Large-scale Ontological Reasoning via Datalog

Mario Alviano[0000−0002−2052−2063] and Marco Manna[0000−0003−3323−9328]

University of Calabria, 87036 Rende, Italy
{alviano,manna}@mat.unical.it

Abstract. Reasoning over OWL 2 is a very expensive task in general, and therefore the W3C identified tractable profiles exhibiting good computational properties. Ontological reasoning for many fragments of OWL 2 can be reduced to the evaluation of Datalog queries. This paper surveys some of these compilations, and in particular the one addressing queries over Horn-$\mathcal{SHIQ}$ knowledge bases and its implementation in DLV2 enanched by a new version of the Magic Sets algorithm.

Keywords: Datalog · Ontology Reasoning · Query Answering.

1 Introduction

Datalog is a rule-based language originally designed in the context of *deductive databases*, a field that has benefited from the cross-fertilization between logic programming and database theory [1,26,27,42]. The language is nowadays successfully applied in several contexts, spanning from Boolean optimization and constraint satisfaction problems [15] to ontological design and reasoning in the Semantic Web [20,33]. Specifically, standard reasoning tasks in the Semantic Web are often reduced to query evaluation over (deductive) databases, so to satisfy the fundamental prerequisite of efficient large-scale reasoning.

This paper focuses on the use of Datalog in the context of Semantic Web, in particular on its application to *ontology-based query answering*, for short OBQA [19,45]. In OBQA, a Boolean query q has to be evaluated against a *logical theory* (a.k.a. *knowledge base*, or KB) consisting of an extensional *database* (a.k.a. ABox) D paired with an *ontology* (a.k.a. TBox) Σ. The problem is usually stated as $D \cup \Sigma \models q$, and is equivalent to checking whether q is satisfied by all models of $D \cup \Sigma$ according to the classical *open-world assumption* (OWA) of first-order logic [1]. Several fields of Computer Science have shown interest in OBQA, from Artificial Intelligence [8,23,30] to Database Theory [14,16,31] and Logic [11,32,46]. From these fields, two families of formal knowledge representation languages to specify Σ emerged, namely Description Logics (DLs) [7] and Datalog$^{\pm}$ [19]. For both of them, OBQA is undecidable in general [18,36,49], and therefore syntactic decidable fragments have been singled out with the aim of offering a good balance between computational complexity and expressiveness. The same idea lead to the definition of *OWL 2 Web Ontology Language Profiles*.

In fact, reasoning over OWL 2 is generally a very expensive task: fact entailment (i.e., checking whether an individual is an instance of a concept) is

already 2NExpTime-hard, while decidability of conjunctive query answering is still an open problem. To balance expressiveness and scalability, the W3C identified three tractable profiles, namely OWL 2 EL, OWL 2 QL, and OWL 2 RL, exhibiting good computational properties: the evaluation of conjunctive queries over KBs falling in these fragments is in PTime in data complexity (that is, when query and TBox are fixed) and in PSpace in combined complexity (that is, in the general case) [54]. Horn-$\mathcal{SHIQ}$ is another fragment of OWL 2 exhibiting good computational properties and high expressivity: conjunctive queries are evaluated in PTime in data complexity, and in ExpTime in combined complexity; it generalizes both OWL 2 QL and OWL 2 RL, and captures all OWL 2 EL constructs but role chain [41].

From a theoretical viewpoint much has been done: OBQA has been addressed in many ontological settings by reductions to the evaluation of Datalog queries [24,28,39,53,57]. Some of these rewriting are briefly mentioned in Section 3.1. From a pragmatic viewpoint, reasoning services have been developed, among them MASTRO [22], ONTOP [21], and RDFOX [44]. Concerning Horn-$\mathcal{SHIQ}$, Eiter et al. [28] showed that OBQA can be addressed by means of Datalog queries: in a nutshell, given a Horn-$\mathcal{SHIQ}$ TBox paired with a SPARQL query [56], it is possible to construct an equivalent Datalog query independently from the ABox. This idea has been also implemented in a specific branch of DLV2 [3], with promising results. The rewriting implemented by DLV2 is further processed by a new version of the *magic sets algorithm* [5], which inhibits the creation of new recursive dependencies and partially unroll the magic sets rewriting if binding information are lost. The inhibition of new recursive definitions is important because magic sets were originally introduced for Datalog programs [10], and their extension to programs with stratified negation was nontrivial; indeed, the perfect model semantics [48] is not applicable to the rewritten program if recursive negation is introduced by magic sets, and several semantics were considered in the literature to overcome this limitation [9,13,37,38,52]. By inhibiting the creation of new recursive dependencies, the semantic issue disappears. The technique is briefly explained in Section 3.2.

2 Background

2.1 Basics

Fix three pairwise disjoint discrete sets C, P and V, respectively of *constants*, *predicate symbols* and *variables*. Constants and variables all together form what we call *terms*. Each predicate symbol p has an arity, consisting of a non-negative integer $arity(p)$. An *atom* is an expression α of the form $p(\mathbf{t})$, where p is a predicate symbol, $\mathbf{t} = t_1, ..., t_m$ is a sequence of terms possibly with repetitions, m is the arity of p, $\alpha[i] = t_i$ for each $i \in [1..m]$, and the set $\{t_1, ..., t_m\}$ is denoted by $dom(\alpha)$. By definition, $arity(\alpha) = arity(p)$. An instance I is any set of atoms over constants.

2.2   Description Logics and OWL

Description Logics (DLs) are a family of formal knowledge representation languages that model concepts, roles, individuals, and their relationships. Let N_C (*concepts*), N_R (*roles*) and N_I (*individuals*) be mutually disjoint discrete sets. Hereinafter, we assume that $N_C \cup N_R \subset \mathsf{P}$ and that $N_I \subset \mathsf{C}$. Accordingly, concepts are basically unary predicates whereas roles are binary predicates. Moreover, in this context, concepts and roles are denoted by uppercase letters. A DL *knowledge base* (KB) in normal form is any pair $\mathcal{K} = (\mathcal{A}, \mathcal{T})$ where:

(*i*) $\mathcal{A}$, the ABox (assertional box), is a finite set of *assertions* (i.e., atoms) of the form $A(a)$ or $R(a, b)$, with $a, b \in N_I$, $A \in N_C$, and $R \in N_R$. Roughly, an ABox can be transparently seen as a database (i.e., a finite instance).

(*ii*) $\mathcal{T}$, the TBox (terminological box), is a finite set of *concept inclusions* (CIs) together with a finite set of *role inclusions* (RIs). Table 1 and Table 2 report only those inclusions that are at the basis of the OWL 2 Web Ontology Language Profiles introduced below. Accordingly, we consider the following classes of Description Logics: $\mathcal{EL}{++}$ [6], Horn-$\mathcal{SHIQ}$ [35], $\mathcal{ELH}$ [17], DL-Lite$_\mathsf{R}$ [50], and DLP [34]. The semantics of concept (resp., role) inclusions is given in Table 1 (resp., 2) in terms of first-order expressions [7].

The OWL 2 Web Ontology Language, informally OWL 2, is an ontology language for the Semantic Web with formally defined meaning. OWL 2 ontologies are stored as Semantic Web documents and provide classes, properties, individuals, and data values. The most expressive OWL 2 profile is called OWL 2 DL.

Table 1. Concept inclusions, where $A, B, B_1, B_2 \in N_C$ and $R \in N_R$. In the last column, all occurrences of the variables x, y, y_1 and y_2 are intended to be universally quantified.

$\mathcal{EL}{++}$	Horn-$\mathcal{SHIQ}$	$\mathcal{ELH}$	DL-Lite$_\mathsf{R}$	DLP	concept inclusions	Equivalent first-order expression
✓	✓	✓	✓	✓	$B \sqsubseteq A$	$B(x) \rightarrow A(x)$
✓	✓	✓		✓	$B_1 \sqcap B_2 \sqsubseteq A$	$B_1(x), B_2(x) \rightarrow A(x)$
	✓			✓	$B \sqsubseteq \forall R.A$ $\exists R^-.B \sqsubseteq A$	$B(x), R(x, y) \rightarrow A(y)$
✓	✓	✓		✓	$\exists R.B \sqsubseteq A$	$R(x, y), B(y) \rightarrow A(x)$
✓	✓	✓	✓	✓	$\exists R.\top \sqsubseteq A$ $\mathsf{dom}(R) \sqsubseteq A$	$R(x, y) \rightarrow A(x)$
✓	✓		✓	✓	$\mathsf{ran}(R) \sqsubseteq A$	$R(x, y) \rightarrow A(y)$
✓	✓	✓	✓		$B \sqsubseteq \exists R.A$	$B(x) \rightarrow \exists z R(x, z), A(x)$
	✓		✓	✓	$B \sqsubseteq \neg A$	$B(x), A(x) \rightarrow \bot$
	✓			✓	$B \sqsubseteq\, \leqslant 1\, R.A$	$B(x), R(x, y_1), R(x, y_2),$ $A(y_1), A(y_2), y_1 \neq y_2 \rightarrow \bot$

Reasoning over OWL 2 DL is a very expensive task, in general. To balance expressiveness and scalability, the World Wide Web Consortium (W3C, for short)[1] identified also the following profiles:[2] OWL 2 EL, OWL 2 QL, and OWL 2 RL, each exhibiting better computational properties. Moreover, we point out that $\mathcal{EL}++$ is the logic underpinning OWL 2 EL, DL-Lite$_R$ is the logic underpinning OWL 2 QL, and DLP is the logic underpinning OWL 2 RL. Among these three profiles, OWL 2 RL is the only one that does not admit the usage of existential quantification in superclass expressions in the right-hand side of concept inclusions (i.e., $B \sqsubseteq \exists R.A$ in DL notation).

2.3   Ontology-based query answering

In this section we formally define *ontology-based query answering*, one of the most important ontological reasoning service needed in the development of the Semantic Web.

A *conjunctive query* is any first-order expression of the form $q(\bar{x}) \equiv \exists \bar{y}\, \phi(\bar{x}, \bar{y})$, where ϕ is a conjunction of atoms over the variables $\bar{x} \cup \bar{y}$, possibly with constants.

A *model* of a KB $\mathcal{K} = (\mathcal{A}, \mathcal{T})$ is any instance $I \supseteq \mathcal{A}$ satisfying all the axioms of $\mathcal{T}$, written $I \models \mathcal{T}$, where CIs and RIs, as said, can be regarded as first-order expressions.

The set of all models of $\mathcal{K}$ is denoted by $\mathsf{mods}(\mathcal{K})$. To comply with the so-called *open world assumption* (OWA), note that I might contain individuals that do not occur in $\mathcal{K}$. The *answers* to a query $q(\bar{x})$ over an instance I is the set $q(I) = \{\bar{a} \in N_I^{|\bar{x}|} \mid I \models q(\bar{a})\}$ of $|\bar{x}|$-tuples of individuals obtained by evaluating q over I. Accordingly, the *certain answers* to q under OWA is the set $\mathsf{cert}(\mathcal{K}, q) = \bigcap_{I \in \mathsf{mods}(D, \Sigma)} q(I)$. Finally, ontology-based query answering (OBQA) is the problem of computing $\mathsf{cert}(\mathcal{K}, q)$.

---

[1] See `https://www.w3.org/`

[2] See `http://www.w3.org/TR/owl2-profiles/`

Table 2. Role inclusions, where $R, S, P \in N_R$. In the last column, all occurrences of the variables x, y and z are intended to be universally quantified.

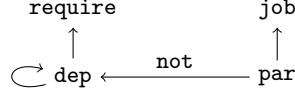| $\mathcal{EL}++$	Horn-$\mathcal{SHIQ}$	$\mathcal{ELH}$	DL-Lite$_R$	DLP	rule inclusions	Equivalent first-order expression
✓	✓	✓	✓	✓	$S \sqsubseteq R$	$R(x, y) \rightarrow S(x, y)$
	✓		✓	✓	$S^- \sqsubseteq R$	$S(x, y) \rightarrow R(y, x)$
✓	✓			✓	$R^+ \sqsubseteq R$	$R(x, y), R(y, z) \rightarrow R(x, z)$
✓					$S \circ P \sqsubseteq R$	$S(x, y), P(y, z) \rightarrow R(x, z)$
	✓		✓	✓	$S \sqsubseteq \neg R$	$S(x, y), R(x, y) \rightarrow \bot$

Fig. 1. Dependency graph of program P_{job} from Example 1.

2.4 Datalog

A Datalog program P is a finite set of rules of the form

$$\alpha_0 :- \alpha_1, \ldots, \alpha_m, \texttt{not } \alpha_{m+1}, \ldots, \texttt{not } \alpha_n. \tag{1}$$

where $n \geq m \geq 0$, and each α_i is an atom; atoms $\alpha_1, \ldots, \alpha_m$ are also called *positive literals*, while $\texttt{not } \alpha_{m+1}, \ldots, \texttt{not } \alpha_n$ are also called *negative literals*. A predicate p occurring in P is said *extensional* if all rules of P with p in their heads are facts; otherwise, p is said *intentional*. For any *expression* (atom, literal, rule, program) E, let $At(E)$ denote the set of atoms occurring in E.

For a rule r of the form (1), define $H(r) := \alpha$, the *head* of r; $B(r) := \{\alpha_1, \ldots, \alpha_m, \texttt{not } \alpha_{m+1}, \ldots, \texttt{not } \alpha_n\}$, the *body* of r; $B^+(r) := \{\alpha_1, \ldots, \alpha_m\}$; and $B^-(r) := \{\alpha_{m+1}, \ldots, \alpha_n\}$. Intuitively, $B(r)$ is interpreted as a conjunction, and we will use $\alpha :- S \wedge S'$ to denote a rule r with $H(r) = \alpha$ and $B(r) = S \cup S'$; abusing of notation, we also permit S and S' to be literals. If $B(r)$ is empty, the symbol $:-$ is usually omitted, and the rule is called a *fact*.

A rule r is *safe* if every variable occurring in r also occurs in $B^+(r)$. In the following, only safe rules are considered, and programs are required to satisfy *stratification of negation*, defined next. The *dependency graph* $\mathcal{G}_P$ of a program P has nodes for each predicate occurring in P, and an arc from p to p' if there is a rule r of P such that p occurs in $H(r)$, and p' occurs in $B(r)$; the arc is marked with $\texttt{not}$ if p' occurs in $B^-(r)$. P satisfies stratification of negation if $\mathcal{G}_P$ has no cycle involving marked arcs.

Semantics. A *substitution* σ is a mapping from variables to constants; for an expression E, let $E\sigma$ be the expression obtained from E by replacing each variable X by $\sigma(X)$. Let $C_1, \ldots, C_n$ (for some $n \geq 1$) be the *strongly connected components* (SCCs) of $\mathcal{G}_P$, sorted so that for all $1 \leq i < j \leq n$, for all $p \in C_i$ and for all $p' \in C_j$, there is no path from p to p' in $\mathcal{G}_P$. Let $heads(P, C_i)$ denote the set of rules of P whose head predicates belong to C_i. The *immediate logical consequence operator* of P at stage i, denoted T_P^i, is defined as

$$T_P^i(I) := \{H(r)\sigma \mid r \in heads(P, C_i), B^+(r)\sigma \subseteq I, B^-(r)\sigma \cap I = \emptyset\} \tag{2}$$

for $i = 1..n$ and any interpretation I. Let $I_0 := \emptyset$, and $I_i := T_P^i \Uparrow I_{i-1}$, for $i = 1, \ldots, n$. The semantics of P is defined as the interpretation I_n, in the following denoted $TP(P)$.

Example 1. Consider a set of jobs to be executed. Some jobs can be started only after some other jobs terminate. Pairs of jobs that can be potentially run in parallel are identified by the following program P_{job}:

```
dep(X,Y) :- require(X,Y).
dep(X,Y) :- require(X,Z), dep(Z,Y).
par(X,Y) :- job(X), job(Y), not dep(X,Y), not dep(Y,X).
```

The dependency graph $\mathcal{G}_{P_{job}}$ is shown in Figure 1. A possible order for the predicates of P_{job} is job, require, dep, parallel. Given the database D_{job}

```
job(a). job(b).           require(a,b).
job(c). job(d). job(e).   require(c,d). require(d,e).
```

the interpretation $TP(D \cup P_{job})$ extends D (stage 1 and 2) with

```
dep(a,b). dep(c,d). dep(d,e). (stage 3, application 1)
dep(c,e). (stage 3, application 2)
par(a,c). par(a,d). par(a,e). par(b,c). par(b,d). par(b,e).
par(c,a). par(d,a). par(e,a). par(c,b). par(d,b). par(e,b).
```

where the last two lines are obtained at stage 4, with one application of $T_{P_{job}}^4$. ∎

3   Ontology Reasoning via Datalog

3.1   Datalog Compilations

For the main DL fragments described in the previous sections, OBDA can be performed via rewriting the knowledge base and the query into a Datalog program (a database and a set of Datalog rules possibly including stratified negation or negative constraints) where a special *output predicate* collects all the answers.

More specifically, from a knowledge base $\mathcal{K} = (\mathcal{A}, \mathcal{T})$ and a conjunctive query $q(\bar{x})$, the general approach is to rewrite: (*i*) both $\mathcal{A}$ and $\mathcal{T}$ into a database D; and (*ii*) both $\mathcal{T}$ and $q(\bar{x})$ into a Datalog program P with an output predicate goal of arity $|\bar{x}|$ in such a way that, $\mathsf{cert}(\mathcal{K}, q) = \{\bar{a} \mid \mathsf{goal}(\bar{a}) \in TP(D \cup P)\}$. (Note that predicate goal never occurs in rule-bodies.)

We can distinguish, however, different cases. In general, this rewriting method goes under the name of *combined approach*; whereas in case D is independent from $\mathcal{T}$ (i.e., D coincides with $\mathcal{A}$), we talk about a *pure approach*. Moreover, in both cases, we can distinguish two sub-cases, namely whether P is an arbitrary Datalog program or it is non-recursive (which is equivalent to a union of conjunctive queries).

For complexity and expressiveness reasons, the pure approach into non-recursive Datalog is only possible for DL-Lite$_R$. Moreover, we also know that the combined approach into non-recursive Datalog is possible for $\mathcal{ELH}$. For all the other fragments, to the best of our knowledge, what we know from the literature is that the rewriting methods (pure or combined) target full Datalog.

In the last decade, multiple rewriting algorithms have been proposed, even for the same DL fragment. They may differ, apart from the rewriting methods, also from the size of the rewritings, the time used to compute them, and their quality measured in terms of time and space needed during the evaluation process over classical benchmarks.

In [28], the authors provide a pure Datalog rewriting for query answering over Horn-$\mathcal{SHIQ}$. In particular, in the generated program P, only the rules having goal in the head depend both on $\mathcal{T}$ and q; conversely, all the other rules of P only depend on $\mathcal{T}$. Program P contains only unary and binary predicates, and all the rules without the goal predicate contain a constant number of variables. However, due to the exponential time complexity of query answering in Horn-$\mathcal{SHIQ}$, the size of P is, in the worst case, exponential in the size of $\mathcal{T}$. Related to this work, there is also a more recent paper [25] that consider Horn-$\mathcal{SHIQ}$ extended with the axiom $S \circ P \sqsubseteq R$ (see Table 2), a.k.a. Horn-$\mathcal{SRIQ}_\sqcap$. However, the authors here provide a pure Datalog rewriting in case of fact entailment, namely when q is a ground atom α of the form $A(a)$ or $R(a,b)$. In this setting, P contains the rule "goal :- α." plus a set of other rules that only depend on $\mathcal{T}$. Since P contains only unary and binary predicates, and since the number of variables in rule bodies is constant, P can be evaluated in polynomial time. However, due to the double exponential time complexity of fact entailment in Horn-$\mathcal{SRIQ}_\sqcap$, the size of P is, in the worst case, also double exponential in the size of $\mathcal{T}$.

In [43], the authors provide a combined Datalog rewriting for a superclass of $\mathcal{ELH}$, called $\mathcal{ELH}_\perp^{dr}$, which includes also $\mathsf{ran}(R) \sqsubseteq A$ and $B \sqsubseteq \neg A$. More precisely, the TBox and the query are rewritten into an a first-order query formula, which in turn can be translated into a non-recursive Datalog program with negation. Both the database D and the program P can be constructed in polynomial time. Moreover, P depends only on the portion of $\mathcal{T}$ containing role inclusions.

Concerning DL-Lite$_\mathsf{R}$, several rewriting algorithms have been proposed. Among these, we recall Presto [51], QuOnto [2] and Requiem [47]. The former, produces a pure Datalog rewriting of polynomial size. Whereas, the latter ones, produce a union of conjunctive queries (i.e., a set of Datalog rules all of which have predicate goal in the head) of exponential size in the worst case.

Finally, concerning DLP, one may observe that a pure Datalog rewriting (including negative constraints) can be directly obtained by taking, for each concept or role inclusion, its equivalent the first-order expression and by adding to P the conjunctive query as a rule having goal in its head. Existing rewriting approach are Orel [40], OwlOntDB [29], RDFox [44] and DReW [58].

3.2   Magic Sets

The magic sets algorithm is a top-down rewriting of the input program P that restricts the range of the object variables so that only the portion of $TP(P)$ that is relevant to answer the query is materialized by a bottom-up evaluation of the rewritten program [10,12,9,55,4]. In a nutshell, magic sets introduce rules defining additional atoms, called *magic atoms*, whose intent is to identify relevant atoms to answer the input query, and these magic atoms are added in the bodies of the original rules to restrict the range of the object variables. The procedure is reported as Algorithm 1. The notions of adornment, magic atom, sideway information passing strategy (SIPS), and a description of the algorithm are given next.

Algorithm 1: $\mathrm{MS}(Q(\mathbf{T})$: a query atom, P: a program)

---

1 Let $\mathbf{s}$ be such that $|\mathbf{s}| = |\mathbf{T}|$, and $\mathbf{s}_i = b$ if $\mathbf{T}_i$ is a constant, and f otherwise, for all $i \in [1..|\mathbf{s}|]$;

2 $P' := \{Q^{\mathbf{s}}(\mathbf{T}).\};$ `// rewritten program: start with the magic seed`

3 $S := \{\langle Q, \mathbf{s}\rangle\};$ `// set of produced adorned predicates`

4 $D := \emptyset;$ `// set of processed (or done) adorned predicates`

5 $G := \mathcal{G}_P \cup \{\langle p, m\#p\rangle \mid p \text{ is a predicate occurring in } P\};$ `// monitor SCCs`

6 **while** $S \neq D$ **do**

7 $\langle q, \mathbf{s}\rangle :=$ any element in $S \setminus D$;   `// select an undone adorned predicate`

8   **foreach** $r \in P$ *such that* $H(r) = q(\mathbf{t})$ *for some list* $\mathbf{t}$ *of terms* **do**

9 $P' := P' \cup \{q(\mathbf{t}) :\!\!- q^{\mathbf{s}}(\mathbf{t}) \wedge B(r).\};$ `// restrict range of variables`

10   Let $(\prec, bnd)$ be the SIPS for r with respect to $\mathbf{s}$;

11   **foreach** $\ell \in B(r)$ *such that* $p(\mathbf{t}') \in At(\ell)$ *and* p *is an intentional predicate of* P **do**

12 $G := G \cup \{\langle m\#p, m\#q\rangle\};$

13 $B := \emptyset;$ `// restrict SIPS to preserve SSCs`

14   **foreach** $\ell' \in B(r)$ *such that* $\ell' \prec \ell$ *and* $p'(\mathbf{t}'') \in At(\ell')$ **do**

15   **if** $\{C \cap At(P) \mid C \in SCCs(G \cup \{\langle m\#p, p'\rangle\})\} = SCCs(\mathcal{G}_P)$ **then**

16 $B := B \cup \{\ell'\}; \quad G := G \cup \{\langle m\#p, p'\rangle\};$

17   Let $\mathbf{s}'$ be such that $|\mathbf{s}'| = |\mathbf{t}'|$, and $\mathbf{s}'_i = b$ if $\mathbf{t}'_i$ is a constant or belongs to $bnd(\ell')$ for some $\ell' \in \{H(r)\} \cup B$ such that $\ell' \prec \ell$, and f otherwise, for all $i \in [1..|\mathbf{s}'|]$;

18 $P' := P' \cup \{p^{\mathbf{s}'}(\mathbf{t}') :\!\!- q^s(\mathbf{t}) \wedge B.\};$ `// add magic rule`

19 $S := S \cup \{\langle p, \mathbf{s}'\rangle\};$ `// keep track of adorned predicates`

20 $D := D \cup \{\langle q, \mathbf{s}\rangle\};$ `// flag the adorned predicate as done`

21 **return** P';

An adornment for a predicate p of arity k is any string $\mathbf{s}$ of length k over the alphabet $\{b, f\}$. The i-th argument of p is *bound* with respect to $\mathbf{s}$ if $\mathbf{s}_i = b$, and *free* otherwise, for all $i \in [1..k]$. For an atom $p(\mathbf{t})$, let $p^{\mathbf{s}}(\mathbf{t})$ be the (magic) atom $m\#p\#\mathbf{s}(\mathbf{t}')$, where $m\#p\#\mathbf{s}$ is a predicate not occurring in the input program, and $\mathbf{t}'$ contains all terms in $\mathbf{t}$ associated with bound arguments according to $\mathbf{s}$.

A SIPS for a rule r with respect to an adornment $\mathbf{s}$ for $H(r)$ is a pair $(\prec, bnd)$, where $\prec$ is a strict partial order over $\{H(r)\} \cup B(r)$, and bnd maps $\ell \in \{H(r)\} \cup B(r)$ to the variables of ℓ that are made bound after processing ℓ. Moreover, a SIPS satisfies the following conditions:

- $H(r) \prec \ell$ for all $\ell \in B(r)$ (binding information originates from head atoms);
- $\ell \prec \ell'$ and $\ell \neq H(r)$ implies that $\ell \in B^+(r)$ (new bindings are created only by positive literals);
- $bnd(H(r))$ contains the variables of $H(r)$ associated with bound arguments according to $\mathbf{s}$;
- $bnd(\ell) = \emptyset$ if ℓ is a negative literal.

Example 2. Consider program P_{job} from Example 1, and suppose we are interested in determining whether jobs `a` and `c` could be run in parallel. The magic atom par^{bb}(`a,b`), that is, `m#par#bb(a,b)`, would represent such interest. Similarly, the interest on all jobs that could be run in parallel to job `a` is represented by the magic atom par^{bf}(`a,Y`), that is, `m#par#bf(a)`. There can be several SIPS $(\prec, bnd)$ for rule

```
dep(X,Y) :- require(X,Z), dep(Z,Y).
```

w.r.t. the adornment *bb*. If body literals are processed left-to-right, and binding information is always passed when possible, then `dep(X,Y)` $\prec$ `require(X,Z)` $\prec$ `dep(Z,Y)`, $bnd(\texttt{dep(X,Y)}) = \{X, Y\}$, $bnd(\texttt{require(X,Z)}) = \{Z\}$ (or $\{X, Z\}$), and $bnd(\texttt{dep(Z,Y)})$ is irrelevant. If instead body literals are processed in parallel, then `dep(X,Y)` $\prec$ `require(X,Z)`, `dep(X,Y)` $\prec$ `dep(Z,Y)`, $bnd(\texttt{dep(X,Y)}) = \{X, Y\}$, and $bnd(\texttt{require(X,Z)})$ and $bnd(\texttt{dep(Z,Y)})$ are irrelevant.   ∎

Algorithm 1 starts by producing the *magic seed*, obtained from the predicate and the constants in the query (lines 1–2). After that, the algorithm processes each produced adorned predicate (lines 6–7): each rule defining the predicate is modified so to restrict the range of the head variables to the tuples that are relevant to answer the query (lines 8–9); such a relevance is encoded by the magic rules, which are produced for all intentional predicates in the bodies of the modified rules (lines 10–18). Note that lines 5 and 12–16 implement a restriction of SIPS guaranteeing that no SCCs of $\mathcal{G}_P$ are merged during the application of magic sets; there, $G \cup E$ denotes the graph obtained from the graph G by adding each arc in the set E, and $SCCs(G)$ is the set of SCCs of G. More in detail, a graph G is initialized with the arcs of $\mathcal{G}_P$ and arcs connecting each predicate p with a *representative magic predicate* $m\#p$ (line 5). After that, before creating a new magic rule, elements of $B(r)$ that would cause a change in the SCCs of G are discarded (lines 13–16). Graph G is updated with new arcs involving original predicates and representative magic predicates, so that it represents a superset of the graph obtained from $\mathcal{G}_{P'}$ by merging all pairs of nodes of the form $m\#p\#\mathbf{s}$, $m\#p\#\mathbf{s}'$.

Example 3. Consider program P_{job} from Example 1, and the query `par(a,c)`. The algorithm first produces the magic seed `m#par#bb(a,b)` and the modified rule

```
par(X,Y) :- m#par#bb(X,Y), job(X), job(Y), not dep(X,Y), not dep(Y,X).
```

Relevance of `dep(a,c)` and `dep(c,a)` is captured by the magic rules

```
m#dep#bb(X,Y) :- m#par#bb(X,Y).
m#dep#bb(Y,X) :- m#par#bb(X,Y).
```

Hence, new modified rules are produced:

```
dep(X,Y) :- m#dep#bb(X,Y), require(X,Y).
dep(X,Y) :- m#dep#bb(X,Y), require(X,Z), dep(Z,Y).
```

At this point, if left-to-right SIPS are used, the magic rule

---

Algorithm 2: FullFree(P: a program obtained by executing magic sets)

---

1 foreach $m\#p\#f\cdots f$ *occurring in P* **do**
2   **foreach** $m\#p\#\mathbf{s}$ *occurring in P such that* $\mathbf{s} \neq f\cdots f$ **do**
3   remove all rules of Π having $m\#p\#\mathbf{s}$ in their bodies;
4   replace $m\#p\#\mathbf{s}(\mathbf{t})$ by $m\#p\#f\cdots f$ in all rule heads of P;

5 return P;

```
m#dep#bb(Z,Y) :- m#dep#bb(X,Y), require(X,Z).
```

is added and the algorithm terminates; the rewritten program is the following:

```
m#par#bb(a,c).
m#dep#bb(X,Y) :- m#par#bb(X,Y).
m#dep#bb(Y,X) :- m#par#bb(X,Y).
m#dep#bb(Z,Y) :- m#dep#bb(X,Y), require(X,Z).

par(X,Y) :- m#par#bb(X,Y), job(X), job(Y), not dep(X,Y), not dep(Y,X).
dep(X,Y) :- m#dep#bb(X,Y), require(X,Y).
dep(X,Y) :- m#dep#bb(X,Y), require(X,Z), dep(Z,Y).
```

Note that using the parallel SIPS from Example 2 would lead to the production of the magic rule

```
m#dep#fb(Y) :- m#dep#bb(X,Y).
```

and therefore to further iterations of the algorithm to process predicate `dep` with respect to the adornment fb. ∎

Depending on the processed input program, and on the adopted SIPS, some predicates may be associated with adornments containing only fs in the rewritten program. Essentially, this means that all instances of these predicates in $TP(P)$ are relevant to answer the given query. Hence, the range of the object variables of all rules defining such predicates cannot be actually restricted, and indeed the magic sets rewriting includes a copy of these rules with a magic atom obtained from the full-free adornment. Possibly, the magic sets rewriting includes other copies of these rules obtained by different adornments, which may deteriorate the bottom-up evaluation of the rewritten program. Luckily, those copies can be removed if magic rules are properly modified. The idea is that magic rules associated with predicates for which a full-free adornment has been produced have to become definitions of the magic atom obtained from the full-free adornment. The strategy is summarized in Algorithm 2, and can be efficiently implemented in two steps: a first linear traversal of the program to identify predicates of the form $m\#p\#f\cdots f$ and to flag predicate p; a second linear traversal of the program to remove and rewrite rules with predicate $m\#p\#\mathbf{s}$, for all flagged predicates p.

Example 4. Consider program P_{job} from Example 1, the query `par(a,Y)`, and parallel SIPS. The output of the magic sets algorithm is the following (rules in the order of production):

```
m#par#bf(a).
par(X,Y) :- m#par#bf(X), job(X), job(Y), not dep(X,Y), not dep(Y,X).
m#dep#bf(X) :- m#par#bf(X).
m#dep#fb(X) :- m#par#bf(X).
dep(X,Y) :- m#dep#bf(X), require(X,Y).
dep(X,Y) :- m#dep#bf(X), require(X,Z), dep(Z,Y).
m#dep#ff :- m#dep#bf(X).
dep(X,Y) :- m#dep#fb(Y), require(X,Y).
dep(X,Y) :- m#dep#fb(Y), require(X,Z), dep(Z,Y).
m#dep#ff :- m#dep#fb(Y).
dep(X,Y) :- m#dep#ff, require(X,Y).
dep(X,Y) :- m#dep#ff, require(X,Z), dep(Z,Y).
m#dep#ff :- m#dep#ff.
```

Processing the above program with Algorithm 2 results into the program

```
m#par#bf(a).
par(X,Y) :- m#par#bf(X), job(X), job(Y), not dep(X,Y), not dep(Y,X).
m#dep#ff :- m#par#bf(X).
m#dep#ff :- m#par#bf(X).
dep(X,Y) :- m#dep#ff, require(X,Y).
dep(X,Y) :- m#dep#ff, require(X,Z), dep(Z,Y).
m#dep#ff :- m#dep#ff.
```

Essentially, since everything is relevant to answer the given query using parallel SIPS, the magic sets rewriting was eventually unrolled.   ∎

4   Conclusion

The semantics of many constructs of Description Logics is defined in terms of First-Order Logic expressions, and these expressions are often naturally expressible in Datalog. A prominent example is the rewriting proposed by Eiter et al. [28], which can be applied to Horn-$\mathcal{SHIQ}$ knowledge bases. One of the clear advantages of such compilations into Datalog is the availability of many efficient reasoners, employing several state-of-the-art techniques to optimize query answering. As a prominent example, the magic sets rewriting drives the bottom-up evaluation of Datalog programs according to the query given in input; recently proposed improvements to the magic sets algorithm inhibit the creation of recursive definitions and partially unroll the rewriting for predicates whose extension cannot be limited.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995), http://webdam.inria.fr/Alice/

2. Acciarri, A., Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: Quonto: Querying ontologies. In: Proceedings of AAAI. pp. 1670–1671. AAAI Press / The MIT Press (2005)

3. Alviano, M., Calimeri, F., Dodaro, C., Fuscà, D., Leone, N., Perri, S., Ricca, F., Veltri, P., Zangari, J.: The ASP system DLV2. In: Balduccini, M., Janhunen, T. (eds.) Logic Programming and Nonmonotonic Reasoning - 14th International Conference, LPNMR 2017, Espoo, Finland, July 3-6, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10377, pp. 215–221. Springer (2017). https://doi.org/10.1007/978-3-319-61660-5_19, `https://doi.org/10.1007/978-3-319-61660-5_19`

4. Alviano, M., Faber, W., Greco, G., Leone, N.: Magic sets for disjunctive datalog programs. Artif. Intell. **187**, 156–192 (2012). https://doi.org/10.1016/j.artint.2012.04.008, `https://doi.org/10.1016/j.artint.2012.04.008`

5. Alviano, M., Leone, N., Veltri, P., Zangari, J.: Enhancing magic sets with an application to ontological reasoning. Theory Pract. Log. Program. **19**(5-6), 654–670 (2019). https://doi.org/10.1017/S1471068419000115, `https://doi.org/10.1017/S1471068419000115`

6. Baader, F., Brandt, S., Lutz, C.: Pushing the EL envelope. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI). pp. 364–369. Professional Book Center (2005), `http://ijcai.org/Proceedings/05/Papers/0372.pdf`

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

8. Baget, J., Leclère, M., Mugnier, M., Salvat, E.: On rules with existential variables: Walking the decidability line. Artif. Intell. **175**(9-10), 1620–1654 (2011). https://doi.org/10.1016/j.artint.2011.03.002, `https://doi.org/10.1016/j.artint.2011.03.002`

9. Balbin, I., Port, G.S., Ramamohanarao, K., Meenakshi, K.: Efficient bottom-up computation of queries on stratified databases. J. Log. Program. **11**(3&4), 295–344 (1991). https://doi.org/10.1016/0743-1066(91)90030-S, `https://doi.org/10.1016/0743-1066(91)90030-S`

10. Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.D.: Magic sets and other strange ways to implement logic programs. In: Silberschatz, A. (ed.) Proceedings of the Fifth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems, March 24-26, 1986, Cambridge, Massachusetts, USA. pp. 1–15. ACM (1986). https://doi.org/10.1145/6012.15399, `https://doi.org/10.1145/6012.15399`

11. Bárány, V., Gottlob, G., Otto, M.: Querying the guarded fragment. Logical Methods in Computer Science **10**(2) (2014). https://doi.org/10.2168/LMCS-10(2:3)2014, `https://doi.org/10.2168/LMCS-10(2:3)2014`

12. Beeri, C., Ramakrishnan, R.: On the power of magic. J. Log. Program. **10**(3&4), 255–299 (1991). https://doi.org/10.1016/0743-1066(91)90038-Q, `https://doi.org/10.1016/0743-1066(91)90038-Q`

13. Behrend, A.: Soft stratification for magic set based query evaluation in deductive databases. In: Neven, F., Beeri, C., Milo, T. (eds.) Proceedings of the Twenty-Second ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 9-12, 2003, San Diego, CA, USA. pp. 102–110. ACM (2003). https://doi.org/10.1145/773153.773164, `https://doi.org/10.1145/773153.773164`

14. Bienvenu, M., ten Cate, B., Lutz, C., Wolter, F.: Ontology-based data access: A study through disjunctive datalog, csp, and MMSNP. ACM Trans. Database Syst. **39**(4), 33:1–33:44 (2014). https://doi.org/10.1145/2661643, `https://doi.org/10.1145/2661643`

15. Bodirsky, M., Dalmau, V.: Datalog and constraint satisfaction with infinite templates. J. Comput. Syst. Sci. **79**(1), 79–100 (2013). https://doi.org/10.1016/j.jcss.2012.05.012

16. Bourhis, P., Manna, M., Morak, M., Pieris, A.: Guarded-based disjunctive tuple-generating dependencies. ACM Trans. Database Syst. **41**(4), 27:1–27:45 (2016). https://doi.org/10.1145/2976736, `https://doi.org/10.1145/2976736`

17. Brandt, S.: Polynomial time reasoning in a description logic with existential restrictions, GCI axioms, and - what else? In: de Mántaras, R.L., Saitta, L. (eds.) Proceedings of the 16th Eureopean Conference on Artificial Intelligence (ECAI). pp. 298–302. IOS Press (2004)

18. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. J. Artif. Intell. Res. **48**, 115–174 (2013). https://doi.org/10.1613/jair.3873, `https://doi.org/10.1613/jair.3873`

19. Calì, A., Gottlob, G., Lukasiewicz, T.: Tractable query answering over ontologies with datalog+/-. In: Grau, B.C., Horrocks, I., Motik, B., Sattler, U. (eds.) Proceedings of the 22nd International Workshop on Description Logics (DL). CEUR Workshop Proceedings, vol. 477. CEUR-WS.org (2009), `http://ceur-ws.org/Vol-477/paper_46.pdf`

20. Calì, A., Gottlob, G., Lukasiewicz, T.: A general datalog-based framework for tractable query answering over ontologies. J. Web Semant. **14**, 57–83 (2012). https://doi.org/10.1016/j.websem.2012.03.001, `https://doi.org/10.1016/j.websem.2012.03.001`

21. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web **8**(3), 471–487 (2017)

22. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rodriguez-Muro, M., Rosati, R., Ruzzi, M., Savo, D.F.: The MASTRO system for ontology-based data access. Semantic Web **2**(1), 43–53 (2011). https://doi.org/10.3233/SW-2011-0029

23. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. Artif. Intell. **195**, 335–360 (2013). https://doi.org/10.1016/j.artint.2012.10.003, `https://doi.org/10.1016/j.artint.2012.10.003`

24. Carral, D., Dragoste, I., Krötzsch, M.: The combined approach to query answering in horn-alchoiq. In: KR. pp. 339–348. AAAI Press (2018)

25. Carral, D., González, L., Koopmann, P.: From horn-sriq to datalog: A data-independent transformation that preserves assertion entailment. In: Proceedings of AAAI. pp. 2736–2743. AAAI Press (2019)

26. Ceri, S., Gottlob, G., Tanca, L.: What you always wanted to know about datalog (and never dared to ask). IEEE Trans. Knowl. Data Eng. **1**(1), 146–166 (1989)

27. Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Surveys in computer science, Springer (1990)

28. Eiter, T., Ortiz, M., Simkus, M., Tran, T., Xiao, G.: Query rewriting for horn-shiq plus rules. In: Hoffmann, J., Selman, B. (eds.) Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence, July 22-26, 2012, Toronto, Ontario, Canada. AAAI Press (2012), `http://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/4931`

29. Faruqui, R.U., MacCaull, W.: OwlOntDB: A scalable reasoning system for OWL 2 RL ontologies with large aboxes. In: FHIES. Lecture Notes in Computer Science, vol. 7789, pp. 105–123. Springer (2012)

30. Gottlob, G., Kikot, S., Kontchakov, R., Podolskii, V.V., Schwentick, T., Zakharyaschev, M.: The price of query rewriting in ontology-based data access. Artif. Intell. **213**, 42–59 (2014). https://doi.org/10.1016/j.artint.2014.04.004, `https://doi.org/10.1016/j.artint.2014.04.004`

31. Gottlob, G., Orsi, G., Pieris, A.: Query rewriting and optimization for ontological databases. ACM Trans. Database Syst. **39**(3), 25:1–25:46 (2014). https://doi.org/10.1145/2638546, `https://doi.org/10.1145/2638546`

32. Gottlob, G., Pieris, A., Tendera, L.: Querying the guarded fragment with transitivity. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M.Z., Peleg, D. (eds.) Proceedings of the 40th International Colloquium on Automata, Languages, and Programming (ICALP). Lecture Notes in Computer Science, vol. 7966, pp. 287–298. Springer (2013). https://doi.org/10.1007/978-3-642-39212-2_27, `https://doi.org/10.1007/978-3-642-39212-2_27`

33. Grau, B.C., Kharlamov, E., Kostylev, E.V., Zheleznyakov, D.: Controlled query evaluation for datalog and OWL 2 profile ontologies. In: IJCAI (2015)

34. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: combining logic programs with description logic. In: Hencsey, G., White, B., Chen, Y.R., Kovács, L., Lawrence, S. (eds.) Proceedings of the Twelfth International World Wide Web Conference (WWW). pp. 48–57. ACM (2003). https://doi.org/10.1145/775152.775160, `https://doi.org/10.1145/775152.775160`

35. Hustadt, U., Motik, B., Sattler, U.: Data complexity of reasoning in very expressive description logics. In: Kaelbling, L.P., Saffiotti, A. (eds.) Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence (IJCAI). pp. 466–471. Professional Book Center (2005), `http://ijcai.org/Proceedings/05/Papers/0326.pdf`

36. Johnson, D.S., Klug, A.C.: Testing containment of conjunctive queries under functional and inclusion dependencies. J. Comput. Syst. Sci. **28**(1), 167–189 (1984). https://doi.org/10.1016/0022-0000(84)90081-3, `https://doi.org/10.1016/0022-0000(84)90081-3`

37. Kemp, D.B., Srivastava, D., Stuckey, P.J.: Bottom-up evaluation and query optimization of well-founded models. Theor. Comput. Sci. **146**(1&2), 145–184 (1995). https://doi.org/10.1016/0304-3975(94)00153-A, `https://doi.org/10.1016/0304-3975(94)00153-A`

38. Kerisit, J., Pugin, J.: Efficient query answering on stratified databases. In: FGCS. pp. 719–726 (1988)

39. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: IJCAI (2011)

40. Krötzsch, M., Mehdi, A., Rudolph, S.: Orel: Database-driven reasoning for OWL 2 profiles. In: Description Logics. CEUR Workshop Proceedings, vol. 573. CEUR-WS.org (2010)

41. Leone, N.: The AI system DLV: ontologies, reasoning, and more. In: IC3K. pp. 5–16 (2018)

42. Lloyd, J.W.: Foundations of Logic Programming, 2nd Edition. Springer (1987)

43. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic EL using a relational database system. In: Proceedings of IJCAI. pp. 2070–2075 (2009)

44. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: Rdfox: A highly-scalable RDF store. In: Arenas, M., Corcho, Ó., Simperl, E., Strohmaier, M., d'Aquin, M., Srinivas, K., Groth, P.T., Dumontier, M., Heflin, J., Thirunarayan, K., Staab, S. (eds.) Proceedings of the 14th International Semantic Web Conference (ISWC). Lecture Notes in Computer Science, vol. 9367, pp. 3–20. Springer (2015). https://doi.org/10.1007/978-3-319-25010-6_1, `https://doi.org/10.1007/978-3-319-25010-6_1`
45. Ortiz, M.: Ontology based query answering: The story so far. In: Bravo, L., Lenzerini, M. (eds.) Proceedings of the 7th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW). CEUR Workshop Proceedings, vol. 1087. CEUR-WS.org (2013), `http://ceur-ws.org/Vol-1087/keynote3.pdf`
46. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. Applied Logic **8**(2), 186–209 (2010). https://doi.org/10.1016/j.jal.2009.09.004, `https://doi.org/10.1016/j.jal.2009.09.004`
47. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. J. Appl. Log. **8**(2), 186–209 (2010). https://doi.org/10.1016/j.jal.2009.09.004, `https://doi.org/10.1016/j.jal.2009.09.004`
48. Przymusinski, T.C.: On the declarative and procedural semantics of logic programs. J. Autom. Reasoning **5**(2), 167–205 (1989). https://doi.org/10.1007/BF00243002, `https://doi.org/10.1007/BF00243002`
49. Rosati, R.: The limits of querying ontologies. In: Schwentick, T., Suciu, D. (eds.) Proceedings of the 11th International Conference on Database Theory (ICDT). Lecture Notes in Computer Science, vol. 4353, pp. 164–178. Springer (2007). https://doi.org/10.1007/11965893_12, `https://doi.org/10.1007/11965893_12`
50. Rosati, R., Almatelli, A.: Improving query answering over dl-lite ontologies. In: Lin, F., Sattler, U., Truszczynski, M. (eds.) Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning (KR). AAAI Press (2010), `http://aaai.org/ocs/index.php/KR/KR2010/paper/view/1400`
51. Rosati, R., Almatelli, A.: Improving query answering over dl-lite ontologies. In: Proceedings of KR. AAAI Press (2010)
52. Ross, K.A.: Modular stratification and magic sets for datalog programs with negation. J. ACM **41**(6), 1216–1266 (1994). https://doi.org/10.1145/195613.195646, `https://doi.org/10.1145/195613.195646`
53. Stefanoni, G., Motik, B., Horrocks, I.: Small datalog query rewritings for EL. In: DL. CEUR Workshop Proceedings, vol. 846 (2012)
54. Stefanoni, G., Motik, B., Krötzsch, M., Rudolph, S.: The complexity of answering conjunctive and navigational queries over OWL 2 EL knowledge bases. J. Artif. Intell. Res. **51**, 645–705 (2014)
55. Stuckey, P.J., Sudarshan, S.: Compiling query constraints. In: Vianu, V. (ed.) Proceedings of the Thirteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, May 24-26, 1994, Minneapolis, Minnesota, USA. pp. 56–67. ACM Press (1994). https://doi.org/10.1145/182591.182598, `https://doi.org/10.1145/182591.182598`
56. W3C: SPARQL 1.1 entailment regimes. `https://www.w3.org/TR/sparql11-entailment/` (2013)
57. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. In: IJCAI (2018)
58. Xiao, G., Eiter, T., Heymans, S.: The drew system for nonmonotonic dl-programs. In: CSWS. pp. 383–390. Springer (2012)