Uniform Evaluation of Nonmonotonic DL-Programs*

Thomas Eiter, Thomas Krennwallner, Patrik Schneider, and Guohui Xiao

Institut für Informationssysteme, Technische Universität Wien
Favoritenstraße 9-11, A-1040 Vienna, Austria
{eiter,tkren,patrik,xiao}@kr.tuwien.ac.at

Abstract. Nonmonotonic description logic programs are a major formalism for a loose coupling of rules and ontologies, formalized in logic programming and description logics, respectively. While this approach is attractive for combining systems, the impedance mismatch between different reasoning engines and the API-style interfacing are an obstacle to efficient evaluation of dl-programs in general. Uniform evaluation circumvents this by transforming programs into a single formalism, which can be evaluated on a single reasoning engine. In this paper, we consider recent and ongoing work on this approach which uses relational first-order logic (and thus relational database engines) and datalog with negation as target formalisms. Experimental data show that significant performance gains are possible and suggest the potential of this approach.

1 Introduction

In the past decade, the growing importance of the Web and its envisioned future development has triggered a lot of research on accessing and processing data based on semantic approaches. The distributed nature of the Web poses a challenge for semantic integration, even at the level of weak interoperability of different sites. To mitigate this problem, standard knowledge representation formats have been conceived in the layered architecture of the so called Semantic Web, in which the Web Ontology language (OWL) and the more recent Rule Interchange Format (RIF) play a prominent role.

In this context, the issue of combining rules and ontologies has been considered in a number of works; see [12, 41, 15] for some recent surveys. Among several approaches, loose coupling of rules and ontologies is one which aims at combining respective knowledge bases by means of a clean interfacing semantics, in which roughly speaking inferences are mutually exchanged such that the one knowledge base takes the imported information into account, and exports in turn conclusions to the other knowledge base. This approach is fostered by nonmonotonic description logic (dl-) programs [17], where this exchange is handled by a generalization of the answer set semantics of nonmonotonic logic programs [23]. Follow up work has adapted this approach to other formalisms (e.g., [51, 29, 20]) and considered alternative semantics (e.g. [39, 52, 16]).

The loose coupling approach is attractive in several regards. First, legacy knowledge bases, powered by different reasoning engines, can be combined. Second, thanks to

* This work has been supported by the Austrian Science Fund (FWF) projects P20840 & P20841 and by the EC ICT Integrated Project Ontorule (FP7 231875).

the interfacing and loose semantics connection, it is fairly easy to incorporate further knowledge formats besides rules and OWL (description logic) ontologies, e.g. RDF knowledge base; HEX-programs [20] are a respective generalization of dl-programs, which in fact allow for incorporating arbitrary software. And third, view based data access of loose coupling is in support of privacy, as the internal structure of a knowledge base remains hidden.

On the other hand, the impedance mismatch of different formalisms and reasoning engines comes at a price. A simple realization of the loose coupling considers the interface calls as an API which makes computation expensive, in particular if rules lead to choices via the underlying semantics. The black box view of other knowledge bases hinders optimization and is a major obstacle for scalability.

There is a ray of hope, however, if information about the internal structure of an accessed knowledge base is available. The simplest way is to give up privacy and make the knowledge base transparent such that its axioms and semantics are known ("open source"). Other possibility is to not reveal all information, but some abstract properties [19]. This still, however, leaves the impedance mismatch between different reasoning engines.

To overcome the latter, a suggestive approach is to convert the evaluation problem into one for a single reasoning engine, which means to transform a dl-program into an (equivalent) knowledge base in one formalism for evaluation (ideally, in one already considered). This opens a middle ground for evaluation and privacy, as the transformation may hide or blur the internal structure of the knowledge base.[1]

This idea of a "uniform evaluation" approach raises several issues.

1. Naturally, the cost of a transformation, and whether such a transformation is efficiently possible. In a sense, efficiency means that the overall evaluation cost does not increase with respect to some measure (typically, worst case complexity). Here, notions of embedding of a formalism into another might be considered, and besides computational also semantic properties like modularity are of interest, cf. [31].
2. At a more foundational level, whether a transformation to a target formalism does exist after all, if resources for its computation are disregarded, or even allowed to be not computable. Here one may further consider whether the transformation is ad hoc, for concrete knowledge bases embraced with a dl-program, or whether it is independent of their data (factual resp. assertional) parts.
3. As for the evaluation, the complexity of the target formalism, where –as common in the study of data and knowledge representation formalisms– the data complexity (i.e., complexity under varying data) deserves particular attention. Transformation to a formalism with lower complexity comes inevitably at some cost, which usually means an exponential increase in the size of the knowledge base. Popular examples of this in Description Logics are first-order rewriting of conjunctive query answering over DL-Lite ontologies [8] or the reduction of $\mathcal{SHIQ}$ to disjunctive datalog [30].
4. The feasibility of transformations for practical concerns, in particular for evaluation using available technology and platforms. In this regard, it is of interest to see whether theoretical results, as obtained for the items 1 and 3, live up to practical

[1] Note, however, that uniform evaluation is different from tight integration of KBs in a single unifying logic, cf. [41].

realization. For example, an exponential blowup of the DL-Lite rewriting in [8], which is prohibitive in some cases, can be mitigated [45] or avoided using other notions of rewritings [36].

As for items 1 and 2, embeddings of dl-programs into various well-known nonmonotonic logics have been studied, among them Autoepistemic Logic [13], Equilibrium Logic [21] (a logic-based version of Answer Set Semantics), Reiter's Default Logic [53], and MKNF [41]; however, these works targeted more semantic aspects than evaluation.

In recent and ongoing works, uniform evaluation of various fragments of dl-programs has been considered at the KBS group of TU Wien; in particular, transformation to first-order logic [16] and to datalog with negation [28, 55]. This paper reviews some of this work with a focus on items 1 and 4 from above, and reports some experimental data.

The main observation is that, expectedly, uniform evaluation leads to significant performance improvements compared to simple (standard) evaluation of dl-programs coupling different reasoning engines, as done by the dlvhex reasoner.[2] The results show that the approach has potential, although further work is needed to boost scalability.[3]

The remainder of this article is organized as follows. In the next section, we briefly recall dl-programs. After that, we consider in Section 3 transformation of dl-programs to first-order logic, which makes evaluation using relational database technology possible. This requires to exclude recursion from rules and to have ontologies that are first-order rewritable. Section 4 considers transformation to datalog with negation, which hosts recursion in rules and for encodings of ontologies; as discussed in Section 5, the program can be naturally expressed in a modular version of datalog. The final Section 6 provides a discussion and gives an outlook on possible future work.

2 DL-Programs

We recall nonmonotonic description logic programs (simply dl-programs) under the answer set [17] and the well-founded semantics [16]. They combine Description Logics and nonmonotonic logic programs in a loose coupling, under a strict semantic separation.

2.1 Description Logics

Description Logics (DLs) are a well-known family of KR formalisms based on fragments of *first-order (FO) logic*. The vocabulary of a DL consists of *individual*, *class*, and *role names*; *knowledge bases* (KB) consist of a *terminological box* (TBox), which contains axioms about relations between classes and roles, and an *assertional box* (ABox), which contains factual knowledge about individuals. For the purpose of this paper, we first recall DL-Lite [8, 2], the logical underpinning of OWL 2 QL; later, we consider $\mathcal{EL}$ [3, 4], the underpinning of OWL 2 EL, and $\mathcal{LDL}^+$, which is a DL strongly related to Datalog and OWL 2 RL (for all these OWL 2 profiles, see [40]).

DL-Lite. Consider a vocabulary of individual names $\mathbf{I}$, *atomic concepts* $\mathbf{C}$, and *atomic roles* $\mathbf{R}$. Then, for A and P being an atomic concept and atomic role, respectively,

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

[3] Implementation information, benchmark instances, and further details on all benchmarks is available at http://www.kr.tuwien.ac.at/research/systems/drew/experiments.html.

we define *basic* concepts B and *basic* roles R, *complex* concepts C and *complex* role expressions E as

$$\begin{array}{lcl@{\qquad\qquad}lcl} B & ::= & A \mid \exists R & C & ::= & B \mid \neg B \\ R & ::= & P \mid P^- & E & ::= & R \mid \neg R \end{array}$$

where P^- is the inverse of P.

A DL-Lite$_R$ *knowledge base* is a pair $L = (\mathcal{T}, \mathcal{A})$ where the TBox $\mathcal{T}$ consists of a finite set of *inclusion assertions* of the form $B \sqsubseteq C$ and $R \sqsubseteq E$, and the ABox $\mathcal{A}$ is a finite set of *membership assertions* on atomic concepts and on atomic roles of the form $A(a)$ and $P(a, b)$, where a and b are individual names of $\mathbf{I}$.

The semantics of DL-Lite$_R$ is given in terms of FO interpretations $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a nonempty domain and $\cdot^{\mathcal{I}}$ an *interpretation function* such that $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$ for all $a \in \mathbf{I}$, $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ for all $A \in \mathbf{C}$, $P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$ for all $P \in \mathbf{R}$, and

- $(P^-)^{\mathcal{I}} = \{(a_2, a_1) \mid (a_1, a_2) \in P^{\mathcal{I}}\}$;
- $(\exists R)^{\mathcal{I}} = \{a_1 \mid$ there exists some a_2 such that $(a_1, a_2) \in R^{\mathcal{I}}\}$;
- $(\neg B)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus B^{\mathcal{I}}$; and
- $(\neg R)^{\mathcal{I}} = \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}} \setminus R^{\mathcal{I}}$.

An interpretation $\mathcal{I}$ *satisfies* a concept inclusion $C_1 \sqsubseteq C_2$ (resp. role inclusion $E_1 \sqsubseteq E_2$), if ${C_1}^{\mathcal{I}} \subseteq {C_2}^{\mathcal{I}}$ (resp. ${E_1}^{\mathcal{I}} \subseteq {E_2}^{\mathcal{I}}$), and satisfies a TBox $\mathcal{T}$, if it satisfies each inclusion assertion in $\mathcal{T}$. Furthermore, $\mathcal{I}$ satisfies $C(a)$, if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ and satisfies $R(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$, and $\mathcal{I}$ satisfies an ABox $\mathcal{A}$, if it satisfies each assertion in $\mathcal{A}$. Finally, $\mathcal{I}$ *satisfies* $L = (\mathcal{T}, \mathcal{A})$, if it satisfies $\mathcal{T}$ and $\mathcal{A}$. A KB L (resp. TBox $\mathcal{T}$) *logically implies* an assertion α, if all models of L (resp. $\mathcal{T}$) satisfy α. As usual, satisfaction and logical entailment are denoted with $\models$.

2.2 Description Logic Programs

Description logic programs (L, P) have rules similar as logic programs with negation as failure, but the rule bodies may also contain *queries to* L in their bodies.

Suppose $\Phi = (\mathcal{P}, \mathcal{C})$, is a vocabulary of finite sets $\mathcal{P}$ and $\mathcal{C}$ of predicate and constant symbols, respectively, and a set $\mathcal{X}$ of variables. As usual, elements from $\mathcal{C} \cup \mathcal{X}$ are *terms*, and atoms have the form $p(t_1, \ldots, t_n)$, where $p \in \mathcal{P}$ has arity n and all t_i are terms.

Queries to L occur in so-called dl-atoms. A *dl-query* $Q(\mathbf{t})$ is either

(a) a concept inclusion axiom F or its negation $\neg F$; or
(b) of the forms $C(t)$ or $\neg C(t)$, where C is a concept, and t is a term; or
(c) of the forms $R(t_1, t_2)$ or $\neg R(t_1, t_2)$, where R is a role, and t_1 and t_2 are terms.

A *dl-atom* has then the form

$$DL[S_1 op_1 p_1, \ldots, S_m op_m\ p_m; Q](\mathbf{t})\,, \qquad m \geq 0, \tag{1}$$

where each S_i is either a concept or a role; $op_i \in \{\uplus, \uplus\}$; p_i is a unary (resp.,binary) predicate symbol, if S_i is a concept (a role); and $Q(\mathbf{t})$ is a dl-query. Intuitively, $op_i = \uplus$ (resp. $\uplus$) increases S_i ($\neg S_i$) by the extension of p_i.

A *dl-rule* r is of the form

$$a \ \leftarrow\ b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_m,\ m \geq k \geq 0, \tag{2}$$

where a is an atom (the *head*) and each b_i is either an atom or a dl-atom, and not is negation as failure (default negation). A *dl-program* $KB = (L, P)$ consists of a DL knowledge base L and a finite set of dl-rules P; it is *positive*, if P is positive.

Example 1. Let $KB = (L, P)$ where $L = \{C \sqsubseteq D\}$ and P is the set of rules

$$p(a); \quad p(b); \quad q(c);$$
$$s(X) \leftarrow DL[C \uplus p; D](X),\ not\ DL[C \uplus q, C \uplus p; D](X)\ .$$

Intuitively, we extend in the first dl-atom concept C by predicate p and retrieve then all instances from D in this extended ABox. With the second dl-atom we extend C and $\neg C$ by the extensions of q and p, resp. Thus, the intuitive model for this dl-program would be $\{p(a), p(b), q(c), s(a), s(b)\}$.

Semantics. The *Herbrand base* of P, denoted HB_P, is the set of all atoms $p(c_1 \ldots, c_n)$ where $p \in \mathcal{P}$ occurs in P and all c_i are from $\mathcal{C}$. An *interpretation I relative to P* is any subset of HB_P. Such an I *satisfies* (*models*) a ground (i.e.,variable-free) atom or dl-atom a under L, denoted $I \models_L a$, if the following holds:

- $a \in I$, if $a \in HB_P$;
- $L(I; \lambda) \models Q(\mathbf{c})$, where $\lambda = S_1 op_1 p_1, \ldots, S_m op_m p_m$, $L(I; \lambda) = L \cup \bigcup_{i=1}^{m} A_i(I)$ and, for $1 \leq i \leq m$,

$$A_i(I) = \begin{cases} \{S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \uplus, \\ \{\neg S_i(\mathbf{e}) \mid p_i(\mathbf{e}) \in I\}, & \text{if } op_i = \cup, \end{cases}$$

  if a is a ground dl-atom $DL[\lambda; Q](\mathbf{c})$.

I satisfies a ground dl-rule r of form (2) if either (i) $I \not\models_L a$, or (ii) $I \not\models_L b_i$ for some $1 \leq i \leq k$ or (iii) $I \models_L b_j$ for some $k < j < m$. I satisfies a dl-program $KB = (L, P)$, denoted $I \models KB$, iff $I \models_L r$ for every rule $r \in ground(P)$, where $ground(P)$ is the set of all ground instances of rules in P (relative to HB_P).

It is easy to see that every positive KB has some model and, like every Horn logic program, a unique minimal (under inclusion $\subseteq$) model, denoted M_{KB}. This model naturally captures the semantics of KB.

Answer Set Semantics. The *answer sets* of a general dl-program $KB = (L, P)$ are defined by a reduction to positive dl-programs. The *(strong) dl-transform* of P relative to L and an interpretation $I \subseteq HB_P$, denoted sP_L^I, results from $ground(P)$ by deleting (i) every dl-rule r such that $I \models_L a$ for some $a \in B^-(r)$, and (ii) the negative bodies of all remaining dl-rules; note that sP_L^I generalizes the Gelfond-Lifschitz reduct P^I [23]. Let $KB^I = (L, sP_L^I)$. Since KB^I is positive, it has a unique minimal (the least) model, denoted $\mathsf{LM}(KB^I)$. Then $I \subseteq HB_P$ is a *(strong) answer set* of KB, if $I = \mathsf{LM}(KB^I)$. We write $KB \models a$ for a ground atom a if $I \models_L a$ for every answer set of KB.

Well-founded Semantics. Define the operator γ_{KB} on interpretations I of KB by $\gamma_{KB}(I) = \mathsf{LM}(KB^I)$. As γ_{KB} is anti-monotone, $\gamma_{KB}^2(I) = \gamma_{KB}(\gamma_{KB}(I))$ is monotone and has a least fixpoint, which is the set of *well-founded* atoms of KB, denoted $WFS(KB)$ [16]; we denote with $KB \models_{wf} a$ that $a \in WFS(KB)$.

Example 2. KB from Example 1 has the single answer set $\{p(a), p(b), q(c), s(a), s(b)\}$, which coincides with $WFS(KB)$. If we replace the facts for p in P by the "guessing" rules $p(a) \leftarrow not\ p(b)$; $\quad p(b) \leftarrow not\ p(a)$, the resulting KB has the two answer sets $\{p(a), q(c), s(a)\}$ and $\{p(b), q(c), s(b)\}$, while $q(c)$ is the only well-founded atom.

3 First-Order Rewritability

In this section, we introduce the first-order rewritable case of dl-programs. This case is motivated by the already introduced DL-Lite family, which allows the uniform evaluation of restricted classes of dl-programs on relational database technology.

3.1 First-Order Rewritable dl-Programs

We understand FO rewritability of a dl-program $KB = (L, P)$ in the sense that query answering from KB, i.e., to decide whether $KB \models p(\boldsymbol{c})$ for an atom $p(\boldsymbol{c})$, is expressible by a FO formula $\phi(\boldsymbol{x})$ over the *relational schema* induced by the vocabulary of L, such that $KB \models p(\boldsymbol{c})$ iff $\mathcal{A} \models \phi(\boldsymbol{c})$, where ϕ depends on p, P and L; in a data-centric view, it only depends on the TBox of L, but not on the concrete ABox $\mathcal{A}$. For the DL-Lite family, and in particular for DL-Lite$_R$, the analog property for answering conjunctive queries is also called *FOL-reducibility*.[4] Like for the latter, query answering for FO rewritable dl-programs is feasible on a Relational Data Base Management System (RDBMS) by casting the FO formula into SQL statements.

As recursion is not expressible in FO logic, it must be banned for FO rewritability from KB; this is achieved by acyclicity. Let $\mathcal{P}_P$ be the set of all predicates symbols in P. Then P is acyclic, if there exists a mapping $\mathcal{K}\colon \mathcal{P}_P \to \{0, \ldots, n\}$ such that for every rule $r \in P$ of form (2) and $1 \leq i \leq n$, it holds that $\mathcal{K}(p) > \mathcal{K}(q)$ where p occurs in a and q occurs in b_i (in case of a dl-atom of form (1), q has to occur among the p_j). Note that every acyclic KB has a unique answer set, which coincides with $WFS(KB)$ [16]. Thus, for every ground atom a, $KB \models a$ iff $KB \models_{wf} a$.

An important result regarding FO rewritability of dl-programs under the well-founded semantics was given in [16]. We recall the results in Theorem 1 and 2.

Theorem 1 (FO rewritable dl-programs [16]). *Let $KB = (L, P)$ be an acyclic dl-program, and $p(\boldsymbol{c})$ an atom, such that (1) every dl-query in P is FO-rewritable, and (2) if the operator $\uplus$ occurs in P, then L is defined over a DL that (2a) is* CWA-satisfiable *(i.e., for every DL KB L', the union of L' and all membership assertions that are not entailed by L' is satisfiable), and (2b) allows for FO-rewritable concept and role memberships. Then, deciding $KB \models_{wf} p(\boldsymbol{c})$ is FO-rewritable.*

The proof is based on an induction on mapping $\mathcal{K}$ and the following assumptions:

(a) every dl-atom δ can be expressed as FO formula over the ABox of L;
(b) every predicate of rank 0 is easily expressed as FO formula over the facts of P;
(c) every other predicate p_I can be expressed by the disjunction of the existentially quantified bodies of the rules which share p_I in their heads, and where the NAF atoms are interpreted as classical negation.

Concerning (a), let $\delta = DL[\lambda; Q](\boldsymbol{c})$ be a dl-atom of form (1) such that $\lambda = \lambda^+, \lambda^-$ is the list of m positive ($op_i = \uplus$) and negative ($op_i = \cup$) extensions of L, and $Q(\boldsymbol{c})$ a dl-query. Each extension $S_i\ op_i\ p_i$ in λ can be expressed in terms of a FO formula $\psi_{S_i}(\boldsymbol{y})$ over L. The dl-query $Q(\boldsymbol{c})$ can be expressed as a FO formula $\alpha(\boldsymbol{x})$ over L. Every input

[4] Conjunctive queries over dl-programs can be expressed by rules, thus atomic queries suffice.

predicate p_j in λ is as a FO formula $\psi_j(\boldsymbol{x})$ over P. Then, the FO formula $\delta(\boldsymbol{x})$ over the ABox $\mathcal{A}$ of L and facts of P

$$\delta(\boldsymbol{x}) = \alpha^{\lambda^+}(\boldsymbol{x}) \vee \bigvee_{j=1}^{m} \exists \boldsymbol{y} \left(\psi_{S_j}^{\lambda^+}(\boldsymbol{y}) \wedge \psi_j(\boldsymbol{y}) \right) \quad , \tag{3}$$

where α^{λ^+} (resp. $\psi_{S_j}^{\lambda^+}$) is obtained from α (resp. ψ_{S_j}) by replacing every $S_i(s)$, such that S_i occurs in λ^+ by $S_i(s) \vee \psi_{i_1}(s) \vee \cdots \vee \psi_{i_{k_i}}(s)$, where $S_{i_1}, \ldots, S_{i_{k_i}}$ are all occurrences of S_j in λ^+. For the meaning of the assumptions, we refer to [16].

Example 3. Consider $KB = (L, P)$ from Example 1. To illustrate all parts of the transformation, we create a variant of KB as $KB' = (L, P')$, where $P' = P \cup \{s(X) \leftarrow q(X)\}$. We can express query $D(X)$ (after the *perfect rewriting*) by the FO formula $\alpha(X) = C(x) \vee D(x)$ over $\mathcal{A}$ ($= \emptyset$). As FO formulas, p and q are $\psi_p = p(x)$ and $\psi_q = q(x)$ over $F = \{p(a), p(b), q(c)\}$. Then the dl-atom $DL[C \uplus p; D](X)$ is translated into $\delta_1(x) = C(x) \vee D(x) \vee p(x)$, while the dl-atom $DL[C \uplus q; C \uplus p; D](X)$ is translated into $\delta_2(x) = C(x) \vee D(x) \vee q(x) \vee \exists y((C(y) \vee q(y)) \wedge p(y))$, both over F. The rules for predicate s are translated into $\exists x.q(x) \vee \exists x.\, (\delta_1(x) \wedge \neg\delta_2(x))$ over F.

Theorem 2 (FO-rewritable dl-program over the DL-Lite family [16]). *For any vocabulary Φ, acyclic dl-program $KB = (L, P)$, and atom $p(\boldsymbol{c})$, such that 1. L is in a DL of the DL-Lite family, and 2. all dl-queries in P are of the form $C \sqsubseteq D$, $\neg(C \sqsubseteq D)$, $C(t)$ or $R(t, s)$, where C is an atomic concept, D is an (possibly negated) atomic concept, and t, s are terms of L, deciding $KB \models_{wf} p(\boldsymbol{c})$ is first-order rewritable.*

Since the DL-Lite family is CWA-satisfiable [8], operator $\uplus$ is allowed in P and dl-queries of the form $C(t)$ and $R(t, s)$ are immediately FO rewritable. Furthermore, dl-queries of the form $C \sqsubseteq D$ can be reduced to queries as follows: $L' \models C \sqsubseteq D$ iff $L' \cup \{C(e), D'(e), D' \sqsubseteq \neg D, A'(d), A' \sqsubseteq \neg A\} \models A(d)$. Similar, $\neg(C \sqsubseteq D)$ can be reduced to $L' \models \neg(C \sqsubseteq D)$ iff $L' \cup \{C \sqsubseteq D, A'(d), A' \sqsubseteq \neg A\} \models A(d)$, where d and e are fresh individuals, and A, A', and D' are fresh atomic concepts.

3.2 Implementation and Experiments

Based on the ideas above, the experimental system MOR evaluates conjunctive queries over an acyclic dl-program KB using an RDBMS (which we call the database), viz. PostgreSQL 8.4. MOR has three main modules: a Datalog-to-SQL rewriter, a DL-Lite plugin, and an adaption of the DL-Lite$_R$ reasoner Owlgres (see [48]).

- The Datalog-to-SQL rewriter, which is based on well-known techniques, cf. [49], can also handle limited rule recursion (see Subsection 3.3). However, different from DLV$^{\text{DB}}$[49], SQL views are not materialized and recursion is handled differently. In MOR the focus is merely on linear recursion based on a *direct evaluation algorithm* of the transitive closure, and relies on the native implementation of the RDBMS. On the other hand in DLV$^{\text{DB}}$, full recursion is implemented based on an optimized *semi-naive algorithm*, where iteratively SQL statements are executed until a fixpoint is reached (in [43] both algorithms are compared).

– The DL-Lite plugin transforms dl-atoms according to the rewriting above. In that, it exploits a modified version of Owlgres to obtain the result of the $\mathrm{PerfectRef}$ algorithm [8], i.e., the *perfect rewriting* of a query and the TBox, without execution.

Given $KB = (L, P)$ and a conjunctive query Q, the rewriting puts the facts of P and the ABox of L in the database and rewrites the rules of P into cascading VIEWS. For every predicate p occurring in some rule head, one VIEW_p is created, consisting of the UNION of SELECT−PROJECT−JOIN (SPJ) statements for the bodies of rules with p in the head, where negated atoms $not\ a$ are cast to $\mathrm{NOT\ IN}\ (\cdot)$ statements. Acyclicity of KB ensures a proper evaluation order of all views; for details, see [46]. The conjunctive query $Q(\boldsymbol{x}) = p_1(\boldsymbol{x}_1), \ldots, p_m(\boldsymbol{x}_m)$, which can be seen as a rule $Q(\boldsymbol{x}) \leftarrow p_1(\boldsymbol{x}_1), \ldots, p_m(\boldsymbol{x}_m)$, is rewritten into a single (SPJ) SQL statement.

With the $\uplus$ and $\cup$ operator and a *static* ABox in the RDBMS, the rewriting of dl-atoms is more involved. We have to modify the ABox $\mathcal{A}$ temporally to $\mathcal{A}'$ prior to evaluating the dl-query over $\mathcal{A}'$. In addition, the internal DB schema of Owlgres must be respected. Again, acyclicity of KB ensures an evaluation order $\mathcal{K}$ for the temporary modified ABoxes, which is realized for each $DL[\lambda; Q](c)$ as follows:

(a) create new VIEWS representing ABoxes $\mathcal{A}'_{\mathcal{K}}$, building the union of $\mathcal{A}'_{\mathcal{K}-1}$ and all $S_i\ op_i\ p_i$ of λ (where p_i is an existing view);
(b) Modify the perfect rewriting of Q to use $\mathcal{A}'_{\mathcal{K}}$ instead of $\mathcal{A}$.

Clearly, after the evaluation of the dl-program, $\mathcal{A}$ has to be reverted to its original state. MOR allows to access other plugins than the DL-Lite plugin, which may support dl-atoms for querying other DLs, or even other kinds of knowledge sources (e.g., a spatial database). The basic requirement is that the plugin has to return an SQL rewriting.

For experimentation, we considered three different categories of benchmark instances: (1) randomly generated sets of facts (R_n); (2) a simplified version of DBpedia (D_n); and (3) the well-known Lehigh University Benchmark (LUBM) [26] (U_n). As LUBM is not fully in DL-Lite$_R$, we altered roughly 10% of the TBox axioms like transitive roles to normal roles and equality axioms of the form $B \equiv C_1 \sqcap C_2$ to $B \sqsubseteq C_1$ and $B \sqsubseteq C_2$. The index n in our instances denote the number of facts in R_n and the ABox assertions in D_n and U_n; we report the outcome for runs with $n \in \{10\mathrm{k}, 100\mathrm{k}, 250\mathrm{k}, 500\mathrm{k}, 1\mathrm{M}\}$. The test data of R_n was randomly generated allowing a high selectivity among the join attributes. For D_n, different sets of books, periodicals, and publications were extracted from DBpedia, including a single role. The test data for U_n was generated by the LUBM instance generator; e.g., university $U_{100\mathrm{k}}$ has about 12k individuals.

As a baseline system, we compared MOR to DLV [38], DLV$^{\mathrm{DB}}$[49], and dlvhex [18]. DLV$^{\mathrm{DB}}$ is a tight coupling of DLV with a relational DBMS, in which SQL queries over an (external) database can be evaluated. In case of dlvhex, we used its standard DL plug-in (interfacing RacerPro 1.9.2 [27]), which we refer to as dlvhex[DL]. The benchmark runs FO_1–FO_3 are summarized in Table 1. Here, FO_1 has no ontology access; it serves to assess the rule rewriting. In FO_2, the dl-program extended the ABox by constantly 60 individuals, imported from a randomly generated relation. The program in the runs of FO_3 resembles Query 9 of [26] with a triangular pattern of low selectivity roles plus an additional NAF atom.

We conducted our experiments on an openSUSE 11.1 (x86_64) server having a Intel Xeon CPU E5450 3.00GHz and 15.7 GB of RAM. The rewriting was evaluated on a

Table 1: Benchmark Overview

Name	Description	Systems	Data	Reference
FO_1	Tree of binary joins (with negation)	DLV; DLV$^{\text{DB}}$	Random	[46, Ex. 5.2.1]
FO_2	Select a range of the KB upon extension with books from an external source	DLV$^{\text{DB}}$; dlvhex[DL]	DBpedia	[46, Ex 5.3.3]
FO_3	Seek students taking courses of faculty advisors who are not full professors	dlvhex[DL]	LUBM	[46, Ex 5.4.2]
FO_4	Transitive closure of the organization hierarchy fed to the DL KB	dlvhex[DL]	LUBM	[46, Ex 5.4.3]

Table 2: Benchmark Results for FO-Rewritable DL-Programs (Runtime in secs)

(a) Benchmark FO_1

Instance	MOR	DLV$^{\text{DB}}$	DLV
$R_{10\text{k}}$	<1	<1	1
$R_{100\text{k}}$	1	1	105
$R_{250\text{k}}$	3	4	977
$R_{500\text{k}}$	5	9	2,795
$R_{1\text{M}}$	11	19	11,446

(b) Benchmark FO_2

Instance	MOR	DLV$^{\text{DB}}$	dlvhex[DL]
$D_{10\text{k}}$	1	<1	7
$D_{100\text{k}}$	4	6	—
$D_{250\text{k}}$	9	25	—
$D_{500\text{k}}$	18	50	—
$D_{1\text{M}}$	42	145	—

(c) Benchmark FO_3

Instance	MOR	dlvhex[DL]
$U_{10\text{k}}$	1	36
$U_{100\text{k}}$	4	117
$U_{250\text{k}}$	11	—
$U_{500\text{k}}$	20	—
$U_{1\text{M}}$	44	—

(d) Benchmark FO_4

Instance	MOR	dlvhex[DL]
$U_{10\text{k}}$	1	35
$U_{100\text{k}}$	1	108
$U_{250\text{k}}$	2	—
$U_{500\text{k}}$	4	—
$U_{1\text{M}}$	11	—

PostgreSQL 8.4 database, with increased `shared_buffers` and `work_mem` parameters to utilize the available RAM. For each benchmark, the average of five runs was calculated, having a timeout of 6 hours, and a memout of 14.7 GB for each run.

The results are shown in Table 2; the entries with "—" (*out of memory*) for dlvhex are due to RacerPro's usage of the AllegroGraph library, which limited the instance size. (The optimized RacerPro 2.0 release is not available with dlvhex at present.) The results let observe a linear runtime behavior of MOR for all benchmarks except FO_4. The use of indexes in the database might lead to performance gains. Owlgres creates default indexes during KB loading, but none exploiting the particular dl-program structure. In case of FO_1 (Table 2a, MOR and DLV$^{\text{DB}}$ scale similarly, but the use of views is faster than materialization as applied in the standard version of DLV$^{\text{DB}}$. On the other hand, the rule rewriting of DLV$^{\text{DB}}$ appears to be effective. In case of FO_2 (Table 2b), the temporary update of the DL knowledge base did not hit much on MOR's performance and also here quasi-linear runtime was achieved; DLV$^{\text{DB}}$ also scales well but at a more increasing pace. Further details are available on the benchmark webpage.[3]

3.3 Limited Recursion

FO rewritability is mainly motivated by its link to RDBMS and SQL, which allows for exploiting efficient database engines. The SQL:1999 standard, however, also foresees a limited form of linear recursion in queries, such that, e.g., the transitive closure of a base relation is definable. The respective classic rule-based definition for base relation a

$$p(X,Y) \leftarrow a(X,Y). \qquad\qquad p(X,Y) \leftarrow a(X,Z),\ p(Z,Y).$$

is written in SQL as

WITH RECURSIVE p AS
(SELECT $*$ FROM a UNION SELECT a.1 p.2 FROM a, p WHERE $a.2 = p.1$)
SELECT $*$ FROM p

To exploit SQL:1999 linear recursion features for dl-programs, we impose in addition to linearity the following syntactic restriction: (1) no predicate depends negatively on itself (i.e., negation is stratified), and (2) no predicate cycles through some dl-atom; the latter still allows recursive predicates as input of dl-atoms. Informally, this supports linear stratified negation with non-recursive DL KB access.

Related to the experiments above, we performed benchmark FO_4 based on the LUBM ontology U_n and a linear recursive dl-program. As DL-Lite$_R$ misses transitive roles, we calculate the transitive closure of the sparse, tree-like structure of the $subOrganization$ role (thus the organizational hierarchy of the LUBM university) in the rules part and update the DL KB. We observed in the results (in Table 2d) an indication for quadratic runtime. We also point to another experiment (see [46, Ex. 5.2.3]), where randomly generated, cyclic data led to irregular runtime; in one case, runs with $U_{50\mathrm{k}}$ were faster than with $U_{25\mathrm{k}}$ due to depth and cycles. We further encountered a limitation in Postgres' recursive query implementation on cyclic data. As it iterates joins without cycle detection,[5] queries even may not terminate. To avoid this, we used Postgres's LIMIT parameter to safely bound the iterations.

4 Datalog$^\neg$ Rewritability

In this section, we consider uniform evaluation of dl-programs by rewriting to Datalog$^\neg$, which basically are dl-programs as in Section 2.2 but without dl-atoms. Positive such programs are known as *plain* Datalog and have a canonical semantics, while under negation different semantics (answer sets, well-founded atoms) are widely used. Datalog$^\neg$ is more expressive than FO logic, and compared with FO rewriting, we do not pose any restriction in the rule part, thus we can naturally use recursive rules. The ontologies in dl-programs can also be "inlined" into some Datalog$^\neg$ programs in a modular way.

4.1 DL Datalog Rewritability and Inline Evaluation

For Datalog$^\neg$ rewritability of dl-programs, we need a suitable notion of rewritability of a DL knowledge base. To this end, [28] defined a description logic $\mathcal{DL}$ to be Datalog-*rewritable*, if there exists a transformation $\Phi_{\mathcal{DL}}$ from $\mathcal{DL}$ KBs to Datalog programs such that, for every $\mathcal{DL}$ KB L,

[5] http://archives.postgresql.org/pgsql-hackers/2008-02/msg00642.php

(i) $L \models Q(\mathbf{o})$ iff $\Phi_{\mathcal{DL}}(L) \models Q(\mathbf{o})$ for any concept or role name Q from L, and individuals $\mathbf{o}$ from L;
(ii) $\Phi_{\mathcal{DL}}$ is *modular*, i.e., for $L = \langle \mathcal{T}, \mathcal{A} \rangle$ where $\mathcal{T}$ is a TBox and $\mathcal{A}$ an ABox, $\Phi_{\mathcal{DL}}(L) = \Phi_{\mathcal{DL}}(\mathcal{T}) \cup \mathcal{A}$;

DL-programs $KB = (L, P)$ over Datalog-rewritable DLs can be readily transformed to Datalog$^\neg$ programs. Let $\Lambda_P = \{\lambda \mid DL[\lambda; Q] \text{ occurs in } P\}$ be the set of all input lists of dl-atoms appearing in P. The transformation $\Psi(KB)$ of a dl-program KB is then defined as $\Phi_{\mathcal{DL}}(L_{\Lambda_P}) \cup P^{ord} \cup \rho(\Lambda_P) \cup T_P$, where

- $L_{\Lambda_P} = \bigcup_{\lambda \in \Lambda_P} L_\lambda$, where L_λ is L with all concept and role names subscripted with λ. Intuitively, each input signature of a dl-atom in P will influence L differently. As we want to cater for these influences in one program, we have to differentiate between the KBs with different inputs;
- $\rho(\Lambda_P)$ is a Datalog program that contains for each $\lambda = S_1 \uplus p_1, \ldots, S_m \uplus p_m \in \Lambda_P$ the rules $S_{i\lambda}(\mathbf{X_i}) \leftarrow p_i(\mathbf{X_i})$, $1 \leq i \leq m$, where the arity of $\mathbf{X_i}$ matches the one of S_i. Intuitively, we add the extension of p_i to the appropriate concept or role;
- P^{ord} is P with each dl-atom $DL[\lambda; Q](\mathbf{t})$ replaced by a new atom $Q_\lambda(\mathbf{t})$;
- T_P consists of Datalog facts $\top(a)$ and $\top^2(a, b)$ for all a, b in the Herbrand domain of P to ensure their introduction in L.

Example 4. Consider $KB' = (L, P')$, which replaces the last rule of P in Example 1 by $s(X) \leftarrow DL[C \uplus p; D](X), not\ DL[C \uplus q; D](X)$. Here, L is rather simple. We can transform L_{Λ_P} to $\Phi(L_{\Lambda_P}) = \{D_{\lambda_1}(X) \leftarrow C_{\lambda_1}(X);\ D_{\lambda_2}(X) \leftarrow C_{\lambda_2}(X)\}$. Then $\Lambda_P = \{\lambda_1 = C \uplus p, \lambda_2 = C \uplus q\}$ and $\rho(\Lambda_P) = \{C_{\lambda_1}(X) \leftarrow p(X);\ C_{\lambda_2}(X) \leftarrow q(X)\}$. Program P^{ord} consists of all facts of P' and the rule $s(X) \leftarrow D_{\lambda_1}(X), not\ D_{\lambda_2}(X)$. Finally, we add $T_P = \{\top(o) \mid o \in \{a, b, c\}\} \cup \{\top^2(o_1, o_2) \mid \{o_1, o_2\} \subset \{a, b, c\}\}$.

The following result allows us to reduce reasoning from dl-programs to Datalog$^\neg$ under well-founded semantics.

Theorem 3 ([28]). *Let $KB = (L, P)$ be a dl-program over a* Datalog*-rewritable DL and $a \in HB_P$. Then, $KB \models_{wf} a$ iff $\Psi(KB) \models_{wf} a$.*

A similar result holds for answer set semantics.

Theorem 4. *Let KB be a dl-program over a* Datalog*-rewritable DL. Then the answer sets of KB correspond 1-1 to the answer sets of $\Psi(KB)$, such that (i) every answer set of KB is expendable to an answer set of $\Psi(KB)$; and (ii) for every answer set J of $\Psi(KB)$, its restriction $I = J\,|_{HB_P}$ to HB_P is an answer set of KB.*

The reduction from dl-programs to Datalog$^\neg$ above only considers the operation $\uplus$ and positive dl-queries. Negative dl-queries can be reduced to inconsistency checking (see Sec. 4.3 for a discussion). Next we consider concrete datalog rewritable DLs.

4.2 Inline Evaluation over $\mathcal{DL}^+$ Ontologies

The description logic $\mathcal{DL}^+$ [28, 55] is designed as a lightweight ontology language which is expressive enough to capture many real life ontologies. It imposes syntactic restrictions on axioms $\alpha \sqsubseteq \beta$, distinguishing between the "body" α and the "head" β, shown in Table 3a.

An $\mathcal{DL}^+$ ontology is a pair $L = \langle \mathcal{T}, \mathcal{A} \rangle$ of a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$, where

Table 3: Syntax of $\mathcal{LDL}^+$ and $\mathcal{SROEL}(\sqcap, \times)$

(a) head (h-) and body (b-) restrictions on roles and concepts in $\mathcal{LDL}^+$ axioms

- *h-roles* (h for *head*) S, T are (i) *role names* R, (ii) *role inverses* S^-, (iii) *role conjunctions* $S \sqcap T$, and (iv) *role top* $\top^2$;
- *b-roles* (b for *body*) S, T are the same as h-roles, plus (v) *role disjunctions* $S \sqcup T$, (vi) *role sequences* $S \circ T$, (vii) *transitive closures* S^+, (viii) and *role nominals* $\{(o_1, o_2)\}$, where o_1, o_2 are individuals.
- *basic concepts* C, D are concept names A, $\top$, and conjunctions $C \sqcap D$;
- *h-concepts* are (i) *basic concepts* B, and (ii) *value restrictions* $\forall S.B$ where S is a *b-role*;
- *b-concepts* C, D are (i) *basic concepts* B, (ii) *disjunctions* $C \sqcup D$, (iii) *exists restrictions* $\exists S.C$, (iv) *atleast restrictions* $\geq nS.C$, and (v) *nominals* $\{o\}$, where S is a b-role, and o is an individual.

(b) Syntax of $\mathcal{SROEL}(\sqcap, \times)$

- Concept constructors are (i) top $\top$, (ii) bottom $\bot$, (iii) conjunction $C \sqcap D$, (iv) existential restriction $\exists R.C$, (v) nominal $\{a\}$;
- Axioms are (i) concept assertion $C(a)$, (ii) role assertion $R(a, b)$, (iii) concept inclusion (GCI) $C \sqsubseteq D$, (iv) role inclusion $R \sqsubseteq T$, (v) generalized role inclusion $R \circ S \sqsubseteq T$, (vi) role conjunction $S_1 \sqcap S_2 \sqsubseteq T$, (vii) concept production $C \times D \sqsubseteq T$, $R \sqsubseteq C \times D$.

- $\mathcal{T}$ is a set of *terminological axioms* $B \sqsubseteq H$, where B is a b-concept and H is an h-concept, and *role axioms* $S \sqsubseteq T$, where S is a b-role and T is an h-role, and
- $\mathcal{A}$ is a set of assertions of the form $C(o)$ and $S(o_1, o_2)$ where C is an h-concept and S is an h-role.

Essentially, $\mathcal{LDL}^+$ extends OWL 2 RL [40] with singleton nominals, role conjunctions, and transitive closure. Designing a transformation $\Phi_{\mathcal{LDL}^+}$ of $\mathcal{LDL}^+$ into Datalog is straightforward; each TBox (ABox) axiom is transformed into an equivalent Datalog rule (fact), similarly as in [25]. For example, if $Person \sqcap \exists headOf.Dept \sqsubseteq Chair \in L$, then we add a rule $Chair(X) \leftarrow Person(X), headOf(X, Y), Dept(Y)$ to the Datalog program $\Phi_{\mathcal{LDL}^+}(L)$. For details of the rewriting, please see [28].

4.3 Inline Evaluation over EL Ontologies

We consider the DL $\mathcal{SROEL}(\sqcap, \times)$ [37], whose syntax is given in Table 3b; it is a superset of OWL 2 EL [40] disregarding datatypes, and adds concept production, which can be seen as a generalization of domain and role restriction.

Krötzsch [37] shows a Datalog encoding for $\mathcal{SROEL}(\sqcap, \times)$ describing a proof system. Every TBox axiom, ABox axiom, concept name, role name, and individual is transformed to a *fact* by an input translation I_{inst}. A fixed set P_{inst} contains the derivation rules, which are independent of the concrete $\mathcal{SROEL}(\sqcap, \times)$ ontology (see [37]).

In the following, for simplicity, when we say $\mathcal{EL}$, we always mean $\mathcal{SROEL}(\sqcap, \times)$. For an $\mathcal{EL}$ ontology L, define a Datalog transformation by

$$\Phi_{\mathcal{EL}}(L) = P_{inst} \cup \{I_{inst}(\alpha) \mid \alpha \in L\} \cup \{I_{inst}(s) \mid s \in N_I \cup N_C \cup N_R\} \ .$$

Theorem 5 ([37]). *Given an $\mathcal{EL}$ ontology L, the transformation $\Phi_{\mathcal{EL}}$ is sound and complete w.r.t. instance checking, i.e., (i) $L \models C(a)$ iff $\Phi_{\mathcal{EL}}(L) \models isa(a, C)$, and (ii) $L \models R(a, b)$ iff $\Phi_{\mathcal{EL}}(L) \models triple(a, R, b)$.*

Example 5. Let $L_1 = \{A(a),\ A \sqsubseteq \exists R.B,\ B \sqsubseteq C,\ \exists R.C \sqsubseteq D\}$, and suppose we want to decide $L_1 \models D(a)$. The axioms of L_1 and the signatures N_I, N_C, and N_R are transformed to facts in $\Phi_{\mathcal{EL}}(L_1)$:

$$\left\{ \begin{array}{l} isa(a,A);\ supEx(A,R,B,e^{A \sqsubseteq \exists R.B});\ subClass(B,C);\ subEx(R,C,D); \\ nom(a);\ cls(A);\ cls(B);\ cls(C);\ cls(D);\ rol(R) \end{array} \right\} .$$

Then, P^{inst} is added to $\Phi_{\mathcal{EL}}(L_1)$; in particular, also the rules

$$\begin{aligned} isa(X,X) &\leftarrow nom(X) \\ isa(X,Z) &\leftarrow subClass(Y,Z), isa(X,Y) \\ isa(X_1,Z) &\leftarrow subEx(V,Y,Z), triple(X_1,V,X_2), isa(X_2,Y) \\ triple(X_1,V,X_2) &\leftarrow supEx(Y,V,Z,X_2), isa(X_1,Y) \\ isa(X_2,Z) &\leftarrow supEx(Y,V,Z,X_2), isa(X_1,Y) \end{aligned}$$

From these rules and the above facts, $isa(a,D)$ is derivable, and thus $\Phi_{\mathcal{EL}}(L_1) \models D(a)$.

Note that strictly, $\Phi_{\mathcal{EL}}(L)$ is not a datalog rewriting as defined above. The mismatch is that ABox assertions (e.g., $C(a)$) are transformed into reified versions (e.g., $isa(a,C)$); this is easily fixed by using reification rules

$$\begin{aligned} P^{re} = \{&C(X) \leftarrow isa(X,C); \quad isa(X,C) \leftarrow C(X) \mid C \in N_C\} \cup \\ &\{R(X,Y) \leftarrow triple(X,r,Y); \quad triple(X,R,Y) \leftarrow r(X,Y) \mid R \in N_R\} \ . \end{aligned}$$

Then $\Phi'_{\mathcal{EL}}(L) = (\Phi_{\mathcal{EL}}(L) \setminus \{I_{inst}(\alpha) \mid \alpha \in \mathcal{A}\}) \cup P^{re} \cup \mathcal{A}$ is a proper Datalog rewriting. However, in the following, for convenience, we will use $\Phi_{\mathcal{EL}}$, instead of $\Phi'_{\mathcal{EL}}$.

Negative dl-Queries. Trivially, $L \models \neg C(a)$ is equivalent to unsatisfiability of $L \cup \{C(a)\}$. To answer a negative query $\neg C(X)$, we need to bind X to every possible individual, and reduce it to unsatisfiability checking. This one by one checking can be elegantly achieved via datalog encoding. The idea is to extend $isa/2$ with two more arguments, representing the individual and the concept name, to $isa\_n/4$; in P_{inst}, each isa is uniformly replaced with isa_n, and each $triple$ uniformly with $triple\_n$, yielding $P^{\neg}_{inst}$. This set includes e.g. the following rules, which propagate subclass and conjunctive subclass membership:

$$\begin{aligned} isa\_n(X,Z,C,J) &\leftarrow subClass(Y,Z), isa\_n(X,Y,C,J) \\ isa\_n(X,Z,C,J) &\leftarrow subConj(Y_1,Y_2,Z), isa\_n(X,Y_1,C,J), isa\_n(X,Y_2,C,J) \end{aligned}$$

The individual unsatisfiability checks are then accomplished with rules $P^{\neg}$, which for each check add $C(a)$, expand all $isa(X,Y)$ atoms with a and C, and make an atom $isnota(a,C)$ true iff the test is successful:

$$\begin{aligned} isa\_n(X,Y,Y,X) &\leftarrow nom(X), cls(Y) \\ isa\_n(X_1,Y_1,Y_2,X_2) &\leftarrow isa(X_1,Y_1), cls(Y_2), nom(X_2) \\ isnota(X,Y) &\leftarrow isa\_n(N,Z,Y,X), nom(N), bot(Z), nom(X) \end{aligned}$$

Let the extended $\mathcal{EL}$ reduction for negative query of an $\mathcal{EL}$ ontology L be defined as $\Phi^{\neg}_{\mathcal{EL}}(L) = P^{\neg}_{inst} \cup \{I_{inst}(\alpha) \mid \alpha \in L\} \cup \{I_{inst}(s) \mid s \in N_I \cup N_C \cup N_R\} \cup P^{\neg}$.

Proposition 1. *For every $\mathcal{EL}$ ontology L, $L \models \neg C(a)$ iff $\Phi^{\neg}_{\mathcal{EL}}(L) \models isnota(a,C)$.*

An application of dl-programs where negative dl-queries are important is terminological default logic over DLs [5, 11]. Here, Reiter-style default rules are applied to named individuals; we show the classic birds&penguins example for illustration.

Example 6. The knowledge base $\Delta = \langle L, D\rangle$ consists of a DL KB $L = \{Flier \sqcap NonFlier \sqsubseteq \bot, Penguin \sqsubseteq Bird,\ Penguin \sqsubseteq NonFlier,\ Bird(tweety)\}$ (which is in $\mathcal{EL}$), and a (singleton) set $D = \{Bird(X) : Flier(X)/Flier(X)\}$ of default rules (informally, birds fly by default). The semantics of the KB Δ is captured by the following dl-program $KB = (L, \Pi(D))$ under answer set semantics (i.e., default extensions correspond to answer sets), where $\Pi(D)$ is

$$
\begin{aligned}
in\_Flier(X) &\leftarrow not\ out\_Flier(X)\\
out\_Flier(X) &\leftarrow not\ in\_Flier(X)\\
Flier^{+}(X) &\leftarrow DL[\lambda; Bird](X), not\ \ DL[\lambda'; \neg Flier](X)\\
fail &\leftarrow DL[\lambda'; Flier](X), out\_Flier(X), not\ fail\\
fail &\leftarrow DL[\lambda; Flier](X), in\_Flier(X), not\ fail\\
fail &\leftarrow DL[\lambda; Flier](X), out\_Flier(X), not\ fail
\end{aligned}
$$

where $\lambda = \{Flier \uplus in\_Flier\}$ and $\lambda' = \{Flier \uplus Flier^{+}\}$. To rewrite KB to Datalog$^{\neg}$, we use for the dl-atom $DL[\lambda'; \neg Flier](X)$ the rewriting $\Phi^{\neg}_{\mathcal{EL}}(L_{\lambda'}) \cup \rho(\lambda') \models isnota_{\lambda'}(X, Flier)$, and for the other dl-atoms the rewriting $\Phi_{\mathcal{EL}}(L_\gamma) \cup \rho(\gamma) \models isa_\gamma(X, Flier)$, $\gamma \in \{\lambda, \lambda'\}$. The Datalog$^{\neg}$ program has a single answer set, which contains $Flier(tweety)$, as expected.

4.4 Implementation and Experiments

We implemented the datalog rewriting of dl-programs in the DReW reasoner. Initial results over $\mathcal{LDL}^{+}$ ontologies were given in [56], which compares DReW to dlvhex[DL]. The results clearly show that DReW outperforms the latter. For the present paper, we extended the rewriter to the $\mathcal{EL}$ encoding given above. Thus, DReW can now be used as a DL-reasoner and a dl-program-reasoner for $\mathcal{LDL}^{+}$ and $\mathcal{EL}$ ontologies.

The experiments have been run on an Ubuntu Linux 11.10 system on an AMD Opteron Magny-Cours 6176 SE 2.3GHz system with 24 cores and 128GB RAM. Further details are given on the benchmark webpage.[3]

Instance retrieval With Large $\mathcal{EL}$ TBoxes. The experiment shows the efficiency of the $\mathcal{EL}$ datalog rewriting for a large TBox. As test ontology we used an $\mathcal{EL}$ variant of Galen,[6] a large biomedical ontology.[7] In the test, we created four ontology instances G_1 to G_4 that have a fixed Galen TBox and increasing ABoxes with $10i$ assertions, each using roughly ten concepts and roles. We performed four instance retrieval queries over G_i: (i) $q_1(X) = Substance(X)$, (ii) $q_2(X) = Animal(X)$, (iii) $q_3(X) = MaleAdult(X)$, and (iv) $q_4(X) = Human(X)$.

We were using two Datalog engines for computing the models of the DReW encodings: clingo 3.0.3 [22] and DLV 2010-10-14 [38]. We compared DReW with the DL reasoners HermiT 1.3.5 [42] and Pellet 2.3.0 [47]. We differentiate two DReW benchmark run settings: (i) DReW[clingo] and (ii) DReW[DLV] (in brackets the employed model builder is given). The results are shown in Table 4a (the results for Pellet are omitted as it timed out in all tests).

DReW is superior to HermiT and Pellet; even with small ABoxes, the general purpose DL-reasoner Pellet could not answer the queries within one hour. Note that

[6] http://condor-reasoner.googlecode.com/files/EL-GALEN.owl

[7] http://www.opengalen.org/

Table 4: Benchmark Results for Datalog Rewriting (Runtime in secs)

(a) $\mathcal{EL}$ Instance Retrieval with Galen (HT=HermiT; Pellet always timed out after one hour)

Ontology		DReW		HT	Ontology		DReW		HT
Query		[DLV]	[clingo]		Query		[DLV]	[clingo]	
G_1	q_1	2.0	1.3	8.1	G_3	q_1	2.0	1.3	9.5
	q_2	2.0	1.3	8.2		q_2	2.1	1.4	9.5
	q_3	2.0	1.3	8.		q_3	2.0	1.4	9.7
	q_4	2.0	1.4	8.1		q_4	2.1	1.4	9.5
G_2	q_1	2.0	1.3	8.9	G_4	q_1	2.1	1.4	10.3
	q_2	2.0	1.4	8.9		q_2	2.1	1.4	10.2
	q_3	2.1	1.4	8.7		q_3	2.1	1.4	10.2
	q_4	2.0	1.3	9.0		q_4	2.1	1.4	10.2

(b) Default reasoning with Policy Benchmark

KB	Typing	DReW [DLV]	[clingo]
Δ_1	5	1.1	0.8
	50	2.4	1.3
	100	6.0	3.0
Δ_5	5	6.6	4.4
	50	8.3	5.0
	100	12.2	7.4
Δ_{10}	5	13.9	9.4
	50	15.7	10.1
	100	20.5	13.3
Δ_{25}	5	35.8	26.0
	50	40.0	26.4
	100	43.7	32.7

Fig. 1: Access policy control $\Delta = (L, D)$, $L = (\mathcal{T}, \mathcal{A})$, in terminological default logic

$$\mathcal{T} = \left\{ \begin{array}{l} \mathit{Staff} \sqsubseteq \mathit{User}, \quad \mathit{Blacklisted} \sqsubseteq \mathit{Staff}, \quad \mathit{Deny} \sqcap \mathit{Grant} \sqsubseteq \bot, \\ \mathit{UserRequest} \equiv \exists \mathit{hasAction}.\mathit{Action} \sqcap \exists \mathit{hasSubject}.\mathit{User} \sqcap \exists \mathit{hasTarget}.\mathit{Project}, \\ \mathit{StaffRequest} \equiv \exists \mathit{hasAction}.\mathit{Action} \sqcap \exists \mathit{hasSubject}.\mathit{Staff} \sqcap \exists \mathit{hasTarget}.\mathit{Project}, \\ \mathit{BlacklistedStaffRequest} \equiv \mathit{StaffRequest} \sqcap \exists \mathit{hasSubject}.\mathit{Blacklisted} \end{array} \right\}$$

$$D = \left\{ \begin{array}{l} \mathit{UserRequest}(X) : \mathit{Deny}(X) / \mathit{Deny}(X), \\ \mathit{StaffRequest}(X) : \neg \mathit{BlacklistedStaffRequest}(X) / \mathit{Grant}(X), \\ \mathit{BlacklistedStaffRequest}(X) : \top / \mathit{Deny}(X) \end{array} \right\}$$

other reasoners like CB [33] and ELK [34] classify EL-Galen very fast, but they can not be used for instance retrieval out of the box.

Default Reasoning with $\mathcal{EL}$ Policy Ontology. This experiment was conducted on terminological default KBs with an $\mathcal{EL}$ ontology. As shown in Example 6, default reasoning can be encoded into dl-programs with recursive and unstratified rules, which are not easy to handle. For this example, dlvhex[DL] scales exponentially with the number of birds, while DReW, on top of DLV as well as clingo, runs much faster.

As a more interesting benchmark, we consider here an access control policy, borrowed from [6] and couched into a terminological default KB $\Delta = \langle L, D \rangle$, where the the TBox of L and the defaults D are shown in Figure 1; informally, D expresses that users normally are denied access to files, staff is normally granted access to files, while to blacklisted staff any access is denied.

In the test, we created ontology instances L_i, $i \in \{1, 5, 10, 25\}$, that have a fixed TBox and increasing Aboxes with $i{*}1000$ instances of user requests.

The query imposed was then whether a set of particular individuals, designated by concepts Q_k, $k \in \{5, 50, 100\}$, are granted access (under answer set semantics); the application of defaults, using Q_k as a typing concepts, is thus restricted to the k queried

individuals. As we see in Table 4b, DReW scales sublinearly in this experiment, on top of both DLV and clingo.

5 Modular ASP

In the transformations of dl-programs to Datalog$^\neg$ above, dl-atoms are encoded by subprograms emulating their evaluation. We can make the modular structure of the programs explicitly visible, if we use an extension of Datalog$^\neg$ which caters for structured programming, e.g., [32]. In modular nonmonotonic logic programs [9], program modules can be defined akin to modules or procedures in common programming paradigms, such that on input of suitable predicates to a module, the valuation of "output" predicates in an answer set of a module is obtained based on a call-by-value mechanism.

For example, a module $kernel[e]$ may compute, in each answer set, a kernel of a graph stored by its edges in a binary predicate e, in a predicate k; here e is a formal parameter, which in a module call must be replaced by a predicate name. E.g., $kernel[my\_e].k(a)$ asks if the node a is in the (nondeterministically) computed kernel k of the concrete graph my_e.

A *modular logic program* (MLP) is a collection $\mathbf{P} = (P_1, \ldots, P_n)$ of modules, where each P_i has a list $\boldsymbol{q}_i = q_{i,1}, \ldots, q_{i,n_i}$ of $n_i \geq 0$ formal input parameters; one module (say P_1) is the main module and has $n_0 = 0$. The "implementations" of the P_i consist of logic program rules of form (2), where the b_i may be atoms or *call atoms* of the form $P_j[\boldsymbol{p}].a'$, where $\boldsymbol{p}$ is a list of predicate names matching $\boldsymbol{q}$ and a' an atom. Answer sets of a MLP are defined in [9] in a natural way, generalizing the answer set semantics of ordinary programs.

We will briefly discuss how the Datalog$^\neg$ transformation above can be expressed as MLPs, and examine possible tradeoffs between code repetition and performance, based on a prototype implementation for an MLP fragment.

Inline evaluation of Datalog-rewritable dl-programs. To exemplify the rewriting, consider Example 4 above. We can encode $KB = (L, P')$ as an MLP $\mathbf{P} = (P_1[], P_{DL}[C])$, where P_1 is the main module that consists of the rules

$$p(a) \leftarrow \quad p(b) \leftarrow \quad q(c) \leftarrow$$
$$s(X) \leftarrow P_{DL}[p].D(X), \mathit{not}\ P_{DL}[q].D(X) \qquad (r_s)$$

and $P_{DL}[C]$ is a library module consisting simply of the rules $T_P \cup \{D(X) \leftarrow C(X)\}$.

On the surface, the semantics for MLPs creates for the two input lists p and q of our module atoms of P_1 (among others) two instantiations of $P_{DL}[C]$ on-the-fly; the first instantiation for module atom $P_{DL}[p].D(X)$ contains the rules of $P_{DL}[C]$ and the input facts $F_1 = \{C(a), C(b)\}$, and the other instantiation for module atom $P_{DL}[q].D(X)$ has the rules of $P_{DL}[C]$ and input fact $F_2 = \{C(c)\}$. The input F_1 stems from adding the extension of p as C's to $P_{DL}[C]$, while F_2 has been created from the extension of q.

Now the module atom $P_{DL}[p].D(X)$ is retrieving all instances of concept D from $L \cup F_1$ in the instantiation with input F_1. Similarly, $P_{DL}[q].D(X)$ retrieves all instances of D from $L \cup F_2$ in the instantiation with input F_2. The former is true for $P_{DL}[p].D(a)$ and $P_{DL}[p].D(b)$, as we add both $C(a)$ and $C(b)$ to the module, thus the rule $D(X) \leftarrow C(X)$ fires for $X \in \{a, b\}$, simulating the concept inclusion $C \sqsubseteq D$ of L. The ground

Table 5: Benchmark dl-programs DReW vs. TD-MLP (Runtime in secs)

(a) Ontology U_1

Program	DReW		TD-MLP	
	[clingo]	[DLV]	[clingo]	[DLV]
P_0	0.31	0.45	1.98	2.88
P_1	0.32	0.44	1.69	2.47
P_2	0.32	0.44	2.63	3.82
P_3	0.31	0.43	1.66	2.42
P_4	0.32	0.45	2.45	3.63
P_5	0.61	0.86	1.66	2.46
P_6	1.79	2.76	5.65	8.41
P_7	2.70	4.30	4.87	7.84
P_8	2.76	4.26	9.70	14.12
P_9	2.73	4.31	8.04	11.60

(b) Ontology U_{15}

Program	DReW		TD-MLP	
	[clingo]	[DLV]	[clingo]	[DLV]
P_0	6.49	10.27	30.43	42.53
P_1	4.00	6.27	21.22	30.12
P_2	3.95	6.08	32.65	45.24
P_3	3.98	6.13	20.94	30.33
P_4	4.15	6.43	28.19	39.93
P_5	7.97	12.66	21.54	30.87
P_6	23.52	40.56	72.86	103.76
P_7	36.33	64.05	115.03	162.21
P_8	36.58	61.71	128.01	181.41
P_9	35.26	62.19	108.38	145.04

rules with module atom $P_{DL}[q].D(a)$ and $P_{DL}[q].D(b)$ are false on the other hand, as F_2 only adds $C(c)$ and $D(a)$ and $D(b)$ cannot be derived. Hence, both $not\ P_{DL}[q].D(a)$ and $not\ P_{DL}[q].D(a)$ are true and our rule r_s derives $s(a)$ and $s(b)$ in module $P_1[]$.

Implementation and experimental results. The algorithm for solving input-call stratified MLPs [10] has been implemented in the TD-MLP solver [54], which is based on the dlvhex system. Using this solver, we could perform initial experiments with the modular datalog encoding sketched in this section. The hardware specification for the experiments are the same as in Section 4.4.

We were testing with dl-programs $KB = (U_i, P_j)$, where U_i ($i \in \{1, 15\}$) are $\mathcal{EL}$ versions of the LUBM ontology [26] and program P_j ($j \in \{0, \ldots, 9\}$) encode variants of the LUBM queries; all dl-programs where acyclic. We denote with U_i the LUBM ontology instance that incorporates i universities in the ABox. The original LUBM is not fully in $\mathcal{EL}$ (inverse roles and datatypes are not part of $\mathcal{EL}$): there are 2 violating axioms in the TBox, and 2857 (33154) ABox axioms with datatype are violated in U_1 (U_{15}). The resulting $\mathcal{EL}$ version of LUBM then contains 86 TBox axioms using 43 concepts and 25 roles, and 5738 (67691) ABox axioms with 1555 (17174) individuals in instance U_1 (U_{15}). The rules are from the DReW LUBM benchmark queries and consist of ten programs P_0–P_9. They can be split into two categories: (C1) P_0–P_4 have between 2 and 5 dl-atoms but no input list, while (C2) P_5–P_9 have between 2 and 9 dl-atoms, each with distinct input list.

We used two Datalog engines to compute the models of the native and the modular encodings: clingo 3.0.3 [22] and DLV 2010-10-14 [38]. We compared TD-MLP to DReW using four benchmark run settings (the systems in square brackets denote the model builders used to calculate the model): (i) DReW[clingo], (ii) DReW[DLV], (iii) TD-MLP[clingo], and (iv) TD-MLP[DLV]. The test results are shown in Table 5a for $KB = (U_1, P_j)$ and in Table 5b for $KB = (U_{15}, P_j)$. More details are available on our benchmark webpage.[3]

DReW outperforms TD-MLP in all tests. But DReW's lead is shrinking if we increase the number of dl-atoms in the dl-programs from category (C2), i.e., (U_i, P_5)–(U_i, P_9). The reason is that with DReW we create copies of the ontology as Datalog rules for every dl-atom upfront, thus creating a large single Datalog program. In TD-MLP, we can always use a single copy of the rewritten ontology and let the MLP semantics create the copies for us. As the current TD-MLP implementation is not sophisticated enough, the overhead for instantiating modules during evaluation is prevalent.

6 Discussion and Conclusion

The experimental implementations that we used are not optimized, and in order to get a clearer picture, a more extensive experimentation is necessary. However, the results above show that the uniform evaluation approach has great potential for dl-programs, and that it outperforms a reasoner-coupling implementation substantially. Regarding the various formalisms that we considered, we make the following observations.

FO-Rewritability. The notion of FO-rewritability in Section 3 may be relaxed to permit modifications of the ABox, in spirit of the combined approach [36], where the ABox is extended dependent on the TBox but independent of the query, which is rewritten to a FO query over the extended ABox. For (acyclic) dl-programs to stay within FO, we only can have ABox extensions which do not depend on the contents of the ABox, as its precise contents in evaluating a dl-atom under updates is not known a priori. This still leaves the possibility for auxiliary relations that depend on the TBox and the original ABox (e.g., a linear ordering of the individuals, or arithmetic operations). Such generalized FO-rewritability increases the expressiveness in general.

By exploiting recursive query processing in SQL, uniform evaluation of dl-programs with DL-Lite$_R$ allows for performing computations like transitive closure on top of a knowledge base. However, in current RDBMS, recursive query processing is still not very advanced and has lots of room for improvement. Our experience with Postgres is in analogy to tests with commercial RDBMS (cf. [49]). A useful improvement of Postgres in this regard would be efficient cycle detection, eliminating the LIMIT parameter.[8] To cater for (also non-linear) recursion, languages like Datalog seem more attractive.

Datalog$^\neg$-Rewritability. Similar as for FO-rewritability, we also can consider here relaxed notions of Datalog-rewritability that allow for auxiliary relations. For Datalog$^\neg$ rewritability of a dl-program, we required that dl-atoms resp. the underlying description logic are Datalog rewritable. One also could consider a more liberal notion of Datalog$^\neg$ rewritability of dl-atoms, which however would need to deal with the fact that in principle, the program $\Phi_{\mathcal{DL}}(L)$ could have multiple answer sets (or none). Furthermore, simply plugging in $\Phi_{\mathcal{DL}}(L_\lambda)$ for some dl-atom $DL[\lambda, Q](\boldsymbol{t})$ may lead to unwanted effects, due to nonmonotonicity; e.g., additional answer sets might emerge. Syntactic restrictions (e.g., acyclicity, or more general that dl-atoms are not involved in cycles) can avoid this.

Modular Logic Programs. The modular program encoding lags in total runtime behind the ad hoc inlined approach, but it scale at a slower growth rate. This looks promising as the gap for large amounts of data will be closed. In designing modules for dl-atoms, one

[8] http://www.postgresql.org/docs/8.4/static/queries-with.html

has a range of possibilities, from few, more general modules to many but very specialized ones (in the extremal case, a single universal module for all dl-atoms (similarly as in dlvhex), vs. one module for each dl-atom). Determining the effects of such design choices, and the resulting transformations is an interesting subject fur future study.

6.1 Further Work and Outlook

The uniform evaluation approach we proposed is a flexible framework. Once a description logic has a transformation into the suitable logic, it can be easily integrated.

At the software level, in case of FO a respective component can be integrated in the MOR prototype and in case of Datalog in the DReW reasoner. In particular, (relaxed) Datalog rewritings of description logics apart from $\mathcal{EL}$ have gained attention recently, e.g. [44, 24], including conjunctive query answering in Horn-$\mathcal{SHIQ}$, which subsumes all the OWL 2 profiles EL, RL and QL; an implementation is in progress [50].

We considered some formalisms for uniform evaluation of dl-programs, for which evaluation engines are available. Of course, further such formalisms (e.g., FO($\cdot$) Logic [14], or F-logic [35]) may be considered. But also formalisms for which reasoning engines are yet emerging might be of interest, e.g. Datalog $^{\pm}$ [7]. The latter extends Datalog with existential quantification in rule head and at the same time restricts the syntax such that reasoning remains decidable. Datalog $^{\pm}$ is more expressive than various description logics, including DL-Lite (which is captured elegantly), and allows for handling unknown individuals in the reasoning. It seems to be an attractive formalism in particular to host the combination of rules and ontologies.

Finally, the prototypes we used for experimentation are not optimized, and there is considerable room for improvement. Furthermore, alternative encodings and reductions might be considered, and specific optimization methods and techniques developed.

References

1. Acciarri, A., Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Palmieri, M., Rosati, R.: Quonto: Querying ontologies. In: AAAI'05. pp. 1670–1671 (2005)
2. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The dl-lite family and relations. J. Artif. Intell. Res. 36, 1–69 (2009)
3. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope. In: IJCAI'05. pp. 364–369. (2005)
4. Baader, F., Brandt, S., Lutz, C.: Pushing the $\mathcal{EL}$ envelope further. In: OWLED08-DC (2008)
5. Baader, F., Hollunder, B.: Embedding defaults into terminological knowledge representation formalisms. J. Autom. Reasoning 14(1), 149–180 (1995)
6. Bonatti, P.A., Faella, M., Sauro, L.: Adding default attributes to EL++. In: AAAI'11. (2011)
7. Calì, A., Gottlob, G., Lukasiewicz, T., Marnette, B., Pieris, A.: Datalog+/-: A family of logical knowledge representation and query languages for new applications. In: LICS'10. pp. 228–242. (2010)
8. Calvanese, D., de Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. J Autom. Reasoning 39(3), 385–429 (2007)
9. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Modular nonmonotonic logic programming revisited. In: ICLP'09. pp. 145–159. Springer (2009)

10. Dao-Tran, M., Eiter, T., Fink, M., Krennwallner, T.: Relevance-driven evaluation of modular nonmonotonic logic programs. In: LPNMR'09, pp. 87–100. Springer (2009)
11. Dao-Tran, M., Eiter, T., Krennwallner, T.: Realizing default logic over description logic knowledge bases. In: ECSQARU'09, pp. 602–613. Springer (2009)
12. de Bruijn, J., Bonnard, P., Citeau, H., Dehors, S., Heymans, S., Pührer, J., Eiter, T.: Combinations of rules and ontologies: State-of-the-art survey of issues. Tech. Rep. Ontorule D3.1, Ontorule Project Consortium (2009)
13. de Bruijn, J., Eiter, T., Tompits, H.: Embedding approaches to combining rules and ontologies into autoepistemic logic. In: KR'08, pp. 485–495 (2008)
14. Denecker, M., Ternovska, E.: A logic of non-monotone inductive definitions. ACM Trans. Comput. Log. 9(2), 14, 52pp. (2008)
15. Drabent, W., Eiter, T., Ianni, G., Krennwallner, T., Lukasiewicz, T., Małuszyński, J.: Hybrid reasoning with rules and ontologies. In: Semantic Techniques for the Web: The REWERSE perspective, chap. 1, pp. 1–49. Springer (2009)
16. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R.: Well-founded semantics for description logic programs in the Semantic Web. ACM Trans. Comput. Log. 12(2), 11, 41pp. (2011)
17. Eiter, T., Ianni, G., Lukasiewicz, T., Schindlauer, R., Tompits, H.: Combining answer set programming with description logics for the Semantic Web. Artif. Intell. 172(12-13), 1495–1539 (2008)
18. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: Effective integration of declarative rules with external evaluations for semantic-web reasoning. In: ESWC'06. pp. 273–287. Springer (2006)
19. Eiter, T., Fink, M., Krennwallner, T.: Decomposition of declarative knowledge bases with external functions. In: IJCAI'09, pp. 752–758. (2009)
20. Eiter, T., Ianni, G., Schindlauer, R., Tompits, H.: A uniform integration of higher-order reasoning and external evaluations in answer set programming. In: IJCAI'05, pp. 90–96. (2005)
21. Fink, M., Pearce, D.: A logical semantics for description logic programs. In: JELIA'10. pp. 156–168. Springer (2010)
22. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The Potsdam Answer Set Solving Collection. AI Commun. 24(2), 107–124 (2011)
23. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing 9, 365–385 (1991)
24. Gottlob, G., Schwentick, T.: Rewriting ontological queries into small nonrecursive datalog programs. In: DL'11. CEUR-WS, vol. 745. (2011)
25. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: WWW'03. pp. 48–57. ACM (2003)
26. Guo, Y., Pan, Z., Heflin, J.: Lubm: A benchmark for OWL knowledge base systems. J. Web Sem. 3(2-3), 158 – 182 (2005)
27. Haarslev, V., Hidde, K., Möller, R., Wessel, M.: The RacerPro knowledge representation and reasoning system. In: Semant. Web J. http://www.semantic-web-journal.net/, to appear
28. Heymans, S., Eiter, T., Xiao, G.: Tractable reasoning with DL-programs over datalog-rewritable description logics. In: ECAI'10. IOS Press (2010)
29. Heymans, S., Korf, R., Erdmann, M., Pührer, J., Eiter, T.: Loosely coupling f-logic rules and ontologies. In: WI'10, pp. 248–255. IEEE Computer Society (2010)
30. Hustadt, U., Motik, B., Sattler, U.: Reasoning in description logics by a reduction to disjunctive datalog. J. Autom. Reasoning 39(3), 351–384 (2007)
31. Janhunen, T.: On the intertranslatability of non-monotonic logics. Ann. Math. Artif. Intell. 27(1-4), 79–128 (1999)
32. Janhunen, T., Oikarinen, E., Tompits, H., Woltran, S.: Modularity Aspects of Disjunctive Stable Models. J. Artif. Intell. Res. 35, 813–857 (2009)

33. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In: IJCAI'09. pp. 2040–2045 (2009)
34. Kazakov, Y., Krötzsch, M., Simancik, F.: Concurrent classification of el ontologies. In: ISWC'11, pp. 305–320. Springer (2011)
35. Kifer, M., Lausen, G., Wu, J.: Logical foundations of object-oriented and frame-based languages. J. ACM 42(4), 741–843 (1995)
36. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in dl-lite. In: KR'10. AAAI Press (2010)
37. Krötzsch, M.: Efficient rule-based inferencing for OWL EL. In: IJCAI'11, pp. 2668–2673. (2011)
38. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV System for Knowledge Representation and Reasoning. ACM Trans. Comput. Log. 7(3) (2006)
39. Lukasiewicz, T.: A novel combination of answer set programming with description logics for the semantic web. IEEE Trans. Knowl. Data Eng. 22(11), 1577–1592 (2010)
40. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C. (eds.): OWL 2 Web Ontology Profiles. W3C (2009), W3C Rec. 27 Oct. 2009, http://www.w3.org/TR/owl2-profiles/
41. Motik, B., Rosati, R.: Reconciling Description Logics and Rules. J. ACM 57(5), 1–62 (2010)
42. Motik, B., Shearer, R., Horrocks, I.: Hypertableau Reasoning for Description Logics. J. Artif. Intell. Res. 36, 165–228 (2009)
43. Ordonez, C.: Optimization of linear recursive queries in SQL. IEEE Trans. Knowl. Data Eng. 22(2), 264–277 (2010)
44. Ortiz, M., Rudolph, S., Simkus, M.: Worst-case optimal reasoning for the horn-DL fragments of OWL 1 and 2. In: KR'10. AAAI Press (2010)
45. Rosati, R., Almatelli, A.: Improving query answering over dl-lite ontologies. In: KR'10. AAAI Press (2010)
46. Schneider, P.: Evaluation of description logic programs using an RDBMS. Master's thesis, TU Wien (2010)
47. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. J. Web Sem. 5(2), 51–53 (2007)
48. Stocker, M., Smith, M.: Owlgres: A scalable OWL reasoner. In: OWLED'01 (2008)
49. Terracina, G., Leone, N., Lio, V., Panetta, C.: Experimenting with recursive queries in database and logic programming systems. Theor. Pract. Log. Prog. 8(2), 129–165 (2008)
50. Tran, T.K.: Query answering in the description logic Horn-SHIQ. Master's thesis, TU Wien (2011)
51. Wang, K., Billington, D., Blee, J., Antoniou, G.: Combining description logic and defeasible logic for the semantic web. In: RuleML'04, pp. 170–181. Springer (2004)
52. Wang, Y., You, J.H., Yuan, L.Y., Shen, Y.D.: Loop formulas for description logic programs. Theor. Pract. Log. Prog. 10(4-6), 531–545 (2010)
53. Wang, Y., You, J.H., Yuan, L.Y., Shen, Y.D., Eiter, T.: Embedding description logic programs into default logic. CoRR abs/1111.1486 (2011)
54. Wijaya, T.: Top-Down Evaluation Techniques for Modular Nonmonotonic Logic Programs. Master's thesis, TU Wien (2011), http://media.obvsg.at/AC07811177-2001
55. Xiao, G., Eiter, T.: Inline evaluation of hybrid knowledge bases – PhD description. In: RR'11, pp. 300–305. Springer (2011)
56. Xiao, G., Heymans, S., Eiter, T.: DReW: a reasoner for datalog-rewritable description logics and dl-programs. In: BuRO'10 (2010), http://ontorule-project.eu/attachments/075_buro2010-proceedings.pdf