Benchmarking Ontology-based Query Rewriting Systems

Martha Imprialou[1], **Giorgos Stoilos**[2] and **Bernardo Cuenca Grau**[1]

[1]Department of Computer Scinece,
University of Oxford, Oxford, UK
martha.imprialou@balliol.ox.ac.uk
bernardo.cuenca.grau@cs.ox.ac.uk

[2]School of Electrical and Computer Engineering
National Technical University of Athens, Greece
gstoil@image.ece.ntua.gr

Abstract

Query rewriting is a prominent reasoning technique in ontology-based data access applications. A wide variety of query rewriting algorithms have been proposed in recent years and implemented in highly optimised reasoning systems. Query rewriting systems are complex software programs; even if based on provably correct algorithms, sophisticated optimisations make the systems more complex and errors become more likely to happen. In this paper, we present an algorithm that, given an ontology as input, synthetically generates "relevant" test queries. Intuitively, each of these queries can be used to verify whether the system correctly performs a certain set of "inferences", each of which can be traced back to axioms in the input ontology. Furthermore, we present techniques that allow us to determine whether a system is unsound and/or incomplete for a given test query and ontology. Our evaluation shows that most publicly available query rewriting systems are unsound and/or incomplete, even on commonly used benchmark ontologies; more importantly, our techniques revealed the precise causes of their correctness issues and the systems were then corrected based on our feedback. Finally, since our evaluation is based on a larger set of test queries than existing benchmarks, which are based on hand-crafted queries, it also provides a better understanding of the scalability behaviour of each system.

Introduction

An important application of ontologies is *data access*, where an ontology provides the knowledge describing the meaning of application's data as well as the vocabulary used to formulate user queries. In this setting, query languages are often based on conjunctive queries (CQs), and the answers to queries reflect both the application's data and the knowledge captured by the ontology (Calvanese et al. 2007; Poggi et al. 2008; Glimm et al. 2007; Lutz, Toman, and Wolter 2009; Ortiz, Calvanese, and Eiter 2006).

The need for efficient query answering in ontology-based data access applications has motivated the development of (families of) *lightweight* ontology languages, such as DL-Lite (Calvanese et al. 2007) (which underpins the OWL 2 QL profile (Motik et al. 2009)), DLP (which underpins the OWL 2 RL profile), as well as (fragments of) datalog$^{\pm}$ (Calì et al. 2010). These languages are tailored such that query answering becomes tractable in data complexity (and even in logarithmic space in the case of the logics underlying OWL 2 QL) .

In this setting, *query rewriting* is a common reasoning technique of choice. Intuitively, a rewriting of a query $\mathcal{Q}$ w.r.t. an ontology $\mathcal{T}$ is another query (typically a union of CQs or a datalog query) that captures the information in $\mathcal{T}$ relevant for answering $\mathcal{Q}$ w.r.t. $\mathcal{T}$ and arbitrary data. Due to the growing interest in ontology-based data access, many rewriting algorithms have been proposed in recent years and implemented in (both commercial and non-commercial) reasoning systems. Examples of such systems include QuOnto (Acciarri et al. 2005), Requiem (Pérez-Urbina, Horrocks, and Motik 2009), Presto (Rosati and Almatelli 2010), Nyaya (Gottlob, Orsi, and Pieris 2011), Rapid (Chortaras, Trivela, and Stamou 2011), and IQAROS.[1]

Despite being based on provably correct algorithms, sophisticated optimisations needed in practice make these query rewriting systems rather complex and error-prone software programs. Consequently, both system and application developers would greatly benefit from practical and reliable benchmarking approaches to query rewriting.

To the best of our knowledge, little work has been done so far towards the systematic benchmarking of ontology-based query rewriting systems, and developers have relied mostly on "hand-crafted" ontologies and queries for evaluating their systems and comparing them to others. An example of a popular benchmark for query rewriting, which was first used in (Pérez-Urbina, Horrocks, and Motik 2009), consists of nine ontologies and five hand-crafted queries for each of them; systems are evaluated regarding the size of the computed rewritings and the rewriting computation time.

Existing benchmarks, however, have several important limitations. First, they are ad hoc: test queries are manually generated for each specific ontology using the developers' personal knowledge about the ontologies in the benchmark. Second, given a test query $\mathcal{Q}$ and ontology $\mathcal{T}$, it is currently not possible to automatically check whether the relevant systems are *sound* (i.e., they compute only actual query answers to $\mathcal{Q}$ w.r.t. $\mathcal{T}$ and arbitrary data) and *complete* (i.e., they compute all answers to $\mathcal{Q}$ w.r.t. $\mathcal{T}$ and arbitrary data).

In this paper, we present a novel approach for addressing

[1]http://code.google.com/p/iqaros/

these important limitations. First, we present an algorithm that synthetically generates test queries for an ontology $\mathcal{T}$. Our algorithm is *generic*, in the sense that it is applicable to a wide range of ontologies and, in particular, to (Horn) ontologies that can be captured by *existential rules* (Baget et al. 2011; Calì et al. 2010). Furthermore, our algorithm is *fully automatic* and hence does not require experts' involvement. Most importantly, each generated query is *relevant* to $\mathcal{T}$, in the sense that it can be used to verify whether a system correctly performs a certain set of "inferences", each of which can be traced back to axioms in $\mathcal{T}$. Second, we present techniques that allow us to determine whether a given system is unsound and/or incomplete for given $\mathcal{Q}$ and $\mathcal{T}$; these techniques are easy to implement and in many cases help detect errors with marginal manual effort.

Our evaluation shows that most publicly available query rewriting systems are unsound and/or incomplete, even on commonly used benchmark ontologies; most importantly, our techniques revealed the precise causes of their unsoundness and incompleteness issues, and as a result some of the evaluated systems were actually corrected by their developers based on our feedback. Finally, since our collection of tests queries is more "exhaustive" than those in existing benchmarks, our evaluation provides further insights into the scalability behaviour of each system.

Preliminaries

We use standard notions of first-order constants, (free and bound) variables, function symbols, terms, substitutions, predicates, atoms, (ground) formulae, and sentences. A *fact* is a ground atom, and an *instance* is a finite set of facts. For ϕ a formula, with $\phi(\vec{x})$ we denote that $\vec{x}$ are the free variables of ϕ, while for σ a substitution, $\phi\sigma$ is the result of applying σ to ϕ. Satisfiability and entailment are defined as usual.

Ontologies We use description logics in the wider framework of first order logic, and identify a DL TBox $\mathcal{T}$ with a finite set of first order sentences. Specialised DL syntax will sometimes be used in examples, and we assume that the reader is familiar with the basics of such syntax (Baader et al. 2002).

Existential Rules An *existential rule* (Baget et al. 2011; Calì et al. 2010) is a sentence of the form

$$\forall\vec{x}.\forall\vec{z}.[\phi(\vec{x},\vec{z}) \rightarrow \exists\vec{y}.\psi(\vec{x},\vec{y})] \tag{1}$$

where $\phi(\vec{x},\vec{z})$ and $\psi(\vec{x},\vec{y})$ are conjunctions of function-free atoms and $\vec{x},\vec{y}$ and $\vec{z}$ are pair-wise disjoint. Formula ϕ is the *body*, formula ψ is the *head*, and universal quantifiers are often omitted. Note that, by definition, existential rules are *safe*–that is, all variables in $\vec{x}$ occur both in the body and the head. If $\vec{y}$ is empty, the rule is *datalog*. Finally, the *instantiation* of a datalog rule r w.r.t. a substitution σ mapping all variables in r to constants is the instance I^r_σ consisting all facts $B\sigma$ with B a body atom in r. If σ is an injective mapping, then I^r_σ is an *injective instantiation* of r.

Many popular DLs (e.g., those underlying the OWL 2 profiles (Motik et al. 2009)) can be captured by existential rules.

Queries A query $\mathcal{Q}$ is a finite set of sentences containing a distinct query predicate Q. A tuple of constants $\vec{a}$ is an answer to $\mathcal{Q}$ w.r.t. TBox $\mathcal{T}$ and instance I if the arity of $\vec{a}$ agrees with the arity of Q and $\mathcal{T} \cup I \cup \mathcal{Q} \models Q(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup I)$ the answers to $\mathcal{Q}$ w.r.t. $\mathcal{T} \cup I$. A query $\mathcal{Q}$ is a union of conjunctive queries (UCQ) if it is a set of datalog rules containing Q in the head but not in the body. In this case, we sometimes abuse notation and use Q to denote the head atom, rather than the head predicate. A UCQ is a conjunctive query (CQ) if it has exactly one rule.

Query Rewriting Intuitively, a *rewriting* of $\mathcal{Q}$ w.r.t. $\mathcal{T}$ is another query that captures all the information from $\mathcal{T}$ relevant for answering $\mathcal{Q}$ over an arbitrary instance I (Calvanese et al. 2007; Pérez-Urbina, Motik, and Horrocks 2010; Gottlob, Orsi, and Pieris 2011). UCQs and datalog are common target languages for query rewriting; furthermore, virtually all TBoxes to which existing rewriting techniques are applicable can be captured by existential rules.

Definition 1. A *datalog rewriting* of a CQ $\mathcal{Q}$ w.r.t. TBox $\mathcal{T}$ is a tuple $\langle \mathcal{R}_D, \mathcal{R}_Q \rangle$ with $\mathcal{R}_D$ a set of datalog rules not mentioning Q and $\mathcal{R}_Q$ a UCQ with query predicate Q whose body atoms mention only predicates from $\mathcal{T}$, and where for each I using only predicates from $\mathcal{T}$ we have

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

The rewriting $\langle \mathcal{R}_D, \mathcal{R}_Q \rangle$ is a *UCQ rewriting* if $\mathcal{R}_D = \emptyset$.

The Chase Given a set of existential rules $\mathcal{R}$ and an instance I, CQ answering over $\mathcal{R} \cup I$ can be characterised using the *chase*: a technique that computes a (possibly infinite) set $\mathsf{chase}(\mathcal{R} \cup I)$ of facts implied by $\mathcal{R}$ and I in a forward-chaining manner; the result is a universal model over which the CQ can then be evaluated. Although many chase procedures have been proposed in the literature (Fagin et al. 2005; Deutsch, Nash, and Remmel 2008; Marnette 2009), all variants of the chase satisfy the following basic properties:

- $\mathsf{cert}(\mathcal{Q}, \mathcal{R} \cup I) = \mathsf{cert}(\mathcal{Q}, \mathsf{chase}(\mathcal{R} \cup I))$ for all $\mathcal{Q}, \mathcal{R}, I$.
- For each ground fact α mentioning only constants and functions from $\mathcal{R} \cup I$, $\mathcal{R} \cup I \models \alpha$ iff $\alpha \in \mathsf{chase}(\mathcal{R} \cup I)$.

Query Rewriting Systems

Many optimised query rewriting systems for various ontology languages have been developed in recent years (Calvanese et al. 2007; Rosati and Almatelli 2010; Chortaras, Trivela, and Stamou 2011; Gottlob, Orsi, and Pieris 2011). To abstract from the implementation details, and focus only on the properties of the system that we want to test for, we introduce an abstract notion of a query rewriting system.

Definition 2. A *query rewriting system* rew for a DL $\mathcal{L}$ is a computable function that, for each TBox $\mathcal{T} \in \mathcal{L}$ and each CQ $\mathcal{Q}$ with head predicate Q computes in a finite number of steps a tuple $\mathsf{rew}(\mathcal{Q}, \mathcal{T}) = \langle \mathcal{R}_D, \mathcal{R}_Q \rangle$ where $\mathcal{R}_D$ is a set of datalog rules not containing Q and $\mathcal{R}_Q$ is a UCQ with head predicate Q whose body atoms mention only predicates from $\mathcal{T}$. The system rew is *$(\mathcal{Q}, \mathcal{T})$-sound* if $\mathsf{cert}(\mathcal{R}_Q, \mathcal{R}_D \cup I) \subseteq \mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup I)$ for every instance I containing only predicates from $\mathcal{T}$. It is *$(\mathcal{Q}, \mathcal{T})$-complete* if $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup I) \subseteq \mathsf{cert}(\mathcal{R}_Q, \mathcal{R}_D \cup I)$ for every instance I containing only predicates from $\mathcal{T}$.

Most query rewriting systems are based on algorithms whose behaviour on input $\mathcal{T}$ and $\mathcal{Q}$ can be characterised by the application of the following two steps:

1. *Pre-processing*, where the input TBox $\mathcal{T}$ is transformed into a set $\mathcal{R}_\mathcal{T}$ of existential rules.
2. *Query rewriting*, where a calculus (often a variant of resolution) is used to derive $\mathcal{R}_D$ and $\mathcal{R}_Q$ from $\mathcal{R}_\mathcal{T}$ and $\mathcal{Q}$.

Consequently, two main sources of implementation errors for rewriting systems can be identified, namely those caused by either an incorrect pre-processing, or an incorrect implementation of the calculus.

Example 3. Let $\mathcal{T}$ consist of the following DL axiom:

$$\mathsf{Student} \equiv \mathsf{Person} \sqcap \exists \mathsf{enrolled}.\mathsf{Course} \quad (2)$$

TBox $\mathcal{T}$ can be translated into the following set $\mathcal{R}$ of equivalent existential rules:

$$\mathsf{Student}(x) \rightarrow \mathsf{Person}(x) \quad (3)$$

$$\mathsf{Person}(x) \wedge \mathsf{enrolled}(x, y) \wedge \mathsf{Course}(y) \rightarrow \mathsf{Student}(x) \quad (4)$$

$$\mathsf{Student}(x) \rightarrow \exists y.(\mathsf{enrolled}(x, y) \wedge \mathsf{Course}(y)) \quad (5)$$

Some rewriting systems that would accept $\mathcal{T}$ as a valid input do not handle qualified existential restrictions (i.e., concepts such as $\exists$enrolled.Course). If pre-processing $\mathcal{T}$ naively, one such system rew_1 might "ignore" the conjunction involving sub-concept $\exists$enrolled.Course and translate $\mathcal{T}$ into $\mathcal{R}^1_\mathcal{T}$ consisting only of rule (3) and rule $\mathsf{Person}(x) \rightarrow \mathsf{Student}(x)$. Such system would be unsound for $\mathcal{T}$: given query $\mathcal{Q}_1 = \mathsf{Student}(x) \rightarrow Q(x)$, it would compute the UCQ rewriting $\langle \emptyset, \{\mathcal{Q}_1, \mathcal{Q}'_1\}\rangle$, with $\mathcal{Q}'_1 = \mathsf{Person}(x) \rightarrow Q(x)$, which for instance $I_1 = \{\mathsf{Person}(c)\}$ leads to the incorrect answer c.

Consider now a system rew_2 that can handle qualified existential restrictions, but which, due to an implementation error, translates $\mathcal{T}$ into $\mathcal{R}^2_\mathcal{T}$ consisting of rules (3), (4), and rule $\mathsf{Student}(x) \rightarrow \exists y.(\mathsf{enrolled}(x, y))$, thus leaving out conjunct $\mathsf{Course}(y)$ in rule (5). Such system is incomplete for $\mathcal{T}$, as witnessed by $\mathcal{Q}_2 = \mathsf{enrolled}(x, y) \wedge \mathsf{Course}(y) \rightarrow Q(x)$ and $I_2 = \{\mathsf{Student}(d)\}$.

As described later on in our evaluation section, the aforementioned normalisation issues are related to actual errors we detected in the Rapid system (Chortaras, Trivela, and Stamou 2011) using our approach, and which were not revealed by existing benchmarks.

Finally, as already mentioned, state of the art implementations of rewriting calculi are highly optimised. As described later on, our approach also allowed us to detect several correctness issues in implemented optimisations; these include errors in ordering criteria for query atoms in IQAROS, and issues with subsumption optimisations in Requiem (Pérez-Urbina, Horrocks, and Motik 2009), among others. ♢

Example 3 suggests that implementation errors in both pre-processing and rewriting steps can be effectively "uncovered" by a suitable choice of test queries and subsequent analysis of the systems' outputs for such relevant queries. In what follows, we present an algorithm for computing test queries that are relevant to a TBox. We then present techniques for checking systems' correctness for a test query.

Algorithm 1 $\mathsf{QueryGeneration}(\mathcal{R}, bound, \Sigma)$

input: Existential rules $\mathcal{R}$; integer $bound$.

1: $I_0 := \emptyset$ **and** $\mathsf{root} := \emptyset$
2: **for all** $r \in \mathcal{R}$ **do**
3: $\quad I^r_\mu :=$ injective instantiation of r to fresh const. $\vec{a}$.
4: $\quad I_0 := I_0 \cup I^r_\mu$
5: $\quad \mathsf{root} := \mathsf{root} \cup \{\vec{a}\}$
6: **end for**
7: $I_c := I_0$
8: **repeat**
9: $\quad I_c := I_c \cup \mathsf{applyChaseRule}(\mathcal{R}, I_c)$
10: **until** $\mathsf{terminate}(I_c, bound)$
11: $I := I_c \setminus I_0$ **and** $\mathsf{CQ} := \emptyset$
12: **for all** $\vec{a} \in \mathsf{root}$ **do**
13: $\quad$ **for all** $I' \in \mathsf{paths}(\vec{a}, I, \Sigma)$ **do**
14: $\quad\quad$ Let σ map each constant in I' to fresh variable
15: $\quad\quad \mathsf{CQ} := \mathsf{CQ} \cup \{\bigwedge_{P(\vec{b}) \in I'} P(\sigma(\vec{b})) \rightarrow Q(\sigma(\vec{a}))\}$
16: $\quad$ **end for**
17: **end for**
18: **return** CQ

Test Query Generation

Virtually all TBoxes to which state of the art rewriting systems are applicable can be captured by existential rules containing only unary and binary predicates. Thus, we only consider such TBoxes $\mathcal{T}$ in this section. Translation of $\mathcal{T}$ into existential rules $\mathcal{R}$ can be performed using well-known structural transformations, which might introduce "fresh" symbols Σ (e.g., see (Motik, Shearer, and Horrocks 2009)).

Algorithm 1 can then be used to compute test queries from $\mathcal{R}$. The algorithm directly exploits the chase technique, and works in three main stages:

1. *Chase initialisation*, via instantiation of body atoms in $\mathcal{R}$.
2. *Chase expansion*, up to a pre-defined bound.
3. *Query generation*, by "navigating" the expanded chase while replacing ground terms by fresh variables.

Chase initialisation For each rule $r \in \mathcal{R}$, Algorithm 1 instantiates all body atoms using fresh constants, which are marked as *root constants*, thus constructing an instance I_0 that "triggers" the application of each rule in $\mathcal{R}$ (lines 2–6). For the rules in Example 3 we obtain the following initial instance, where $\mathsf{root} = \{a, (b, c)\}$:[2]

$$I_0 = \{\mathsf{Student}(a), \mathsf{Person}(b), \mathsf{enrolled}(b, c), \mathsf{Course}(c)\}$$

Chase expansion A chase procedure is then used to materialise new implied facts from existing ones by forward chaining application of rules in $\mathcal{R}$ (lines 8–10). Intuitively, when initialised with I_0, the properties of the chase ensure that each rule application represents an inference that is relevant to derive some answer to some query $\mathcal{Q}$ w.r.t. some input instance (and for the given, fixed, TBox $\mathcal{T}$).

[2]Note that rules (3) and (5) share the same body, so it suffices to instatiate one of them.

Since the chase might not terminate, the algorithm accepts an integer bound (e.g., maximum number of generated facts), which can be used as a termination condition. Alternatively, one can use any of the available *acyclicity conditions* for existential rules (e.g., (Fagin et al. 2005; Marnette 2009)), which analyse the information flow between the rules to ensure that no cyclic generation of fresh terms occurs. If $\mathcal{R}$ satisfies any such condition, the chase terminates, and one can dispense with the input bound. In our example, chase expansion would lead to the following instance I_c, where a_1 and b_1 are fresh constants generated by the application of rule (5).

$$\begin{aligned} I_c \;=\; & I_0 \cup \{\mathsf{Person}(a), \mathsf{enrolled}(a, a_1), \mathsf{Course}(a_1) \\ & \mathsf{Student}(b), \mathsf{enrolled}(b, b_1), \mathsf{Course}(b_1)\} \end{aligned}$$

Query generation Instance $I = I_c \setminus I_0$ contains all facts that have been derived from I_0 during chase expansion. The properties of the chase ensure that I is *forest-shaped*. More precisely, since I contains only unary and binary predicates, we can define $\mathcal{G}(I)$ as the graph whose nodes are the constants in I and which contains an (undirected) edge between c and d iff c and d occur together in a fact from I; then $\mathcal{G}(I)$ consists of a set of disconnected trees each of which is rooted at an individual from root.

Algorithm 1 generates queries by traversing $\mathcal{G}(I)$ (lines 12–17) along "paths". More precisely, $\mathsf{paths}(\vec{a}, I, \Sigma)$ (see line 13) is the set of all instances $I' \subseteq I$ not containing predicates from Σ and s.t. $\mathcal{G}(I')$ is a path in $\mathcal{G}(I)$ involving a constant from $\vec{a}$. Each such I' is then transformed into a CQ by mapping each constant to a fresh variable s.t. root constants are mapped to answer variables in the head of the query. Clearly, all queries mentioned in Example 3 would be generated by Algorithm 1 for input rules (3)–(5).

Note that, in the worst-case, Algorithm 1 might generate exponentially many queries w.r.t. the size of the input. As shown in our evaluation, however, the number of generated queries for commonly-used benchmark ontologies is relatively modest, and they can be computed in just a few seconds. Finally, note that relevant test queries could also be generated from I by considering instances $I' \subseteq I$ whose graph is a sub-tree of $\mathcal{G}(I)$, rather than simply a path; however, on the one hand, this results in a blowup in the number of queries in practice and, on the other hand, our evaluation suggests that path queries suffice for uncovering many errors in systems and for analysing and comparing their behaviour.

Testing Correctness

Algorithm 1 provides a flexible way for generating relevant test queries. In this section, we present practical techniques for testing soundness and completeness of a rewriting system for a given test query and a given TBox.

Testing Soundness

Assume that $\mathsf{rew}(\mathcal{Q}, \mathcal{T}) = \langle \mathcal{R}_D, \mathcal{R}_Q \rangle$. Clearly, if we have $\mathcal{T} \cup \mathcal{Q} \models \mathcal{R}_D \cup \mathcal{R}_Q$, the properties of first-order logic entailment ensure that each answer to $\mathcal{Q}$ w.r.t. $\mathcal{R}_D \cup \mathcal{R}_Q$ and an instance I is also an answer to $\mathcal{Q}$ w.r.t. $\mathcal{T} \cup I$, and hence the query rewriting system rew is $(\mathcal{Q}, \mathcal{T})$-sound.

Proposition 4 shows that $\mathcal{T} \cup \mathcal{Q} \models \mathcal{R}_D \cup \mathcal{R}_Q$ can be checked using any reasoner that is sound and complete for checking entailment of a fact by $\mathcal{T}$ and an instance. For standard DLs, example such reasoners include HermiT (Motik, Shearer, and Horrocks 2009) and Pellet (Sirin et al. 2007).

Proposition 4. *Let* $\mathsf{rew}(\mathcal{Q}, \mathcal{T}) = \langle \mathcal{R}_D, \mathcal{R}_Q \rangle$. *Then, we have that* $\mathcal{T} \cup \mathcal{Q} \models \mathcal{R}_D \cup \mathcal{R}_Q$ *iff the following conditions hold:*

1. $\mathcal{T} \cup I^r_\sigma \models H\sigma$ *for each* $r \in \mathcal{R}_D$, *where* H *is the head of* r *and* I^r_σ *an injective instantiation.*
2. $\mathcal{T} \cup \mathcal{Q} \cup I^r_\sigma \models Q\sigma$ *for each* $r \in \mathcal{R}_Q$, *where* Q *is the head of* r *and* I^r_σ *is an injective instantiation.*

Proof. Since Q does not occur in $\mathcal{R}_D$, we have that $\mathcal{T} \cup \mathcal{Q} \models \mathcal{R}_D \cup \mathcal{R}_Q$ iff $\mathcal{T} \models \mathcal{R}_D$ and $\mathcal{T} \cup \mathcal{Q} \models \mathcal{R}_Q$. Furthermore, it is easy to check that a set $\mathcal{F}$ of sentences entails a datalog rule r with head H iff $\mathcal{F} \cup I^r_\sigma \models H\sigma$ for I^r_σ an injective instantiation; thus, since $\mathcal{R}_D$ and $\mathcal{R}_Q$ consist of datalog rules, we have $\mathcal{T} \models \mathcal{R}_D$ iff Condition 1 holds and $\mathcal{T} \cup \mathcal{Q} \models \mathcal{R}_Q$ iff Condition 2 holds. □

In practice, however, checking soundness of rew using a fully-fledged DL reasoner, as suggested by Proposition 4, has the drawback that the process is subject to implementation errors of the DL reasoner. Although this problem can be alleviated by performing the required entailment tests using several DL reasoners to detect possible discrepancies, an alternative is to use a chase procedure, such as the one we used for query generation (see Algorithm 1). Such procedure does not need to be optimised, and consequently its implementation can be rather simple and transparent.

The following proposition shows how to exploit the chase for checking $(\mathcal{Q}, \mathcal{T})$-soundness of rew.

Proposition 5. *Let* $\mathsf{rew}(\mathcal{Q}, \mathcal{T}) = \langle \mathcal{R}_D, \mathcal{R}_Q \rangle$ *and let* $\mathcal{R}$ *be a set of existential rules such that* $\mathcal{T}$ *and* $\mathcal{R}$ *are logically equivalent. The following conditions imply that* rew *is* $(\mathcal{Q}, \mathcal{T})$*-sound:*

1. $H\sigma \in \mathsf{chase}(\mathcal{R} \cup I^r_\sigma)$ *for each* $r \in \mathcal{R}_D$, *where* H *is the head of* r *and* I^r_σ *an injective instantiation.*
2. $\vec{a} \in \mathsf{cert}(\mathcal{Q}, \mathsf{chase}(\mathcal{R} \cup I^r_\sigma))$ *for each* $r \in \mathcal{R}_Q$, *where* Q *is the head of* r *and* I^r_σ *an injective instantiation mapping the variables in* Q *to* $\vec{a}$.

Furthermore, if Condition 2 does not hold, then rew *is not* $(\mathcal{Q}, \mathcal{T})$*-sound.*

Proof. To show the first claim in the proposition, it suffices to prove that Condition 1 (resp. Condition 2) in the proposition implies Condition 1 (resp. Condition 2) in Proposition 4. Let $r \in \mathcal{R}_D$, and assume that $H\sigma \in \mathsf{chase}(\mathcal{R} \cup I^r_\sigma)$ with I^r_σ an injective instantiation; since $H\sigma$ mentions only individuals from I^r_σ (recall that rules are assumed to be safe), the properties of the chase (see preliminaries) ensure that $\mathcal{R} \cup I^r_\sigma \models H\sigma$; but then, since $\mathcal{T} \models \mathcal{R}$, we have $\mathcal{T} \cup I^r_\sigma \models H\sigma$, as required. Finally, let $r \in \mathcal{R}_Q$ and let $\vec{a} \in \mathsf{cert}(\mathcal{Q}, \mathsf{chase}(\mathcal{R} \cup I^r_\sigma))$, where I^r_σ is an injective instantiation mapping the variables in Q to $\vec{a}$. The properties of chase ensure that $\vec{a} \in \mathsf{cert}(\mathcal{Q}, \mathcal{R} \cup I^r_\sigma)$, which implies $\mathcal{R} \cup \mathcal{Q} \cup I^r_\sigma \models Q\sigma$. Since $\mathcal{T} \models \mathcal{R}$, we then have $\mathcal{T} \cup \mathcal{Q} \cup I^r_\sigma \models Q\sigma$, as required.

To show proposition's last claim, consider $r \in \mathcal{R}_Q$ and an injective instantiation I^r_σ, where $\vec{x}$ are the variables in Q and $\vec{a} = \sigma(\vec{x})$. Clearly, $\vec{a} \in \mathsf{cert}(\mathcal{R}_Q, \mathcal{R}_D \cup I^r_\sigma)$; furthermore, I^r_σ contains only predicates from $\mathcal{T}$. Next, assume $\vec{a} \notin \mathsf{cert}(\mathcal{Q}, \mathsf{chase}(\mathcal{R} \cup I^r_\sigma))$; then, the properties of the chase imply that $\vec{a} \notin \mathsf{cert}(\mathcal{Q}, \mathcal{R} \cup I^r_\sigma)$; finally, since $\mathcal{R} \models \mathcal{T}$, we have $\vec{a} \notin \mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup I^r_\sigma)$. Thus, rew is not $(\mathcal{Q}, \mathcal{T})$-sound. □

Note that Proposition 5 provides only a *sufficient* condition for $(\mathcal{Q}, \mathcal{T})$-soundness of rew; indeed, it may be the case that Condition 1 fails (i.e., $H\sigma \notin \mathsf{chase}(\mathcal{R} \cup I^r_\sigma)$ for some $r \in \mathcal{R}_D$), but rew is $(\mathcal{Q}, \mathcal{T})$-sound. In contrast, as shown, failure of Condition 2 for some $r \in \mathcal{R}_Q$ provides a counter-example for $(\mathcal{Q}, \mathcal{T})$-soundness. Thus, if $\mathsf{rew}(\mathcal{Q}, \mathcal{T})$ is a UCQ (i.e., $\mathcal{R}_D = \emptyset$), Proposition 5 provides both a *necessary and sufficient* condition for $(\mathcal{Q}, \mathcal{T})$-soundness.

Example 6. Consider TBox $\mathcal{T}$, rules $\mathcal{R}$, queries $\mathcal{Q}_1, \mathcal{Q}'_1$ and system rew_1 from Example 3. For $\mathcal{Q}_1$ and $\mathcal{T}$, rew_1 computes $\mathsf{rew}_1(\mathcal{Q}_1, \mathcal{T}) = \langle \emptyset, \{\mathcal{Q}_1, \mathcal{Q}'_1\}\rangle$. Let $I_\sigma = \{\mathsf{Person(a)}\}$ be an injective instantiation of $\mathcal{Q}'_1$ for $\sigma = \{x \mapsto a\}$. Clearly, $\mathcal{T} \models \mathcal{R}$ and $a \notin \mathsf{cert}(\mathcal{Q}_1, \mathsf{chase}(\mathcal{R} \cup I_\sigma))$. Thus, Proposition 5 implies that rew_1 is indeed unsound for $\mathcal{Q}_1$ and $\mathcal{T}$. ♢

In practice, to ensure $(\mathcal{Q}, \mathcal{T})$-soundness of rew it suffices to check that Conditions 1 and 2 in Proposition 5 hold already for a subset of the corresponding chase, in which case chase termination does not need to be ensured. On the one hand, these conditions involve checking whether a particular fact can be found in the chase (or whether a particular answer can be derived from the chase); on the other hand, chase construction is monotonic (i.e., derived facts are never deleted), so if a fact can be found in (or an answer can be derived from) a finite subset of the chase, then the fact (or answer) is guaranteed to hold in the final chase.

To prove unsoundness, however, one needs to check failure of Condition 2, which requires the full expansion of $\mathsf{chase}(\mathcal{R} \cup I^r_\sigma)$ for each relevant $r \in \mathcal{R}_Q$; thus, chase termination becomes an important issue. Our evaluation will show that these issues can be dealt with, and Proposition 5 can be used effectively to determine soundness and unsoundness in practice for typical benchmark ontologies.

Testing Completeness

Our strategy for detecting sources of incompleteness is to look for and (semi)automatically explicate "disagreements" between query rewriting systems for given $\mathcal{Q}$ and $\mathcal{T}$. The following definition provides the required notions to compare systems' outputs for completeness evaluation purposes.

Definition 7. Let $\mathsf{rew}_i(\mathcal{Q}, \mathcal{T}) = \langle \mathcal{R}^i_D, \mathcal{R}^i_Q\rangle$ for $i \in \{1, 2\}$. We say that rew_2 *extends* rew_1 for $\mathcal{Q}$ and $\mathcal{T}$ if the following condition holds for each instance I mentioning only predicates from $\mathcal{T}$:

$$\mathsf{cert}(\mathcal{R}^1_Q, \mathcal{R}^1_D \cup I) \subseteq \mathsf{cert}(\mathcal{R}^2_Q, \mathcal{R}^2_D \cup I) \qquad (6)$$

If rew_2 *extends* rew_1 and the inclusion in Condition (6) is proper for some I, we say that rew_2 *strictly extends* rew_1.

Intuitively, if rew_2 extends rew_1, then it is "at least as complete as" rew_1 for the given $\mathcal{Q}$ and $\mathcal{T}$. Thus, when evaluating several systems in practice, if one of them extends all the others, we have strong evidence to believe that such system is $(\mathcal{Q}, \mathcal{T})$-complete. Furthermore, if a system rew_2 is $(\mathcal{Q}, \mathcal{T})$-sound and strictly extends rew_1 for $\mathcal{Q}$ and $\mathcal{T}$, we can conclude that rew_1 is *incomplete* for $\mathcal{Q}$ and $\mathcal{T}$.

The following proposition provides sufficient conditions for checking whether rew_2 (strictly) extends rew_1; hence, when used in combination with Proposition 5, it allows us to draw conclusions about systems' completeness.

Proposition 8. *Let* $\mathsf{rew}_i(\mathcal{Q}, \mathcal{T}) = \langle \mathcal{R}^i_D, \mathcal{R}^i_Q\rangle$ *for* $i \in \{1, 2\}$. *If both of the following conditions hold, then* rew_2 *extends* rew_1 *for* $\mathcal{Q}$ *and* $\mathcal{T}$*:*

1. $H\sigma \in \mathsf{chase}(\mathcal{R}^2_D \cup I^r_\sigma)$ *for each* $r \in \mathcal{R}^1_D$ *with head* H*, where* I^r_σ *is an injective instantiation.*
2. $Q\sigma \in \mathsf{chase}(\mathcal{R}^2_Q \cup I^r_\sigma)$ *for each* $r \in \mathcal{R}^1_Q$ *with head* Q*, where* I^r_σ *is an injective instantiation.*

Furthermore, if $\mathcal{R}^1_D = \mathcal{R}^2_D = \emptyset$, rew_2 *extends* rew_1 *for* $\mathcal{Q}$ *and* $\mathcal{T}$*, and* $Q\sigma \notin \mathsf{chase}(\mathcal{R}^1_Q \cup I^r_\sigma)$ *for some* $r \in \mathcal{R}^2_Q$ *with* I^r_σ *injective, then* rew_2 *strictly extends* rew_1 *for* $\mathcal{Q}$ *and* $\mathcal{T}$*.*

Proof. Assume that $H\sigma \in \mathsf{chase}(\mathcal{R}^2_D \cup I^r_\sigma)$ for each $r \in \mathcal{R}^1_D$; the properties of chase ensure that $\mathcal{R}^2_D \cup I^r_\sigma \models H\sigma$ for each $r \in \mathcal{R}^1_D$; but then, since I^r_σ is an injective instantiation of r, this implies that $\mathcal{R}^2_D \models \mathcal{R}^1_D$. Similarly, if Condition 2 holds, then we have $\mathcal{R}^2_Q \models \mathcal{R}^1_Q$. As a result, (6) holds by the properties of entailment, as required.

Finally, assume that $\mathcal{R}^1_D = \mathcal{R}^2_D = \emptyset$ and $r \in \mathcal{R}^2_Q$ exists such that $Q\sigma \notin \mathsf{chase}(\mathcal{R}^1_Q \cup I^r_\sigma)$ for I^r_σ an injective instantiation. The properties of the chase then ensure that $\mathcal{R}^1_Q \cup I^r_\sigma \not\models Q\sigma$; since $\mathcal{R}^2_Q \cup I^r_\sigma \models Q\sigma$, and rew_2 extends rew_1, the inclusion in (6) is indeed proper. □

Note that $\mathcal{R}^i_D$ and $\mathcal{R}^i_Q$ consist of datalog rules; thus, the chase computations involved in Proposition 8 are guaranteed to terminate. Furthermore, if we determine that rew_2 strictly extends rew_1, we can track the application of chase rules in the corresponding chase expansion to detect the source of the systems' disagreement in a (semi)automatic way. As described in the evaluation section, this allowed us to detect the sources of incompleteness in practice.

Example 9. Consider $\mathcal{Q}_2$, $\mathcal{T}$ and rew_2 from Example 3 and let $\mathsf{rew}_2(\mathcal{Q}_2, \mathcal{T}) = \langle \emptyset, \mathcal{R}^2_Q\rangle$. Since rew_2 computes rewritings based on $\mathcal{R}^2_\mathcal{T}$ in Example 3, we have $\mathcal{Q}'_2 \notin \mathcal{R}^2_Q$ for $\mathcal{Q}'_2 = \mathsf{Student}(x) \rightarrow Q(x)$. Let rew_3 be a sound system s.t. $\mathsf{rew}_3(\mathcal{Q}_2, \mathcal{T}) = \langle \emptyset, \mathcal{R}^2_Q \cup \{\mathcal{Q}'_2\}\rangle$. rew_3 strictly extends rew_2 since $Q(a) \notin \mathsf{chase}(\mathcal{R}^2_Q \cup I_\sigma)$ for $I_\sigma = \{\mathsf{Student}(a)\}$; thus, rew_2 is not $(\mathcal{Q}_2, \mathcal{T})$-complete. ♢

Experiments

We have implemented a test query generator (see Algorithm 1) that is applicable to TBoxes in the the DL $\mathcal{ELHI}$.[3] We then applied our query generator to the benchmark ontologies described in (Pérez-Urbina, Horrocks, and Motik 2009).

[3] Available at http://code.google.com/p/sygenia/

For all test ontologies, except for A, AX, and S, the chase naturally terminated in just a few seconds. Instead of fixing an a priori integer bound for the "cyclic" ontologies (as required by Algorithm 1), our implementation annotates each skolem constant generated by the chase with the rules with existentials in the head that have been applied so far to generate them. If such a rule r is applicable to facts involving a skolem constant, but r has been used before in order to generate this particular skolem constant, then we detect a "cycle"; what we bound is the number of such detected cycles. Table 1 summarises our results. We can observe that test queries for all ontologies can be generated rather efficiently and the number of them is also relatively small.

$\mathcal{T}$	$t(ms)$	♯CQs	$\mathcal{T}$	$t(ms)$	♯CQs
A	1 264	114	S	1 225	185
AX	2 856	453	U	847	81
P1	398	1	UX	1 548	101
P5	446	15	V	3 601	102
P5X	574	130			

Table 1: Statistics of query generation

We used these queries to evaluate and compare CGLLR,[4] REQUIEM,[5] Rapid,[6] IQAROS,[7] and Nyaya;[8] we didn't evaluate Presto and QuOnto, which aren't publicly available.

Correctness Evaluation

We ran all systems over all ontologies and all our test queries and analysed their output (UCQ) rewritings. Contrary to previous evaluations (Pérez-Urbina, Horrocks, and Motik 2009; Chortaras, Trivela, and Stamou 2011; Rosati and Almatelli 2010), which were based on the same ontologies, but which used "hand-crafted" test queries instead, we discovered important correctness issues in all these systems (see Table 2).

System	Ontology	♯Queries	Issue
Requiem CGLLR	AX	8	incomplete
Rapid	U	36	unsound
	U	18	incomplete
IQAROS	S	25	incomplete
Nyaya	S	82	incomplete
	U, UX	23	incomplete

Table 2: Statistics about unsoundness and incompleteness

All systems, except for Rapid, were reported sound for all ontologies and queries using the results in Proposition 5. Rapid was reported unsound for ontology U; since Rapid computes a UCQ rewriting for ontology U and all test queries, we could prove unsoundess and find suitable "witness" instances by again exploiting Proposition 5. Further inspection of the problematic queries revealed that unsoundness was due to a normalisation error similar to the one described in Example 3. The error was reported to the developers of Rapid and fixed in a subsequent release of the tool.

As shown in Table 2, all evaluated systems were found incomplete for some test ontologies and queries. In particular, we were able to exploit the results in Proposition 8 to detect discrepancies in the UCQ rewritings computed by each system; a more detailed inspection of the problematic queries and witness instances confirmed that discrepancies were indeed due to incompleteness issues.

Rapid's incompleteness was again due to normalisation issues similar to those described in Example 3. Incompleteness in both CGLLR and Requiem was due to an implementation bug in query subsumption and redundancy elimination optimisations. Incompleteness in IQAROS was due to implementation bugs in ordering criteria for query atoms. These errors were again reported to systems' developers and fixed based on our feedback.

Performance Evaluation

Performance was evaluated by using the systems' latest versions; in the case of CGLLR, Requiem, Rapid, and IQAROS, the latest versions include fixes for all the correctness issues we identified in our experiments.

$\mathcal{T}$	CG	Req	Rap	IQAR	Nyaya
A	65.4	1.9	1.6	0.4	662.2
AX	5,680.0	10,442.2	586.8	64.2	>5h
P5	163.0	9.4	0.2	1.1	37.5
P5X	310.0	78.7	4.8	4.0	>5h
S	0.7	1.2	0.6	0.1	0.4
U	0.3	0.4	0.2	0.04	0.4
UX	0.3	0.6	0.3	0.06	0.6
V	2.3	3.4	0.7	0.8	5.7

Table 3: Sum of rewriting times (seconds) for all test queries

For each system, we measured the time to compute a UCQ rewriting for each test query and for each ontology. Due to space limitations we cannot present results for each test; hence, for illustration purposes, Table 3 provides for each ontology and each system the total time that the system took to compute a rewriting for all test queries. We can observe that Rapid and IQAROS were significantly faster than the other systems.[9] Moreover, Nyaya could not finish rewriting the test queries for ontologies AX and P5X after 5 hours.

Finally, we have evaluated systems' redundancy elimination mechanisms by measuring the size of the computed UCQ rewritings (i.e., the number of CQs they contain). Table 4 presents, for each ontology and each system, the sum of the sizes of all (UCQ) rewritings computed for all the test queries.[10] We can observe that CGLLR presents the highest degrees of redundancy in its output, whereas Rapid and IQAROS computed the most succinct rewritings.

[4] http://www.cs.ox.ac.uk/projects/requiem/C.zip

[5] http://www.cs.ox.ac.uk/projects/requiem/home.html

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

[7] http://code.google.com/p/iqaros/

[8] http://mais.dia.uniroma3.it/Nyaya/Home.html

[9] We do not present results for ontology P1 since it was trivial (under 50 milliseconds) for all systems.

[10] See Table 1 for total number of test queries for each ontology.

$\mathcal{T}$	CG	Req	Rap	IQAR	Nyaya
A	70,350	4,645	4,133	5,445	4,073
AX	815,921	976,151	369,817	159,252	-
P5	33,363	90	90	90	90
P5X	74,552	33,292	13,599	8,269	-
S	5,148	4,493	857	835	2,288
U	2,856	1,933	489	486	1,640
UX	3,060	2,949	705	702	2,561
V	13,439	13,356	3,737	3,737	8,609

Table 4: Sum of rewriting sizes for all test queries.

Related Work and Conclusions

To the best of our knowledge all benchmarks for ontology-based systems rely on hand-crafted test queries (Pérez-Urbina, Motik, and Horrocks 2010; Guo, Pan, and Heflin 2005; Ma et al. 2006). The work that is closest to ours is the formal study of incompleteness of ontology reasoners (Stoilos, Cuenca Grau, and Horrocks 2010; Cuenca Grau and Stoilos 2011), where the authors describe techniques for synthetic query generation; however, these were limited to very lightweight DLs ($\mathcal{EL}$ and DL-Lite) under very strict acyclicity conditions, and the queries were used to evaluate incomplete materialisation-based reasoners, such as Jena, Sesame, and OWLim.

In this paper, we presented a test query generation algorithm that can be applied to Horn ontologies. We also presented correctness evaluation techniques which have already been proved useful to uncover serious implementation errors in state of the art rewriting systems. These were not revealed by existing benchmarks and hence our techniques have already proved valuable to systems' developers.

Acknowledgements

This work was supported by the EU FP7 project SEALS and by the EPSRC projects ConDOR, ExODA, and LogMap. B. Cuenca Grau is supported by a Royal Society University Research Fellowship. Giorgos Stoilos is supported by a Marie Curie FP7-Reintegration-Grants within European Union's Seventh Framework Programme (FP7/2007-2013).

References

Acciarri, A.; Calvanese, D.; Giacomo, G. D.; Lembo, D.; Lenzerini, M.; Palmieri, M.; and Rosati, R. 2005. Quonto: Querying ontologies. In *Proc. of AAAI-05*.

Baader, F.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. 2002. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge Uni. Press.

Baget, J.-F.; Leclère, M.; Mugnier, M.-L.; and Salvat, E. 2011. On rules with existential variables: Walking the decidability line. *Artificial Intelligence* 175(9–10):1620–1654.

Calì, A.; Gottlob, G.; Lukasiewicz, T.; Marnette, B.; and Pieris, A. 2010. Datalog+/-: A family of logical knowledge representation and query languages for new applications. In *Proc. of LICS*, 228–242.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning* 39(3):385–429.

Chortaras, A.; Trivela, D.; and Stamou, G. 2011. Optimized query rewriting in OWL 2 QL. In *Proc. of CADE-23*.

Cuenca Grau, B., and Stoilos, G. 2011. What to ask to an incomplete semantic web reasoner? In *Proc. of IJCAI 2011*, 2226–2231. AAAI Press.

Deutsch, A.; Nash, A.; and Remmel, J. B. 2008. The chase revisited. In *Proc. of the 27th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2008)*, 149–158.

Fagin, R.; Kolaitis, P. G.; Miller, R. J.; and Popa, L. 2005. Data exchange: semantics and query answering. *Theoretical Computer Science* 336(1):89–124.

Glimm, B.; Horrocks, I.; Lutz, C.; and Sattler, U. 2007. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of IJCAI-07*.

Gottlob, G.; Orsi, G.; and Pieris, A. 2011. Ontological queries: Rewriting and optimization. In *Proc. of ICDE 2011*.

Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics* 3(2):158–182.

Lutz, C.; Toman, D.; and Wolter, F. 2009. Conjunctive query answering in the description logic $\mathcal{EL}$ using a relational database system. In *Proc. of IJCAI-09*.

Ma, L.; Yang, Y.; Qiu, Z.; Xie, G. T.; Pan, Y.; and Liu, S. 2006. Towards a complete OWL ontology benchmark. In *Proc. of ESWC*, 125–139.

Marnette, B. 2009. Generalized schema-mappings: from termination to tractability. In *Proc. PODS*, 13–22.

Motik, B.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; Fokoue, A.; and Lutz, C. 2009. OWL 2 Web Ontology Language Profiles. *W3C Recommendation*.

Motik, B.; Shearer, R.; and Horrocks, I. 2009. Hypertableau reasoning for description logics. *J. Artificial Intelligence Research (JAIR)* 36:165–228.

Ortiz, M.; Calvanese, D.; and Eiter, T. 2006. Characterizing data complexity for conjunctive query answering in expressive description logics. In *Proc. of AAAI-06*.

Pérez-Urbina, H.; Horrocks, I.; and Motik, B. 2009. Efficient query answering for OWL 2. In *Proc. of ISWC 09*.

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

Poggi, A.; Lembo, D.; Calvanese, D.; Giacomo, G. D.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *J. Data Semantics* 10:133–173.

Rosati, R., and Almatelli, A. 2010. Improving query answering over DL-Lite ontologies. In *Proc. of KR-10*.

Sirin, E.; Parsia, B.; Grau, B. C.; Kalyanpur, A.; and Katz, Y. 2007. Pellet: A practical OWL DL reasoner. *J. Web Semantics* 5(2):51–53.

Stoilos, G.; Cuenca Grau, B.; and Horrocks, I. 2010. How incomplete is your semantic web reasoner? In *Proc. of AAAI-10*.