# Ontology-Based Data Access Using Rewriting, OWL 2 RL Systems and Repairing

Giorgos Stoilos

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

**Abstract.** In previous work it has been shown how an OWL 2 DL ontology $\mathcal{O}$ can be 'repaired' for an OWL 2 RL system $\mathsf{ans}$—that is, how we can compute a set of axioms $\mathcal{R}$ that is independent from the data and such that $\mathsf{ans}$ that is generally incomplete for $\mathcal{O}$ becomes complete for all SPARQL queries when used with $\mathcal{O} \cup \mathcal{R}$. However, the initial implementation and experiments were very preliminary and hence it is currently unclear whether the approach can be applied to large and complex ontologies. Moreover, the approach so far can only support instance queries. In the current paper we thoroughly investigate repairing as an approach to scalable (and complete) ontology-based data access. First, we present several non-trivial optimisations to the first prototype. Second, we show how (arbitrary) conjunctive queries can be supported by integrating well-known query rewriting techniques with OWL 2 RL systems via repairing. Third, we perform an extensive experimental evaluation obtaining encouraging results. In more detail, our results show that we can compute repairs even for very large real-world ontologies in a reasonable amount of time, that the performance overhead introduced by repairing is negligible in small to medium sized ontologies and noticeable but manageable in large and complex one, and that the hybrid reasoning approach can very efficiently compute the correct answers for real-world challenging scenarios.

## 1 Introduction

The use of OWL ontologies to provide a formal and semantically rich conceptualisation of the underlying data sources is becoming the basis in many modern applications [7, 10]. However, the expressive power of OWL 2 DL comes at a price of high computational complexity [12], hence, even after intense implementation work and the design of modern optimisations, fully-fledged OWL 2 DL reasoners are still not able to cope with large datasets containing billions of triples.

As a consequence, in real-world applications, developers often employ efficient and provably scalable query answering systems which usually support only the OWL 2 RL fragment of OWL 2 DL. Prominent examples include OWLim [7], Oracle's Semantic Graph [19], Apache Jena,[1] and many more. Such systems can load any OWL ontology but they will ignore all its parts that do not fall within

[1] `http://jena.apache.org/`

the fragment they support. As a result they are *incomplete*—that is, for some ontology, query and dataset they will fail to compute all certain answers.

Although scalability is very attractive, incomplete query answering may, on the one hand, not be acceptable in several critical applications like healthcare or defense and, on the other hand, improving completeness by computing as many 'missed' answers as possible without affecting performance would be beneficial for many applications. Hence, approaches for improving the completeness of incomplete reasoners have recently been investigated [20, 13, 8]. A common technique in most of these works is to use a fully-fledged OWL 2 DL reasoner to 'materialise' certain kinds of axioms which, when taken together with the input ontology and the data, will 'help' the system compute more answers than it would normally do. However, in all previous approaches there are still combinations of inputs for which the systems would miss answers even after materialisation.

Stoilos et al. [17] investigated whether it is possible to compute *all* query answers by using the materialisation approach. They introduced the notion of a *repair* of an ontology $\mathcal{O}$ for an incomplete system $\mathsf{ans}$ which, roughly speaking, is an ontology $\mathcal{R}$ such that $\mathsf{ans}$ that is generally incomplete for $\mathcal{O}$ becomes complete for *all* SPARQL queries *and* datasets when used with $\mathcal{O} \cup \mathcal{R}$. Interestingly, by recent results [4, 3] it follows that repairs always exists for Horn fragments of OWL 2 DL and in many cases even for arbitrary OWL 2 DL ontologies. Moreover, since repairs are independent from the data and the query, they only need to be computed once at a pre-processing step. Hence, repairing is a promising approach to scalable and complete query answering and ontology-based data access.

However, despite initial encouraging results there are still several open issues and questions. First, computing a repair is a computationally very expensive process and the original implementation was using arguably obsolete systems and featured no optimisations. Second, the experimental evaluation was very preliminary and it used two rather small and simple ontologies, namely LUBM and a (very small) fragment of Galen. Hence, the practicality of the approach when it comes to large and complex ontologies is unclear. Third, although instance (SPARQL) queries are highly important the approach still does not support arbitrary queries which are needed in several real-world applications.

In the current paper we extensively study repairing as an approach to scalable (and complete) query answering. First, we investigate on how to efficiently compute repairs by providing several optimisations to the first prototype. Second, we show how general queries can be supported by integrating well-known query rewriting techniques [2] with OWL 2 RL systems, hence, providing a hybrid approach to query answering. Third, we perform an extensive experimental evaluation using both synthetic and real-world benchmarks. More precisely, first, we apply our tool over a large number of well-known ontologies to see how efficiently repairs can be computed in practice. Interestingly, our results show that computing repairs is practically feasible even for large and complex ontologies mostly due to the new optimisations. Second, we investigate how much repairing affects the performance of the OWL 2 RL reasoner in practice. Our results show that in medium sized ontologies the overhead is negligible, while in very large ones it can become noticeable. Still, however, this overhead regards only a pre-processing (loading) step and, in return, after repairing the system is indistinguishable from two OWL 2 DL reasoners over a well-known benchmark (UOBM). Third, we evaluate our hybrid query answering approach obtaining encouraging results. In more detail, the system computed all correct answers over a real-world highly expressive ontology almost instantaneously. Our current repair tool supports the DL language $\mathcal{ELHI}$, hence, given the proven scalability of the used OWL 2 RL system it is safe to conclude that this is currently one of the most scalable approaches to answering arbitrary queries over an important fragment of OWL 2.

## 2 Preliminaries

We use standard notions from first-order logic, like variable, predicate, atom, constant, satisfiability, and entailment (denoted by $\models$).

**Description Logics and OWL 2**. We assume that the reader is familiar with the basics of the OWL 2 DL language[2] and its relation to Description Logics (DLs) [1]. As usual, we make a distinction between the *schema* of an ontology, called TBox $\mathcal{T}$, which consists of all class and property axioms, and the *data*, called ABox $\mathcal{A}$, which consists of all class and property assertions (we assume only simple assertions). Then, an ontology is a set of the form $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$.

Due to the high computational complexity of query answering over OWL 2 DL ontologies [12] a number of profiles have been defined. A prominent example is the OWL 2 RL language[3] for which many empirically scalable systems have been implemented and deployed in real-world applications.

**Datalog and Conjunctive Queries**. A *datalog rule* $r$ is an expression of the form $H \leftarrow B_1 \wedge \ldots \wedge B_n$ where $H$, called *head*, is a function-free atom, $\{B_1, \ldots, B_n\}$, called *body*, is a set of function-free atoms, and each variable in the head also occurs in the body. A *datalog program* $\mathcal{P}$ is a finite set of datalog rules. A *union of conjunctive queries* (UCQ) $\mathcal{Q}$ is a set of datalog rules such that their head atoms share the same predicate, called *query predicate*, which does not appear anywhere in the body. A *conjunctive query* (CQ) is a UCQ with exactly one rule. Variables that appear in the body and not the head are called *non-distinguished* variables. CQs with no non-distinguished variables form the basis of SPARQL hence. in the following, we call them *SPARQL queries*.

We often abuse notation and identify a CQ with the only rule it contains instead of a singleton set. For a query $\mathcal{Q}$ with query predicate $Q$, a tuple of constants $\vec{a}$ is an answer of $\mathcal{Q}$ w.r.t. a TBox $\mathcal{T}$ and an ABox $\mathcal{A}$ if the arity of $\vec{a}$ agrees with the arity of $Q$ and $\mathcal{T} \cup \mathcal{A} \cup \mathcal{Q} \models Q(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$ the answers to $\mathcal{Q}$ w.r.t. $\mathcal{T} \cup \mathcal{A}$.

**Ontology and Query Rewriting**. Rewriting is a prominent approach to query answering over ontologies. In such an approach the input TBox $\mathcal{T}$ (and query $\mathcal{Q}$)

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

is transformed into a new set of sentences that capture all the information that is relevant from $\mathcal{T}$ for answering any SPARQL CQ (resp. answering $\mathcal{Q}$) over an arbitrary ABox $\mathcal{A}$ [9, 2]. The typical target language for computing rewritings is datalog in an effort to exploit mature (deductive) database technologies to compute the answers over the original TBox.

**Definition 1.** *Let $\mathcal{T}$ be a TBox. A $\mathcal{T}$-*rewriting *is a datalog program $\mathsf{Rew}_D$ such that for each $\mathcal{A}$ consistent with $\mathcal{T}$ and each SPARQL CQ $\mathcal{Q}_g$ we have:*

$$\mathsf{cert}(\mathcal{Q}_g, \mathcal{T} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{Q}_g, \mathsf{Rew}_D \cup \mathcal{A})$$

*Let in addition $\mathcal{Q}$ be a CQ with query predicate $Q$. A $(\mathcal{Q}, \mathcal{T})$-*rewriting *is a set of the form $\mathsf{Rew}_D \uplus \mathsf{Rew}_Q$ with $\mathsf{Rew}_D$ a set of datalog rules not mentioning $Q$ and $\mathsf{Rew}_Q$ a UCQ with query predicate $Q$, and where for each ABox $\mathcal{A}$ consistent with $\mathcal{T}$ we have:*

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

Note that a $\mathcal{T}$-rewriting is only complete for all SPARQL queries.

*Example 1.* Consider the TBox $\mathcal{T}$ consisting of the following axioms:

$$\mathsf{PhDSt} \sqsubseteq \mathsf{GradSt} \qquad \mathsf{GradSt} \sqsubseteq \exists \mathsf{takes}.\mathsf{Course}$$
$$\exists \mathsf{takes}.\mathsf{Course} \sqsubseteq \mathsf{Student} \qquad \mathsf{Student} \sqsubseteq \mathsf{Person}$$

and consider also the CQ $\mathcal{Q} = Q(x) \leftarrow \mathsf{takes}(x, y) \wedge \mathsf{Course}(y)$.

The set $\mathsf{Rew}_1 = \{\mathcal{Q}, \mathcal{Q}_1, \mathcal{Q}_2\}$, where $\mathcal{Q}_1, \mathcal{Q}_2$ are presented next, is a $(\mathcal{Q}, \mathcal{T})$-rewriting while the set $\mathsf{Rew}_2 = \{r_1, r_2, r_3, r_4\}$ is a $\mathcal{T}$-rewriting.

$$\mathcal{Q}_1 = Q(x) \leftarrow \mathsf{GradSt}(x) \qquad \mathcal{Q}_2 = Q(x) \leftarrow \mathsf{PhDSt}(x)$$

$$r_1 = \mathsf{Person}(x) \leftarrow \mathsf{Student}(x) \qquad r_2 = \mathsf{Student}(x) \leftarrow \mathsf{takes}(x, y) \wedge \mathsf{Course}(y)$$
$$r_3 = \mathsf{Student}(x) \leftarrow \mathsf{GradSt}(x) \qquad r_4 = \mathsf{GradSt}(x) \leftarrow \mathsf{PhDSt}(x)$$

It can be seen that $\mathsf{Rew}_1$ ($\mathsf{Rew}_2$) captures all information that is relevant for answering $\mathcal{Q}$ (any SPARQL CQ) over $\mathcal{T}$. For example, $\mathcal{Q}_1$ captures the fact that according to $\mathcal{T}$ graduate students take some course, hence, in any ABox that contains an assertion of the form $\mathsf{GradSt}(a)$, $a$ is a certain answer. Similarly, $r_3$ captures the fact that graduate students are also students.

**Abstract query answering systems**. In the following, in order to abstract away from concrete systems we recall the notion of a query answering system [17].

**Definition 2.** *A* (query answering) system $\mathsf{ans}$ *is a procedure that takes as input an OWL 2 DL TBox $\mathcal{T}$, an ABox $\mathcal{A}$, and a CQ $\mathcal{Q}$ and returns a set of tuples $\mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$ that have the same arity as the query predicate of $\mathcal{Q}$. Let $\mathcal{L}$ be a fragment of OWL 2 DL and let $\mathcal{T}|_{\mathcal{L}}$ denote all $\mathcal{L}$-axioms of a TBox $\mathcal{T}$. Then, $\mathsf{ans}$ is called* complete *for $\mathcal{L}$ if for each CQ $\mathcal{Q}$ and ABox $\mathcal{A}$ we have $\mathsf{cert}(\mathcal{Q}, \mathcal{T}|_{\mathcal{L}} \cup \mathcal{A}) \subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$.*

Most OWL 2 RL reasoners known to us can be captured by the above definition. More precisely, for $\mathcal{T}|_{\mathsf{rl}}$ all the OWL 2 RL-axioms of a TBox $\mathcal{T}$, these systems essentially return $\mathsf{cert}(\mathcal{Q}, \mathcal{T}|_{\mathsf{rl}} \cup \mathcal{A})$. Note that $\mathsf{ans}$ need not be sound.

## 3 Repairing Incompleteness in a Nutshell

Stoilos et al. [17] provided the first systematic approach to improving the completeness of (incomplete) OWL 2 RL systems via ABox independent materialisation. They have introduced the notion of a *repair* of a TBox $\mathcal{T}$ for a system $\mathsf{ans}$ which, roughly speaking, is a set of axioms $\mathcal{R}$ such that i) $\mathcal{T} \models \mathcal{R}$ and ii) for each SPARQL CQ $\mathcal{Q}$ and ABox $\mathcal{A}$ we have $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{R} \cup \mathcal{A})$. For example, the set $\mathcal{R} = \{\mathsf{GradSt} \sqsubseteq \mathsf{Student}\}$ is a repair of the TBox $\mathcal{T}$ of Example 1 for an OWL 2 RL system $\mathsf{ans}$.

It was additionally shown that for systems complete for OWL 2 RL a repair exists if a $\mathcal{T}$-rewriting for the input TBox exists. Interestingly, by recent results such rewritings always exist for TBoxes expressed in Horn-$\mathcal{SHIQ}$ [4] (a fairly expressive fragment of OWL 2) and they might also exist even for arbitrary OWL 2 TBoxes [3]. Hence, repairing is a promising approach to scalable (and complete) ontology-based data access. Finally, it was shown how to minimise a repair (cf. steps 2. and 3. next). Overall the procedure of computing a repair $\mathcal{R}$ of a TBox $\mathcal{T}$ for an OWL 2 RL system $\mathsf{ans}$, denoted by Repair($\mathcal{T}$), is summarised by the following three steps:

1. Compute an initial repair $\mathcal{R}^1$ using a $\mathcal{T}$-rewriting $\mathsf{Rew}$.
2. Remove from $\mathcal{R}^1$ all axioms $\alpha$ such that $\mathcal{T}|_{\mathsf{rl}} \models \alpha$. Moreover, for each pair of distinct axioms $\alpha_1, \alpha_2$ remove $\alpha_2$ if $\mathcal{T}|_{\mathsf{rl}} \cup \{\alpha_1\} \models \alpha_2$. Let $\mathcal{R}^2$ be the resulting set of this step.
3. Finally, perform again a similar procedure like that in step 2 but this time using $\mathsf{ans}$. For example, roughly speaking, remove from $\mathcal{R}^2$ all elements $\alpha$ such that $\mathcal{T}|_{\mathsf{rl}} \models_{\mathsf{ans}} \alpha$ and remove all $\alpha_2$ such that for some $\alpha_1$ we have $\mathcal{T}|_{\mathsf{rl}} \cup \{\alpha_1\} \models_{\mathsf{ans}} \alpha_2$.[4] The result of this step is the desired repair.

## 4 Computing Repairs in Practice

In previous work it was argued that computing a repair can be done easily by using any state-of-the-art (query) rewriting system, OWL 2 DL reasoner, and OWL 2 RL system in order to implement steps 1, 2, and 3 of procedure Repair, respectively [17]. However, this is far from being true for at least two reasons that are related to the efficiency of steps 1 and 2.

Regarding step 1 the issue is that, before computing a $\mathcal{T}$-rewriting, many state-of-the-art systems would normalise an input TBox $\mathcal{T}$ by replacing complex classes with *fresh* atomic ones. For example, if $\mathcal{T}$ contains $\exists R.(E \sqcap F) \sqsubseteq A$ then this axiom would be transformed into the pair $\exists R.A_0 \sqsubseteq A$ and $E \sqcap F \sqsubseteq A_0$, where $A_0$ is a new class. Hence, the computed rewriting, call it $\mathsf{Rew}'_D$ in the following, would also mention such fresh predicates, e.g., in this case it would contain the rules $A(x) \leftarrow R(x,y) \wedge A_0(y)$ and $A_0(x) \leftarrow E(x) \wedge F(x)$ (we informally call such rewritings *normalised*). As a consequence, $\mathsf{Rew}'_D$ cannot be used as a basis for computing a repair—that is, if $\mathcal{R}$ is the output of procedure Repair($\mathcal{T}$)

[4] The reader is referred to [17] for details about how $\models_{\mathsf{ans}}$ can be checked in practice.

when computing $\mathsf{Rew}'_D$ at step 1, then we will generally have $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \not\subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{R} \cup \mathcal{A})$ for some ABox $\mathcal{A}$.

The obvious solution to the above problem is to eliminate the fresh symbols in $\mathsf{Rew}'_D$ by 'unfolding' their definitions creating new rules which contain only symbols from $\mathcal{T}$ (we informally call such rewritings *unfolded*). In the previous example by unfolding the rule containing $A_0(x)$ into the one containing $A_0(y)$ we can compute the new rewriting $\mathsf{Rew}_D$ that instead contains the rule $A(x) \leftarrow R(x, y) \wedge E(y) \wedge F(y)$. Clearly, $\mathcal{T} \models \mathsf{Rew}_D$ and hence $\mathsf{Rew}_D$ can be used to compute an initial repair. However, first, it is well known that this unfolding transformation can cause an exponential blow-up in the size of the rewriting [16, 6] (and hence of the repair) and, second, experimental evaluation has shown that it is very time consuming or even impossible to complete within a reasonable amount of time in large and complex TBoxes.

Although normalised rewritings would generally not lead to repairs of the input TBox, as shown next, they do lead to repairs of the *normalised* input TBox, which provides a way to apply repairing even on large and complex TBoxes.

**Proposition 1.** *Let $\mathcal{T}$ be an OWL 2 DL TBox, let* $\mathsf{ans}$ *be an OWL 2 RL system, let* $\mathsf{Rew}' = \mathsf{Rew}'_D \uplus \mathsf{Rew}'_Q$ *be a $\mathcal{T}$-rewriting computed by some query rewriting system, and let $\mathcal{T}'$ be the version of $\mathcal{T}$ that it used to compute* $\mathsf{Rew}'$*, i.e., let $\mathcal{T}'$ be such that for each $\mathcal{A}$ and SPARQL CQ $\mathcal{Q}$* $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \mathcal{T}' \cup \mathcal{A})$ *and* $\mathsf{cert}(\mathcal{Q}, \mathcal{T}' \cup \mathcal{A}) = \mathsf{cert}(\mathsf{Rew}'_Q, \mathsf{Rew}'_D \cup \mathcal{A})$*. Finally, let $\mathcal{R}'$ be the output of* $\textsc{Repair}(\mathcal{T})$ *when* $\mathsf{Rew}'$ *is computed at step 1. Then, for every $\mathcal{Q}$ and $\mathcal{A}$ we have* $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{T}' \cup \mathcal{R}' \cup \mathcal{A})$.

Besides computational efficiency, as shown next, normalised repairs also tend to be smaller in size.

*Example 2.* Consider the following TBox $\mathcal{T}$ and CQ $\mathcal{Q}$:

$$\mathcal{T} = \{A \sqsubseteq B \sqcap \exists R.(\{o\} \sqcup D), B \sqsubseteq C\} \qquad \mathcal{Q} = Q(x) \leftarrow C(x)$$

Since the first axiom is not in OWL 2 RL we have $\mathcal{T}|_{\mathsf{rl}} = \{B \sqsubseteq C\}$ and hence any OWL 2 RL system $\mathsf{ans}$ would be in general incomplete; e.g., for $\mathcal{A} = \{A(a)\}$ we have $\mathsf{cert}(\mathcal{Q}, \mathcal{T}|_{\mathsf{rl}} \cup \mathcal{A}) = \emptyset$ while $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) = \{a\}$. Hence, any repair $\mathcal{R}$ of $\mathcal{T}$ for $\mathsf{ans}$ must contain the axiom $A \sqsubseteq C$; then, $\mathsf{cert}(\mathcal{Q}, \mathcal{T}|_{\mathsf{rl}} \cup \mathcal{R} \cup \mathcal{A}) = \{a\}$.

However, after normalisation we have $\mathcal{T}' = \{A \sqsubseteq B, A \sqsubseteq \exists R.(\{o\} \sqcup D), B \sqsubseteq C\}$, hence also $\mathcal{T}'|_{\mathsf{lr}} = \{A \sqsubseteq B, B \sqsubseteq C\}$ and thus $\mathsf{ans}(\mathcal{Q}, \mathcal{T}'|_{\mathsf{lr}} \cup \mathcal{A}) = \{a\}$. Consequently, the empty set is a repair of $\mathcal{T}'$ for $\mathsf{ans}$. $\diamondsuit$

Unfortunately, the normalised TBox can be quadratically larger than the input TBox. Hence, reasoning over the former and the respective repair might be more time consuming compared to reasoning over the input TBox and the standard repair. Indeed, as our evaluation will show, such repairs should be used only in cases where an unfolded rewriting for a (complex) TBox cannot be computed.

Regarding the efficiency of step 2, as can be seen, this step consists of two loops (the second one of which is quadratic) over the set $\mathcal{R}^1$, in which a number of entailment checks using a fully-fledged OWL 2 DL reasoner are performed. Since the computation of $\mathcal{R}^1$ is based on a $\mathcal{T}$-rewriting $\mathsf{Rew}$, then $\mathcal{R}^1$ can be exponentially larger than $\mathcal{T}$. Hence, despite how optimised an OWL 2 DL reasoner is, the number of entailment checks in large and complex TBoxes would simply be too much for this step to behave well in practice.

Fortunately, its performance can be significantly improved by observing that most parameters in these entailment checks are fixed or rarely changing. More precisely, in both entailment checks the TBox ($\mathcal{T}|_{\mathsf{rl}}$) is always fixed and in the quadratic loop, the axiom $\alpha_1$ changes only when all entailments $\mathcal{T}|_{\mathsf{rl}} \cup \{\alpha_1\} \models \alpha_2$ for each $\alpha_2 \in \mathcal{R}^1$ have been checked. This can be exploited as follows. First, we can exhaustively apply the calculus of the OWL 2 DL reasoner over $\mathcal{T}|_{\mathsf{rl}}$ and mark the completion of the execution. Then, in the first case, we can check $\mathcal{T}|_{\mathsf{rl}} \models \alpha$ by resuming the execution from the previous point while, in the second case, the same strategy can be followed for $\mathcal{T}|_{\mathsf{rl}} \cup \{\alpha_1\}$ and each check $\mathcal{T}|_{\mathsf{rl}} \cup \{\alpha_1\} \models \alpha_2$. As we will see, this strategy leads to significant time savings.

Finally, we note that similar observations can also be made for the for-loops in step 3. However, due to the minimisations performed in step 2 we expect that the size of the repair at this point is quite small and hence this step should behave well in practice.

## 5 Supporting Queries With Non-Distinguished Variables

Despite the fact that, after repairing, the OWL 2 RL system can answer correctly all SPARQL queries, there are still certain applications where answering queries containing non-distinguished variables is of great importance. For these cases the straightforward approach would be to compute a $(\mathcal{Q}, \mathcal{T})$-rewriting $\mathsf{Rew}$ and then use a datalog engine to evaluate $\mathsf{Rew}$ over the given dataset $\mathcal{A}$. However, in many cases $\mathsf{Rew}$ can be large and complex and hence this process might not scale well in practice, requiring the integration of techniques for minimising and/or simplifying the structure of $\mathsf{Rew}$ [14, 15]. Although to a great extent successful, these techniques usually depend on the data assuming also additional conditions on them which in some cases might not hold, they so far have been designed to work only over ontologies expressed in rather inexpressive languages (e.g., OWL 2 QL), and they require manual effort to implement and integrate.

Interestingly, repairing can potentially provide the basis for a practical approach to efficiently answer arbitrary CQs. It suffices to observe that $\mathcal{R}$ together with $\mathcal{T}$ capture all ground entailments—that is, for any $(\mathcal{Q}, \mathcal{T})$-rewriting $\mathsf{Rew}_D \uplus \mathsf{Rew}_Q$, any ABox $\mathcal{A}$, and any assertion $\alpha$ such that $\mathcal{T} \cup \mathcal{A} \models \alpha$, we have $\mathsf{Rew}_D \cup \mathcal{A} \models \alpha$ if and only if $\mathcal{T} \cup \mathcal{R} \cup \mathcal{A} \models \alpha$. Hence, we have the following result.

**Proposition 2.** *Let $\mathcal{T}$ be an OWL 2 DL TBox, let $\mathcal{Q}$ be a CQ, and let* $\mathsf{ans}$ *be a query answering system complete for OWL 2 RL. Let also* $\mathsf{Rew}_D \uplus \mathsf{Rew}_Q$ *be a* $(\mathcal{Q}, \mathcal{T})$*-rewriting and let* $\mathcal{R}$ *be a repair of* $\mathcal{T}$ *for* $\mathsf{ans}$*. Then, for every ABox* $\mathcal{A}$ *we have* $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(\mathsf{Rew}_Q, \mathcal{T} \cup \mathcal{R} \cup \mathcal{A})$.

The previous proposition suggests the following approach to answering queries with non-distinguished:

1. Compute a repair $\mathcal{R}$ of $\mathcal{T}$ for $\mathsf{ans}$ using procedure Repair.
2. Load the dataset $\mathcal{A}$, the input TBox $\mathcal{T}$, and the repair $\mathcal{R}$ to $\mathsf{ans}$.
3. For a CQ $\mathcal{Q}$ with non-distinguished variables, compute a $(\mathcal{Q},\mathcal{T})$-rewriting $\mathsf{Rew}_D \uplus \mathsf{Rew}_Q$ using any rewriting system and then evaluate $\mathsf{Rew}_Q$ using $\mathsf{ans}$.

The above approach has at least three advantages: first, for a TBox $\mathcal{T}$ steps 1 and 2 need to be done *only once* as a pre-processing step; second, $\mathsf{Rew}_Q$ is usually expected to be small and simple in structure, hence, step 3 would potentially behave well in practice; and third, the approach is very easy to implement and it can easily exploit any existing and future development in query rewriting and OWL 2 RL systems without requiring to adapt or modify them.

## 6 Implementation and Evaluation

We have implemented a prototype ontology repair and query answering tool called Hydrowl.[5] The tool uses Rapid [18], a highly-optimised query rewriting system,[6] the OWL 2 DL reasoner HermiT [11], and the OWL 2 RL reasoner OWLim [7].

Regarding whether an unfolded or a normalised rewriting was used at step 1 of Repair, our system supports three modes, namely *no-normalisation*, where the rewriting is unfolded as much as possible, *lite-normalisation*, where only some parts are unfolded, and *full-normalisation*, where no unfolding occurs. Furthermore, to implement our incremental optimisation for step 2 we have modified HermiT internally to mark the completion of the application of the calculus and to be able to backtrack to such points after each entailment check.

Regarding experimental evaluation we performed three experiments which we will present next. First, we evaluated how efficiently repairs can be computed in practice by applying our tool over a large ontology corpus containing many challenging ontologies. Second, we loaded some of the repaired ontologies into OWLim to see how much loading is affected by repairing. Since most OWL 2 RL systems perform reasoning during loading then, w.r.t. SPARQL CQs, this reflects the total performance overhead introduced by repairing. Finally, we used the approach illustrated in the previous section to answer queries with non-distinguished variables over a real-world large and complex ontology.

Our test dataset contains 145 ontologies from the Gardiner corpus, a well-known library [5] that consists of many real-world ontologies containing more than 1000 axioms (we discarded all ontologies for which we either encountered a parsing error or they are expressed in OWL full) and the well-known ontologies FoodWine, UOBM, Propreo, CIDOC-CRM, nci 3.09d, Galen-doctored, and Fly; hence we have a total of 152 ontologies.

All experiments were conducted on an average speed machine (Intel© Core$^{\text{TM}}$ 2 Duo E8400 3.00GHz) with 2GB of memory assigned to the JVM.

[5] `http://www.image.ece.ntua.gr/~gstoil/hydrowl/`

[6] Since Rapid currently supports the DL $\mathcal{ELHI}$, our tool is guaranteed to repair only the $\mathcal{ELHI}$ fragment of an ontology. However, as mentioned, repairing larger fragments is theoretically possible and work towards practical algorithms is ongoing.

## 6.1 Repairing a Large Ontology Corpus

From our 152 ontologies, we managed to compute a repair using no-normalisation for 146 of them, while for the remaining 6 we had to use some of the two normalisation modes (no-normalisation either threw an out of memory exception or after 45 minutes it was still at a very early stage of step 1, hence we aborted). This also verifies in practice that there are ontologies for which completing step 1 of REPAIR is not trivial and using normalisation is a necessity.

Regarding computation time, all aforementioned 146 ontologies were processed in about 13 minutes with only four requiring more than a minute. More precisely, nci required 251 seconds (the longest time), GO 179 seconds, propreo 155 seconds, and Family 116 seconds. Moreover, only two required more than 10 seconds, namely UOBM which required 16.1 seconds and MadCows which required 10.6 seconds. All other ontologies required less than a couple of seconds and in most cases just a few milliseconds. Hence, we see that for many real-world ontologies repairs can be computed very efficiently in practice.

Regarding the size of the repairs, interestingly for 134 out of the 146 ontologies we computed an empty repair. For the remaining 12 we ran Hydrowl using all normalisation modes in order to investigate on the differences and properties of the different modes. The results are summarised in Table 1 where $|\mathcal{T}|$ denotes the number of axioms of the $\mathcal{ELHI}$ fragment of the input ontology (recall that Hydrowl currently supports $\mathcal{ELHI}$), $|\mathcal{T}'|$ the number of axioms of the normalised ontology, $t$ the computation time in seconds, $|\mathcal{R}|$ the number of axioms of the repair, the columns denoted by $\sqsubseteq$, $\sqcap$, and $\exists$, denote how many axioms of the form $A \sqsubseteq B$, $A \sqcap B \sqsubseteq C$, and $\exists R.C \sqsubseteq B$, respectively the repair has, '$d$' the maximum depth encountered in axioms of the form $\exists R.C \sqsubseteq B$, and Inv the number of inverse object properties in $\mathcal{R}$. The results for each ontology are split into three lines which correspond to no-, light-, and full-normalisation, respectively; we do not present results for full-normalisation over Koala, FoodWine, mindswappers, and Family since lite-normalisation computed an empty repair.

From the table we can observe that in general repairs are rather small and simple and usually contain axioms of the first two forms, however, only half of them contain only axioms of the form $A \sqsubseteq B$; hence, previous approaches [20, 13, 8] that mainly classify the input are indeed going to miss answers in many practical cases. The most complex repair was computed for propreo using no-normalisation which was the only ontology where a depth of 2 in axioms of the form $\exists R.C \sqsubseteq B$ was observed. Moreover, we note that normalisation usually doubles the size of the ontology which is better than a quadratic increase, however, it is noticeable. Finally, repairs computed using normalisation are usually smaller and simpler as they rarely contain axioms of the form $\exists R.C \sqsubseteq B$.

For the cases where using different modes of our tool has yielded differences in the repairs (e.g., differences with respect to size) we have performed a further analysis. More precisely, for each of the axioms of each repair, we extracted a justification—that is, a minimal subset $\mathcal{J} \subseteq \mathcal{T}$ such that $\mathcal{J} \models \alpha$, and we have manually examined them to get an insight about their differences. In the following we present our conclusions for some interesting cases.

**Table 1.** Results for the 12 ontologies with non-empty repairs.

| $\mathcal{T}$ | $\lvert\mathcal{T}\rvert$ | $\lvert\mathcal{T}'\rvert$ | $t$ | $\lvert\mathcal{R}\rvert$ | $\sqsubseteq$ | $\sqcap$ | $\exists$ | $\mathsf{Inv}$ | $d$ | $\mathcal{T}$ | $\lvert\mathcal{T}\rvert$ | $\lvert\mathcal{T}'\rvert$ | $t$ | $\lvert\mathcal{R}\rvert$ | $\sqsubseteq$ | $\sqcap$ | $\exists$ | $\mathsf{Inv}$ | $d$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | 1.7 | | | | | | | | | | 0.9 | | | | | | |
| mged-1 | 449 | 456 | 1.4 | 2 | 2 | 0 | 0 | 0 | 0 | CopyRight | 193 | 266 | 1.2 | 3 | 3 | 0 | 0 | 0 | 0 |
| | | | 0.8 | | | | | | | | | | 0.6 | | | | | | |
| | | | 0.8 | | | | | | | | | | 3.2 | 1 | 1 | 0 | 0 | 0 | 0 |
| mged-2 | 407 | 415 | 0.9 | 1 | 1 | 0 | 0 | 0 | 0 | PeoplePets | 53 | 130 | 1.3 | 19 | 16 | 0 | 3 | 0 | 1 |
| | | | 0.6 | | | | | | | | | | 0.7 | 18 | 18 | 0 | 0 | 0 | 0 |
| | | | 3.1 | 4 | 0 | 1 | 3 | 1 | 1 | | | | 10.6 | 13 | 12 | 0 | 1 | 1 | 1 |
| Travel | 48 | 68 | 0.2 | 3 | 0 | 3 | 0 | 0 | 0 | MadCows | 48 | 129 | 0.3 | 15 | 12 | 0 | 3 | 0 | 1 |
| | | | 0.2 | 2 | 2 | 0 | 0 | 0 | 0 | | | | 0.5 | 15 | 15 | 0 | 0 | 0 | 0 |
| Koala | 22 | 31 | 2.2 | 2 | 0 | 0 | 2 | 1 | 1 | FoodWine | 166 | 212 | 5.6 | 1 | 1 | 0 | 0 | 0 | 0 |
| | | | 0.1 | 0 | - | - | - | - | - | | | | 2.6 | 0 | - | - | - | - | - |
| | | | 155.1 | 15 | 0 | 0 | 15 | 18 | 2 | | | | 16.1 | 9 | 5 | 1 | 3 | 1 | 1 |
| Propreo | 450 | 715 | 4.3 | 2 | 2 | 0 | 0 | 0 | 0 | UOBM | 118 | 161 | 0.5 | 6 | 6 | 0 | 0 | 0 | 0 |
| | | | 1.4 | 2 | 2 | 0 | 0 | 0 | 0 | | | | 0.3 | 6 | 6 | 0 | 0 | 0 | 0 |
| Mindsw. | 101 | 125 | 0.5 | 1 | 0 | 0 | 1 | 1 | 2 | Family | 48 | 95 | 116.4 | 13 | 13 | 0 | 0 | 0 | 0 |
| | | | 0.2 | 0 | - | - | - | - | - | | | | 1.4 | 0 | - | - | - | - | - |

Interestingly, in FoodWine the no-normalisation mode computed a repair containing a single axiom while the two normalisation modes computed an empty repair. The additional axiom computed was $\alpha = \mathsf{WhiteNonSweetWine} \sqsubseteq \mathsf{PotableLiquid}$ and its justification consisted of the following axioms:

$$\begin{aligned}\mathsf{WhiteNonSweetWine} &\equiv \mathsf{WhiteWine} \sqcap \exists \mathsf{hasSugar}.\{dry, offDry\}\\ \mathsf{WhiteWine} &\equiv \mathsf{Wine} \sqcap \exists \mathsf{hasColor}.\{white\}\\ \mathsf{Wine} &\sqsubseteq \mathsf{PotableLiquid}\end{aligned}$$

Consequently, the reasons for the observed differences are similar to those highlighted in Example 2—that is, since concept $\{dry, offDry\}$ is outside OWL 2 RL, then the TBox $\mathcal{T}|_{\mathsf{rl}}$ will not contain the first axiom which is important for deducing $\alpha$. In contrast, in the normalised TBox $\mathcal{T}'$ the former axiom is transformed (amongst others) to $\mathsf{WhiteNonSweetWine} \sqsubseteq \mathsf{WhiteWine}$ which is in OWL 2 RL and hence for the OWL 2 RL fragment of $\mathcal{T}'$ we have $\mathcal{T}'|_{\mathsf{rl}} \models \alpha$. Consequently, in the first case the repair contains $\alpha$ while in the latter it doesn't. Similar observations can also be made for Koala, mindswapper, Family, for the 13 additional axioms computed for propreo, and for the 3 additional axioms computed for UOBM by the no-normalisation mode.

Another noteworthy case is observed in PeoplePets where the repair computed using no-normalisation contains a single axiom while the repairs in both normalisation modes many more. One extra axiom in the two normalisation modes is $\mathsf{OldLady} \sqsubseteq \mathsf{CatOwner}$ and its justification is the following:

$$\begin{aligned}\mathsf{OldLady} &\sqsubseteq \exists \mathsf{hasPet}.\mathsf{Animal} \sqcap \forall \mathsf{hasPet}.\mathsf{Cat}\\ \mathsf{CatOwner} &\equiv \exists \mathsf{hasPet}.\mathsf{Cat}\end{aligned}$$

**Table 2.** Results for the ontologies that can only be processed using normalisation.

| $\mathcal{T}$ | $\lvert\mathcal{T}\rvert$ | $\lvert\mathcal{T}'\rvert$ | mode | $t'$ | $t$ | $\lvert\mathcal{R}\rvert$ | $\sqsubseteq$ | $\sqcap$ | $\exists$ | $d$ |
|---|---|---|---|---|---|---|---|---|---|---|
| DOLCE-lt | 260 | 350 | lite | | 9.0 | 0 | - | - | - | - |
| xobjects | 264 | 1087 | lite | | 13.1 | 0 | - | - | - | - |
| Not-Galen | 5471 | 10967 | full | 1706 | 298(42) | 3015(4153) | 3015 | 0 | 0 | 0 |
| Galen | 4229 | 8559 | full | 1157 | 257(24) | 3012(3062) | 2667 | 345 | 0 | 0 |
| Galen-doc | 4229 | 8763 | full | 3427 | 1152(28) | 6051(6176) | 3743 | 1412 | 896 | 1 |
| Fly | 19845 | 24594 | full | 13758 | 2884(178) | 10361(12368) | 10361 | 0 | 0 | 0 |

Although, the first axiom contains an existential restriction and hence is outside OWL 2 RL, we concluded that when there is a pair of axioms of the form $A \sqsubseteq \exists R.C \sqcap \forall R.D$ and $\exists R.D \sqsubseteq B$, OWLim is able to deduce that $A \sqsubseteq B$. In contrast, in the normalised ontology the first axiom in the justification is split into $\mathsf{OldLady} \sqsubseteq \exists \mathsf{hasPet}.\mathsf{Animal}$ and $\mathsf{OldLady} \sqsubseteq \forall \mathsf{hasPet}.\mathsf{Animal}$, then the former is discarded and hence the interaction is not identified. Indeed, we have verified our speculations using two small tests. Similar observations also apply to the discrepancies observed in the MadCows ontology. Consequently, besides increasing the completeness of an OWL 2 RL reasoner normalisation can in some rare cases also decrease it and hence force repairs to be larger.

Finally, Table 2 presents the results for the 6 challenging ontologies, where all columns are like in Table 1 with the addition of column 'mode', which denotes which normalisation mode was used, and column '$t'$', which denotes the time to compute a repair without our optimisations for step 2. As can be seen, even for these very challenging ontologies we were able to compute a repair in a fairly reasonable amount of time given the size and complexity of each ontology. This is mostly due to normalisation and our optimisation for step 2 which as shown by contrasting columns '$t'$' and '$t$' makes a large difference in practice. Moreover, due to the size and complexity of these ontologies the computed repairs are quite large (about half the size of the input ontology), however, they are usually simple in structure (an exception being Galen-doc) and are never exponentially large.

It is also interesting to note that most of the computation time was spent in the second part of step 2 (the quadratic loop). In columns '$t$' and $|\mathcal{R}|$ in parenthesis we give the computation time and the size of the repair after executing only step 1 and the first part of step 2, discarding its second phase. As can be seen these steps are completed in a matter of seconds and the respective repairs are not much larger than the optimal one. As we will see next, using these non-optimal repairs does not seem to have a huge difference in practice.

## 6.2 Loading Under the Presence of Repairs

From our ontology dataset we selected the Fly ontology which comes with an ABox containing more than 6,000 assertions and UOBM for which there exists a data generator[7] that can be used to generate ABoxes of arbitrary size. For these

[7] `http://www.cs.ox.ac.uk/isg/tools/UOBMGenerator/`

**Table 3.** Loading times for Fly and UOBM for the various ABoxes.

| | 1 | 2 | 5 | 10 | 20 |
|---|---|---|---|---|---|
| UOBM | 4.1 | 6.8 | 16.2 | 31.9 | 73.2 |
| UOBM$\cup\mathcal{R}$ | 4.4 | 8.3 | 24.3 | 44.9 | 108.1 |
| UOBM$'\cup\mathcal{R}'$ | 5.6 | 13.0 | 40.0 | 98.9 | 276.9 |

(a) UOBM

| | $\mathcal{A}_1$ | $\mathcal{A}_2$ | $\mathcal{A}_3$ | $\mathcal{A}_4$ | $\mathcal{A}_5$ |
|---|---|---|---|---|---|
| Fly | 14.0 | 21.9 | 22.7 | 27.9 | 31.5 |
| Fly$\cup\mathcal{R}$ | 31.9 | 55.1 | 68.5 | 93.0 | 119.3 |
| Fly$\cup\mathcal{R}^-$ | 33.2 | 62.1 | 70.1 | 100.6 | 118.2 |

(b) Fly

ontologies we load the TBox with and without the repair together with ABoxes of varying size and measure the time that OWLim requires to load the data.

We used the UOBM data generator to generate ABoxes for 1, 2, 5, 10, and 20 universities. Then, we loaded them to OWLim using the original ontology and two repaired versions of UOBM, one computed using no-normalisation and one computed using lite-normalisation. Table 3(a) presents the results where UOBM$'\cup\mathcal{R}'$ denotes the normalised ontology and the repair computed using lite-normalisation. As we can see, repairing does introduce some additional overhead, however, this is relatively small (loading UOBM$\cup\mathcal{R}$ was at most 50% slower than without the repair). The penalty when using UOBM$'\cup\mathcal{R}'$ was much larger, which suggests that normalisation should be used mainly when unfolded rewritings cannot be computed. Moreover, we have tested the completeness of OWLim for the 13 test queries of UOBM.[8] When using the original ontology OWLim was found incomplete for three of them, while when we also loaded the repairs it computed the same answers as HermiT and Pellet for all of them.

Regarding Fly, we have replicated the original ABox up to 5 times.[9] Table 3(b) shows the loading time for each ABox using the original ontology and two repairs, the minimal one ($\mathcal{R}$) computed after completing all steps of Repair and a non-minimal one ($\mathcal{R}^-$) computed discarding the second phase of step 2. As can be seen, in contrast to UOBM, there is a relatively significant increase in loading time which reflects the size and complexity of Fly. However, since loading is performed only once and, as we will see in the next section, despite the high expressivity of this ontology afterwards we are able to compute all answers to user queries in a matter of milliseconds, we feel that this penalty is worth it. Moreover, interestingly, using the non-minimal instead of the minimal repair does not seem to make a large difference in practice. Hence, this suggests that one could perhaps completely dispense with the second phase of step 2 if this takes a considerable amount of time in the computation of a repair.

## 6.3 Evaluating Hybrid Query Answering

The Fly ontology comes with five real-world queries, four of which contain non-distinguished variables. As illustrated in Section 5, to compute answers for them, we first loaded Fly together with its repair and the ABox to OWLim and then

[8] The UOBM benchmark has two more test queries but computing answers for them requires reasoning over constructors which Hydrowl does not support yet.

[9] To the best of our knowledge, no better method to 'scale-up' an ABox exists.

**Table 4.** Results for Answering the Fly Queries

| $\mathcal{Q}_1$ | | | $\mathcal{Q}_2$ | | | $\mathcal{Q}_3$ | | | $\mathcal{Q}_4$ | | | $\mathcal{Q}_5$ | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| $\lvert\mathsf{R}_Q\rvert$ | $t_{\mathsf{rew}}$ | $t_{\mathsf{ans}}$ | $\lvert\mathsf{R}_Q\rvert$ | $t_{\mathsf{rew}}$ | $t_{\mathsf{ans}}$ | $\lvert\mathsf{R}_Q\rvert$ | $t_{\mathsf{rew}}$ | $t_{\mathsf{ans}}$ | $\lvert\mathsf{R}_Q\rvert$ | $t_{\mathsf{rew}}$ | $t_{\mathsf{ans}}$ | $\lvert\mathsf{R}_Q\rvert$ | $t_{\mathsf{rew}}$ | $t_{\mathsf{ans}}$ |
| 64 | 0.31 | 0.31 | 2880 | 0.90 | 1.28 | 1 | 0.00 | 0.00 | 91 | 0.07 | 0.04 | 6 | 0.05 | 0.02 |

for each query we computed a $(\mathcal{Q}, \mathcal{T})$-rewriting $\mathsf{Rew}$ using Rapid and evaluated only its UCQ part over the initialised OWLim. Table 4 presents the size of the UCQ part of the rewriting ($\mathsf{R}_Q$), the time Rapid required to compute it ($t_{\mathsf{rew}}$), and the time OWLim required to evaluate it over the loaded ontology, repair, and data ($t_{\mathsf{ans}}$).

As we can see the results are highly promising. In most cases we can compute and evaluate a rewriting almost instantaneously with the exception of query $\mathcal{Q}_2$ where, due to the large size of $\mathsf{R}_Q$, it required around two seconds; still a small number though. Furthermore, our system computed the *same* answers as the ones reported in [21] (i.e., all the correct answers) even though, interestingly, the Fly ontology is expressed in the highly expressive DL $\mathcal{SRI}$. All in all, computing a non-optimal repair, loading it into OWLim together with the original Fly TBox and ABox, loading Fly on Rapid and, finally, computing the answers for all 5 queries required a total of 233.2 seconds (computing the repair required 178 seconds, loading into OWLim and Rapid around 48.2 seconds and computing and evaluating all rewritings over OWLim around 7 seconds). In contrast, as mentioned in [21, 22], over a much faster machine than the one used here, HermiT requires several hours to compute the answers, while the approach proposed in [21, 22] requires 657 seconds to pre-process the Fly ontology and an average of 117 seconds *per* query to compute the answers.

## 7 Conclusions

In this paper we investigate on ontology repairing for OWL 2 RL reasoners as a practical approach to scalable (and complete) ontology-based data access. First, we revisit our previous implementation and propose novel optimisations for its two complex and time consuming steps, namely steps 1 and 2, in order to be able to cope with large and complex ontologies. More precisely, for step 1 we show how datalog rewritings (which can be computed efficiently by state-of-the-art rewriting systems) can be used to compute repairs, while, for step 2 we show how the internals of an OWL 2 DL reasoner can be changed in order to avoid repeating much of the necessary work. Second, we push the envelope of ontology repairing by showing how we can also support queries containing non-distinguished variables by integrating query rewriting with (repaired) OWL 2 RL systems. Our techniques have many advantages as they delegate most of the hard work to a pre-processing step (i.e., computing the repairs and loading everything to the OWL 2 RL system) that can be performed only once and leave for on-line processing either the task of simply matching the query to the data (case of SPARQL CQs) or computing the UCQ part of a rewriting and matching that over the data (case of non-distinguished variables). Moreover, our approach is easy to implement and reuses existing technology without any internal modifications (only HermiT was modified for the goal of further optimising it).

Finally, our experimental evaluation has provided with very promising results. First, we were able to compute repairs very efficiently (in a matter of milliseconds) for the vast majority of ontologies and even able to process large and complex ones in a reasonable amount of time (in less than 1 hour). Since for a fixed or rarely changing ontology computing repairs is performed mostly once as a pre-processing step we feel that this is a tolerable time. Even more, our results suggest that the most expensive step of the repair computation procedure can possibly be discarded, in which case repairs even for large ontologies can be computed in a matter of seconds. Second, our experiments also showed that loading the repair in addition to the standard input provides with an additional overhead only in very large ontologies (e.g., Fly) while in UOBM the penalty was fairly unimportant. Still, even in large ontologies, if we take into account that loading is performed mostly once and that, after repairing, the OWL 2 RL system is indistinguishable from OWL 2 DL reasoners w.r.t. SPARQL queries we feel that this extra overhead is worth paying for. Finally, with respect to queries containing non-distinguished variables, we were able to compute all the correct answers to the queries of the Fly ontology almost instantaneously although Fly is expressed in the highly expressive DL $\mathcal{SRI}$. To the best of our knowledge, no other system can match these times.

Regarding directions for future work we plan to extend our implementation to support more expressive fragments of OWL like Horn-$\mathcal{SHIQ}$ [4] or even non-Horn fragments of OWL [3] and conduct further experiments. This is far from trivial as, to the best of our knowledge, algorithms for computing rewritings in such languages either do not exist or have not shown to scale over large and complex ontologies.

**Acknowledgements** The work was funded by a Marie Curie FP7-Reintegration-Grant within EU's 7th Framework Programme (2007-2013) under REA grant agreement 303914 and by project Europeana Fashion within EU's Competitiveness and Innovation Framework Programme under grant agreement 297167. We would also like to thank Yujiao Zhou for providing fly_anatomy and its queries.

## References

1. F. Baader, D. McGuinness, D. Nardi, and P.F. Patel-Schneider. *The Description Logic Handbook: Theory, implementation and applications.* Cambridge University Press, 2002.
2. Diego Calvanese, Giuseppe De Giacomo, Domenico Lembo, Maurizio Lenzerini, and Riccardo Rosati. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Autom. Reas.*, 39(3):385–429, 2007.
3. Bernardo Cuenca Grau, Boris Motik, Giorgos Stoilos, and Ian Horrocks. Computing datalog rewritings beyond horn ontologies. In *Proceedings of the Twenty-Third International Joint Conference on Artificial Intelligence (IJCAI 2013)*, 2013.

4. Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao. Query rewriting for Horn-$\mathcal{SHIQ}$ plus rules. In *Proc. of AAAI*, 2012.
5. Tom Gardiner, Dmitry Tsarkov, and Ian Horrocks. Framework for an automated comparison of description logic reasoners. In *Proc. of the 5th International Semantic Web Conference (ISWC 2006)*, pages 654–667, 2006.
6. Stanislav Kikot, Roman Kontchakov, Vladimir V. Podolskii, and Michael Zakharyaschev. Long rewritings, short rewritings. In *Proc. of DL 2012*, 2012.
7. 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 *Workshop on Semantic Data Management (SemData)*, 2010.
8. G. Meditskos and N. Bassiliades. Combining a DL reasoner and a rule engine for improving entailment-based OWL reasoning. In *ISWC 08*, pages 277–292, 2008.
9. Boris Motik. Description Logics and Disjunctive Datalog—More Than just a Fleeting Resemblance? In *Proc. of M4M-4*, volume 194, pages 246–265, 2005.
10. 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, 2012.
11. Boris Motik, Rob Shearer, and Ian Horrocks. Hypertableau Reasoning for Description Logics. *Journal of Artificial Intelligence Research*, 36:165–228, 2009.
12. 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.
13. Zhengxiang Pan, Xingjian Zhang, and J. Heflin. DLDB2: A scalable multi-perspective semantic web repository. In *Proc. International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '08)*, pages 489–495, 2008.
14. Mariano Rodriguez-Muro and Diego Calvanese. Dependencies: Making ontology based data access work. In *Proc. of AMW 2011*, 2011.
15. Riccardo Rosati. Prexto: Query rewriting under extensional constraints in dl-lite. In *9th Extended Semantic Web Conference (ESWC 2012)*, pages 360–374, 2012.
16. Riccardo Rosati and Alessandro Almatelli. Improving query answering over DL-Lite ontologies. In *Proc. of KR 2010*, 2010.
17. Giorgos Stoilos, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks. Repairing ontologies for incomplete reasoners. In *Proceedings of the 10th International Semantic Web Conference (ISWC-11), Bonn, Germany*, pages 681–696, 2011.
18. Depoina 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.
19. Zhe Wu, George Eadon, Souripriya Das, Eugene Inseok Chong, Vladimir Kolovski, Melliyal Annamalai, and Jagannathan Srinivasan. Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In *Proc. of ICDE*, pages 1239–1248. IEEE, 2008.
20. Jian Zhou, Li Ma, Qiaoling Liu, Lei Zhang, Yong Yu, and Yue Pan. Minerva: A scalable OWL ontology storage and inference system. In *Proceedings of the First Asian Semantic Web Conference (ASWC 2006)*, pages 429–443, 2006.
21. Yujiao Zhou, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, and Jay Banerjee. Making the most of your triple store: Query answering in owl 2 using an rl reasoner. In *Proc, WWW 2013*, pages 1569–1580, 2013.
22. Yujiao Zhou, Yavor Nenov, Bernardo Cuenca Grau, and Ian Horrocks. Complete query answering over horn ontologies using a triple store. In *Proc. of the 12th International Semantic Web Conference (ISWC)*. Springer LNCS, 2013.