Optimising Resolution-Based Rewriting Algorithms for OWL Ontologies☆

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

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

Abstract

An important approach to query answering over OWL ontologies is via *rewriting* the input ontology (and query) into a new set of axioms that are expressed in logics for which scalable query answering algorithms exist. This approach has been studied for many important fragments of OWL like $\mathcal{SHIQ}$, Horn-$\mathcal{SHIQ}$, OWL 2 QL, and OWL 2 EL. An important family of rewriting algorithms is the family of *resolution-based* algorithms, mostly because of their ability to adapt to any ontology language (such algorithms have been proposed for all aforementioned logics) and the long years of research in resolution theorem-proving. However, this generality comes with performance prices and many approaches that implement algorithms that are tailor-made to a specific language are more efficient than the (usually) general-purposed resolution-based ones.

In the current paper we revisit and refine the resolution approaches in order to design efficient rewriting algorithms for many important fragments of OWL. First, we present an algorithm for the language DL-Lite$_{R,\sqcap}$ which is strongly related to OWL 2 QL. Our calculus is optimised in such a way that it avoids performing many unnecessary inferences, one of the main problems of typical resolution algorithms. Subsequently, we extend the algorithm to the language $\mathcal{ELHI}$ which is strongly related to OWL 2 EL. This is a difficult task as $\mathcal{ELHI}$ is a relatively expressive language, however, we show that the calculus for DL-Lite$_{R,\sqcap}$ requires small extensions. Finally, we have implemented all algorithms and have conducted an extensive experimental evaluation using many well-known large and complex OWL ontologies. On the one hand, this is the first evaluation of rewriting algorithms of this magnitude, while, on the other hand, our results show that our system is in many cases several orders of magnitude faster than the existing systems even though it uses an additional backwards subsumption checking step.

Keywords: Query answering, Query rewriting, Ontology, Description Logics, Resolution

1. Introduction

Efficient management and querying of large amounts of (possibly distributed) data that are formally described using complex structures like ontologies is an important problem for many modern applications [26, 34, 12]. In such settings answers to user queries reflect both the stored data as well as the axioms that have been encoded in the ontology. However, query answering over OWL ontologies is a very challenging task mainly due to its very high computational complexity [37, 17, 30]. Even after intense implementation work and the design of modern sophisticated optimisations, direct (tableaux-based) approaches integrated in systems such as HermiT [27], Pellet [43], and Racer [46] are not yet able to cope with very large datasets. Moreover, in several important profiles of OWL 2 [33], like OWL 2 QL and OWL 2 EL, different methods for query answering have been investigated.

A prominent (indirect) approach to query answering over OWL ontologies is via rewriting the input into axioms expressed in formalisms for which efficient data management and retrieval systems are already available. More precisely, the input ontology $\mathcal{O}$ and query $\mathcal{Q}$ are transformed into a set of sentences $\mathcal{R}$, typically a datalog program (or in some cases even a union of conjunctive queries) called *rewriting*, such that for any dataset $\mathcal{D}$ the answers to $\mathcal{Q}$ w.r.t. $\mathcal{D}$ and $\mathcal{O}$ coincide with the answers to $\mathcal{Q}$ w.r.t. $\mathcal{D}$ and $\mathcal{R}$ discarding $\mathcal{O}$ [22, 9, 39]. Since $\mathcal{R}$ is a (disjunctive) datalog program query answering can be delegated to existing scalable (deductive) database systems.

Computing rewritings has been studied for various fragments of OWL. One of the first approaches supported the language $\mathcal{SHIQ}$ [22], a large fragment of OWL, and the proposed techniques led to the development of KAON2 [35], one of the first practical systems for answering SPARQL queries over OWL ontologies. Recently, the technique has received considerable attention as it consitutes (perhaps) the standard approach to query answering over ontologies expressed in the languages DL-Lite [9, 41], $\mathcal{ELHI}$ [39], and Horn-$\mathcal{SHIQ}$ [14]. DL-Lite and $\mathcal{ELHI}$ are particularly important as they are strongly related to the OWL 2 QL and OWL 2 EL profiles of OWL 2 [33]. Besides the theoretical works many prototype systems have been developed, prominent examples of which include Mastro [10], Presto [41], Quest [40], Rapid [13], Nyaya [36],[1]

☆This is a revised and extended version of the work presented in [13, 44].

*Corresonding Author

Email addresses: despoina@image.ntua.gr (Despoina Trivela), gstoil@image.ece.ntua.gr (Giorgos Stoilos), achort@cs.ntua.gr (Alexandros Chortaras), gstam@cs.ntua.gr (Giorgos Stamou)

[1]Nyaya actually supports linear Datalog$^{\pm}$.

IQAROS [45], and Ontop [31] which support DL-Lite, Requiem [39], which supports $\mathcal{ELHI}$, and Clipper [14], which supports Horn-$\mathcal{SHIQ}$.

Some approaches for computing rewritings have exploited the resolution-based calculi [5]. In this setting, the input is first transformed into a set of clauses which is then saturated using resolution to derive new clauses. The latter can either contain function symbols or be function-free, while the output rewriting consists of all the derived function-free clauses. Using resolution has at least two benefits. First, such calculi are worst-case optimal and allow for a large number of existing optimisations developed in the field of theorem-proving. Second, since there exist many resolution-based decision procedures for expressive fragments of first-order logic [15, 23] it is (relatively) easier to design a resolution-based rewriting algorithm for an expressive fragment of OWL 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 fragments of OWL, while a resolution-based algorithm for all aforementioned fragments exists.

However, the efficiency of resolution-based approaches has also been criticised [42]. Even with all the existing optimisations the saturation produces many clauses unnecessarily. More precisely, it can produce several clauses that contain function symbols and which are not subsequently used to derive other function-free clauses. Since these are neither part of the output rewriting nor do they contribute to the derivation of members of the rewriting their generation is superfluous with respect to query answering. Moreover, exhaustive application of the resolution rule is likely to create long derivations of clauses that are eventually redundant (subsumed) and the standard optimisations of resolution are not enough to provide a scalable approach. Consequently, the first generation systems (e.g., Requiem) have already been surpassed [24].

Motivated by the desire to design efficient rewriting algorithms that can also support expressive fragments of OWL we present novel resolution-based rewriting algorithms. We start from DL-Lite and we show how a rewriting can be computed by greatly restricting the standard (binary) resolution calculus initially used in [38]. Roughly speaking, our calculus generates intermediate clauses that contain function symbols only when it is known that these will contribute to the generation of other function-free clauses. This is implemented by a new resolution inference rule, called *shrinking*, which packages many inference steps into one macro-step and employs certain restrictions over the resolvents.

Subsequently, we extend our approach to the ontology language $\mathcal{ELHI}$ by investigating whether a rewriting algorithm that is again based on the shrinking rule can be defined. This is technically a very challenging task as the structure of $\mathcal{ELHI}$ axioms implies many complex interactions between the clauses (note that, in contrast to DL-Lite, checking concept subsumption in $\mathcal{ELHI}$ is in ExpTime). However, we show that a rewriting can be computed by an algorithm that contains an (arguably small) extension of the shrinking rule of DL-Lite, called *n-shrinking*, plus a new resolution rule, called *function* rule, which captures a very specific type of interaction between roles (binary predicates of the form $R(x, y)$) and their inverses (i.e., $R(y, x)$). Moreover, this new rule is strongly related to the extension of shrinking to n-shrinking. More precisely, if the new rule is never applied, then n-shrinking reduces precisely to the shrinking rule of DL-Lite. Hence, our algorithm has very good "pay as you go" characteristics. That is, if the ontology is expressed in $\mathcal{ELH}$ (i.e., does not allow for inverse roles), then it is guaranteed that the new rule is never applied and n-shrinking can be simplified to shrinking, while the more inverse roles are used in axioms the more the interaction between these two rules, which can create a bottleneck. However, realistic ontologies usually contain few inverse roles, hence we expect that the algorithm would usually behave well in practice. Experimental evaluation and analysis verify our remarks.

Next, we discuss some implementation and optimisation issues which led us to the design and implementation of Rapid, a practical resolution-based system for computing rewritings. More precisely, we discuss how one can present the rewriting in a compact form reducing its size as well as some further optimisation for pruning redundant clauses.

Finally, we conducted an extensive experimental evaluation using a new test suite that includes several real-world large-scale DL-Lite and $\mathcal{ELHI}$ ontologies hence greatly extending all existing benchmarks. Regarding the experiments, our comparison against several state-of-the-art systems has provided many encouraging results. More precisely, our results show that existing systems cannot always handle large-scale and complex ontologies as in several cases they fail to terminate after running for more than 3 hours. In contrast Rapid is in the vast majority of cases able to compute a rewriting within a few seconds. Hence, to the best of our knowledge, Rapid is currently the only system that can handle ontologies of this complexity and size. Yet, there are still many difficult cases that no system can handle.

2. Preliminaries

In this section we introduce the ontology languages $\mathcal{ELHI}$ and DL-Lite which are strongly related to OWL 2 EL and OWL 2 QL respectively; we briefly recall some basic notions from first-order logic and resolution theorem-proving; we provide the definition of conjunctive queries and of query rewriting; and we present an overview of the query rewriting algorithm implemented in the Requiem system since our calculi can be seen as a refinement of this algorithm.

2.1. OWL Ontologies and Description Logics

We focus on OWL (2) ontologies interpreted under the direct semantics which are related to Description Logics (DL) [3]. DLs provide the theoretical underpinning for many fragments of OWL and there is a close connection between the functional syntax of OWL and DLs [21, 19]. For brevity we will adopt the DL notation and terminology; hence, we will call classes and object properties as atomic concepts and roles, respectively.

Let *CN*, *RN*, and *IN* be countable pairwise disjoint sets of atomic concepts, atomic roles, and individuals. $\mathcal{ELHI}$-*concepts* and $\mathcal{ELHI}$-*roles* are defined using the syntax in the

left-hand side of the upper two parts of Table 1, while on the right-hand side the corresponding OWL functional syntax is given. An $\mathcal{ELHI}$*-ontology* $\mathcal{O}$ is a finite set of $\mathcal{ELHI}$*-axioms* of the form depicted in the lower part of Table 1. The Description Logic DL-Lite$_{R,\sqcap}$ (for simplicity DL-Lite in the following) is obtained from $\mathcal{ELHI}$ by disallowing concepts of the form $\exists R.C$ in the left-hand side of axioms. We call axioms of this form *RA*-axioms while all the rest are called DL-Lite-axioms.

Table 1: $\mathcal{ELHI}$-concepts, -role, and -axioms and corresponding OWL functional syntax.

$\mathcal{ELHI}$**-roles**	
R^-	ObjectInverseOf(R)
$\mathcal{ELHI}$**-concepts**	
$\top$	owl:Thing
$C_1 \sqcap C_2$	ObjectIntersectionOf(C_1 C_2)
$\exists R.C$	ObjectSomeValuesFrom(R C)
Axioms	
$C_1 \sqsubseteq C_2$	SubClassOf(C_1 C_2)
$R_1 \sqsubseteq R_2$	SubObjectPropertyOf(R_1 R_2)
$a : C$	ClassAssertion(C a)
$(a, b) : R$	ObjectPropertyAssertion(R a b)

In the following and without loss of generality we assume that ontologies are *normalised* [2, 39], i.e., they contain only axioms 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 $A_{(i)} \in CN \cup \{\top\}$, and $R \in RN \cup \{P^- \mid P \in RN\}$. We also make the standard distinction used in DLs between the schema of an ontology, called TBox $\mathcal{T}$, which consists of all axioms except assertion axioms and the data, called ABox $\mathcal{A}$, which consists of all class and object property assertions.

The semantics of concepts, roles, and axioms in a DL/OWL ontology $\mathcal{O}$ are given by means of interpretations over a domain $\Delta^{\mathcal{J}}$. An interpretation maps individuals to objects of the domain, concepts to subsets of the domain, and roles to sets of pairs of domain objects. Concept $\top$ is mapped to $\Delta^{\mathcal{J}}$ while all other $\mathcal{ELHI}$-concepts and $\mathcal{ELHI}$-roles are mapped to subsets of $\Delta^{\mathcal{J}}$ and $\Delta^{\mathcal{J}} \times \Delta^{\mathcal{J}}$, respectively, using standard conditions listed in [21, 19]. For example, an interpretation $\mathcal{J}$ maps $C_1 \sqcap C_2$ to the intersection of the sets that C_1 and C_2 are mapped to by $\mathcal{J}$—that is, $(C_1 \sqcap C_2)^{\mathcal{J}} = C_1^{\mathcal{J}} \cap C_2^{\mathcal{J}}$. Moreover, $\mathcal{J}$ *satisfies* an $\mathcal{ELHI}$-axiom again if $\mathcal{J}$ satisfies well-known conditions [21, 19]. For example, $\mathcal{J}$ satisfies $C_1 \sqsubseteq C_2$ if $\mathcal{J}$ maps C_1 to a subset of the set that C_2 is mapped to. If all axioms of $\mathcal{O}$ are satisfied by $\mathcal{J}$ then $\mathcal{O}$ is called *satisfiable* (or *consistent*) and $\mathcal{J}$ a *model* of $\mathcal{O}$; otherwise it is called *unsatisfiable* (or *inconsistent*).

Finally, note that in the literature extensions of $\mathcal{ELHI}$ and DL-Lite are considered that allow for disjointness axioms, that is axioms of the form $A_1 \sqsubseteq \neg A_2$ and $R_1 \sqsubseteq \neg R_2$, where $A_{(i)}$ is an atomic concept and $R_{(i)}$ an atomic role or its inverse. The use of such axioms can lead to inconsistencies, e.g., if $\mathcal{T} = \{A \sqsubseteq \neg B\}$ and $\mathcal{A} = \{A(a), B(a)\}$ then $\mathcal{T} \cup \mathcal{A}$ is inconsistent. Query answering over inconsistent ontologies is meaningless unless special techniques and semantics are used [28, 6]. In the current paper we consider query answering only over consistent ontologies; hence axioms like the above ones are superfluous and we have discarded them.

2.2. Resolution-Based Calculi

We use the standard notions of first-order variables, denoted by letters $x, y, z \ldots$, constants, denoted by letters $a, b, c, \ldots$, unary function symbols, denoted by letters $f, g, \ldots$ (e.g., $f(x), g(y), \ldots$), terms which are also denoted by letters $s, t, \ldots$, atoms (e.g., $C(x), R(x, y)$), the entailment relation $\models$, and of substitutions, denoted by letters $\sigma, \mu, \ldots$. The *depth of a term* t is defined as follows: $\mathsf{depth}(t) = 0$, if t is a constant or a variable and $\mathsf{depth}(f(s)) = 1 + \mathsf{depth}(s)$ if t contains a functional symbol. The *depth of an atom* is the maximum depth of its terms.

A list of terms of the form $(s_1, \ldots, s_n)$ is abbreviated by $\vec{s}$, while we also often abbreviate a conjunction of the form $B_1(s) \wedge \ldots \wedge B_n(s)$ by $\mathbf{B}(s)$. In addition, we use the notion of a (Horn) *clause* C, that is, a disjunction of atoms of the form $H \vee \neg B_1 \vee \neg B_2 \vee \ldots \vee \neg B_n$ which can also be written as $H \leftarrow B_1 \wedge B_2 \wedge \ldots \wedge B_n$. H is called the *head* of the clause and the set $\{B_1, \ldots, B_n\}$ is called the *body* and is denoted by $\mathsf{body}(C)$. For an atom A we use $\mathsf{var}(A)$ to denote the set of its variables; var can be extended to clauses in the obvious way. Finally, we use the notation $[B(x)]$ ($[\mathbf{B}(x)]$) to indicate that the presence of atom $B(x)$ (conjunction $\mathbf{B}(x)$) is optional. For example, $A(x) \leftarrow C(x) \wedge [B(x)]$ denotes either $A(x) \leftarrow C(x) \wedge B(x)$ or $A(x) \leftarrow C(x)$.

We also use standard notions from first-order theorem proving, like most general unifier (mgu) [16]. Next, we recapitulate some basic notions. An *inference rule*, or simply *inference* is an $n + 2$-ary relation usually written as follows:

$$\frac{C \quad C_1 \ \ldots \ C_n}{C'}$$

where clause C is called the *main premise*, $C_1, \ldots, C_n$ are called the *side premises* and C' is called the *conclusion* or *resolvent*; both main and side premises are also called *premises*. An *inference system* $\mathcal{I}$, also called *calculus*, is a collection of inference rules. Let Σ be a set of clauses, C a clause and $\mathcal{I}$ an inference system. A *derivation* of C from Σ by $\mathcal{I}$, written $\Sigma \vdash^{\mathcal{I}} C$ (or simply $\Sigma \vdash C$ if $\mathcal{I}$ is clear from the context), is a sequence of clauses $C_1, \ldots, C_m$ such that $C_m = C$, each C_i is either a member of Σ or the conclusion of an inference by $\mathcal{I}$ from $\Sigma \cup \{C_1, \ldots, C_{i-1}\}$. In that case we say that C is *derivable* from Σ by $\mathcal{I}$ and that the derivation *starts* with C_1. We write $\Sigma \vdash_i C$ to denote that the depth of the corresponding *derivation tree* [11] constructed for C from Σ is less than or equal to i. We also often write $\Sigma, C \vdash C'$ instead of $\Sigma \cup \{C\} \vdash C'$. A set of clauses Σ is *saturated* with respect to an inference system $\mathcal{I}$ if the conclusion of any inference by $\mathcal{I}$ from Σ is an element of Σ.

A form of derivation with particular interest to us is *SLD* derivation [29]. An *SLD derivation* of a clause C from a set of clauses Σ is a sequence of clauses $C_1, \ldots, C_n$ such that $C_1 \in \Sigma$, $C_n = C$ and C_{i+1} is a resolvent of C_i and some clause in Σ. For a set of clauses Σ we call *SLD calculus*, $\mathcal{I}^{\Sigma}_{\mathsf{SLD}}$, the inference

system that consists of the (standard) binary resolution rule restricted to producing only SLD derivations as defined before. If it is clear from the context we simply write $\mathcal{I}_{\mathsf{SLD}}$.

2.3. Datalog and Conjunctive Queries

A *datalog clause* r is a function-free Horn clause where the variables occurring in the head also occur in the body. A variable that appears at least twice in the body and not in the head is called *ej-variable*; we use $\mathsf{ejvar}(r)$ to denote all ej-variables of a clause r. A *datalog program* $\mathcal{P}$ is a finite set of datalog clauses.

A *union of conjunctive queries* (UCQ) Q is a set of datalog clauses such that their head atoms share the same predicate Q, called *query predicate*, which does not appear anywhere in the body. A *conjunctive query* (CQ) is a UCQ with exactly one clause. We often abuse notation and identify a CQ with the single clause it contains instead of a singleton set. The variables that appear in the head of a CQ Q are called *answer variables* and are denoted by $\mathsf{avar}(Q)$. For a query Q with query predicate Q, a tuple of constants $\vec{a}$ is a *certain answer* to 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 $\mathcal{T} \cup \mathcal{A} \cup Q \models Q(\vec{a})$. We use $\mathsf{cert}(Q, \mathcal{T} \cup \mathcal{A})$ to denote all certain answers to Q w.r.t. $\mathcal{T}$ and $\mathcal{A}$.

Given two CQs Q_1, Q_2 with head predicates Q_1, Q_2 of the same arity respectively, we say that Q_1 subsumes Q_2 if there exists a substitution θ such that $Q_1\theta = Q_2$ and every atom in $\mathsf{body}(Q_1\theta)$ also appears in $\mathsf{body}(Q_2)$.

2.4. Query Rewriting and the Requiem System

Query rewriting is a prominent technique for answering queries over ontologies. Intuitively, a *rewriting* of a query Q w.r.t. a TBox $\mathcal{T}$ is a set of sentences (usually a datalog program or a UCQ) that captures all the information that is relevant from $\mathcal{T}$ for answering Q over an *arbitrary* ABox $\mathcal{A}$ [9, 39, 41]. This intuition is formalised next.

Definition 1. Let 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 Q w.r.t. $\mathcal{T}$ is a datalog program whose clauses 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 for each $\mathcal{A}$[2] and using only predicates from $\mathcal{T}$ we have:

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

If $\mathcal{R}_D = \emptyset$, then $\mathcal{R}$ is called a *UCQ rewriting*.

Note that if $\mathcal{T}$ is expressed in DL-Lite, then a UCQ rewriting always exists. This property is referred to as *first-order rewritability* [9, 1].

Example 1. Consider the following TBox and query:

$$\begin{aligned} \mathcal{T}_1 &= \{A \sqsubseteq \exists R.B, R \sqsubseteq S, B \sqcap E \sqsubseteq C\} \\ Q_1 &= Q(x) \leftarrow S(x,y) \wedge C(y) \end{aligned}$$

[2]Note that by our previous definitions $\mathcal{A}$ is always consistent w.r.t. $\mathcal{T}$.

Table 2: Translating $\mathcal{ELHI}$ axioms into clauses

Axiom	Clause
$B \sqsubseteq A$	$A(x) \leftarrow B(x)$
$B \sqcap C \sqsubseteq A$	$A(x) \leftarrow B(x) \wedge C(x)$
$\exists R \sqsubseteq A$	$A(x) \leftarrow R(x,y)$
$\exists R^- \sqsubseteq A$	$A(x) \leftarrow R(y,x)$
$A \sqsubseteq \exists R.B$	$R(x, f(x)) \leftarrow A(x)$ $B(f(x)) \leftarrow A(x)$
$A \sqsubseteq \exists R^-.B$	$R(f(x), x) \leftarrow A(x)$ $B(f(x)) \leftarrow A(x)$
$\exists R.C \sqsubseteq A$	$A(x) \leftarrow R(x,y) \wedge C(y)$
$\exists R^-.C \sqsubseteq A$	$A(x) \leftarrow R(y,x) \wedge C(y)$
$P \sqsubseteq R$	$R(x,y) \leftarrow P(x,y)$
$P \sqsubseteq R^-$	$R(x,y) \leftarrow P(y,x)$

It can be verified that the set $\mathcal{R} = \{Q_1, C_1, C_2\}$ where $C_1 = S(x,y) \leftarrow R(x,y)$ and $C_2 = C(x) \leftarrow B(x) \wedge E(x)$ is a datalog rewriting of Q_1 w.r.t. $\mathcal{T}_1$. ◇

Many query rewriting algorithms for various ontology languages have been developed in recent years [41, 40, 14, 45]. Since the focus in the current paper is on resolution-based rewriting algorithms, next, we will briefly introduce the algorithm implemented in the Requiem system [39].

The behaviour of Requiem over a given input CQ Q and TBox $\mathcal{T}$ can be described by the following steps:

1. *Clausification*: First, the input TBox $\mathcal{T}$ is transformed into a set of (Horn) clauses $\mathcal{T}_C$ by using the well-known equivalence of DL axioms with first-order clauses and by skolemising existential variables with new function symbols [4] (see also Example 2). The equivalence of DL axioms with Horn clauses is also depicted in Table 2, where each function symbol f is uniquely associated to the specific occurrence of concept $\exists R.B$ ($\exists R^-.B$).
2. *Saturation*: Next, the clausified TBox together with the input query are saturated by using (binary) resolution parameterised with a selection function [5] producing a new set of clauses $\mathcal{T}_C^{sat}$. When the calculus is applied to the clauses of the form depicted in Table 2 there are only specific types of resolution inferences that can be performed as well as specific types of new clauses that can be produced. The types of clauses that can be derived by Requiem's calculus are given in Table 3. We give the possible interactions among them in Appendix B. Note that in DL-Lite, clauses of type 4.1 and 4.2 can only appear without the conjunct $\mathbf{C}(y)$ and clauses of type 3.3 appear without functions. We refer the reader to [39] for the precise definition of the selection function. In brief, it is defined as follows: for clauses of type 2.1, 2.2 it selects the head atom; for clauses of type 2.3 if there are body atoms that contain a functional term, then all these are selected; otherwise it selects the head atom; for clauses of type 3.1, 3.2, 4.1, 4.2 it selects the role atom in the body; for clauses of type 3.3 it selects the deepest body atom; for clauses of

Table 3: Types of DL-Lite or $\mathcal{ELHI}$ clauses, where $\vec{s}$ is a list of terms $(s_1, s_2, \ldots, s_n)$ and $D_j(\vec{t_j})$ denotes either a concept atom $D_j(t_{j_1})$, or a role atom $D_j(t_{j_1}, t_{j_2})$

Type	Clause
1	$Q(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j})$
2.1	$R(x, f(x)) \leftarrow A(x)$
2.2	$R(f(x), x) \leftarrow A(x)$
2.3	$B(f(x)) \leftarrow \mathbf{C}(x) \wedge [\mathbf{A}(f(x))]$
3.1	$R(x, y) \leftarrow P(x, y)$
3.2	$R(x, y) \leftarrow P(y, x)$
3.3	$A(x) \leftarrow \mathbf{C}(x) \wedge [\mathbf{B}(f(x))]$
4.1	$A(x) \leftarrow R(x, y) \wedge [\mathbf{C}(y)]$
4.2	$A(x) \leftarrow R(y, x) \wedge [\mathbf{C}(y)]$

type 1, if the head contains a functional term, then it selects the head, otherwise it selects the deepest body atom.

3. *Unfolding*: It applies on $\mathcal{T}_C^{sat}$ the so called unfolding step which produces a rewriting of an optimal form [39]. During the unfolding step clauses of type 3.1, 3.2, function-free clauses of type 3.3, and clauses of type 4.1, 4.2 of the form $A(x) \leftarrow R(x, y)$ and $A(x) \leftarrow R(y, x)$, are exhaustively resolved with other function-free clauses. For example, clause $A(x) \leftarrow C(x)$ will resolve with clause $B(x) \leftarrow A(x)$ to obtain $B(x) \leftarrow C(x)$. Although this step is in a sense optional, our proofs of correctness assume that this is also part of the Requiem calculus. Moreover, we consider a slightly extended version of unfolding, where also clauses of the form $A(x) \leftarrow \mathbf{C}(x)$ are considered in the unfolding.
4. *Post-processing*: Finally, Requiem returns all function-free clauses in $\mathcal{T}_C^{sat}$.

We use $\mathcal{I}_{\mathsf{REQ}}$ to denote the calculus used at step 2. and $\mathsf{RQR}(\mathcal{Q}, \mathcal{T})$ to denote the datalog program returned at step 4.

All the previous steps are illustrated through the following example.

Example 2. Consider the TBox and query of Example 1. As stated above the first step is to transform $\mathcal{T}_1$ into a set of clauses. According to Table 2 the clauses that are produced from TBox $\mathcal{T}_1$ are the following:

$$\underline{R(x, f(x))} \leftarrow A(x) \quad (1)$$
$$\underline{B(f(x))} \leftarrow A(x) \quad (2)$$
$$S(x, y) \leftarrow \underline{R(x, y)} \quad (3)$$
$$C(x) \leftarrow \underline{B(x)} \wedge \underline{E(x)} \quad (4)$$

As it can be seen, the axiom $A \sqsubseteq \exists R.B$ produces two clauses that contain the function symbol f and which is uniquely associated with the specific occurrence of concept $\exists R.B$ in axiom $A \sqsubseteq \exists R.B$.

Next, $\mathcal{I}_{\mathsf{REQ}}$ is applied on Q_1 and all clauses (1)-(4), where we have underlined the atoms that Requiem's selection function will select. Then, according to the possible interactions depicted in Table B.10 the following inferences are performed:

$$(3), (1) \vdash \underline{S(x, f(x))} \leftarrow A(x) \quad (5)$$
$$(4), (2) \vdash C(f(x)) \leftarrow A(x) \wedge \underline{E(f(x))} \quad (6)$$
$$Q_1, (5) \vdash Q(x) \leftarrow A(x) \wedge \underline{C(f(x))} \quad (7)$$

Referring to Table B.10 the first inference is of the form 3.1+2.1=2.1, the second one is of the form 3.3+2.3=2.3, while the third one is of the form 1+2.1=1.

It is not hard to see that no new clauses different up to variable renaming can be produced using $\mathcal{I}_{\mathsf{REQ}}$. By discarding the unfolding step mentioned above, the algorithm can terminate and return all function-free clauses, i.e., the set $\mathcal{R} = \{Q_1, (3), (4)\}$. If we also consider the unfolding step, then the algorithm unfolds clause (3) into Q_1 to produce $Q_2 = Q(x) \leftarrow R(x, y) \wedge C(y)$, clause (4) into Q_1 to obtain $Q_3 = Q(x) \leftarrow S(x, y) \wedge B(x) \wedge E(x)$, and finally clause (4) into Q_2 to obtain $Q_4 = Q(x) \leftarrow R(x, y) \wedge B(x) \wedge E(x)$.[3] Then, the rewriting returned would be $\mathcal{R}' = \{Q_1, Q_2, Q_3, Q_4\}$. ◇

Since our topic is resolution algorithms which, to the best of our knowledge, always transform the input into clauses, in the following we assume that ontologies are already given in a clausal form to simplify the presentation. Moreover, clauses that are produced by clausifying *RA*-axioms and DL-Lite-axioms are called *RA*-clauses and DL-Lite-clauses, respectively. These notions are extended to include also clauses of the form $A(x) \leftarrow \mathbf{B}(x)$ and $A(x) \leftarrow R(x, y) \wedge \mathbf{C}(y)$.

3. Resolution-Based Rewriting for DL-Lite

In the current section we present our resolution based rewriting algorithm for DL-Lite. We first illustrate the most important deficiencies of current resolution-based rewriting algorithms which led us to the new design. Then, we present the calculus and finally we show its correctness, i.e., that it computes a rewriting for a given query and DL-Lite-TBox.

Example 3. Consider the TBox $\mathcal{T}_1$ and query Q_1 from Example 2 as well as the inferences performed by $\mathcal{I}_{\mathsf{REQ}}$ on $\mathcal{T}_1 \cup Q_1$. As it can be observed the algorithm performed three unnecessary inferences. More precisely, all inferences producing clauses (5)–(7) are not necessary because they produce clauses that are discarded in the final step (they contain a function symbol) and no inference using either of them as a side or main premise leads to clauses that are members of the rewriting $\mathcal{R}$ for $\mathcal{T}_1, Q_1$.

Assume now that $\mathcal{T}_1$ additionally contained $C(x) \leftarrow B(x)$. This clause resolves with (2) producing $C(f(x)) \leftarrow A(x)$ which subsequently resolves with (7) producing $Q(x) \leftarrow A(x)$. Since the latter clause is function-free it is a member of a rewriting for Q_1 w.r.t $\mathcal{T}_1 \cup \{C(x) \leftarrow B(x)\}$. Clearly, in that case the inferences that produced clauses (5) and (7) are necessary and cannot be omitted. ◇

[3] The last two unfoldings are not performed by the original version of Requiem [39] but as mentioned above we consider this trivial extension here.

As it has been argued [42], TBoxes of real-world ontologies typically contain many clauses of the form $R(x, f_i(x)) \leftarrow A_i(x)$. Together with clauses like clause (3) and Q_1 of Example 2 this implies that typical resolution-based rewriting algorithms, like Requiem's, would produce many clauses of the form $S(x, f_i(x)) \leftarrow A_i(x)$ and $Q(x) \leftarrow A(x) \wedge C(f_i(x))$. Most of these clauses would not subsequently participate in additional inferences hence their generation is superfluous and it would be of benefit to avoid it especially in the case of large ontologies.

In general, allowing the production of clauses containing function symbols is a too fine-grained approach and can lead to the construction of the same clause many times, as shown in the following example.

Example 4. Consider the query $Q_3 = Q(x) \leftarrow S(x, y) \wedge P(y, z) \wedge D(z)$ and the TBox $\mathcal{T}_2 = \mathcal{T}_1 \cup \{C_1, C_2\}$, where $\mathcal{T}_1$ is as defined in Example 2 and C_1, C_2 are as follows:

$$\begin{aligned} C_1 &= P(x, g(x)) \leftarrow B(x) \\ C_2 &= D(g(x)) \leftarrow B(x) \end{aligned}$$

Then, $\mathcal{I}_{\mathsf{REQ}}$ would perform the following inferences:

$$\begin{aligned} Q_3, (5) &\vdash Q(x) \leftarrow A(x) \wedge P(f(x), z) \wedge D(z) && (8) \\ (8), C_1 &\vdash Q(x) \leftarrow A(x) \wedge B(f(x)) \wedge D(g(f(x))) && (9) \\ (9), C_2 &\vdash Q(x) \leftarrow A(x) \wedge B(f(x)) && (10) \\ (10), (2) &\vdash Q(x) \leftarrow A(x) && (11) \\ Q_3, C_1 &\vdash Q(x) \leftarrow S(x, y) \wedge B(y) \wedge D(g(y)) && (12) \\ (12), C_2 &\vdash Q(x) \leftarrow S(x, y) \wedge B(y) && (13) \\ (13), (5) &\vdash (10) \end{aligned}$$

Construction of clauses (8) to (11) corresponds to one path in the derivation while construction of all clauses after (12) to another. We can note that both paths in the derivation lead to the production of the same clause—that is, clause (10), which can then be used to produce clause (11). It is also obvious that the second path is preferable as in addition it leads to the production of the function-free clause (13). ◇

An initial workaround to the above issues can be achieved by applying the calculus $\mathcal{I}_{\mathsf{REQ}}$ in the following way: first, saturate the clauses of $\mathcal{T}$ to obtain a new set of clauses $\mathcal{T}_{sat}$; then, perform only those inferences that if they first introduce a clause that contains an atom with a function symbol in the body, then $\mathcal{T}_{sat}$ also contains some clause that would subsequently "eliminate" this atom and create a function-free clause. In Example 2, this refined strategy would resolve Q_1 with (5) only if clause $C(f(x)) \leftarrow A(x)$ also exists in $\mathcal{T}_{sat}$. If it does, then it is guaranteed that after generating clause (7) that contains conjunct $C(f(x))$ in the body, clause $C(f(x)) \leftarrow A(x)$ can be used as a side premise in an inference to eliminate $C(f(x))$ from (7) and generate the function-free clause $Q(x) \leftarrow A(x)$. Actually, both these inferences can be performed as one macro-inference with main premise Q_1 and side premises (5) and $C(f(x)) \leftarrow A(x)$—that is, two clauses that have the same function symbol in the head. In Example 4, this strategy would only generate clauses (13) and (11) of the second path, while the first part would be completely discarded.

However, to implement this macro-inference approach we need to perform a quadratic loop over the set $\mathcal{T}_{sat}$ to look for pairs of clauses that together introduce and then eliminate some term that contains a function symbol. Since $\mathcal{T}_{sat}$ can be quadratically larger than $\mathcal{T}$ [39] this approach might not scale well in practice.

The difficulty to efficiently implement the above macro-inference step is mostly because the $\mathcal{I}_{\mathsf{REQ}}$ calculus follows a "forward" style approach to apply resolution which generates new clauses that contain function symbols (e.g., clause (5) was generated by propagating f from clause (1) to (3)) hence causing a blow-up in the size of $\mathcal{T}_{sat}$. To implement the aforementioned macro-resolution step in an efficient way we need to pick pairs of clauses from $\mathcal{T}$ rather than $\mathcal{T}_{sat}$. To achieve this our calculus is based on a rather goal-oriented "backwards" style approach that resembles derivations by $\mathcal{I}_{\mathsf{SLD}}$.

Example 5. Consider Q_1 and $\mathcal{T}_1$ from Example 3. Let also the TBox $\mathcal{T}' = \mathcal{T}_1 \cup \{C(x) \leftarrow B(x)\}$.

Assume that we have a calculus that instead of resolving clauses (1) and (3) to propagate the function f (like $\mathcal{I}_{\mathsf{REQ}}$ did in Example 2) it resolves Q_1 with (3) to produce $Q_2 = Q(x) \leftarrow R(x, y) \wedge C(y)$. Subsequently, Q_2 resolves with clause (1) creating $Q(x) \leftarrow A(x) \wedge C(f(x))$ but, as mentioned above, it can additionally check if $\mathcal{T}'$ contains a clause of the form $C(f(x)) \leftarrow A(x)$ which can be used afterwards to create a new resolvent that does not contain $C(f(x))$. Such a clause does not exist so the inference is avoided. Next, clause Q_2 resolves with $C(x) \leftarrow B(x)$ creating $Q_3 = Q(x) \leftarrow R(x, y) \wedge B(y)$. Then, Q_3 can be resolved with (1) since also clause (2) is in $\mathcal{T}'$ and by resolving them in turn with Q_3 we can generate the function-free clause $Q(x) \leftarrow A(x)$. ◇

The crucial difference between $\mathcal{I}_{\mathsf{REQ}}$ and our hypothetical calculus is that, in the latter case, to implement the resolution step we pick clauses from a TBox rather than from its saturation. In addition to being much smaller, a (clausified) DL-Lite-TBox can contain *at-most two* clauses with a term containing the same function symbol. This is because function symbols are unique per occurrence of concepts $\exists R.B$ and no new clauses containing terms with function symbols are generated by the above calculus. The latter can be exploited in the implementation by using proper indexes in order to efficiently retrieve pairs of clauses for the macro-inference step.

All the previous observations motivate our resolution-based rewriting algorithm for DL-Lite ontologies defined next.

Definition 2. Let Q be a CQ and let $C_{(i)}$ be DL-Lite-clause(s). With $\mathcal{I}_{lite}$ we denote the inference system that consists of the following inference rules:

- unfolding:

$$\frac{Q \quad C}{Q'\sigma} \quad \text{where}$$

1. $Q'\sigma$ is function-free resolvent of Q and C, and

2. if $x \mapsto f(y) \in \sigma$, then $x \notin \mathsf{ejvar}(Q)$.

- shrinking:

$$\frac{Q \quad C_1\ [C_2]}{Q'\sigma} \quad \text{where}$$

1. $Q'\sigma$ is a function-free resolvent of Q, C_1, and C_2, and
2. some $x \mapsto f(y) \in \sigma$ exists such that $x \in \mathsf{ejvar}(Q)$.

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

Inferences by the unfolding rule correspond to inferences by (classical) binary resolution where the resolvent is function-free. This is achieved by the condition on σ. In contrast, shrinking is a macro-inference rule that packages many inferences into one and captures the intuition illustrated in our discussion and examples above—that is, it represents derivations of the form $Q, Q_1, \ldots, Q_n, Q'$ where Q is function-free, Q_1 contains some function symbol f and then all subsequent inferences attempt to eliminate from each Q_i all terms containing f until we reach a function-free clause Q'.[4] Since Q is function-free and Q_1 mentions f, the mgu of the first inference must map an ej-variable of Q to a term of the form $f(y)$. This is captured by the second condition on σ. Moreover, as mentioned above, these inferences can involve at most two different side premises that mention f and which are from $\mathcal{T}$. Hence, the side premises in the shrinking rule can be of the form $R(x, f(x)) \leftarrow B(x)$ and $A(f(x)) \leftarrow B(x)$. Furthermore, since the resolvent is function-free, the ej-variable of the main premise is eliminated and since the DL-Lite-clauses do not contain ej-variables in their bodies the resolvent has (strictly) fewer ej-variables than the main premise. Finally, note that the side premises always belong in $\mathcal{T}$ and hence $\mathcal{I}_{lite}$ actually produces SLD derivations.

Applied to $\mathcal{T}'$ and Q_1 from Example 5, the calculus $\mathcal{I}_{lite}$ would perform the steps described in the example. That is, Q_2 is obtained as an unfolding on Q_1 and (3), Q_3 is again produced by unfolding on Q_2 and $C(x) \leftarrow B(x)$, while $Q(x) \leftarrow A(x)$ is obtained by shrinking on Q_3, (2), and (1).

3.1. Correctness of $\mathcal{I}_{lite}$

To show that our rewriting algorithm indeed produces a rewriting we will show that a derivation constructed by $\mathcal{I}_{\mathsf{REQ}}$ can be transformed into a derivation by $\mathcal{I}_{lite}$. Hence, saturating $\mathcal{T} \cup Q$ by $\mathcal{I}_{lite}$ will create all necessary members of a rewriting. To do so we proceed in two steps. First, we show that each Requiem derivation can be transformed into a derivation by $\mathcal{I}_{\mathsf{SLD}}$. This is done by showing how a Requiem inference of the form $Q, C \vdash Q'$ where $\mathcal{T} \vdash_i C$, can be "unfolded" into two inferences of the form $Q, C_1 \vdash Q'', Q'', C_2 \vdash Q'$ where $\mathcal{T} \vdash_{i-1} C_1, \mathcal{T} \vdash_{i-1} C_2$ and $C_1, C_2 \vdash C$. Hence, by repeated applications of this claim we obtain inferences with main premises type 1 clauses and side premises only clauses from $\mathcal{T}$, i.e., a derivation by $\mathcal{I}_{\mathsf{SLD}}$ from $\mathcal{T}$.

[4] Note that the number of these inferences can be more than two if, for example, Q is of the form $Q(x) \leftarrow R(x, y) \wedge R(z, y) \wedge R(w, y) \wedge D(y)$.

Lemma 1. *Let $\mathcal{T}$ be a DL-Lite-TBox and let Q be a CQ. Every type 1 clause derivable from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{REQ}}$ is also derivable from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{SLD}}$.*

Second, we show that each derivation by $\mathcal{I}_{\mathsf{SLD}}$ can be transformed into a derivation by $\mathcal{I}_{lite}$. Since unfolding corresponds to standard binary resolution, the non-trivial part is clearly the shrinking inferences. As stated, an inference by shrinking $Q, C_1, [C_2] \vdash Q'$ corresponds to many inferences of the form $Q, C_{i_1} \vdash Q_1, Q_1, C_{i_2} \vdash Q_2, \ldots, Q_{n-1}, C_{i_n} \vdash Q'$ for $i_j \in \{1, 2\}$, where Q, Q' are function-free and all Q_k, $1 \leq k \leq n-1$ and C_1, C_2 mention the *same* function symbol f. Hence, intuitively, derivations by $\mathcal{I}_{lite}$ are in a sense "compact" with respect to terms containing function symbols—that is, one clause containing a function symbol (Q_1) is created from a function-free clause (Q) and then many inferences follow which try to eliminate the terms introduced using side premises that also mention the same function symbol. We call such derivations function-compact.

Definition 3. Let Σ be a set of Horn clauses and $Q_1, Q_2, \ldots, Q_n$ a derivation by $\mathcal{I}_{\mathsf{SLD}}$ where $Q_i, C_i \vdash Q_{i+1}$ and $C_i \in \Sigma$ for each $1 \leq i < n$. Assume also that all $Q_2, \ldots, Q_{n-1}$ contain a term of the form $f(s_i)$ and of the same depth, and that Q_1 and Q_n are function-free. We say that the derivation is *function-compact* if all side premises C_i with $1 \leq i < n$ used in the derivation also contain a term of the form $f(x_i)$.

Hence, to show correctness of our algorithm we show that any derivation by $\mathcal{I}_{\mathsf{SLD}}$ using Horn clauses as side premises can be transformed into one that is function-compact. This is done by showing that in every non function-compact derivation $Q_1, \ldots, Q_n$, there exists some inference with side premise a clause C_k that does not mention f which can be moved "outside" of the sequence either at the beginning, hence having $Q_1, C_k \vdash Q_2'$, or at the end, hence having $Q_{n-1}, C_k \vdash Q_n$. In the former case, since Q_1 is function-free and C_k does not mention f, Q_2' must be function-free. In the latter case since Q_n is function-free and C_k does not mention f, Q_{n-1}' must be function-free. Hence, this re-ordering either pushes downwards (in the former case) or upwards (in the latter case) all inferences with side premises that mention f. By repeated application of this re-ordering only inferences with side premises that mention f would appear in between the first clause where f was introduced until the last one that mentions it. First, we show a rather general result that is not based on DL-Lite and which will also be used later when we extend the calculus to $\mathcal{ELHI}$.

Lemma 2. *Any derivation built by inferences that have as side premises Horn clauses that can contain terms with function symbols only in the head can be transformed into one that is function-compact.*

The next proposition follows straightforwardly by the form of clauses of DL-Lite-TBoxes.

Proposition 1. *Let $\mathcal{T}$ be a DL-Lite-TBox and Q a query. Any derivation from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{SLD}}$ is produced by using as side premises Horn clauses that have terms with function symbols only in the head.*

Summarising, a derivation from $\mathcal{T} \cup \mathcal{Q}$ by $\mathcal{I}_{\mathsf{REQ}}$ can be transformed into a derivation by $\mathcal{I}_{\mathsf{SLD}}$ which can then be transformed into a function-compact one. Moreover, the inference rules of $\mathcal{I}_{lite}$ are trivially sound as they are based on resolution. These imply that $\mathcal{I}_{lite}$ indeed produces a rewriting for an input CQ and DL-Lite-TBox (see proofs in Appendix C). Finally, termination follows from the fact that there is a bounded number of symbols (variables, constants, roles, and concepts) with which we can construct a clause that can be derived from $\mathcal{I}_{lite}$. More precisely, $\mathcal{I}_{lite}$ always produces function-free clauses, hence no resolvent containing a term of the form $f(x)$ can be produced by $\mathcal{I}_{lite}$. Moreover, every resolvent produced by $\mathcal{I}_{lite}$ has at most the same number of ej-variables as the input query Q: first, only clauses from $\mathcal{T}$ are used as side premises and by Table 3 it is clear that their body atoms that are introduced in the resolvent after the inference, do not contain any new ej-variable. For shrinking, in more detail, the produced resolvent has strictly fewer ej-variables: the main premise must contain two atoms of, e.g., the form $R(x, y)$ and $C(y)$, y is the ej-variable, the side premises are of the form $R(x, f(x)) \leftarrow A(x)$ and $C(f(x)) \leftarrow A(x)$, the mgu maps y to $f(x)$, and since the resolvent is function-free, y is eliminated. An inference via unfolding can, however, increase the number of variables of the main premise, e.g., if the main premise is of the form $Q(x) \leftarrow A(x)$ and the side premise of the form $A(x) \leftarrow R(x, y)$, then the resolvent is of the form $Q(x) \leftarrow R(x, y)$ and y is a new variable. However, y can never become an ej-variable and, moreover, it is necessarily associated with an answer or an ej-variable (e.g., x here). This implies, that for k the number of clauses in $\mathcal{T}$ of the form $A(x) \leftarrow R(x, y)$, the number of these new variables that can be introduced by subsequent unfoldings is bounded by k and the number of answer and ej-variables of the main premise (which as stated is bounded). Hence, by all the previous it follows that the number of clauses produced by $\mathcal{I}_{lite}$ is bounded by the number of answer and ej-variables of the input CQ and the number of symbols and axioms in $\mathcal{T}$.

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

4. Resolution-Based Rewriting for $\mathcal{ELHI}$

In the current section we extend the inference system $\mathcal{I}_{lite}$ in order to provide a resolution-based rewriting algorithm for $\mathcal{ELHI}$ ontologies. Our goal is to design an optimised algorithm that is going to use as much as possible the macro-inference (shrinking) of $\mathcal{I}_{lite}$.

In addition to clauses stemming from DL-Lite axioms, $\mathcal{ELHI}$ also allows for *RA*-clauses, i.e., clauses of the form $E(x) \leftarrow R(x, y) \wedge \mathbf{F}(y)$. A straightforward approach to obtain a calculus for $\mathcal{ELHI}$ ontologies would be to extend Definition 2 to allow for arbitrary *RA*-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 6. Let $\mathcal{T}$ be the $\mathcal{ELHI}$-TBox consisting of the following clauses:[5]

$$C(x) \leftarrow S(x, y) \wedge D(y) \tag{14}$$
$$S(f(x), x) \leftarrow B(x) \tag{15}$$
$$K(x) \leftarrow S(y, x) \wedge C(y) \tag{16}$$

Let also the query $Q_1 = Q(x) \leftarrow K(x)$.

By unfolding on Q_1 and (16) we obtain the clause $Q_2 = Q(x) \leftarrow S(y, x) \wedge C(y)$; then, by unfolding on Q_2 and (14) we obtain the clause $Q_3 = Q(x) \leftarrow S(y, x) \wedge S(y, z) \wedge D(z)$; finally, by shrinking on Q_3 and (15) we can obtain the clause $Q_4 = Q(x) \leftarrow B(x) \wedge D(x)$. It can be verified that $\mathcal{R} = \{Q_1, Q_2, Q_3, Q_4\}$ is a rewriting of Q w.r.t. $\mathcal{T}$. ◇

However, as the previous example shows, using *RA*-clauses as side premises can produce resolvents that contain more variables than the main premise of the inference (e.g. clause Q_3 contains a variable z that does not appear in Q_2) and hence to variable proliferation which implies termination problems. In general, one could remedy this issue by deciding a bound on the number of times (variables) that such clauses can be used as side premises (introduced to the resolvent). This problem is loosely related to query and predicate boundedness and although some recent results have been obtained [7], most of the proposed algorithms are based on complex automata which, to the best of our knowledge, have not yet been implemented.

Consequently, to provide an efficient and terminating algorithm the calculus should not allow *RA*-clauses as side premises. But then, to be able to deal with $\mathcal{ELHI}$ ontologies the new algorithm would need to produce intermediate clauses that are derivable by *RA*-clauses in a similar way to the standard $\mathcal{I}_{\mathsf{REQ}}$ calculus. The next example shows how $\mathcal{I}_{\mathsf{REQ}}$ would behave when applied to the input of Example 6 which we will use to illustrate the extensions that are needed to $\mathcal{I}_{lite}$.

Example 7. Consider the TBox $\mathcal{T}$ and query Q_1 from Example 6. When applied to $\mathcal{T}$ and Q_1 the Requiem algorithm would perform the following inferences with the respective conclusions:

$$(14), (15) \vdash C(f(x)) \leftarrow B(x) \wedge D(x) \tag{17}$$
$$(16), (15) \vdash K(x) \leftarrow B(x) \wedge C(f(x)) \tag{18}$$
$$(18), (17) \vdash K(x) \leftarrow B(x) \wedge D(x) \tag{19}$$
$$Q_1, (19) \vdash Q(x) \leftarrow B(x) \wedge D(x) \tag{20}$$

The set $\mathcal{R} = \{Q_1, (14), (16), (20)\}$ is a (datalog) rewriting of 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 (19) from clause (16) can be captured by a single shrinking inference over clause (16) with side premises (15) and (17). However, clause (17) that is used as side premise in this shrinking inference does not belong in $\mathcal{T}$ but is produced by resolving an *RA*-clause together with a DL-Lite-clause. This interaction cannot be captured by any of the rules of $\mathcal{I}_{lite}$. ◇

[5] Note that these clauses are produced by clausifying the DL axioms $\exists S.D \sqsubseteq C$, $B \sqsubseteq \exists S$, and $\exists S^{-}.C \sqsubseteq K$, respectively.

Motivated by the above example, our calculus for $\mathcal{ELHI}$-ontologies consists, first, of a new inference rule which can produce clauses like clause (17) from *RA*-clauses and clauses of the same form as clause (15) and, second, of an extension of unfolding and shrinking to allow as main premises besides type 1 clauses also *RA*-clauses, hence being able to compute, e.g., clause (19) from (16), (15), and (17). Because of the new inference rule that can produce new clauses with function symbols in the head (cf. clause (17)), there can now be more than two clauses that can mention the same function symbol f. Hence, shrinking has to be extended further to allow for (possibly) n side premises instead of at most two.

Our calculus for $\mathcal{ELHI}$-TBoxes is defined next.

Definition 4. Let Υ be either a CQ or an *RA*-clause and let $C_{(i)}$ be DL-Lite-clause(s). With $\mathcal{I}_{\mathcal{EL}}$ we denote the inference system consisting of the following rules:

- unfolding:

$$\frac{\Upsilon \quad C}{\Upsilon'\sigma} \quad \text{where}$$

  1. $\Upsilon'\sigma$ is a function-free resolvent of Υ and C, and
  2. if $x \mapsto f(y) \in \sigma$ then $x \notin \mathsf{ejvar}(\Upsilon)$.

- n-shrinking:

$$\frac{\Upsilon \quad C_1\ [C_2\ \ldots\ C_n]}{\Upsilon'\sigma} \quad \text{where}$$

  1. $\Upsilon'\sigma$ is a function-free resolvent of Υ and all $C_1, \ldots, C_n$ for $n \geq 1$, and
  2. some $x \mapsto f(y) \in \sigma$ exists such that $x \in \mathsf{ejvar}(\Upsilon)$.

- function:

$$\frac{B(x) \leftarrow R(x,y) \wedge [\mathbf{C}(y)] \quad R(f(x),x) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge [\mathbf{C}(x)]} \quad \text{or}$$

$$\frac{B(x) \leftarrow R(y,x) \wedge [\mathbf{C}(y)] \quad R(x,f(x)) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge [\mathbf{C}(x)]}$$

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

Note that an inference by n-shrinking with more than 2 side premises is only possible if the function rule has been previously "fired" to produce new type 2.3 clauses with some function symbol in the head; however, the function rule captures a quite complex interaction between a clause containing $R(x, f(x))$ ($R(f(x), x)$) and an *RA*-clause containing the inverse $R(y, x)$ ($R(x, y)$) which, as shown next in our experimental evaluation, does not happen often in practice. Moreover, note that the application of the calculus $\mathcal{I}_{\mathcal{EL}}$ over some input $\mathcal{T} \cup Q$ can be partitioned into two phases. First, we can saturate $\mathcal{T}$ by $\mathcal{I}_{\mathcal{EL}}$ having as a main premise only *RA*-clauses. Second, we can collect all DL-Lite-clauses from $\mathcal{T}$ and those produced in the previous step and use them as side premises in inferences of $\mathcal{I}_{\mathcal{EL}}$ (modulo the function rule) with main premise only type 1 clauses. All the above imply excellent "pay as you go" properties for $\mathcal{I}_{\mathcal{EL}}$. That is, in case $\mathcal{T}$ is expressed in DL-Lite the first phase is simply omitted; if $\mathcal{T}$ is expressed in $\mathcal{ELH}$, i.e., it does not contain inverse roles, then the function rule is never applied and n-shrinking can be restricted to $n = 2$, i.e., to the DL-Lite shrinking. Finally, note that the number of inverse roles and *RA*-clauses in $\mathcal{T}$ is likely to affect the number of times the function rule is applied and hence the parameter n of the n-shrinking rule.

Example 8. Consider Example 7. The inference between clauses (14) and (15) that produces clause (17) corresponds to an inference using the function rule. Then, clause (19) can be produced by n-shrinking over (16) with side premises clauses (15) and (17), while (20) is produced by unfolding on Q and (19), hence computing the required rewriting. ◇

4.1. Correctness of $\mathcal{I}_{\mathcal{EL}}$

As mentioned in Section 3 correctness of $\mathcal{I}_{lite}$ is based on showing that derivations by $\mathcal{I}_{\mathsf{REQ}}$ can be transformed into derivations by $\mathcal{I}_{lite}$. Since $\mathcal{I}_{lite}$ always uses as side premises clauses from $\mathcal{T}$, our first intermediate step was to show that derivations by $\mathcal{I}_{\mathsf{REQ}}$ can be unfolded into derivations by $\mathcal{I}_{\mathsf{SLD}}$ which are closer to the derivations produced by $\mathcal{I}_{lite}$. In a similar way, we also show here that derivations by $\mathcal{I}_{\mathsf{REQ}}$ can be transformed into extended forms of SLD derivations and then into derivations by $\mathcal{I}_{\mathcal{EL}}$. This will be done by analysing the structure of the derivations produced by $\mathcal{I}_{\mathcal{EL}}$ and by taking again the two step approach.

Consider an inference of the form $\Upsilon, C \vdash \Upsilon'$, where $\mathcal{T} \vdash^{\mathcal{I}_{\mathsf{REQ}}} C$. If the derivation of C does not involve an *RA*-clause, then (like in DL-Lite) we can show that Υ' can be derived from Υ by $\mathcal{I}_{\mathsf{SLD}}$ using as side premises only DL-Lite-clauses. This can be done via our unfolding technique of $\Upsilon, C \vdash \Upsilon'$ into $\Upsilon, C_1 \vdash \Upsilon_2, \ldots, \Upsilon_{n-1}, C_n \vdash \Upsilon'$ where for each $1 \leq i \leq n$ we have $C_i \in \mathcal{T}$ and C_i is a DL-Lite-clause. In a different case, as explained in the previous section, the calculus $\mathcal{I}_{\mathcal{EL}}$ is expected to produce new clauses using the new inference rules that do not allow for *RA*-clauses as side premises. Hence, to match the derivations constructed by $\mathcal{I}_{\mathcal{EL}}$ the previous inference should be unfolded up to a certain level—that is, for some C_k we might have $C_k \notin \mathcal{T}$ and C_k might be derivable by a sequence that begins with an *RA*-clause. By inspecting Definition 4 we can see that $\mathcal{I}_{\mathcal{EL}}$ can produce either of the following non-type 1 clauses, hence, unfolding should stop when either of the following cases occurs:

1. C_k is of type 2.3 of the form $A(f(x)) \leftarrow B(x) \wedge [\mathbf{C}(x)]$ produced by the function rule over an *RA*-clause, or
2. C_k is a function-free type 3.3 clause produced by n-shrinking and unfolding starting with an *RA*-clause. This is because from an *RA*-clause of the form $A(x) \leftarrow R(x, y) \wedge \mathbf{B}(y)$ n-shrinking can produce a clause of the form $A(x) \leftarrow C(x) \wedge [\mathbf{D}(x)]$, i.e., a function-free type 3.3. Note that this inference captures many inference steps which start from the *RA*-clause and then several type 3.3

clauses containing a function symbol follow until we reach the function-free type 3.3 clause.

Consequently, to show correctness we first show that derivations by $\mathcal{I}_{\mathsf{REQ}}$ can be *partially* unfolded into an extended form of SLD derivations, which (modulo the macro-inference) resemble those produced by $\mathcal{I}_{\mathcal{EL}}$. These derivations are built by using inferences which allow as side premises DL-Lite-clauses from $\mathcal{T}$, clauses produced by the function rule (case 1. above), or function-free clauses of type 3.3 possibly produced previously (case 2. above).

Definition 5. Let Σ be a set of Horn clauses. An *extended*-SLD derivation from Σ is a sequence of clauses $C_1, \ldots, C_n$ such that each C_i can be one of the following:

- a type 1 or an *RA*-clause from Σ; or
- the conclusion of an inference by binary resolution having as a main premise a type 1, type 3.3, or *RA*-clause from $\{C_1, \ldots, C_{i-1}\}$ and as a side premise a clause from Σ, a function-free type 3.3 clause from $\{C_1, \ldots, C_{i-1}\}$, or a clause produced by the function rule with side premise from Σ.

A system producing such derivations is denoted by $\mathcal{I}_{\mathsf{SLD}^+}$.

Note that the first condition allows a kind of restart step, i.e., one can copy some *RA*-clause from Σ into the sequence and use it for deriving new clauses.

Next we show that indeed each derivation by $\mathcal{I}_{\mathsf{REQ}}$ can be transformed into an extended-SLD derivation.

Lemma 3. *Let $\mathcal{T}$ be an $\mathcal{ELHI}$-TBox and let Υ be a CQ (resp. RA-clause). Then, every type 1 clause (resp. type* 3.3 *clause) derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{REQ}}$ starting with Υ is also derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$ starting with Υ.*

Moreover, it follows by the structure of extended-SLD derivations that side premises can only contain function symbols in the head.

Proposition 2. *Let $\mathcal{T}$ be an $\mathcal{ELHI}$-TBox and Q a query. Any extended-SLD derivation from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{SLD}^+}$ is produced by using as side premises clauses that have function symbols only in the head.*

Proof. By Definition 5 $\mathcal{I}_{\mathsf{SLD}^+}$ uses as side premises either clauses from $\mathcal{T}$, which can contain function symbols only in the head, clauses produced by the function rule, which by Definition 4 only contain function symbols in the head, or function-free type 3.3 clauses. □

Consequently, Lemma 2 applies and any extended-SLD derivation of function-free type 1 or type 3.3 clauses can be transformed into a function-compact one. Similarly to the case of DL-Lite, the above together with soundness of the inference rules can be used to show correctness of $\mathcal{I}_{\mathcal{EL}}$. Finally, regarding termination we have the following: first, only a finite number (up to renaming of variables) of *RA*-clauses can be produced (these can be produced only by unfolding); second, the function rule can also be applied only a finite number of times (its side premise is *always* a clause from $\mathcal{T}$ and its main premise is an *RA*-clause and as stated there can only be a finite number of them); third, like in DL-Lite, n-shrinking and unfolding on either type 1 or *RA*-clauses can only produce a finite number of clauses.

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

5. Practical Implementation and Optimisations

A resolution calculus provides a general mechanism for reasoning over a given knowledge base, while to provide with a well-behaved practical implementation several aspects need to be taken into consideration like the strategy of rule application. In the current section we discuss some implementation and optimisation issues that help us provide a well-behaved (optimised) rewriting algorithm that is based on the calculus $\mathcal{I}_{lite}$.

As illustrated by the following example, inferences using the unfolding rule can, in many cases, lead to the generation of redundant or even previously computed queries.

Example 9. Let $\mathcal{T}$ be a TBox consisting of the following clauses:

$$R(x, f(x)) \leftarrow A(x) \tag{21}$$
$$C(f(x)) \leftarrow A(x) \tag{22}$$
$$D(x) \leftarrow C(x) \tag{23}$$
$$S(x, y) \leftarrow R(x, y) \tag{24}$$

and let also the query $Q_1 = Q(x) \leftarrow A(x) \wedge S(x, y) \wedge D(y)$.

By applying $\mathcal{I}_{lite}$ on $\mathcal{T} \cup \{Q_1\}$ we obtain the following inferences and the respective conclusions:

$$Q_1, (23) \vdash Q(x) \leftarrow A(x) \wedge S(x, y) \wedge C(y) \tag{25}$$
$$Q_1, (24) \vdash Q(x) \leftarrow A(x) \wedge R(x, y) \wedge D(y) \tag{26}$$
$$(25), (24) \vdash Q(x) \leftarrow A(x) \wedge R(x, y) \wedge C(y) \tag{27}$$
$$(26), (23) \vdash (27)$$
$$(27), (21), (22) \vdash Q(x) \leftarrow A(x) \tag{28}$$

It can be verified that the set containing Q_1 and (25)–(27) is a UCQ rewriting for Q_1 w.r.t. $\mathcal{T}$.

However, we can note that query (27) is produced twice. In large ontologies this issue can occur quite often and hence adversely affect performance. Second, we can also note that query (28) subsumes all queries Q_1 and (25)–(27). Ideally, if we could compute query (28) directly from Q_1 and then identify that it subsumes Q_1, we could subsequently discard Q_1 and hence also avoid constructing all queries (25)–(27). ◇

In general, queries produced via shrinking are likely to subsume queries produced by unfolding, since the former contain less variables than the latter (recall that shrinking eliminates an ej-variable). A technical difficulty is that shrinking might only

be applicable after first applying several unfolding inferences. More precisely, in the previous example query (28) can only be produced after first generating query (27) via unfolding.

To minimise any redundancies due to unfolding our algorithm does not explicitly construct such queries by applying the rule. Instead, it computes only a minimal amount of information that is sufficient to actually construct them. This is made precise in the following definition.

Definition 6. Let $\mathcal{T}$ be a DL-Lite-TBox, let Q be a CQ, and let A be some atom in the body of Q. Let also Q_A be a query such that $\mathsf{body}(Q_A) = \{A\}$ and $\mathsf{avar}(Q_A) = \mathsf{var}(A) \cap \mathsf{ejvar}(Q)$. The *unfolding set* of A w.r.t. $Q, \mathcal{T}$ is the set defined as follows:

$$\{\mathsf{body}(Q') \quad | \quad Q' \text{ derivable from } \mathcal{T} \cup Q_A \text{ using only the unfolding rule}\}$$

Clearly, for a query Q the unfolding sets of its atoms fully characterise the queries that are derivable from Q via unfolding.

Proposition 3. *Let $\mathcal{T}$ be a DL-Lite-TBox, and let Q be a CQ with body atoms $A_1, \ldots, A_n$ and respective unfolding sets $S_1, \ldots, S_n$. A query Q' can be derived from $\mathcal{T} \cup Q$ via unfolding iff for every S_i there exists $A \in S_i$ such that $A \in \mathsf{body}(Q')$.*

Example 10. Consider the TBox $\mathcal{T}$ and query Q_1 from Example 9. The unfolding set S_D of $D(y)$ w.r.t. Q_1 contains all body atoms of queries produced via unfolding from $\mathcal{T} \cup \{Q_D(y) \leftarrow D(y)\}$, that is, the body atoms of queries $Q_D(y) \leftarrow D(y)$ and $Q_D(y) \leftarrow C(y)$. Similarly, the unfolding set S_S of $S(x, y)$ w.r.t. $Q_1, \mathcal{T}$ contains the body atoms of $Q_S(x, y) \leftarrow S(x, y)$ and $Q_S(x, y) \leftarrow R(x, y)$, while the unfolding set S_A of $A(x)$ contains only $A(x)$.

It can be seen that all queries (25)–(27) of Example 9 can be obtained from the previous unfolding sets. For example, query (27) can be constructed by atom $A(x)$ from S_A, atom $R(x, y)$ from S_S, and atom $C(y)$ from S_D. ◇

Using the information computed in the unfolding sets, our algorithm constructs queries that would otherwise be computed later on via shrinking, directly and without explicitly performing all intermediate steps.

Example 11. Consider again Example 9 and the query Q_1. Its body atoms $S(x, y)$ and $D(y)$ share the ej-variable y, while the unfolding set of $S(x, y)$ contains $R(x, y)$ and the unfolding set of $D(y)$ contains $C(y)$. Moreover, the TBox contains clauses $R(x, f(x)) \leftarrow A(x)$ and $C(f(x)) \leftarrow A(x)$ and their body atoms are unifiable. Therefore, we can conclude that there exists a query constructed after several steps of unfolding (i.e., query (27) of Example 9) over which shrinking with side premises clauses (21) and (22) would be applicable. Hence, from Q_1 we can directly construct query (27) by replacing $S(x, y)$ and $D(y)$ with atom $A(x)$. Finally, the algorithm can detect that query (27) subsumes Q_1 and hence none of the queries (25)–(27) are generated. ◇

We stress that computing the unfolding sets is in most cases much more efficient than directly computing the queries derivable by unfolding. First, the queries in the unfolding set contain very few atoms (typically at most one); hence, inferences can be performed more efficiently. Second, by proper variable renamings previously computed unfolding sets can be reused. For example, the unfolding set of some atom $D(y)$ can be readily used to compute the unfolding set of the atom $D(z)$ by renaming all y to z. Third, by avoiding computing these queries explicitly we avoid the redundancy issue highlighted in Example 9.

In case a query Q is not subsumed by the one produced via shrinking, the algorithm can use the unfolding sets to explicitly compute all the respective queries as shown in Example 10. However, such a step can clearly create an exponential number of queries (one needs to take all possible combinations of atoms from all unfolding sets). Alternatively, the system can provide a more compact representation of the rewriting by encoding the information in the unfolding sets using datalog clauses. In Example 10 the unfolding set of $D(y)$ contains $C(y)$. Hence, instead of using $C(y)$ to compute new queries from Q_1 the algorithm can return the datalog clause $r_1 = D(x) \leftarrow C(x)$. Similarly, for the unfolding set for $S(x, y)$ the algorithm can encode this information via the datalog clause $r_2 = S(x, y) \leftarrow R(x, y)$. It can be verified that $\{Q_1, r_1, r_2\}$ is a datalog rewriting of Q_1 w.r.t. $\mathcal{T}$.

However, there may be cases where one wants to compute all the respective queries using the unfolding sets (e.g., if one wants to compute a UCQ rewriting if one exists). Since as stated above this process can be exponential it should be optimised as much as possible. As the following example shows, the information in the unfolding sets can also be used to easily identify whether some of these queries will be redundant and hence discard them.

Example 12. Consider query $Q_1 = Q(x) \leftarrow A(x) \wedge D(x)$ and assume the unfolding sets $\{A(x), E(x)\}$ and $\{D(x), E(x)\}$ for atoms $A(x)$ and $D(x)$, respectively. According to these unfolding sets the queries that are generated via unfolding are (amongst others) $Q_2 = Q(x) \leftarrow E(x) \wedge D(x)$, $Q_3 = Q(x) \leftarrow A(x) \wedge E(x)$, and $Q_4 = Q(x) \leftarrow E(x)$. It can be easily seen that Q_4 subsumes Q_2 and Q_3. Intuitively, this is because both unfolding sets contain the atom $E(x)$. Hence, any query constructed by picking atom $E(x)$ from one of the two unfolding sets and another atom different than $E(x)$ from the other, will eventually be subsumed by the query that is constructed by picking $E(x)$ from both sets. ◇

Our intuition in the previous example is formalised in the following proposition which can be used by the algorithm in order to avoid generating queries that would eventually be redundant in the result.

Proposition 4. *Let $\mathcal{T}$ be a DL-Lite TBox, and let Q be a CQ with body atoms $A_1, \ldots, A_n$ and the respective unfolding sets $S_1, \ldots, S_n$. Let also a query Q' obtained via unfolding from $\mathcal{T} \cup Q$. If there exist atoms $\{B_i, B_j\} \subseteq \mathsf{body}(Q')$, an unfolding set S_i such that $\{B_i, B'_i\} \subseteq S_i$, and a mapping $\theta : \mathsf{var}(B'_i) \setminus \mathsf{ejvar}(Q) \mapsto \mathsf{var}(Q')$ such that $B'_i\theta = B_j$, then there exists a query Q'' obtained via unfolding from $\mathcal{T} \cup Q$ that subsumes Q'.*

Proof. Let Q' be derived via unfolding from $\mathcal{T} \cup Q$ and assume that for some $\{B_k, B_\ell\} \subseteq \mathsf{body}(Q')$, there exists an unfolding set

S_k with atoms $\{B_k, B'_k\} \in S_k$, and there also exists $\theta : \mathsf{var}(B'_k) \setminus \mathsf{ejvar}(Q) \mapsto \mathsf{var}(Q')$ such that $B'_k\theta = B_\ell$.

Now, let Q'' be the query constructed from Q' by replacing atom B_k with atom B'_k. By the definition of unfolding sets, it is easy to see that $\mathsf{ejvar}(Q'') \subseteq \mathsf{ejvar}(Q)$; hence θ only maps variables of B'_k. Consequently, for each $i \neq k$ we have $B_i\theta = B_i$ while for B'_k we have $B'_k\theta = B_\ell$; thus, $Q''\theta \subseteq Q'$. Finally, by Proposition 3 it also follows that Q'' can be derived from $\mathcal{T} \cup Q$ via unfolding, as required. □

The above lemma gives sufficient conditions for deciding whether a query is redundant. The following example shows that indeed a query can be redundant without satisfying the conditions of the proposition.

Example 13. Let the query $Q = Q(x) \leftarrow R(x, y) \wedge A(y) \wedge P(x, z) \wedge A(z)$ and assume that for some TBox $\mathcal{T}$ we have the unfolding sets $\{R(x, y), P(x, y)\}$ and $\{P(x, z), R(x, z)\}$ for atoms $R(x, y)$ and $P(x, z)$, respectively. Hence, by Proposition 3 the query $Q' = Q(x') \leftarrow P(x', y') \wedge A(y') \wedge R(x', z') \wedge A(z')$ can be produced by unfolding on Q. For $\theta = \{y \mapsto z', z \mapsto y'\}$ we clearly have $Q\theta \subseteq Q'$, however, $y, z \in \mathsf{ejvar}(Q)$ hence Q' does not satisfy conditions of Proposition 4. ◇

However, note that in all previous examples we have that two different unfolding sets contain an atom with the same predicate name (e.g., E in $E(x)$ of Example 12 and R and P in Example 13). If this does not happen (i.e., all unfolding sets contain different predicate names) then we can deduce that the queries produced by them are non-redundant. Our algorithm uses such strategies to identify non-redundant queries in a conservative way which can very often speed up the algorithm.

6. Evaluation

We have implemented the calculus $\mathcal{I}_{\mathcal{EL}}$ together with the optimisations outlined in Section 5 in our prototype tool Rapid[6] [13, 44]. Our system can either output the unfolding sets as a datalog program or optionally create the respective queries attempting to construct a UCQ rewriting (see Section 5).

We evaluated Rapid by comparing it against the query rewriting systems Requiem [39], Presto [41], and Clipper [14] (we did not use Presto in the $\mathcal{ELHI}$ ontologies as it only supports DL-Lite). In every experiment Rapid uses a final backwards subsumption deletion step, while no other system does; both Clipper and Presto do not apply it by default and we used Requiem in the "naive" mode which does not apply it. Only in the experiments reported in Table 5 we used the "greedy" mode of Requiem that does apply backwards subsumption.

We have significantly extended the existing benchmark suites for query rewriting systems by including many real-world large scale and complex ontologies. In particular, we used DL-Lite versions of the OBO Protein,[7] NCI 3.12e,[8] and the OpenGALEN2[9] ontologies, and $\mathcal{ELHI}$ versions of the LUBM,[10] OBO Protein, NASA SWEET 2.3,[11] PERIODIC Table,[12] NotGALEN,[13] GALEN-Doctored [20], the OpenGALEN2, and the DOLCE 397 ExtendedDNS[14] ontologies. Table 4 provides statistics for the ontologies. For many of these ontologies no test queries exist, hence we manually constructed five. Further statistics and details about them can be found in Appendix A. All the previous ontologies and queries are available online.[6] In addition, we also attempted to use the query generator proposed in [24] to automatically construct test queries for them. Unfortunately, due to the complexity of the used ontologies in most cases the tool failed to terminate. Even after imposing various bounds, the tool produced such a large set of test queries (hundreds of thousands) that would be practically impossible to perform all these tests with all systems. Hence, we considered its output only for NASA SWEET (706 queries) and the ExtendedDNS ontology (7603 queries). The results for these queries are presented in Table 8. Finally, we also used the DL-Lite ontologies and test queries proposed in [38].

Table 4: Statistics of the used test ontologies

Ontology	♯Concepts	♯Roles	♯GCIs	♯RIAs
DL-Lite				
OBO Protein	35351	6	43351	0
NCI	29173	66	53341	0
OpenGALEN2	23193	851	49046	882
$\mathcal{ELHI}$				
OBO Protein	37560	6	52383	0
NASA SWEET	4278	535	6004	411
PERIODIC Table	4282	22	9564	15
NotGALEN	5252	413	10551	416
GALEN-Doctored	4670	413	8140	416
OpenGALEN2	30048	851	63726	882
ExtendedDNS	168	186	664	189

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 3 hours.

In all subsequent tables column O indicates the name of the ontology, column "Time" the time to produce a rewriting (discarding loading the inputs into the systems), and column "Rewriting size" the number of clauses that the computed rewriting contains. Moreover, "t/o" denotes a timeout.

6.1. DL-Lite

In Table 5 we present the results for some of the ontologies and queries in the test suite proposed in [38]. These ontologies

[6] http://www.image.ece.ntua.gr/~achort/rapid/

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

[8] http://evs.nci.nih.gov/ftp1/NCI_Thesaurus

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

[10] http://swat.cse.lehigh.edu/projects/lubm

[11] http://sweet.jpl.nasa.gov/ontology

[12] http://www.cs.man.ac.uk/~stevensr/ontology

[13] http://www.cs.ox.ac.uk/isg/ontologies/lib/GALEN/not-galen

[14] http://www.loa.istc.cnr.it

Table 6: Evaluation results for large DL-Lite ontologies

	Time (hh:mm:ss.msec)				Rewriting size			
$\mathcal{O}$	Rapid	Requiem	Presto	Clipper	Rapid	Requiem	Presto	Clipper
OBO Protein	0.154	4.781	59:09.975	1:04.782	29	27	48	29
	1.160	45.473	1:04:36.706	1:07.770	1356	1356	2621	1356
	7.264	9:51.364	1:17:22.561	1:07.084	33919	33887	33888	33919
	10.613	12:31.979	59:35.577	1:03.496	34879	34733	35416	34879
	5:26.680	t/o	1:09:05.760	1:08.245	27907	t/o	2670	54430
NCI	0.057	4.423	1:58:34.415	11:49.193	488	5002	469	488
	0.043	36.895	2:02:14.930	12:00.461	1804	1765	1766	1804
	0.154	t/o	2:02:32.860	13:16.689	4143	t/o	3546	4143
	0.063	1:17:51.520	2:04:19.429	13:55.669	1875	219150	1917	1875
	0.041	9:39.122	2:09:21.690	13:27.232	256	64500	208	340
OpenGALEN2	0.001	0.024	t/o	t/o	3	2	t/o	t/o
	0.099	6:04.373	t/o	t/o	1276	1152	t/o	t/o
	0.001	0.111	t/o	t/o	18	16	t/o	t/o
	0.004	5:46.020	t/o	t/o	155	147	t/o	t/o
	0.001	0.022	t/o	t/o	1	1	t/o	t/o

Table 5: Evaluation results using Requiem's test suite

	Time (hh:mm:ss.msec)			Rewriting size		
$\mathcal{O}$	Rapid	Requiem	Presto	Rapid	Requiem	Presto
P5X	0.004	0.012	0.026	14	14	14
	0.021	0.137	0.103	25	25	26
	0.022	0.313	0.203	58	58	37
	0.040	4.160	2.005	179	179	82
	0.445	2:17.683	51.538	718	718	251
UX	0.002	0.023	0.413	5	5	5
	0.010	0.170	0.047	1	1	1
	0.013	0.691	0.049	12	12	12
	0.003	10.317	0.043	5	5	5
	0.005	34.232	0.052	25	25	25
AX	0.016	0.028	1.856	41	41	41
	0.158	1.595	4.108	1431	1431	1431
	0.108	16.751	1:00.466	4466	4466	4466
	0.389	13.068	33.729	3159	3159	3159
	0.964	1:14:24	1:21:06	32921	32921	36330

are relatively small and simple and hence we present the results only for the most interesting ones. Because of their simple structure a UCQ rewriting can be computed in reasonable time. The purpose of this evaluation is mostly to illustrate Rapid's performance for computing a UCQ rewriting when using the optimisations described in Section 5. The results show that Rapid has either similar performance to the other systems or it is much faster, as e.g. in query 5 over AX requires only 1 second while Requiem and Presto require about 1 hour. The analysis showed that Requiem spends most of the time in the final backwards subsumption step (recall that for these experiments we used the greedy mode of Requiem that applies subsumption). For example, in query 5 over ontology P5X, rewriting takes around 45 seconds while subsumption around 1 minute and 30 seconds while for query 5 over ontology AX rewriting takes around 14 minutes while backwards subsumption around 1 hour. Hence, even without the final subsumption step Requiem is considerably slow. In contrast, due to the optimisations illustrated in Section 5 Rapid can identify subsumed queries very efficiently and produce the minimal UCQ rewriting.

The results for the large DL-Lite ontologies are shown in Table 6. Since these ontologies are quite large we ran all systems in the mode of computing compact datalog rewritings. First, we observe that neither Presto nor Clipper managed to compute a rewriting for OpenGALEN2, Requiem required up to 6 minutes depending on the query, while Rapid required only milliseconds. Similar observations can be made for the the other two ontologies. Notable cases are all queries over NCI for Clipper, where it required 12-14 minutes for each of them, query 5 over OBO Protein and query 3 over NCI for Requiem, for which it did not manage to compute a rewriting, and all queries over OBO Protein and NCI for Presto for which it required about 1 and 2 hours, respectively. Rapid was slower only in query 5 over OBO Protein, for which it required 5:26 minutes. However, most of this time was spent to the final backwards subsumption deletion that Rapid applies (which guarantees a compact result with no duplicate or subsumed queries), while the actual rewriting time was only a few seconds. Recall that, no other system performs backwards subsumption and hence, in contrast to Rapid, in query 5 over OBO Protein Clipper computes a rewriting that is twice the size of the one computed by Rapid, since it contains many duplicate (up to variable renaming) clauses; Presto computed a rewriting with only 2670 clauses which is a surprisingly small number (also in comparison to the rewritings it computed in previous queries) but we did not investigate further. Regarding the rewriting size in other cases, all systems computed rewritings of roughly similar size, except for query 5 over OBO Protein (as stated above), and queries 1, 4, 5 over NCI for Requiem. As stated, small differences in the sizes of the rewritings can be attributed to the different form of the datalog program that each system produced. For example, in the case of query 1 over the OpenGALEN2 ontology Requiem produces the datalog program $\{Q(x) \leftarrow \mathit{Heme}(x),\ Q(x) \leftarrow \mathit{Haem}(x)\}$, while Rapid produces the program $\{Q(x) \leftarrow \mathit{Heme}(x),\ \mathit{Heme}(y) \leftarrow \mathit{Haem}(y),\ \mathit{Haem}(z) \leftarrow \mathit{Heme}(z)\}$. Clearly, the two rewritings produce the same certain answer when evaluated over an ABox.

Table 7: Evaluation results for large $\mathcal{ELHI}$ ontologies

	Time (hh:mm:ss.msec)			Rewriting size		
$\mathcal{O}$	Rapid	Requiem	Clipper	Rapid	Requiem	Clipper
OBO Protein	9.626	t/o	1:42.249	51641	t/o	51641
	29.764	t/o	1:45.742	52877	t/o	52877
	6.247	t/o	1:46.322	51614	t/o	51614
	16.940	t/o	1:44.356	52407	t/o	52407
	11:23.172	t/o	1:49.475	79427	t/o	105950
NASA SWEET	0.095	0.410	t/o	170	288	t/o
	0.015	0.840	t/o	507	1800	t/o
	0.013	0.753	t/o	660	1945	t/o
	0.028	2.551	t/o	1097	3380	t/o
	0.031	42.703	t/o	1107	19515	t/o
PERIODIC Table	0.147	3:31.069	20.512	1103	6800	2892
	0.035	4:11.178	19.857	879	6941	2892
	0.064	4:26.803	19.610	1653	6889	2892
	0.054	4:35.427	20.817	1609	8077	2849
	0.094	10:30.913	20.936	1743	57054	2893
NotGALEN	0.002	0.006	23:57.628	1	1	1
	1.527	t/o	24:35.363	11907	t/o	11756
	0.972	t/o	22:50.269	11913	t/o	11769
	0.001	t/o	22:28.542	11	t/o	11
	0.988	t/o	22:59.788	11913	t/o	11760
GALEN-Doctored	0.004	0.009	t/o	3	2	t/o
	1.772	t/o	t/o	9563	t/o	t/o
	0.772	t/o	t/o	9566	t/o	t/o
	0.001	t/o	t/o	11	t/o	t/o
	0.931	t/o	t/o	9576	t/o	t/o
OpenGALEN2	0.002	0.01	t/o	3	2	t/o
	2:49:21.804	t/o	t/o	160817	t/o	t/o
	t/o	t/o	t/o	t/o	t/o	t/o
	t/o	t/o	t/o	t/o	t/o	t/o
	t/o	t/o	t/o	t/o	t/o	t/o
ExtendedDNS	0.272	t/o	t/o	1996	t/o	t/o
	0.085	t/o	t/o	1999	t/o	t/o
	0.002	t/o	t/o	96	t/o	t/o
	0.009	t/o	t/o	281	t/o	t/o
	0.077	t/o	t/o	2042	t/o	t/o

6.2. $\mathcal{ELHI}$

The results for the $\mathcal{ELHI}$ ontologies are shown in Table 7. As it can be seen, Rapid is faster in the case of the medium sized ontologies, while in the case of the large and more complex ones it greatly outperforms all other systems by several orders of magnitude. Actually, in several occasions the competing systems did not terminate within the assigned time frame. However, again in query 5 over the OBO Protein ontology, Rapid performed worse than Clipper due to the final backwards subsumption checking and Clipper computed a rewriting that is of about the double size comparing to that produced by Rapid. A notable case is that, unlike Rapid, both Requiem and Clipper fail to handle the ExtendedDNS ontology, which is in general not a very large ontology.

The various versions of the GALEN ontology that we used allow us to make some additional useful remarks. NotGALEN and GALEN-Doctored are based on two early versions of the GALEN ontology that have a relatively simple structure. The results show that Rapid scales well and can compute rewritings within a few seconds. Clipper needs about 25 minutes for NotGALEN (which is the more simple between the two), and timeouts for GALEN-Doctored; Requiem timeouts for both. The situation is different for the OpenGALEN2 ontology, whose complex structure poses a challenge to all systems. Clipper fails on all queries, Requiem succeeds for query 1, but Rapid is able to compute rewritings for queries 1, 2 after requiring, however, a significant amount of time for query 2 (about 3 hours). For queries 3, 4, and 5 Rapid also did not manage to terminate.

After analysis we concluded that this failure is due to the extensive application of the function rule (after several minutes the rule has already been applied more than a million times), hence the number of side premises that need to be considered in n-shrinking is vast. In contrast, this number is much smaller in the other ontologies. In particular, OBO Protein and NotGALEN are actually $\mathcal{ELH}$ ontologies hence the rule is never applied while PERIODIC Table contains very few RA-clauses of the form $\exists R^-.C \sqsubseteq D$ and no axiom of the form $A \sqsubseteq \exists S^-.B$, hence again the function rule is never triggered (it would re-

Table 8: Evaluation results for the queries generated using the techniques in [24]

	Time (hh:mm:ss.msec)			Rewriting size		
$\mathcal{O}$	Rapid	Req.	Clip.	Rapid	Req.	Clip.
NASA	0.027	19.776	*t/o*	510.90	3818.74	*t/o*
ExtDNS	0.065	*t/o*	*t/o*	1613.91	*t/o*	*t/o*

quire an axiom $\exists R^{-}.C \sqsubseteq D$ paired with $A \sqsubseteq \exists R.B$ or an axiom $\exists R.C \sqsubseteq D$ paired with $A \sqsubseteq \exists R^{-}.B$). Moreover, in NASA SWEET, which contains only 12 inverse role axioms, the function rule is applied from 71 (query 2) to at most 536 times (queries 4 and 5), in ExtendedDNS, which contains 98 inverse role axioms, the rule is applied from 11 (query 3) to 510 times (query 5), while in GALEN-Doctored, which contains 207 inverse role axioms, the function rule, depending on the query, is either not applied at all (queries 1 and 4) or it is applied more than 20 millions times (queries 2, 3, 5).

Another interesting case is query 1 for the various versions of the GALEN ontology. This query retrieves all objects of the concept Heme which is found low in the GALEN hierarchy and is "isolated" with respect to the complex part of the ontology (also evident by its small rewriting). We note that Rapid and Requiem are able to recognize this fact and answer instantly, while Clipper apparently still processes the whole ontology before computing a rewriting.

Finally, Table 8 presents the results for the ontologies for which we could compute test queries using the query generator in [24]. Due to the large number of queries we present average times and rewriting sizes. Again, we can see that Rapid outperforms Requiem and Clipper.

Summarising the above, we can see that the new calculus can indeed compute a rewriting very efficiently for the vast majority of ontologies. This is also the case for large $\mathcal{ELHI}$ ontologies since the function rule and n-shrinking do not interact that much. However, for well-known problematic ontologies (e.g., OpenGALEN2) computing a rewriting still poses a significant challenge to all state-of-the-art systems.

7. Related Work

Query answering via rewriting has been extensively studied for various fragments of OWL during the last decade. The problem has been studied both from a theoretical point of view by producing many complexity results and characterising rewritability [8, 25, 18, 7], as well as from a practical point of view by designing and developing many algorithms and practical systems for computing rewritings [9, 39, 41, 36, 13, 40, 14, 45].

The works that are most closest to ours are those that propose a resolution-based rewriting algorithm. The first algorithm to be proposed was the one implemented in the KAON2 system [35]. It was based on basic superposition and ordered hyperresolution and it supported queries with distinguished variables over the languages $\mathcal{SHIQ}$ and Horn-$\mathcal{SHIQ}$. Later, Pérez-Urbina et al. [39] used binary resolution with free-selection to provide a rewriting algorithm that supports arbitrary queries over DL-Lite and $\mathcal{ELHI}$. Recently, resolution was also used to provide a rewriting algorithm for linear Datalog$^{\pm}$ [36]. The calculus was based on SLD resolution but with forward-chaining. Forward-chaining can create clauses with increasing nesting of function terms (terms with increasing depth). Hence, an additional condition was employed to ensure termination.

Even in the case of KAON2 that applies restrictions on the ordering of terms and the selection function, all the aforementioned calculi explicitly produce resolvents that contain terms with function symbols, although these clauses are always discarded from the final rewriting and, as it has been shown [42], in several cases they do not contribute to the generation of members of the rewriting. To the best of our knowledge, this is the first work on resolution-based rewriting for OWL ontologies that attempts to further optimise the resolution calculus by constructing such clauses only when they are guaranteed to contribute to the generation of members of the rewriting.

Finally, it is worth mentioning that the idea behind shrinking as well as the technique of postponing the generation of queries created via unfolding outlined in Section 5 is related to the approach of eliminating ej-variables followed in Presto [41]. However, the crucial difference is that Presto does not provide native support of qualified existential restrictions on the right-hand side of axioms but it relies on the same encoding as in [9]. This encoding increases the number of the input clauses and the overhead of dealing with them is evident by our performance evaluation. Moreover, Presto only supports DL-Lite.

8. Conclusions

In the current paper we have studied the problem of efficiently computing rewritings for ontologies expressed in various fragments of OWL using the framework of resolution.

First, we studied the problem for the language DL-Lite which is strongly related to the language OWL 2 QL. We designed a resolution-based rewriting algorithm which tries to (implicitly) generate clauses with function symbols only when these are "necessary"—that is, only when it is guaranteed that these will further contribute to the generation of function-free clauses. To achieve this we designed a new rule, namely shrinking, which accumulates many inferences into a macro-step and has the additional restriction that its conclusion must be function-free. Interestingly, due to the structure and properties of DL-Lite axioms this rule is amenable to many optimisations and can be implemented highly-efficiently.

Second, we studied the problem for the language $\mathcal{ELHI}$, which is strongly related to the language OWL 2 EL. $\mathcal{ELHI}$ is far more expressive than DL-Lite and the structure of its axioms makes the task very challenging; it is worth mentioning that deciding concept subsumption in $\mathcal{ELHI}$ is in ExpTime while deciding concept subsumption in DL-Lite is in P. However, by careful analysis of previous resolution-based rewriting approaches we can see that a calculus which follows similar principles like the ones designed for DL-Lite can be defined. More precisely, it suffices to extend the shrinking step so that it allows for n side premises and for type 1 and *RA*-clauses as

main premises, and to add one more inference rule that captures a very specific interaction between roles and their inverses. Interestingly, there is a close connection between the number of side premises to be considered in the n-shrinking rule and the application of the new rule. In realistic ontologies we expect that there are few inverse roles and therefore we expect that the function rule is rarely applied in practice and that the number of side premises to consider in the n-shrinking rule is small. Hence, the nice properties of the DL-Lite calculus are largely being preserved even for the much more expressive $\mathcal{ELHI}$ language.

Although optimised in its design, there are several issues to be considered when implementing our proposed resolution calculus. We have subsequently discussed some issues of the algorithm and have shown how some of its properties can be used to further optimise it. For example, we have shown how we can reduce the size of the computed rewriting by encoding much of the information using datalog clauses. Moreover, we have shown how we can construct queries produced eventually via shrinking before applying unfolding.

Finally, we have implemented all algorithms into the rewriting system Rapid and we have conducted an extensive experimental evaluation. Our test suite includes many well-known large and complex ontologies and hence significantly extends all previous benchmarks that mostly included toy ontologies [38, 24]. Moreover, our comparison against many state-of-the-art systems has provided many encouraging and important results. More precisely, when tested over the large and complex ontologies we see that, in most cases, existing systems fail to terminate within a timeout of 3 hours, while Rapid manages to compute a rewriting in just a few seconds. Despite this favourable performance there are still cases that pose a significant challenge for Rapid, like the highly-complex GALEN ontology. After analysis we concluded that the structure of GALEN forces many applications of the function rule (the additional rule for $\mathcal{ELHI}$) which consequently increases the number of side premises that need to be considered in n-shrinking. Hence, there is further room to improve the computations of rewritings.

Regarding directions for future work we plan to investigate whether the principles underlying the shrinking step can be used in resolution-based rewriting algorithms for even more expressive ontology languages. We plan to target both languages that can be rewritten into datalog, like linear Datalog$^{\pm}$ and Horn-$\mathcal{SHIQ}$ but also much more expressive languages like $\mathcal{ALC}$. The latter is an extremely challenging task as $\mathcal{ALC}$ can only be rewritten into disjunctive datalog [32].

Acknowledgements

The work by Giorgos Stoilos was funded by a Marie Curie Career Reintegration Grant within European Union's 7th Framework Programme (FP7/2007-2013) under REA grant agreement 303914.

Table A.9: Statistics of the test queries

	OBO Protein					NCI				
	Q_1	Q_2	Q_3	Q_4	Q_5	Q_1	Q_2	Q_3	Q_4	Q_5
$\|Q_i\|$	2	2	1	2	4	3	1	3	3	5
ejvar	1	1	0	1	2	1	0	1	1	2
	NASA SWEET					PERIODIC Table				
	Q_1	Q_2	Q_3	Q_4	Q_5	Q_1	Q_2	Q_3	Q_4	Q_5
$\|Q_i\|$	3	5	5	1	5	1	1	1	3	3
ejvar	2	1	2	0	1	0	0	0	1	1
	NotGALEN/GALEN-Doctored									
	Q_1		Q_2		Q_3	Q_4			Q_5	
$\|Q_i\|$	1		1		2	1			2	
ejvar	0		0		1	0			1	
	OpenGALEN2					ExtendedDNS				
	Q_1	Q_2	Q_3	Q_4	Q_5	Q_1	Q_2	Q_3	Q_4	Q_5
$\|Q_i\|$	1	2	2	4	2	1	1	2	1	2
ejvar	0	0	1	0	1	0	1	1	2	1

Appendix A. Statistics on Manually Constructed CQs

In Table A.9 we provide statistics for all manually constructed test queries we created for each of the test ontologies; with $|Q_i|$ we indicate the number of conjuncts in the query body and with ejvar the number ej-variables of the respective query. All queries contain one answer variable except for queries 2 and 5 over the NASA SWEET ontology which contain 2. Finally, note that all queries are tree-shaped.

To construct the queries we have tried to use concepts and roles that appear both low as well as high in the hierarchy of the ontology or appear in axioms with qualified existential restrictions. For example, queries 1, 2 for NotGALEN are the queries $Q(x) \leftarrow \mathsf{Heme}(x)$, $Q(x) \leftarrow \mathsf{BodyProcess}(x)$ where concepts Heme and BodyProcess have seven and three ancestors in the class hierarchy, respectively. Query 4 posed for ExtendedDNS is the query $Q(x) \leftarrow \mathsf{duses}(x, y) \wedge \mathsf{task}(y) \wedge \mathsf{duses}(x, z) \wedge \mathsf{description}(z)$, where concepts task, description and role duses participate in several axioms with qualified existentials. More precisely, the ontology contains the axioms $\mathsf{plan} \sqsubseteq \exists \mathsf{duses.task}$, $\mathsf{description} \sqsubseteq \exists \mathsf{dusedby}^{-}.\top$, and $\mathsf{dusedby} \sqsubseteq \mathsf{duses}^{-}$.

Appendix B. Requiem types of inferences

Table B.10 presents all interactions (types of inferences) that are possible when applying $\mathcal{I}_{\mathsf{REQ}}$ over clauses expressed in the language $\mathcal{ELHI}$.

In DL-Lite there are no RA-clauses, hence these are of the form $B(x) \leftarrow P(x, y)$ and $B(x) \leftarrow P(y, x)$. Consequently, inferences of the form 4.1+2.1=3.3 are simplified to:

$$\frac{B(x) \leftarrow P(x, y) \quad P(x, f(x)) \leftarrow A(x)}{B(x) \leftarrow A(x)}$$

where the resolvent is function-free. Moreover, according to Table 3 the input ontology can only contain function-free type 3.3 clause. Consequently, in DL-Lite all type 3.3 clauses are function-free; therefore, inference 3.3+2.3=3.3 in Table B.10 is never performed in DL-Lite.

Furthermore, since we consider normalised ontologies, in DL-Lite there can only be clauses of the form $B(f(x)) \leftarrow F(x)$. Hence, inferences of the form 3.3+2.3=2.3 are simplified to be of the following form:

$$\frac{C(x) \leftarrow A(x) \wedge [\mathbf{B}(x)] \quad A(f(x)) \leftarrow F(x)}{C(f(x)) \leftarrow F(x) \wedge [\mathbf{B}(f(x))]}$$

That is, there is exactly one function-free atom in the body of the resolvent. Moreover, the above resolvent can subsequently be used only as a main premise in an inference of the form 2.3+2.3=2.3, which in the case of DL-lite is simplified to be of the following form:

$$\frac{C(f(x)) \leftarrow F(x) \wedge B(f(x)) \wedge \mathbf{D}(f(x)) \quad B(f(x)) \leftarrow F(x)}{C(f(x)) \leftarrow F(x) \wedge \mathbf{D}(f(x))}$$

That is, the body part of the side premise is the same at the single function-free atom of the main premise. This is because each function symbol f is associated with a single body atom.

Appendix C. Proofs of Section 3

Lemma 1. *Let $\mathcal{T}$ be a DL-Lite-TBox and let Q be a CQ. Every type 1 clause derivable from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{REQ}}$ is also derivable from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{SLD}}$.*

Proof. We show using induction that for each clause Q' of type 1 such that $\mathcal{T}, Q \vdash_i^{\mathcal{I}_{\mathsf{REQ}}} Q'$ (i.e., that is derivable at depth i) we also have $\mathcal{T}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q'$.

Base case (i=0): In that case Q' is actually the input CQ Q and hence we clearly have $\mathcal{T}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q'$.

Induction step: For i some derivation depth assume that for every $\ell \leq i$ and Q' such that $\mathcal{T}, Q \vdash_\ell^{\mathcal{I}_{\mathsf{REQ}}} Q'$ we have $\mathcal{T}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q'$ (induction hypothesis). Assume now that at a next step a clause Q'' of type 1 is produced, i.e., $\mathcal{T}, Q \vdash_{i+1}^{\mathcal{I}_{\mathsf{REQ}}} Q''$. By the definition of $\mathcal{I}_{\mathsf{REQ}}$ Q'' is produced by an inference of the form $Q', C \vdash^{\mathcal{I}_{\mathsf{REQ}}} Q''$, i.e., one that has as a main premise another clause of type 1 such that $\mathcal{T}, Q \vdash_\ell^{\mathcal{I}_{\mathsf{REQ}}} Q'$ and as a side premise a clause C such that $\mathcal{T} \vdash^{\mathcal{I}_{\mathsf{REQ}}} C$. Clearly, we have $Q', C \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$. If $C \in \mathcal{T}$ then, by the previous and the induction hypothesis we immediately obtain $\mathcal{T}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$. Otherwise, we have $\mathcal{T} \vdash_j^{\mathcal{I}_{\mathsf{REQ}}} C$ with $j > 0$, and it suffices to show that Q'' can also be derived from Q_i by $\mathcal{I}_{\mathsf{SLD}}$ using only clauses of $\mathcal{T}$. This follows by the following claim:

> *Claim 1*: For each Q'' and C such that $Q', C \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$, $\mathcal{T} \vdash_j^{\mathcal{I}_{\mathsf{REQ}}} C$, and $j > 0$ there exist C_1 and C_2 such that the following hold:
>
> - $\mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} C_1$ and $\mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} C_2$, and
> - $Q', C_1 \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q^*, Q^*, C_2 \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$.

We show Claim 1 by a case analysis on the types of clauses that can be deduced from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{REQ}}$. According to $\mathcal{I}_{\mathsf{REQ}}$, only clauses of type 2.1, 2.2, 2.3 (of the form $B(f(x)) \leftarrow C(x) \wedge [\mathbf{A}(f(x))]$), or 3.3 (of the form $A(x) \leftarrow B(x)$) can be derived from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{REQ}}$. We show each case next:

Table B.10: Possible Types of Inferences of $\mathcal{I}_{\mathsf{REQ}}$

3.3+2.3=2.3

$$\frac{C(x) \leftarrow A(x) \wedge \mathbf{B}(x) \quad A(f(x)) \leftarrow \mathbf{F}(x)}{C(f(x)) \leftarrow \mathbf{B}(f(x)) \wedge \mathbf{F}(x)}$$

3.3+2.3=3.3

$$\frac{E(x) \leftarrow B(f(x)) \wedge \mathbf{C}(f(x)) \wedge \mathbf{D}(x) \quad B(f(x)) \leftarrow \mathbf{G}(x)}{E(x) \leftarrow \mathbf{C}(f(x)) \wedge \mathbf{G}(x) \wedge \mathbf{D(x)}}$$

2.3+2.3=2.3

$$\frac{E(f(x)) \leftarrow B(f(x)) \wedge \mathbf{C}(f(x)) \wedge \mathbf{D}(x) \quad B(f(x)) \leftarrow \mathbf{G}(x)}{E(f(x)) \leftarrow \mathbf{C}(f(x)) \wedge \mathbf{G}(x) \wedge \mathbf{D(x)}}$$

4.1+2.1=3.3

$$\frac{B(x) \leftarrow P(x,y) \wedge [\mathbf{C}(y)] \quad P(x, f(x)) \leftarrow A(x)}{B(x) \leftarrow A(x) \wedge [\mathbf{C}(f(x))]}$$

4.1+2.2=2.3

$$\frac{B(x) \leftarrow P(x,y) \wedge [\mathbf{C}(y)] \quad P(f(x), x) \leftarrow A(x)}{B(f(x)) \leftarrow A(x) \wedge [\mathbf{C}(x)]}$$

3.1+2.1=2.1

$$\frac{S(x,y) \leftarrow P(x,y) \quad P(x, f(x)) \leftarrow A(x)}{S(x, f(x)) \leftarrow A(x)}$$

3.1+2.2=2.2

$$\frac{S(x,y) \leftarrow P(x,y) \quad P(f(x), x) \leftarrow A(x)}{S(f(x), x) \leftarrow A(x)}$$

1+2.3=1

$$\frac{Q(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge R(t_1, t_2) \wedge B(t) \quad B(f(x)) \leftarrow \mathbf{C}(x)}{Q(\vec{s})\sigma \leftarrow \bigwedge D_j(\vec{t_j})\sigma \wedge R(t_1, t_2)\sigma \wedge \mathbf{C}(x)\sigma}$$

where σ is an mgu for $\{B(t), B(f(x))\}$

1+2.1=1

$$\frac{Q(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge R(t_1, t_2) \wedge B(t) \quad R(x, f(x)) \leftarrow C(x)}{Q(\vec{s})\sigma \leftarrow \bigwedge D_j(\vec{t_j})\sigma \wedge C(x)\sigma \wedge B(t)\sigma}$$

where σ is an mgu for $\{R(t_1, t_2), R(x, f(x))\}$

1. C is of type 2.1. Then, $Q', C \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ is of the form:

$$\frac{Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge R(v,u) \quad R(x, f(x)) \leftarrow A(x)}{Q_P(\vec{s})\sigma \leftarrow \bigwedge D_j(\vec{t_j})\sigma \wedge A(x)\sigma}$$

where σ is an mgu for $\{R(v,u), R(x, f(x))\}$. Moreover, $\mathcal{I}_{\mathsf{REQ}}$ produces clauses of type 2.1 by resolving clauses C_1, C_2 of either type 3.1 and 2.1 or of type 3.2 and 2.2. We show the case that C_1 is of type 3.1 ($R(x,y) \leftarrow P(x,y)$) and C_2 of type 2.1 ($P(x, f(x)) \leftarrow A(x)$); the case of types 3.2 and 2.2 is similar. Assume w.l.o.g. that the mgu in the inference of C_1 and C_2 is $\sigma_1 = \{y/f(x)\}$. Clearly, Q' resolves with C_1 producing clause:

$$Q^* = Q_P(\vec{s})\theta \leftarrow \bigwedge D_j(\vec{t_j})\theta \wedge P(x,y)\theta$$

with θ the mgu for $\{R(v,u), R(x,y)\}$. The mapping θ_1 =

$\{x \mapsto v, y \mapsto u\}$ is an mgu for $\{R(v,u), R(x,y)\}$. However, if v, u are variables, then $\theta_2 = \{v \mapsto x, u \mapsto y\}$ is also a possible mgu. But, the result of applying θ_1 or θ_2 to a clause is equivalent up to renaming of variables. Hence, w.l.o.g. we can assume that $\theta = \theta_1$. Moreover, since x, y do not appear in Q', clause Q^* is actually of the form $Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge P(v,u)$. Consequently, since the variables that appear in $P(v,u)$ are the same as those in $R(v,u)$, Q^* resolves with C_2 with mgu σ to obtain clause Q''. Therefore, there are inferences of the form $Q', C_1 \vdash Q^*$ and $Q^*, C_2 \vdash Q''$. Since $\mathcal{I}_{\mathsf{REQ}}$ never produces clauses of type 3.1, $C_1 \in \mathcal{T}$ and for C_2 we clearly have $\mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} C_2$.

2. C is of type 2.2. This case is symmetric to the previous one.
3. C is of type 2.3. Then, $Q', C \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ is of the following form:

$$\frac{Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge B(v) \quad B(f(x)) \leftarrow A(x) \wedge [\mathbf{A}'(f(x))]}{Q_P(\vec{s})\sigma \leftarrow \bigwedge D_j(\vec{t_j})\sigma \wedge A(x)\sigma \wedge [\mathbf{A}'(f(x))\sigma]}$$

where σ is an mgu for $\{B(v), B(f(x))\}$. Moreover, $\mathcal{I}_{\mathsf{REQ}}$ produces clauses of type 2.3 by resolving clauses C_1, C_2 of either type 3.3 and 2.3, or of type 4.1 and 2.2, or of type 4.2 and 2.1.

First, let C_1 be of type 3.3 ($B(x') \leftarrow C(x') \wedge [\mathbf{A}'(x')]$) and C_2 of type 2.3 ($C(f(x)) \leftarrow A(x)$), while C is produced by resolving them with mgu $\sigma_1 = \{x'/f(x)\}$. Clearly, Q' resolves with C_1 producing clause:

$$Q^* = Q_P(\vec{s})\theta \leftarrow \bigwedge D_j(\vec{t_j})\theta \wedge C(x')\theta \wedge [\mathbf{A}'(x')\theta]$$

with θ mgu for $\{B(v), B(x')\}$. As before, we can assume that θ is the mgu $\{x' \mapsto v\}$. Moreover, since x' does not appear in Q^*, clause Q^* is actually of the form $Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge C(v) \wedge [\mathbf{A}'(v)]$. Hence, since the variables that appear in $C(v), A'(v)$ are the same as those in $B(v)$, Q^* resolves with C_2 with mgu σ to obtain clause Q''. Therefore, there are derivations of the form $Q', C_1 \vdash Q^*$ and $Q^*, C_2 \vdash Q''$. Moreover, we clearly have $\mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} C_1$ and $\mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} C_2$.

Second, let C_1 be of type 4.1 ($B(x') \leftarrow R(x', y)$) and C_2 is of type 2.2 ($R(f(x), x) \leftarrow A(x)$), while is produced using mgu $\sigma_1 = \{x'/f(x), y \mapsto x\}$; the case of 4.2 and 2.1 is similar. Again, Q' resolves with C_1 producing clause:

$$Q^* = Q_P(\vec{s})\theta \leftarrow \bigwedge D_j(\vec{t_j})\theta \wedge R(x', y)\theta$$

with θ mgu for $\{B(v), B(x')\}$ and θ can be the mgu $\{x' \mapsto v\}$; hence, we have $Q^* = Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge R(v, y)$. Since v also appears in $B(v)$ of Q' and y is new in Q^*, Q^* resolves with C_2 with mgu $\sigma' = \sigma \cup \{y \mapsto x\}$ which produces Q''. Thus, again, there is a derivation of the form $Q', C_1 \vdash Q^*$, $Q^*, C_2 \vdash Q''$, while since $\mathcal{I}_{\mathsf{REQ}}$ never produces clauses of type 4.1, $C_1 \in \mathcal{T}$, and finally $\mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} C_2$, as required.

4. C is of type 3.3. Then, $Q', C \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ is of the following form:

$$\frac{Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge A(v) \quad A(x) \leftarrow B(x)}{Q_P(\vec{s})\sigma \leftarrow \bigwedge D_j(\vec{t_j})\sigma \wedge B(x)\sigma}$$

where σ is an mgu for $\{A(v), A(x)\}$. Moreover, $\mathcal{I}_{\mathsf{REQ}}$ produces clauses of type 3.3 by resolving clauses C_1, C_2 of either type 4.1 and 2.1 or of type 3.2 and 2.2. Assume that C_1 is of type 4.1 ($A(x) \leftarrow R(x, y)$) and C_2 of type 2.1 ($R(x, f(x)) \leftarrow B(x)$) while C is produced using the mgu $\sigma_1 = \{y/f(x)\}$ (the case of 4.2 and 2.2 is similar). Clearly, Q' resolves with C_1 producing clause:

$$Q^* = Q_P(\vec{s})\theta \leftarrow \bigwedge D_j(\vec{t_j})\theta \wedge R(x, y)\theta$$

Again we can assume that θ is the mgu $\{x \mapsto v\}$ and Q^* is actually of the form $Q^* = Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge R(v, y)$. Hence, since v appears both in $R(v, y)$ and $A(v)$ and y does not appear in Q', clause Q^* resolves with C_2 with mgu σ to obtain clause Q''.

The previous claim implies that for a clause C such that $\mathcal{T} \vdash^{\mathcal{I}_{\mathsf{REQ}}} C$ the inference $Q', C \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ can be transformed into a sequence of inferences of the form $Q', C_1 \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q_1^*$, $Q_1^*, C_2 \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q_2^*, \ldots, Q_{n-1}^*, C_n \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$, such that for all $1 \leq i \leq n$ we have $C_i \in \mathcal{T}$; hence $\mathcal{T}, Q' \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ and combined with the induction hypothesis ($\mathcal{T}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q'$) we get $\mathcal{T}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ as required. □

Lemma 2. *Any derivation built by inferences that have as side premises Horn clauses that can contain terms with function symbols only in the head can be transformed into one that is function-compact.*

Proof. Consider a derivation $Q_1, Q_2, \ldots, Q_n$ by $\mathcal{I}_{\mathsf{SLD}}$ such that all clauses $Q_2, \ldots, Q_{n-1}$ contain a term that mentions a function symbol f while Q_1 and Q_n don't mention f. In the following to also quantify over the side premises used in a derivation we write such a derivation as $\langle Q_1, C_1\rangle, \langle Q_2, C_2\rangle, \ldots, \langle Q_n, C_n\rangle$, where $Q_i, C_i \vdash Q_{i+1}$.

The lemma follows by the following claim:

> *Claim 2:* For every derivation like the above and for which there exists Q_j and C_j such that C_j does not mention f, then there also exists C_k that does not mention f and is such that either of the following are valid derivations by $\mathcal{I}_{\mathsf{SLD}}$:
>
> (a) $\langle Q_1, C_k\rangle, \langle Q'_2, C_1\rangle, \ldots, \langle Q'_k, C_{k-1}\rangle, \langle Q_{k+1}, C_{k+1}\rangle, \ldots, \langle Q_n, C_n\rangle$ or
>
> (b) $\langle Q_1, C_1\rangle, \langle Q_2, C_2\rangle, \ldots, \langle Q_k, C_{k+1}\rangle, \langle Q'_{k+1}, C_{k+2}\rangle, \ldots, \langle Q'_{n-1}, C_k\rangle, \langle Q_n, C_n\rangle$

The claim says that some inference with side premise C_k that does not mention f can be moved "outside" of the relevant part of the derivation. In Case 1., since Q_1 and C_k do not mention f, so does Q'_2. Hence, then the first clause that mentions f is Q'_3. In Case 2., again since Q_n, C_k do not mention f, so does Q_{n-1} (a function-free clause cannot be created from an inference with main a clause containing a functional symbol and a function-free side premise). Consequently, by repeated applications of the above transformation we can remove all the inferences that have a side premise that does not mention f and create new function-free "first" and "last" clauses that are closer together.

The transformation clearly terminates and it can be done to all parts of an SLD derivation. Hence after a finite number of steps we can obtain a function-compact derivation.

We now show the *Claim 2*.

First, we show that if a pair of inferences $Q_1, C_1 \vdash Q_2$ and $Q_2, C_2 \vdash Q_3$ can be rewritten as $Q_1, C_2 \vdash Q'$ and $Q', C_1 \vdash Q''$, then Q'' is actually Q_3—that is, if we can reorder the pair, then after the reordering we obtain the same final derived clause. Thus, in Claim 2, after C_k has been pushed to the beginning (end), we indeed have that $Q'_k, C_{k-1} \vdash Q_{k+1}$ ($Q'_{n-1}, C_k \vdash Q_n$).

In more detail, the inference $Q_1, C_1 \vdash Q_2$ is of the form:

$$\frac{Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge A \wedge B \quad A_1 \leftarrow A_2}{Q_P(\vec{s})\sigma \leftarrow \bigwedge D_j(\vec{t_j})\sigma \wedge A_2\sigma \wedge B\sigma}$$

where σ is the m.g.u. for $\{A, A_1\}$, while $Q_2, C_2 \vdash Q_3$ is of the form:

$$\frac{Q_P(\vec{s})\sigma \leftarrow \bigwedge D_j(\vec{t_j})\sigma \wedge A_2\sigma \wedge B\sigma \quad B_1 \leftarrow B_2}{Q_P(\vec{s})\sigma\theta \leftarrow \bigwedge D_j(\vec{t_j})\sigma\theta \wedge A_2\sigma\theta \wedge B_2\theta}$$

where θ is the m.g.u. for $\{B\sigma, B_1\}$. After reordering, the inference $Q_1, C_2 \vdash Q'$ is of the form:

$$\frac{Q_P(\vec{s}) \leftarrow \bigwedge D_j(\vec{t_j}) \wedge A \wedge B \quad B_1 \leftarrow B_2}{Q_P(\vec{s})\theta' \leftarrow \bigwedge D_j(\vec{t_j})\theta' \wedge A\theta' \wedge B_2\theta'}$$

for θ' a most general unifier for $\{B, B_1\}$, while $Q', C_2 \vdash Q''$ is of the form:

$$\frac{Q_P(\vec{s})\theta' \leftarrow \bigwedge D_j(\vec{t_j})\theta' \wedge A\theta' \wedge B_2\theta' \quad A_1 \leftarrow A_2}{Q_P(\vec{s})\theta'\sigma' \leftarrow \bigwedge D_j(\vec{t_j})\theta'\sigma' \wedge A_2\sigma' \wedge B_2\theta'\sigma'}$$

with σ' an m.g.u. for $\{A_1, A\theta'\}$. Clause Q'' contains the same predicates as Q_3. To show that Q'' is equal (up to renaming of variables to Q_3) we need to show that unifiers θ' and σ' exist such that $\theta' \circ \sigma'$ maps variables of Q in the same way as $\sigma \circ \theta$. To do so, given a mapping $s_1 \mapsto s_2 \in \sigma$ and a mapping $t_1 \mapsto t_2 \in \theta$, we show how we can build mappings $s'_1 \mapsto s'_2 \in \sigma'$ and $t'_1 \mapsto t'_2 \in \theta'$ such that, w.r.t. the variables that are being mapped, applying σ and then θ has the same result as applying first θ' and then σ'. Consequently, since this can be done for any combination of mappings in σ and θ, applying θ' and then σ' would have the same effect on all the variables of the query. Since there are unifiers there are also most general unifiers and we will have that $Q_3 = Q''$. Unifiers θ' and σ' are constructed based on σ and θ as follows, where 1. and 2. are applied exhaustively first, and then 3., 4., 5., and 6. in that order:

1. If σ contains $x \mapsto s$ and θ contains $w \mapsto s$, then add $w \mapsto x$ to θ' and $x \mapsto s$ to σ'. Clearly, $\theta' \circ \sigma'$ maps x and w to the same terms as $\sigma \circ \theta$.
2. If σ contains $x \mapsto s$ and θ contains $w \mapsto g(s)$ for some function g, then add $w \mapsto g(x)$ to θ' and $x \mapsto s$ to σ'. Again, $\sigma \circ \theta$ maps x and w to the same terms as $\sigma \circ \theta$.
3. In all other cases copy all mappings of θ to θ'.
4. Let σ contain $x \mapsto y$ and let x be a variable of Q_1. If θ' contains no mapping of the form $y \mapsto s$, then add $x \mapsto y$ to σ'; otherwise, add $x \mapsto s$ to σ'; similarly if $x \mapsto f(y) \in \sigma$ and $y \mapsto s \in \theta$, then add $x \mapsto f(s)$ to σ'. Again, $\theta' \circ \sigma'$ maps variables x and y to the same terms as $\sigma \circ \theta$.
5. In all other cases copy all mappings of σ to σ'.
6. If σ contains a mapping of the form $x \mapsto s$ and x is a variable of C_1, then add $x \mapsto s\theta'$ to σ'. In this case this mapping has no effects on the variables of Q'' as x belongs in C_1.

Note that the construction is well-founded: Cases 1. and 2. are independent from each other and do not depend on previously introduced mappings. Moreover, Case 3. is also independent. Case 4. has an if condition over θ'. However, θ' is never altered again, hence the if condition is either always satisfied or not and the construction of σ' is well-defined. Finally, Cases 5. and 6. are also well-defined.

Before proceeding we note that a pair of inferences $Q_1, C_1 \vdash Q_2$ and $Q_2, C_2 \vdash Q_3$ cannot be rewritten as $Q_1, C_2 \vdash Q'$ and $Q', C_1 \vdash Q''$ only if the head of C_2 resolves with some atom that was introduced when resolving Q_1 with C_1 —that is, C_1 is of the form $A \leftarrow \mathbf{B}$, C_2 is of the form $B \leftarrow \mathbf{C}$ and $B \in \mathbf{B}$ and hence the inference with side premise C_2 requires that the inference with side premise C_1 occurs first. If reordering of the previous inference is possible then we say that C_2 is *independent* of C_1, while if they cannot then we say that it is *dependent*.

Finally, we show that clauses that do not mention a functional symbol can indeed be pushed either at the beginning or at the end of the sequence.

Consider a sequence $Q_1, \ldots, Q_n$ where all $Q_2, \ldots, Q_{n-1}$ mention a function symbol f while Q_1 and Q_n do not. Assume also in contrast that none of the clauses C_i with $2 \leq i \leq n-1$ that does not mention f can be moved either at the beginning or at the end.

Let C_k be the first side premise that does not mention f. By assumption it cannot be moved at the beginning of the sequence. Hence, C_k is of the form $A \leftarrow \mathbf{B}$ and there is some previous inference $Q_j, C_j \vdash Q_{j+1}$ with $j > 1$, where C_j is of the form $F \leftarrow \mathbf{A}$ with $A \in \mathbf{A}$ and hence C_k cannot be moved outside. If C_j is function-free then the assumption also applies to it and hence there is some other clause before over which C_j depends. It follows that for some clause that cannot be moved outside the clause obstructing the move contains f in the head. Let that be C_j, that is, for C_j we additionally have that F contains the function symbol f. It also follows that no clause C_ℓ in between, i.e., where $j < \ell < k$ can depend on C_j for otherwise the inference with C_ℓ would eliminate A and hence C_k would not depend on C_j. Consequently, all these clauses in between can be moved before C_j, that is, the sequence can be reordered such that $j = k - 1$.

Next, again by assumption, C_k cannot be moved downwards either. Hence, this implies that there is also some clause C_m that depends on C_k—that is, it is of the form $B \leftarrow \mathbf{C}$, where $B \in \mathbf{B}$. Again no clause in between C_k and C_m can depend on C_k. Moreover it can also not depend on C_j as A is eliminated by the inference with side premise C_k. Hence, all these clauses can also be moved before clause C_j, i.e., C_{k-1} in the reordered sequence. By applying this reasoning repeatedly we can reorder the sequence such that after clause C_{k-1} we have inferences with

side premises clauses C_{k-1}, C_k, C_{k+1} such that the head of each clause resolves with some body atom of its clause immediately on the left. Let C^{k+1} be the clause produced by the inferences $C_{k-1}, C_k \vdash C'$, $C', C_{k+1} \vdash C^{k+1}$. By the form of the clauses used in these inferences (see before) C^{k+1} must be of the form $F \leftarrow \mathbf{C}$. Moreover, we can easily see that instead of obtaining Q_{k+2} through the inferences $Q_{k-1}, C_{k-1} \vdash Q_k, Q_k, C_k \vdash Q_{k+1}$, and $Q_{k+1}, C_{k+1} \vdash Q_{k+2}$ we can obtain it as a resolvent of Q_{k-1} and C^{k+1}. Furthermore, since Q_{k-1} mentions f, the inference $Q_{k-1}, C_{k-1} \vdash Q_k$ creates a clause that also mentions f, and C^{k+1} has the same head as C_{k-1}, then so would Q_{k+2} mention f.

Consequently, if C_{k+1} is the last clause in the sequence, i.e., C_{n-1}, then we reach a contradiction: in that case Q_{k+2} is Q_n and as shown before it must mention f. Otherwise, there are other inferences and side premises between C_{k+1} and C_{n-1}. First note that these can only depend on C_{k+1}. If some dependent clause exists then, reorder the sequence by moving it after C_{k+1}, and compute the clause C^{k+2} where $C^{k+1}, C_{k+2} \vdash C^{k+2}$; otherwise, starting from the topmost, push all independent clause one by one before C_{k-1}. We can repeat this reordering until we reach a sequence where all inferences have side premises of the form $C_{k-1}, C_k, C_{k+1}, C_{k+2}, \ldots, C_{n-1}$, all clauses depend on the clause immediately on the left and as stated before Q_n can be obtained as a resolvent of Q_{k-1} and some clause C^{n-1} which is built by resolving one-by-one in an iterative way all the side premises of the previous list. Hence, we can reach a contradiction. □

Theorem 1. *Let $\mathcal{T}$ be a DL-Lite$_R$TBox and let Q be a CQ. Every derivation from $\mathcal{T} \cup Q$ by $\mathcal{I}_{lite}$ terminates. Moreover,* Rapid-Lite$(Q, \mathcal{T})$ *is a UCQ rewriting for $Q, \mathcal{T}$.*

Proof. Termination follows by our observations stated in Section 3. Next we show correctness. Let $\mathcal{R}_Q$ = Rapid-Lite$(Q, \mathcal{T})$. To show that $\mathcal{R}_Q$ is a UCQ rewriting we need to show that for each $\mathcal{A}$ we have cert$(Q, \mathcal{T} \cup \mathcal{A})$ = cert$(\mathcal{R}_Q, \mathcal{A})$.

In order to show that cert$(Q, \mathcal{T} \cup \mathcal{A}) \supseteq$ cert$(\mathcal{R}_Q, \mathcal{A})$, it suffices to show that $\mathcal{T} \cup Q \models \mathcal{R}_Q$, which is equivalent to showing $\mathcal{T} \cup Q \models Q_i$ for each $Q_i \in \mathcal{R}_Q$. This follows trivially since each Q_i is built using a resolution based calculus.

Now we show that cert$(Q, \mathcal{T} \cup \mathcal{A}) \subseteq$ cert$(\mathcal{R}_Q, \mathcal{A})$. Consider the Requiem algorithm. It suffices to show that for each $Q' \in$ RQR$(Q, \mathcal{T})$ a query $Q'' \in \mathcal{R}_Q$ exists such that Q' and Q'' are equivalent up to renaming of variables. RQR$(Q, \mathcal{T})$ is constructed in two steps. First, $\mathcal{T} \cup Q$ is saturated by $\mathcal{I}_{\mathsf{REQ}}$ to obtain a (non-recursive) datalog program P, and then P is unfolded to obtain a UCQ.

By Lemma 1, every clause of type 1 in P can also be derived from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{SLD}}$, while by Lemma 2 the SLD derivation can additionally be assumed to be function-compact. Hence, it suffices to show that for each function-free clause of type 1 derivable from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{SLD}}$is also derivable from $\mathcal{T} \cup Q$ by $\mathcal{I}_{lite}$.

Let Q_i be the i-th query derived from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{SLD}}$. We use induction.

Base case (i=0): In that case $Q_0 = Q$ and by definition of Rapid-Lite$(Q, \mathcal{T})$, we have $Q_0 \in R_Q$.

Induction step: Assume that for all $1 \leq i \leq n$, all function-free clauses Q_i derivable by $\mathcal{I}_{\mathsf{SLD}}$ are also derivable by $\mathcal{I}_{lite}$ and assume that a new function-free clause Q_{i+1} is derived next by $\mathcal{I}_{\mathsf{SLD}}$. There are two cases:

1. Q_{i+1} is derived from some function-free Q_j with $j \leq i$ and side premise some clause $C_j \in \mathcal{T}$. Since both Q_j and Q_{i+1} are function-free this inference can be captured by the unfolding inference rule; hence, $Q_j, C_j \vdash^{\mathcal{I}_{lite}} Q_{i+1}$ and thus $Q_{i+1} \in \mathcal{R}_Q$.
2. Q_{i+1} is derived from some Q_j by $\mathcal{I}_{\mathsf{SLD}}$ with an SLD derivation of the form $Q_j, Q_{j+1}, \ldots, Q_{j+k}$ where $Q_{j+k} = Q_{i+1}$, all intermediate clauses contain a term $f(s_j)$, while also by function-compactness all side premises contain $f(t_j)$. By definition of $\mathcal{I}_{\mathsf{SLD}}$ all side premises belong to $\mathcal{T}$ and $\mathcal{T}$ is normalised; hence, function symbols are unique per axiom of the form $\exists R.A$ and consequently, all side premises are either of the form $R(x, f(x)) \leftarrow \mathbf{D}$ or $A(f(x)) \leftarrow \mathbf{C}$, while the atoms of all intermediate clauses of type 1 over which they resolve are of the form $R(u_j, f(s_j))$, $R(f(s_j), u_j)$ or $A(f(s_j))$ for s_j, u_j terms. Moreover, since Q_j is function-free and Q_{j+1} mentions f, the atom of Q_j that resolves with C_j is of the form $R(s, t)$ or $A(t)$ and the mgu must contain a mapping of the form $x \mapsto f(y)$ for x an ej-variable of Q_j. Let S_1 contain all the role atoms of Q_j that participate in the inference and S_2 all the concept atoms. (Note that either can be empty) Since there is an mgu for all intermediate SLD inferences, there is also a simultaneous mgu for $S_1 \cup \{R(x, f(x))\}$ and $S_2 \cup \{A(f(x)\}$ [16]. Moreover, as analysed above and by construction of the simultaneous mgu, this must contain a mapping of the form $x \mapsto f(y')$ for x an ej-variable. Consequently, there is an inference of the form $Q_j, C_1, [C_2] \vdash Q_{i+1}$ using the shrinking rule.

Hence, in either case all function-free clauses of type 1 are derivable by $\mathcal{I}_{lite}$.

Finally, we show that all queries derived using unfolding are also derivable by $\mathcal{I}_{lite}$. This is straightforward: again all such inferences can be unfolded to be captured by inferences using as side premises only clauses from $\mathcal{T}$ and moreover all these always produce function-free type 1 clauses; hence, they can be captured by the unfolding rule of $\mathcal{I}_{lite}$ and hence all such queries belong to $\mathcal{R}_Q$. □

Appendix D. Proofs of Section 4

Lemma 3 *Let $\mathcal{T}$ be an $\mathcal{ELHI}$-TBox and let Υ be a CQ (resp. RA-clause). Then, every type 1 clause (resp. type* 3.3 *clause) derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{REQ}}$ starting with Υ is also derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$ starting with Υ.*

Proof. First, let Υ be an arbitrary *RA*-clause that produces some clause of type 3.3 from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{REQ}}$. We show the claim via induction. Assume that at some point all 3.3 clauses derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{REQ}}$ starting with an *RA*-clause are also derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$ (IH1). Assume that a new 3.3 clause C is derived in some part of the derivation that is starting with Υ. We

show only the case that Υ is of the form $A(x) \leftarrow R(x,y) \wedge \mathbf{D}(y)$, i.e., of type 4.1; for Υ of type 4.2 the proof is similar.

Clause C is of type 3.3. According to Table B.10 such clauses can be produced by inferences of the form $C_1, \mathcal{D}_1 \vdash^{\mathcal{I}_{\mathsf{REQ}}} C$ where C_1 and $\mathcal{D}_1$ can be one of the following forms:

1. Clause C_1 is of type 4.1 and clause $\mathcal{D}_1$ is of type 2.1; the case C_1 is of type 4.2 and $\mathcal{D}_1$ is of type 2.2 is similar. Since $\mathcal{I}_{\mathsf{REQ}}$ never produces clauses of type 4.1, $C_1 \in \mathcal{T}$. In contrast, $\mathcal{D}_1$ can be produced by an inference of the form $\mathcal{E}_2, \mathcal{D}_2 \vdash \mathcal{D}_1$, where $\mathcal{E}_2$ is of type 3.1 and $\mathcal{D}_2$ is again of type 2.1. Consequently, as shown in the proof of Lemma 1, $C_1, \mathcal{D}_1 \vdash^{\mathcal{I}_{\mathsf{REQ}}} C$ can be unfolded into the inferences $C_1, \mathcal{E}_2 \vdash C_2$ and $C_2, \mathcal{D}_2 \vdash C$. Clauses of type 3.1 are never produced by $\mathcal{I}_{\mathsf{REQ}}$, hence $\mathcal{E}_2 \in \mathcal{T}$, while $\mathcal{D}_2$ (again of type 2.1) is of lower derivation depth. By a straightforward inductive claim we can show that we can exhaustively unfold these inferences until we have a derivation of the form $C_1, \ldots, C_n, C$ where $C_i, \mathcal{E}_i \vdash C_{i+1}$, all C_i are of type 4.1, $C_1 \in \mathcal{T}$, all $\mathcal{E}_i$ are of type 3.1 (or 3.2) also in $\mathcal{T}$ and C is derived by some $C_n, \mathcal{D}_n \vdash C$, where $\mathcal{D}_n$ is of type 2.1 (or 2.2) and again in $\mathcal{T}$. Consequently, C is derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$.
2. Clause C_1 is of type 3.3 and clause $\mathcal{D}_1$ is some clause. Since C is derived by a sequence starting with Υ, so does C_1. Hence, by induction hypothesis IH1 C_1 is derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$. Now, we turn our attention to $\mathcal{D}_1$. By a second induction we can show that if $\mathcal{D}_1$ is derivable by inferences over other clauses, then $C_1, \mathcal{D}_1 \vdash^{\mathcal{I}_{\mathsf{REQ}}} C$ can either be unfolded using these clauses which are of lower derivation depth or $\mathcal{D}_1$ is derivable. More precisely, we have the following claim:

   > *Claim 3*: For each C_1, C, and $\mathcal{D}_1$ such that $C_1, \mathcal{D}_1 \vdash C$ and $\mathcal{T} \vdash_j^{\mathcal{I}_{\mathsf{REQ}}} \mathcal{D}_1$ we have that if $\mathcal{D}_1$ is of type 3.3., then it is function-free and moreover either $\mathcal{D}_1$ is produced by $\mathcal{I}_{\mathsf{SLD}^+}$ or there exist clauses $\mathcal{F}$ and $\mathcal{G}$ such that $\mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} \mathcal{F}, \mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} \mathcal{G}$, and $C_1, \mathcal{F} \vdash C', C', \mathcal{G} \vdash C$

   proof of Claim 3: We study different cases according to the form of $\mathcal{D}_1$:
   (a) $\mathcal{D}_1$ is of type 2.3. Such clauses can be produced by inferences of the form $\mathcal{F}, \mathcal{G} \vdash^{\mathcal{I}_{\mathsf{REQ}}} \mathcal{D}_1$, where $\mathcal{F}$ and $\mathcal{G}$ can be one of the following forms:
      i. $\mathcal{F}$ is of type 4.1 (4.2) and $\mathcal{G}$ is of type 2.2 (2.1). We show only the case 4.1+2.2.
      First, assume that $\mathcal{F}$ contains the conjunct $[C(y)]$, i.e., it is of the form $A(x) \leftarrow R(x,y) \wedge \mathbf{C}(y)$. Then, inference $\mathcal{F}, \mathcal{G} \vdash^{\mathcal{I}_{\mathsf{REQ}}} \mathcal{D}_1$ corresponds to an inference by the function rule and moreover, recall that since $\mathcal{F}$ is of type 4.1 we have $\mathcal{F} \in \mathcal{T}$. Consequently, if additionally $\mathcal{G} \in \mathcal{T}$, then $\mathcal{D}_1$ can be derived by $\mathcal{I}_{\mathsf{SLD}^+}$ as required. In contrast, $\mathcal{G}$ can be produced by an inference between a clause $\mathcal{J}_1$ of type 3.1 and a clause $\mathcal{G}_1$ of type 2.2. Then, the inference $\mathcal{F}, \mathcal{G} \vdash \mathcal{D}_1$ can be unfolded into $\mathcal{F}, \mathcal{J}_1 \vdash \mathcal{F}_1$ and $\mathcal{F}_1, \mathcal{G}_1 \vdash \mathcal{D}_1$. Furthermore, we have $\mathcal{J}_1 \in \mathcal{T}$ since such clauses are never produced by $\mathcal{I}_{\mathsf{REQ}}$ and hence $\mathcal{F}_1$ can be derived by $\mathcal{I}_{\mathsf{SLD}}$ from $\mathcal{F}$. This can be done exhaustively until we reach a derivation of the form $\mathcal{F}_0, \mathcal{F}_1, \ldots, \mathcal{F}_n, \mathcal{D}_1$ where for $0 \leq i < n$ we have $\mathcal{F}_i, \mathcal{J}_{i+1} \vdash \mathcal{F}_{i+1}$, $\mathcal{F}_0 = \mathcal{F}$, all $\mathcal{J}_{i+1}$ are of type 3.1, and $\mathcal{F}_n, \mathcal{G}_n \vdash C_i$ for some $\mathcal{G}_n \in \mathcal{T}$. Since all $\mathcal{J}_i$ are of type 3.1 and $\mathcal{F}_0 \in \mathcal{T}$, then $\mathcal{F}_n$ is derivable by $\mathcal{I}_{\mathsf{SLD}}$. Moreover, the last inference in the sequence corresponds the function rule and since $\mathcal{F}_n$ is derivable by $\mathcal{I}_{\mathsf{SLD}}$ and $\mathcal{G}_n \in \mathcal{T}$, then we can conclude that $\mathcal{D}_1$ can be produced from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$.
      Second, assume that $\mathcal{F}$ does not contain the conjunct $[\mathbf{C}(y)]$, i.e., it is of the form $A(x) \leftarrow R(x,y)$. Then, the inference $\mathcal{F}, \mathcal{G} \vdash \mathcal{D}_1$ can be used to unfold the inference $C_1, \mathcal{D}_1 \vdash C$ into $C_1, \mathcal{F} \vdash C'$ and $C', \mathcal{G} \vdash C$, where $\mathcal{G}$ and $\mathcal{F}$ are derivable at lower depths.
      ii. $\mathcal{F}$ is of type 3.3, $\mathcal{G}$ is of type 2.3 and according to Table B.10 $\mathcal{F}$ is function-free. In this case the inference $C_1, \mathcal{D}_1 \vdash C$ can be unfolded into $C_1, \mathcal{F} \vdash C'$ and $C', \mathcal{G} \vdash \mathcal{D}_1$ and $\mathcal{F}$ and $\mathcal{G}$ obviously have smaller derivation depths and according to Table B.10 clause $\mathcal{F}$ is function-free; hence, all conditions in *Claim 3* are satisfied.
      iii. both $\mathcal{F}$ and $\mathcal{G}$ are of type 2.3. Then, inference $C_1, \mathcal{D}_1 \vdash C$ can be unfolded into inferences of the form $C_1, \mathcal{F} \vdash C'$, $C', \mathcal{G} \vdash C$ where $\mathcal{F}$ and $\mathcal{G}$ are derivable at lower depths.
   (b) $\mathcal{D}_1$ is of type 3.3. By induction hypothesis of *Claim 3* it follows that $\mathcal{D}_1$ is function-free, thus it is of the form $A(x) \leftarrow B(x) \wedge [\mathbf{C}(x)]$. We now have the following two cases:
      First, assume that $\mathcal{D}_1$ is actually of the form $A(x) \leftarrow B(x)$ (i.e., no conjunct $C(x)$). Then, for $\mathcal{F}$ and $\mathcal{G}$ we have the following cases:
      i. $\mathcal{F}$ is of the form $A(x) \leftarrow R(x,y)$ (type 4.1 without $[\mathbf{C}(y)]$) and $\mathcal{G}$ is of the form $R(x, f(x)) \leftarrow B(x)$ (type 2.1) or $\mathcal{F}$ is of the form $A(x) \leftarrow R(y,x)$ (type 4.2 again without $[\mathbf{C}(y)]$) and $\mathcal{G}$ is of the form $R(f(x), x) \leftarrow B(x)$ (type 2.2). In this case we can unfold the inference $C_1, \mathcal{D}_1 \vdash C$ into $C_1, \mathcal{F} \vdash C'$, $C', \mathcal{G} \vdash C$ where $\mathcal{F}$ and $\mathcal{G}$ are derivable at lower depths.
      ii. $\mathcal{F}$ is of the form $A(x) \leftarrow D(f(x)) \wedge B(x)$ (i.e., type 3.3) and $\mathcal{G}$ is of the form $D(f(x)) \leftarrow B(x)$ (i.e., type 2.3). Since the input TBox $\mathcal{T}$ never contains clauses of the form $A(x) \leftarrow D(f(x)) \wedge B(x)$, $\mathcal{F}$ must have been produced by applying $\mathcal{I}_{\mathsf{REQ}}$ on $\mathcal{T}$. $\mathcal{F}$ can be produced again by an inference of the above form, i.e. 3.3+2.3, where the main premise must contain a functional-term in its body. Consequently, again this type 3.3 clause cannot belong to $\mathcal{T}$. We can conclude that in this case the derivation producing $\mathcal{D}_1$ must start with an *RA*-clause. But

then, by the induction hypothesis IH1, we have that $\mathcal{D}_1$ has been produced from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$.

Second assume that $\mathcal{D}_1$ is of the form $A(x) \leftarrow B(x) \wedge \mathbf{C}(x)$ (i.e., with a conjunct $\mathbf{C}(x)$). Again like in the previous case, we can conclude from the form of inferences that $\mathcal{D}_1$ is produced by a derivation starting from an *RA*-clause; hence, by induction hypothesis IH1 $\mathcal{D}_1$ is produced from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$.

This concludes the case where Υ is an *RA*-clause.

Second, we show the case that Υ is a type 1 clause Q producing another type 1 clause Q'—that is, for Q, Q' we have $\mathcal{T}, Q \vdash_i^{\mathcal{I}_{\mathsf{REQ}}} Q'$.

Let $\mathcal{P}_{lt}$ be all DL-Lite-clauses derivable from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{SLD}^+}$. We will show using induction that $\mathcal{P}_{lt}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q'$.

Base case (i=0): Then, $Q' = Q$ and we trivially have $\mathcal{P}_{lt}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q$.

Induction step: Assume that for every $\ell \leq i$ and Q' such that $\mathcal{T}, Q \vdash_\ell^{\mathcal{I}_{\mathsf{REQ}}} Q'$ we have $\mathcal{P}_{lt}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q'$ (induction hypothesis). Assume now that at a next step a clause Q'' of type 1 is produced, i.e., $\mathcal{T}, Q \vdash_{i+1}^{\mathcal{I}_{\mathsf{REQ}}} Q''$. By the definition of $\mathcal{I}_{\mathsf{REQ}}$ Q'' is produced by an inference of the form $Q', C \vdash^{\mathcal{I}_{\mathsf{REQ}}} Q''$, i.e., one that has as a main premise another clause of type 1 such that $\mathcal{T}, Q \vdash_\ell^{\mathcal{I}_{\mathsf{REQ}}} Q'$ and as a side premise a clause C such that $\mathcal{T} \vdash^{\mathcal{I}_{\mathsf{REQ}}} C$. Clearly, we have $Q', C \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$. If $C \in \mathcal{T}$ then by the previous, the induction hypothesis, and since $\mathcal{P}_{lt}$ is initialised to $\mathcal{T}$ we immediately obtain $\mathcal{P}_{lt}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$. Otherwise, we have $\mathcal{T} \vdash_j^{\mathcal{I}_{\mathsf{REQ}}} C$ with $j > 0$ and we need to show that either $C \in \mathcal{P}_{lt}$ or that Q'' can also be derived from Q' by $\mathcal{I}_{\mathsf{SLD}}$ using only clauses of $\mathcal{P}_{lt}$. This follows by the following claim:

> *Claim 4*: For each Q'' and C such that $Q', C \vdash Q''$, $\mathcal{T} \vdash_j^{\mathcal{I}_{\mathsf{REQ}}} C$, $j > 0$, either $C \in \mathcal{P}_{lt}$ or $C \notin \mathcal{P}_{lt}$ and the following conditions hold:
>
> - There exist clauses C_1 and C_2 such that $\mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} C_1, \mathcal{T} \vdash_{j-1}^{\mathcal{I}_{\mathsf{REQ}}} C_2$, and $Q', C_1 \vdash Q^*, Q^*, C_2 \vdash Q''$.
> - If C is of type 3.3 then it is function-free.

We show Claim 4 by a case analysis on the types of clauses that can be deduced from $\mathcal{T}$ by $\mathcal{I}_{\mathsf{REQ}}$, i.e., on the forms of clause C:

1. C is of type 2.1 or of type 2.2. Then, by Table B.10 such clauses can be derived by inferences involving only DL-Lite-clauses (cf. inferences 3.1+2.1=2.1 and 3.1+2.2=2.2). Hence, the claim follows from the proof of Claim 1 in Lemma 1.
2. C is of type 2.3 or of type 3.3. These cases are similar to cases 2a and 2b in the proof of *Claim 3*, hence we dispense with the details.

The previous claim implies that for a clause $C \notin \mathcal{P}_{lt}$ such that $\mathcal{T} \vdash_j^{\mathcal{I}_{\mathsf{REQ}}} C$, and $j > 0$, the inference $Q', C \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ can be transformed into a sequence of inferences of the form $Q', C_1 \vdash Q_1^*$, $Q_1^*, C_2 \vdash Q_2^*, \ldots, Q_{n-1}^*, C_n \vdash Q''$, such that for all $1 \leq i \leq n$ we have $C_i \in \mathcal{P}_{lt}$; hence $\mathcal{P}_{lt}, Q' \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ and using the induction hypothesis ($\mathcal{P}_{lt}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q'$) we get $\mathcal{P}_{lt}, Q \vdash^{\mathcal{I}_{\mathsf{SLD}}} Q''$ as required. □

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

Proof. Using the same arguments as in the case of Rapid-Lite$(Q, \mathcal{T})$ it can be shown that only a finite number of *RA*-clauses and of clauses of type 1 can be produced by the unfolding and n-shrinking rule application. More precisely, none of the side premises allowed by $\mathcal{I}_{\mathcal{EL}}$ contains (and hence introduces) a new ej-variable to the resolvent. Moreover, the function rule uses *RA*-clauses in the main premise position and clauses from $\mathcal{T}$ in the side premise position; hence it can only be applied a finite number of times.

Now let $\mathcal{R}$ = Rapid-EL$(Q, \mathcal{T})$ that is partitioned into $\mathcal{R}_Q$ and $\mathcal{R}_D$. Again, by soundness of the inference system $\mathcal{I}_{\mathcal{EL}}$ we trivially have $\mathsf{cert}(Q, \mathcal{T} \cup \mathcal{A}) \supseteq \mathsf{cert}(\mathcal{R}_Q, \mathcal{R}_D \cup \mathcal{A})$ for each $\mathcal{A}$. Next, we need to show that we also have $\mathsf{cert}(Q, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{cert}(\mathcal{R}_Q, \mathcal{R}_D \cup \mathcal{A})$.

Consider the Requiem output $\mathcal{R}^r \in \mathsf{RQR}(Q, \mathcal{T})$. $\mathcal{R}^r$ can be partitioned into $\mathcal{R}_D^r$ and $\mathcal{R}_Q^r$ satisfying the conditions of Definition 1.

By Lemma 3 every derivation of a type 1 Q' clause from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{REQ}}$ can be transformed into an extended-SLD derivation. Moreover, by Proposition 2 the derivation of Q' can be transformed into a function-compact one. Hence, by a similar induction as in the proof of Theorem 1 it follows that every Q' derived from $\mathcal{T} \cup Q$ by $\mathcal{I}_{\mathsf{REQ}}$ is also in $\mathcal{R}$.

Now consider the datalog clauses produced by $\mathcal{I}_{\mathsf{REQ}}$. This consists of clauses of one of the following forms (we have omitted analogous clauses with inverses):

$$A(x) \leftarrow R(x, y) \wedge C(y) \quad \text{(D.1)}$$

$$A(x) \leftarrow R(x, y) \quad \text{(D.2)}$$

$$R(x, y) \leftarrow S(x, y) \quad \text{(D.3)}$$

$$A(x) \leftarrow B(x) \wedge [C(x)] \quad \text{(D.4)}$$

Clauses of the form (D.1)–(D.3) are never produced by $\mathcal{I}_{\mathsf{REQ}}$. Hence, it follows that all such clauses are also in $\mathcal{R}$. Clauses of the form (D.4) either appear in $\mathcal{T}$ or are produced by sequences starting with an *RA*-clause as shown in Lemma 3. In the latter case it follows by Lemma 3 and Proposition 2 that these clauses can be produced by unfolding and n-shrinking, and thus such clauses are also in $\mathcal{R}$. In the former case (D.4) are unfolded into type 1 clause and as shown in DL-Lite Requiem unfolding corresponds to many inferences of $\mathcal{I}_{\mathcal{EL}}$ using the unfolding rule.

Finally, $\mathcal{R}_D^r$ is produced by applying the Requiem unfolding using the produced datalog clauses and $\mathcal{R}_Q^r$ by unfolding the produced queries using the unfolded clause. As shown before, $\mathcal{R}_D^r$ must be equivalent to the datalog part of $\mathcal{R}$. Moreover, like in Theorem 1 all unfoldings over queries can be transformed into inferences with side premises clauses from $\mathcal{T}$ or function-free type 3.3 clauses in $\mathcal{R}$. Hence, $\mathcal{R}_Q^r \subseteq \mathcal{R}$. □

References

[1] Alessandro Artale, Diego Calvanese, Roman Kontchakov, and Michael Zakharyaschev. The DL-Lite family and relations. *Journal of Artificial Intelligence Research*, 36(1):1–69, 2009.

[2] Franz Baader, Sebastian Brandt, and Carsten Lutz. Pushing the $\mathcal{EL}$ Envelope. In *Proceedings of the 19th International Joint Conference on AI (IJCAI-05)*, volume 5, pages 364–369, 2005.

[3] Franz Baader, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press, 2002.

[4] Franz Baader and Werner Nutt. Basic description logics. In *The Description Logic Handbook*, pages 43–95. Cambridge University Press, 2003.

[5] Leo Bachmair and Harald Ganzinger. Resolution Theorem Proving. In *Handbook of Automated Reasoning*, pages 19–99. Elsevier and MIT Press, 2001.

[6] Meghyn Bienvenu. On the Complexity of Consistent Query Answering in the Presence of Simple Ontologies. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*, 2012.

[7] Meghyn Bienvenu, Carsten Lutz, and Frank Wolter. First-Order Rewritability of Atomic Queries in Horn Description Logics. In *Proceedings of the 23rd International Joint Conference on Artificial Intelligence (IJCAI 2013)*, pages 754–760. AAAI Press, 2013.

[8] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Data Complexity of Query Answering in Description Logics. In *Proceedings of the 10th International Conference on Principles of Knowledge Representation and Reasoning (KR 06)*, pages 260–270, 2006.

[9] 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. *Journal of Automated Reasoning*, 39(3):385–429, 2007.

[10] Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, Antonella Poggi, Mariano Rodriguez-Muro, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. The MASTRO system for ontology-based data access. *Semantic Web Journal*, 2(1):43–53, 2011.

[11] Chin-Liang Chang and Richard C. T. Lee. *Symbolic logic and mechanical theorem proving*. Computer science classics. Academic Press, 1973.

[12] Pierre Chaussecourte, Birte Glimm, Ian Horrocks, Boris Motik, and Laurent Pierre. The Energy Management Adviser at EDF. In *Proceedings of the 12th International Semantic Web Conference (ISWC 2013)*, pages 49–64. Springer, 2013.

[13] Alexandros Chortaras, Despoina Trivela, and Giorgos Stamou. Optimized Query Rewriting in OWL 2 QL. In *Proceedings of the of 23rd International Conference on Automated Deduction (CADE-23)*, pages 192–206. Springer, 2011.

[14] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query Rewriting for Horn-$\mathcal{SHIQ}$ Plus Rules. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*, 2012.

[15] 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.

[16] Melvin Fitting. *First-order logic and automated reasoning (2. ed.)*. Graduate texts in computer science. Springer, 1996.

[17] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler. Conjunctive Query Answering for the Description Logic $\mathcal{SHIQ}$. *Journal of Artificial Intelligence Research (JAIR)*, 31:157–204, 2008.

[18] Georg Gottlob and Thomas Schwentick. Rewriting Ontological Queries into Small Nonrecursive Datalog Programs. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, 2012.

[19] Bernardo Cuenca Grau, Ian Horrocks, Boris Motik, Bijan Parsia, Peter Patel-Schneider, and Ulrike Sattler. OWL 2: The next step for OWL. *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(4):309–322, 2008.

[20] Ian Horrocks. *Optimising Tableaux Decision Procedures for Description Logics*. PhD thesis, University of Manchester, 1997.

[21] Ian Horrocks, Peter F Patel-Schneider, and Frank Van Harmelen. From $\mathcal{SHIQ}$ and RDF to OWL: The making of a web ontology language. *Journal of Web semantics*, 1(1):7–26, 2003.

[22] Ullrich Hustadt, Boris Motik, and Ulrike Sattler. Deciding Expressive Description Logics in the Framework of Resolution. *Information and Computation*, 206(5):579–601, 2008.

[23] Ullrich Hustadt and Renate A Schmidt. Issues of decidability for description logics in the framework of resolution. In *Proceedings Automated Deduction in Classical and Non-Classical Logics*, pages 191–205. Springer, 2000.

[24] Martha Imprialou, Giorgos Stoilos, and Bernardo Cuenca Grau. Benchmarking Ontology-based Query Rewriting Systems. In *Proceedings of the 26th AAAI Conference on Artificial Intelligence (AAAI 2012)*, 2012.

[25] Stanislav Kikot, Roman Kontchakov, and Michael Zakharyaschev. On (In)Tractability of OBDA with OWL 2 QL. In *Proceedings of the 24th International Workshop on Description Logics (DL 2011)*, 2011.

[26] Atanas Kiryakov, Barry Bishoa, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov. The Features of BigOWLIM that Enabled the BBCs World Cup Website. In *Proceedings Workshop on Semantic Data Management (SemData)*, 2010.

[27] Ilianna Kollia and Birte Glimm. SPARQL Query Answering over OWL Ontologies. *Journal of Artificial Intelligence Research (JAIR)*, 48:253–303, 2013.

[28] Domenico Lembo, Maurizio Lenzerini, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Query Rewriting for Inconsistent DL-Lite Ontologies. In *Proceedings of the 5th International Conference on Web Reasoning and Rule Systems (RR 2011)*, pages 155–169. Springer, 2011.

[29] John W. Lloyd. *Foundations of Logic Programming*. Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[30] Carsten Lutz. The Complexity of Conjunctive Query Answering in Expressive Description Logics. In *Proceedings of the 4th International Joint Conference on Automated Reasoning (IJCAR)*, pages 179–193. Springer, 2008.

[31] Roman Kontchakov Mariano Rodriguez-Muro and Michael Zakharyaschev. Query Rewriting and Optimisation with Database Dependencies in Ontop. In *Proccedings of the 26th International Workshop on Description Logics (DL 2013)*, pages 917–929, 2013.

[32] Boris Motik. Description Logics and Disjunctive Datalog—More Than just a Fleeting Resemblance? In *Proceedings of the 4th Workshop on Methods for Modalities (M4M-4)*, volume 194, pages 246–265, 2005.

[33] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. OWL 2 Web Ontology Language Profiles. W3C Recommendation, 27 October 2009.

[34] Boris Motik, Ian Horrocks, and Su Myeon Kim. Delta-Reasoner: A Semantic Web Reasoner for an Intelligent Mobile Platform. In *Proceedings of the 21st International World Wide Web Conference (WWW 2012)*, pages 63–72. ACM, 2012.

[35] Boris Motik, Ulrike Sattler, and Rudi Studer. Query Answering for OWL-DL with rules. *Journal of Web Semantics*, 3(1):41–60, 2005.

[36] Giorgio Orsi and Andreas Pieris. Optimizing Query Answering under Ontological Constraints. *Journal of Very Large Database (VLDB) Endowment*, 4(11):1004–1015, 2011.

[37] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter. Data Complexity of Query Answering in Expressive Description Logics via Tableaux. *Journal of Automated Reasoning*, 41(1):61–98, 2008.

[38] Héctor Pérez-Urbina, Ian Horrocks, and Boris Motik. Efficient Query Answering for OWL 2. In *Proceedings of the International Semantic Web Conference (ISWC2009)*, pages 489–504. Springer, 2009.

[39] 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.

[40] Mariano Rodrıguez-Muro and Diego Calvanese. High Performance Query Answering over DL-Lite Ontologies. In *Proceedings of the 13th International Conference on Principles of Knowledge Representation and Reasoning (KR 2012)*, 2012.

[41] Riccardo Rosati and Alessandro Almatelli. Improving Query Answering over DL-Lite Ontologies. In *Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR-10)*, 2010.

[42] Frantisek Simancik, Yevgeny Kazakov, and Ian Horrocks. Consequence-Based Reasoning beyond Horn Ontologies. In *Proceedings of the International Joint Conference on Artificial Inteligence (IJCAI)*, pages 1093–1098. AAAI Press, 2011.

[43] Evren Sirin and Bijan Parsia. SPARQL-DL: SPARQL Query for OWL-

DL. In *3rd OWL Experiences and Directions Workshop (OWLED-2007)*, 2007.

[44] Despoina Trivela, Giorgos Stoilos, Alexandros Chortaras, and Giorgos Stamou. Optimising Resolution-Based Rewriting Algorithms for DL Ontologies. In *Proceedings of the 26th Workshop on Description Logics (DL 2013)*, 2013.

[45] Tassos Venetis, Giorgos Stoilos, and Giorgos Stamou. Query Extensions and Incremental Query Rewriting for OWL 2 QL Ontologies. *Journal on Data Semantics*, 3(1):1–23, 2014.

[46] Sebastian Wandelt and Ralf Moeller. Distributed island-based query answering for expressive ontologies. In *Proceedings of the 5th International Conference on Advances in Grid and Pervasive Computing*, pages 461–470. Springer-Verlag, 2010.