Optimising Resolution-Based Rewriting Algorithms for DL Ontologies*

Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos Stamou

School of Electrical and Computer Engineering,
National Technical University of Athens, Greece

Abstract. Resolution-based rewriting algorithms have been widely used for computing disjunctive datalog rewritings for DL TBoxes, and recently, also for computing datalog rewriting for queries over TBoxes expressed in DL-Lite and $\mathcal{ELHI}$. Although such algorithms are general enough to support a wide variety of (even very complex) DLs, this generality comes with performance prices. In the current paper we present a resolution-based (query) rewriting algorithm for $\mathcal{ELHI}$ that is based on a hyper-resolution like inference rule and which avoids performing many redundant inferences. We have implemented the algorithm and have conducted an experimental evaluation using large and complex ontologies.

1 Introduction

A prominent approach to query answering over DL ontologies is via rewriting the given input into formalisms for which efficient data retrieval systems exist. More precisely, in *TBox rewriting* (resp. *query rewriting*) the input TBox $\mathcal{T}$ (resp. TBox $\mathcal{T}$ and query $\mathcal{Q}$) is transformed into a set of sentences $\mathcal{P}$, typically a (disjunctive) datalog program called *rewriting*, such that for any dataset D the answers of any ground query $\mathcal{Q}_g$ (resp. of the query $\mathcal{Q}$) w.r.t. D and $\mathcal{T}$ coincide with the answers of $\mathcal{Q}_g$ (resp. $\mathcal{Q}$) w.r.t. D and $\mathcal{P}$ [12, 3, 19]. Consequently, after computing $\mathcal{P}$ the problem of query answering can be delegated to efficient and scalable (deductive) database and datalog evaluation systems by either directly evaluating $\mathcal{P}$ using off-the-shelf systems [8], by implementing customised engines [16], or by integrating $\mathcal{P}$ in an optimal way into data saturation engines [23].

Numerous rewriting systems for DLs of varying expressivity have been developed the last decade. One of the first practical systems for answering ground queries over OWL DL ontologies, namely KAON2 [16], was based on TBox-rewriting. Also recently, a large number of rewriting-based systems for answering arbitrary queries over less expressive DLs have been developed. Prominent examples include QuOnto [1], Presto [21], Quest [20], Rapid [7], Nyaya [17], and IQAROS [24], which support DL-Lite, and Requiem [19] and Clipper [8] which support $\mathcal{ELHI}$ and Horn-$\mathcal{SHIQ}$, respectively.

An important approach for computing rewritings is by using resolution calculi [2]. In this setting, the input is first transformed into a set of clauses which

* Work was supported by an FP7 Marie Curie CIG grant and Europeana Creative.

is then saturated using resolution to derive new (datalog) clauses. On the one hand, such calculi are worst-case optimal and allow for a large number of existing optimisations, like ordering restrictions and subsumption deletion. On the other hand, since there exist many resolution-based decision procedures for expressive fragments of first order logic [9, 13] it is (relatively) easier to design a resolution-based rewriting algorithm for an expressive DL compared to designing a custom made one. For example, to the best of our knowledge, none of the tailor made systems for DL-Lite can currently support more expressive DLs.

However, the efficiency of resolution-based approaches has also been criticised [22]. Even with all existing optimisations the saturation can perform many redundant inferences producing clauses that contain function symbols and which don't subsequently lead to the generation of function-free (datalog) clauses that are part of the rewriting. Hence, tailor made approaches have greatly surpassed the resolution-based ones [22, 14]. To circumvent this issue and provide an efficient resolution-based query rewriting algorithm for DL-Lite, an optimised hyper-resolution like inference that avoids producing (intermediate) redundant clauses that contain function symbols has been proposed [7]. The calculus was implemented in Rapid and as shown also by independent evaluations, Rapid is currently one of the fastest query rewriting systems [7, 14], however, like most of them it can only support DL-Lite.

In the current paper we investigate whether a resolution-based rewriting algorithm that is based on a similar hyper-resolution step can be defined for for $\mathcal{ELHI}$ TBoxes. This is a technically very challenging task as the structure of $\mathcal{ELHI}$ axioms implies many complex interactions between the clauses (in contrast to DL-Lite, $\mathcal{ELHI}$ is not FO-rewritable and subsumption checking is in ExpTime). However, we show that a rewriting can be computed by a calculus that contains an extension of the hyper-resolution step of DL-Lite plus a new resolution rule that we expect to be rarely applied in practice. To achieve this we first cast the Rapid calculus for DL-Lite to the framework of resolution and sketch its correctness. Finally, we conducted an experimental evaluation using large-scale real-world ontologies. Our results show that in many tests state-of-the-art systems cannot compute a rewriting within a reasonable amount of time (more than one hour) compared to the new implementation.

2 Preliminaries

We use the standard notions of first-order term, atom, variable, sentence, constant, function symbols, functional terms, entailment ($\models$), and the like.

Resolution-Based Calculi We use standard notions from (resolution) theorem-proving like clause, (hyper-)resolvent and most general unifier (mgu). An *inference rule*, or simply *inference* is an $n+1$-ary relation usually written as follows:

$$\frac{\mathcal{C}_1 \quad \mathcal{C}_2 \ \ldots \ \mathcal{C}_n}{\mathcal{C}} \tag{1}$$

where $\mathcal{C}_1$ is called the *main premise*, $\mathcal{C}_2, \ldots \mathcal{C}_n$ are called the *side premises* and $\mathcal{C}$ is called the *conclusion* or *resolvent*. An *inference system* $\mathcal{I}$, also called *calculus*,

is a collection of inference rules. Let Σ be a set of clauses, $\mathcal{C}$ a clause and $\mathcal{I}$ an inference system. A *derivation* of $\mathcal{C}$ from Σ by $\mathcal{I}$, written $\Sigma \vdash^{\mathcal{I}} \mathcal{C}$ (or simply $\Sigma \vdash \mathcal{C}$ if $\mathcal{I}$ is clear from the context), is a sequence of clauses $\mathcal{C}_1, \ldots, \mathcal{C}_m$ such that $\mathcal{C}_m = \mathcal{C}$, each $\mathcal{C}_i$ is either a member of Σ or the conclusion of an inference by $\mathcal{I}$ from $\Sigma \cup \{\mathcal{C}_1, \ldots, \mathcal{C}_{i-1}\}$. In that case we say that $\mathcal{C}$ is *derivable* from Σ by $\mathcal{I}$. We write $\Sigma \vdash_i \mathcal{C}$ to denote that the depth of the corresponding *derivation tree* [6] constructed for $\mathcal{C}$ from Σ is less or equal to i. We also often write $\Sigma, \mathcal{C} \vdash \mathcal{C}'$ instead of $\Sigma \cup \{\mathcal{C}\} \vdash \mathcal{C}'$. An inference system with particular interest to us is SLD, denoted by $\mathcal{I}_{\mathsf{SLD}}$, where all side premises are members of Σ.

Description Logics Let $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{I}$ be countable, pairwise disjoint sets of *atomic concepts*, *atomic roles*, and *individuals*. An $\mathcal{ELHI}$*-role* is either an atomic role P or its *inverse* P^-. The set of $\mathcal{ELHI}$-concepts is defined inductively as follows, where $A \in \mathbf{C}$, R is an $\mathcal{ELHI}$-role, and $C_{(i)}$ are $\mathcal{ELHI}$-concepts: $C := \top \mid \bot \mid A \mid C_1 \sqcap C_2 \mid \exists R.C$ An $\mathcal{ELHI}$-TBox $\mathcal{T}$ is a finite set of GCIs $C_1 \sqsubseteq C_2$, with C_i $\mathcal{ELHI}$-concepts, and RIAs $R_1 \sqsubseteq R_2$ with R_i $\mathcal{ELHI}$-roles. We assume from now on that $\mathcal{ELHI}$-TBoxes are normalised, i.e., they contain only GCIs of the form $A_1 \sqsubseteq A_2$, $A_1 \sqcap A_2 \sqsubseteq A$, $A_1 \sqsubseteq \exists R.A_2$, or $\exists R.A_2 \sqsubseteq A_1$, where $A_{(i)} \in \mathbf{C} \cup \{\top\}$, and $R \in \mathbf{R}$. An ABox $\mathcal{A}$ is a finite set of assertions $A(a)$ or $P(a, b)$, for $A \in \mathbf{C}$, $P \in \mathbf{R}$, and $a, b \in \mathbf{I}$. An $\mathcal{ELHI}$-ontology is a set $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$. DL-Lite is obtained from $\mathcal{ELHI}$ by disallowing GCIs of the form $\exists R.A_1 \sqsubseteq A_2$ where $A_1 \neq \top$.[1] We call such GCIs RA-GCIs while all the rest DL-Lite-GCIs.

Queries and Query Rewriting A *datalog rule* r is an expression of the form $H \leftarrow B_1 \wedge \ldots \wedge B_n$ where H, called *head*, is a (possibly empty) function-free atom, $\{B_1, \ldots, B_n\}$, called *body*, is a set of function-free atoms, and each variable in the head also occurs in the body. A variable that appears twice in the body and not in the head is called *ej-variable*; we use $\mathsf{ejvar}(r)$ to denote all ej-variables of r. A *datalog program* $\mathcal{P}$ is a finite set of datalog rules. A *union of conjunctive queries* (UCQ) $\mathcal{Q}$ is a set of datalog rules such that their head atoms share the same predicate, called *query predicate*, which does not appear anywhere in the body. A *conjunctive query* (CQ) is a UCQ with exactly one rule. We often abuse notation and identify a CQ with the only rule it contains instead of a singleton set. For a query $\mathcal{Q}$ with query predicate Q, a tuple of constants $\vec{a}$ is an answer of $\mathcal{Q}$ w.r.t. a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ if the arity of $\vec{a}$ agrees with the arity of Q and $\mathcal{T} \cup \mathcal{A} \cup \mathcal{Q} \models Q(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$ the answers to $\mathcal{Q}$ w.r.t. $\mathcal{T} \cup \mathcal{A}$. Finally, we recall the notion of *(query) rewriting* [3, 19, 21].

Definition 1. *Let $\mathcal{Q}$ be a CQ with query predicate Q and let $\mathcal{T}$ be a TBox. A* datalog rewriting *(or simply* rewriting*) $\mathcal{R}$ of a CQ $\mathcal{Q}$ w.r.t. $\mathcal{T}$ is a datalog program whose rules can be partitioned into two disjoint sets, $\mathcal{R}_D$ and $\mathcal{R}_Q$ such that $\mathcal{R}_D$ does not mention Q, $\mathcal{R}_Q$ is a UCQ with query predicate Q, and where*

[1] In the literature, this DL is called DL-Lite$_{\mathcal{R},\sqcap}$ [4], however, for simplicity we call it DL-Lite. Typically, DL-Lite also allows for axioms of the form $A_1 \sqsubseteq \neg A_2$ and $R_1 \sqsubseteq \neg R_2$, however, these do not have any effects in query *answering* when $\mathcal{T} \cup \mathcal{A}$ is consistent (see, e.g., [19]); hence we will discard them here.

for each $\mathcal{A}$ consistent w.r.t. $\mathcal{T}$ and using only predicates from $\mathcal{T}$ we have:

$$\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{R}_Q, \mathcal{R}_D \cup \mathcal{A}).$$

The Requiem System Throughout the paper we will use the inference system implemented by Requiem as a yardstick, to highlight the inefficiencies of typical resolution-based rewriting algorithms and sketch completeness of the refined approach. Like most rewriting systems, the behaviour of Requiem on input $\mathcal{Q}$ and $\mathcal{T}$ can be characterised by the application of the following three steps:

1. *Clausification*: the input TBox $\mathcal{T}$ is transformed into a set of clauses $\mathcal{T}_\mathcal{C}$, using standard techniques like skolemisation (see Example 1).
2. *Saturation*: $\mathcal{T}_\mathcal{C} \cup \mathcal{Q}$ is saturated by using (binary) resolution with free selection [2] (denoted as $\mathcal{I}_{\mathsf{REQ}}$) computing a new set $\mathcal{T}_\mathcal{C}^{sat}$.
3. *Post-processing*: all function-free clauses in $\mathcal{T}_\mathcal{C}^{sat}$ are returned.

Next, we briefly illustrate the Requiem calculus through an example.

Example 1. Consider the TBox $\mathcal{T}$ consisting of the following GCIs (left side), together with the respective clauses produced at step 1. (right side):

$$A \sqsubseteq \exists R.B \quad \rightsquigarrow \quad R(x, f(x)) \leftarrow A(x) \tag{2a}$$
$$B(f(x)) \leftarrow A(x) \tag{2b}$$
$$R \sqsubseteq S \quad \rightsquigarrow \quad S(x, y) \leftarrow R(x, y) \tag{3}$$

As can be seen, the first GCI produces two clauses, where f is a skolem function that is uniquely associated with the specific occurence of concept $\exists R.B$.

Consider now the query $\mathcal{Q}_1 = Q(x) \leftarrow S(x, y) \wedge C(y)$. When applied to $\mathcal{Q}_1$ and $\mathcal{T}$ the Requiem algorithm would perform the following inferences:

$$\text{(2a) and (3) produces } S(x, f(x)) \leftarrow A(x) \tag{4}$$
$$\mathcal{Q}_1 \text{ and (4) produces } \mathcal{Q}_2 = Q(x) \leftarrow A(x) \wedge C(f(x)) \tag{5}$$

Finally, the function-free set returned is $\mathcal{R} = \{\mathcal{Q}_1, (3)\}$. ♢

Conventions To simplify the presentation, in the following we assume that TBoxes are given in a clausal form. Clauses that are produced by clausfying RA-GCIs and DL-Lite-GCIs are called RA-clauses and DL-Lite-clauses, respectively. Finally, clauses with the query predicate in the head are called *Q-clauses*; note that such clauses can contain functional terms in the body (e.g., clause $\mathcal{Q}_2$ in Example 1).

3 An Optimised Calculus for DL-Lite

In the current section we present a hyper-resolution like rewriting algorithm for DL-Lite, which characterises the algorithm implemented in Rapid using the framework of resolution. In addition, we sketch its proof of correctness, which implies correctness of Rapid; this is also important subsequently for $\mathcal{ELHI}$.

Example 2. Consider the TBox $\mathcal{T}$ and query $\mathcal{Q}_1$ from Example 1 as well as the inferences performed by $\mathcal{I}_{\mathsf{REQ}}$ on $\mathcal{T} \cup \mathcal{Q}_1$. As it can be observed the algorithm performed two redundant inferences since neither clauses (4) and $\mathcal{Q}_2$ nor any other clause derived from these is a member of the computed rewriting $\mathcal{R}$.

These clauses would be of importance only if a clause of the form $C(x) \leftarrow B(x)$ also existed in $\mathcal{T}$. Then, the latter would resolve with (2b) producing $C(f(x)) \leftarrow A(x)$ which would subsequently resolve with $\mathcal{Q}_2$ to produce $Q(x) \leftarrow A(x)$, that would be part of $\mathcal{R}$. $\diamondsuit$

As it has been argued in [22], TBoxes typically contain many clauses of the form $R(x, f_i(x)) \leftarrow A_i(x)$. Together with clause (3) this implies that the algorithm would produce many clauses of the form $S(x, f_i(x)) \leftarrow A_i(u)$ and $Q(x) \leftarrow A(x) \wedge C(f_i(x))$ which can adversely affect performance.

One could try to remedy the above issue by applying the following approach/refinement on $\mathcal{I}_{\mathsf{REQ}}$: first, saturate the clauses of $\mathcal{T}$ obtaining $\mathcal{T}_{sat}$ and then, resolve $\mathcal{Q}_1$ with (4) only if a clause of the form $C(f(x)) \leftarrow A(x)$ also exists in $\mathcal{T}_{sat}$. If it does, then $Q(x) \leftarrow A(x)$ can be obtained directly from $\mathcal{Q}_1$ as a hyper-resolvent; if it doesn't, then we can avoid computing $\mathcal{Q}_2$. However, to implement this hyper-resolution step the algorithm needs to perform a quadratic loop over the set $\mathcal{T}_{sat}$, which in practice can be very large. As we will show in the next section this issue is even more acute in more expressive ontologies like $\mathcal{ELHI}$ which allow for RA-clauses (see also discussion in [22]).

Our inability to efficiently implement the above refinement is because $\mathcal{I}_{\mathsf{REQ}}$ follows a "forward" style approach to apply resolution which generates new clauses that contain function symbols (e.g., clause (4)). In contrast Rapid is based on a goal-oriented "backwards" style approach that resembles SLD derivations. In the previous example, it will first resolve $\mathcal{Q}_1$ with (3) to obtain $\mathcal{Q}'_2 = Q(x) \leftarrow R(x, y) \wedge C(y)$, while then, it will resolve $\mathcal{Q}'_2$ with clause (2a) only if also a clause of the form $C(f(x)) \leftarrow A(x)$ exists in $\mathcal{T}$. The crucial difference is that, in the latter case, to implement the hyper-resolution step we pick clauses from $\mathcal{T}$ rather than $\mathcal{T}_{sat}$. In addition to $\mathcal{T}$ being much smaller than $\mathcal{T}_{sat}$, as explained in Example 1, functional terms are unique per occurence of $\exists R.B$, and this can be exploited in practice using indexes over the functional terms. The next definition characterises the Rapid algorithms using the framework of resolution.

Definition 2. *Let $\mathcal{Q}$ be a CQ, let $\mathcal{C}_{(i)}$ be (not necessarily distinct) DL-Lite clause(s) with head atom(s) $C_{(i)}$. With $\mathcal{I}_{lite}$ we denote the inference system that consists of the following inference rules, where $\mathcal{Q}'\sigma$ is a function-free (hyper-)resolvent of $\mathcal{Q}$ and $\mathcal{C}_{(i)}$:*

$$\textit{unfolding:} \quad \frac{\mathcal{Q} \qquad \mathcal{C}}{\mathcal{Q}'\sigma} \qquad \textit{s.t. if } x \mapsto f(y) \in \sigma, \textit{ then } x \notin \mathsf{ejvar}(\mathcal{Q})$$

$$\textit{shrinking:} \quad \frac{\mathcal{Q} \qquad \mathcal{C}_1 \;\; \mathcal{C}_2}{\mathcal{Q}'\sigma} \qquad \textit{s.t. there exists } x \mapsto f(y) \in \sigma \textit{ with } x \in \mathsf{ejvar}(\mathcal{Q})$$

Finally, for $\mathcal{Q}$ a CQ and $\mathcal{T}$ a DL-Lite-TBox let $\mathsf{Rapid\text{-}Lite}(\mathcal{Q}, \mathcal{T})$ *be all function-free clauses derivable from $\mathcal{Q} \cup \mathcal{T}$ by $\mathcal{I}_{lite}$.*

Intuitively, unfolding corresponds to all classical binary resolution inferences that won't introduce function symbols in $\mathcal{Q}$, while shrinking is a hyper-resolution step which "eliminates" an ej-variable of $\mathcal{Q}$ using clauses that mention a functional term f.

Example 3. Consider $\mathcal{T}$ and $\mathcal{Q}_1$ from Example 1 and consider also the TBox $\mathcal{T}' = \{C(x) \leftarrow B(x)\} \cup \mathcal{T}$. Applying $\mathcal{I}_{lite}$ to $\mathcal{T} \cup \mathcal{Q}_1$ performs the following inferences:

$$\begin{aligned} \text{unfolding on } \mathcal{Q}_1 \text{ and (3)} \;&\text{ produces }\; \mathcal{Q}'_2 = Q(x) \leftarrow R(x,y) \wedge C(y) \\ \text{unfolding on } \mathcal{Q}'_2 \text{ and } C(x) \leftarrow B(x) \;&\text{ produces }\; \mathcal{Q}'_3 = Q(x) \leftarrow R(x,y) \wedge B(y) \\ \text{shrinking on } \mathcal{Q}'_3, \text{(2a), and (2b)} \;&\text{ produces }\; \mathcal{Q}_4 = Q(x) \leftarrow A(x) \end{aligned}$$

The set $\mathcal{R}' = \{\mathcal{Q}_1, \mathcal{Q}'_2, \mathcal{Q}'_3, \mathcal{Q}_4\}$ is a rewriting of $\mathcal{Q}_1$ w.r.t. $\mathcal{T}$. ♢

Correctness of our rewriting algorithm follows by showing how a derivation constructed by $\mathcal{I}_{\mathsf{REQ}}$ can be transformed into a derivation by $\mathcal{I}_{lite}$; hence, saturating $\mathcal{T} \cup \mathcal{Q}$ by $\mathcal{I}_{lite}$ will create all necessary members of a rewriting. This is done in two steps. First, we show that each Requiem derivation can be transformed into an SLD derivation.

Lemma 1. *Let $\mathcal{T}$ be a DL-Lite-TBox and let $\mathcal{Q}$ be a CQ with query predicate Q. Every Q-clause $\mathcal{Q}'$ derivable from $\mathcal{T} \cup \mathcal{Q}$ by $\mathcal{I}_{\mathsf{REQ}}$ is also derivable from $\mathcal{T} \cup \mathcal{Q}$ by $\mathcal{I}_{\mathsf{SLD}}$.*

The Lemma is proven by showing how to "unfold" a Requiem inference $\mathcal{Q}, \mathcal{C} \vdash \mathcal{Q}'$ where $\mathcal{T} \vdash_i \mathcal{C}$ into $\mathcal{Q}, \mathcal{C}_1 \vdash \mathcal{Q}'', \mathcal{Q}'', \mathcal{C}_2 \vdash \mathcal{Q}'$ where $\mathcal{T} \vdash_{i-1} \mathcal{C}_1, \mathcal{T} \vdash_{i-1} \mathcal{C}_2$ and $\mathcal{C}_1, \mathcal{C}_2 \vdash \mathcal{C}$. By repeated application of this unfolding we will eventually (fully) unfold any derivation of a Q-clause $\mathcal{Q}'$ into inferences of the form $\mathcal{Q}, \mathcal{C}_1 \vdash \mathcal{Q}_1, \ldots, \mathcal{Q}_{n-1}, \mathcal{C}_n \vdash \mathcal{Q}'$, where $\mathcal{C}_i \in \mathcal{T}$, i.e., into an SLD derivation of $\mathcal{Q}'$.

Second, we show that each SLD derivation can be transformed into a derivation by $\mathcal{I}_{lite}$. By definition of $\mathcal{I}_{lite}$ and Example 4 we can see that inferences using the unfolding rule directly corresponds to one SLD inference, while shrinking corresponds to many SLD inferences where, the main premise is a function-free CQ, the side premises contain the same function symbol f, and the conclusion is also a function-free CQ. Hence, we need to show that each SLD derivation can be transformed into one that satisfies the following property.

Definition 3. *Let $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n$ be an SLD derivation of $\mathcal{Q}_n$ such that $\mathcal{Q}_i, \mathcal{C}_i \vdash \mathcal{Q}_{i+1}$ for some clause $\mathcal{C}_i$. Assume also that all $\mathcal{Q}_2, \ldots, \mathcal{Q}_{n-1}$ contain a term that mentions a function symbol f while $\mathcal{Q}_1$ and $\mathcal{Q}_n$ are function-free. We say that the derivation is* function compact *if all side premises $\mathcal{C}_i$ with $1 \leq i < n$ used in the derivation also mention f.*

The following can be shown for Horn clauses with one existential variable.

Lemma 2. *Any SLD derivation can be transformed to a function compact one.*

Using Lemmas 1 and 2 we can show the following.

Theorem 1. *Let a DL-Lite-TBox $\mathcal{T}$ and a CQ $\mathcal{Q}$. Every derivation from $\mathcal{T} \cup \mathcal{Q}$ by $\mathcal{I}_{lite}$ terminates. Moreover,* Rapid-Lite$(\mathcal{Q}, \mathcal{T})$ *is a rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{T}$.*

4 A Rewriting Algorithm for $\mathcal{ELHI}$

In the current section we extend the inference system $\mathcal{I}_{lite}$ in order to provide a (query) rewriting algorithm for $\mathcal{ELHI}$ ontologies.

$\mathcal{ELHI}$ additionally allows for RA-clauses of the form $E(x) \leftarrow R(x,y) \wedge F(x)$. According to the Requiem calculus this clause interacts with clause (3) of Example 2 producing $E(x) \leftarrow A(x) \wedge F(f(x))$. Again this inference is of interest only if also a clause of the form $F(f(x)) \leftarrow G(x)$ can be deduced and hence then produce $F(x) \leftarrow A(x) \wedge G(x)$ which is function-free. As argued in [22] ontologies typically have many clauses of these forms which can lead to a quadratic number of redundant inferences.

A straightforward approach to obtain an efficient calculus for $\mathcal{ELHI}$ ontologies would be to extend Definition 2 to allow for arbitrary $\mathcal{ELHI}$-clauses as side premises in shrinking and unfolding. Then, Lemmas 1 and 2 apply with few modifications and hence this calculus would produce a rewriting.

Example 4. Let $\mathcal{T}$ be the $\mathcal{ELHI}$-TBox consisting of the following clauses:

$$C(x) \leftarrow S(x,y) \wedge D(y) \tag{6}$$
$$S(f(x),x) \leftarrow B(x) \tag{7}$$
$$K(x) \leftarrow S(y,x) \wedge C(y) \tag{8}$$

and let also the query $\mathcal{Q}_1 = Q(x) \leftarrow K(x)$.

By unfolding on $\mathcal{Q}_1$ and (9) we obtain $\mathcal{Q}_2 = Q(x) \leftarrow S(y,x) \wedge C(y)$; then, by unfolding on $\mathcal{Q}_2$ and (7) we obtain $\mathcal{Q}_3 = Q(x) \leftarrow S(y,x) \wedge S(y,z) \wedge D(z)$; finally, by shrinking on $\mathcal{Q}_3$ and (8) we can obtain $\mathcal{Q}_4 = Q(x) \leftarrow B(x) \wedge D(x)$. It can be verified that $\mathcal{R} = \{\mathcal{Q}_1, \mathcal{Q}_2, \mathcal{Q}_3, \mathcal{Q}_4\}$ is a rewriting of $\mathcal{Q}$ w.r.t. $\mathcal{T}$ $\diamondsuit$

However, as the previous example shows, unfolding using RA-clauses produces clauses that contain more variables than the main premise and hence to variable proliferation which implies termination problems. There are two solutions to this issue. We either decide a bound on the number of times that such clauses can be used as side premises or, like in $\mathcal{I}_{lite}$, we still only allow DL-Lite-clauses as side premises. But then, such a calculus would require additional inference rules in order to be able to derive intermediate clauses that can then be side premises in inferences. Next, we show how the Requiem calculus would behave when applied to the input of Example 5 which motivates our $\mathcal{ELHI}$ calculus.

Example 5. Consider the TBox $\mathcal{T}$ and query $\mathcal{Q}_1$ from Example 5. When applied to $\mathcal{T}$ and $\mathcal{Q}_1$ the Requiem algorithm would perform the following inferences with the respective conclusions:

$$(7) \text{ and } (8) \text{ produces } C(f(x)) \leftarrow B(x) \wedge D(x) \tag{9}$$
$$(9) \text{ and } (8) \text{ produces } K(x) \leftarrow B(x) \wedge C(f(x)) \tag{10}$$
$$(11) \text{ and } (10) \text{ produces } K(x) \leftarrow B(x) \wedge D(x) \tag{11}$$
$$\mathcal{Q}_1 \text{ and } (12) \text{ produces } Q(x) \leftarrow B(x) \wedge D(x) \tag{12}$$

The set $\mathcal{R} = \{\mathcal{Q}_1, (7), (9), (13)\}$ is a (datalog) rewriting of $\mathcal{Q}_1$ w.r.t. $\mathcal{T}$.

First, we can observe that, if we extend shrinking to accept RA-clauses as main premises, then the inferences performed to produce clause (12) from clause (9) can be captured by a single shrinking inference over clause (9) with side premises (8) and (10). However, we can observe that clause (10) is produced by resolving together an RA-clause with a DL-Lite-clause and this cannot be captured by any of the inferences of $\mathcal{I}_{lite}$. ♢

Motivated by the above example, our caluclus for $\mathcal{ELHI}$-ontologies consists of a new inference rule which can infer clauses like clause (10), together with unfolding and (an extension of) shrinking that permit as a main premise either a CQ or an RA-clause (i.e., $\mathcal{I}_{lite}$ with main premise clauses that are non DL-Lite).

Definition 4. *With $\mathcal{I}_{\mathcal{EL}}$ we denote the inference system consisting of unfolding, where the main premise is either a CQ or an RA-clause, together with the following inference rules, where Υ is either a CQ or an RA-clause, for $n \geq 2$ each $\mathcal{C}_i$ is a DL-Lite-clause, and $\Upsilon'\sigma$ is a function-free hyper-resolvent of Υ and $\mathcal{C}_i$:*

n-shrinking: $\dfrac{\Upsilon \qquad \mathcal{C}_1 \ \ldots \ \mathcal{C}_n}{\Upsilon'\sigma}$ *s.t. there exists* $x \mapsto f(y) \in \sigma$ *with* $x \in \mathsf{ejvar}(\Upsilon)$

function:
$$\frac{B(x) \leftarrow R(x,y) \wedge C(y) \qquad R(f(x),x) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge C(x)} \text{ or}$$
$$\frac{B(x) \leftarrow R(y,x) \wedge C(y) \qquad R(x,f(x)) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge C(x)}$$

Finally, for $\mathcal{Q}$ a CQ and $\mathcal{T}$ an $\mathcal{ELHI}$-TBox, Rapid-EL$(\mathcal{Q}, \mathcal{T})$ *is defined as all function-free clauses derivable from $\mathcal{Q} \cup \mathcal{T}$ by $\mathcal{I}_{\mathcal{EL}}$.*

First, note that we had to extend shrinking to accept possibly more that two side premises. This is because due to the function rule we can now deduce new clauses that contain function symbols; hence there can be more than two clauses containing some function f, unlike the case in DL-Lite. In practice, however, we expect that n is small since such clauses are only produced by the function rule and only when there is a complex interaction between a clause containing $R(x, f(x))$ $(R(f(x), x))$ and an RA-clause containing the inverse $R(y, x)$ $(R(x, y))$.

Example 6. Consider Example 6. The inference between clauses (7) and (8) that produces clause (10) corresponds to an inference using the function rule. Then, clause (12) can be produced by n-shrinking over (9) with side premises clauses (8) and (10), while (13) is produced by unfolding on $\mathcal{Q}$ and (9), hence computing the required rewriting. ♢

As mentioned in the previous section correctness of Rapid-Lite is heavily based on showing that Requiem derivations can be unfolded into SLD derivations. Consider now $\mathcal{ELHI}$ and an inference of the form $\Upsilon, \mathcal{C} \vdash \Upsilon'$, where $\mathcal{T} \vdash^{\mathcal{I}_{\mathsf{REQ}}} \mathcal{C}$.

If the derivation of $\mathcal{C}$ does not involve an RA-clause, then again we can fully unfold into an SLD derivation, like in DL-Lite. If it does, then this inference can be unfolded into $\Upsilon, \mathcal{C}_1 \vdash \Upsilon_2, \ldots, \Upsilon_{n-1}, \mathcal{C}_n \vdash \Upsilon'$ up to a certain level; i.e., for some $\mathcal{C}_i$ we might have $\mathcal{C}_i \notin \mathcal{T}$ with $\mathcal{C}_i$ derivable by some RA-clause. By inspecting the types of inferences performed by $\mathcal{I}_{\mathsf{REQ}}$ using RA-clauses (cf. [18, Table 4]) we can see that we can unfold up to the point where $\mathcal{C}_i$ has one of the following forms, where the notation $[C(x)]$ means that $C(x)$ can be omitted:

$$A(x) \leftarrow B(x) \wedge [C(x)] \tag{13}$$

$$B(f(x)) \leftarrow A(x) \wedge [C(x)] \tag{14}$$

Again by inspection of $\mathcal{I}_{\mathsf{REQ}}$ we can see that clauses of form (15) can be produced from RA-clauses by an inference which is exactly the one captured by the function rule. The derivation of clauses of form (14) is more involved and is characterised next.

Lemma 3. *Let $\mathcal{P}$ be a set of DL-Lite-clauses and let Υ_1 be an RA-clause. Consider also a derivation $\Upsilon_1, \ldots, \Upsilon_n$ by $\mathcal{I}_{\mathsf{REQ}}$, such that $\Upsilon_i, \mathcal{C}_i \vdash \Upsilon_{i+1}$ and $\mathcal{C}_i$ is derivable from $\mathcal{P}$ by $\mathcal{I}_{\mathsf{REQ}}$. Assume also that Υ_n is of the form $A(x) \leftarrow B(x) \wedge [C(x)]$ while no Υ_i with $i < n$ is of the same form as Υ_n. Then, Υ_n can be derived from $\mathcal{P} \cup \Upsilon_1$ by unfolding and n-shrinking.*

Using Lemma 3 and our observation regarding clauses of form (15) we can show the following.

Lemma 4. *Let $\mathcal{T}$ be an $\mathcal{ELHI}$-TBox and let $\mathcal{Q}$ be a CQ with query predicate Q. Let also $\mathcal{P}_{lt}$ be all DL-Lite-clauses derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathcal{EL}}$. Then, every Q-clause $\mathcal{Q}'$ derivable from $\mathcal{T} \cup \mathcal{Q}$ by $\mathcal{I}_{\mathsf{REQ}}$ is also derivable from $\mathcal{P}_{lt} \cup \mathcal{Q}$ by $\mathcal{I}_{\mathsf{SLD}}$.*

Finally, we can show the following.

Theorem 2. *Let an $\mathcal{ELHI}$-TBox $\mathcal{T}$ and a CQ $\mathcal{Q}$. Every derivation from $\mathcal{T} \cup \mathcal{Q}$ by $\mathcal{I}_{\mathcal{EL}}$ terminates. Moreover,* Rapid-EL$(\mathcal{Q}, \mathcal{T})$ *is a datalog rewriting of $\mathcal{Q}, \mathcal{T}$.*

5 Evaluation

We have extended the Rapid[2] query rewriting system [7] to support $\mathcal{ELHI}$ ontologies. Rapid attempts to compactly represent many unfolding inferences as datalog rules in an effort to compute small (datalog) rewritings.

We conducted an experimental evaluation comparing against Requiem [19], Presto [21], and Clipper [8], using real-world large scale DL-Lite and $\mathcal{ELHI}$ ontologies. Regarding DL-Lite, we used DL-Lite versions of the OpenGALEN2 (`http://www.opengalen.org/`), OBO protein (`http://www.obofoundry.org/`), and NCI 3.12e (`http://evs.nci.nih.gov/ftp1/NCI_Thesaurus`) ontologies, which we denote by $\mathbf{G}_d$, $\mathbf{P}_d$, and $\mathbf{N}$, respectively. This part extends the evaluation in [7] for large scale ontologies, and uses datalog rewritings instead of UCQs.

[2] `http://www.image.ece.ntua.gr/~achort/rapid.zip`

Table 1. Statistics of the used test ontologies

$\mathcal{O}$	concepts	roles	GCIs	RIAs	$\mathcal{O}$	concepts	roles	GCIs	RIAs	$\mathcal{O}$	concepts	roles	GCIs	RIAs
$\mathbf{G}_d$	23193	851	49046	882	**S**	4298	519	6004	372	$\mathbf{G}_e$	30048	851	63726	882
$\mathbf{P}_d$	35351	6	43351	0	**C**	4282	22	9564	15	$\mathbf{P}_e$	37560	6	52383	0
					N	29173	66	53341	0					

Regarding $\mathcal{ELHI}$, we used $\mathcal{ELHI}$ versions of the NASA SWEET 2.3 (`http://sweet.jpl.nasa.gov/ontology/`), periodic table (`http://www.cs.man.ac.uk/~stevensr/ontology/`), OpenGALEN2, and OBO protein ontologies, which we denote by **S**, **C**, $\mathbf{G}_e$, and $\mathbf{P}_e$, respectively. The DL-Lite and $\mathcal{ELHI}$ versions of the ontologies where obtained by normalizing the ontologies and keeping the appropriate subset of axioms. Table 1 provides statistics for the ontologies. For each of them we manually constructed 5 test queries. All tests were performed on a dual core 1.8GHz Intel Celeron processor laptop running Windows 8 and JVM 1.7 with 3.6GB maximum heap size. The timeout limit was set to 2 hours.

The results for DL-Lite are shown in the upper part of Table 2, which includes the computation time and the size of the computed rewriting; "t/o" denotes timeout. First, we observe that neither Presto nor Clipper managed to compute a rewriting for $\mathbf{G}_d$, Requiem required 9–18 minutes, while Rapid required only milliseconds. Similar observations can be made for the rest of the DL-Lite ontologies. Notable cases are queries 1–5 over **N** for Clipper, where it required about 13 minutes for each of them, query 5 over $\mathbf{P}_d$ and query 3 over **N** for Requiem, for which it did not manage to compute a rewriting, and all queries over $\mathbf{P}_d$ and **N** for Presto for which it required 1 to 2 hours, being in general much slower also from Requiem and Clipper. Rapid was slower only for query 5 over $\mathbf{P}_d$, where it required 7:15 minutes. The analysis showed that 7:00 out of the 7:15 minutes are spent in backwards subsumption checking (which guarantees a compact result with no equivalent or subsumed queries), while the actual rewriting time is only 15 seconds. Note that no other system performs backwards subsumption. In fact, in this case e.g. Clipper returns several rewritings that are equivalent up to variable renaming, which justifies also the difference in the rewriting sizes.

The results for $\mathcal{ELHI}$ are shown in the lower part of Table 2; we did not run Presto since it does not support $\mathcal{ELHI}$. Our conclusions are similar to those of DL-Lite—that is, in the large scale ontologies Rapid greatly outperforms all other systems while in query 5 of the $\mathcal{ELHI}$ version of the OBO protein ontology ($\mathbf{P}_e$) it performed worse than Clipper since it spent 13:27 out of the 13:52 minutes in backwards subsumption. Note also that for $\mathbf{G}_e$, Rapid was the only system that managed to compute a rewriting within 'reasonable' time (8–11 minutes).

In summary, we can see that computing even some rewriting over large scale and complex ontologies is still an unresolved issue as in many cases many state-of-the-art systems did not manage to terminate, required several minutes, or even hours. This could be acceptable if a rewriting is computed once for a fixed ontology, however, practice has shown that ontologies are very often dynamic [5, 11, 10]. Moreover, in design time, ontology engineers tend to run reasoners often

Table 2. Evaluation results.

	Time (hh:ss:mm.msec)				Rewriting size			
$\mathcal{O}$	Rapid	Requiem	Presto	Clipper	Rapid	Requiem	Presto	Clipper
DL-Lite								
$\mathbf{G}_d$	.11	18:34.96	t/o	t/o	225	30870	t/o	t/o
	.09	9:01.05	t/o	t/o	1276	1152	t/o	t/o
	.11	t/o	t/o	t/o	973	t/o	t/o	t/o
	.02	12:08.44	t/o	t/o	667	11306	t/o	t/o
	.34	t/o	t/o	t/o	3267	t/o	t/o	t/o
$\mathbf{P}_d$	.07	4.72	1:15:07.96	1:03.97	29	27	48	29
	.81	45.58	58:19.05	1:04.94	1356	1356	2621	1356
	1.40	9:50.54	1:16:36.30	1:08.40	33919	33887	33888	33919
	6.99	11:30.87	1:21:28.37	1:08.70	34879	34733	35416	34879
	7:14.56	t/o	1:12:51.73	1:08.86	27907	t/o	2670	54430
$\mathbf{N}$	0.06	4.08	1:51:31.96	12:09.45	488	5002	469	488
	0.04	37.70	1:56:40.12	13:02.02	1804	1765	1766	1804
	0.15	t/o	1:59:20.03	13:15.10	4143	t/o	3546	4143
	0.05	9:42.13	1:57:19.87	13:17.46	1875	64500	1917	1875
	0.03	1:06:26.66	1:57:09.73	13:08.27	256	219150	208	340
$\mathcal{ELHI}$								
$\mathbf{S}$	.08	.17	n/a	13.67	172	298	n/a	171
	.16	.36	n/a	13.32	473	1523	n/a	367
	.02	.44	n/a	13.79	629	1674	n/a	518
	.05	1.09	n/a	13.32	1065	2861	n/a	949
	.04	38.08	n/a	13.30	1075	18716	n/a	959
$\mathbf{C}$	.06	3:36.54	n/a	21.42	1103	6800	n/a	2892
	.05	3:40.24	n/a	19.73	879	6941	n/a	2892
	.09	3:47.85	n/a	18.95	1653	6889	n/a	2892
	.08	3:31.29	n/a	20.48	1609	8077	n/a	2849
	.15	9:10.73	n/a	20.33	1743	57054	n/a	2893
$\mathbf{G}_e$	11:01.71	t/o	n/a	t/o	142566	t/o	n/a	t/o
	8:45.87	t/o	n/a	t/o	140640	t/o	n/a	t/o
	10:18.01	t/o	n/a	t/o	143636	t/o	n/a	t/o
	9:26.00	t/o	n/a	t/o	136679	t/o	n/a	t/o
	7:47.52	t/o	n/a	t/o	128566	t/o	n/a	t/o
$\mathbf{P}_e$	4.80	t/o	n/a	1:42.08	51641	t/o	n/a	51641
	29.73	t/o	n/a	1:42.30	52877	t/o	n/a	52877
	2.59	t/o	n/a	1:45.35	51614	t/o	n/a	51614
	15.71	t/o	n/a	1:43.19	52407	t/o	n/a	52407
	13:51.97	t/o	n/a	1:47.10	79427	t/o	n/a	105950

to see what effects their changes have on reasoning. In this case, any computation requiring more than a couple of minutes might not be acceptable.

6 Conclusions

We have presented an efficient resolution-based rewriting algorithm for $\mathcal{ELHI}$. Despite the complex interactions in $\mathcal{ELHI}$ we show that a calculus using extensions of the unfolding and shrinking rules of Rapid for DL-Lite, plus a new rule can be defined. Our experimental evaluation shows that there are large scale ontologies which existing systems cannot handle, while the new algorithm only requires milliseconds. This is to a large extent due to the hyper-resolution inference performed by shrinking, which avoids well-known inefficiencies of resolution-based rewriting algorithms [22].

Concerning future work, we plan to investigate whether a similar hyper-resolution step can be used to optimise resolution-based rewriting algorithms for non-Horn DLs like $\mathcal{ALC}$, like those implemented in the KAON2 system [15].

References

1. Andrea Acciarri, Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Mattia Palmieri, and Riccardo Rosati. Quonto: Querying ontologies. In *AAAI*, pages 1670–1671, 2005.
2. Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.
3. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning*, 39(3):385–429, 2007.
4. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data complexity of query answering in description logics. *Artificial Intelligence*, 195:335–360, 2013.
5. Diego Calvanese, Evgeny Kharlamov, Werner Nutt, and Dmitriy Zheleznyakov. Evolution of dl-lite knowledge bases. In *International Semantic Web Conference (1)*, pages 112–128, 2010.
6. Chin-Liang Chang and Richard C. T. Lee. *Symbolic logic and mechanical theorem proving*. Computer science classics. Academic Press, 1973.
7. A. Chortaras, D. Trivela, and G. Stamou. Optimized query rewriting in OWL 2 QL. In *Proc. of CADE 23*, pages 192–206, 2011.
8. Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for Horn-$\mathcal{SHIQ}$ plus rules. In *AAAI*, 2012.
9. Christian G Fermüller, Alexander Leitsch, Ullrich Hustadt, and Tanel Tammet. Resolution decision procedures. In *Handbook of Automated Reasoning*, pages 1791–1849. Elsevier Science Publishers BV, 2001.
10. Giorgos Flouris, Dimitris Manakanatas, Haridimos Kondylakis, Dimitris Plexousakis, and Grigoris Antoniou. Ontology change: classification and survey. *Knowledge Eng. Review*, 23(2):117–152, 2008.
11. Rafael S. Gonçalves, Bijan Parsia, and Ulrike Sattler. Analysing the evolution of the nci thesaurus. In *CBMS*, pages 1–6, 2011.
12. Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Deciding Expressive Description Logics in the Framework of Resolution. *Information & Computation*, 206(5):579–601, 2008.
13. Ullrich Hustadt and Renate A Schmidt. Issues of decidability for description logics in the framework of resolution. In *Automated Deduction in Classical and Non-Classical Logics*, pages 191–205. Springer, 2000.
14. Martha Imprialou, Giorgos Stoilos, and Bernardo Cuenca Grau. Benchmarking ontology-based query rewriting systems. In *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence (AAAI 2012)*. AAAI Press, 2012.
15. Boris Motik. Description Logics and Disjunctive Datalog—More Than just a Fleeting Resemblance? In *Proc. of the 4th Workshop on Methods for Modalities (M4M-4)*, volume 194, pages 246–265, 2005.
16. Boris Motik, Ulrike Sattler, and Rudi Studer. Query answering for OWL-DL with rules. *J. Web Sem.*, 3(1):41–60, 2005.
17. Giorgio Orsi and Andreas Pieris. Optimizing query answering under ontological constraints. *VLDB Endowment*, 4(11):1004–1015, 2011.
18. Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Rewriting Conjunctive Queries under Description Logic Constraints. In Andrea Cal'i, Georg Gottlob, Laks V.S. Lakshmanan, and Davide Martinenghi, editors, *Proc. of the Int. Workshop on Logic in Databases (LID 2008)*, Rome, Italy, May 19–20 2008.

19. Héctor Pérez-Urbina, Boris Motik, and Ian Horrocks. Tractable query answering and rewriting under description logic constraints. *Journal of Applied Logic*, 8(2):186–209, 2010.
20. Mariano Rodriguez-Muro and Diego Calvanese. High performance query answering over DL-Lite ontologies. In *KR*, 2012.
21. Riccardo Rosati and Alessandro Almatelli. Improving query answering over DL-Lite ontologies. In *Proc. of KR-10*, 2010.
22. Frantisek Simancik, Yevgeny Kazakov, and Ian Horrocks. Consequence-based reasoning beyond horn ontologies. In *IJCAI*, pages 1093–1098, 2011.
23. Giorgos Stoilos, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. Repairing ontologies for incomplete reasoners. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11), Bonn, Germany*, pages 681–696, 2011.
24. Tassos Venetis, Giorgos Stoilos, and Giorgos Stamou. Query extensions and incremental query rewriting for OWL 2 QL ontologies. *Journal on Data Semantics, Accepted*, 2013.