevant part of the TBox. This approach is particularly attractive, as the logical difference for $\mathcal{EL}$ was intensively studied, e.g. by Lutz, Walther, and Wolter (2007) and Konev et al. (2012), and polynomial algorithms are available for $\mathcal{EL}$-*terminologies*; we thus confine ourselves here to the latter.

To present our approximation approach formally, we first recall some notions introduced by Konev et al. (2012).

Definition 27 (Σ-concept difference) *The* Σ-concept difference *between $\mathcal{EL}$-terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ is the set $\mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ of all $\mathcal{EL}$-inclusions α over Σ such that $\mathcal{T}_1 \models \alpha$ and $\mathcal{T}_2 \not\models \alpha$.*

Example 28 *For the terminologies $\mathcal{T}_1 = \{B \sqsubseteq E, \quad E \sqsubseteq \exists r.\top, \quad C \sqsubseteq A \sqcap B\}$ and $\mathcal{T}_2 = \{C \sqsubseteq A, \quad D \sqsubseteq B, \quad D \equiv C\}$ it holds that $\mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2) = \emptyset$ for $\Sigma = \{A, B, C\}$, while $\mathsf{cDiff}_{\Sigma'}(\mathcal{T}_1, \mathcal{T}_2) = \{B \sqsubseteq \exists r.\top\}$ for $\Sigma' = \{B, r\}$.* $\square$

If two $\mathcal{EL}$-terminologies entail the same concept subsumptions over the signature Σ, i.e. it holds that $\mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2) = \mathsf{cDiff}_\Sigma(\mathcal{T}_2, \mathcal{T}_1) = \emptyset$, then they are called Σ-*concept inseparable*, which is denoted by $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$. E.g. in Example 28 we have that $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$ and $\mathcal{T}_1 \not\equiv^C_{\Sigma'} \mathcal{T}_2$.

The logical difference in terms of instance queries is defined as follows.

Definition 29 (Σ-instance difference) *The* Σ-instance difference *between terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ is the set $\mathsf{iDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ of pairs of the form $(\mathcal{A}, \alpha)$, where $\mathcal{A}$ is a Σ-ABox and α a Σ-instance assertion, such that $\mathcal{T}_1 \cup \mathcal{A} \models \alpha$ and $\mathcal{T}_2 \cup \mathcal{A} \not\models \alpha$. We say that $\mathcal{T}_1$ and $\mathcal{T}_2$ are* Σ-instance inseparable*, in symbols $\mathcal{T}_1 \equiv^i_\Sigma \mathcal{T}_2$ if $\mathsf{iDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2) = \mathsf{iDiff}_\Sigma(\mathcal{T}_2, \mathcal{T}_1) = \emptyset$.*

As easily seen, $\mathcal{T}_1 \equiv^i_\Sigma \mathcal{T}_2$ implies $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$. The converse is not obvious but also holds.

Theorem 30 (cf. Lutz & Wolter, 2010) *For any $\mathcal{EL}$-terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$ and signature Σ, $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$ iff $\mathcal{T}_1 \equiv^i_\Sigma \mathcal{T}_2$.*

4.2 Partial Support Family Construction

We now show that a DL-atom has the same set of support sets under Σ-concept inseparable terminologies. Prior to that, we establish the following lemma.

Lemma 31 *Let $d = \mathrm{DL}[\lambda;\, Q](\vec{t})$ be a DL-atom, let $\mathcal{O} = \langle \mathcal{T}_1, \mathcal{A} \rangle$ be an $\mathcal{EL}$ ontology, and let $\mathcal{T}_2$ be a TBox. If $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$, where $\Sigma = \mathit{sig}(\mathcal{A}) \cup \mathit{sig}(Q) \cup \{P \mid P \circ p \in \lambda\}$, then $\mathcal{T}_{1d} \equiv^C_{\Sigma'} \mathcal{T}_{2d}$, where $\Sigma' = \Sigma \cup \mathit{sig}(\mathcal{A}_d)$.*

Armed with this, we obtain the following result on equivalence of nonground support families.

Proposition 32 *Let $d = \mathrm{DL}[\lambda;\, Q](\vec{X})$ be a DL-atom and let $\mathcal{T}_1$, $\mathcal{T}_2$ be $\mathcal{EL}$-terminologies such that $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$ where $\Sigma = \mathit{sig}(\mathcal{A} \cup \mathcal{A}_d \cup Q) \cup \{P \mid P \circ p \in \lambda\}$. If $\mathbf{S}_1$ and $\mathbf{S}_2$ are complete nonground support families for d w.r.t. $\mathcal{O}_1 = \langle \mathcal{T}_1, \mathcal{A} \rangle$ and $\mathcal{O}_2 = \langle \mathcal{T}_2, \mathcal{A} \rangle$, respectively, then $\mathbf{S}_1$ and $\mathbf{S}_2$ are ground-identical.*

Given two $\mathcal{EL}$-terminologies $\mathcal{T}_1$ and $\mathcal{T}_2$, the inclusions $C \sqsubseteq A \in \mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ (resp. $A \sqsubseteq C \in \mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$) are following Konev et al. (2012) called *left* (resp. *right*) *witnesses* and denoted as $\mathsf{cWTn}^{rhs}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$ (resp. $\mathsf{cWTn}^{lhs}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$). It was shown that every inclusion $C \sqsubseteq D$ in the Σ-concept difference of $\mathcal{T}_1$ and $\mathcal{T}_2$ “contains” either a left or a right witness.

Theorem 33 (cf. Konev et al., 2012) *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be $\mathcal{EL}$-terminologies and Σ a signature. If $\phi \in \mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$, then either $C \sqsubseteq A$ or $A \sqsubseteq D$ is a member of $\mathsf{cDiff}_\Sigma(\mathcal{T}_1, \mathcal{T}_2)$, where $A \in \mathit{sig}(\phi)$ is a concept name and C and D are $\mathcal{EL}$-concepts occurring in ϕ.*

Algorithm 1: $PartSupFam$: compute partial support family

Input: DL-atom $d = \mathrm{DL}[\lambda;\, Q](\vec{X})$, ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$
Output: Partial nonground support family $\mathbf{S} \subseteq \mathbf{Supp}_{\mathcal{O}}(d)$ for d
(a) $\Sigma \leftarrow \{\mathsf{sig}(\mathcal{A} \cup \mathcal{A}_d) \cup Q\}$
(b) $\mathcal{T}_d \leftarrow \mathcal{T} \cup \{P_p \sqsubseteq P \mid P \uplus p \in \lambda\}$
(c) $\mathcal{T}_d' \leftarrow \mathcal{T}_d \backslash \{C \sqsubseteq D \mid C \notin \{A, \exists r.\top\}$ or $D \notin \{A, \exists r.\top\}\}$
(d) $lrw \leftarrow \mathsf{cWTn}_{\Sigma}^{rhs}(\mathcal{T}_d, \mathcal{T}_d') \cup \mathsf{cWTn}_{\Sigma}^{lhs}(\mathcal{T}_d, \mathcal{T}_d')$
(e) $\mathcal{T}_d'' \leftarrow \mathcal{T}_d' \cup \{C \sqsubseteq D \in lrw \mid C, D \in \{A, \exists r.\top\}\}$
(f) $\mathbf{S} \leftarrow \{ComplSupFam(d, \mathcal{T}_d'')\}$
return S

The logical difference between two $\mathcal{EL}$-terminologies in its compact representation consists only of inclusions with an atomic concept name on either the left or the right hand side. Some may have inclusions with atomic concepts on both sides or role restrictions of the form $\exists r.\top$, which fall into our target language of $DL\text{-}Lite_{core}$ DL, and can be therefore reintroduced.

We are now ready to describe the algorithm $PartSupFam$ (see Algorithm 1) to compute partial families of support sets. As input we are given a DL-atom $d = \mathrm{DL}[\lambda;\, Q](\vec{X})$ and an ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is an $\mathcal{EL}$-terminology. We first set the signature Σ in (a) to predicates relevant for support set computation for d. We then construct the TBox $\mathcal{T}_d$ in (b) and its simplified version $\mathcal{T}_d'$ in (c) by removing from $\mathcal{T}_d$ all axioms of the form $C \sqsubseteq D$, where C or D is a complex concept, i.e. all axioms that are not in the $DL\text{-}Lite_{core}$ fragment. In (d) we compute right-hand side and left-hand side witnesses between $\mathcal{T}_d$ and $\mathcal{T}_d'$ for Σ and store them in lrw. After that, in (e) we construct the TBox $\mathcal{T}_d''$ by extending $\mathcal{T}_d'$ with all axioms from lrw, having concepts of the form A or $\exists r$ on both sides of inclusions. Based on the support set construction method for $DL\text{-}Lite_{\mathcal{A}}$ of Eiter et al. (2014d), we then obtain a complete support family $\mathcal{S}$ for $\mathcal{T}_d''$ in (f), which is a partial support family for $\mathcal{T}$.

Proposition 34 *The family* $\mathbf{S}$ *computed by Algorithm 1 fulfills* $\mathbf{S} \subseteq \mathbf{Supp}_{\mathcal{O}}(d)$*, i.e.,* $\mathbf{S}$ *is a partial support family for a given DL-atom* d *w.r.t.* $\mathcal{T}$ *where* $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$.

If $lwr = \emptyset$ in (d) or $\mathsf{cDiff}_{\Sigma}(\mathcal{T}_d, \mathcal{T}_d'') = \emptyset$ in (e), then $\mathbf{S}$ is guaranteed to be complete by Proposition 32. While in general Algorithm 1 can be used for computing support families for DL-atoms accessing arbitrary TBoxes[4], practically efficient procedures for (d) are available only for acyclic $\mathcal{EL}$-terminologies (Konev et al., 2012).

5. Bounded Support Sets

In this section, we analyze the size and the number of support sets that a given DL-atom can have. With bounds on these quantities at hand, one can limit the search space of support sets. More precisely, we aim at support set families that are sufficient for evaluating the DL-atom. As support sets S' that are (properly) subsumed by another support set S (i.e., $S \subseteq_{\theta} S'$) can be dropped, we consider non-ground support families that subsume any other (in particular, any complete) support family. More formally,

4. For computing logical difference between arbitrary TBoxes recent results by Feng, Ludwig, and Walther (2015) might be exploited.

Definition 35 (θ-complete support family) *We say a nonground support family* $\mathbf{S}$ *for a DL-atom* d *is* θ-complete *w.r.t. an ontology* $\mathcal{O}$, *if* $\mathbf{S} \subseteq_\theta \mathbf{S}'$ *for* $\mathbf{S}' \in \mathbf{Supp}_{\mathcal{O}}(d)$.

Thus the question are bounds on the size of support sets in $\mathbf{S}$ and the cardinality of a smallest $\mathbf{S}$.

Throughout this section, we tacitly assume that TBoxes are acyclic, i.e. they do not entail inclusions of the form $\exists R.C \sqsubseteq C$.

5.1 Estimation of Support Set Size Bounds

We first consider an estimate on the maximal size of support sets in the smallest θ-complete support family by analyzing the syntactic properties of a given TBox. To start with, we recall from the work of Konev et al. (2012) that an atomic concept A is *primitive* in a terminology $\mathcal{T}$, if it occurs in no axiom of $\mathcal{T}$ on the left-hand side, and *pseudo-primitive*, if it is either primitive or occurs only on the left-hand side of axioms $A \sqsubseteq C$, where C is an arbitrary $\mathcal{EL}$ concept.

For an $\mathcal{EL}$-terminology $\mathcal{T}$ and every pseudo-primitive A such that $\mathcal{T} \models D \sqsubseteq A$, where $D = A_1 \sqcap \ldots \sqcap A_n \sqcap \exists r_1.C_1 \ldots \exists r_m.C_m$, some (atomic) conjunct A_i in D exists such that $\mathcal{T} \models A_i \sqsubseteq A$ (Konev et al., 2012, Lemma 15). From this we obtain:

Proposition 36 *Let* $d = \mathrm{DL}[\lambda; Q](\vec{t})$ *be a DL-atom, and let* $\mathcal{T}$ *be an* $\mathcal{EL}$*-terminology. If* Q *is pseudo-primitive in* $\mathcal{T}$*, then* $maxsup(d) = 1$.

Proposition 36 exploits a specific case, in which the support set size bound is 1. For providing more liberal syntactic conditions on $\mathcal{T}$ that ensure bounded size of support sets, we use *ontology hypergraphs* (Nortje et al., 2013; Ecke et al., 2013). The latter have been widely studied for extracting modules of ontologies (Nortje et al., 2013), determining concept difference between $\mathcal{EL}$ terminologies (Ecke et al., 2013), efficient reasoning in OWL 2 QL (Lembo, Santarelli, & Savo, 2013), and other important tasks.

First let us recall the notion of a *directed hypergraph*, which is a natural generalization of a directed graph, proposed by Ausiello, D'Atri, and Saccà (1983) in the context of databases to represent functional dependencies.

Definition 37 (directed hypergraph) *A* directed hypergraph *is a pair* $\mathcal{G} = (\mathcal{V}, \mathcal{E})$*, where* $\mathcal{V}$ *is a set of nodes of the graph and* $\mathcal{E}$ *is a set of directed* hyperedges *of the form* $e = (H, H')$*, where* $H, H' \subseteq \mathcal{V}$ *are nonempty sets called* hypernodes.

Given a hyperedge $e = (H, H')$, we call H the *tail* of e and H' the *head* of e, denoted by $tail(e)$ and $head(e)$, respectively. A hypernode is a *singleton*, if $|H| = 1$, and a *binary hypernode*, if $|H| = 2$; in abuse of notation, for a singleton $\{v\}$, we also simply write v. The notion of an ontology hypergraph for DL $\mathcal{EL}$ introduced by Ecke et al. (2013) is as follows.

Definition 38 (ontology hypergraph) *Let* $\mathcal{T}$ *be an* $\mathcal{EL}$ *TBox in normal form, and let* $\Sigma \subseteq \mathbf{C} \cup \mathbf{R}$*. The* ontology hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T}}$ *of* $\mathcal{T}$ *is a directed hypergraph* $\mathcal{G}^{\Sigma}_{\mathcal{T}} = (\mathcal{V}, \mathcal{E})$*, where*

$$
\begin{aligned}
\mathcal{V} = {} & \{x_A \mid A \in \mathbf{C} \cap (\Sigma \cap \mathit{sig}(\mathcal{T}))\} \cup \{x_r \mid r \in \mathbf{R} \cap (\Sigma \cap \mathit{sig}(\mathcal{T}))\} \cup \{x_\top\}, \text{ and} \\
\mathcal{E} = {} & \{(\{x_A\}, \{x_B\}) \mid A \sqsubseteq B \in \mathcal{T}\} \cup \\
& \{(\{x_A\}, \{x_r, x_Y\}) \mid A \sqsubseteq \exists r.Y \in \mathcal{T}, Y \in \mathbf{C} \cup \{\top\}\} \cup \\
& \{(\{x_r, x_Y\}, \{x_A\}) \mid \exists r.Y \sqsubseteq A \in \mathcal{T}, Y \in \mathbf{C} \cup \{\top\}\} \cup \\
& \{(\{x_{B_1}, x_{B_2}\}, \{x_A\}) \mid B_1 \sqcap B_2 \sqsubseteq A \in \mathcal{T}\}.
\end{aligned}
$$

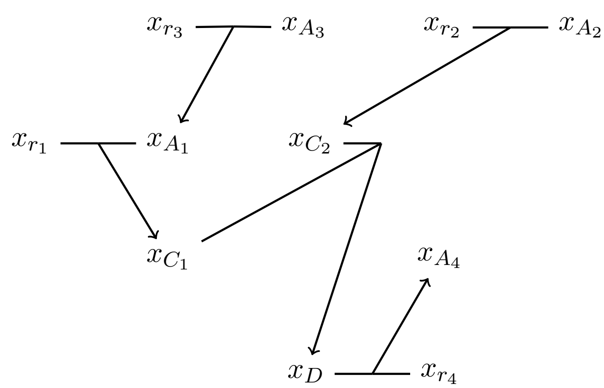


Figure 4: Hypergraph $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ from Example 39

Example 39 *Consider the following TBox in normal form:*

$$\mathcal{T} = \left\{ \begin{array}{ll} \textit{(1)}\ \exists r_1.A_1 \sqsubseteq C_1 & \textit{(4)}\ C_1 \sqcap C_2 \sqsubseteq D \\ \textit{(2)}\ \exists r_2.A_2 \sqsubseteq C_2 & \textit{(5)}\ A_3 \sqsubseteq A_2 \\ \textit{(3)}\ \exists r_3.A_3 \sqsubseteq A_1 & \textit{(6)}\ D \sqsubseteq \exists r_4.A_4 \end{array} \right\}.$$

The ontology hypergraph $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ for Σ=sig($\mathcal{T}$) is depicted in Figure 4. □

We now define the notions of *directed path* between two nodes and *incoming path* to a singleton node in an ontology hypergraph; both are natural generalizations of a path in a standard graph.

Definition 40 (directed path, incoming path) *Suppose that $\mathcal{T}$ is an $\mathcal{EL}$ TBox in a normal form, $\mathcal{G}_{\mathcal{T}}^{\Sigma} = (\mathcal{V}, \mathcal{E})$ is an ontology hypergraph, and $x, y \in \mathcal{V}$ are singleton nodes occurring in $\mathcal{G}_{\mathcal{T}}^{\Sigma}$. Then a* directed path *between x and y in $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ is a sequence $\pi = e_1, e_2, \ldots, e_n$ of (hyper) edges, such that:*

(i) $tail(e_1) \supseteq x$*;*

(ii) $head(e_n) \supseteq y$*;*

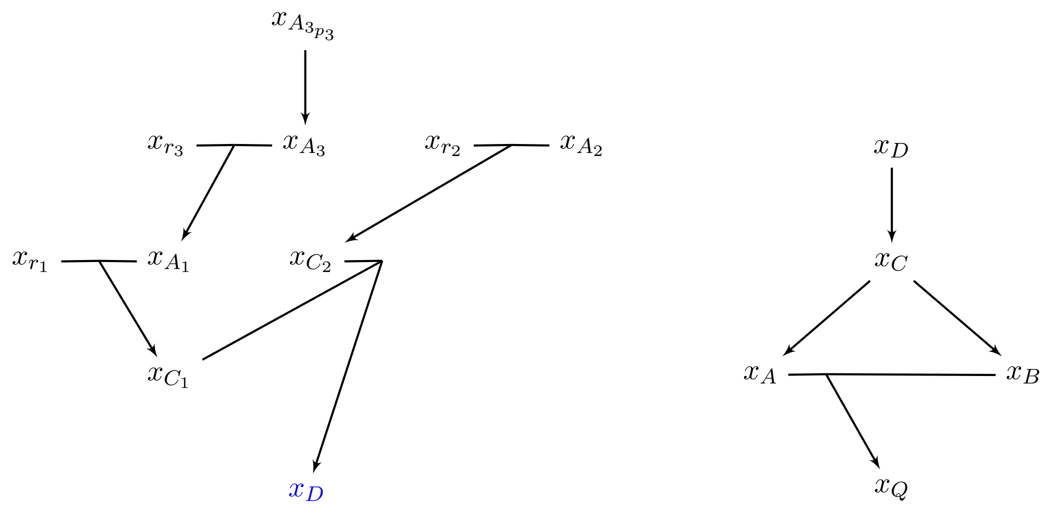
(iii) for every e_i, $i < n$, some successor $s(e_i) = e_j$ of e_i exists in $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ such that $j > i$, $head(e_i) \subseteq tail(e_j)$, and $s(e_i) = s(e_{i'})$ implies $head(e_i) \neq head(e_{i'})$ for $i \neq i'$.

An incoming path *to a singleton node $x \in \mathcal{V}$ in $\mathcal{G}_{\mathcal{T}}^{\Sigma} = (\mathcal{V}, \mathcal{E})$ is a directed path $\pi = e_1, \ldots, e_n$ from any node $y \in \mathcal{V}$ to x, such that $head(e_n) = x$. The set of all incoming paths to a node x in a hypergraph $\mathcal{G}$ is denoted by $Paths(x, \mathcal{G})$.*

Intuitively, hyperedges in an ontology hypergraph $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ model inclusion relations between (complex) concepts over Σ in $\mathcal{T}$. Consequently, an incoming path to a singleton node x_C in $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ models a chain of inclusions that logically follow from $\mathcal{T}$, such that C is the rightmost element of the chain.

Example 41 *Let us look at the ontology hypergraph $\mathcal{G}_{\mathcal{T}}^{\Sigma}$ in Figure 4. The sequence of edges*

$$\pi_1 = (\{x_{r_3}, x_{A_3}\}, x_{A_1}), (\{x_{r_1}, x_{A_1}\}, x_{C_1})$$



(a) $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ from Example 43 (b) $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ from Example 45

Figure 5: Examples of support hypergraphs

is an incoming path to x_{C_1} in $\mathcal{G}^{\Sigma}_{\mathcal{T}}$ that reflects the inclusions $\exists r_1.A_1 \sqsubseteq C_1$ and $\exists r_1.(\exists r_3.A_3) \sqsubseteq C_1$; the sequence

$$\pi_2 = (\{x_{r_3}, x_{A_3}\}, x_{A_1}), (\{x_{r_1}, x_{A_1}\}, x_{C_1}), (\{x_{r_2}, x_{A_2}\}, x_{C_2}), (\{x_{C_1}, x_{C_2}\}, x_D)$$

is an incoming path to the singleton x_D, from which the following set of inclusions can be extracted: (1) $C_1 \sqcap C_2 \sqsubseteq D$, (2) $\exists r_2.A_2 \sqcap C_1 \sqsubseteq D$, (3) $\exists r_2.A_2 \sqcap \exists r_1.A_1 \sqsubseteq D$, and (4) $\exists r_2.A_2 \sqcap \exists r_1.(\exists r_3.A_3) \sqsubseteq D$. $\Box$

We now introduce our notion of a support hypergraph for a DL-atom.

Definition 42 (support hypergraph) *A* support hypergraph *for a DL-atom d=DL$[\lambda;\, Q](\vec{t})$ over a normal ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ is a hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ constructed as follows:*

1. *build the ontology hypergraph $\mathcal{G}^{\Sigma}_{\mathcal{T}_d} = (\mathcal{V}, \mathcal{E})$, where $\Sigma = \mathsf{sig}(\mathcal{A} \cup \mathcal{A}_d) \cup \{Q\}$;*
2. *leave all nodes and edges in $Paths(x_Q, \mathcal{G}^{\Sigma}_{\mathcal{T}_d})$ and remove all other nodes and edges;*
3. *for $x_C \in \mathcal{G}^{\Sigma}_{\mathcal{T}_d}$ with $C \notin \Sigma$, if in $Paths(x_C, \mathcal{G}^{\Sigma}_{\mathcal{T}_d})$ a (hyper) node N exists such that $\{P \mid x_P \in N\} \subseteq \Sigma$ then leave x_C, otherwise remove it and all of its corresponding edges;*
4. *for $x_r \in \mathcal{G}^{\Sigma}_{\mathcal{T}_d}$, such that $r \notin \Sigma$, leave $e = (\{x_r, y\}, x_{C'})$ if $(x_C, \{x_r, y\})$ exists in $\mathcal{G}^{\Sigma}_{\mathcal{T}_d}$, where $y \in \{x_D, \top\}$, otherwise remove e.*

Let us illustrate the notion of a support hypergraph on the following example:

Example 43 *Let $\mathcal{T}$ from Example 39 be accessed by the DL-atom $d = $ DL$[A_3 \uplus p_3;\, D](\vec{X})$, and $\mathcal{T}_d = \mathcal{T} \cup \{A_{3_{p_3}} \sqsubseteq A_3\}$. The support hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ for d with $\Sigma = \mathsf{sig}(\mathcal{T}_d)$ is shown in Figure 5a. The node x_D colored in blue corresponds to the DL-query of d. The edge $(\{x_D\}, \{x_{r_4}, x_{A_4}\})$ is not in $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$, as it does not lie on the incoming path to x_D.* $\Box$

Before describing the approach of extracting support sets for a DL-atom from a hypergraph, we introduce the notion of *tree-acyclicity*. For alternative definitions we refer the reader to the works, e.g. by Ausiello, D'Atri, and Saccà (1986), Gallo, Longo, and Pallottino (1993) and Thakur and Tripathi (2009).

Definition 44 (tree-acyclicity) *A hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is called* tree-acyclic*, if (i) at most one directed path exists in $\mathcal{G}$ between any singleton nodes $x, y \in \mathcal{V}$, and (ii) $\mathcal{G}$ has no paths $\pi = e_1, \ldots, e_k$ such that $tail(e_1) \cap head(e_k) \neq \emptyset$.*

We refer to hypergraphs that are not tree-acyclic as *tree-cyclic*.

Example 45 *$\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ in Figure 5a is tree-acyclic, while $\mathcal{G}' = \mathcal{G}^{\Sigma'}_{supp(d),\mathcal{T}'}$ with $\mathcal{T}' = \mathcal{T} \cup \{B \sqsubseteq A_3,\ B \sqsubseteq A_2\}$ and $\Sigma' = \Sigma \cup \{B\}$ is not, and neither is $\mathcal{G}'' = \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}''}$, where $\mathcal{T}'' = \mathcal{T} \cup \{A_1 \sqsubseteq C_2\}$.*

The hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ for $d = \mathrm{DL}[;\, Q](X)$, $\mathcal{T} = \{D \sqsubseteq C; C \sqsubseteq A; C \sqcap B; A \sqcap B \sqsubseteq Q\}$ and $\Sigma = \mathit{sig}(\mathcal{T})$ given in Figure 5b is tree-cyclic, since it contains two paths between x_D and x_Q, namely $\pi_1 = x_D, x_C, x_A, \{x_A, x_B\}, x_Q$ and $\pi_2 = x_D, x_C, x_B, \{x_A, x_B\}, x_Q$. ☐

The support hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}} = (\mathcal{V}, \mathcal{E})$ for a DL-atom $d = \mathrm{DL}[\lambda;\, Q](X)$ contains all incoming paths to x_Q that start from nodes corresponding to predicates in $\mathcal{A} \cup \mathcal{A}_d$ by construction, i.e. it reflects all inclusions with Q on the right-hand side and predicates over $\mathcal{A} \cup \mathcal{A}_d$ on the left hand-side that are entailed from $\mathcal{T}_d$. Hence, by traversing edges of all incoming paths to x_Q, we can construct sufficiently many query rewritings of Q over the TBox $\mathcal{T}_d$ corresponding to nonground support sets that allow to subsume every nonground support family w.r.t. $\mathcal{O}$.

If a support hypergraph for a given DL-atom is tree-acyclic, then support sets can be conveniently constructed from it by annotating nodes with variables X_i, $i \in \mathbb{N}$ in a way as described below. We use subscripts for annotations, e.g. $x_C^{\langle X_i \rangle}$ means that the node x_C is annotated with the variable X_i, while $x_r^{\langle X_i, X_j \rangle}$ states that x_r is annotated with the ordered pair of variables X_i, X_j.

The approach proceeds as follows. We start from the node x_Q, which we annotate with X_0, i.e. $x_Q^{\langle X_0 \rangle}$; then we traverse the hypergraph backwards, going from a head of an edge to its tail. For every edge e that we encounter we annotate $tail(e)$ based on its form and on the annotation of $head(e)$, with variable names that occur in annotation of $head(e)$ and/or fresh variable names X_i, $i \in \mathbb{N}$, in the following way:

(1) if $|tail(e)| = 1$, then

  (1.1) if $head(e) = \{x_{C_1}^{\langle X_i \rangle}\}$, then $tail(e)$ is annotated with $\langle X_i \rangle$;

  (1.2) if $head(e) = \{x_{r_1}^{\langle X_{i_1}, X_{i_2} \rangle}, x_{C_1}^{\langle X_{i_3} \rangle}\}$, then $tail(e) = x_{C_2}$ is annotated with $\langle X_{i_1} \rangle$, i.e. we obtain $x_{C_2}^{\langle X_{i_1} \rangle}$;

(2) if $|tail(e)| = 2$ and $head(e) = \{x_C^{\langle X_i \rangle}\}$, then

  (2.1) if $tail(e) = \{x_{C_1}, x_{C_2}\}$, then both x_{C_1} and x_{C_2} are annotated with X_i, i.e. $\{x_{C_1}^{\langle X_i \rangle}, x_{C_2}^{\langle X_i \rangle}\}$;

  (2.2) if $tail(e) = \{x_{r_1}, x_{C_1}\}$, then we get $\{x_{r_1}^{\langle X_i, X_{i_1} \rangle}, x_{C_1}^{\langle X_{i_1} \rangle}\}$, From every annotated hypernode N, one can create a set of nonground atoms with predicate names extracted from labels of hypernodes and variable names from their annotations. The nonground support sets for $d = \mathrm{DL}[\lambda;\, Q](X_0)$ are then constructed from the incoming paths to x_Q.

We pick some incoming path π_1 to x_Q containing n edges, and start traversing it from the edge e_n with $head(e_n) = \{x_Q\}$. The first immediate support set is $S_1 = \{Q(X_0)\}$; the next one, S_2, is extracted from the annotated tail of e_n by taking nonground predicates of labels and variables. We then pick an edge e_k such that $head(e_k) \subseteq tail(e_n)$, and obtain further support sets by substituting nonground atoms that correspond to $head(e_k) \cap tail(e_n)$ in S_2 with the atoms extracted from $tail(e_k)$; this is repeated. One can in fact construct the incoming path backwards along with the support set extraction, until a maximal path is obtained.

Example 46 *Consider the maximal incoming path to x_D of $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ from Figure 5a:*

$$\pi = \underbrace{(x_{A_{3_{p_3}}}, x_{A_3})}_{e_1}, \underbrace{(\{x_{r_3}, x_{A_3}\}, x_{A_1})}_{e_2}, \underbrace{(\{x_{r_1}, x_{A_1}\}, x_{C_1})}_{e_3}, \underbrace{(\{x_{r_2}, x_{A_2}\}, x_{C_2})}_{e_4}, \underbrace{(\{x_{C_1}, x_{C_2}\}, x_D)}_{e_5}.$$

Traversing the path backwards, i.e. edges in the order e_5, e_4, e_3, e_2, e_1, we obtain: $\underbrace{(x^{\langle X_3\rangle}_{A_{3_{p_3}}}, x^{\langle X_3\rangle}_{A_3})}_{e_1}$, $\underbrace{(\{x^{\langle X_2,X_3\rangle}_{r_3}, x^{\langle X_3\rangle}_{A_3}\}, \{x^{\langle X_2\rangle}_{A_1}\})}_{e_2}, \underbrace{(\{x^{\langle X_0,X_2\rangle}_{r_1}, x^{\langle X_2\rangle}_{A_1}\}, x^{\langle X_0\rangle}_{C_1})}_{e_3}, \underbrace{(\{x^{\langle X_0,X_1\rangle}_{r_2}, x^{\langle X_1\rangle}_{A_2}\}, x^{\langle X_0\rangle}_{C_2})}_{e_4}, \underbrace{(\{x^{\langle X_0\rangle}_{C_1}, x^{\langle X_0\rangle}_{C_2}\}, x^{\langle X_0\rangle}_{D})}_{e_5}$.
The nonground support sets for d are extracted from the resulting annotated path as follows:

- $S_0 = \{D(X_0)\}$ *is immediately obtained from* $head(x_D)$*;*
- *the first incoming path to consider is $\pi_1 = e_5$, from which we get $S_1 = \{C_1(X_0), C_2(X_0)\}$;*
- *next is the path $\pi_2 = e_4, e_5$ as $head(e_4) \subseteq tail(e_5)$, yielding the support set $S_2 = \{C_1(X_0), r_2(X_0, X_1), A_2(X_0, X_1)\}$;*
- *then, from $\pi_3 = e_3, e_5$ we get $S_3 = \{C_1(X_0), r_1(X_0, X_2), A_1(X_2)\}$;*
- *$\pi_4 = e_3, e_4, e_5$ yields $S_4 = \{r_2(X_0, X_1), A_2(X_1), r_1(X_0, X_2), A_1(X_2)\}$;*
- *from $\pi_5 = e_2, e_3, e_5$, we extract $S_5 = \{r_1(X_0, X_2), r_3(X_2, X_3), A_3(X_3), C_2(X_0)\}$;*
- *$\pi_6 = e_2, e_3, e_4, e_5$ yields $S_6 = \{r_1(X_0, X_2), r_3(X_2, X_3), A_3(X_3), r_2(X_0, X_1), A_2(X_1)\}$;*
- *from $\pi_7 = e_1, e_2, e_3, e_5$, we extract $S_7 = \{r_1(X_0, X_2), r_3(X_2, X_3), A_{3_{p_3}}(X_3), C_2(X_0)\}$;*
- *finally, from $\pi_8 = e_1, e_2, e_3, e_4, e_5$ we get $S_8 = \{r_1(X_0, X_2), r_3(X_2, X_3), A_{3_{p_3}}(X_3), r_2(X_0, X_1), A_2(X_1)\}$.* $\square$

The following lemma formally asserts the correctness of the procedure.

Lemma 47 *Let $\mathbf{S}_{\mathcal{G}}$ be the support family constructed from a tree-acyclic hypergraph $\mathcal{G}{=}\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ for $d = \mathrm{DL}[\lambda;\, Q](\vec{X})$. Then $\mathbf{S}_{\mathcal{G}}$ is θ-complete for d w.r.t. $\mathcal{O}$, i.e., $\mathbf{S}_{\mathcal{G}} \subseteq_{\theta} \mathbf{S}$ for every $\mathbf{S} \in \mathbf{Supp}_{\mathcal{O}}(d)$.*

In particular, Lemma 47 holds for each complete $\mathbf{S}$ for d w.r.t. the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$. Thus we can determine sufficiently many nonground support sets for d by just looking at its support hypergraph. Note that the restriction to tree-acyclic TBoxes is crucial for correctness of the procedure from above, as it ensures that every node of a hypergraph is annotated only once.

Lemma 47 allows us to reason about the structure and size of support sets by analyzing only parameters of the support hypergraph. One such parameter, for instance, is the maximal number $n(\pi, \mathcal{G})$ of hyperedges with a singleton head node excluding $(\{x_r, \top\}, x_A)$, occurring on some incoming path π to x_Q of a hypergraph $\mathcal{G}$.

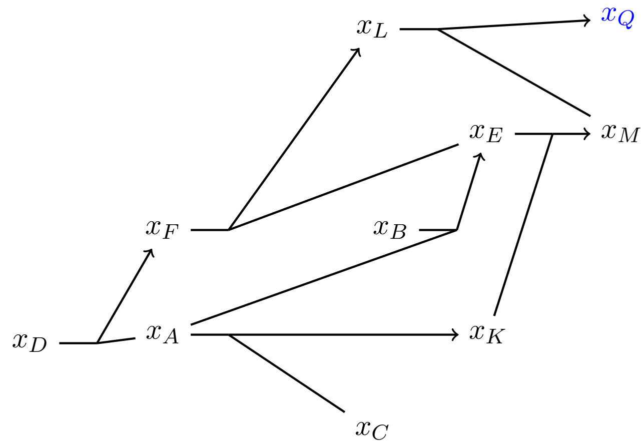


Figure 6: Support hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ from Example 49

Proposition 48 *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{EL}$ ontology with $\mathcal{T}$ in a normal form, and let $d = \mathrm{DL}[\lambda;\ Q](\vec{X})$ be a DL-atom with a tree-acyclic support hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$. Then*

$$maxsup(d) \leq max_{\pi \in \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}}(n(\pi, \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}})) + 1. \tag{6}$$

For tree-cyclic hypergraphs, the bound from above is not tight, which we illustrate next.

Example 49 *Consider the DL-atom $d(X) = \mathrm{DL}[;\ Q](X)$ accessing the TBox $\mathcal{T}_d$:*

$$\mathcal{T}_d = \left\{ \begin{array}{ll} \textit{(1) } A \sqcap D \sqsubseteq F & \textit{(4) } E \sqcap F \sqsubseteq L \\ \textit{(2) } A \sqcap C \sqsubseteq K & \textit{(5) } E \sqcap K \sqsubseteq M \\ \textit{(3) } A \sqcap B \sqsubseteq E & \textit{(6) } M \sqcap L \sqsubseteq Q \end{array} \right\}.$$

The support hypergraph for d is depicted in Figure 6, where $\Sigma = \mathsf{sig}(\mathcal{T}_d)$. There are six hyperedges with singleton head nodes, but the maximal support set size for $d(X)$ is 4, e.g. $S = \{A(X), B(X), D(X), K(X)\}$. $\square$

We next define out- and in-degrees of nodes in a hypergraph.

Definition 50 (hyper-outdegree and -indegree) *Given a directed hypergraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the* hyper-outdegree *denoted by $hd^+(x)$ (resp.,* hyper-indegree *$hd^-(x)$) of a singleton node $x \in \mathcal{V}$ is the number of hyperedges $e \in \mathcal{E}$ such that $tail(e) \supseteq x$ (resp., $head(e) \supseteq x$) and either $|tail(e)| = 2$ or $|head(e)| = 2$. Similarly, the* outdegree *$d^+(x)$ (resp.,* indegree *$d^-(x)$) of x is the number of edges $e \in \mathcal{E}$ such that $tail(e) = \{x\}$ (resp., $head(e) = \{x\}$) and $|head(e)| = |tail(e)| = 1$.*

Example 51 *All nodes $X \in \mathcal{V} \backslash \{x_{A_{3_p}}, x_D\}$ in the hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ of Figure 5a have hyper-outdegree 1, while for $x_{A_{p_3}}$ and x_D we have $hd^+(x_{A_{p_3}}) = hd^+(x_D) = 0$, moreover, $d^+(x_{A_{p_3}}) = 1$. For hyper-indegrees we have $hd^-(x_{A_3}) = hd^-(x_{A_1}) = hd^-(x_{C_1}) = hd^-(x_{C_2}) = 1$. In the graph $\mathcal{G}' = \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}} \cup (\{x_{C_2}, x_{A_2}\}, x_D)$, it holds that $hd^+(x_{C_2}) = hd^+(x_{A_2}) = hd^-(x_D) = 2$, moreover, $d^-(x_{A_3}) = 1$.* $\square$

Now let us define that

$$s_{max}(x, \mathcal{G}) = max_{\pi \in Paths(x,\mathcal{G})}(n(\pi, \mathcal{G}) - m(\pi, \mathcal{G}) + 1), \tag{7}$$

where $m(\pi, \mathcal{G}) = \sum_{x_A \in \pi}(hdc^+(x_A) - 1)$, and $hdc^+(x_A)$ is the number of hyperedges of form $(\{x_A, x_B\}, x_C)$ on π.

Example 52 *Consider $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ in Figure 5a, where $Paths(x_D, \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}})$ contains a single maximal path to x_D, viz. $\pi = (x_{A_{3_{p_3}}}, x_{A_3}), (\{x_{r_3}, x_{A_3}\}, x_{A_1}), (\{x_{r_2}, x_{A_2}\}, x_{C_2}), (\{x_{r_1}, x_{A_1}\}, x_{C_1})$, $(\{x_{C_1}, x_{C_2}\}, x_D)$. We have $n(\pi, \mathcal{G}) = 4$, as four hyperedges on π have a singleton head node, and $m(\pi, \mathcal{G}) = 0$, as all nodes have hyper-outdegree at most 1; hence $s_{max}(x_Q, \mathcal{G}) = 4 - 0 + 1 = 5$. The hypergraph in Figure 6 has a single maximal incoming path π to x_Q, and $n(\pi, \mathcal{G}) = 6$, $m(\pi, \mathcal{G}) = (hdc^+(x_A) - 1) + (hdc^+(x_E) - 1) = 3$; thus $s_{max}(x_Q, \mathcal{G}) = 6 - 3 + 1 = 4$.* ☐

We generalize the bound on the maximal support set size for d from Proposition 48 using the parameter $s_{max}(x_Q, \mathcal{G})$ for a node corresponding to the DL-query Q of a DL-atom d, and obtain the following result for hypergraphs that are possibly tree-cyclic:

Proposition 53 *Let $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ be an $\mathcal{EL}$ ontology with $\mathcal{T}$ in a normal form, and let $d = \mathrm{DL}[\lambda;\, Q](\vec{X})$ be a DL-atom with support hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$, such that Σ has no role predicates. Then $maxsup(d) \leq s_{max}(x_Q, \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}})$.*

Example 54 *For the tree-cyclic hypergraph in Figure 6 we have $s_{max}(x_Q, \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}) = 4$, and 4 is indeed the maximal support set size for $d = \mathrm{DL}[;\, Q](X)$. The hypergraph in Figure 5a has 3 hyperedges, and for every node $x \in \mathcal{V}$, $hd^+(x) \leq 1$. Thus, $s_{max}(x_Q, \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}) = 4$, which coincides with $maxsup(d)$, where $d = \mathrm{DL}[A_3 \uplus p_3;\, Q](X)$.* ☐

Note that in Proposition 53, we take in computing $m(\pi, \mathcal{G})$ only outgoing hyperedges of the form $(\{x_C, x_D\}, x_E)$ into account, where C, D, E are concepts, and moreover, no roles occur in Σ. Multiple outgoing hyperedges involving roles r with $r \in \Sigma$ do not influence the support set size.

Example 55 *Let a support hypergraph for $d = \mathrm{DL}[\lambda;\, Q](X)$ have the hyperedges $(\{x_r, x_C\}, x_D)$, $(\{x_C, x_s\}, x_M)$, $(\{x_D, x_M\}, x_Q)$ where $r \in \Sigma$, reflecting the axioms $\exists r.C \sqsubseteq D$, $\exists s.C \sqsubseteq M$ and $M \sqcap D \sqsubseteq Q$. A largest minimal support set for d is $S = \{r(X,Y), C(Y), s(X,Z), C(Z)\}$; its size is $n + 1$, where n is the number of hyperedges with a singleton head node, while $hd^+(x_C) = 2$.* ☐

5.2 Number of Support Sets

Orthogonal to the question considered in the previous section is under which conditions a given number n of support sets is sufficient to obtain a θ-complete support family. This problem is tightly related to counting minimal solutions for an abduction problem, which was analyzed by Hermann and Pichler (2010) for propositional theories under various restrictions. In particular, counting $\subseteq$-minimal explanations was shown to be $\# \cdot \mathsf{coNP}$-complete for general propositional theories and $\#P$-complete for Horn propositional theories; as $\mathcal{EL}$ subsumes propositional Horn logic, determining the size of a smallest θ-complete support family is at least $\#P$-hard and thus intractable.

Like for the size of support sets, the support hypergraph can be fruitfully exploited for estimating the maximal number of support sets for a given DL-atom. To provide such an estimate, we traverse the support hypergraph forward starting at the leaves and label every node x_P with the number of rewritings for P. To conveniently compute the labels, we introduce support weight functions.

Definition 56 (support weight function) *Let $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}} = (\mathcal{V},\mathcal{E})$ be a support hypergraph for a DL-atom d. A* support weight function $ws : \mathcal{V} \to \mathbb{N}$ *assigns to every node $x_A \in \mathcal{V}$ the number $ws(x_A)$ of rewritings of A over $\mathcal{T}$ w.r.t. Σ.*

For every node in a tree-acyclic support hypergraph, the value of ws can be conveniently computed in a recursive manner.

Proposition 57 *Let $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ be a tree-acyclic support hypergraph for a DL-atom d over a (normalized) ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$. Then ws is given as follows, where $\mathcal{V}_{\mathbf{C}} \subseteq \mathcal{V}$ is the set of nodes for concepts:*

$$ws(x) = \begin{cases} 1, & \text{if } hd^-(x)=0 \text{ and } d^-(x)=0 \text{ or } x \notin \mathcal{V}_{\mathbf{C}}, \\ 1 + \sum_{T\in T^-(x)} \prod_{x'\in T} ws(x') & \\ \quad + \sum_{T\in T^-(x), T\not\subseteq \mathcal{V}_{\mathbf{C}}} \sum_{(\{x'\},T)\in\mathcal{E}} ws(x'), & \text{otherwise.} \end{cases} \tag{8}$$

where $T^-(x) = \{T \mid (T,\{x\}) \in \mathcal{E}\}$.

We demonstrate the usage of Proposition 57 by the following examples.

Example 58 *To compute $ws(x)$ for the nodes of $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ in Figure 5a, we traverse the graph from leaves to the root, and for $x \in \{x_{r_1}, x_{A_2}, x_{C_2}, x_{r_2}, x_{A_{3p_3}}, x_{r_3}\}$ we obtain $ws(x) = 1$; furthermore, $ws(x_{A_3}) = ws(x_{C_2}) = 2$, $ws(x_{A_1}) = 3$, $ws(x_{C_1}) = 4$. Finally, $ws(x_D) = 1 + ws(x_{C_1}) * ws(x_{C_2}) = 1 + 4 * 2 = 9$, which is the number of rewritings for $D(X)$ (and hence support sets for $d(X) = \mathrm{DL}[A_3 \uplus p_3;\ D](X)$) identified in Example 46.*

Example 59 *Consider the TBox $\mathcal{T} = \{A \sqcap B \sqsubseteq Q; C \sqsubseteq A; D \sqsubseteq A; E \sqsubseteq A; F \sqsubseteq B; G \sqsubseteq B; H \sqsubseteq B; A \sqsubseteq L\}$ and a DL-atom $d = \mathrm{DL}[;\ Q](X)$, whose support hypergraph for $\Sigma = \mathsf{sig}(\mathcal{T})$ is in Figure 7. We have that $ws(x_Q) = 1 + ws(x_B) * ws(x_A) = 1 + 4 * 4 = 17$, and indeed there are 17 rewritings for $Q(X)$, namely $S_1 = \{A(X), B(X)\}$, $S_2 = \{C(X), B(X)\}$, $S_3 = \{D(X), B(X)\}$, $S_4 = \{E(X), B(X)\}$, $S_5 = \{A(X), F(X)\}$, $S_6 = \{A(X), G(X)\}$, $S_7 = \{A(X), H(X)\}$, $S_8 = \{C(X), F(X)\}$, $S_9 = \{C(X), G(X)\}$, $S_{10} = \{C(X), H(X)\}$, $S_{11} = \{D(X), F(X)\}$, $S_{12} = \{D(X), G(X)\}$, $S_{13} = \{D(X), H(X)\}$, $S_{14} = \{E(X), F(X)\}$, $S_{15} = \{E(X), G(X)\}$, $S_{16} = \{E(X), H(X)\}$, and $S_{17} = \{Q(X)\}$.*

As an immediate corollary of Proposition 57, we obtain

Corollary 60 *Let $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}} = (\mathcal{V},\mathcal{E})$ be a tree-acyclic support hypergraph for the DL-atom $d = \mathrm{DL}[\lambda;\ Q](\vec{X})$ over an $\mathcal{EL}$ ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A}\rangle$. If each edge $e \in \mathcal{E}$ satisfies $|tail(e)| = |head(e)| = 1$, then*

$$ws(v) = \sum_{e\in\mathcal{E} \mid head(e)=v} ws(tail(e)) + 1. \tag{9}$$

Thus for the query node x_Q, we get $ws(x_Q) = |\mathcal{E}| + 1$. In fact, Proposition 57 leads to this simple bound on the size of $\subseteq_\theta$-minimal complete support families in more general cases.

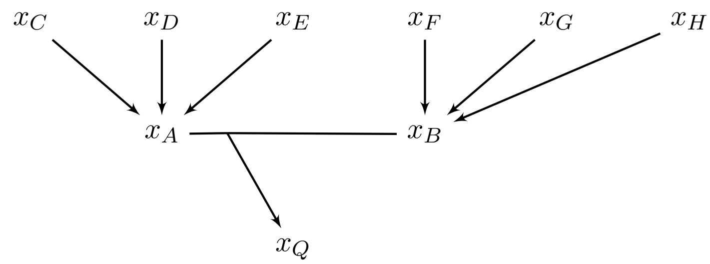


Figure 7: Hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$

Proposition 61 *Let $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}} = (\mathcal{V}, \mathcal{E})$ be a tree-acyclic support hypergraph for the DL-atom $d = \mathrm{DL}[\lambda;\, Q](\vec{X})$ over an $\mathcal{EL}$ ontology, such that for every edge $e = (\{x, y\}, z) \in \mathcal{E}$ and edges $e_1, e_2 \in \mathcal{E}$ such that $head(e_i) \subseteq \{x, y\}$, $i \in \{1, 2\}$, it holds that $head(e_1) = head(e_2)$. Then $|\mathbf{S}_{\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}}| = |\mathcal{E}| + 1$.*

Example 62 *The hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ in Figure 5a has a single maximal path of length 5, and its hyperedges satisfy the condition of Corollary 61. As d has 6 support sets, $|\mathbf{S}| = |\mathcal{E}| + 1$ holds.* $\square$

If the condition of Proposition 61 on e and e_1, e_2 is violated, then the maximal size of a $\subseteq_\theta$-minimal complete support family can not be assessed that easily. For instance, the support hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ from Figure 7 contains 7 edges, but d has 17 support sets. It can be shown that if k nodes in $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ violate the condition, then $\mathbf{S}_{\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}}$ contains at most $|\mathcal{E}|^{k+1} + 1$ support sets; for the considered example, this yields a bound of $7^2 + 1 = 50$, which is far from tight.

We note that Proposition 57 can not be applied for tree-cyclic support hypergraphs.

Example 63 *Consider a tree-cyclic support hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ for $d = \mathrm{DL}[;\, Q](X)$, $\mathcal{T} = \{D \sqsubseteq C;\ C \sqsubseteq A;\ C \sqsubseteq B;\ A \sqcap B \sqsubseteq Q\}$ and $\Sigma = \mathsf{sig}(\mathcal{T})$, which is shown in Figure 5b. Using Proposition 57 we get $ws(x_D) = 1$, $ws(x_C) = 2$, $ws(x_A) = 3$, $ws(x_B) = 3$, $ws(x_Q) = 3{*}3{+}1 = 10$. However, $Q(X)$ has only 4 rewritings: (1) $S_1 = \{Q(X)\}$, (2) $S_2 = \{A(X), B(X)\}$, (3) $S_3 = \{C(X)\}$, and (4) $S_4 = \{D(X)\}$.*

Intuitively, for tree-cyclic hypergraphs the support weight function ws may also account for non-minimal rewritings $\{B(X), C(X)\}$, $\{A(X), C(X)\}$, $\{A(X), D(X)\}$, $\{B(X), D(X)\}$, and some rewritings can be counted multiple times. Thus in general, $ws(x)$ provides only an upper bound for the number of rewriting. Likewise, the bound in Proposition 61 is not tight even for simple tree-cyclic support hypergraphs; e.g., the one for the DL-atom $d = \mathrm{DL}[;\, Q](X)$ w.r.t. the TBox $A \sqsubseteq B_i$, $B_i \sqsubseteq Q$, $1 \leq i \leq n$, contains $2 * n$ edges, but d has only $n + 2$ support sets. $\square$

6. Repair Computation Based on Partial Support Families

In this section, we present our algorithm *SoundRAnsSet* for computing deletion repair answer sets. As shown by Stepanova (2015), deciding whether a given DL-program $\Pi = \langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$ over an $\mathcal{EL}$ ontology has some deletion repair answer set is Σ^P_2-complete in the general case, where the membership part is established by guessing a candidate repair ABox $\mathcal{A}' \subseteq \mathcal{A}$ along with a candidate answer set I for $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', \mathcal{P} \rangle$, and the suitability of the guess is checked using an NP oracle. Clearly this is not efficient, as there are $|2^n|$ candidate repair ABoxes for $n = |\mathcal{A}|$, even if finding an answer set I of Π' would be cheap.

We restrict the search space of repairs in our approach as in the work of Eiter et al. (2014d) by exploiting support families for DL-atoms; however, in contrast to the results by Eiter et al. (2014d), the support families are not required to be complete. If the families are complete (which may be known or asserted in their construction), then $SoundRAnsSet$ is guaranteed to be complete; otherwise, it may miss repair answer set, but an easy extension ensures completeness.

Our algorithm for repair answer set computation, shown as Algorithm 2, proceeds as follows.

- We start at (a) by computing a family $\mathbf{S}$ of nonground support sets for each DL-atom.
- Next in (b) the so-called *replacement program* $\hat{\Pi}$ is constructed.

  The replacement program is obtained by a simple rewriting of $gr(\Pi)$, where each DL-atom d is replaced by an ordinary atom e_d (called *replacement atom*), and a disjunctive choice rule $e_d \vee ne_d \leftarrow$ is added that informally guesses the truth value of d, where e_d (respectively ne_d) stands for the value true (respectively false). Each repair answer set of Π augmented with the proper choice of e_d resp. ne_d is an answer set of $\hat{\Pi}$ (Eiter et al., 2013, Proposition 13); thus the search can be confined to answer sets $\hat{I}$ of $\hat{\Pi}$, which can be found using a standard ASP solver.
- In (c) the answer sets $\hat{I}$ of $\hat{\Pi}$ are computed one by one.
- For $\hat{I}$, we determine in (d) the sets D_p (resp. D_n) of DL-atoms that are guessed true (resp. false) in it and then use the function $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$ which instantiates $\mathbf{S}$ for the DL-atoms in $D_p \cup D_n$ to relevant ground support sets, i.e., those compatible with $\hat{I}$.
- In (e) we loop through all minimal hitting sets $H \subseteq \mathcal{A}$ of the support sets for DL-atoms in D_n that only consist of ABox assertions, and in (f) we construct for each H the set D'_p of atoms from D_p that have at least one support set which is disjoint from H (thus removing H from $\mathcal{A}$ does not affect the values of atoms in D'_p).
- Then in (g) we evaluate in a *postcheck* the atoms in D_n and $D_p \backslash D'_p$ over $\mathcal{T} \cup \mathcal{A} \backslash H$ w.r.t. $\hat{I}$.

  A Boolean flag rep stores the evaluation result of a function $eval_n$(resp. $eval_p$). More specifically, given D_n (resp. D_p), $\hat{I}$ and $\mathcal{T} \cup \mathcal{A} \backslash H$, the function $eval_n$ (resp. $eval_p$) returns $true$, if all atoms in D_n (resp. D_p) evaluate to false (resp. true).
- If rep is $true$ and the foundedness check $\mathit{flpFND}(\hat{I}, \mathcal{T} \cup \mathcal{A} \setminus H, \mathcal{P})$ succeeds, then in (h) the restriction $\hat{I}|_{\Pi}$ of $\hat{I}$ to the original language of Π is output as repair answer set.

We remark that in many cases, the foundedness check might be trivial or superfluous (Eiter, Fink, Krennwallner, Redl, & Schüller, 2014a), e.g., when there are no loops through DL-atoms; if we consider weak answer sets (Eiter et al., 2013), it can be entirely skipped.

Example 64 *Let* Π *be the DL-program from Example 1 with equivalence* ($\equiv$) *in the axioms (2) and (3) weakened to* $\sqsupseteq$*, and with further assertions* $Project(p1)$ *and* $BlacklistedStaffRequest(r1)$ *added to the ABox* $\mathcal{A}$*. Moreover, assume that* $d_1(r1) = \mathrm{DL}[Project \uplus projfile;\ Staffrequest](r1)$*,* $d_2(r1){=}\mathrm{DL}[Staff \uplus chief;\ BlacklistedStaffRequest](r1)$*,* $d_3(r1,p1){=}\mathrm{DL}[;\ hasTarget](r1,p1)$

Algorithm 2: $SoundRAnsSet$: compute deletion repair answer sets

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

Output: a set of repair answer sets of Π

(a) compute a set $\mathbf{S}$ of nonground support families for the DL-atoms in Π

(b) construct the replacement program $\hat{\Pi}$

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

(d) $\quad D_p\leftarrow\{d\mid e_d\in\hat{I}\}$; $D_n\leftarrow\{d\mid ne_d\in\hat{I}\}$; $\mathbf{S}^{\hat{I}}_{gr}\leftarrow Gr(\mathbf{S},\hat{I},\mathcal{A})$;

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

(f) $\quad\quad D'_p\leftarrow\{d\in D_p\mid\exists S\in\mathbf{S}^{\hat{I}}_{gr}(d)\text{ s.t. }S\cap H=\emptyset\}$

(g) $\quad\quad rep\leftarrow eval_n(D_n,\hat{I},\mathcal{T}\cup\mathcal{A}\backslash H)\wedge eval_p(D_p\backslash D'_p,\hat{I},\mathcal{T}\cup\mathcal{A}\backslash H)$

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

$\quad$ **end**

end

and $d_4(r1,john)=\mathrm{DL}[;\,hasSubject](r1,john)$*. Then in (b) the following replacement program* $\hat{\Pi}$ *is constructed:*

$$\hat{\Pi}=\left\{\begin{array}{l}(1)\ e_{d_1}(r1)\vee ne_{d_1}(r1);\ \ (2)\ e_{d_2}(r1)\vee ne_{d_2}(r1);\ \ (3)\ e_{d_3}(r1,p1)\vee ne_{d_3}(r1,p1);\\ (4)\ e_{d_4}(r1,john)\vee ne_{d_4}(r1,john);\ \ (5)\ projfile(p1);\ \ (6)\ hasowner(p1,john);\\ (7)\ chief(john)\leftarrow hasowner(p1,john),projfile(p1);\\ (8)\ grant(r1)\leftarrow e_{d_1}(r1),not\ deny(r1);\\ (9)\ deny(r1)\leftarrow e_{d_2}(r1);\\ (10)\ \bot\leftarrow hasowner(p1,john),not\ grant(r1),e_{d_3}(r1,p1),e_{d_4}(r1,john).\end{array}\right\}.$$

Suppose that $\hat{I}=\{e_{d_1},ne_{d_2},e_{d_3},e_{d_4},hasowner(p1,john),projfile(p1),chief(john)\}$ *is returned at (c) and that the following partial support families are obtained in (d):*

– $\mathbf{S}^{\hat{I}}_{gr}(d_1)=\{S_1,S_2\}$*, where* $S_1=\{hasAction(r1,read),hasSubject(r1,john),\ Action(read),$ $Staff(john),hasTarget(r1,p1),Project_{projfile}(p1)\}$ *and* $S_2=\{StaffRequest(r1)\}$*;*

– $\mathbf{S}^{\hat{I}}_{gr}(d_2)=\{S'_1,S'_2\}$*, where* $S'_1=\{StaffRequest(r1),hasSubject(r1,john),Blacklisted(john)\}$ *and* $S'_2=\{BlacklistedStaffRequest(r1)\}$*.*

– $\mathbf{S}^{\hat{I}}_{gr}(d_3)=\{S''_1\}$*, where* $S''_1=\{hasTarget(r1,p1)\}$*;*

– $\mathbf{S}^{\hat{I}}_{gr}(d_4)=\{S'''_1\}$*, where* $S'''_1=\{hasSubject(r1,john)\}$*.*

At (e) we get a hitting set $H=\{StaffRequest(r1),BlacklistedStaffRequest(r1)\}$*, which is disjoint with* S_1*,* S''_1 *and* S'''_1*. Thus in (f) we obtain* $D'_p=\{d_1,d_3,d_4\}$ *and then in (g) we check whether* d_2 *is false under* $\mathcal{A}\backslash H$*. As this is not true,* $rep=false$ *and we pick a different hitting set* H'*, e.g.* $\{Blacklisted(john),\ BlacklistedStaffRequest(r1)\}$*. Proceeding with* H'*, we get to (g), and as* $eval_n(d_2,\hat{I},\mathcal{T}\cup\mathcal{A}\cap H)=true$ *and the flp-check succeeds at (f), the interpretation* $\hat{I}|_\Pi$ *is output.* $\square$

The following results state that our algorithm works properly.

Theorem 65 *Algorithm* $SoundRAnsSet$ *is sound, i.e., given a program* $\Pi = \langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$*, every output* I *is a deletion repair answer set of* Π.

If we know in addition that the support families are complete, then the postchecks at (g) are redundant. If $D'_p = D_p$, then we set $rep = true$, otherwise $rep = false$.

Theorem 66 *Suppose that for the input program* $\Pi = \langle \mathcal{T} \cup \mathcal{A}, \mathcal{P} \rangle$ *of Algorithm* $SoundRAnsSet$*, it holds that for each DL-atom in* Π *the support family in* $\mathbf{S}$ *computed in Step (a) of* $SoundRAnsSet$ *is* θ*-complete. Then Algorithm* $SoundRAnsSet$ *is complete, i.e., it outputs every deletion repair answer set of* Π.

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

6.1 Optimizations and Extensions

Research in repairing databases (see the work by Bertossi, 2011, for overview) suggests several techniques, which are of potential interest for DL-programs, and could be exploited for optimizing and extending our repair approach. *Localization* of repairs proposed by Eiter, Fink, Greco, and Lembo (2008) is one such technique, where cautiously a part of the data that is affected by inconsistency is identified and the search of repairs is narrowed down to this part. Using localization, in our setting the ontology ABox can be split into a safe set of facts, which will not be touched by any repair, and a set of facts that are (probably) affected. After the affected part is repaired, the result is then combined with the safe ABox part to get the final solution. To find a suitable ABox split, meta knowledge about the ontology (e.g. modules, additional domain information) can be used.

Another common approach for tackling an inconsistency problem, which proved to be effective for databases, is *decomposition* (Eiter et al., 2008). Here, the available knowledge is decomposed into parts, such that the reasons for inconsistency are identified in each part separately, and then the respective repairs are conveniently merged. While for databases decomposition is natural, it is in general unclear how an inconsistent DL-program can be effectively decomposed. One way to approach this problem is by determining DL-atoms whose replacement atoms are guessed true (resp. false) in all answer sets of $\hat{\Pi}$. Given a set of such DL-atoms, one can aim at first searching for a repair under which every such DL-atom has the desired value, and then extend the solution to get the final result. Modules of DL-programs (as identified by the DLVHEX solver) can also be exploited for program decomposition.

As not all repairs are equally useful for a certain setting, various filterings on repairs can be applied to get the most plausible candidates. Here, qualitative and domain-specific aspects of repairs are of crucial importance for their practicability. These can be formulated in terms of additional local constraints that express for instance that facts involving certain predicates or constants must be preserved (resp. should be checked for removal). Furthermore, the number of facts/predicates/constants allowed for deletion can be bounded. These filterings are incorporated in our repair approach. Yet there are several further extensions possible like conditional predicate dependence. For example, a user might be willing to express the condition that $StaffRequest(r)$ can only be eliminated if $hasAction(r, read)$ holds in the data part, or $Blacklisted$ staff members can not be removed, if they own files, for modifying which a separate $StaffRequest$ has been issued by a non-blacklisted staff member.

(r_1^*) $Sup_d(\vec{X}) \leftarrow S_d^{\mathcal{P}}(\vec{Y})$

(r_2^*) $Sup_d(\vec{X}) \leftarrow S_d^{\mathcal{A},\mathcal{P}}(\vec{Y})$

(r_3^*) $S_d^{\mathcal{P}}(\vec{Y}) \leftarrow rb(S_d^{p}(\vec{Y}))$

(r_4^*) $S_d^{\mathcal{A},\mathcal{P}}(\vec{Y}) \leftarrow rb(S_d^{\mathcal{A},\mathcal{P}}(\vec{Y})), nd(S_d^{\mathcal{A},\mathcal{P}}(\vec{Y}))$

(r_5^*) $\perp \leftarrow ne_d(\vec{X}), S_d^{\mathcal{P}}(\vec{Y})$

(r_6^*) $\bar{P}_{1d}(\vec{Y}) \vee \ldots \vee \bar{P}_{nd}(\vec{Y}) \leftarrow ne_d(\vec{X}), S_d^{\mathcal{A},\mathcal{P}}(\vec{Y})$

(r_7^*) $eval_d(\vec{X}) \leftarrow e_d(\vec{X}), not\ C_d, not\ Sup_d(\vec{X})$

(r_8^*) $eval_d(\vec{X}) \leftarrow ne_d(\vec{X}), not\ C_d$

(r_9^*) $\perp \leftarrow e_d(\vec{X}), C_d, not\ Sup_d(\vec{X})$

Figure 8: Rules $\mathcal{R}_d$ for declarative implementation

6.2 Implementation

We have implemented our repair approach in C++ in a system prototype (dlliteplugin of the DLVHEX system, 2015).

As discussed, the support sets for $\mathcal{EL}$ ontologies are of a rich structure, and thus for their computation, TBox classification as in the work of Eiter et al. (2014d) is insufficient. Indeed, we need to identify not only inclusions between atomic concepts, but also all inclusions of the form $C \sqsubseteq B$, where C is an arbitrarily complex concept and B is atomic. For constructing support sets we thus exploit the REQUIEM tool (Pérez-Urbina et al., 2010), which rewrites the target query over the TBox using datalog rewriting techniques. By limiting the number (resp. size) of the rewritings, partial support families can be computed.

In principle some support sets may be subsumed by smaller support sets (e.g., $\{R(c,d),A(c)\}$ by $\{A(c)\}$). These support sets are redundant and thus we eliminate them in our implementation.

After the support families are constructed, we use a declarative approach for determining repair answer sets, in which the minimal hitting set computation is accomplished by rules. To this end, for each DL-atom $d(\vec{X})$ the three fresh predicates (i) $Sup_d(\vec{X})$, (ii) $S_d^{\mathcal{P}}(\vec{Y})$, and (iii) $S_d^{\mathcal{A},\mathcal{P}}(\vec{Y})$ are introduced, where $\vec{Y} = \vec{X}\vec{X}'$, which intuitively say that $d(\vec{X})$ has (i) some support set, (ii) some support set involving only rule predicates, and (iii) some support set involving ABox predicates (and possibly rule predicates), called mixed support set. Furthermore, for every DL-atom $d(X)$, the rules $\mathcal{R}_d$ in Figure 8 are added to the replacement program $\hat{\Pi}$.

In these rules, the atom C_d informally says that the support family for $d(\vec{X})$ is known to be complete. Information about completeness of support families for certain DL-atoms can be added to the declarative program in the form of facts C_d. The rules (r_1^*)-(r_4^*) reflect information about support sets of $d(\vec{X})$ under a potential repair; $rb(S)$ stands for a rule body representing a support set S, i.e. $rb(S) = A_1, \ldots A_k$ if $S = \{A_1, \ldots, A_k\}$; $nd(S) = not\ \bar{p}_{P_{1d}}(\vec{Y}), \ldots, not\ \bar{p}_{P_{nd}}(\vec{Y})$, where $\{p_{P_{1d}}(\vec{Y}), \ldots, p_{P_{nd}}(\vec{Y})\}$, encodes the ontology part of S and $\bar{p}_{P_{id}}(\vec{Y})$ states that the assertion $P_{id}(\vec{Y})$ is marked for deletion. The constraint (r_5^*) forbids that $d(\vec{X})$, if guessed false has a matching support set that consists only of input assertions; (r_6^*) means that if instead $d(\vec{X})$ has a matching mixed support set, then some assertion from its ontology part must be eliminated. The rule (r_7^*) says that if $d(\vec{X})$ is guessed true, completeness of its support family is unknown and no matching support set is available, then an evaluation postcheck is necessary ($eval_d(\vec{X})$); rule (r_8^*) is similar for $d(\vec{X})$ guessed false. The rule (r_9^*) states that a DL-atom guessed true must have some support set, if its support family is known to be complete.

The set of facts $facts(\mathcal{A}) = \{p_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A}\}$, encoding the ABox assertions and $\mathcal{COMP} \subseteq \{C_d \mid \mathcal{S}_d$ is a complete support family for $d\}$ are added to the program $\hat{\Pi}$, and then its answer sets

$$
\hat{\Pi} \cup \mathcal{R} = \left\{ \begin{array}{l}
(1)\ \mathit{projfile}(p1); \quad (2)\ \mathit{hasowner}(p1, \mathit{john}); \quad (3)\ \mathit{issued}(\mathit{john}, r1); \\
(4)\ \mathit{chief}(\mathit{john}) \leftarrow \mathit{hasowner}(p1, \mathit{john}), \mathit{projfile}(p1); \\
(5)\ \mathit{deny}(r1) \leftarrow e_d(r1); \\
(6)\ \bot \leftarrow \mathit{hasowner}(p1, \mathit{john}), \mathit{issued}(\mathit{john}, r1), \mathit{deny}(r1); \\
(7)\ e_d(r1) \vee ne_d(r1); \\
(8)\ \mathit{sup}_d(X) \leftarrow p_{\mathit{BlacklistedStaffRequest}}(X), \mathit{not}\ \ \bar{p}_{\mathit{BlacklistedStaffRequest}}(X); \\
(9)\ \bar{p}_{\mathit{BlacklistedStaffRequest}}(X) \leftarrow ne_d(X), p_{\mathit{BlacklistedStaffRequest}}(X); \\
(10)\ \mathit{sup}_d(X) \leftarrow p_{\mathit{StaffRequest}}(X), \mathit{not}\ \ \bar{p}_{\mathit{StaffRequest}}(X), p_{\mathit{hasSubject}}(X, Y), \\
\qquad\qquad\qquad \mathit{not}\ \ \bar{p}_{\mathit{hasSubject}}(X, Y), p_{\mathit{Blacklisted}}(Y), \mathit{not}\ \ \bar{p}_{\mathit{Blacklisted}}(Y); \\
(11)\ \bar{p}_{\mathit{StaffRequest}}(X) \vee \bar{p}_{\mathit{hasSubject}}(X, Y) \vee \bar{p}_{\mathit{Blacklisted}}(Y) \leftarrow ne_d(X), p_{\mathit{Blacklisted}}(Y), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad p_{\mathit{StaffRequest}}(X), \\
\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad p_{\mathit{hasSubject}}(X, Y); \\
(12)\ \mathit{eval}_d(X) \leftarrow e_d(X), \mathit{not}\ \ C_d, \mathit{not}\ \ \mathit{sup}_d(X); \\
(13)\ \mathit{eval}_d(X) \leftarrow ne_d(X), \mathit{not}\ \ C_d; \\
(14)\ \bot \leftarrow e_d(X), C_d, \mathit{not}\ \ \mathit{sup}_d(X).
\end{array} \right\}
$$

Figure 9: Program $\hat{\Pi} \cup \mathcal{R}$ from Example 68

are computed. For each such answer set $\hat{I}$, we proceed with an evaluation postcheck for all atoms $d(\vec{c})$ for which the fact $\mathit{eval}_d(\vec{c})$ is in the answer set. If all evaluation postchecks succeed, then we extract the repair answer set $I = \hat{I}|_{\Pi}$ of the original program Π from $\hat{I}$. This way one identifies weak repair answer sets; for *flp*-repair answer sets, an additional minimality check is needed. In many cases, however, the *flp* and weak answer sets coincide (cf. Eiter et al., 2014a); in particular, this holds for the example and benchmark programs that we consider.

We now formally show that the described approach indeed correctly computes weak repair answer sets.

Proposition 67 *Let $\Pi = \langle \mathcal{O}, \mathcal{P} \rangle$ be a ground DL-program, where $\mathcal{O}$ is an $\mathcal{EL}$ ontology, let for each DL-atom d of Π be $\mathbf{S}_d \in \mathbf{Supp}_{\mathcal{O}}(d)$, and let $\mathcal{R}_d$ be the set of rules (r_1^*)-(r_9^*) for d. Define*

$$\Pi_1 = \hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP},$$

where $\mathcal{R} = \bigcup_d \mathcal{R}_d$, $facts(\mathcal{A}) = \{p_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A}\}$ and $\mathcal{COMP} \subseteq \{C_d \mid \mathbf{S}_d$ is θ-complete for d w.r.t. $\mathcal{O}\}$. Suppose $\hat{I} \in AS(\Pi_1)$ is such that the evaluation postcheck succeeds for every DL-atom d with $C_d \notin \mathcal{COMP}$. Then $\hat{I}|_{\Pi} \in RAS_{\mathit{weak}}(\Pi)$. Moreover, if $C_d \in \mathcal{COMP}$ for every DL-atom d, then $RAS_{\mathit{weak}}(\Pi) = \{\hat{I}|_{\Pi} \mid \hat{I} \in AS(\Pi_1)\}$.

Let us demonstrate the usage of the declarative implementation through an example.

Example 68 *Consider in Figure 9 the replacement program $\hat{\Pi}$ and the rules $\mathcal{R}$ of $\Pi = \langle \mathcal{P}, \mathcal{O} \rangle$, where $\mathcal{O}$ is as in Example 1, and $\mathcal{P}$ is as follows:*

$$
\mathcal{P} = \left\{ \begin{array}{l}
(1)\ \mathit{projfile}(p1); \quad (2)\ \mathit{hasowner}(p1, \mathit{john}); \quad (3)\ \mathit{issued}(\mathit{john}, r1); \\
(4)\ \mathit{chief}(\mathit{john}) \leftarrow \mathit{hasowner}(p1, \mathit{john}), \mathit{projfile}(p1); \\
(5)\ \mathit{deny}(r1) \leftarrow \mathrm{DL}[\mathit{Staff} \uplus \mathit{chief};\ \mathit{BlacklistedStaffRequest}](r1); \\
(6)\ \bot \leftarrow \mathit{hasowner}(p1, \mathit{john}), \mathit{issued}(\mathit{john}, r1), \mathit{deny}(r1).
\end{array} \right\} .
$$

Assume that for $d(X) = \mathrm{DL}[\mathit{Staff} \uplus \mathit{chief}; \mathit{BlacklistedStaffRequest}](X)$ *we are given an incomplete support family* $\mathbf{S}_d = \{S_1, S_2\}$*, where* $S_1 = \{\mathit{BlackListedStaffRequest}(X)\}$ *and* $S_2 = \{\mathit{StaffRequest}(X), \mathit{hasSubject}(X, Y), \mathit{Blacklisted}(Y)\}$*. Then the interpretation* $\hat{I} \supset \{ne_d(r1), \bar{p}_{\mathit{StaffRequest}}(r1), \bar{p}_{\mathit{Blacklisted}}(john), eval_d\}$ *is among the answer sets of* $\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A})$*. As* $eval_d \in \hat{I}$*, a post-check is needed for* $d(r1)$*; this test succeeds, and thus* $\hat{I}|_{\Pi}$ *is a repair answer set.* $\square$

7. Evaluation

The repair answer set computation approach is implemented within the DLVHEX system; the details can be found in the work of Stepanova (2015), and the software is freely online available (dlliteplugin, 2015). Our approach was evaluated on a multi-core Linux server running DLVHEX 2.4.0 under the HTCondor load distribution system (HTCondor, 2012), which is a specialized workload management system for compute-intensive tasks, using two cores (AMD 6176 SE CPUs) and 8GB RAM.

To the best of our knowledge, no similar system for repairing inconsistent DL-programs exists. The list of systems for evaluating DL-programs includes the DREW system (DReW, 2012; Xiao, 2014) and the dlplugin of the DLVHEX system (dlplugin, 2007). The DREW system exploits datalog rewritings for evaluating DL-programs over $\mathcal{EL}$ ontologies; however, it can not handle inconsistencies, which are the focus of our work. Thus DREW per se could not be used as a baseline for experiments. To facilitate a comparison, we have thus extended DREW with a naive repair technique, where a guess of a repair ABox is followed by a check of its suitability. However, this immediate implementation turned out to be infeasible even on small instances, as in general the search space for repairs is ways too large for its full exploitation; guided search is needed to ensure scalability. The dlplugin of the DLVHEX system invokes RACERPRO reasoner (RacerPro, 2007) as a back-end for evaluating calls to the ontology. However, for lightweight ontologies even in the standard evaluation mode without any repair extensions, it scales worse than the dlliteplugin (Eiter et al., 2014b); thus we focus on the latter in our experiments.

7.1 Evaluation Workflow

The general workflow of the experimental evaluation was as follows. In the first step, we constructed benchmarks by building rules and constraints on top of existing ontologies such that for some data parts the constructed programs become inconsistent. The instances were generated using shell scripts (DL-program benchmark generation scripts, 2015) with the size of the conflicting data part as a parameter. The benchmarks were then run using the HTCondor system, and the times were extracted from the log files of the runs. In each run, we measured the time for computing the first repair answer set, including support set computation, with a timeout of 300 seconds.

For each benchmark, we present our experimental results in tables. The first column p specifies the size of the instance (varied according to certain parameters specific for each benchmark), and in parentheses the number of generated instances. E.g., the value $10(20)$ in the first column states that a set of 20 instances of size 10 were tested. The other columns represent particular repair configurations, grouped into three sets.

The first set refers to the settings where θ-complete support families were exploited, while the second and the third refer to the settings in which the size, respectively the number of computed support sets was restricted. For the θ-complete setting, we in addition limit the number of facts (lim_f), predicates (lim_p) and constants (lim_c) involved in facts that can be removed; e.g., $lim\_p = 2$ states that the set of removed facts can involve at most two predicates. The parameter del_p stores predicates that can be deleted; e.g., $del\_p = StaffRequest$ means that repairs can be obtained by removing only facts over $StaffRequest$.

In the restricted configurations, the column $size = n$ (resp. $num = n$) states that in the computed partial support families the size (resp. number) of support sets is at most n; if $n = \infty$, then in fact all support sets were computed, but the system is not aware of the θ-completeness. We exploit partial θ-completeness for the number and size restriction cases, i.e. if no more support sets for an atom are computed and the number/size limits were not yet reached, then the support family for the considered atom is θ-complete.

In an entry $t(m)[n]$, t is the total average running time (including support set generation and timeouts), m is the number of timeouts and n is the number of found repair answer sets.

7.2 Benchmarks

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

(1) The policy benchmark is a variant of Example 1, in which the rule (14) of $\mathcal{P}$ is changed to $deny(X) \leftarrow \mathrm{DL}[Staff \uplus chief; UnauthorizedStaffRequest](X)$, and two further axioms, namely $UnauthorizedStaffRequest \equiv StaffRequest \sqcap \exists hasSubject.Unauthorized$ and $Blacklisted \sqsubseteq Unauthorized$ are added to $\mathcal{T}$.

(2) The OpenStreetMap benchmark contains a set of rules over the ontology for enhanced personalized route planning with semantic information (MyITS ontology, 2012) extended by an ABox containing data from the OpenStreetMap project (OSM, 2012).

(3) The LUBM benchmark comprises rules on top of the well-known LUBM ontology (LUBM, 2005) in $\mathcal{EL}$.

We now describe the benchmark results in details. All experimental data are online available (Experimental data, 2015).

7.2.1 ACCESS POLICY CONTROL

We considered ABoxes $\mathcal{A}_n$ with n staff members, for $n \in \{10, 250, 500\}$. Each data set has 5 projects and 3 possible actions; furthermore 20% of the staff members are unauthorized and 40% are blacklisted. For generating instances, we used the probability $p/100$ (with p from column 1) that a fact $hasowner(p_i, s_i)$ is added to the rules part $\mathcal{P}$ for each s_i, p_i, such that $Staff(s_i),\ Project(p_i) \in \mathcal{A}$ (i.e., instances vary only on facts $hasowner(p_i, s_i)$ in $\mathcal{P}$.) as a parameter. Here, p ranges from 20, 30, etc. to 90 for $\mathcal{A}_{10}$ and from 5, 10 etc. to 40 for $\mathcal{A}_{250}$ and $\mathcal{A}_{500}$. The total average running times for these settings are shown in Tables 2–4, where SR stands for $StaffRequest$. The experiments were performed for the ABoxes of the chosen size (i.e., $\mathcal{A}_{10}, \mathcal{A}_{250}, \mathcal{A}_{500}$) to demonstrate how our approach works on small, medium and large data.

As regards $\mathcal{A}_{10}$, limiting in the θ-complete setting the number of predicates for removal slightly increases the running times. Restricting repairs to removing facts only over $StaffRequest$ does not slow down the repair computation compared to the unrestricted case, as many of the actual repairs indeed satisfy this condition. The results for bounded number and size of support sets are

p	θ-complete support families			Incomplete support families			$num = \infty$
	$no\ restr.$	$lim\_p = 2$	$del\_p = SR$	$size = 3$	$size = 5$	$num = 3$	
20 (20)	1.92 (0)[20]	2.70 (0)[20]	1.91 (0)[20]	38.51 (0)[20]	33.86 (0)[20]	1.93 (0)[20]	1.92 (0)[20]
30 (20)	1.94 (0)[20]	2.72 (0)[20]	1.94 (0)[20]	86.35 (1)[19]	80.52 (1)[19]	1.95 (0)[20]	1.93 (0)[20]
40 (20)	1.93 (0)[20]	2.71 (0)[20]	1.93 (0)[20]	98.69 (1)[19]	96.45 (1)[19]	1.94 (0)[20]	1.93 (0)[20]
50 (20)	1.92 (0)[20]	2.70 (0)[20]	1.92 (0)[20]	100.46 (2)[18]	98.06 (2)[18]	1.93 (0)[20]	1.91 (0)[20]
60 (20)	1.94 (0)[20]	2.72 (0)[20]	1.95 (0)[20]	182.16 (3)[17]	186.20 (3)[17]	1.96 (0)[20]	1.94 (0)[20]
70 (20)	1.95 (0)[20]	2.73 (0)[20]	1.95 (0)[20]	153.66 (2)[18]	152.66 (2)[18]	1.96 (0)[20]	1.94 (0)[20]
80 (20)	1.94 (0)[20]	2.72 (0)[20]	1.95 (0)[20]	227.81 (6)[14]	223.24 (6)[14]	1.96 (0)[20]	1.95 (0)[20]
90 (19)	1.96 (0)[19]	2.74 (0)[19]	1.96 (0)[19]	267.52 (11)[8]	267.89 (12)[8]	1.96 (0)[19]	1.95 (0)[19]

Table 2: Policy benchmark, $\mathcal{A}_{10}$

p	θ-complete support families			Incomplete support families		$num = \infty$
	$no\ restr.$	$lim\_p = 2$	$del\_p = SR$	$size = 6$	$num = 3$	
5(20)	6.06(0)[20]	8.28 (0)[20]	6.05 (0)[20]	6.06 (0)[20]	6.07 (0)[20]	6.05 (0)[20]
10(20)	6.68(0)[20]	8.90 (0)[20]	6.68 (0)[20]	6.67 (0)[20]	6.69 (0)[20]	6.67 (0)[20]
15(20)	8.37(0)[20]	10.56 (0)[20]	8.35 (0)[20]	8.33 (0)[20]	8.34 (0)[20]	8.34 (0)[20]
20(20)	9.39(0)[20]	11.61 (0)[20]	9.40 (0)[20]	9.40 (0)[20]	9.43 (0)[20]	9.41 (0)[20]
25(20)	11.41(0)[20]	13.62 (0)[20]	11.41 (0)[20]	11.46 (0)[20]	11.40 (0)[20]	11.40 (0)[20]
30(20)	14.04(0)[20]	16.24 (0)[20]	14.09 (0)[20]	14.10 (0)[20]	14.05 (0)[20]	14.04 (0)[20]
35(20)	15.17(0)[20]	17.32 (0)[20]	15.19 (0)[20]	15.12 (0)[20]	15.16 (0)[20]	15.17 (0)[20]
40(20)	17.49(0)[20]	19.64 (0)[20]	17.47 (0)[20]	17.46 (0)[20]	17.45 (0)[20]	17.43 (0)[20]

Table 3: Policy benchmark, $\mathcal{A}_{250}$

p	θ-complete support families			Incomplete support families		$num = \infty$
	$no\ restr.$	$lim\_p = 2$	$del\_p = SR$	$size = 6$	$num = 3$	
5 (20)	14.99 (0)[20]	18.71 (0)[20]	14.98 (0)[20]	15.00 (0)[20]	14.97 (0)[20]	14.97 (0)[20]
10 (20)	23.57 (0)[20]	27.14 (0)[20]	23.52 (0)[20]	23.50 (0)[20]	23.51 (0)[20]	23.43 (0)[20]
15 (20)	35.07 (0)[20]	38.85 (0)[20]	35.09 (0)[20]	35.02 (0)[20]	35.12 (0)[20]	35.13 (0)[20]
20 (20)	73.43 (2)[18]	53.27 (0)[20]	73.29 (2)[18]	73.50 (2)[18]	73.32 (2)[18]	85.33 (3)[17]
25 (20)	152.29 (8)[12]	64.91 (0)[20]	152.33 (8)[12]	164.34 (9)[11]	152.25 (8)[12]	164.32 (9)[11]
30 (20)	288.06 (19)[1]	97.32 (1)[19]	288.08 (19)[1]	276.11 (18)[2]	288.05 (19)[1]	300.00 (20)[0]
35 (20)	300.00 (20)[0]	153.03 (5)[15]	300.00 (20)[0]	300.00 (20)[0]	300.00 (20)[0]	300.00 (20)[0]
40 (20)	300.00 (20)[0]	206.96 (10)[10]	300.00 (20)[0]	300.00 (20)[0]	300.00 (20)[0]	300.00 (20)[0]

Table 4: Policy benchmark, $\mathcal{A}_{500}$

almost constant, except when the size is limited to 5 or smaller (just size 3 and size 5 are shown). Here support sets exceed the bound and post-evaluation checks often fail, which visibly impacts the running times. While the support sets are large, there are just few of them; this can be seen from the insignificant difference between the times for $num = 3$ and $num = \infty$.

For the significantly larger ABox $\mathcal{A}_{250}$, we get that for each value of p the considered settings perform almost identical except that $lim\_p = 2$ is a bit slower. Moreover, the running times increase gracefully with the value of p. While bounding the support set size to 5 produces only timeouts (thus the column is omitted), computing support sets of size 6 is always sufficient to identify repairs.

For the largest setting $\mathcal{A}_{500}$, in the θ-complete case finding an arbitrary repair is faster than under the restriction $lim\_p = 2$, but only up to $p = 15$. From $p = 20$ the results for $lim\_p = 2$

p	θ-complete support families			Incomplete support families				$num = \infty$
	$no\ restr.$	$lim\_f = 5$	$lim\_c = 10$	$size = 1$	$size = 3$	$num = 1$	$num = 3$	
10 (20)	13.01 (0)[20]	16.50 (0)[20]	16.46 (0)[20]	16.39 (0)[11]	13.03 (0)[20]	13.23 (0)[20]	13.06 (0)[20]	12.99 (0)[20]
20 (20)	13.04 (0)[20]	16.49 (0)[20]	16.48 (0)[20]	20.98 (0)[5]	13.04 (0)[20]	13.35 (0)[20]	13.01 (0)[20]	13.02 (0)[20]
30 (20)	13.05 (0)[20]	16.54 (0)[20]	16.49 (0)[20]	24.56 (0)[0]	13.06 (0)[20]	13.51 (0)[20]	13.02 (0)[20]	13.05 (0)[20]
40 (20)	13.10 (0)[20]	16.58 (0)[20]	16.47 (0)[20]	59.26 (0)[1]	13.07 (0)[20]	13.55 (0)[20]	13.09 (0)[20]	13.05 (0)[20]
50 (20)	13.04 (0)[20]	16.60 (0)[20]	16.51 (0)[20]	123.80 (0)[0]	13.10 (0)[20]	13.56 (0)[20]	13.04 (0)[20]	13.06 (0)[20]
60 (20)	13.08 (0)[20]	16.61 (0)[20]	16.55 (0)[20]	106.63 (1)[0]	13.06 (0)[20]	13.60 (0)[20]	13.08 (0)[20]	13.08 (0)[20]
70 (20)	13.11 (0)[20]	16.68 (0)[20]	16.58 (0)[20]	139.08 (2)[0]	13.07 (0)[20]	13.61 (0)[20]	13.07 (0)[20]	13.13 (0)[20]
80 (20)	13.07 (0)[20]	16.70 (0)[20]	16.53 (0)[20]	211.33 (5)[0]	13.06 (0)[20]	13.61 (0)[20]	13.06 (0)[20]	13.08 (0)[20]
90 (20)	13.12 (0)[20]	16.81 (0)[20]	16.59 (0)[20]	260.36 (11)[0]	13.10 (0)[20]	13.67 (0)[20]	13.10 (0)[20]	13.08 (0)[20]

Table 5: Open Street Map benchmark results

outperform the unrestricted setting, as the posed limitation restricts the search space of repairs effectively. Removing only facts over $StaffRequest$ is no longer always sufficient, which is witnessed by the decreased number of identified repairs for $del\_p = StaffRequest$ compared to $lim\_p = 2$. Again the time increases rather gracefully with p as long as repair answer sets are found.

7.2.2 OPEN STREET MAP

For the second benchmark, we added rules on top of the ontology developed in the MyITS project. The fixed ontology contains 4601 axioms, where 406 axioms are in the TBox and 4195 are in the ABox. The fragment $\mathcal{T}'$ of $\mathcal{T}$ relevant for our scenario and the rules $\mathcal{P}$ are shown in Figure 10. Intuitively, $\mathcal{T}'$ states that building features located inside private areas are not publicly accessible and a covered bus stop is a bus stop with a roof. The rules $\mathcal{P}$ check that public stations do not lack public access, using CWA on private areas.

We used the method introduced by Eiter, Schneider, Šimkus, and Xiao (2014) to extract data from the OpenStreetMap repository (OSM, 2012). We constructed an ABox $\mathcal{A}$ by extracting the sets of all bus stops (285) and leisure areas (682) of the Irish city Cork, as well as $isLocatedInside$ relations between them (9) (i.e., bus stops located in leisure areas). As the data has been gathered by many volunteers, chances of inaccuracies may be high (e.g. imprecise GPS data). Since the data about roofed bus stops and private areas was yet unavailable, we randomly made 80% of the bus stops roofed and 60% of leisure areas private. Finally, we added for each bs_i such that $isLocatedInside(bs_i, la_j) \in \mathcal{A}$ the fact $busstop(bs_i)$ to $\mathcal{P}$ with probability $p/100$. Some instances are inconsistent since in our data set there are roofed bus stops located inside private areas.

The results are shown in Table 5. For the θ-complete setting arbitrary repairs are computed about 3.5 seconds faster than the repairs with bounded changes. For the restricted configuration the times do not vary much except for $size = 1$, where a significant time increase is observed, and repairs are found only for smaller instances. Like in the previous benchmark computing a small number of support sets is often sufficient, but the configuration $num = 1$ is as expected slightly slower than $num = 3$ (computing support sets is here cheap, while postchecks take some time).

$$\mathcal{T}' = \left\{ \begin{array}{l} (1)\ \mathit{BuildingFeature} \sqcap \exists \mathit{isLocatedInside}.\mathit{Private} \sqsubseteq \mathit{NoPublicAccess} \\ (2)\ \mathit{BusStop} \sqcap \mathit{Roofed} \sqsubseteq \mathit{CoveredBusStop} \end{array} \right\}$$

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

Figure 10: DL-program over OpenStreetMap ontology

p	θ-complete support families				Incomplete support families		$num = \infty$
	$\mathit{no\ restr.}$	$\mathit{lim\_f} = 5$	$\mathit{lim\_p} = 2$	$\mathit{lim\_c} = 20$	$\mathit{size} = 1$	$\mathit{size} = 3$	
5 (20)	37.14 (0)[20]	47.77 (0)[20]	43.74 (0)[20]	43.88 (0)[20]	42.57 (0)[20]	36.52 (0)[20]	36.26 (0)[20]
15 (20)	35.74 (0)[20]	34.93 (0)[11]	42.74 (0)[20]	41.51 (0)[19]	42.02 (0)[20]	35.96 (0)[20]	35.49 (0)[20]
25 (20)	35.71 (0)[20]	26.94 (0)[5]	42.80 (0)[20]	41.71 (0)[19]	41.91 (0)[20]	35.80 (0)[20]	35.49 (0)[20]
35 (20)	36.07 (0)[20]	20.53 (0)[0]	43.04 (0)[20]	26.91 (0)[7]	42.22 (0)[20]	36.00 (0)[20]	35.65 (0)[20]
45 (20)	35.98 (0)[20]	20.50 (0)[0]	43.11 (0)[20]	19.54 (0)[1]	41.94 (0)[20]	36.40 (0)[20]	35.66 (0)[20]
55 (20)	35.92 (0)[20]	20.51 (0)[0]	43.11 (0)[20]	18.47 (0)[0]	42.31 (0)[20]	35.98 (0)[20]	35.60 (0)[20]
65 (20)	36.13 (0)[20]	20.43 (0)[0]	43.44 (0)[20]	18.33 (0)[0]	41.81 (0)[20]	36.02 (0)[20]	35.92 (0)[20]
75 (20)	36.07 (0)[20]	20.63 (0)[0]	43.45 (0)[20]	18.28 (0)[0]	42.09 (0)[20]	36.21 (0)[20]	35.85 (0)[20]
85 (20)	36.11 (0)[20]	20.30 (0)[0]	43.35 (0)[20]	18.04 (0)[0]	42.22 (0)[20]	36.15 (0)[20]	35.83 (0)[20]
95 (20)	36.38 (0)[20]	20.55 (0)[0]	43.24 (0)[20]	18.20 (0)[0]	42.52 (0)[20]	36.17 (0)[20]	35.62 (0)[20]

Table 6: LUBM benchmark results

7.2.3 LUBM

We have also tested our approach on DL-programs $\Pi = \langle \mathcal{P}, \mathcal{O} \rangle$ built over an $\mathcal{EL}$ version of the LUBM ontology, whose TBox was extended with the following axioms:

$$\begin{array}{l} (1)\ \mathit{GraduateStudent} \sqcap \exists \mathit{assists}.\mathit{Lecturer} \sqsubseteq \mathit{TA} \\ (2)\ \mathit{GraduateStudent} \sqcap \exists \mathit{teaches}.\mathit{UndergraduateStudent} \sqsubseteq \mathit{TA} \end{array}$$

The rules of Π are as follows:

$$\mathcal{P} = \left\{ \begin{array}{l} (3)\ \mathit{stud}(X) \leftarrow \mathit{not}\,\mathrm{DL}[;\ \mathit{Employee}](X), \mathrm{DL}[;\ \mathit{TA}](X); \\ (4)\ \bot \leftarrow \mathrm{DL}[\mathit{Student}\ \uplus\ \mathit{stud};\ \mathit{TAof}](X, Y), \mathit{takesexam}(X, Y) \end{array} \right\};$$

here (3) states that unless a teaching assistant (TA) is known to be an employee, he/she is a student, while (4) forbids teaching assistants to take exams in the courses they teach.

The ABox contains information about one university with more then 600 students, 29 teaching assistants, constructed by a dedicated ABox generator (LUBM data generator, 2013). For pairs of constants t, c, such that $\mathit{teachingAssistantOf}(t, c)$ is in $\mathcal{A}$, the facts $\mathit{takesexam}(t, c)$ are randomly added to the rules part with probability $p/100$, thus the contradicting part in the DL-program is growing with respect to p.

The results for this benchmark are provided in Table 6. Bounding in the θ-complete setting the number of removed facts to 5 slows down the computation, if repairs satisfying the condition exist. For instances with $p \geq 35$ (i.e., inconsistency is more entrenched), more than 5 facts must be dropped to obtain a repair; moreover, they often involve more than 20 constants according to column 5. The absence of repairs for $lim\_f = 5$ and $lim\_c = 20$ is found faster than a repair in the unrestricted mode.

Limiting the support set size to 1 allows one to find repairs for all instances with a delay of less than 10 seconds compared to the θ-complete setting. However, there are many support sets for this benchmark, and thus bounding their number is less effective.

7.3 General Results Discussion

One can observe that for θ-complete settings and settings where post-evaluation checks are fast, the running times vary only slightly with growing p. This is due to our declarative implementation, in which computing repairs is reduced to finding answer sets of the program $\Pi_1 = \hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP}$ followed by possible evaluation postchecks. In our benchmarks the difference between instances of size p_i and p_{i+1} is the data part of the logic program, which is small compared to the part $facts(\mathcal{A})$ of Π_1 that is constant for all p. Thus as long as postchecks are not needed, the times required for repairing Π do not differ much even though the programs become "more inconsistent."

As expected, using θ-complete support families works well in practice. Naturally, it takes more time to compute restricted repairs rather than arbitrary repairs; however, when the imposed restrictions are too strong such that no repair can satisfy them, the solver may recognize this faster.

As reported by Hansen et al. (2014), $\mathcal{EL}$-TBoxes that originate from real-world applications admit FO-rewritings (of reasonable size) in almost all cases. This provides some evidence that real-world $\mathcal{EL}$-TBoxes hardly contain involving constraints on the conceptual level, and that hence either the size or number of support sets for DL-atoms often turn out to be limited. The novel algorithms for deletion repair answer set computation demonstrated their applicability for DL-programs over some real world data (Open Street Map benchmark results in Table 5).

While most of the other benchmarks that we have run are synthetic, they still vary w.r.t. TBox and ABox sizes. The capability of our algorithms for handling such diverse DL-programs confirms the potential of our approach.

8. Related Work

Inconsistencies in DL-programs were studied in several works (Pührer et al., 2010; Fink, 2012; Eiter et al., 2013, 2014d). Pührer et al. proposed an inconsistency tolerant semantics. Keeping the ontology untouched, the DL-atoms that introduce inconsistency as well as rules involving them are deactivated. The repair problem, outlined as an open issue by Pührer et al., was formalized by Eiter et al. (2013), where the notions of repair and repair answer sets together with a naive algorithm for their computation were proposed. The latter was then optimized by Eiter et al. (2014d, 2015) for $DL\text{-}Lite_{\mathcal{A}}$ by effectively exploiting complete support families for DL-atoms. Our approach is more general, and it differs from the one of Eiter et al. (2014d, 2015) in that it uses partial (not necessarily complete) support families and can be applied to ontologies in any DL, though with a possible impact on complexity.

In other hybrid formalisms, inconsistency management has concentrated on inconsistency tolerance rather than on repair. For instance, Huang et al. (2013) presented a four-valued paraconsistent semantics based on Belnap's logic (Belnap, 1977) for hybrid MKNF knowledge bases (Motik & Rosati, 2010), which are the most prominent tightly coupled combination of rules and ontologies. Inspired by the paracoherent stable semantics of Sakama and Inoue (1995), the work of Huang et al. (2013) was extended by Huang, Hao, and Luo (2014) to handle also incoherent MKNF KBs, i.e. programs in which inconsistency arises as a result of the dependency of an atom on its default negation in analogy to the work of Fink (2012). Another direction of inconsistency handling for hybrid MKNF KBs is using the three-valued (well-founded) semantics of Knorr, Alferes, and Hitzler (2011), which avoids incoherence for disjunction-free stratified programs. Most recently, this has been extended by Kaminski et al. (2015) with additional truth values to evaluate contradictory pieces of knowledge. These works aim at inconsistency tolerance rather than repair, and are geared in spirit to query answering that is inherent to well-founded semantics; as such, it is limited to normal logic programs, while DL-programs allow for disjunctive rule heads.

In the context of Description Logics, repairing ontologies has been studied intensively, foremost to handle inconsistency. Our DL-program repair is related to ABox cleaning (Masotti, Rosati, & Ruzzi, 2011; Rosati, Ruzzi, Graziosi, & Masotti, 2012). However, the latter differs in various respects: it aims at restoring consistency of an inconsistent ontology by deleting $\subseteq$-minimal sets of assertions (i.e., computing $\subseteq$-maximal deletion repairs); we deal with inconsistency incurred on top of a consistent ontology, by arbitrary (non-monotonic) rules which access it with a query interface. Furthermore, we must consider multiple ABoxes at once (via updates), and use $\mathcal{EL}$ instead of $DL\text{-}Lite$. Refining our algorithm to compute $\subseteq$-maximal deletion repairs is possible.

The problem of computing support families is tightly related to finding solutions to an abduction problem, which was considered by Bienvenu (2008) for theories $\mathcal{T}$ expressed in $\mathcal{EL}$-terminologies. A hypothesis $\mathcal{H} = \{A_1, \dots, A_n\}$ is a set of atomic concepts, and an observation is another atomic concept. A solution to the abduction problem is any set $S \subseteq \mathcal{H}$, such that $\mathcal{T} \models \bigsqcap_{A_i \in S} A_i \sqsubseteq O$. Our setting is more general and involves also roles along with atomic concepts. Abduction has been studied in various related areas e.g., for $DL\text{-}Lite$ ontologies by Calvanese, Ortiz, Simkus, and Stefanoni (2013), for propositional logic by Eiter and Makino (2007), for datalog by Eiter et al. (1997) and Gottlob, Pichler, and Wei (2007), etc. Using incomplete support families for DL-atoms is related in spirit to approximate inconsistency-tolerant reasoning in DLs using restricted support sets as considered by Bienvenu and Rosati (2013); however, we focus on repair computation and model generation while Bienvenu and Rosati target inference from all repairs.

Our methods for constructing partial support families exploit the results on the logical difference between $\mathcal{EL}$ terminologies presented by Konev et al. (2012) and Ecke et al. (2013); recently they were extended to $\mathcal{ELHR}$ by Ludwig and Walther (2014) and to general TBoxes by Feng et al. (2015).

Repairing inconsistent non-monotonic logic programs has been investigated in the work of Sakama and Inoue (2003), where an approach for deleting rules based on extended abduction was studied; however, to restore consistency addition of rules is also possible. The latter was considered by Balduccini and Gelfond (2003), where under Occam's razor consistency-restoring rules may be added. Methods for explaining why the inconsistency arises in a logic program were studied, e.g., by Syrjänen (2006), who exploited model-based diagnosis of Reiter (1987) to debug a logic program. Generalized debugging of logic programs was investigated e.g., by Gebser, Pührer, Schaub, and Tompits (2008). Most recently, Schulz, Satoh, and Toni (2015) considered a characterization of reasons for inconsistency in extended logic programs (i.e., disjunction-free logic programs with both strong ("classical") negation and weak negation) in terms of "culprit" sets of literals, based on the well-founded and maximal partial stable model semantics, and a derivation-based method to explain such culprits has been described; however, it remains open how debugging of logic programs based on culprit sets could be done and whether this could be fruitfully extended to debugging DL-programs. The latter has been addressed by Oetsch, Pührer, and Tompits (2012) and is related to the challenging but, to the best of our knowledge, unexplored problem of repairing the rule part of a DL-program.

9. Conclusion

We have considered computing repair answer sets of DL-programs over $\mathcal{EL}$ ontologies, for which we generalized the support set approach of Eiter et al. (2014d, 2014b) for $DL\text{-}Lite_{\mathcal{A}}$ to work with incomplete families of supports sets; this advance is needed since in $\mathcal{EL}$ complete support families can be large or even infinite. We discussed how to generate support sets, by exploiting query rewriting over ontologies to datalog (Lutz et al., 2009; Rosati, 2007; Stefanoni et al., 2012), which is in contrast to the work by Eiter et al. (2014d), where TBox classification is invoked. Moreover, we have developed alternative techniques for effective computation of partial support families. Our approach is to approximate a relevant part of the TBox to $DL\text{-}Lite_{core}$ exploiting a notion of logical difference between $\mathcal{EL}$-terminologies, and then compute complete support families over an approximated TBox using methods of Eiter et al. (2014d). The obtained support family is complete, if the approximated TBox is logically equivalent to the original one.

To estimate the maximal size of support sets, we have analyzed the properties of a novel support hypergraph, which corresponds to a subgraph of an ontology hypergraph (Nortje et al., 2013; Ecke et al., 2013), where nodes encode ontology predicates (or pairs of them), while (hyper) edges reflect TBox inclusions. We have shown how traversing a support hypergraph one can conveniently compute an upper bound for the number of support sets for a given DL-atom. If, in addition, the support hypergraph satisfies certain conditions (e.g. tree-acyclicity), then an exact estimate can be obtained.

We developed a sound algorithm for computing deletion repair answer sets for DL-programs over $\mathcal{EL}$ ontologies, which is complete in case all support families are also known to be complete. The algorithm trades answer completeness for scalability (a simple variant ensures completeness). We have implemented the novel algorithm using declarative means within a system prototype, that invokes a REQUIEM reasoner for partial support family computation. For an experimental assessment of our repair approach, a set of novel benchmarks has been constructed including real world data. While the availability of complete support families adds to the scalability of the repair computation, partial support families work surprisingly well in practice due to the structure of the benchmark instances: the support sets are either small or there are just few of them, and thus post-evaluation checks do not cause much overhead. Overall, our experimental evaluation has revealed a promising potential of the novel repair methodology for practical applications.

9.1 Outlook

The directions for future work in the considered area are manifold. They cover both theoretical and practical aspects of our inconsistency handling approach. On the theoretical side, a relevant open issue are sufficient conditions under which computing all nonground support sets for a DL-atom accessing an $\mathcal{EL}$ ontology becomes tractable. Like in the work of Gebser et al. (2008) bounded tree-width might be considered, but also other parameters like density of a support hypergraph or various acyclicity properties. Analyzing the complexity of counting support sets in a complete support family might give hints to possible restricted settings, in which support family computation is efficient, but such a complexity analysis is also an interesting problem as such. On the practical side, optimization of the current implementation and extending the range of applications to real use cases is another issue.

Repair may be intermingled with stepping techniques used for debugging DL-programs (Oetsch et al., 2012). We considered the DL-programs as monolithic structures when applying our repair techniques, that is the repair computation was performed on a DL-program taken as a whole. It is an interesting and a relevant quest to extend the approach for dealing with modular DL-programs. Splitting a program into separate components that can be individually evaluated is a well-known programming technique, which has been studied in the context of DL-programs (Eiter et al., 2008). It is not clear, however, to which extent and for which program classes the repair methods can be adapted for the modular setting.

While we have considered $\mathcal{EL}$ in this paper, the basic algorithm and approach is applicable also to other DLs. Extensions of our work to $\mathcal{EL}^+$ and $\mathcal{EL}^{++}$ are easily possible. The main difference is negation, which is expressible via the $\perp$ concept; the ontology can get inconsistent through the updates of DL-atoms, leading to an increased number of support sets that need to be effectively computed and appropriately handled. The extension to expressive DLs such as $\mathcal{SHIQ}$, $\mathcal{SHOIN}$ or even $\mathcal{SROIQ}$ is more challenging, as efficient methods for support set construction remain to be developed; by the relatively high complexity of these DLs, this comes at a computational cost. On the other hand, the computation may be done once (even offline) and can be reused; fortunately, support families need not be complete, but we may expect a return of investment of time in support set construction for the overall running time.

Orthogonal to other DLs, one can study various additional repair possibilities, e.g. bounded addition; for overview of repair possibilities see the work of Eiter et al. (2013). Here we have concentrated on repairing the data part of the ontology, but it is also natural to allow changes in rules and interfaces. For repairing rules, the works on ASP debugging by Frühstück, Pührer, and Friedrich (2013), Gebser et al. (2008), and Syrjänen (2006) can be used as a starting point, but the problem is challenging as the search space of possible changes is large. Priorities on the rules and atoms involving them might be applied to ensure high quality of rule repairs. The interfaces similarly admit numerous modifications, which makes this type of repair as difficult; user interaction is most probably required.

Last but not least one could develop methods for repairing other hybrid formalisms including tightly-coupled hybrid KBs or even more general representations like HEX-programs (Eiter et al., 2005), where instead of an ontology arbitrary sources of computation can be accessed from a logic program. Heterogeneity of external sources in HEX-programs makes both repair and paraconsistent reasoning a very challenging task.

Acknowledgments

We thank the anonymous reviewers for their detailed and constructive suggestions which helped to improve this work. This article significantly extends preliminary work of Eiter, Fink, and Stepanova (2014c). The research has been supported by the Austrian Science Fund (FWF) projects P24090 and P27730.

Appendix A. Proofs for Section 3

A.1 Proof of Proposition 15

($\Rightarrow$) By Proposition 10, $I \models^{\mathcal{O}} d$ iff $\mathcal{T}_d \cup \mathcal{A} \cup \mathcal{A}_I \models Q(\vec{t})$, where $\mathcal{A}_I = \{P_p(\vec{t}) \in \mathcal{A}_d \mid p(\vec{t}) \in I\}$. Thus, $S = \mathcal{A} \cup \mathcal{A}_I$ is a support set of d w.r.t. $\mathcal{O}$, and it is coherent with I by construction.

($\Leftarrow$) If $S \in \mathit{Supp}_{\mathcal{O}}(d)$ is coherent with I, then S is of the form $S = \mathcal{A}' \cup \mathcal{A}'_I$ where $\mathcal{A}' \subseteq \mathcal{A}$ and $\mathcal{A}'_I \subseteq \mathcal{A}_I$, and thus $S \subseteq \mathcal{A} \cup \mathcal{A}_I$. As $\mathcal{T}_d \cup S \models Q(\vec{t})$, by monotonicity $\mathcal{T}_d \cup \mathcal{A} \cup \mathcal{A}_I \models Q(\vec{t})$, hence by Proposition 10 $I \models^{\mathcal{O}} d$.

A.2 Proof of Proposition 24

Consider any instance $S\theta = \{P_1(Y_1\theta), \ldots, P_k(Y_k\theta)\}$ of a set S of form (5) for $d(\vec{X})$, where $\theta : V \to \mathcal{C}$. We show that $S\theta$ is a support set w.r.t. $\mathcal{O}_{\mathcal{C}} = \langle \mathcal{T}, \mathcal{A}_{\mathcal{C}} \rangle$ (recall that $\mathcal{A}_{\mathcal{C}}$ is the set of all possible ABox assertions over $\mathcal{C}$), i.e., $S\theta \subseteq \mathcal{A}_{\mathcal{C}} \cup \mathcal{A}_d$ (which clearly holds) and $\mathcal{T}_d \cup S\theta \models Q(\vec{X}\theta)$. The latter is equivalent to $\mathcal{T}_{dnorm} \cup S\theta \models Q(\vec{X}\theta)$, which in turn by Lemma 23 is equivalent to $\mathit{Prog}_{Q,\mathcal{T}_{dnorm}} \cup S\theta \models Q(\vec{X}\theta)$. Let $\mathit{Prog}^0 = \mathit{Prog}_{Q,\mathcal{T}_{dnorm}}$, and let Prog^{i+1}, for each $i \geq 0$, be the program that results from Prog^i by unfolding a rule w.r.t. the target query $Q(\vec{X}\theta)$. Then $\mathit{Prog}^{i+1} \cup S\theta \models Q(\vec{X}\theta)$ iff $\mathit{Prog}^i \cup S\theta \models Q(\vec{X}\theta)$ holds. Now by construction of S, there is a rule r of the form (4) in some Prog^i. Clearly $\{r\theta\} \cup S\theta \models Q(\vec{X}\theta)$ and thus $\mathit{Prog}^i \cup S\theta \models Q(\vec{X}\theta)$. It follows that $\mathit{Prog}^0 \cup S\theta \models Q(\vec{X}\theta)$ and hence $\mathcal{T}_{dnorm} \cup S\theta \models Q(\vec{X}\theta)$ and $\mathcal{T}_d \cup S\theta \models Q(\vec{X}\theta)$.

Appendix B. Proofs for Section 4

B.1 Proof of Lemma 31

Towards a contradiction, assume $\mathcal{T}_{1d} \not\equiv^C_{\Sigma'} \mathcal{T}_{2d}$. Then w.l.o.g. $\mathcal{T}_{1d} \models P_1 \sqsubseteq P_2$ but $\mathcal{T}_{2d} \not\models P_1 \sqsubseteq P_2$, where $P_1, P_2 \in \Sigma'$. Observe that Σ and Σ' differ only on predicates P_p, such that $P \circ p$ occurs in λ, and that $\mathcal{T}' = \mathcal{T}_{1d} \backslash \mathcal{T}_1 = \mathcal{T}_{2d} \backslash \mathcal{T}_2$ consists only of axioms $P_p \sqsubseteq P$ where P_p does not occur in $\mathcal{T}_1$ or $\mathcal{T}_2$. We first show that $P_2 \in \Sigma$ must hold. Indeed, otherwise $P_2 \in \Sigma' \setminus \Sigma$ and thus $P_2 = P_p \in \mathsf{sig}(\mathcal{A}_d)$ for some $P \circ p$ from λ. Now let $\mathcal{A}' = \{P_1(c)\}$ if $P_1 \in \Sigma$, and $\mathcal{A}' = \{P_1(c), P_{1p}(c)\}$ otherwise (i.e., $P_1 \in \Sigma' \setminus \Sigma$), for an arbitrary $c \in \mathbf{I}$. Then $\mathcal{T}_{1d} \cup \mathcal{A}'$ has a model $\mathcal{I}$ in which $c^{\mathcal{I}} \in P_1^{\mathcal{I}}$ (resp. $c^{\mathcal{I}} \in P_1^{\mathcal{I}}$ and $c^{\mathcal{I}} \in P_{1p}^{\mathcal{I}}$) and $P_p{}^{\mathcal{I}} = \emptyset$ (thus $P_1 \neq P_2$), as $\mathcal{EL}$ is negation-free and P_p occurs in axioms only on the left. As $\mathcal{I} \not\models P_1 \sqsubseteq P_p$, it follows $\mathcal{T}_{1d} \not\models P_1 \sqsubseteq P_2$, which is a contradiction. This proves $P_2 \in \Sigma' \setminus \Sigma$. Now there are two cases.

(i) $P_1 \in \Sigma$: $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$ implies $\mathcal{T}_2 \models P_1 \sqsubseteq P_2$; by monotonicity $\mathcal{T}_{2d} \models P_1 \sqsubseteq P_2$, a contradiction.

(ii) $P_1 \in \Sigma' \setminus \Sigma$: then $P_1 = P_p$, where $P \circ p$ occurs in λ, and $P \in \Sigma$. We claim that $\mathcal{T}_1 \models P \sqsubseteq P_2$. Indeed, otherwise $\mathcal{T}_1$ has a model $\mathcal{I}$ such that $P^{\mathcal{I}} \not\subseteq P_2{}^{\mathcal{I}}$. Then as easily seen the interpretation $\mathcal{I}'$ that coincides with $\mathcal{I}$ on Σ and has $P_p{}^{\mathcal{I}'} = P^{\mathcal{I}} \setminus P_2{}^{\mathcal{I}}$ and $P'_p{}^{\mathcal{I}'} = \emptyset$ for each $P'_p \in \Sigma' \setminus \Sigma$ is a model of $\mathcal{T}_{1d}$; however, $\mathcal{I}' \not\models P_p \sqsubseteq P_2$, which would be a contradiction. This proves the claim. Now from the claim and $\mathcal{T}_1 \equiv^C_\Sigma \mathcal{T}_2$, it follows $\mathcal{T}_2 \models P \sqsubseteq P_2$ and by monotonicity $\mathcal{T}_{2d} \models P \sqsubseteq P_2$. As $P_p \sqsubseteq P \in \mathcal{T}_{2d}$, it follows $\mathcal{T}_{2d} \models P_1 \sqsubseteq P_2$; this is a contradiction.

B.2 Proof of Proposition 32

Suppose that $\mathbf{S}_1$ is a complete nonground support family w.r.t. $\mathcal{O}_1$ and let $S\theta$ be any instance of any $S \in \mathbf{S}_1$; then $S\theta = \mathcal{A}' \cup \mathcal{A}'_d \subseteq \mathcal{A}_{\mathcal{C}} \cup \mathcal{A}_d$. By Lemma 31, $\mathcal{T}_{1d} \equiv^C_\Sigma \mathcal{T}_{2d}$; thus by Theorem 30, $\mathcal{T}_{1d} \equiv^i_\Sigma$

$\mathcal{T}_{2d}$ as well. By definition of Σ-instance inseparability, for all Σ-ABoxes $\mathcal{A}''$ and Σ-assertions α such that $\mathcal{T}_{1d} \cup \mathcal{A}'' \models \alpha$, it holds that $\mathcal{T}_{2d} \cup \mathcal{A}'' \models \alpha$; hence $\mathcal{T}_{2d} \cup \mathcal{A}' \cup \mathcal{A}'_d \models Q(\vec{c})$. Consequently, $S\theta = \mathcal{A}' \cup \mathcal{A}'_d$ is a (ground) support set of d w.r.t. $\mathcal{O}_2$. If $\mathbf{S}_2$ is a complete nonground support family w.r.t. $\mathcal{O}_2$, it follows that $S\theta$ is an instance of some $S' \in \mathbf{S}_2$. The converse membership is symmetric. Hence, $\mathbf{S}_1$ and $\mathbf{S}_2$ are ground-identical.

B.3 Proof of Proposition 34

Towards a contradiction, assume some $S' \in \mathbf{S} \setminus \mathbf{Supp}_{\mathcal{O}}(d)$ exists. Then a grounding θ exists such that $S'\theta \cup \mathcal{T}_d \not\models d(X\theta)$. However, $S'\theta \cup \mathcal{T}''_d \models d(X\theta)$, as according to (f), S' is a nonground support set for d w.r.t. $\mathcal{T}''_d = \mathcal{T}'_d \cup lrw$. Consequently, $\mathcal{T}_d \not\models \mathcal{T}''_d$, which is a contradiction, because $\mathcal{T}'_d \subseteq \mathcal{T}_d$ by construction in (c) and $lrw = \{C' \sqsubseteq D' \mid \mathcal{T}_d \models C' \sqsubseteq D',\ \mathcal{T}'_d \not\models C' \sqsubseteq D'\} \subseteq \mathcal{T}_d$ by (d) and definition of $\mathsf{cWTn}^{rhs}_{\Sigma}$ and $\mathsf{cWTn}^{lhs}_{\Sigma}$.

Appendix C. Proofs for Section 5

C.1 Proof of Lemma 47

The construction of support sets from a given hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ for d w.r.t. the ontology $\mathcal{O} = \langle \mathcal{T}, \mathcal{A} \rangle$ that we have presented mimics the DL-query unfolding over the TBox $\mathcal{T}_d$. We now formally show that (i) each set S extracted in the described way is indeed a nonground support sets for d, and (ii) for each ground instance $S\theta$ of a nonground support set S for d, a (nonground) support set S' can be constructed following our procedure such that $S'\theta' \subseteq S\theta$ for some suitable ground substitution θ'. This proves that $\mathbf{S}_{\mathcal{G}} \subseteq_{\theta} \mathbf{S}$ holds.

We first prove (i) by induction on the length n of incoming paths, from which the support sets are extracted.

Base: n=1. Consider any path π in the hypergraph $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$. Assume that there is a single (hyper-) edge e in π. By construction, this hyperedge must have x_Q as a head node, i.e. $head(e) = x_Q$. There are four possibilities: (1) $tail(e) = \{x_C\}$, (2) $tail(e) = \{x_r, x_C\}$, (3) $tail(e) = \{x_C, x_D\}$ or (4) $tail(e) = \{x_r, \top\}$. We annotate the nodes of a path by variables as described above, and extract the nonground atoms from labels and annotations of the nodes. As a result for the case (1) we obtain $\{C(X_0)\}$, for (2): $\{r(X_0, X_1), C(X_1)\}$, for (3): $\{C(X_0), D(X_0)\}$, and for (4): $\{r(X_0, X_1)\}$, where X_1 is a fresh variable. By construction of the hypergraph the edges of the forms (1)-(4) correspond to the TBox axioms $C \sqsubseteq Q$, $\exists r.C \sqsubseteq Q$, $C \sqcap D \sqsubseteq Q$ and $\exists r.\top \sqsubseteq Q$ respectively. Therefore, the sets that have been constructed in all of the considered cases reflect the DL-query unfoldings of d, and hence they represent nonground support sets for d by Proposition 24.

Induction step: Suppose that the statement is true for n, i.e. from a path with n edges all sets extracted in the way described above are nonground support sets for d. Consider a path $\pi = e_0, \dots, e_n$ with $n+1$ edges, and let $e = e_0$ be the first edge of π. By the induction hypothesis, all sets extracted from the path $\pi \setminus e = e_1, \dots, e_n$ following our approach are support sets for d. There are several possibilities for the form of e: (1) $tail(e) = \{x_C\}$ and $head(e) = \{x_D\}$, (2) $tail(e) = \{x_r, x_C\}$ and $head(e) = \{x_D\}$, (3) $tail(e) = \{x_C, x_D\}$ and $head(e) = \{x_B\}$, (4) $tail(e) = \{x_r, \top\}$ and $head(e) = \{x_C\}$, or (5) $tail(e) = \{x_C\}$ and $head(e) = \{x_r, x_D\}$.

As for (1), by construction both x_C and x_D are annotated with X_i. Let $\mathbf{S}$ be a family of sets extracted from $\pi \setminus e$. We pick a set S in which $C(X_i)$ occurs. We substitute $C(X_i)$ in S with $D(X_i)$, and obtain a set S'. By the induction hypothesis S must be a support set for d. However, then clearly S' is also a support set, as it mimics an additional unfolding step that accounts for the rule $C(X) \leftarrow D(X)$ of the datalog rewriting of $\mathcal{T}_d$.

Let us look at (2). Assume a set $S \supseteq D(X_i)$ of nonground atoms has been constructed using our procedure. Then X_i must be an annotation for x_D. According to our construction $\{x_r, x_D\}$ is annotated with $\{\langle X_i, X_j\rangle, \langle X_j\rangle\}$, where X_j is a fresh variable. The sets S' that we get from π result by substituting $D(X_i)$ in some S with $\{r(X_i, X_j), C(X_j)\}$. The latter mimics the unfolding step for Q that accounts for the rule $D(X_i) \leftarrow r(X_i, X_j), C(X_j)$ of the rewriting $\mathcal{T}_d$. As S is a support set for d by the induction hypothesis, S' must be a support set for d as well. The cases (3)-(5) can be analyzed analogously. Thus all sets of size $n+1$ extracted from π are support sets for d.

It remains to prove (ii). Towards a contradiction, assume that some ground instance $S\theta$ of some $S \in \mathbf{Supp}_{\mathcal{O}}(d)$ exists, such that for each ground instance $S'\theta'$ of every $S' \in \mathbf{Supp}_{\mathcal{O}}(d)$ constructed by our procedure we have $S'\theta' \not\subseteq S\theta$. As $S\theta$ is a support set, by definition $\mathcal{T}_{dnorm} \cup S\theta \models Q(\vec{c})$, thus by Lemma 23 $Prog_{Q,\mathcal{T}_{dnorm}} \cup S\theta \models Q(\vec{c})$. This in turn means that $Q(\vec{c})$ has a backchaining proof $S_0, S_1, \dots, S_m$ from $Prog_{Q,\mathcal{T}_{dnorm}} \cup S\theta$ of the form $S_0 = Q(\vec{X})\theta_0$ and $S_m = \emptyset$, where θ_0 is the substitution $\vec{X} \mapsto \vec{c}$, and $S_i = (S_{i-1} - H_i + B_i)\theta_i$, $i \geq 1$, where $H_i \leftarrow B_i$ is a rule resp. fact in $Prog_{Q,\mathcal{T}_{dnorm}} \cup S$ and θ_i is the most general unifier of H_i with some atom in S_{i-1}. Without loss of generality, we have $H_i = A_2(o_{A_2})$ if $H_{i-1} = R_2(X, o_{A_2})$ and all i such that B_i is empty are at the end, i.e. at the positions $k, k+1, \dots, m$. Then each S_j resp. S_{j+1}, $0 \leq j \leq k$ amounts to an instance of a support set S'_j resp. S'_{j+1} of d generated from $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$. In particular, S_{k-1} is an instance of S'_{k-1} and consequently $\{H_k, H_{k+1}, \dots, H_m\}$ $(\subseteq S\theta)$ is an instance of S'_{k-1} as well. But this means $S'\theta' \subseteq S\theta$ for some instance $S'\theta'$ of $S' = S_{k-1}$, a contradiction.

C.2 Proof of Proposition 48

We prove the statement by induction on the number n of hyperedges with a singleton head node in $\mathcal{G} = \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ for the DL-atom DL[λ; Q](X).

Base: $n = 0$. We show that $maxsup(d) = 1$ if no hyperedges of the required form exist in $\mathcal{G}$. Several cases are possible: (i) $\mathcal{G}$ contains only hyperedges of the form $(x_C, \{x_r, x_D\})$; (ii) $\mathcal{G}$ has only hyperedges of the form $(\{x_r, \top\}, x_C)$ or $(x_C, \{x_r, \top\})$; or (iii) $\mathcal{G}$ has no hyperedges.

(i) Consider some hyperedge in π. Then some e_j must exist in π, such that $head(e_i) \subseteq tail(e_j)$. The latter implies that e_j is of the form $(\{x_r, x_D\}, x_{D'})$ but then $n \neq 0$, i.e. contradiction.

For (ii) and (iii), by construction $\mathcal{T}$ contains only GCIs $C \sqsubseteq D$ such that C, D are either atomic or of the form $\exists r.\top$. These axioms fall into the $DL\text{-}Lite_{core}$ fragment, for which all $\subseteq_\theta$-minimal support sets S have size at most 2; moreover, $|S| = 2$ reflects in $DL\text{-}Lite_{core}$ inconsistency arising in the updated ontology (Eiter et al., 2014d). As negation is not available nor expressible in $\mathcal{EL}$, no such S exists and thus the maximal support set size for d is 1.

Induction Step: Suppose that the statement is true for n; we prove it for $n+1$. Let $\pi = e_1, \dots, e_k$ be an incoming path to x_Q in $\mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$ with a maximal number $n+1$ of hyperedges with a singleton head node. Assume that e_i is the first hyperedge of the required form occurring in π. Let us split π into two parts: $e_1, \dots, e_i$ and $e_{i+1}, \dots, e_k$. Consider the hypergraph $\mathcal{G}'' = (\mathcal{V}, \mathcal{E}'')$, where $\mathcal{E}'' = \mathcal{E} \setminus \{e_1, \dots, e_i\}$, and the TBox $\mathcal{T}''$ reconstructed from it. By the induction hypothesis, $maxsup(d)$ w.r.t. $\mathcal{O}'' = \langle \mathcal{T}'', \mathcal{A}\rangle$ is bounded by $n+1$. Now let the hypergraph $\mathcal{G}' = (\mathcal{V}, \mathcal{E}')$ with $\mathcal{E}' = \mathcal{E}'' \cup \{e_i\}$ correspond to the TBox $\mathcal{T}'$. By our assumption $head(e_i) = x_A$, i.e. e_i either reflects $B \sqcap C \sqsubseteq A$ or $\exists r.B \sqsubseteq A$. Two cases are possible: either $A = Q$ or $A \neq Q$. In the

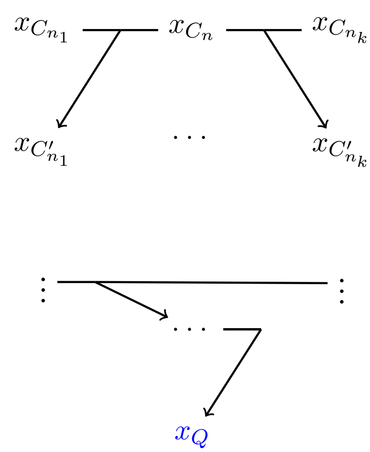


Figure 11: Fragment of a hypergraph used for illustration in the proof of Proposition 53

former case, e_i is a single hyperedge on π, i.e. $n = 1$. Support sets obtained from rewriting Q over $B \sqcap C \sqsubseteq Q$ or $\exists r.B \sqsubseteq Q$ are of size at most 2. The other support sets are constructed by combining query rewritings of predicates occurring on the left hand side of GCIs reflected by e_i; each of these rewritings has size at most 1 as shown in the base case. Thus the overall support set size for d w.r.t. $\mathcal{T}'$ is bounded by $2 \leq n + 1$.

Suppose now that $A \neq Q$, i.e. e_i reflects either $B \sqcap C \sqsubseteq A$ or $\exists r.B \sqsubseteq A$. By definition of an incoming path a (hyper) edge e_j must exist, such that $head(e_i) \subseteq tail(e_j)$. Moreover, note that e_j is a unique (hyper) edge connected to e_i on π, as otherwise the given hypergraph is tree-cyclic, i.e. contradiction. We distinguish two cases: (1) $head(e_i) = tail(e_j)$ and e_j corresponds to $A \sqsubseteq \dots$; (2) $head(e_i) \subset tail(e_j)$ and e_j reflects $A \sqcap B \sqsubseteq \dots$.

1. Consider a maximal support set S for d w.r.t. $\mathcal{T}''$, and suppose $A(Y) \in S$ holds. By induction hypothesis $|S| \leq n$. As $\mathcal{G}'' = \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}''}$ is tree-acyclic, only a single atom over A might occur in S. Adding the edge e_i to $\mathcal{G}''$ from S we obtain a support set S' with the atom $A(Y)$ substituted with atoms $B(Y)$ and $C(Y)$, or $r(Z, Y)$ and $B(Z)$ as a result of an additional query unfolding step. Hence the support set size of S' will be bounded by $n + 2$.
2. If e_j reflects $A \sqcap B \sqsubseteq \dots$, then a support set $S \supseteq \{A(Y), B(Y)\}$ must exist. By unfolding the respective datalog rule, we get the bound $n + 2$ on the support set S' for d w.r.t. $\mathcal{T}'$. $\square$

C.3 Proof Sketch of Proposition 53

Observe that in tree-acyclic hypergraphs all nodes have a hyper out-degree at most 1, and hence $m(\pi, \mathcal{G}) = 0$. Thus, if $\mathcal{G}$ is tree-acyclic, then by Proposition 48 the support set size for a given DL-atom is bounded by $n(\pi, \mathcal{G}) - 0 + 1$, which equals s_{max}. We now show that the claimed bound is also correct for tree-cyclic hypergraphs. Intuitively, $m(\pi, \mathcal{G})$ must be subtracted from $n(\pi, \mathcal{G})$ to avoid that certain atoms in a support set are counted multiple times. Regarding the structure of the support hypergraph we distinguish two cases: (i) no roles appear in a hypergraph; (ii) for all $x_r \in \mathcal{G}$, it holds that $r \notin \Sigma$.

First we consider (i). Since only concepts appear in the support hypergraph by our assumption, all support sets will contain atoms in which only a single variable X_0 occurs. Consider some node x_{C_n} in π such that $hdc^+(x_{C_n}) = k$, where $k > 1$, i.e., there are k outgoing hyperedges from x_{C_n} containing nodes corresponding to concepts: $(\{x_{C_{n_1}}, x_{C_n}\}, x_{C'_{n_1}})\dots(\{x_{C_{n_k}}, x_{C_n}\}, x_{C'_{n_k}})$ (see Figure 11). From support sets $S \supseteq \{C'_{n_1}(X_0), \dots, C'_{n_k}(X_0)\}$ we will get support sets $S' \supseteq \{C_{n_1}(X_0), \dots, C_{n_k}(X_0), C_n(X_0)\}$. Estimating the maximal support set size as the number of hyperedges in the hypergraph, $C_n(X_0)$ is counted k times, but it appears only once (as its variable is guaranteed to be X_0). To avoid such multiple countings, $m(\pi, \mathcal{G})$ must be subtracted from $n(\pi, \mathcal{G})$.

Consider now (ii). By construction of $\mathcal{G}$, for every hypernode $\{x_r, x_C\} \in \pi$ edges $e_1 = (x_A, \{x_r, x_C\})$ and $e_2 = (\{x_r, x_C\}, x_B)$ exist in $\mathcal{G}$. Thus if x_r occurs in π, then consider a support set $S \supseteq \{B(X)\}$. Rewriting the TBox axiom reflected by e_2, we get a datalog rule $B(X) \leftarrow r(X, Y), C(Y)$. Then the axiom $\exists r.C \sqsubseteq A$ reflected by e_1 is rewritten to datalog rules $r(X, o_C) \leftarrow A(X);\ C(o_C) \leftarrow A(X)$. Unifying Y with o_C we obtain an unfolding $A(X)$. This essentially shows that if no role occurring in a support hypergraph is in Σ, then all support sets involve only a single variable; in this case, as shown in (i), the provided bound is correct.

C.4 Proof of Proposition 57

The proof is by induction on the number n of (hyper) edges in $\mathcal{G} = \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$. *Base: n=0.* If $\mathcal{G}$ has no (hyper) edges, each node has one support set.

Induction step: Suppose the statement holds for n; we show it holds for $\mathcal{G}$ with $n+1$ (hyper) edges. Obviously, it holds for $x \in \mathcal{V}_{\mathbf{R}}$. As $\mathcal{G}$ is tree-acyclic and $\mathcal{T}$ is in normal form, $\mathcal{G}$ has a node x such that $hd^+(x) = d^+(x) = 0$, i.e., there are no outgoing (hyper) edges, and $hd^-(x) \neq 0$ or $d^-(x) \neq 0$, i.e., there is some incoming (hyper) edge. As $\mathcal{G}$ is tree-acyclic, the rewriting of the set $Q'_x = \{A(X)\}$, where $x = x_A$ consists of Q'_x and the rewritings of all sets $Q'_{tail(e)}$ of (hyper) nodes $tail(e)$ such that $head(e) = x$. If $tail(e)$ is $\{x_B\}$ (resp., $\{x_B, x_C\}$, $\{x_r, x_C\}$) these are all rewritings of $\{B(X)\}$ (resp. $\{B(X), C(X)\}$, $\{R(X, Y), C(Y)\}$). That is, $ws(x_A)$ is the sum of the number of all rewritings of each $Q'_{tail(e)}$ denoted $Q'_{tail(e)}$, plus 1. Consider now an arbitrary e with $head(e) = x_A$ and let $\mathcal{G}' = \mathcal{G} \backslash e$. As $\mathcal{G}'$ has n edges and is tree-acyclic, by the induction hypothesis for each node $x \in \mathcal{V}$ in $\mathcal{G}'$, the value of $ws(x)$, denoted $ws_{\mathcal{G}'}(x)$, is as in (8). Furthermore, $ws(Q'_{tail(e)})$ and $ws(x')$, $x' \neq x_A$ is in $\mathcal{G}'$ the same as in $\mathcal{G}$. We thus get for $x = x_A$:

$$\begin{aligned}
ws_{\mathcal{G}}(x) &= ws_{\mathcal{G}'}(x) + ws(Q'_{tail(e)}) \\
&= 1 + \sum_{T \in T^{-\prime}(x)} \prod_{x' \in T} ws_{\mathcal{G}'}(x') + \sum_{T \in T^{-\prime}(x), T \not\subseteq \mathcal{V}_{\mathbf{C}}} \sum_{(\{x'\}, T) \in \mathcal{E}'} ws(x') + ws(Q'_{tail(e)}) \\
&= 1 + \sum_{T \in T^{-\prime}(x)} \prod_{x' \in T} ws_{\mathcal{G}}(x') + \sum_{T \in T^{-\prime}(x), T \not\subseteq \mathcal{V}_{\mathbf{C}}} \sum_{(\{x'\}, T) \in \mathcal{E}'} ws(x') + ws(Q'_{tail(e)}) \\
&= 1 + \sum_{T \in T^{-}(x)} \prod_{x' \in T} ws_{\mathcal{G}}(x') + \sum_{T \in T^{-}(x), T \not\subseteq \mathcal{V}_{\mathbf{C}}} \sum_{(\{x'\}, T) \in \mathcal{E}} ws(x')
\end{aligned}$$

where $T^{-\prime}(x) = \{T \mid (T, \{x\}) \in \mathcal{E}'\}$ and $\mathcal{E}' = \mathcal{E} \setminus \{e\}$, and $T^-(x)$ is as above. To obtain $ws(Q'_{tail}(e))$, we simply need to count the combinations of the rewritings of each node in $tail(e)$, and in case $tail(e) = \{x_r, x_B\}$ (where $ws(x_r) = 1$), we need to add the number of rewritings of the tail of each hyperedge $(T, \{x_r, x_B\})$ (as $\mathcal{T}$ is in normal form, T must be of the form $\{x_C\}$).

C.5 Proof of Corollary 60

This is immediate from Proposition 57: under the hypothesis, in (8) each T is of form $\{y\} \subseteq \mathcal{V}_{\mathbf{C}}$; thus $\prod_{x' \in T} ws(x') = ws(y)$, i.e., $ws(tail(e))$ and the rightmost term is 0.

C.6 Proof Sketch of Proposition 61

Under the condition on e and e_1, e_2, every set $T \in T^-(x)$ in Equation (8) such that $|T| = \{x, y\} > 1$ contains (at least) one element, say x, such that $ws(x) = 1$, and thus $\prod_{x' \in T} ws_{\mathcal{G}}(x')$ equals $ws(y)$ in $\mathcal{G} = \mathcal{G}^{\Sigma}_{supp(d),\mathcal{T}}$. By an inductive argument, we then obtain that for every node $x_A \in \mathcal{V}_{\mathbf{C}}$, $ws(x_A) - 1$ is the number of distinct edges in $\mathcal{G}$ that occur on incoming paths to x_A and any $x_B \in \mathcal{V}_{\mathbf{C}}$ such that an edge $(\{x_B\}, \{x_r, x_A\})$ is in $\mathcal{E}$, plus the number of all such edges. This in turn implies that for the query node x_Q, $ws(x_q) = |\mathcal{E}| + 1$ holds, as by construction each edge $e \in \mathcal{E}$ is among the respective edges for x_Q. From this the result follows immediately.

Appendix D. Proofs for Section 6

D.1 Proof of Theorem 65

Suppose $SupRAnsSet$ outputs $I = \hat{I}|_{\Pi}$. We can get to (h) only if $\hat{I}$ is an answer set of $\hat{\Pi}$, and if the foundedness check of I w.r.t. the ontology $\mathcal{T} \cup \mathcal{A}'$, where $\mathcal{A}' = \mathcal{A} \backslash H$ succeeded. It thus remains to show that $\hat{I}$ is a compatible set for $\mathcal{T} \cup \mathcal{A}'$, i.e., that for each DL-atom d in Π, $d \in D_p$ iff $I \models^{\mathcal{O}'} d$ and $d \in D_n$ iff $I \not\models^{\mathcal{O}'} d$. Towards a contradiction, suppose that this is not the case. In (d) we partitioned the DL-atoms into two sets: D_p and D_n, corresponding to DL-atoms d guessed to be true and false in $\hat{I}$, respectively, and set $\mathbf{S}^{\hat{I}}_{gr}$ to $Gr(\mathbf{S}, \hat{I}, \mathcal{A})$. Since we assume that $\hat{I}$ is not compatible, one of the following must hold:

(1) For some DL-atom d in D_n, we have $I \models^{\mathcal{O}'} d$. There are two possibilities: (i) either there is a support set $S \in \mathbf{S}^{\hat{I}}_{gr}(d)$ or (ii) no support sets for d were identified. In case (i), we are guaranteed that all support sets S for d are such that $S \cap \mathcal{A} \neq \emptyset$, since otherwise no hitting sets H are found in (e). Hence there must exist some support set S such that $S \cap \mathcal{A} \neq \emptyset$. According to (e) $S \cap H \neq \emptyset$ and thus $S \notin Supp_d(\mathcal{O}')$. Now as $rep = true$ at (h), a post-check of d must have succeeded in (g), i.e. $I \not\models^{\mathcal{O}'} d$ must hold. This is a contradiction. In case (ii), likewise post-evaluation of d must have succeeded in (h), which again raises a contradiction.

(2) For some DL-atom d in D_p, we have $I \not\models^{\mathcal{O}'} d$. Hence $\mathbf{S}^{\hat{I}}_{gr}(d) = \emptyset$, $d \notin D'_p$, and post-evaluation is performed for d in (g). The latter, however, must have succeeded, as $rep = true$ at (h); this is a contradiction. Hence $\hat{I}$ is a compatible set for Π', and thus a deletion repair answer set of Π.

D.2 Proof of Theorem 66

The following lemmas are useful to prove Theorem 66.

Lemma 69 *Let $I \in AS_x(\Pi)$ where $x \in \{flp, weak\}$ and $\Pi = \langle \mathcal{T}, \mathcal{A}, \mathcal{P} \rangle$ is a ground DL-program. Then $\hat{I} = I \cup \{e_d \mid d \in DL_{\Pi}, I \models^{\mathcal{T} \cup \mathcal{A}} d\} \cup \{ne_d \mid d \in DL_{\Pi}, I \not\models^{\mathcal{T} \cup \mathcal{A}} d\}$ is an answer set of $\hat{\Pi}$, where DL_{Π} is the set of all DL-atoms occurring in Π.*

This lemma follows from a more general result on *compatible sets* as the basis of the evaluation approach of HEX-programs in the DLVHEX-solver (cf. (Eiter et al., 2014a)).

Lemma 70 *Let $\Pi = \langle \mathcal{T}, \mathcal{A}, \mathcal{P} \rangle$ be a ground DL-program and let $\hat{I} \in AS(\hat{\Pi})$ such that $I = \hat{I}|_{\Pi} \in AS_x(\Pi)$, where $x \in \{flp, weak\}$. Suppose $\mathcal{A}' \supseteq \mathcal{A}$ is such that for each DL-atom d occurring in $\mathcal{P}$, it holds that $I \models^{\mathcal{T} \cup \mathcal{A}} d$ iff $I \models^{\mathcal{T} \cup \mathcal{A}'} d$. Then $I \in AS_x(\Pi')$ where $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$.*

Proof. We note that for $I = \hat{I}|_{\Pi}$, $\mathcal{P}_x^{I,\mathcal{T}\cup\mathcal{A}}$ and $\mathcal{P}_x^{I,\mathcal{T}\cup\mathcal{A}'}$ coincide; as $I \in AS_x(\Pi)$, it is a minimal model of $\mathcal{P}_x^{I,\mathcal{T}\cup\mathcal{A}}$. Consequently, I is also a model of $\mathcal{P}_x^{I,\mathcal{T}\cup\mathcal{A}'}$. Moreover, I is minimal, as if some $J \subset I$ satisfies $\mathcal{P}_x^{I,\mathcal{T}\cup\mathcal{A}'}$, then $J \models \mathcal{P}_x^{I,\mathcal{T}\cup\mathcal{A}}$; hence I is not an answer set of $\mathcal{P}_x^{I,\mathcal{T}\cup\mathcal{A}}$, a contradiction. $\square$

Suppose $I \in RAS_x(\Pi)$. This implies that $I \in AS_x(\Pi')$ where $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', \mathcal{P} \rangle$, for some $\mathcal{A}' \subseteq \mathcal{A}$. By Lemma 69 $\hat{I}$ is an answer set of $\hat{\Pi}$ and thus is considered in (c). In (d), D_p and D_n are set to the (correct) guess for $I \models^{\mathcal{O}'} d$ for each DL-atom d, where $\mathcal{O}' = \mathcal{T} \cup \mathcal{A}'$. From Proposition 15 and θ-completeness of $\mathbf{S}$, we obtain for each $d \in D_p$ that $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(d) \neq \emptyset$ and for each $d \in D_n$ that $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(d) = \emptyset$. As $Gr(\mathbf{S}, \hat{I}, \mathcal{A}')(d) \subseteq Gr(\mathbf{S}, \hat{I}, \mathcal{A})(d)$ holds for each DL-atom d, it follows for each $d \in D_n$ and $S \in \mathbf{S}_{gr}^{\hat{I}}(d)$ that $S \cap (\mathcal{A} \setminus \mathcal{A}') \neq \emptyset$; this means that $H' = \mathcal{A} \setminus \mathcal{A}'$ is a hitting set of $\bigcup_{d' \in D_n} \mathbf{S}_{gr}^{\hat{I}}(d')$, and hence some minimal hitting set $H \subseteq H'$ will be considered in (e). In (f), D'_p will be set to D_p as for each $d \in D_p$ some $S \in \mathbf{S}_{gr}^{\hat{I}}(d)$ exists such that $S \cap H' = \emptyset$, and hence $S \cap H = \emptyset$. Thus in (g) the call $eval_p(\cdots)$ yields true, and likewise the call $eval_n(\cdots)$ as $Gr(\mathbf{S}, \hat{I}, \mathcal{A} \setminus H)(d) = \emptyset$; thus rep is true. Eventually, in (h) the test *flpFND*$(\hat{I}, \langle \mathcal{T} \cup \mathcal{A} \setminus H, \mathcal{P} \rangle)$ will succeed, as I is an x-answer set of $\Pi' = \langle \mathcal{T} \cup \mathcal{A}', \mathcal{P} \rangle$, and by Lemma 70 also of $\Pi'' = \langle \mathcal{T} \cup \mathcal{A} \setminus H, \mathcal{P} \rangle$, as $\mathcal{A}' \subseteq \mathcal{A} \setminus H$. Thus in step (h) $I = \hat{I}|_{\Pi}$ is output. $\square$

D.3 Proof of Proposition 67

We first show that for every $\hat{I} \in AS(\Pi_1)$, it holds that $\hat{I}|_{\Pi} \in RAS_{weak}(\Pi)$. Towards a contradiction, suppose some $\hat{I} \in AS(\Pi_1)$ exists such that $\hat{I}|_{\Pi} \notin RAS_{weak}(\Pi)$. Then for every $\mathcal{A}' \subseteq \mathcal{A}$ we have that $\hat{I}|_{\Pi} \notin AS_{weak}(\Pi')$ with $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. In particular, for $\mathcal{A}'' = \mathcal{A} \backslash \{P(\vec{c}) \mid \bar{p}_P(\vec{c}) \in I\}$ it holds that $\hat{I}|_{\Pi} \notin AS_{weak}(\Pi'')$ with $\Pi'' = \langle \mathcal{T}, \mathcal{A}'', \mathcal{P} \rangle$ There are several possibilities: (i) no extension of $\hat{I}|_{\Pi}$ with a guess for the replacement atoms e_d, ne_d is a model of $\widehat{\Pi''}$; (ii) no such extension of $\hat{I}|_{\Pi}$ is a compatible set for Π''; (iii) some interpretation $\hat{J} \subset \hat{I}|_{\Pi}$ is a model of $\mathcal{P}_{weak}^{\hat{I}|_{\Pi},\mathcal{O}''}$.

The case (i) is impossible: $\hat{\Pi} = \widehat{\Pi''}$ and hence it follows that $\hat{I}|_{\hat{\Pi}} \models \widehat{\Pi''}$.

Assume that (ii) is true. Consider the interpretation $\hat{I}|_{\hat{\Pi}}$. Towards a contradiction, assume that it is not compatible for Π''. Then for some DL-atom d either (1) $\hat{I}|_{\Pi} \models^{\mathcal{O}''} d$ and $ne_d \in \hat{I}|_{\hat{\Pi}}$, or (2) $\hat{I} \not\models^{\mathcal{O}''} d$, and $e_d \in \hat{I}|_{\hat{\Pi}}$ holds. In case (1), as $\hat{I}|_{\Pi} \models^{\mathcal{O}''} d$, some support set S for d that is coherent with $\hat{I}|_{\hat{\Pi}}$ exists. Now consider whether $S \in \mathcal{S}_d$ or $S \notin \mathcal{S}_d$. In the former case, S must contain ABox assertions $S_d^{\mathcal{A}}$, as otherwise some constraint of the form (r_5^*) is violated. Due to the rule (r_6^*) at least one assertion P_{id} in $S_d^{\mathcal{A}}$ must be marked for deletion. Note that then P_{id} is not present in $\mathcal{A}''$, and S is not a relevant support set for d w.r.t. $\mathcal{A}''$. If $\mathcal{S}_d$ is known to be complete, then we immediately arrive at a contradiction. Otherwise, the rule of the form (r_8^*) is applied, and as the evaluation postcheck for d succeeded by our assumption, we get a contradiction. If $S \notin \mathcal{S}_d$, then $\mathcal{S}_d$ is not known to be complete, and again the rule of the form (r_8^*) is applied; due to the successful evaluation postcheck, a contradiction is obtained. Now suppose that (2) is true. As $\hat{I}|_{\Pi} \not\models^{\mathcal{O}'} d$, no support set for d exists w.r.t. $\mathcal{O}$ that is coherent with $\hat{I}|_{\Pi}$. If $\mathcal{S}_d$ is known to be complete, then the constraint (r_9^*) is violated; but this contradicts $\hat{I} \models \Pi_1$. Thus, the body of the rule (r_7^*) is satisfied, and an evaluation postcheck is issued for d that fails; hence we get a contradiction.

Finally, assume that (iii) holds, i.e. some interpretation $\hat{J} \subset \hat{I}|_\Pi$ is a model of $\mathcal{P}_{weak}^{\hat{I}|_\Pi,\mathcal{O}''}$. The set $M = \hat{I}|_\Pi \backslash \hat{J}$ contains only atoms over the signature of Π. Let us consider $I_M = \hat{I} \setminus M$. We know that $\hat{I} \in AS(\Pi_1)$; Hence some rule $r_{gl}^{\hat{I}}$ must exist in ${\Pi_1}_{gl}^{\hat{I}}$ such that $I_M \models B(r_{gl}^{\hat{I}})$, but $I_M \not\models H(r_{gl}^{\hat{I}})$. Recall that $\Pi_1 = (\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})$. Now $r_{gl}^{\hat{I}}$ can not be in $(\hat{\Pi} \cup facts(\mathcal{A}) \cup \mathcal{COMP})_{gl}^{\hat{I}}$, as $r_{gl}^{\hat{I}} \in \mathcal{P}_{weak}^{\hat{I}|_\Pi,\mathcal{O}''}$ iff $r_{gl}^{\hat{I}} \in \mathcal{P}_{weak}^{\hat{I}|_\Pi,\mathcal{O}}$ and $\hat{J} \not\models \mathcal{P}_{weak}^{\hat{I}|_\Pi,\mathcal{O}}$ by construction of the GL and weak reducts, which is a contradiction. Therefore, $r_{gl}^{\hat{I}}$ must be in $\mathcal{R}_{gl}^{\hat{I}}$. However, the latter also raises a contradiction: no rule in $\mathcal{R}_{gl}^{\hat{I}}$ has atoms over the signature of Π in its head and I_M and $\hat{I}$ coincide on the rule head; thus it follows $\hat{I} \not\models B(\mathcal{R}_{gl}^{\hat{I}})$, which is a contradiction. Therefore, $\hat{I}|_\Pi \in AS(\Pi'')$ holds, and we have a global contradiction, i.e. $\hat{I}|_\Pi \in RAS_{weak}(\Pi)$ follows.

We now consider the case where each support family $\mathbf{S}_d$ is known to be complete, and prove that then $AS|_\Pi(\Pi_1) = RAS_{weak}(\Pi)$. From what has been shown above, it remains to check that $AS|_\Pi(\Pi_1) \supseteq RAS_{weak}(\Pi)$. Towards a contradiction, assume some $I \in RAS_{weak}(\Pi)$ exists such that $\hat{I} \notin AS(\Pi_1)$ for every extension $\hat{I}$ of I. As $I \in RAS_{weak}(\Pi)$, some ABox $\mathcal{A}' \subseteq \mathcal{A}$ exists such that $I \in AS(\Pi')$ with $\Pi' = \langle \mathcal{T}, \mathcal{A}', \mathcal{P} \rangle$. We construct an extension $\hat{I}$ of I as follows:

$$\begin{aligned}
\hat{I} \;=\; & I \cup \{e_d \mid I \models^{\mathcal{O}'} d\} \cup \{ne_d \mid I \not\models^{\mathcal{O}'} d\} \cup \\
& \{\bar{p}_P(\vec{c}) \mid P(\vec{c}) \in \mathcal{A}\backslash\mathcal{A}'\} \cup facts(\mathcal{A}) \cup \mathcal{COMP} \cup \\
& \{Sup_d(\vec{c}) \mid d(\vec{c}) \text{ has some support set from } \mathbf{S}_d \text{ coherent with } I\} \cup \\
& \{S_d^{\mathcal{P}}(\vec{c}) \mid I \models rb(S_d^{\mathcal{A},\mathcal{P}}(\vec{c}))\} \cup \{S_d^{\mathcal{A},\mathcal{P}}(\vec{c}) \mid \hat{I} \models rb(S_d^{\mathcal{A},\mathcal{P}}(\vec{c})), nd(S^{\mathcal{A},\mathcal{P}}(\vec{c}))\}.
\end{aligned}$$

Since by our assumption $\hat{I} \notin AS(\Pi_1)$, one of the following must hold:

(i) $\hat{I} \not\models (\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})_{gl}^{\hat{I}}$ or

(ii) some $\hat{J} \subset \hat{I}$ exists, such that $\hat{J} \models (\hat{\Pi} \cup \mathcal{R} \cup facts(\mathcal{A}) \cup \mathcal{COMP})_{gl}^{\hat{I}}$.

First assume that (i) is true. By construction of $\hat{I}$, it satisfies $\hat{\Pi}$ and all rules of the forms (r_1^*)-(r_4^*). Moreover, constraints of the form (r_5^*) can not be violated, as no DL-atom $d(\vec{c})$ with $I \not\models^{\mathcal{O}'} d(\vec{c})$ can have a support set that consists only of input assertions. The rules (r_7^*) and (r_8^*) are not present in the reduct ${\Pi_1}_{gl}^{\hat{I}}$, as $\hat{I} \models C_d$ for each DL-atom $d(\vec{c})$.

Thus the rule r from Π_1 such that $\hat{I} \not\models r_{gl}^{\hat{I}}$ could only be of the form (r_6^*) or (r_9^*). In case of form (r_6^*), some DL-atom $d(\vec{c})$ would exist such that $I \not\models^{\mathcal{O}'} d(\vec{c})$. By Proposition 15 no support set for $d(\vec{c})$ would exist that is coherent with I, and by construction $S_d^{\mathcal{A},\mathcal{P}}(\vec{c}) \notin \hat{I}$. Hence, r must be of the form (r_9^*); however, as $I \models^{\mathcal{O}'} d(\vec{c})$ by completeness of $\mathbf{S}_d$ and Proposition 15, by construction we have $Sup_d(\vec{c}) \in \hat{I}$, which implies that r can not be violated.

Now let (ii) hold, i.e. some $\hat{J} \subset \hat{I}$ exists s.t. $\hat{J} \models {\Pi_1}_{gl}^{\hat{I}}$. As $\hat{J}$ contains for each DL-atom $d(\vec{c})$ exactly one out of $e_d(\vec{c})$ and $ne_d(\vec{c})$ and Π_1 contains $e_d(\vec{c}) \vee ne_d(\vec{c})$, the interpretations $\hat{J}$ and $\hat{I}$ coincide on all replacement atoms $e_d(\vec{c})$ and $ne_d(\vec{c})$. Suppose that $\hat{I} \setminus \hat{J}$ contains some atoms from the language of Π. Then $\hat{J}|_\Pi \not\models \mathcal{P}_{weak}^{I,\mathcal{O}'}$; hence some rule $r_{weak}^{I,\mathcal{O}'} \in \mathcal{P}_{weak}^{I,\mathcal{O}'}$, exists such that $\hat{J}|_\Pi \models B(r_{weak}^{I,\mathcal{O}'})$, but $\hat{J}|_\Pi \not\models H(r_{weak}^{I,\mathcal{O}'})$. Consider the respective rule $r_{gl}^{\hat{I}}$ in $\hat{\Pi}_{gl}^{\hat{I}}$. As $\hat{J} \not\models H(r_{gl}^{\hat{J}})$, we must have $\hat{J} \not\models B(r_{gl}^{\hat{I}})$. By construction of the weak and GL reduct, respectively, the positive normal atoms in $B(r_{gl}^{\hat{J}})$ and in $B(r_{weak}^{\hat{I},\mathcal{O}'})$ are the same. Hence, some replacement atom $e_d(\vec{c})$ (resp. $ne_d(\vec{c})$) must occur positively in $B(r_{gl}^{\hat{I}})$, such that $e_d(\vec{c}) \in \hat{I} \setminus \hat{J}$ (resp. $ne_d(\vec{c}) \in \hat{I} \setminus \hat{J}$). As we have already argued, the latter is not possible, leading to a contradiction.

Consequently, $\hat{I} \setminus \hat{J}$ must contain only atoms from the language of $\mathcal{R}$. For every rule $r_{gl}^{\hat{I}}$ of form (r_3^*) or (r_4^*) we have $\hat{J} \models B(r_{gl}^{\hat{I}})$ iff $\hat{I} \models B(r_{gl}^{\hat{I}})$, thus $\hat{I}$ and $\hat{J}$ agree on all atoms $S_d^{\mathcal{P}}(\vec{c})$ and $S_d^{\mathcal{A},\mathcal{P}}(\vec{c})$. Similarly, via (r_1^*) and (r_2^*) we must have that $\hat{I}$ and $\hat{J}$ agree on all atoms $Sup_d(\vec{c})$. Finally, the same holds for all $p_P(\vec{c})$ and $\bar{p}_P(\vec{c})$ by the rules (r_6^*) and the construction of $\hat{I}$. In conclusion, $\hat{J} = \hat{I}$ holds, which violates (ii).

Thus, it follows that $\hat{I} \in AS(\Pi_1)$. Consequently, $AS(\Pi_1) \supseteq RAS_{weak}(\Pi_1)$ holds; this proves the result.

References

Alchourrón, C. E., Gärdenfors, P., & Makinson, D. (1985). On the logic of theory change: Partial meet contraction and revision functions. *J. Symbolic Logic*, *50*(2), 510–530.

Aranguren, M. E., Bechhofer, S., Lord, P. W., Sattler, U., & Stevens, R. D. (2007). Understanding and using the meaning of statements in a bio-ontology: recasting the gene ontology in OWL. *BMC Bioinformatics*, *8*(1), 1–13.

Ausiello, G., D'Atri, A., & Saccà, D. (1983). Graph algorithms for functional dependency manipulation. *J. of the ACM*, *30*(4), 752–766.

Ausiello, G., D'Atri, A., & Saccà, D. (1986). Minimal representation of directed hypergraphs. *SIAM J. on Computing*, *15*(2), 418–431.

Baader, F., Bauer, A., & Lippmann, M. (2009). Runtime verification using a temporal description logic. In *Proc. 7th Int'l Symp. on Frontiers of Combining Systems, FroCoS 2009*, pp. 149–164.

Baader, F., Brandt, S., & Lutz, C. (2005). Pushing the EL envelope. In *Proc. 19th Int'l Joint Conf. on Artificial Intelligence, IJCAI 2005*, pp. 364–369.

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, P. F. (Eds.). (2003). *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press, 2003.

Baader, F., Lutz, C., Milicic, M., Sattler, U., & Wolter, F. (2005). Integrating description logics and action formalisms: First results. In *Proc. 20th National Conf. Artificial Intelligence and 17th Conf. Innovative Applications of Artificial Intelligence*, pp. 572–577.

Balduccini, M., & Gelfond, M. (2003). Logic programs with consistency-restoring rules. In *Int'l Symp. Logical Formalization of Commonsense Reasoning, AAAI 2003 Spring Symposium Series*, pp. 9–18.

Belnap, N. (1977). A useful four-valued logic. In *Modern Uses of Multiple-Valued Logic*, pp. 7–37. Reidel Publishing Company, Boston.

Bertossi, L. E. (2011). *Database Repairing and Consistent Query Answering*. Morgan & Claypool Publishers, Ottawa, Canada.

Bertossi, L. E., Hunter, A., & Schaub, T. (2005). Introduction to inconsistency tolerance. In *Inconsistency Tolerance [result from a Dagstuhl seminar]*, pp. 1–14.

Bienvenu, M. (2008). Complexity of abduction in the $\mathcal{EL}$ family of lightweight description logics. In *Proc. 11th Int'l Conf. on Principles of Knowledge Representation and Reasoning, KR 2008*, pp. 220–230.

Bienvenu, M., & Rosati, R. (2013). New inconsistency-tolerant semantics for robust ontology-based data access. In *Proc. 26th Int'l Workshop on Description Logics*, pp. 53–64.

Bonatti, P. A., Faella, M., & Sauro, L. (2010). $\mathcal{EL}$ with default attributes and overriding. In *Proceedings of the 9th Int'l Semantic Web Conf., ISWC 2010*, pp. 64–79.

Brewka, G. (1989). Preferred subtheories: An extended logical framework for default reasoning. In *Proc. 11th Int'l Joint Conf. on Artificial Intelligence, IJCAI 1989*, pp. 1043–1048.

Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., & Rosati, R. (2007a). Ontology-based database access. In *Proc. 15th Italian Symposium on Advanced Database Systems, SEBD 2007*, pp. 324–331.

Calvanese, D., De Giacomo, G., Lenzerini, M., Lembo, D., Poggi, A., & Rosati, R. (2007b). MASTRO-I: efficient integration of relational data through DL ontologies. In *Proc. 20th Int'l Workshop on Description Logics*.

Calvanese, D., Ortiz, M., Simkus, M., & Stefanoni, G. (2013). Reasoning about explanations for negative query answers in DL-Lite. *J. Artificial Intelligence Research*, *48*, 635–669.

Console, L., Sapino, M. L., & Dupré, D. T. (1995). The role of abduction in database view updating. *J. of Intelligent Information Systems*, *4*(3), 261–280.

Console, M., Mora, J., Rosati, R., Santarelli, V., & Savo, D. F. (2014). Effective computation of maximal sound approximations of description logic ontologies. In *Proc. 13th Int'l Semantic Web Conf., ISWC 2014, Part II*, pp. 164–179.

dlliteplugin of DLVHEX system (2015). `https://github.com/hexhex/dlliteplugin`.

Scripts for DL-program benchmark generation (2015). `https://github.com/hexhex/dlliteplugin/benchmarks`.

dlplugin of the DLVHEX system (2015). `https://github.com/hexhex/dlplugin`.

DREW reasoner for DL-Programs over Datalog-rewritable Description Logics (2012). `http://www.kr.tuwien.ac.at/research/systems/drew/`.

Ecke, A., Ludwig, M., & Walther, D. (2013). The concept difference for $\mathcal{EL}$-terminologies using hypergraphs. In *Proc. Int'l Workshop on Document Changes: Modeling, Detection, Storage and Visualization*.

Eiter, T., Erdem, E., Fink, M., & Senko, J. (2005). Updating action domain descriptions. In *Proc. 19th Int'l Joint Conf. on Artificial Intelligence, IJCAI 2005*, pp. 418–423.

Eiter, T., Fink, M., Greco, G., & Lembo, D. (2008). Repair localization for query answering from inconsistent databases. *ACM Transactions on Database Systems*, *33*(2).

Eiter, T., Fink, M., Krennwallner, T., Redl, C., & Schüller, P. (2014a). Efficient HEX-program evaluation based on unfounded sets. *J. Artificial Intelligence Research*, *49*, 269–321.

Eiter, T., Fink, M., Redl, C., & Stepanova, D. (2014b). Exploiting support sets for answer set programs with external evaluations. In *Proc. 28th Conf. Artificial Intelligence, AAAI 2014*, pp. 1041–1048.

Eiter, T., Fink, M., & Stepanova, D. (2013). Data repair of inconsistent DL-programs. In *Proc. 23rd Int'l Joint Conf. on Artificial Intelligence, IJCAI 2013*, pp. 869–876.

Eiter, T., Fink, M., & Stepanova, D. (2014c). Computing repairs for inconsistent DL-programs over $\mathcal{EL}$ ontologies. In *Proc. 14th Joint European Conf. Logics in Artificial Intelligence, JELIA 2014*, pp. 426–441.

Eiter, T., Fink, M., & Stepanova, D. (2014d). Towards practical deletion repair of inconsistent DL-programs. In *Proc. 21st European Conf. Artificial Intelligence, ECAI 2014*, pp. 285–290.

Eiter, T., Fink, M., & Stepanova, D. (2014d). Data repair of inconsistent DL-programs. Tech. rep. INFSYS RR-1843-15-03, Institut f. Informationssysteme, TU Wien, A-1040 Vienna, Austria.

Eiter, T., Gottlob, G., & Leone, N. (1997). Abduction from logic programs: Semantics and complexity. *Theoretical Computer Science*, *189*(1-2), 129–177.

Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., & Tompits, H. (2008). Combining answer set programming with description logics for the Semantic Web. *J. Artificial Intelligence*, *172*(12-13), 1495–1539.

Eiter, T., Ianni, G., Schindlauer, R., & Tompits, H. (2005). A uniform integration of higher-order reasoning and external evaluations in answer-set programming. In *Proc. 19th Int'l Joint Conf. on Artificial Intelligence, IJCAI 2005*, pp. 90–96.

Eiter, T., & Makino, K. (2007). On computing all abductive explanations from a propositional Horn theory. *J. of the ACM*, *54*(5).

Eiter, T., Schneider, P., Šimkus, M., & Xiao, G. (2014). Using OpenStreetMap data to create benchmarks for description logic reasoners. In *Proc. 2nd Int'l Workshop on OWL Reasoner Evaluation, ORE 2014*, Vol. 1207, pp. 51–57.

Experimental data with inconsistent DL-programs (2015). `http://www.kr.tuwien.ac.at/staff/dasha/jair_el/benchmark_instances.zip`.

Feng, S., Ludwig, M., & Walther, D. (2015). The logical difference for $\mathcal{EL}$: From terminologies towards tboxes. In *Proc. 1st Int'l Workshop on Sem. Technologies, IWOST 2015*, pp. 31–41.

Fink, M. (2012). Paraconsistent hybrid theories. In *Proc. 13th Int'l Conf. on Principles of Knowledge Representation and Reasoning, KR 2012*, pp. 141–151.

Frühstück, M., Pührer, J., & Friedrich, G. (2013). Debugging answer-set programs with ouroboros - extending the sealion plugin. In *Proc. 12th Int'l Conf. Logic Programming and Nonmonotonic Reasoning, LPNMR 2013*, pp. 323–328.

Gallo, G., Longo, G., & Pallottino, S. (1993). Directed hypergraphs and applications. *Discrete Applied Mathematics*, *42*(2), 177–201.

Gärdenfors, P., & Rott, H. (1995). Belief revision. *Handbook of Logic in Artificial Intelligence and Logic Programming*, *4*, 35–132.

Gardiner, T., Tsarkov, D., & Horrocks, I. (2006). Framework for an automated comparison of description logic reasoners. In *Proc. 5th Int'l Semantic Web Conf., ISWC 2006*, pp. 654–667.

Gebser, M., Pührer, J., Schaub, T., & Tompits, H. (2008). A meta-programming technique for debugging answer-set programs. In *Proc. 23rd Conf. Artificial Intelligence, AAAI 2008*, pp. 448–453.

Gelfond, M., & Lifschitz, V. (1991). Classical negation in logic programs and disjunctive databases. *New Generation Computing*, *9*, 365–385.

Gottlob, G., Pichler, R., & Wei, F. (2007). Efficient datalog abduction through bounded treewidth. In *Proc. 22nd Int'l Conf. on Artificial Intelligence, AAAI 2007*, pp. 1626–1631.

Grau, B. C., Horrocks, I., Kazakov, Y., & Sattler, U. (2007). Just the right amount: extracting modules from ontologies. In *Proc. 16th Int'l Conf. World Wide Web, WWW 2007*, pp. 717–726.

Hansen, P., Lutz, C., Seylan, I., & Wolter, F. (2014). Query rewriting under $\mathcal{EL}$ TBoxes: Efficient algorithms. In *Proc. 27th Int'l Workshop on Description Logics*, pp. 197–208.

Hermann, M., & Pichler, R. (2010). Counting complexity of propositional abduction. *J. Computer and System Sciences*, *76*(7), 634–649.

HTCondor load distribution system, version 7.8.7 (2012). `http://research.cs.wisc.edu/htcondor/`.

Huang, S., Hao, J., & Luo, D. (2014). Incoherency problems in a combination of description logics and rules. *J. Applied Mathematics*, 604753:1–6.

Huang, S., Li, Q., & Hitzler, P. (2013). Reasoning with inconsistencies in hybrid MKNF knowledge bases. *Logic J. of the IGPL*, *21*(2), 263–290.

Kaminski, T., Knorr, M., & Leite, J. (2015). Efficient paraconsistent reasoning with ontologies and rules. In *Proc. 24th Int'l Joint Conf. on Artificial Intelligence, IJCAI 2015*, pp. 3098–3105.

Knorr, M., Alferes, J. J., & Hitzler, P. (2008). A coherent well-founded model for hybrid MKNF knowledge bases. In *Proc. 18th European Conf. on Artificial Intelligence, ECAI 2008*, pp. 99–103.

Knorr, M., Alferes, J. J., & Hitzler, P. (2011). Local closed world reasoning with description logics under the well-founded semantics. *Artificial Intelligence*, *175*(9-10), 1528–1554.

Konev, B., Ludwig, M., Walther, D., & Wolter, F. (2012). The logical difference for the lightweight description logic $\mathcal{EL}$. *J. Artificial Intelligence Research*, *44*, 633–708.

Kontchakov, R., Lutz, C., Toman, D., Wolter, F., & Zakharyaschev, M. (2010). The combined approach to query answering in DL-Lite. In *Proc. 12th Int'l Conf. on Principles of Knowledge Representation, KR 2010*, pp. 247–257.

Kotek, T., Simkus, M., Veith, H., & Zuleger, F. (2014). Towards a description logic for program analysis: Extending $\mathcal{ALCQIO}$ with reachability. In *Proc. 27th Int'l Workshop on Description Logics*, pp. 591–594.

Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D. F. (2015). Inconsistency-tolerant query answering in ontology-based data access. *J. Web Sem.*, *33*, 3–29.

Lembo, D., Santarelli, V., & Savo, D. F. (2013). A graph-based approach for classifying OWL 2 QL ontologies. In *Proc. 26th Int'l Workshop on Description Logics*, pp. 747–759.

LUBM benchmark (2005). `http://swat.cse.lehigh.edu/projects/lubm/`.

LUBM data generator (2013). `http://code.google.com/p/combo-obda/`.

Ludwig, M., & Walther, D. (2014). The logical difference for $\mathcal{ELH}r$-terminologies using hypergraphs. In *Proc. 21st European Conf. Artifical Intelligence, ECAI 2014*, pp. 555–560.

Lukasiewicz, T. (2010). A novel combination of answer set programming with description logics for the semantic web. *IEEE Trans. Knowledge and Data Engineering*, *22*(11), 1577–1592.

Lutz, C., Toman, D., & Wolter, F. (2009). Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In Boutilier, C. (Ed.), *Proc. 21st Joint Int'l Conf. Artificial Intelligence, IJCAI 2009*, pp. 2070–2075.

Lutz, C., Walther, D., & Wolter, F. (2007). Conservative extensions in expressive description logics. In *Proc. 20th Int'l Joint Conf. Artificial Intelligence, IJCAI 2007*, pp. 453–458.

Lutz, C., & Wolter, F. (2010). Deciding inseparability and conservative extensions in the description logic $\mathcal{EL}$. *J. of Symbolic Computation*, *45*(2), 194–228.

Martinez, M. V., Molinaro, C., Subrahmanian, V. S., & Amgoud, L. (2013). *A General Framework for Reasoning On Inconsistency*. Springer Briefs in Computer Science. Springer, 2013.

Masotti, G., Rosati, R., & Ruzzi, M. (2011). Practical abox cleaning in *DL-Lite* (progress report). In *Proc. of Description Logics Workshop*.

Motik, B., & Rosati, R. (2010). Reconciling Description Logics and Rules. *J. of the ACM*, *57*(5), 1–62.

MyITS - Personalized Intelligent Mobility Service (2012). `http://www.kr.tuwien.ac.at/research/projects/myits/GeoConceptsMyITS-v0.9-Lite.owl/`.

Nguyen, N. T. (2008). *Advanced Methods for Inconsistent Knowledge Management*. Advanced Information and Knowledge Processing. Springer.

Nortje, R., Britz, A., & Meyer, T. (2013). Module-theoretic properties of reachability modules for $\mathcal{SRIQ}$. In *Proc. 26th Int'l Workshop on Description Logics*, pp. 868–884.

Oetsch, J., Pührer, J., & Tompits, H. (2012). Stepwise debugging of description-logic programs. In *J. of Correct Reasoning*, pp. 492–508.

Open Street Map project (2012). `http://www.openstreetmap.org/`.

Özccep, Ö. L., & Möller, R. (2012). Combining $DL-Lite$ with spatial calculi for feasible geothematic query answering. In *Proc. 25th Int'l Workshop on Description Logics*.

Pan, J. Z., & Thomas, E. (2007). Approximating OWL-DL ontologies. In *Proc. 22nd Int'l Conf. on Artificial Intelligence, AAAI 2007*, pp. 1434–1439.

Pérez-Urbina, H., Motik, B., & Horrocks, I. (2010). Tractable query answering and rewriting under description logic constraints. *J. of Applied Logic*, *8*(2), 186–209.

Pührer, J., Heymans, S., & Eiter, T. (2010). Dealing with inconsistency when combining ontologies and rules using DL-programs . In *Proc. 7th Extended Semantic Web Conf., ESWC 2010, part I*, pp. 183–197.

RACERPRO reasoner for OWL ontologies (2007). `http://franz.com/agraph/racer/`.

Reiter, R. (1987). A theory of diagnosis from first principles. *J. Artificial Intelligence*, *32*(1), 57–95.

Rosati, R. (2007). On conjunctive query answering in $\mathcal{EL}$. In *proceedings of the 20th Int'l Workshop on Description Logics*.

Rosati, R., Ruzzi, M., Graziosi, M., & Masotti, G. (2012). Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies. In *Proc. 11th Int'l Semantic Web Conf., ISWC 2012*, pp. 337–349.

Sakama, C., & Inoue, K. (1995). Paraconsistent stable semantics for extended disjunctive programs. *J. of Logic and Computation*, *5*(3), 265–285.

Sakama, C., & Inoue, K. (2003). An abductive framework for computing knowledge base updates. *Theory and Practice of Logic Programming*, *3*(6), 671–713.

Schulz, C., Satoh, K., & Toni, F. (2015). Characterising and explaining inconsistency in logic programs. In *Proc. 13th Int'l Conf., LPNMR 2015*, pp. 467–479.

Schulz, S., Cornet, R., & Spackman, K. A. (2011). Consolidating SNOMED CT's ontological commitment. *Applied Ontology*, *6*(1), 1–11.

Shen, Y.-D. (2011). Well-supported semantics for description logic programs. In *Proc. 22nd Int'l Joint Conf. on Artificial Intelligence, IJCAI 2011*, pp. 1081–1086.

Stefanoni, G., Motik, B., & Horrocks, I. (2012). Small datalog query rewritings for $\mathcal{EL}$. In *Proc. 25th Int'l Workshop on Description Logics*.

Stepanova, D. (2015). *Inconsistencies in Hybrid Knowledge Bases*. PhD thesis, Vienna University of Technology.

Steve, G., Gangemi, A., & Mori, A. R. (1995). Modelling a sharable medical concept system: Ontological foundation in galen. In *AIME*, pp. 411–412.

Stuckenschmidt, H., Parent, C., & Spaccapietra, S. (Eds.). (2009). *Modular Ontologies: Concepts, Theories and Techniques for Knowledge Modularization*, Vol. 5445 of *Lecture Notes in Computer Science*. Springer.

Syrjänen, T. (2006). Debugging Inconsistent Answer-Set Programs. In *Proc. 11th Int'l Workshop on Nonmonotonic Reasoning, NMR 2006*, pp. 77–83.

Thakur, M., & Tripathi, R. (2009). Linear connectivity problems in directed hypergraphs. *Theoretical Computer Science*, *410*(27-29), 2592–2618.

Tserendorj, T., Rudolph, S., Krötzsch, M., & Hitzler, P. (2008). Approximate OWL-reasoning with Screech. In *Proc. 2nd Int'l Conf. Web Reasoning and Rule Systems, RR 2008*, pp. 165–180.

Wache, H., Groot, P., & Stuckenschmidt, H. (2005). Scalable instance retrieval for the semantic web by approximation. In *Proc. 1st Int'l Workshops on Web Information Systems Engineering, WISE 2005*, pp. 245–254.

Wang, Y., You, J.-H., Yuan, L.-Y., & Shen, Y.-D. (2010). Loop formulas for description logic programs. *Theory and Practice of Logic Programming*, *10*(4-6), 531–545.

Xiao, G. (2014). *Inline Evaluation of Hybrid Knowledge Bases*. Ph.D. thesis, Vienna University of Technology, Austria.

Zhao, Y., Pan, J. Z., & Ren, Y. (2009). Implementing and evaluating a rule-based approach to querying regular $\mathcal{EL}$+ ontologies. In *Proc. 9th Int'l Conf. Hybrid Intelligent Systems, HIS 2009*, pp. 493–498.