Repairing Incomplete Reasoners

Giorgos Stoilos and Bernardo Cuenca Grau

Oxford University Computing Laboratory
Wolfson Building, Parks Road, OX1 3QD, Oxford

Abstract. The pressing need for scalable query answering has motivated the development of many *incomplete* ontology-based reasoners. Improving the completeness of such systems without sacrificing their favourable performance would be beneficial to many applications. In this paper, we address the following problem: given a query q, a TBox $\mathcal{T}$ and an incomplete reasoner ans (i.e., one that accepts q and $\mathcal{T}$ as valid inputs but fails to return all query answers to q w.r.t. $\mathcal{T}$ and some dataset), we aim at computing a (hopefully small) TBox $\mathcal{R}$ (a *repair*) which logically follows from $\mathcal{T}$ and such that ans is *provably complete* w.r.t. q and $\mathcal{T} \cup \mathcal{R}$, regardless of the input data. We identify conditions on q, $\mathcal{T}$ and ans that are sufficient to ensure the existence of such repair and present a practical repair generation algorithm. Our experiments suggest that repairs of small size can be computed for well-known ontologies and reasoners.

1 Introduction

In Semantic Web applications, a description logic TBox is frequently used for describing the meaning of data stored in various sources. A query language (based on conjunctive queries) is then used for data access [13]. Unfortunately, when using an expressive ontology language, computing query answers can be costly, and in a Semantic Web setting, datasets may be extremely large.

Motivated by the need for scalable query answering, there has been a growing interest in the development of *incomplete* ontology-based reasoning systems, which are not guaranteed to compute all query answers for some combinations of queries, TBoxes and datasets accepted as valid inputs. Many such systems (e.g., Jena, OWLim, and Oracle's Semantic Store) are based on rule technologies; others are based on approximate reasoning techniques [7, 11, 1].

Incomplete query answers, however, may not be acceptable in some cases, and even if some incompleteness is acceptable, computing as many answers as possible *without affecting performance* would be beneficial to many applications. Not surprisingly, many techniques for improving the completeness of (incomplete) reasoning systems have been proposed in recent years. A widely use approach relies on the materialisation of certain kinds of TBox consequences, prior to accessing the data. TBox materialisation is indeed a convenient and low-cost solution, as it does not rely on modifying the internals of the reasoning system at hand: it can either be included by reasoning systems' implementors as a pre-processing step (e.g., DLEJena [3], PelletDB, TrOWL [11], Minerva [10]

and DLDB [6]), or it can be performed by application designers, who have little knowledge of and/or control over the reasoner. This approach, however, exhibits several limitations:

1. TBox materialisation is done blindfold, without taking into account neither the capabilities of the incomplete reasoner under consideration, nor the precise sources of its incompleteness for the application at hand.
2. In order to avoid a blowup in the size of the TBox, materialisation involves only simple atomic subsumption relations, and relevant entailments describing more complex dependencies are typically ignored.
3. It is often difficult, or even infeasible, to estimate the extent to which materialisation has improved the system's completeness for the given application at hand, especially if the data is very large, unknown, or frequently changing.

In this paper, we present a novel approach to TBox materialisation that attempts to address many of these limitations. Given a TBox $\mathcal{T}$ expressed in a language $\mathcal{L}$, a conjunctive query (CQ) q, and a reasoning system ans satisfying certain properties known in advance (e.g., soundness), we aim at computing a (hopefully small) set of TBox axioms $\mathcal{R}$, called a $(q, \mathcal{T})$-*repair* for ans, such that $\mathcal{R}$ logically follows from $\mathcal{T}$ and the system is guaranteed to compute all answers to q w.r.t. $\mathcal{T} \cup \mathcal{R}$ (and hence also w.r.t. $\mathcal{T}$), regardless of the input data.

Our TBox materialisation techniques are "guided" by both the input query and TBox and the capabilities of the reasoner at hand, thus limiting the number of materialised consequences. Furthermore, our approach provides a *provable completeness guarantee*: after extending $\mathcal{T}$ with a $(q, \mathcal{T})$-repair, the incomplete reasoner at hand is *indistinguishable* from a complete one w.r.t. q and $\mathcal{T}$, and regardless of the data (which is often unknown or frequently changing). Finally, if a $(q, \mathcal{T})$-repair does not exist for the given reasoner, we also provide means for quantifying the extent to which the reasoner's completeness improved upon materialisation of a given set of TBox consequences.

Our main contributions can be summarised as follows:

- We formalise the notion of a repair for a given CQ, TBox and reasoner.
- We identify sufficient conditions for the existence of a repair. Our conditions establish a bridge between the framework of completeness evaluation [16, 15] and the notion of interpolation in the context of DLs [8, 14, 9].
- We then present a practical algorithm that is guaranteed to compute a repair under certain assumptions. Our algorithm extends well-known query rewriting techniques for DLs [12, 2] with means for computing suitable *interpolants* for each query in the rewriting.
- Although the assumptions required by our algorithm may seem somewhat strict, we provide empirical evidence that repairs can be computed for well-known ontologies and incomplete reasoners. Furthermore, the size of such repairs is surprisingly small and preliminary evaluation has shown that their impact on performance is negligible. Finally, for reasoner which are complete only for a very weak description logic, we show that our techniques can provide "partial repairs", whose impact can be quantified.

Although our main focus is on repairing rule-based reasoners, we feel that our techniques are also highly relevant to recent DL approximation frameworks [11, 1], and provide new insights into the process of approximation.

Proofs for all lemmas and theorems can be found online.[1]

2 Preliminaries

We use DLs that do not allow for nominals and we also assume that only atomic assertions are allowed in ABoxes.

Let $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{I}$ be countable, pairwise disjoint sets of *atomic concepts*, *atomic roles*, and *individuals*. An $\mathcal{ELHI}$*-role* is either an atomic role P or its *inverse* P^-. The set of $\mathcal{ELHI}$-concepts is defined inductively by the following grammar, where $A \in \mathbf{C}$, R is an $\mathcal{ELHI}$-role, and $C_{(i)}$ are $\mathcal{ELHI}$-concepts:

$$C := \top \mid \bot \mid A \mid C_1 \sqcap C_2 \mid \exists R.C$$

An $\mathcal{ELHI}$-TBox $\mathcal{T}$ is a finite set of GCIs $C_1 \sqsubseteq C_2$ with C_i $\mathcal{ELHI}$-concepts and RIAs $R_1 \sqsubseteq R_2$ with R_i $\mathcal{ELHI}$-roles. An ABox $\mathcal{A}$ is a finite set of assertions $A(a)$ or $P(a, b)$, for $A \in \mathbf{C}$, $P \in \mathbf{R}$, and $a, b \in \mathbf{I}$. An $\mathcal{ELHI}$-ontology $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ consists of an $\mathcal{ELHI}$-TBox $\mathcal{T}$ and an ABox $\mathcal{A}$.

Moreover, the DLP fragment of $\mathcal{ELHI}$ [5] (DLP-$\mathcal{ELHI}$) is obtained from $\mathcal{ELHI}$ by disallowing concepts of the form $\exists R.C$ on the right-hand side of GCIs.

A *conjunctive query* (CQ) q is an expression of the form

$$q(x_1, \ldots, x_n) \leftarrow \alpha_1, \ldots, \alpha_m$$

where each α_i is a concept or role atom of the form $A(t)$ or $R(t, t')$ (for t, t' function-free terms and A, R atomic) and each x_j is a *distinguished* variable occurring in some α_i. The remaining variables are called undistinguished. We use the standard notion of a certain answer to q w.r.t. $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$, and we denote with $\mathsf{cert}(q, \mathcal{O})$ the set of all certain answers to q w.r.t. $\mathcal{O}$. Given q_1, q_2 with the same distinguished variables $\vec{x}$ we say that q_1 *subsumes* q_2 w.r.t. $\mathcal{T}$, written $q_2 \sqsubseteq_{\mathcal{T}} q_1$, if the FO-entailment $\mathcal{T} \models \forall \vec{x}.(B_{q_2} \rightarrow B_{q_1})$ holds, with B_{q_i} the formula obtained from the conjunction of all body atoms in q_i by existentially quantifying over all undistinguished variables.

Finally, a *UCQ rewriting* u for a CQ q and TBox $\mathcal{T}$ is a union of conjunctive queries (a set of CQs with the same distinguished variables) that satisfies the following properties for each ABox $\mathcal{A}$ such that $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ is consistent [2]: $\mathsf{cert}(q', \mathcal{A}) \subseteq \mathsf{cert}(q, \mathcal{O})$ for each $q' \in u$, and $\mathsf{cert}(q, \mathcal{O}) \subseteq \bigcup_{q' \in u} \mathsf{cert}(q', \mathcal{A})$.

3 Completeness Repairs

We start by recalling from [16, 15] the notions of a CQ answering algorithm, and of completeness for a query q and TBox $\mathcal{T}$.

[1] https://rapidshare.com/files/460900188/main.pdf

Definition 1. *A* CQ answering algorithm ans *for a DL* $\mathcal{L}$ *is a procedure that, for each* $\mathcal{L}$*-ontology* $\mathcal{O}$ *and CQ* q *computes in a finite number of steps a set* $\mathsf{ans}(q, \mathcal{O})$ *of tuples of constants. It is* sound *if* $\mathsf{ans}(q, \mathcal{O}) \subseteq \mathsf{cert}(q, \mathcal{O})$ *for each* $\mathcal{O}$ *and* q*. It is* complete *if* $\mathsf{cert}(q, \mathcal{O}) \subseteq \mathsf{ans}(q, \mathcal{O})$ *for each* $\mathcal{O}$ *and* q*. It is* monotonic *if* $\mathsf{ans}(q, \mathcal{O}) \subseteq \mathsf{ans}(q, \mathcal{O}')$ *for each* $\mathcal{O}$*,* $\mathcal{O}'$ *and* q *with* $\mathcal{O} \subseteq \mathcal{O}'$*. It is* invariant under renamings *if, for each* $q, \mathcal{O} = \mathcal{T} \cup \mathcal{A}$*,* $\mathcal{O}' = \mathcal{T} \cup \mathcal{A}'$ *and isomorphism* ν *between* $\mathcal{A}$ *and* $\mathcal{A}'$ *we have* $\vec{a} \in \mathsf{ans}(q, \mathcal{O})$ *if and only if* $\nu(\vec{a}) \in \mathsf{ans}(q, \mathcal{O}')$*. It is* well-behaved *if it is sound, monotonic and invariant under renamings. We denote with* $\mathcal{C}^{\mathcal{L}}$ *the class of all well-behaved CQ answering algorithms for* $\mathcal{L}$*.*

Finally, we say that ans *is* $(q, \mathcal{T})$-complete *for a query* q *and TBox* $\mathcal{T}$ *if for each ABox* $\mathcal{A}$ *s.t.* $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ *is consistent, we have that* $\mathsf{cert}(q, \mathcal{O}) \subseteq \mathsf{ans}(q, \mathcal{O})$*.*

This general definition allow us to abstract from the specifics of implemented systems. All incomplete reasoning algorithms known to us are well-behaved: they are sound, query answers can only grow if we add axioms to the ontology, and they do not depend on trivial renamings of ABox individuals. Furthermore, a $(q, \mathcal{T})$-complete algorithm, even if incomplete in general, behaves exactly like a complete one w.r.t. the given query q and TBox $\mathcal{T}$, regardless of the data. Consider, as a running example, the following $\mathcal{ELHI}$-TBox $\mathcal{T}_0$ and query q_0:

$$\begin{aligned} \mathcal{T}_0 \;=\; & \{\exists \mathsf{takes}.\mathsf{Course} \sqsubseteq \mathsf{Student}, \mathsf{GradCo} \sqsubseteq \mathsf{Course}, \\ & \mathsf{GradSt} \sqsubseteq \exists \mathsf{takes}.\mathsf{GradCo}, \mathsf{PhDSt} \sqsubseteq \mathsf{GradSt},\ \mathsf{Student} \sqcap \mathsf{Course} \sqsubseteq \bot\} \\ q_0(x) \leftarrow\; & \mathsf{Student}(x) \end{aligned}$$

Consider an algorithm ans_0 that first translates DLP-$\mathcal{ELHI}$ GCIs in $\mathcal{T}_0$ into a datalog program using standard transformations (and discarding the remaining GCIs), then saturates the input ABox w.r.t. the program, and finally answers q_0 w.r.t. the saturated ABox. Clearly, ans_0 is not $(q_0, \mathcal{T}_0)$-complete: it returns the empty set for $\mathcal{A} = \{\mathsf{GradSt}(a)\}$, whereas $\mathsf{cert}(q_0, \mathcal{T}_0 \cup \mathcal{A}) = \{a\}$. Computing the missing answer requires axiom $\mathsf{GradSt} \sqsubseteq \exists \mathsf{takes}.\mathsf{GradCo}$, which is discarded. Note, however, that one could dispense with the discarded axiom and still recover the missing answer by extending $\mathcal{T}_0$ with the following DLP-$\mathcal{ELHI}$ TBox $\mathcal{R}_0$:

$$\mathcal{R}_0 = \{\mathsf{GradSt} \sqsubseteq \mathsf{Student}\} \tag{1}$$

Since $\mathcal{T}_0 \models \mathcal{R}_0$, extending $\mathcal{T}_0$ with $\mathcal{R}_0$ does not change the meaning of $\mathcal{T}_0$. The behaviour of ans_0 w.r.t. $\mathcal{T}_0 \cup \mathcal{R}_0$, however, is now different because ans_0 translates $\mathcal{R}_0$ into a datalog clause and hence $\mathsf{ans}_0(q_0, \mathcal{T}_0 \cup \mathcal{R}_0 \cup \mathcal{A}) = \{a\}$.

Note also that adding $\mathcal{R}_0$ allows us to recover missing answers w.r.t. many other ABoxes different from $\mathcal{A}$. For example, given $\mathcal{A}' = \{\mathsf{PhDSt}(b)\}$ and $\mathcal{T}_0 \cup \mathcal{A}'$, we have that b is a certain answer to q_0 that is computed by ans_0 only after extending $\mathcal{T}_0$ with $\mathcal{R}_0$.

Our example suggests that given q, $\mathcal{T}$ and an algorithm ans that is incomplete for q and $\mathcal{T}$, it may be possible to recover all missing answers to q (w.r.t. all possible ABoxes) by "repairing" $\mathcal{T}$ for ans—that is, by materialising a (hopefully small) number of $\mathcal{T}$-consequences that ans can process.

Definition 2. *Let $\mathcal{L}$ be a DL, q a CQ, $\mathcal{T}$ an $\mathcal{L}$-TBox, $\mathcal{A}$ an ABox, and* ans *a CQ answering algorithm for $\mathcal{L}$. A $(q, \mathcal{T}, \mathcal{A})$-*repair $\mathcal{R}$ *for* ans *is an $\mathcal{L}$-TBox such that*

1. $\mathcal{T} \not\models \mathcal{R}$*; and*
2. $\mathsf{cert}(q, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(q, \mathcal{T} \cup \mathcal{R} \cup \mathcal{A})$.

*Given a set $\mathbf{A}$ of ABoxes, we say that $\mathcal{R}$ is a $(q, \mathcal{T}, \mathbf{A})$-*repair *for* ans *if it is a $(q, \mathcal{T}, \mathcal{A})$-repair for* ans *for each $\mathcal{A} \in \mathbf{A}$. Finally, we say that $\mathcal{R}$ is a $(q, \mathcal{T})$-*repair *for* ans *if it is a $(q, \mathcal{T}, \mathcal{A})$-repair for* ans *for every ABox $\mathcal{A}$ s.t. $\mathcal{T} \cup \mathcal{A}$ is consistent.*

Unfortunately, deciding, on the one hand, whether ans is $(q, \mathcal{T})$-complete and, on the other hand, whether $\mathcal{R}$ is a $(q, \mathcal{T})$-repair for ans may involve computing query answers w.r.t. an unlimited number of ABoxes. In [16, 15], however, it was shown that under certain conditions on q and $\mathcal{T}$ it is possible to decide $(q, \mathcal{T})$-completeness by computing query answers only w.r.t. a finite (and hopefully small) number of ABoxes. Such finite set of ABoxes is called a *testing base*.

Definition 3. *Let $\mathcal{L}$ be a description logic, let $\mathcal{C}$ be a class of CQ answering algorithms for $\mathcal{L}$, let $\mathcal{T}$ be an $\mathcal{L}$-TBox, and let q be a CQ. A $(q, \mathcal{T})$-*testing base $\mathbf{B}$ *for $\mathcal{C}$ is a finite set of ABoxes $\mathcal{A}$ such that the following property holds for each algorithm* ans *in $\mathcal{C}$: if $\mathsf{cert}(q, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(q, \mathcal{T} \cup \mathcal{A})$ for each $\mathcal{A} \in \mathbf{B}$, then $\mathsf{cert}(q, \mathcal{T} \cup \mathcal{A}') \subseteq \mathsf{ans}(q, \mathcal{T} \cup \mathcal{A}')$ for each ABox $\mathcal{A}'$ consistent with $\mathcal{T}$.*

In our previous work [16, 15] we have shown how to compute a $(q, \mathcal{T})$-testing base (if one exists) for the class $\mathcal{C}^{\mathcal{L}}$ of well-behaved algorithms.

Consider our running example. The set of ABoxes $\mathbf{B}_0 = \{\mathcal{A}_1, \ldots, \mathcal{A}_5\}$ defined as follows is a $(q_0, \mathcal{T}_0)$-testing base for the class of all well-behaved algorithms:

$$\mathcal{A}_1 = \{\mathsf{Student}(a)\},\ \mathcal{A}_2 = \{\mathsf{takes}(a, b), \mathsf{Course}(b)\},$$
$$\mathcal{A}_3 = \{\mathsf{GradSt}(a)\},\ \mathcal{A}_4 = \{\mathsf{takes}(a, b), \mathsf{GradCo}(b)\},\ \mathcal{A}_5 = \{\mathsf{PhDSt}(a)\}$$

Intuitively, in order to compute a $(q, \mathcal{T})$-repair, we only need to consider the (finitely many) ABoxes in a $(q, \mathcal{T})$-testing base instead of all (infinitely many) possible ABoxes.

Theorem 1. *Let $\mathcal{L}$ be a DL, q a CQ, $\mathcal{T}$ an $\mathcal{L}$-TBox, and $\mathbf{B}$ a $(q, \mathcal{T})$-testing base for $\mathcal{C}^{\mathcal{L}}$. The following property holds for each* ans $\in \mathcal{C}^{\mathcal{L}}$*: If $\mathcal{R}$ is a $(q, \mathcal{T}, \mathbf{B})$-repair for* ans*, then $\mathcal{R}$ is also a $(q, \mathcal{T})$-repair for* ans*.*

In our running example, clearly, ans_0 only fails to compute certain answers w.r.t. $\mathcal{A}_3$ and $\mathcal{A}_5$. Furthermore, $\mathcal{R}_0$ is a $(q_0, \mathcal{T}_0, \mathbf{B}_0)$-repair for ans_0. But then, Theorem 1 ensures that $\mathcal{R}_0$ is also a $(q_0, \mathcal{T}_0)$-repair. Thus, by simply adding $\mathcal{R}_0$ to $\mathcal{T}_0$, we can guarantee that ans_0 will compute all certain answers, regardless of the data.

4 Computing Repairs

Theorem 1 provides a sufficient condition for the existence of a $(q, \mathcal{T})$-repair for a well-behaved algorithm ans. To apply the theorem, however, we first need to compute a $(q, \mathcal{T})$-testing base $\mathbf{B}$ and then a $(q, \mathcal{T}, \mathbf{B})$-repair $\mathcal{R}$ for ans.

As shown in [15, 16], a $(q, \mathcal{T})$-testing base $\mathbf{B}$ for $\mathcal{C}^{\mathcal{L}}$ can always be computed from a UCQ rewriting u for q and $\mathcal{T}$. UCQ rewritings are guaranteed to exist if $\mathcal{T}$ is expressed in the DL underpinning the OWL 2 QL profile and they may also exist even if $\mathcal{T}$ is expressed in the DLs underpinning the OWL 2 EL profile; this is often the case in many real-world ontologies. Hence, in this paper we will restrict ourselves to TBoxes and queries for which a UCQ rewriting exists.[2]

Given a $(q, \mathcal{T})$-testing base $\mathbf{B}$, however, the existence of a $(q, \mathcal{T}, \mathbf{B})$-repair may also depend on the capabilities of the algorithm under consideration. For instance, in the case of our running example, no $(q_0, \mathcal{T}_0, \mathbf{B}_0)$-repair exists for an algorithm that ignores the TBox and simply matches the query to the data (even though such algorithm is well-behaved).

Our techniques for computing repairs rely on a particular form of *interpolation* [8, 14]. We next make the connection between repairs and interpolation precise, and describe an algorithm for computing interpolants.

4.1 Completeness Repairs and Interpolation

As already discussed, algorithm ans_0 from our running example misses answers w.r.t. $\mathcal{A}_3, \mathcal{A}_5 \in \mathbf{B}_0$, and $\mathcal{R}_0$ from (1) is a $(q_0, \mathcal{T}_0, \mathbf{B}_0)$-repair for ans_0. Furthermore, $\mathbf{B}_0$ can be obtained by "instantiating" queries from the following UCQ rewriting $u = \{q_1, \ldots, q_5\}$ for q_0 and $\mathcal{T}_0$:

$$q_1(x) \leftarrow \mathsf{Student}(x),\ q_2(x) \leftarrow \mathsf{takes}(x, y), \mathsf{Course}(y),$$
$$q_3(x) \leftarrow \mathsf{GradSt}(x),\ \ q_4(x) \leftarrow \mathsf{takes}(x, y), \mathsf{GradCo}(y),\ q_5(x) \leftarrow \mathsf{PhDSt}(x)$$

In particular, $\mathcal{A}_3 = \{\mathsf{GradSt}(a)\}$ is obtained by instantiating q_3. We then say that q_3 is "relevant" to ans_0. Formal definitions of instantiation and relevance are given next.

Definition 4. *Let q be a CQ, B_q the body atoms in q, and π a mapping from all variables of q to individuals. The following ABox is an* instantiation *of q:*

$$\mathcal{A}^q_\pi := \{A(\pi(x)) \mid A(x) \in B_q\} \cup \{R(\pi(x), \pi(y)) \mid R(x, y) \in B_q\}$$

Definition 5. *Let u be a UCQ rewriting for q and an $\mathcal{L}$-TBox $\mathcal{T}$, let ans be well-behaved and let $\mathbf{B}$ be a $(q, \mathcal{T})$-testing base. We say that $q_i \in u$ is relevant to ans if an instantiation $\mathcal{A}_i$ of q_i exists s.t. $\mathcal{A}_i \in \mathbf{B}$ and $\mathsf{cert}(q, \mathcal{T} \cup \mathcal{A}_i) \not\subseteq \mathsf{ans}(q, \mathcal{T} \cup \mathcal{A}_i)$.*

Note also that q_3 is subsumed by q_0 w.r.t. $\mathcal{T}_0$. We can then identify $\mathcal{R}_0$ as an *interpolant* for the subsumption $q_3 \sqsubseteq_{\mathcal{T}_0} q_0$ since $\mathcal{T}_0 \models \mathcal{R}_0$, $q_3 \sqsubseteq_{\mathcal{R}_0} q_0$ and $\mathcal{R}_0$ only mentions symbols occurring in either q_0 or q_3.

Definition 6. *Let $\mathcal{T}$ be an $\mathcal{L}$-TBox and let q, q' be CQs such that $q' \sqsubseteq_{\mathcal{T}} q$. A TBox $\mathfrak{I}$ expressed in fragment $\mathcal{L}'$ of $\mathcal{L}$ and using only symbols occurring in q or q' is an* $\mathcal{L}'$-interpolant *for $q' \sqsubseteq_{\mathcal{T}} q$ if $\mathcal{T} \models \mathfrak{I}$ and $q' \sqsubseteq_{\mathfrak{I}} q$.*

[2] The notion of a testing base can be extended, and such (extended) testing bases can be computed from *Datalog rewritings*. The study of such extensions is ongoing work.

The following theorem establishes which are the relevant interpolants for computing a repair. Furthermore, if each such interpolants can be expressed in a weak enough language, for which ans is provably complete, we can easily obtain a repair from the union of such interpolants.

Theorem 2. *Let u be a UCQ rewriting for q and an $\mathcal{L}$-TBox $\mathcal{T}$. Let* ans $\in \mathcal{C}^{\mathcal{L}}$ *be an algorithm that is complete for a fragment $\mathcal{L}'$ of $\mathcal{L}$. Let $q_1, \ldots, q_n$ be those queries in u relevant to* ans*. Finally, for each $1 \leq i \leq n$ let $\Im_i$ be an $\mathcal{L}'$-interpolant for $q_i \sqsubseteq_{\mathcal{T}} q$. Then, $\Im = \bigcup_{1 \leq i \leq n} \Im_i$ is a $(q, \mathcal{T})$-repair for* ans*.*

4.2 Computing Interpolants

Assumptions First, we restrict ourselves to TBoxes expressed in $\mathcal{ELHI}$. Systems such as REQUIEM can compute a UCQ rewriting w.r.t. an $\mathcal{ELHI}$-TBox, provided that one exists [12].

Second, we observe that most state-of-the-art incomplete systems are based on deductive database and rule technologies. Given $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ and q as input, the ABox $\mathcal{A}$ is deterministically saturated with new assertions using axioms in $\mathcal{T}$, and then q is answered directly w.r.t. the saturated ABox. Furthermore existing systems rarely extend the input ABox with "fresh" individuals and hence cannot handle existential quantification on the right-hand-side of TBox axioms. In fact, many such systems are complete for DLs of the DLP family. Thus, we will consider DLP-$\mathcal{ELHI}$ as our target language for computing interpolants.

In this setting, we also need to restrict the kinds of queries we are considering to guarantee the existence of the relevant interpolants. It is rather straightforward to find $\mathcal{ELHI}$-TBoxes and queries q for which a UCQ rewriting does exist, but the DLP-$\mathcal{ELHI}$ interpolants required by Theorem 2 do not. For example, let $\mathcal{T} = \{A \sqsubseteq \exists R.C\}$ and consider the following queries

$$q(x) \leftarrow R(x, y), C(y) \quad q'(x) \leftarrow A(x)$$

Clearly, $\{q, q'\}$ is a UCQ rewriting for q and $\mathcal{T}$. There is, however, no DLP-$\mathcal{ELHI}$ interpolant for $q' \sqsubseteq_{\mathcal{T}} q$ since such interpolant would need to contain existential quantification on the right-hand-side of GCIs.

Consequently, from now onwards we focus on queries with only distinguished variables. This restriction is not unreasonable in practice since the standard language for querying ontologies on the Web (SPARQL) treats undistinguished variables as if they were distinguished [4].

The Algorithm Algorithm 1 computes a UCQ rewriting u (if one exists) for an $\mathcal{ELHI}$-TBox $\mathcal{T}$ and a query q with no undistinguished variables. Furthermore, for each $q' \in u$, it computes a DLP-$\mathcal{ELHI}$ interpolant for $q' \sqsubseteq_{\mathcal{T}} q$.

Our algorithm extends the rewriting algorithm implemented in the REQUIEM system [12], which first transforms $\mathcal{T}$ and q into a set of clauses (Lines 1, 2 in Algorithm 1) and then uses a resolution calculus to compute u.

Algorithm 1 Extended Resolution-based algorithm

Algorithm: extendedUCQRewriting$(q, \mathcal{T})$
Input: $\mathcal{T} \in \mathcal{ELHI}$ and q with no undistinguished vars.

1: $\mathcal{T} :=$ clausify$(\mathcal{T})$
2: $q :=$ clausify(q)
3: $\sigma(q) := \emptyset$
4: **for all** $\alpha \in \mathcal{T}$ **do** $\sigma(\alpha) := \{\alpha\}$
5: $u := \mathcal{T} \cup \{q\}$
6: **repeat**
7: Pick clauses C_1, C_2 from u with $C_1 \neq C_2$
8: $C :=$ resolve(C_1, C_2)
9: $u := u \cup \{C\}$
10: $\mathcal{R}_C := \sigma(C_1) \cup \sigma(C_2)$
11: **repeat**
12: Pick clauses D_1, D_2 from $\mathcal{R}_C$ with $D_1 \neq D_2$
13: $\mathcal{R}_C := \mathcal{R}_C \cup \{$resolve$(D_1, D_2)\}$
14: **until** $\mathcal{R}_C$ is *saturated*
15: $\sigma(C) :=$ prune$(\mathcal{R}_C)$
16: **until** u is *saturated*
17: $(u, \sigma) :=$ ff(u, σ)
18: $(u, \sigma) :=$ unfold(u, σ)
19: **if** u is not a UCQ, **return** failure
20: **return** (u, σ)

Interpolants are tracked down during resolution by means of a mapping σ from clauses to sets of clauses. Initially, σ maps q to the empty set (since $q \sqsubseteq_\emptyset q$), and each clause from $\mathcal{T}$ to itself (Lines 3, 4). The rules in the resolution-based calculus from [12] are exhaustively applied in Lines 6–16, and new clauses are generated in Lines 7–9 exactly as in [12]. Then, for each new clause C produced as a resolvent of C_1 and C_2 our algorithm computes its corresponding interpolant $\sigma(C)$. This is done by first saturating $\sigma(C_1) \cup \sigma(C_2)$ (Lines 11–14) and then removing all clauses mentioning a symbol that is neither in C nor in q (Line 15). After u is saturated, the algorithm removes all clauses that contain function symbols (Line 16). Then, at this point, both rewriting and interpolants are given as sets of function-free clauses, so the procedure unfold transforms each $C \in u$ into a datalog rule and "rolls-up" $\sigma(C)$ into a DLP-$\mathcal{ELHI}$ GCI. Finally, the algorithm rejects the input if u is not a UCQ (e.g., a datalog program) and returns both the UCQ rewriting and the interpolant for each query otherwise.

Lemma 1. *If u computed by Algorithm 1 is a UCQ, then u is a UCQ rewriting for q and $\mathcal{T}$. Furthermore, for each $q' \in u$, $\sigma(q')$ is a DLP-$\mathcal{ELHI}$ interpolant for $q' \sqsubseteq_\mathcal{T} q$.*

Algorithm 2 computes a repair by considering only those queries and ABoxes for which ans fails. Correctness follows from Theorem 2 and Lemma 1.

Theorem 3. *Algorithm 2 computes a $(q, \mathcal{T})$-repair for an algorithm* ans *that is complete for DLP-$\mathcal{ELHI}$.*

Algorithm 2 Computing a Repair

Algorithm: computeRepair(ans, $q, \mathcal{T}$)
Input: $\mathcal{T}, q$ as in Algorithm 1, ans well-behaved.

```
1: (u, σ) := extendedUCQRewriting(q, T)
2: B := computeTestingBase(u)
3: Repair := ∅
4: for all A ∈ B do
5:     if cert(q, T ∪ A) ⊈ ans(q, T ∪ A) then
6:         q' := query in u that generated A
7:         Repair := Repair ∪ σ(q')
8:     end if
9: end for
10: return Repair
```

If ans is not complete for DLP-$\mathcal{ELHI}$ (e.g., if it is an RDFS-reasoner), then the TBox $\mathcal{R}$ computed by Algorithm 2 may not be a repair; however, adding $\mathcal{R}$ may still have the desired effect of "mitigating" the reasoner's incompleteness.

5 Evaluation

We have developed a prototype tool based on Algorithms 1 and 2. Our implementation uses REQUIEM [12] and the system described in [16, 15] for computing $(q, \mathcal{T})$-testing bases and relevant queries. Additionally, our tool implements optimisations to eliminate redundancy in the computed repairs.

We have evaluated our techniques first using LUBM's TBox and its 14 test queries (all of which contain only distinguished variables) and then a version of the GALEN TBox together with 4 sample queries for which a UCQ rewriting can be computed using REQUIEM. In each case, we have evaluated the following systems: Sesame 2.3-prl,[3] OWLim,[4] Jena v2.6.3[5] and DLEJena.[6] Our experiments consisted of the following steps:

1. For each TBox $\mathcal{T}$ and test query q we computed a $(q, \mathcal{T})$-testing base and measured the proportion of certain answers that each system is able to compute w.r.t. the ABoxes in the $(q, \mathcal{T})$-testing base (δ_{ini}) [16].
2. For each system and each q for which it was found incomplete (i.e., $\delta_{ini} < 1$), we computed a repair $\mathcal{R}$ (which might be only "partial" for some systems).
3. For each case in Step 2 we have extended $\mathcal{T}$ with the corresponding $\mathcal{R}$, computed a $(q, \mathcal{T} \cup \mathcal{R})$-testing base and obtained a new completeness degree value δ_{fin}.
4. To evaluate the effects that adding TBox axioms had on systems' performance in the LUBM scenario, we used the LUBM performance evaluation

[3] http://www.openrdf.org/
[4] http://www.ontotext.com/owlim/
[5] http://jena.sourceforge.net/
[6] http://lpis.csd.auth.gr/systems/DLEJena/

LUBM													
	Sesame										OWLim/Jena		
	Q2	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q12	Q13	Q6	Q8	Q10
$\sharp u_{rel}$	1	4	4	166	34	27	1	166	2	4	1	4	1
$\sharp\mathcal{R}$	0	0	4	1	1	1	0	1	0	4	1	1	1
δ_{ini}	.75	.68	0	.003	.04	.04	0	.001	.25	.2	.99	.98	.99
δ_{fin}	.75	.68	.75	.004	.05	.05	0	.002	.25	.2	1	1	1

GALEN												
	OWLim				Jena				DLEJena			
	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4	Q1	Q2	Q3	Q4
$\sharp u_{rel}$	36	72	72	12	24	48	48	6	36	72	72	6
$\sharp\mathcal{R}$	2	2	2	2	2	2	2	1	2	2	2	1
δ_{ini}	.84	.83	.84	.77	.94	.94	.94	.92	.84	.83	.84	.84
δ_{fin}	1	1	1	1	1	1	1	1	1	1	1	1

Table 1. Experiments for LUBM & GALEN

tool [6] and measured the loading and query answering times using the original and the repaired TBox.

Our results for LUBM are summarised in the upper part of Table 1, where $\sharp u_{rel}$ represents the number of relevant queries in the UCQ rewriting (see Definition 5), and $\sharp\mathcal{R}$ the number of axioms in the computed repair. The only system not included in the table is DLEJena, which was already complete for all queries (w.r.t. the original TBox). OWLim and Jena were found incomplete for three queries and Sesame for 10 of them. As shown in the table, we were able to repair both OWLim and Jena for all queries. This was a reasonable result, since these systems are considered to be complete for DLP-$\mathcal{ELHI}$. Furthermore, all repairs consisted of the same, unique axiom. In contrast, no query could be fully repaired for Sesame. A slight increase in the completeness degree can be observed in 4 queries and a more significant one only in $Q5$. Again, this is unsurprising, since Sesame is essentially an RDFS reasoner. No measurable performance changes were observed for any system after repair, which suggests that completeness can be improved (at least in this scenario) without affecting scalability. Finally, repairs were computed in times ranging between a second and 3 minutes.

Results for GALEN are given in the lower part of the table. As shown in the table, we were able to fully repair OWLim, Jena and DLEJena, and repairs contained at most two axioms in each case (a relatively small number compared to the number of queries in u_{rel}). All repairs could be computed in less than a minute. Finally, Sesame hasn't been included in the table because no improvement could be measured for any query.

6 Conclusions

In this paper, we have studied the problem of *repairing* an incomplete reasoning systems given a query and a TBox. An important feature of our repairs is that

they provide *data independent completeness guarantees*: once an incomplete system has been repaired, we can ensure that its output answers will be the same as those of a complete one for the given q and $\mathcal{T}$, regardless of how large and complex the dataset is. Furthermore, our preliminary experiments suggest that repairs can indeed be very small and their effect on systems' performance may even be negligible in some applications. Our results will allow application designers to use highly scalable reasoning systems in their application with the guarantee that they will behave as if they were complete, thus bringing together "the best of both worlds". The extension of our techniques to more expressive DLs and a more extensive evaluation are ongoing work.

Acknowledgments Research supported by project SEALS (FP7-ICT-238975). B. Cuenca Grau is supported by a Royal Society University Research Fellowship.

References

1. Botoeva, E., Calvanese, D., Rodriguez-Muro, M.: Expressive approximations in *DL-Lite* ontologies. In: AIMSA. pp. 21–31 (2010)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated Reasoning 39(3), 385–429 (2007)
3. G. Meditskos and N. Bassiliades: Combining a DL reasoner and a rule engine for improving entailment-based OWL reasoning. In: Proc. ISWC. pp. 277–292 (2008)
4. Glimm, B., Krötzsch, M.: SPARQL beyond subgraph matching. In: Proc. of ISWC 2010. pp. 241–256 (2010)
5. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proc. of WWW. pp. 48–57 (2003)
6. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2), 158–182 (2005)
7. Hitzler, P., Vrandecic, D.: Resolution-based approximate reasoning for OWL DL. In: Proc. of ISWC 2005). pp. 383–397 (2005)
8. Konev, B., Walther, D., Wolter, F.: Forgetting and uniform interpolation in large-scale description logic terminologies. In: Proc. of IJCAI 09. pp. 830–835 (2009)
9. Lutz, C., Wolter, F.: Deciding inseparability and conservative extensions in the description logic $\mathcal{EL}$. Journal of Symbolic Computation 45, 194–228 (2010)
10. Ma, L., Yang, Y., Qiu, Z., Xie, G.T., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Proc. of ESWC 2006. pp. 125–139 (2006)
11. Pan, J.Z., Thomas, E.: Approximating OWL-DL Ontologies. In: Proc. of AAAI-07. pp. 1434–1439 (2007)
12. Pérez-Urbina, H., Motik, B., Horrocks, I.: Tractable query answering and rewriting under description logic constraints. Journal of Applied Logic 8(2), 186–209 (2010)
13. Poggi, A., Lembo, D., Calvanese, D., Giacomo, G.D., Lenzerini, M., Rosati, R.: Linking data to ontologies. Journal of Data Semantics X, 133–173 (2008)
14. Seylan, I., Franconi, E., de Bruijn, J.: Effective query rewriting with ontologies over dboxes. In: Proc. of IJCAI 09. pp. 923–925 (2009)
15. Stoilos, G., Cuenca Grau, B., Horrocks, I.: Completeness guarantees for incomplete reasoners. In: Proc. of ISWC 2010. LNCS, Springer (2010)
16. Stoilos, G., Cuenca Grau, B., Horrocks, I.: How incomplete is your semantic web reasoner? In: Proc. of AAAI-10. pp. 1431–1436 (2010)