Repairing Ontologies for Incomplete Reasoners

Giorgos Stoilos, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, OX1 3QD, Oxford

Abstract. The need for scalable query answering often forces Semantic Web applications to use *incomplete* OWL 2 reasoners, which in some cases fail to derive all answers to a query. This is clearly undesirable, and in some applications may even be unacceptable. To address this problem, we investigate the problem of 'repairing' an ontology $\mathcal{T}$—that is, computing an ontology $\mathcal{R}$ such that a reasoner that is incomplete for $\mathcal{T}$ becomes complete when used with $\mathcal{T} \cup \mathcal{R}$. We identify conditions on $\mathcal{T}$ and the reasoner that make this possible, present a practical algorithm for computing $\mathcal{R}$, and present a preliminary evaluation which shows that, in some realistic cases, repairs are feasible to compute, reasonable in size, and do not significantly affect reasoner performance.

1 Introduction

Answering SPARQL queries over RDF data sets structured using an OWL 2 ontology provides the basis for a large number of Semantic Web applications. Such data sets can, however, be extremely large, and reasoning with OWL 2 DL ontologies is known to be of high computational complexity. As a consequence, complete reasoners—that is, reasoners such as Pellet, HermiT, and RACER that are capable (modulo bugs) of correctly computing all answers to all queries for all ontologies and datasets—often fail to deliver the required level of scalability. Application developers thus often use scalable but *incomplete* reasoners—that is, reasoners that, for *some* query, ontology, and dataset, fail to compute all answers to the query. Examples of such incomplete reasoners include state of the art RDF management systems, such as Jena [8], OWLim [6], DLE-Jena [9], and Oracle's Semantic Store [17], which typically provide completeness guarantees only for ontologies expressed in the OWL 2 RL [11] profile of OWL 2 DL.

The lack of a completeness guarantee may be unacceptable for applications in areas such as healthcare and defence, where missing answers may have serious consequences. Furthermore, even if an application can tolerate some level of incompleteness, it is desirable to provide the highest level of completeness that is compatible with the required scalability. Hence, techniques for improving the completeness of incomplete reasoners have recently been investigated. A common approach is to *materialise* certain kinds of ontology consequences before computing query answers. Such a solution does not require modifying the internals of the reasoner since the relevant consequences can be added as ontology axioms in a preprocessing step. In fact, systems such as DLE-Jena [9],

PelletDB,[1] TrOWL [12], Minerva [7], and DLDB [4] internally use a complete OWL 2 DL reasoner to transparently materialise certain axioms. Furthermore, materialisation is used in approximation frameworks [12, 2], where an OWL 2 DL ontology is projected into OWL 2 QL to allow for scalable reasoning.

Existing materialisation approaches, however, exhibit several important limitations. First, materialisation is commonly performed without taking into account the capabilities of the incomplete reasoner and may thus introduce redundant axioms. Second, to avoid a blowup in the ontology size, typically only subsumptions between (named) classes are materialised. Third, the extent to which materialisation improves a reasoner's completeness is often unclear, particularly if the data set is large, frequently changing, or unknown in advance.

In this paper, we present a novel approach to materialisation that addresses these limitations. Given an OWL 2 DL ontology $\mathcal{T}$ and a reasoner complete for OWL 2 RL, we show how to compute a *repair* $\mathcal{R}$ of $\mathcal{T}$ for the given reasoner. Intuitively, $\mathcal{R}$ is a set of OWL 2 RL consequences of $\mathcal{T}$ that, if added to $\mathcal{T}$, allow the reasoner to become *complete for* $\mathcal{T}$—that is, by using $\mathcal{T} \cup \mathcal{R}$ as input, the reasoner can correctly answer all queries w.r.t. $\mathcal{T}$ for all data sets. We focus on achieving completeness w.r.t. *ground certain answers* (i.e., answers obtained by matching query variables to named individuals). This is consistent with the semantics of SPARQL, and it allows us to ensure the existence of a repair whenever $\mathcal{T}$ can be rewritten into an OWL 2 RL ontology. Our technique is 'guided' by both the input ontology and the reasoner, which limits the size of $\mathcal{R}$ and ensures that adding $\mathcal{R}$ to $\mathcal{T}$ has minimal impact on the reasoner's scalability. Towards this goal, we proceed as follows.

In Section 3, similarly to our previous work [16, 15], we devise a way of abstracting concrete reasoners using a notion of a *reasoning algorithm*, and we formalise the notion of an ontology repair for a reasoning algorithm.

In Section 4 we present a practical, two-step technique for computing a repair of an OWL 2 DL ontology $\mathcal{T}$ for a reasoner complete for OWL 2 RL. We first rewrite $\mathcal{T}$ into an OWL 2 RL ontology $\mathcal{T}'$ that is entailed by $\mathcal{T}$ and that preserves all ground answers to arbitrary queries over $\mathcal{T}$, regardless of the data. Based on this rewriting, we subject the incomplete reasoner to a series of tests, whose results identify the subset of the rewriting that constitutes a repair.

In Section 5 we demonstrate empirically that repairs can be computed in practice for well-known ontologies and reasoners. Our experiments show that the size of repairs is typically quite small, and that extending the original ontology with a repair typically has a negligible impact on reasoner performance.

2 Preliminaries

In this paper we use the standard notions of *constants*, *variables*, (function-free) *atoms*, *sentences*, *substitutions*, *satisfiability*, *unsatisfiability*, and *entailment* (written $\models$) from first-order logic. An application of a substitution σ to a term,

[1] `http://clarkparsia.com/files/pdf/pelletdb-whitepaper.pdf`

atom, or formula α is written as $\sigma(\alpha)$. The *falsum* symbol (i.e., the symbol that is false in all interpretations) is written as $\bot$. A *datalog rule* r is an expression of the form $B_1 \wedge \ldots \wedge B_n \rightarrow H$ where H is either $\bot$ or an atom, each B_i is an atom, and each variable occurring in H occurs in some B_i as well. The *body* of r is the set $\mathsf{body}(r) = \{B_1, \ldots, B_n\}$, and the *head* of r is $\mathsf{head}(r) = H$. Both head and body atoms can contain the *equality predicate* $\approx$, and head atoms can also contain the *inequality predicate* $\not\approx$. A datalog rule is interpreted as a universally-quantified first-order implication. It is well known that checking whether a first-order theory entails a datalog rule can be realised as follows.

Proposition 1. *Let $\mathcal{F}$ be a set of first-order sentences and let r be a datalog rule such that $\mathsf{body}(r) = \{B_1, \ldots, B_n\}$ and $\mathsf{head}(r) = H$. Then, for each substitution σ mapping the variables of r to distinct constants not occurring in $\mathcal{F}$ or r, we have $\mathcal{F} \models r$ if and only if $\mathcal{F} \cup \{\sigma(B_1), \ldots, \sigma(B_n)\} \models \sigma(H)$.*

2.1 OWL 2 DL and OWL 2 RL

We assume the reader to be familiar with the OWL 2 DL ontology language [10]. For succinctness, we use the Description Logics (DL) notation to write down OWL 2 DL axioms; please refer to [1] for an overview of the relationship between DLs and OWL. As is common in the literature, we partition an OWL 2 DL ontology into a *TBox* (i.e., a finite set of axioms describing the classes and properties in a domain of discourse) and an *ABox* (i.e., a finite set of facts). For simplicity, we assume that all ABox assertions refer to classes and properties only (i.e., that they do not contain complex class and property expressions); an ABox is thus allowed to contain class and property assertions, equalities, and inequalities, all of which can involve named and/or unnamed individuals.

OWL 2 RL [11] is a prominent profile of OWL 2 DL. Each OWL 2 RL ontology can be translated into an equivalent datalog program using (a straightforward extension of) the transformation presented in [3]. This close connection with datalog makes OWL 2 RL a popular implementation target since OWL 2 RL reasoners can be implemented by extending RDF triple stores with deductive features. For simplicity, in this paper we assume that each OWL 2 RL axiom α can be translated into a *single* datalog rule $\pi(\alpha)$; this can be ensured by transforming axioms using de Morgan identities to eliminate disjunctions and conjunctions in subclass and superclass positions, respectively.

Example 1. Consider the following OWL 2 DL ontology that describes the organisation of a typical university.

$$\exists \mathsf{take}.\mathsf{Co} \sqsubseteq \mathsf{Student} \quad \rightsquigarrow \quad \mathsf{take}(x, y) \wedge \mathsf{Co}(y) \rightarrow \mathsf{Student}(x) \tag{1}$$

$$\mathsf{GradCo} \sqsubseteq \mathsf{Co} \quad \rightsquigarrow \quad \mathsf{GradCo}(x) \rightarrow \mathsf{Co}(x) \tag{2}$$

$$\mathsf{PhDSt} \sqsubseteq \mathsf{GradSt} \quad \rightsquigarrow \quad \mathsf{PhDSt}(x) \rightarrow \mathsf{GradSt}(x) \tag{3}$$

$$\mathsf{Student} \sqcap \mathsf{Co} \sqsubseteq \bot \quad \rightsquigarrow \quad \mathsf{Student}(x) \wedge \mathsf{Co}(x) \rightarrow \bot \tag{4}$$

$$\exists \mathsf{teach}.\top \sqsubseteq \mathsf{Employee} \quad \rightsquigarrow \quad \mathsf{teach}(x, y) \rightarrow \mathsf{Employee}(x) \tag{5}$$

$$\mathsf{GradSt} \sqsubseteq \exists \mathsf{take}.\mathsf{GradCo} \tag{6}$$

$$\mathsf{ResAsst} \sqcap \mathsf{PhDSt} \sqsubseteq \exists \mathsf{teach}.\mathsf{LabPrac} \tag{7}$$

According to the definition of OWL 2 RL [11], axioms (1)–(5) are OWL 2 RL axioms, and so each axiom can be transformed into an equivalent datalog rule shown on the righthand side. In contrast, axioms (6) and (7) contain an existential quantifier (someValuesFrom in OWL 2 jargon) in the superclass position, so they cannot be translated into an equivalent datalog rule. The OWL 2 RL profile therefore disallows axioms such as (6) and (7). ♢

2.2 Queries

A *union of conjunctive queries* (UCQ) $\mathcal{Q}$ with a *query predicate* Q is a datalog program in which each rule contains Q in the head but not in the body. We assume that query predicates do not occur in TBoxes and ABoxes.

Let $\mathcal{Q}$ be a UCQ with query predicate Q; let $\mathcal{F}$ be a set of first-order sentences; let $\mathcal{A}$ be an ABox; let G be a class not occurring in $\mathcal{F}$, $\mathcal{A}$, and $\mathcal{Q}$; let $\mathcal{A}_G$ be the ABox containing the class assertion $G(a)$ for each individual a occurring in $\mathcal{A}$; and let $\mathcal{Q}_G$ be the UCQ obtained from $\mathcal{Q}$ by adding to the body of each rule $r \in \mathcal{Q}$ the atom $G(x)$ for each variable x occurring in r. A tuple of constants $\vec{a}$ is a *certain answer* to $\mathcal{Q}$ w.r.t. $\mathcal{F}$ and $\mathcal{A}$ if the arity of $\vec{a}$ agrees with the arity of Q and $\mathcal{T} \cup \mathcal{A} \cup \mathcal{Q} \models Q(\vec{a})$. The set of all certain answers of $\mathcal{Q}$ w.r.t. $\mathcal{F}$ and $\mathcal{A}$ is written as $\mathsf{cert}(\mathcal{Q}, \mathcal{F}, \mathcal{A})$. If Q is propositional (i.e., if the query is *Boolean*), then $\mathsf{cert}(\mathcal{Q}, \mathcal{F}, \mathcal{A})$ is either empty or it contains the tuple of zero length; in such cases, we commonly write $\mathsf{cert}(\mathcal{Q}, \mathcal{F}, \mathcal{A}) = \mathsf{f}$ and $\mathsf{cert}(\mathcal{Q}, \mathcal{F}, \mathcal{A}) = \mathsf{t}$, respectively. Furthermore, $\vec{a}$ is a *ground certain answer* to $\mathcal{Q}$ w.r.t. $\mathcal{F}$ and $\mathcal{A}$ if the arity of $\vec{a}$ agrees with the arity of Q and $\mathcal{F} \cup \mathcal{A} \cup \mathcal{Q}_G \cup \mathcal{A}_G \models Q(\vec{a})$. The set of all ground certain answers of $\mathcal{Q}$ w.r.t. $\mathcal{F}$ and $\mathcal{A}$ is written as $\mathsf{cert}_G(\mathcal{Q}, \mathcal{F}, \mathcal{A})$.

3 A Framework for Repairing OWL Ontologies

We now introduce the technical framework that the rest of this paper depends on. In particular, in Section 3.1, we formalise the notion of a reasoning algorithm, and in Section 3.2 we formalise the notion of an ontology repair.

3.1 Reasoning Algorithms

As in [16, 15], we abstract concrete reasoners using a notion of a *reasoning algorithm*. This has several benefits: it allows us to precisely specify the assumptions that a reasoner must satisfy for our results to be applicable, it allows us to precisely define the notions of completeness and repair, and it allows us to prove that our algorithm for repairing ontologies indeed guarantees completeness.

Definition 1. *A* reasoning algorithm ans *is a computable function that takes as input an arbitrary OWL 2 DL TBox $\mathcal{T}$, an arbitrary ABox $\mathcal{A}$, and either a special* unsatisfiability *query $*$ or an arbitrary UCQ $\mathcal{Q}$. The return value of* ans *is defined as follows:*

– $\mathsf{ans}(*, \mathcal{T}, \mathcal{A})$ *is either* t *or* f*; and*
– $\mathsf{ans}(\mathcal{Q}, \mathcal{T}, \mathcal{A})$ *is defined only if* $\mathsf{ans}(*, \mathcal{T}, \mathcal{A}) = \mathsf{f}$*, in which case the result is a set of tuples each having the same arity as the query predicate of* $\mathcal{Q}$*.*

Intuitively, $\mathsf{ans}(*, \mathcal{T}, \mathcal{A})$ asks the reasoner to check whether $\mathcal{T} \cup \mathcal{A}$ is unsatisfiable, and $\mathsf{ans}(\mathcal{Q}, \mathcal{T}, \mathcal{A})$ asks the reasoner to evaluate $\mathcal{Q}$ w.r.t. $\mathcal{T}$ and $\mathcal{A}$. If $\mathcal{T} \cup \mathcal{A}$ is unsatisfiable, then each tuple of the same arity as the query predicate of $\mathcal{Q}$ is trivially an answer to $\mathcal{Q}$; therefore, the result of $\mathsf{ans}(\mathcal{Q}, \mathcal{T}, \mathcal{A})$ is of interest only if $\mathsf{ans}(*, \mathcal{T}, \mathcal{A}) = \mathsf{f}$—that is, if ans identifies $\mathcal{T} \cup \mathcal{A}$ as satisfiable.

Example 2. Let rdf, rdfs, rl, and $\mathsf{classify}$ be reasoning algorithms that, given a UCQ $\mathcal{Q}$, an OWL 2 TBox $\mathcal{T}$, and an ABox $\mathcal{A}$, proceed as described next.

The algorithm rdf ignores $\mathcal{T}$ and evaluates $\mathcal{Q}$ w.r.t. $\mathcal{A}$; more precisely, we have $\mathsf{rdf}(*, \mathcal{T}, \mathcal{A}) = \mathsf{f}$ and $\mathsf{rdf}(\mathcal{Q}, \mathcal{T}, \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \emptyset, \mathcal{A})$. Thus, rdf captures the behaviour of RDF reasoners.

The algorithm rdfs constructs a datalog program $\mathcal{P}_{\mathsf{rdfs}}$ by translating each RDFS axiom α in $\mathcal{T}$ into an equivalent datalog rule; then, $\mathsf{rdfs}(*, \mathcal{T}, \mathcal{A})$ is always answered as f; furthermore, $\mathcal{Q}$ is evaluated w.r.t. $\mathcal{T}$ and $\mathcal{A}$ by evaluating $\mathcal{P}_{\mathsf{rdfs}}$ over $\mathcal{A}$—that is, $\mathsf{rdfs}(\mathcal{Q}, \mathcal{T}, \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \mathcal{P}_{\mathsf{rdfs}}, \mathcal{A})$. Thus, rdfs captures the behaviour of RDFS reasoners such as Sesame.

The algorithm rl constructs a datalog program $\mathcal{P}_{\mathsf{rl}}$ by translating each OWL 2 RL axiom α in $\mathcal{T}$ into an equivalent datalog rule; then, $\mathsf{rl}(*, \mathcal{T}, \mathcal{A})$ is answered by checking whether $\mathcal{P}_{\mathsf{rl}} \cup \mathcal{A}$ is satisfiable—that is, $\mathsf{rl}(*, \mathcal{T}, \mathcal{A}) = \mathsf{t}$ if and only if $\mathcal{P}_{\mathsf{rl}} \cup \mathcal{A} \models \bot$; furthermore, $\mathcal{Q}$ is evaluated w.r.t. $\mathcal{T}$ and $\mathcal{A}$ by evaluating $\mathcal{P}_{\mathsf{rl}}$ over $\mathcal{A}$—that is, $\mathsf{rl}(\mathcal{Q}, \mathcal{T}, \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \mathcal{P}_{\mathsf{rl}}, \mathcal{A})$. Thus, rl captures the behaviour of OWL 2 RL reasoners such as Jena and Oracle's Semantic Data Store.

The algorithm $\mathsf{classify}$ first classifies $\mathcal{T}$ using a complete OWL 2 DL reasoner; that is, it computes a TBox $\mathcal{T}'$ containing each subclass axiom $A \sqsubseteq B$ such that $\mathcal{T} \models A \sqsubseteq B$, and A and B are (named) classes occurring in $\mathcal{T}$. The algorithm then proceeds as rl, but considers $\mathcal{T} \cup \mathcal{T}'$ instead of $\mathcal{T}$; more precisely, $\mathsf{classify}(*, \mathcal{T}, \mathcal{A}) = \mathsf{rl}(*, \mathcal{T} \cup \mathcal{T}', \mathcal{A}_{in})$ and $\mathsf{classify}(\mathcal{Q}, \mathcal{T}, \mathcal{A}) = \mathsf{rl}(\mathcal{Q}, \mathcal{T} \cup \mathcal{T}', \mathcal{A})$. In this way, $\mathsf{classify}$ captures the behaviour of OWL 2 RL reasoners such as DLDB and DLE-Jena that try to be 'more complete' by materialising certain consequences of $\mathcal{T}$. ♢

Reasoning algorithms such as the ones specified in Example 2 are incomplete for OWL 2 DL—that is, there exist inputs for which they fail to compute all ground certain answers. These algorithms, however, are complete for a fragment of OWL 2 DL: algorithms rl and $\mathsf{classify}$ are complete for OWL 2 RL inputs, and algorithms rdf and rdfs are complete for RDF and RDFS, respectively. We next formally define the notion of an algorithm being complete for a fragment of OWL 2 DL (w.r.t. ground certain answers). Intuitively, for each UCQ, such an algorithm computes at least all ground certain answers for the UCQ and the part of the TBox that fits into the fragment in question.

Definition 2. *Given an OWL 2 DL TBox* $\mathcal{T}$ *and a fragment* $\mathcal{L}$ *of OWL 2 DL,* $\mathcal{T}|_{\mathcal{L}}$ *is the set of all* $\mathcal{L}$*-axioms in* $\mathcal{T}$*.*

Let ans *be a reasoning algorithm, and let* $\mathcal{L}$ *be a fragment of OWL 2 DL. We say that* ans *is complete for* $\mathcal{L}$ *if the following conditions hold for each OWL 2 DL TBox* $\mathcal{T}$*, each UCQ* $\mathcal{Q}$*, and each ABox* $\mathcal{A}$*:*

- $\mathcal{T}|_{\mathcal{L}} \cup \mathcal{A} \models \bot$ *implies* $\mathsf{ans}(*, \mathcal{T}, \mathcal{A}) = \mathsf{t}$*; and*
- $\mathsf{ans}(*, \mathcal{T}|_{\mathcal{L}}, \mathcal{A}) = \mathsf{f}$ *implies* $\mathsf{cert}_G(\mathcal{Q}, \mathcal{T}|_{\mathcal{L}}, \mathcal{A}) \subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{T}, \mathcal{A})$.

Note that an $\mathcal{L}$-complete reasoning algorithm need not be sound (i.e., it may compute answers that are not certain answers). Although virtually all existing concrete reasoners are based on *sound* algorithms, their implementation may be unsound due to bugs. The results presented in this paper, however, do not require reasoning algorithms to be sound, so we can repair ontologies for concrete reasoners even if they are unsound. This is important in practice since testing reasoners for soundness is currently infeasible.

3.2 The Notion of a Repair

Intuitively, a repair of an OWL 2 DL TBox $\mathcal{T}$ for an algorithm ans is a TBox $\mathcal{R}$ such that adding $\mathcal{R}$ to $\mathcal{T}$ allows ans to correctly compute all ground certain answers for all UCQs and all ABoxes. For a repair to be useful, $\mathcal{R}$ should not introduce new consequences—that is, $\mathcal{R}$ should be a logical consequence of $\mathcal{T}$. This intuition is captured by the following definition.

Definition 3. *Let* $\mathcal{T}$ *be an OWL 2 DL TBox and let* ans *be a reasoning algorithm. A* repair *of* $\mathcal{T}$ *for* ans *is an OWL 2 DL TBox* $\mathcal{R}$ *such that* $\mathcal{T} \models \mathcal{R}$*, and the following conditions hold for each UCQ* $\mathcal{Q}$ *and each ABox* $\mathcal{A}$*:*

- $\mathcal{T} \cup \mathcal{A} \models \bot$ *implies* $\mathsf{ans}(*, \mathcal{T} \cup \mathcal{R}, \mathcal{A}) = \mathsf{t}$*; and*
- $\mathsf{ans}(*, \mathcal{T} \cup \mathcal{R}, \mathcal{A}) = \mathsf{f}$ *implies* $\mathsf{cert}_G(\mathcal{Q}, \mathcal{T}, \mathcal{A}) \subseteq \mathsf{ans}(Q, \mathcal{T} \cup \mathcal{R}, \mathcal{A})$.

Example 3. Let $\mathcal{T}$ be the TBox containing axioms (1)–(7) from Example 1, let $\mathcal{A} = \{\mathsf{PhDSt}(a), \mathsf{ResAsst}(a)\}$, and let $\mathcal{Q}_1$ and $\mathcal{Q}_2$ be the following UCQs:

$$\mathcal{Q}_1 = \{\mathsf{Student}(x) \to Q(x)\} \tag{8}$$

$$\mathcal{Q}_2 = \{\mathsf{Employee}(x) \to Q(x)\} \tag{9}$$

One can check that $\mathsf{cert}_G(\mathcal{Q}_1, \mathcal{T}, \mathcal{A}) = \mathsf{cert}_G(\mathcal{Q}_2, \mathcal{T}, \mathcal{A}) = \{a\}$.

Consider now the algorithm rl from Example 2. Since axioms (6) and (7) are not in OWL 2 RL, the axioms are ignored by the algorithm. Consequently, $\mathcal{P}_{\mathsf{rl}}$ contains only the datalog rules corresponding to axioms (1)–(5), and so $\mathsf{rl}(\mathcal{Q}_1, \mathcal{T}, \mathcal{A}) = \mathsf{rl}(\mathcal{Q}_2, \mathcal{T}, \mathcal{A}) = \emptyset$—that is, rl is not complete for $\mathcal{T}$. One can, however, simulate the relevant consequences of axioms (6) and (7) using the OWL 2 RL TBox $\mathcal{R}_1$ containing the following axioms:

$$\mathsf{GradSt} \sqsubseteq \mathsf{Student} \rightsquigarrow \mathsf{GradSt}(x) \to \mathsf{Student}(x) \tag{10}$$

$$\mathsf{ResAsst} \sqcap \mathsf{PhDSt} \sqsubseteq \mathsf{Employee} \rightsquigarrow \mathsf{ResAsst}(x) \wedge \mathsf{PhDSt}(x) \to \mathsf{Employee}(x) \tag{11}$$

Clearly, $\mathcal{T} \models \mathcal{R}_1$; hence, extending $\mathcal{T}$ with $\mathcal{R}_1$ does not change the consequences of $\mathcal{T}$. The addition of axioms (10) and (11) to $\mathcal{T}$, however, changes the behaviour of algorithm rl; indeed, $\mathsf{rl}(\mathcal{Q}_1, \mathcal{T} \cup \mathcal{R}_1, \mathcal{A}) = \mathsf{rl}(\mathcal{Q}_2, \mathcal{T} \cup \mathcal{R}_1, \mathcal{A}) = \{a\}$. We show in the following section that $\mathcal{R}_1$ is a repair of $\mathcal{T}$ for rl; that is, for an arbitrary UCQ $\mathcal{Q}$ and ABox $\mathcal{A}$, running algorithm rl on $\mathcal{Q}$, $\mathcal{T} \cup \mathcal{R}_1$, and $\mathcal{A}$ computes all ground certain answers of $\mathcal{Q}$ w.r.t. $\mathcal{T}$ and $\mathcal{A}$.

Next, consider the algorithm $\mathsf{classify}$ from Example 2. One can see that $\mathsf{classify}(\mathcal{Q}_1, \mathcal{T}, \mathcal{A}) = \{a\}$ but $\mathsf{classify}(\mathcal{Q}_2, \mathcal{T}, \mathcal{A}) = \emptyset$—that is, $\mathsf{classify}$ is also not complete for $\mathcal{T}$. Moreover, since $\mathsf{classify}$ is complete for OWL 2 RL, TBox $\mathcal{R}_1$ is a repair of $\mathcal{T}$ for $\mathsf{classify}$. Note, however, that the classification of $\mathcal{T}$ takes care of axiom (10). Let $\mathcal{R}_2$ be the TBox containing only axiom (11). One can easily see that $\mathsf{rl}(\mathcal{Q}_2, \mathcal{T} \cup \mathcal{R}_2, \mathcal{A}) = \{a\}$; in fact, we show in the following section that $\mathcal{R}_2$ is a repair of $\mathcal{T}$ for $\mathsf{classify}$.

Finally, consider the algorithm rdfs form Example 2. In spite of the fact that $\mathsf{rdfs}(\mathcal{Q}_1, \mathcal{T} \cup \mathcal{R}_1, \mathcal{A}) = \{a\}$, TBox $\mathcal{R}_1$ is not a repair of $\mathcal{T}$ for rdfs: since (11) is not an RDFS axiom, it is ignored by algorithm rdfs and so $\mathsf{rdfs}(\mathcal{Q}_2, \mathcal{T} \cup \mathcal{R}_1, \mathcal{A}) = \emptyset$. In fact, even if we take $\mathcal{R}'$ to be the maximal set of RDFS axioms that logically follow from $\mathcal{T}$ (which is finite for RDFS), we can see that $\mathsf{rdfs}(\mathcal{Q}_2, \mathcal{T} \cup \mathcal{R}', \mathcal{A}) = \emptyset$; consequently, no repair of $\mathcal{T}$ for rdfs exists. ♢

4 Repairing OWL 2 RL Reasoners

We now turn our attention to the problem of computing a repair for an OWL 2 DL TBox and a reasoning algorithm. In Section 4.1 we present a straightforward way of repairing via so-called *TBox rewritings*, and in Section 4.2 we show how to optimise repairs for reasoning algorithms that are complete for OWL 2 RL.

4.1 TBox Rewritings as Repairs

We next show that a repair of an OWL 2 TBox $\mathcal{T}$ for an algorithm ans can be obtained by *rewriting* $\mathcal{T}$ into the fragment of OWL 2 DL that ans can handle. Before proceeding, we first recapitulate the formal definition of a TBox rewriting.

Definition 4. *Let $\mathcal{T}$ be an OWL 2 DL TBox and let $\mathcal{L}$ be a fragment of OWL 2 DL. An* $\mathcal{L}$-rewriting *of $\mathcal{T}$ is a TBox $\mathcal{T}'$ in fragment $\mathcal{L}$ such that $\mathcal{T} \models \mathcal{T}'$ and the following conditions hold for each ABox $\mathcal{A}$ and each UCQ $\mathcal{Q}$:*

- *$\mathcal{T} \cup \mathcal{A} \models \bot$ implies $\mathcal{T}' \cup \mathcal{A} \models \bot$; and*
- *$\mathsf{cert}_G(\mathcal{Q}, \mathcal{T}, \mathcal{A}) = \mathsf{cert}_G(\mathcal{Q}, \mathcal{T}', \mathcal{A})$.*

Note that, unlike $\mathcal{T}|_{\mathcal{L}}$, an $\mathcal{L}$-rewriting of $\mathcal{T}$ may not be a subset of $\mathcal{T}$, and may even be disjoint from $\mathcal{T}$. Rewritings were introduced mainly to facilitate reasoning in a complex ontology language by reasoning in a simpler language: instead of reasoning directly with an OWL 2 DL TBox $\mathcal{T}$, we compute a TBox $\mathcal{T}'$ in a simpler fragment $\mathcal{L}$ such that, for an arbitrary UCQ $\mathcal{Q}$ and an arbitrary

ABox $\mathcal{A}$, the ground certain answers of $\mathcal{T}$ and $\mathcal{T}'$ coincide; we can then answer queries over $\mathcal{T}$ by applying to $\mathcal{T}'$ a reasoning algorithm complete for $\mathcal{L}$.

Example 4. Let $\mathcal{T}$ be the TBox consisting of axioms (1)–(7). The OWL 2 RL TBox $\mathcal{T}'$ consisting of axioms (1)–(5) and (10)–(11) is an OWL 2 RL rewriting of $\mathcal{T}$. Thus, instead of answering a query over $\mathcal{T}$ using an OWL 2 DL reasoner, we can answer the query over $\mathcal{T}'$ using an OWL 2 RL reasoner.

Note, however, that no RDFS rewriting of $\mathcal{T}$ exists: even if we take $\mathcal{T}''$ to be the maximal set of RDFS axioms that logically follow from $\mathcal{T}$, for $\mathcal{Q}_2$ and $\mathcal{A}$ as defined in Example 3 we have $\mathsf{cert}_G(\mathcal{Q}_2, \mathcal{T}'', A) = \emptyset$. ♢

The following proposition, the proof of which is straightforward, establishes the connection between TBox rewritings and repairs. According to this proposition, the TBox $\mathcal{T}'$ in Example 4 is a repair of $\mathcal{T}$ for algorithm rl.

Proposition 2. *Let $\mathcal{T}$ be an OWL 2 TBox, let $\mathcal{L}$ be a fragment of OWL 2, and let* ans *be a reasoning algorithm complete for $\mathcal{L}$. If $\mathcal{R}$ is an $\mathcal{L}$-rewriting of $\mathcal{T}$, then $\mathcal{R}$ is a repair of $\mathcal{T}$ for* ans.

Although this simple result provides us with a straightforward way of repairing certain OWL 2 DL ontologies, as we discuss in the following section, repairs obtained in this way can be unnecessarily large. Therefore, we develop a technique that optimises a repair for the reasoner at hand.

4.2 Repairing a Class of Algorithms Complete for OWL 2 RL

Reasoners based on RDF triple stores and databases, such as Jena, OWLim, Oracle's Semantic Datastore and DLE-Jena, are typically complete at least for OWL 2 RL. Therefore, in the rest of this section we focus on repairing an OWL 2 DL TBox $\mathcal{T}$ for a reasoner ans that is complete for OWL 2 RL.

By Proposition 2, we can solve the aforementioned problem by computing an OWL 2 RL rewriting of $\mathcal{T}$. Depending on the language that $\mathcal{T}$ is expressed in, systems such as REQUIEM [13] and KAON2 [5] can compute a (possibly disjunctive) datalog rewriting; now whenever the rewriting is a datalog program, each datalog rule in the rewriting can always be 'rolled-up' into an OWL 2 RL axiom. Therefore, in order to simplify the presentation, we consider such rewritings to be OWL 2 RL TBoxes rather than datalog programs.

Note, however, that, if an OWL 2 RL rewriting $\mathcal{T}'$ of $\mathcal{T}$ exists, it must capture *all* OWL 2 RL consequences of $\mathcal{T}$ and can thus be very large; in fact, the size of $\mathcal{T}'$ can in the worst case even be exponential in the size of $\mathcal{T}$. Thus, to make our approach practicable, it is desirable to reduce the size of a repair as much as possible. This can be achieved in (at least) two ways.

First, rewritings often contain redundant axioms, so we can try to *minimise* them—that is, we can identify a smallest subset of $\mathcal{T}'$ that is also a rewriting of $\mathcal{T}$. While minimisation can be computationally very expensive, as a bare minimum we can eliminate from $\mathcal{T}'$ each axiom α for which $\mathcal{T}|_{\mathsf{rl}} \models \alpha$ holds; this can be straightforwardly checked using a sound and complete OWL 2 DL reasoner. A

repair obtained in this way does not contain axioms whose consequences can be derived from $\mathcal{T}|_{\mathsf{rl}}$ by OWL 2 RL complete reasoning algorithms.

Second, we can exploit the fact that, while a reasoning algorithm might be complete only for OWL 2 RL, the algorithm may actually take into account some consequences of the axioms in $\mathcal{T} \setminus \mathcal{T}|_{\mathsf{rl}}$. Consider again algorithm classify from Example 2 and a TBox consisting of axioms (1)–(7). As shown in Example 4, a rewriting of this TBox consists of axioms (1)–(5) and (10)–(11); however, as discussed in Example 3, only axiom (11) is needed to repair the TBox for classify. Based on this observation, in the rest of this section we show how to reduce the size of a repair beyond what is possible via minimisation of a rewriting.

In order to achieve this goal, we first introduce the notion of a *datalog-reproducible* algorithm, which captures the class of reasoners to which our approach is applicable. This notion was inspired by an observation that many state of the art reasoners that can handle (a fragment of) OWL 2 DL are based on deductive database technologies: given a UCQ $\mathcal{Q}$, a TBox $\mathcal{T}$, and an ABox $\mathcal{A}$, these reasoners first 'saturate' $\mathcal{A}$ by adding all assertions that are entailed by $\mathcal{T} \cup \mathcal{A}$; next, they answer $\mathcal{Q}$ by simply evaluating it over the saturated ABox. The ABox saturation process depends only on $\mathcal{T}$ and $\mathcal{A}$, and it can be characterised at an abstract level as evaluating over $\mathcal{A}$ a datalog program that depends only on $\mathcal{T}$. This intuition is formalised by the following definition.

Definition 5. *A reasoning algorithm* ans *is* datalog-reproducible *if, for each OWL 2 DL TBox $\mathcal{T}$, a datalog program $\mathcal{P}_\mathcal{T}$ exists such that the following holds:*

- *for each ABox $\mathcal{A}$ and each UCQ $\mathcal{Q}$,*
  - $\mathsf{ans}(*, \mathcal{T}, \mathcal{A}) = \mathsf{t}$ *if and only if* $\mathcal{P}_\mathcal{T} \cup \mathcal{A} \models \bot$*, and*
  - $\mathsf{ans}(*, \mathcal{T}, \mathcal{A}) = \mathsf{f}$ *implies* $\mathsf{ans}(\mathcal{Q}, \mathcal{T}, \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \mathcal{P}_\mathcal{T}, \mathcal{A})$*; and*
- *algorithm* ans *is* monotonic*—that is, for all OWL 2 DL TBoxes $\mathcal{T}$ and $\mathcal{T}'$, we have $\mathcal{P}_{\mathcal{T} \cup \mathcal{T}'} \models \mathcal{P}_\mathcal{T}$.*

If program $\mathcal{P}_\mathcal{T}$ contains predicates or individuals that do not occur in $\mathcal{T}$, these are considered to be 'private' to ans *and are not accessible elsewhere (e.g., in queries, TBoxes, and ABoxes).*

Note that a datalog-reproducible reasoning algorithm does not need to construct $\mathcal{P}_\mathcal{T}$; what matters is that *some* datalog program $\mathcal{P}_\mathcal{T}$ exists that characterises the behaviour of the algorithm.

Example 5. Algorithms rdf, rdfs and rl from Example 2 explicitly construct a datalog program $\mathcal{P}_\mathcal{T}$, so they are clearly datalog-reproducible. Note, however, that algorithm classify is also datalog-reproducible even through it does not directly construct a datalog program: the algorithm's behaviour can be characterised by a program $\mathcal{P}_\mathcal{T}$ containing all rules corresponding to the axioms in $\mathcal{T}|_{\mathsf{rl}}$ extended with the rule $A(x) \rightarrow B(x)$ for each pair of classes A and B occurring in $\mathcal{T}$ such that $\mathcal{T} \models A \sqsubseteq B$. ♢

Note also that, even if a reasoner uses a particular datalog program as part of its implementation, the actual rules of the program may not be available to

the users of the reasoner. For example, the rules used for reasoning by Oracle's Semantic Data Store are not publicly available; however, the reasoner can still be considered datalog-reproducible as its external behaviour can be captured using a datalog program. As we show next, our approach does not need to know the actual rules in order to repair an ontology: it suffices to know that a suitable datalog program exists.

As we discuss next, not all reasoning algorithms are datalog-reproducible.

Example 6. Reasoning algorithms based on query rewriting (e.g., algorithms underpinning the QuONTO reasoner) are not datalog-reproducible: although they answer queries by first constructing a datalog program, this program depends on *both* on the query and the TBox, and not on the TBox alone.

As another example, consider a reasoning algorithm that behaves as algorithm rdf from Example 2, but that first removes from the input ABox each assertion involving an individual whose IRI belongs to a certain predefined namespace. (This could be done, e.g., for efficiency or trust reasons.) Fact 'removal' cannot be represented using a monotonic theory, so this algorithm is clearly not datalog-reproducible. ♢

We now show how to compute a repair of a TBox $\mathcal{T}$ for an algorithm ans that is datalog-reproducible and complete for OWL 2 RL. Intuitively, the behaviour of ans on $\mathcal{T}$ is characterised by a datalog program $\mathcal{P}_\mathcal{T}$ so, given an OWL 2 RL rewriting $\mathcal{T}'$ of $\mathcal{T}$, we can safely disregard each axiom in $\mathcal{T}'$ that is logically entailed by $\mathcal{P}_\mathcal{T}$. In other words, a repair of $\mathcal{T}$ for ans needs to contain only the *essential* axioms of $\mathcal{T}'$—that is, the axioms that are not entailed by $\mathcal{P}_\mathcal{T}$. Furthermore, the rewriting $\mathcal{T}'$ is an OWL 2 RL TBox, so each axiom $\alpha \in \mathcal{T}'$ corresponds to an equivalent datalog rule $\pi(\alpha)$; but then, by Proposition 1 we can construct from $\pi(\alpha)$ an ABox $\mathcal{A}_\alpha$ and a query $\mathcal{Q}_\alpha$ such that $\mathsf{ans}(\mathcal{Q}_\alpha, \mathcal{T}, \mathcal{A}_\alpha) = \mathsf{t}$ if and only if $\mathcal{P}_\mathcal{T} \models \alpha$. A repair $\mathcal{R}$ of $\mathcal{T}$ for ans can thus be obtained as a TBox that contains each axiom $\alpha \in \mathcal{T}'$ such that $\mathsf{ans}(\mathcal{Q}_\alpha, \mathcal{T}, \mathcal{A}_\alpha) = \mathsf{f}$. Since ans is complete for OWL 2 RL and $\mathcal{R}$ is an OWL 2 RL TBox, extending $\mathcal{T}$ with $\mathcal{R}$ will allow ans to recover the missing consequences of $\mathcal{T}$ and thus become complete.

Definition 6. *Let $\mathcal{T}$ be an OWL 2 TBox, let $\mathcal{T}'$ be an OWL 2 RL rewriting of $\mathcal{T}$, let* ans *be a datalog-reproducible reasoning algorithm, and let λ be a substitution that maps each variable in the signature to a fresh individual. The* essential subset *of $\mathcal{T}'$ for* ans *is the TBox $\mathcal{R}$ that contains each axiom $\alpha \in \mathcal{T}'$ satisfying the following conditions, where $r = \pi(\alpha)$ and $\mathcal{A}^r_\lambda = \{\lambda(B) \mid B \in \mathsf{body}(r)\}$:*[2]

1. $\mathsf{head}(r) = \bot$ *and* $\mathsf{ans}(*, \mathcal{T}, \mathcal{A}^r_\lambda) = \mathsf{f}$*; or*
2. $\mathsf{head}(r) = H$ *with* $H \neq \bot$ *and* $\mathsf{ans}(\{\lambda(H) \rightarrow Q\}, \mathcal{T}, \mathcal{A}^r_\lambda) = \mathsf{f}$*, for Q a propositional query predicate.*

Example 7. Let $\mathcal{T}$ contain axioms (1)–(7), and let $\mathcal{T}'$ be a rewriting of $\mathcal{T}$ that contains axioms (1)–(5) and (10)–(11).

[2] Note that $\pi(\alpha)$ is the translation of α into a datalog rule from Section 2.

The essential subset of $\mathcal{T}'$ for algorithm rl from Example 2 contains (10) and (11). For example, let α be axiom (10), so $\pi(\alpha) = r = \mathsf{GradSt}(x) \rightarrow \mathsf{Student}(x)$. Then for $\mathcal{A}^r_\lambda = \{\mathsf{GradSt}(a)\}$ and $\mathcal{Q} = \{\mathsf{St}(a) \rightarrow Q\}$ we have $\mathsf{rl}(\mathcal{Q}, \mathcal{T}, \mathcal{A}^r_\lambda) = \mathsf{f}$, so α must be included into the essential subset of $\mathcal{T}'$. Analogous reasoning applies to axiom (11).

In contrast, the essential subset of $\mathcal{T}'$ for algorithm $\mathsf{classify}$ contains only (11) since, for $\mathcal{Q}$ and $\mathcal{A}^r_\lambda$ as defined above, we have $\mathsf{classify}(\mathcal{Q}, \mathcal{T}, \mathcal{A}^r_\lambda) = \mathsf{t}$. $\diamondsuit$

We next present the main result of this paper, which shows that essential subsets can be used as repairs.

Theorem 8. *Let $\mathcal{T}$ be an OWL 2 TBox, let $\mathcal{T}'$ be an OWL 2 RL rewriting of $\mathcal{T}$, let ans be a datalog-reproducible algorithm complete for OWL 2 RL, and let $\mathcal{R}$ be the essential subset of $\mathcal{T}'$ for ans. Then, $\mathcal{R}$ is a repair of $\mathcal{T}$ for ans.*

Proof. Assume that ans is complete for OWL 2 RL and let λ be a substitution that maps each variable in the signature to a fresh individual.

We first show that $\mathcal{P}_\mathcal{R} \models \mathcal{R}$. To this end, let $\mathcal{R}_1$ be the subset of all rules $r \in \mathcal{R}$ such that $\mathcal{P}_\mathcal{R} \cup \mathcal{A}^r_\lambda \models \bot$, and let $\mathcal{R}_2 = \mathcal{R} \setminus \mathcal{R}_1$. Furthermore, let $\overline{\mathcal{R}}_1$ be the set of rules obtained by replacing the head atom in each rule in $\mathcal{R}_1$ with $\bot$. Since clearly $\overline{\mathcal{R}}_1 \cup \mathcal{R}_2 \models \mathcal{R}$, it suffices to show that $\mathcal{P}_\mathcal{R} \models \overline{\mathcal{R}}_1 \cup \mathcal{R}_2$. So, let r be an arbitrary rule in $\overline{\mathcal{R}}_1 \cup \mathcal{R}_2$.

- Assume that $r \in \overline{\mathcal{R}}_1$. Then, by the definition of $\overline{R}_1$, we have $\mathcal{P}_\mathcal{R} \cup \mathcal{A}^r_\lambda \models \bot$. But then, since $\mathsf{head}(r) = \bot$, by Proposition 1 we have $\mathcal{P}_\mathcal{R} \models r$, as required.
- Assume that $r \in \mathcal{R}_2$. Then, $\mathcal{P}_\mathcal{R} \cup \mathcal{A}^r_\lambda \not\models \bot$. By the definition of datalog-reproducible algorithms, then $\mathsf{ans}(*, \mathcal{R}, \mathcal{A}^r_\lambda) = \mathsf{f}$. Furthermore, we clearly have $\mathcal{R} \cup \mathcal{A}^r_\lambda \models \lambda(H)$, where $H = \mathsf{head}(r)$. Hence, $\mathsf{cert}_G(\mathcal{Q}, \mathcal{R}, \mathcal{A}^r_\lambda) = \mathsf{t}$ for $\mathcal{Q} = \{\lambda(H) \rightarrow Q\}$. But then, since ans is complete for OWL 2 RL and $\mathcal{R}$ is an OWL 2 RL TBox, we have that $\mathsf{cert}_G(\mathcal{Q}, \mathcal{R}, \mathcal{A}^r_\lambda) \subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{R}, \mathcal{A}^r_\lambda)$ for each UCQ $\mathcal{Q}$. Therefore, $\mathsf{ans}(\mathcal{Q}, \mathcal{R}, \mathcal{A}^r_\lambda) = \mathsf{t}$, so by the definition of datalog-reproducible algorithms we also have $\mathcal{P}_\mathcal{R} \cup \mathcal{A}^r_\lambda \models \lambda(H)$. But then, by Proposition 1, we have $\mathcal{P}_\mathcal{R} \models r$, as required.

We next show that, since $\mathcal{R}$ is an essential subset of $\mathcal{T}'$ for ans, we have $\mathcal{P}_\mathcal{T} \models \mathcal{T}' \setminus \mathcal{R}$. To this end, consider an arbitrary rule in $r \in \mathcal{T}' \setminus \mathcal{R}$. We have the following possibilities:

- $\mathsf{head}(r) = \bot$. In this case, by the definition of essential subset we have that $\mathsf{ans}(*, \mathcal{T}, \mathcal{A}^r_\lambda) = \mathsf{t}$ and hence $\mathcal{P}_\mathcal{T} \cup \mathcal{A}^r_\lambda \models \bot$. But then, by Proposition 1 we have $\mathcal{P}_\mathcal{T} \models r$.
- $\mathsf{head}(r) = H$ where $H \neq \bot$. By the definition of essential subset, we have $\mathsf{ans}(\{\lambda(H) \rightarrow Q\}, \mathcal{T}, \mathcal{A}^r_\lambda) = \mathsf{t}$. But then, by Proposition 1 we have $\mathcal{P}_\mathcal{T} \models r$.

We now show that $\mathcal{P}_\mathcal{T} \models \mathcal{T}' \setminus \mathcal{R}$ and $\mathcal{P}_\mathcal{R} \models \mathcal{R}$ imply $\mathcal{P}_{\mathcal{T} \cup \mathcal{R}} \models \mathcal{T}'$. Since we have $\mathcal{P}_\mathcal{T} \models \mathcal{T}' \setminus \mathcal{R}$, we also clearly have $\mathcal{P}_\mathcal{T} \cup \mathcal{R} \models \mathcal{T}'$; since $\mathcal{P}_\mathcal{R} \models \mathcal{R}$, we have

$\mathcal{P}_\mathcal{T} \cup \mathcal{P}_\mathcal{R} \models \mathcal{T}'$ as well. Since ans satisfies the monotonicity property from Definition 5, we have $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \models \mathcal{P}_\mathcal{T}$ and $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \models \mathcal{P}_\mathcal{R}$; thus, $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \models \mathcal{P}_\mathcal{T} \cup \mathcal{P}_\mathcal{R}$. But then, $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \models \mathcal{T}'$, as required.

We finally use the fact that $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \models \mathcal{T}'$ to show that the following properties hold for each UCQ $\mathcal{Q}$ and each ABox $\mathcal{A}$.

1. $\mathcal{T} \cup \mathcal{A} \models \bot$ implies $\mathsf{ans}(*, \mathcal{T} \cup \mathcal{R}, \mathcal{A}) = \mathsf{t}$; and
2. $\mathsf{ans}(*, \mathcal{T} \cup \mathcal{R}, \mathcal{A}) = \mathsf{f}$ implies then $\mathsf{cert}_G(\mathcal{Q}, \mathcal{T}, \mathcal{A}) \subseteq \mathsf{ans}(Q, \mathcal{T} \cup \mathcal{R}, \mathcal{A})$.

(Property 1). Assume that $\mathcal{T} \cup \mathcal{A} \models \bot$. Since $\mathcal{T}'$ is an OWL 2 RL rewriting of $\mathcal{T}$, we have $\mathcal{T}' \cup \mathcal{A} \models \bot$. But then, since $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \models \mathcal{T}'$, we also have $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \cup \mathcal{A} \models \bot$, as required.

(Property 2). Assume that $\mathsf{ans}(*, \mathcal{T} \cup \mathcal{R}, \mathcal{A}) = \mathsf{f}$ and consider an arbitrary tuple $\vec{a} \in \mathsf{cert}_G(\mathcal{Q}, \mathcal{T}, \mathcal{A})$. Since $\mathcal{T}'$ is an OWL 2 RL rewriting of $\mathcal{T}$, then we have $\vec{a} \in \mathsf{cert}_G(\mathcal{Q}, \mathcal{T}', \mathcal{A})$, so $\mathcal{T}' \cup \mathcal{A} \cup \mathcal{Q}_G \cup \mathcal{A}_G \models Q(\vec{a})$. But then, since $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \models \mathcal{T}'$, we also have that $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \cup \mathcal{A} \cup \mathcal{Q}_G \cup \mathcal{A}_G \models Q(\vec{a})$. Furthermore, since $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}}$ is a datalog program, we have that $\mathcal{P}_{\mathcal{T}\cup\mathcal{R}} \cup \mathcal{A} \cup \mathcal{Q} \models Q(\vec{a})$ and hence we have $\vec{a} \in \mathsf{cert}(\mathcal{Q}, \mathcal{P}_{\mathcal{T}\cup\mathcal{R}}, \mathcal{A})$; consequently, $\vec{a} \in \mathsf{ans}(Q, \mathcal{T} \cup \mathcal{R}, \mathcal{A})$, as required.

We finally show that $\mathcal{R}$ is a repair of $\mathcal{T}$ for ans. Since $\mathcal{T}'$ is an OWL 2 RL rewriting of $\mathcal{T}$, we have $\mathcal{T} \models \mathcal{T}'$; since $\mathcal{R} \subseteq \mathcal{T}'$, we also have $\mathcal{T} \models \mathcal{R}$. This together with Properties 1 and 2 implies our claim. □

Theorem 8 and the observations made in Example 7 thus confirm our claims from Example 3: axioms (10) and (11) constitute a repair of $\mathcal{T}$ for algorithm rl, and axiom (11) alone constitutes a repair for algorithm $\mathsf{classify}$.

5 Evaluation

We developed a prototype tool for computing repairs. Our implementation uses the system REQUIEM [13] for the computation of TBox rewritings. We evaluated our approach using the following two well-known ontologies.

First, we used the well-known Lehigh University Benchmark (LUBM) [4]—an ontology extensively used for evaluating performance of ontology-based systems. We used LUBM's generator of large datasets and the supplied 14 test queries.

Second, we used a small subset of the GALEN ontology [14]—a complex medical ontology. We used a subset of GALEN because REQUIEM was unable to handle the full version of GALEN. Since we are not aware of a large ABox or a data generator for GALEN, we created synthetic data by extending the techniques for ABox generation from [16, 15]; we thus obtained ABoxes with (approximately) 2000, 4000, 8000, 16000 and 32000 assertions. Furthermore, we tested the systems using four atomic queries presented in [15].

We evaluated the following reasoning systems: OWLim v2.9.1,[3] Jena v2.6.3[4] and DLE-Jena v2.0.[5]

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

Table 1. Repairing the LUBM ontology for OWLim, Jena and DLE-Jena

Ontology	$\mathcal{T}_{\mathsf{rew}}$	$\mathcal{T}_{\mathsf{min}}$	$\mathcal{R}_{\mathsf{owlim}}$	$\mathcal{R}_{\mathsf{jena}}$	$\mathcal{R}_{\mathsf{dle\text{-}jena}}$
Ontology size	331	7	3	3	0
Time to compute ontology (in s)	4.7	7.9	3.3	6.3	14

For each test ontology $\mathcal{T}$ and each reasoning system ans mentioned above, we performed the following tasks.

1. We computed an OWL 2 RL rewriting $\mathcal{T}_{\mathsf{rew}}$ of the input TBox $\mathcal{T}$ and recorded the time needed to complete this step.
2. As mentioned in Section 4.2, $\mathcal{T}_{\mathsf{rew}}$ can contain many axioms, so we minimised $\mathcal{T}_{\mathsf{rew}}$ as follows. First, we eliminated each axiom α such that $\mathcal{T}|_{\mathsf{rl}} \models \alpha$. Second, for all pairs of distinct remaining axioms α_1 and α_2, we eliminated α_2 if $\mathcal{T}|_{\mathsf{rl}} \cup \{\alpha_1\} \models \alpha_2$. Let $\mathcal{T}_{\mathsf{min}}$ be the resulting set of axioms; clearly, $\mathcal{T}|_{\mathsf{rl}} \cup \mathcal{T}_{\mathsf{min}}$ is still a rewriting of $\mathcal{T}$. Note that $\mathcal{T}_{\mathsf{min}}$ can depend on the order in which we select α_1 and α_2 in the second step; however, we did not notice significant variance in our tests. We conducted all entailment checks using HermiT—a sound and complete OWL 2 DL reasoner—and we recorded the time needed to complete this step.
3. We extracted from $\mathcal{T}_{\mathsf{min}}$ the essential subset $\mathcal{R}$ for ans as described in Definition 6, and we recorded the time needed to complete this step. By Theorem 8, $\mathcal{R}$ is a repair of $\mathcal{T}$ for ans.
4. To estimate the effect that repairing $\mathcal{T}$ has on the performance of ans, we proceeded as follows. We first applied ans to $\mathcal{T}$ and each corresponding data set and query, and we recorded the load time, the query evaluation time, and the number of certain answers returned. Next, we repeated the experiment by applying ans to $\mathcal{T} \cup \mathcal{R}$. The results obtained using $\mathcal{T} \cup \mathcal{R}$ are compared against Pellet—a sound and complete OWL 2 DL reasoner.

The results of repairing LUBM are shown in Table 1. Although the initial rewriting is quite large, our procedure computes repairs for OWLim and Jena that consist of only the following three axioms:

$$\mathsf{GradStudent} \sqsubseteq \mathsf{Student} \quad \mathsf{Director} \sqsubseteq \mathsf{Employee} \quad \mathsf{ResearchAssist} \sqsubseteq \mathsf{Employee}$$

The repair for DLE-Jena is empty—that is, the system is already complete for LUBM. This is due to the fact that the repair for OWLim and Jena consists only of simple subclass axioms, all of which are derived by DLE-Jena's preprocessing phase (DLE-Jena is similar to the $\mathsf{classify}$ algorithm from Example 2). In all cases computing the repair took less than 15 seconds.

For OWLim and Jena, $\mathcal{T}_{\mathsf{min}} \setminus \mathcal{R}$ is non-empty, which suggests that these systems can process 'more' than just OWL 2 RL. We observed that, for many axioms in $\mathcal{T}_{\mathsf{min}} \setminus \mathcal{R}$ of the form $A \sqsubseteq B$, TBox $\mathcal{T}$ contains an axiom $A \sqsubseteq B \sqcap \exists R.C$. The latter is not an OWL 2 RL axiom, so $\mathcal{T}|_{\mathsf{rl}} \not\models A \sqsubseteq B$ and $A \sqsubseteq B$ is not removed from $\mathcal{T}_{\mathsf{min}}$. The OWL 2 RL/RDF rules from [11], however, correctly handle the conjunction in the superclass position—that is, given an assertion $A(a)$, they

Table 2. Repairing GALEN for OWLim, Jena and DLE-Jena

Ontology	$\mathcal{T}_{\mathsf{rew}}$	$\mathcal{T}_{\mathsf{min}}$	$\mathcal{R}_{\mathsf{owlim}}$	$\mathcal{R}_{\mathsf{jena}}$	$\mathcal{R}_{\mathsf{dle\text{-}jena}}$
Ontology size	1666	291	11	10	5
Time to compute ontology (in s)	380	126	426	1586	-

Table 3. Number of certain answers for GALEN (without repairs)

	2000				4000				8000		16000		32000	
Queries	J	D	O	P	J	D	O	P	O	P	O	P	O	P
$\mathcal{Q}1$	73	65	65	92	226	206	206	280	501	672	1281	1587	1379	1727
$\mathcal{Q}2$	49	43	43	72	158	135	135	217	368	532	1008	1353	1148	1502
$\mathcal{Q}3$	81	74	74	97	234	212	212	283	515	678	1301	1588	1355	1714
$\mathcal{Q}4$	112	226	78	260	334	656	232	756	547	1687	1272	3463	1445	3706
J=Jena, D=DLE-Jena, O=OWLim, P=Pellet														

derive $B(a)$ and $\exists R.C(a)$. This effectively allows OWLim and Jena to use the $A \sqsubseteq B$ 'part' of $A \sqsubseteq B \sqcap \exists R.C$, so the repair does not need to contain $A \sqsubseteq B$. Thus, tailoring the repair to a particular reasoner can exploit the reasoning capabilities of the reasoner at hand and thus produce smaller repairs.

We observed no measurable performance changes for OWLim and Jena after repairing, which is not surprising since the repairs contained only a few simple axioms. Moreover, both OWLim and Jena are complete for the LUBM dataset already when using the original TBox, so no change in the number of answers produced was observed either.

The results of repairing GALEN are shown in Table 2. Again, the repair is quite small, despite the fact that GALEN heavily uses features outside OWL 2 RL such as existential quantification. The repair, however, is more complex than in the case of LUBM, containing axioms such as subsumptions between complex class expressions with several nested existential quantifiers. As with LUBM, the repair for DLE-Jena is smaller than the repairs for OWLim and Jena. Note, however, that DLE-Jena ran out of memory while computing the repair from $\mathcal{T}_{\mathsf{min}}$; thus, since DLE-Jena is 'more complete' than Jena, we produced $\mathcal{R}_{\mathsf{dle\text{-}jena}}$ by applying Definition 6 to $\mathcal{R}_{\mathsf{jena}}$.

Table 3 shows the number of certain answers computed by each system for each dataset, using the original GALEN TBox; please note that we could only load datasets 2000 and 4000 into Jena and DLE-Jena. As expected, all systems returned fewer answers than Pellet. Using the repaired TBoxes, however, *all* systems returned the *same* number of certain answers as Pellet. Thus, repairing an ontology can significantly improve the quality of answers that a reasoner produces for a given ontology.

Table 4 shows the loading times for both the original and the repaired ontologies. As one can see, repairing the ontology leads to an increase in loading times of about 20% on average. The times for the repaired ontology, however, are of the same order of magnitude as the original times; hence, the increase in loading time may be acceptable given that the systems then return complete answers.

Table 4. Loading times for original and repaired TBoxes (in ms)

		2000	4000	8000	16000	32000
OWLim	$\mathcal{T}$	1411	2328	3611	6000	6871
	$\mathcal{T} \cup \mathcal{R}$	1768	2807	4279	7815	8696
Pellet	$\mathcal{T}$	2598	4623	9596	11275	12086
Jena	$\mathcal{T}$	25524	117524	-	-	-
	$\mathcal{T} \cup \mathcal{R}$	34000	139839	-	-	-
DLE-Jena	$\mathcal{T}$	23198	138075	-	-	-
	$\mathcal{T} \cup \mathcal{R}$	24844	139129	-	-	-

Table 5. Query answering times for repaired GALEN (in ms)

	2000				4000				8000		16000		32000	
Queries	J	D	O	P	J	D	O	P	O	P	O	P	O	P
$\mathcal{Q}1$	15	156	39	734	5	155	46	1765	41	5503	53	7460	50	8122
$\mathcal{Q}2$	2	3	1	425	5	11	3	1226	6	2635	15	4271	17	4620
$\mathcal{Q}3$	4	3	1	926	8	8	4	2711	100	6473	18	10188	20	11866
$\mathcal{Q}4$	8	9	1	37	21	21	11	47	20	88	55	127	42	136
J=Jena, D=DLE-Jena, O=OWLim, P=Pellet														

Furthermore, OWLim is much faster than Pellet even on the repaired ontology, which suggests that using an incomplete reasoner with a repaired ontology might be more appropriate in practice than using a complete reasoner.

Table 5 shows the query answering times for the repaired ontology. Since all systems perform reasoning during loading (i.e., they saturate the input ABox), repairing the ontology produced no noticeable difference on the query answering times. Note that all systems are much faster than Pellet, which again suggests that using an incomplete reasoner with a repaired ontology might offer significant advantages compared to using a complete reasoner.

6 Conclusions

In this paper, we studied the problem of *repairing* an ontology for a given incomplete reasoner in a way that guarantees completeness. Our repairs guarantee completeness w.r.t. ground certain answers independently of data and queries: once an ontology has been repaired for a given system, the system will, for any given query and data set, compute all ground certain answers that follow from the original ontology. Our approach tries to limit the size of the repair as much as possible. Our experiments suggest that repairs may indeed be very small, and that their effect on system performance may be negligible. This allows application designers to use highly scalable incomplete reasoners, but with the guarantee that they produce the same answers as provably complete reasoners, thus having the 'the best of both worlds'.

We leave the extension of our techniques to more expressive DLs, as well as a more extensive evaluation, for future work.

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

References

1. Baader, F., McGuinness, D., Nardi, D., Patel-Schneider, P.: The Description Logic Handbook: Theory, implementation and applications. Cambridge University Press (2002)
2. Botoeva, E., Calvanese, D., Rodriguez-Muro, M.: Expressive approximations in dl-lite ontologies. In: Proceedings of the 14th Int. Conf. on Artificial Intelligence: Methodology, Systems, Applications (AIMSA 2010). pp. 21–31. Springer (2010)
3. 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)
4. Guo, Y., Pan, Z., Heflin, J.: LUBM: A Benchmark for OWL Knowledge Base Systems. Journal of Web Semantics 3(2), 158–182 (2005)
5. Hustadt, U., Motik, B., Sattler, U.: Deciding Expressive Description Logics in the Framework of Resolution. Information & Computation 206(5), 579–601 (2008)
6. Kiryakov, A., Ognyanov, D., Manov, D.: Owlim-a pragmatic semantic repository for owl. In: Dean, M., Guo, Y., Jun, W., Kaschek, R., Krishnaswamy, S., Pan, Z., Sheng, Q.Z. (eds.) WISE Workshops. pp. 182–192 (2005)
7. 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)
8. McBride, Brian: Jena: Implementing the RDF Model and Syntax Specification. In: International Workshop on the Semantic Web 2001 (2001)
9. Meditskos, G., Bassiliades, N.: Combining a DL reasoner and a rule engine for improving entailment-based OWL reasoning. In: Proc. of ISWC. pp. 277–292 (2008)
10. Motik, B., Patel-Schneider, P.F., Parsia, B.: OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax, W3C Recommendation. `http://www.w3.org/TR/owl2-syntax/` (October 27 2009)
11. Motik, B., Cuenca Grau, B., Horrocks, I., Wu, Z., Fokoue, A., (Editors), C.L.: OWL 2 Web Ontology Language Profiles. W3C Recommendation (2009)
12. Pan, J.Z., Thomas, E.: Approximating OWL-DL Ontologies. In: Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07). pp. 1434–1439 (2007)
13. 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)
14. Rector, A.L., Rogers, J.: Ontological and practical issues in using a description logic to represent medical concept systems: Experience from galen. In: Barahona, P., Bry, F., Franconi, E., Henze, N., Sattler, U. (eds.) Reasoning Web, Second International Summer School 2006, pp. 197–231 (2006)
15. Stoilos, G., Cuenca Grau, B., Horrocks, I.: Completeness guarantees for incomplete reasoners. In: Proc. of ISWC-10. 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)
17. Wu, Z., Eadon, G., Das, S., Chong, E.I., Kolovski, V., Annamalai, M., Srinivasan, J.: Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In: Proc. of ICDE. pp. 1239–1248. IEEE (2008)