OBDA Using $\mathcal{RL}$ Reasoners and Repairing*

Giorgos Stoilos

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

1 Introduction

Rewriting the input TBox $\mathcal{T}$ (and query $\mathcal{Q}$) into a datalog program, called $\mathcal{T}$-rewriting ($(\mathcal{Q}, \mathcal{T})$-rewriting), is a prominent approach to ontology-based data access [1]. It was used as early as the KAON2 system [5] and it currently consists of (perhaps) the standard approach to answering queries over ontologies expressed in the languages DL-Lite [2, 12] and $\mathcal{ELHI}$ [8, 16].

Apart from computing a rewriting, a problem that has been proven very difficult is how to subsequently (effectively) evaluate it over the data. In many cases, due to its size and/or complexity, additional techniques need to be devised [11, 9, 10]. Unfortunately, all of them have so far been applied only on DL-Lite while for more expressive languages, apart from the work in [3], to the best of our knowledge, no other integrated query answering system has been reported or evaluated with large and complex ontologies.

An interesting observation is that $\mathcal{T}$-rewritings are usually of a particular form—that is, they can be translated back into an $\mathcal{RL}$ TBox [14]. Hence, scalable $\mathcal{RL}$ system, like OWLim, can be used to evaluate them over the data. More precisely, it was shown how, given a $\mathcal{SROIQ}$ TBox $\mathcal{T}$ and an $\mathcal{RL}$ system $\text{ans}$, a rewriting of $\mathcal{T}$ can be used to compute a set of axioms $\mathcal{R}$ (called repair) such that for every CQ $\mathcal{Q}$ with only distinguished variables (i.e., SPARQL CQs) and every ABox $\mathcal{A}$ we have $\text{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \subseteq \text{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{R} \cup \mathcal{A})$. That is, after repairing, $\text{ans}$ is indistinguishable from a $\mathcal{SROIQ}$ reasoner w.r.t. SPARQL CQs.

Although, the experimental results in [14] provided with encouraging results, these were quite preliminary. First, the implementation used an arguably obsolete system (Requiem) and had no optimisations. Second, the evaluation was based on LUBM (an arguably trivial TBox) and a small fragment of Galen. Hence, it was unclear whether repairing can be applied to large and complex TBoxes. Third, the approach could only handle SPARQL CQs.

In the current paper we attempt to extensively study repairing as an approach to OBDA over expressive TBoxes. First, we propose several optimisations and refinements to the first prototype in order to handle large and complex TBoxes. Second, we show how arbitrary queries can be supported. Third, we perform an extensive experimental evaluation which showed that we can handle very large and complex TBoxes (e.g., Galen, GO, Fly) and answer arbitrary CQs over one of them (Fly) within milliseconds. This is an extended abstract of paper [13].

* Funded by an EU FP7 Marie Curie Career Reintegration Grant (No.: 303914).

1 In OWL and Semantic Web jargon that is an OWL 2 RL TBox [4].
2 Computing Repairs Efficiently

Let $\mathcal{T}$ be a TBox and let $\text{ans}$ be an $\mathcal{RL}$ system. Roughly speaking, a repair $\mathcal{R}$ of $\text{ans}$ for $\mathcal{T}$ is computed by first computing a $\mathcal{T}$-rewriting ($\text{Rew}$) for $\mathcal{T}$, then removing from $\text{Rew}$ the parts that $\text{ans}$ can already handle, and finally, minimising the resulting set. For example, if $\mathcal{T} = \{ A \sqcup \exists R, \exists R \sqsubseteq B \}$, then $\text{Rew} = \{ A(x) \rightarrow B(x), R(x, y) \rightarrow B(x) \}$ is a $\mathcal{T}$-rewriting of $\mathcal{T}$, while $\mathcal{R} = \{ A \sqsubseteq B \}$ is the desirable repair for $\text{ans}$. We call $\text{Repair}(\mathcal{T})$ the overall procedure.

Compared to [14] we have performed two important refinements. First, according to [14] a repair $\mathcal{R}$ can only be computed by $\mathcal{T}$-rewritings $\text{Rew}$ such that $\mathcal{T} \models \text{Rew}$. This is quite restrictive as it prohibits the use of many efficient rewriting algorithms which normalise the input TBox $\mathcal{T}$ into $\mathcal{T}'$ by introducing fresh symbols. To be able to use such approaches to compute repairs we notice that, for $\mathcal{T}'$ the normalised TBox used internally by the rewriting system to compute $\text{Rew}$, we have $\text{cert}(Q, \mathcal{T} \cup A) \subseteq \text{ans}(Q, \mathcal{T}' \cup \text{Repair}(\mathcal{T}) \cup A)$. Second, the minimisations steps applied by $\text{Repair}$ are based on expensive FOR-loops over the initial $\mathcal{T}$-rewriting in which HermiT is invoked. Despite how optimised a $\mathcal{SROIQ}$ reasoner is, in large ontologies, these entailment checks could simply be too many. To improve the behaviour we have intervened in the internals of HermiT to implement a form of incremental entailment checking. First, we exhaustively apply the calculus over $\mathcal{T}$ to construct an initial model and we mark the completion of the execution. Then, to check $\mathcal{T} \models \alpha$ we resume the execution from the previous point and after completion we backtrack to the marked point.

3 Supporting non-SPARQL Queries

Let $Q$ be an arbitrary CQ with query predicate $Q$, let $\mathcal{T}$ be a TBox, and let $\text{Rew}_D \sqcup \text{Rew}_Q$ be a $(Q, \mathcal{T})$-rewriting ($\text{Rew}_D$ is a datalog program not mentioning $Q$ and $\text{Rew}_Q$ a UCQ that mentions $Q$). Since $\text{Rew}_D$ does not mention $Q$ it captures only ground entailments of $\mathcal{T}$ over some $A$. But, after repairing, $\text{ans}$ is complete w.r.t. all ground entailments for any $A$; hence $\text{cert}(Q, \mathcal{T} \cup A) \subseteq \text{ans}(\text{Rew}_Q, \mathcal{T} \cup \text{Repair}(\mathcal{T}) \cup A)$. The following summarises our approach:

1. Compute a repair $\mathcal{R}$ of $\mathcal{T}$ for $\text{ans}$ using procedure $\text{Repair}$.
2. Load the dataset $A$, the input TBox $\mathcal{T}$, and the repair $\mathcal{R}$ to $\text{ans}$.
3. For a CQ $Q$, if $Q$ is SPARQL then evaluated it over $\text{ans}$; otherwise compute a $(Q, \mathcal{T})$-rewriting $\text{Rew}_D \sqcup \text{Rew}_Q$ and evaluate $\text{Rew}_Q$ over $\text{ans}$.

Note that most steps, i.e., steps 1 and 2, can be done only once as a pre-processing (changes in $A$ can be handled incrementally).

4 Implementation and Evaluation

We have implemented a prototype ontology repair and query answering tool called Hydrowl. The tool uses Rapid [15] to compute $\mathcal{T}$-rewritings and HermiT [7] and OWLIM [6] to minimise it into the final repair.

\[\text{http://www.image.ece.ntua.gr/~gstoil/hydrowl/}\]
Table 1. $|\mathcal{T}|$ ($|\mathcal{R}|$): number of axioms of the input TBox (repair), $t$: time in seconds.

|   | $|\mathcal{T}|$ | $|\mathcal{R}|$ | $t$ | $|\mathcal{T}|$ | $|\mathcal{R}|$ | $t$ |
|---|----------------|----------------|-----|----------------|----------------|-----|
| Not-Galen | 5471 | 3015(4153) | 298(42) | Galen-doc | 4229 | 6051(6176) | 1152(28) |
| Fly | 19845 | 10361(12368) | 2884(178) | Galen | 4229 | 3012(3062) | 257(24) |

Table 2. Loading times for Fly and UOBM for the various ABoxes.

<table>
<thead>
<tr>
<th>Universities</th>
<th>A $\times$ A $\times$ A $\times$ A $\times$ A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fly</td>
<td>14.0 21.9 22.7 27.9 31.5</td>
</tr>
<tr>
<td>Fly $\cup$ R</td>
<td>31.9 55.1 68.5 93.0 119.3</td>
</tr>
<tr>
<td>Fly $\cup$ (R$^{-}$)</td>
<td>33.2 62.1 70.1 100.6 118.2</td>
</tr>
</tbody>
</table>

(a) UOBM (b) Fly

Using Hydrowl we managed to compute repairs for 151 out of the 152 ontologies of our dataset, which contained some large and highly complex ones. In the vast majority of cases a repair could be computed in less than a second, only a handful required up to a few minutes, and only the very large ones several minutes; Table 1 presents results for the latter. Despite their size and complexity we see that we can compute repairs for them in a reasonable amount of time (recall this is usually done only once). Actually, if we discard a very expensive minimisation step, then we can compute some (non-minimal) repair very efficiently while its size is not considerably larger than the minimal one (see Table 1 numbers in parenthesis). The system in [14] could not handle any of these ontologies.

Next, Table 2 presents loading experiments to OWLim using UOBM (1 to 20 universities) and Fly (ABox multiplied up to 5 times). As can be seen, the overhead introduced by repairing is significantly noticeable only in Fly, mostly due to the size of the computed repair ($\mathcal{R}$), however, recall that loading is usually performed once. In Fly we have also loaded the non-minimal repair ($\mathcal{R}^{-}$) and as it turns out there is no significant difference compared to the minimal one.

Table 3. Results for Answering the Fly Queries

<table>
<thead>
<tr>
<th>$Q_1$</th>
<th>$Q_2$</th>
<th>$Q_3$</th>
<th>$Q_4$</th>
<th>$Q_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rew$Q$</td>
<td>Rew$Q$</td>
<td>Rew$Q$</td>
<td>Rew$Q$</td>
<td>Rew$Q$</td>
</tr>
<tr>
<td>$t_{rew}$</td>
<td>$t_{ans}$</td>
<td>$t_{rew}$</td>
<td>$t_{ans}$</td>
<td>$t_{rew}$</td>
</tr>
<tr>
<td>64</td>
<td>0.31</td>
<td>0.31</td>
<td>2880</td>
<td>0.90</td>
</tr>
</tbody>
</table>

The Fly ontology comes with 4 non-SPARQL queries. Table 3 presents the results of our technique from Section 3. As we can see in most cases we were able to compute and evaluate a rewriting almost instantaneously. This greatly outperforms previous approaches on Fly [17, 18] which require several seconds per query. The good behaviour of our approach can be attributed to the fact that most hard work is pushed to a pre-processing step while Rew$Q$, the only thing computed on-line, is usually small and of simple structure.
References

6. Atanas Kiryakov, Barry Bischoa, Damyan Ongyanoff, 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.