Hybrid Query Answering Over OWL Ontologies

Giorgos Stoilos and **Giorgos Stamou**[1]

Abstract. Query answering over OWL 2 DL ontologies is an important reasoning task for many modern applications. Unfortunately, due to its high computational complexity, OWL 2 DL systems are still not able to cope with datasets containing billions of data. Consequently, application developers often employ provably scalable systems which only support a fragment of OWL 2 DL and which are, hence, most likely incomplete for the given input. However, this notion of completeness is too coarse since it implies that there exists *some* query and *some* dataset for which these systems would miss answers. Nevertheless, there might still be a large number of user queries for which they can compute all the right answers even over OWL 2 DL ontologies. In the current paper, we investigate whether, given a query $\mathcal{Q}$ with only distinguished variables over an OWL 2 DL ontology $\mathcal{T}$ and a system ans, it is possible to identify in an efficient way if ans is complete for $\mathcal{Q}, \mathcal{T}$ and every dataset. We give sufficient conditions for (in)completeness and present a hybrid query answering algorithm which uses ans when it is complete, otherwise it falls back to a fully-fledged OWL 2 DL reasoner. However, even in the latter case, our algorithm still exploits ans as much as possible in order to reduce the search space of the OWL 2 DL reasoner. Finally, we have implemented our approach using a concrete system ans and OWL 2 DL reasoner obtaining encouraging results.

1 INTRODUCTION

Query answering over ontological knowledge expressed in the OWL 2 DL language has attracted the interest of many researchers as well as application developers the last decade [6, 9]. In such a setting, given an ontology $\mathcal{T}$ (also called TBox) expressed in OWL and a set of (possibly distributed) data sources, answers to user queries reflect both the data in the sources as well as the knowledge in $\mathcal{T}$. Unfortunately, query answer over OWL 2 DL ontologies is of very high computational complexity [12, 5] and even after modern optimisations and intense implementation efforts [7] OWL 2 DL systems are still not able to cope with datasets containing billions of data.

The need for efficient query answering has motivated the development of several fragments of OWL 2 DL [8], like OWL 2 EL, OWL 2 QL, and OWL 2 RL for which query answering can be implemented (at-most) in polynomial time with respect to the size of the data. Consequently, for many of these languages there already exist highly scalable systems which have been applied successfully to industrial-strength applications, like OWLim [6], Oracle's RDF Semantic Graph [15], and others. The attractive properties of these systems have led application developers to use them even in cases where the input ontology is expressed in the far more expressive OWL 2 DL language. Clearly, in such cases these systems would most likely be *incomplete*—that is, for some user query and dataset they will fail to compute all the certain answers. However, incomplete query answering may not be acceptable in several critical applications like healthcare or defense. As a result, techniques that attempt to deliver complete query answering even when using scalable systems that are not complete for OWL 2 DL have been proposed [13, 16].

Stoilos et al. [13] show how, given an OWL 2 RL system ans and a TBox $\mathcal{T}$, to compute a set of ontology axioms $\mathcal{R}$ (called *repair*) which is such that ans that is generally incomplete for $\mathcal{T}$ becomes complete for all ground queries and datasets when used with $\mathcal{T} \cup \mathcal{R}$. Zhou et al. [16] present a technique which uses ans to compute upper and lower bounds of the answers to a user query. If the two bounds coincide then the correct answers have been found while if they don't then an OWL 2 DL reasoner is used to check all intermediate possible answers. Unfortunately, both techniques are mainly applicable when the input ontology is expressed in the Horn fragment of OWL 2 DL (e.g., repairs might not exist for arbitrary OWL 2 DL ontologies).

Although systems complete for, e.g., OWL 2 RL, are generally incomplete for an OWL 2 DL ontology they might still be able to compute the correct answers to many user queries. In the current paper, we investigate whether given a (ground) query $\mathcal{Q}$ over an OWL 2 DL TBox $\mathcal{T}$ and a system ans complete for some fragment $\mathcal{L}$ of OWL 2 DL it is possible to identify in an efficient way if ans is complete for $\mathcal{Q}, \mathcal{T}$. We introduce the notion of a *query base* ($\mathcal{U}$) which consists of a set of atomic queries built from the symbols in $\mathcal{T}$ for which ans is known to be complete and we show how $\mathcal{U}$ can be used to conclude that ans is complete. Although our condition is only sufficient for deriving completeness there are, unfortunately, theoretical limitations for providing also a necessary condition. Nevertheless, to alleviate this issue we have designed a sufficient condition which can be used to check if ans is incomplete for a given query. With these two conditions combined we expect that we will be able to correctly identify (in)completeness of ans in most practical cases.

Subsequently, we show how $\mathcal{U}$ can be computed in practice using existing tools and then develop a (hybrid) query answering algorithm which uses the previous techniques to decide whether to evaluate an input query using a scalable system ans or a fully-fledged OWL 2 DL system. In the latter case, our algorithm can still exploit ans to a large extent in order to prune the search space of the OWL 2 DL system considerably. Finally, we have conducted an experimental evaluation which showed that for two well-known ontology benchmarks we were able to efficiently compute query bases, correctly identify the (in)completeness of an OWL 2 RL system for the vast majority of test queries and, moreover, that our hybrid query answering algorithm greatly outperformed a state-of-the-art OWL 2 DL system.

Compared to previous works on deciding completeness of incomplete systems [2], our main focus here is on efficiency and real-time query answering. Moreover, query bases is a novel notion and, interestingly, they are always guaranteed to exist. Hence, the techniques are readily applicable to arbitrary ontologies. Finally, our approach

[1] National Technical University of Athens, emails:gstoil@image.ntua.gr, gstam@cs.ntua.gr

is highly modular allowing any combination of system supporting a profile of OWL 2 DL with a fully-fledged reasoner and is not strongly tighted to OWL 2 RL ones.

2 PRELIMINARIES

We use standard notions from first-order logic, like variable, predicate, atom, constant, (Horn) clause, function symbols, satisfiability, and entailment ($\models$). We use $\vec{t}$ to denote a tuple of constants or variables of the form $(t_1, \ldots, t_n)$ where n is called the *arity* of $\vec{t}$. Moreover, for $\vec{a} = (a_1, \ldots, a_n)$ and $\vec{c}$ we write $\vec{c} \subseteq \vec{a}$ if for $j_1, .., j_m$ a sequence of positive integers such that $n \geq \max\{j_1, \ldots, j_m\}$ and $j_i < j_{i+1}$ we have $\vec{c} = (a_{j_1}, .., a_{j_m})$. Finally, for a set of atoms $\mathcal{B} = \{B_1, \ldots, B_m\}$, we denote with $\bigwedge \mathcal{B}$ the formula $B_1 \wedge \ldots \wedge B_m$.

Description Logic-based ontologies We assume basic familiarity with the DL syntax, semantics and standard reasoning problems, as well as their connection with OWL 2 DL [1]. In the rest of the paper with $\mathcal{L}$ we denote an arbitrary DL that is a fragment of the DL underpinning OWL 2 DL. Next, we recapitulate the DL $\mathcal{ELU}^{\bot}$ which is used in the examples and is a fragment of OWL 2 DL.

Let $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{I}$ be countable, pairwise disjoint sets of *atomic concepts*, *atomic roles*, and *individuals*. The set of $\mathcal{ELU}^{\bot}$-concepts is defined inductively as follows, where $A \in \mathbf{C}$, $R \in \mathbf{R}$, and $C_{(i)}$ are $\mathcal{ELU}^{\bot}$-concepts: $C := \top \mid \bot \mid A \mid C_1 \sqcap C_2 \mid \exists R.C \mid C_1 \sqcup C_2$. An $\mathcal{ELU}^{\bot}$-TBox $\mathcal{T}$ is a finite set of $\mathcal{ELU}^{\bot}$-axioms $C_1 \sqsubseteq C_2$, with C_i $\mathcal{ELU}^{\bot}$-concepts. An ABox $\mathcal{A}$ is a finite set of assertions of the form $A(a)$ or $R(a, b)$, for $A \in \mathbf{C}$, $R \in \mathbf{R}$, and $a, b \in \mathbf{I}$. An $\mathcal{ELU}^{\bot}$-ontology is a set $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$. For $\mathcal{S}$ a concept or TBox, we use $\mathsf{Sig}(\mathcal{S})$ to denote all atomic concepts and roles that appear in $\mathcal{S}$.

We also refer to Horn DLs, that is, fragments $\mathcal{L}_{\mathcal{H}}$ of OWL 2 DL where every $\mathcal{L}_{\mathcal{H}}$-TBox is logically equivalent to a set of Horn clauses (possibly with equality) of the form $B_1 \wedge \ldots \wedge B_n \rightarrow H$, where H is either a function-free atom or the symbol $\bot$, $B_1, \ldots, B_n$, are function-free atoms and all free variables are assumed to be universally quantified. Horn DLs form the logical underpinning of the tractable profiles OWL 2 QL, OWL 2 EL, and OWL 2 RL. For example, the OWL 2 EL axiom $A \sqsubseteq \exists R.\top$ can be transformed into $A(x) \rightarrow R(x, f(x))$.

Queries A *conjunctive query* (CQ) is a formula of the form $\exists \vec{y}.\phi(\vec{x}, \vec{y})$, where ϕ is a conjunction of function-free atoms containing only variables from $\vec{x}$ or $\vec{y}$, and $\vec{x}$ are free variables called *answer* variables. We use $\mathcal{Q}(\vec{x})$ to denote all the answer variables of $\mathcal{Q}$ and $\mathsf{bd}(\mathcal{Q})$ to denote the set of it atoms. Queries without existentially quantified variables form the basis of the W3C standard SPARQL[2] and in the following we will only consider such queries which we will call *SPARQL queries*. A tuple of constants $\vec{a}$ is a *certain answer* of a (SPARQL) query $\mathcal{Q}$ over $\mathcal{T} \cup \mathcal{A}$ if $\mathcal{T} \cup \mathcal{A} \models \phi(\vec{a})$. We denote with $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$ all the certain answers of $\mathcal{Q}$ over $\mathcal{T} \cup \mathcal{A}$.

Abstract query answering systems In the following, we recall the notions of a query answering system [13].

Definition 1 *A* (query answering) system ans *is a procedure that takes as input an OWL 2 DL-TBox $\mathcal{T}$, an ABox $\mathcal{A}$, and a CQ $\mathcal{Q}$ and returns a set of tuples $\mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$ that have the same arity as the answer variables in $\mathcal{Q}$.* ans *is called* $(\mathcal{Q}, \mathcal{T})$-complete *if for every $\mathcal{A}$ consistent with $\mathcal{T}$ we have $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$; otherwise, it is called* $(\mathcal{Q}, \mathcal{T})$-incomplete. *For $\mathcal{L}$ a fragment of OWL 2 DL and $\mathcal{T}$ an OWL 2 DL-TBox, $\mathcal{T}_{\mathcal{L}}$ denotes all $\mathcal{L}$-axioms of $\mathcal{T}$. Then,* ans *is called complete for $\mathcal{L}$ if for each CQ $\mathcal{Q}$ and ABox $\mathcal{A}$ we have $\mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \mathcal{T}_{\mathcal{L}} \cup \mathcal{A})$. In this case we refer to $\mathcal{T}_{\mathcal{L}}$ as the TBox that* characterises ans *over $\mathcal{T}$.*

Most query answering systems known to us can be captured by the above notion. For example, systems such as OWLim and Oracle's Semantic Graph are query answering systems complete for the OWL 2 RL fragment of OWL 2 DL.

3 CHECKING COMPLETENESS OF SYSTEMS

In the current section, we investigate whether it is possible to identify in an efficient way if a system complete for a DL $\mathcal{L}$ is complete for a given (SPARQL) query over an OWL 2 DL TBox.

Example 2 Consider the TBox $\mathcal{T} = \{\exists S.C \sqsubseteq B, A \sqsubseteq D\}$ and consider also a system ans characterised by the TBox $\mathcal{T}_{\mathcal{L}} = \{A \sqsubseteq D\}$, i.e., ans cannot handle axioms of the form $\exists S.C \sqsubseteq B$.[3]

Clearly, for $\mathcal{Q}_1 = S(x, y)$ and $\mathcal{Q}_2 = D(x)$ ans is $(\mathcal{Q}_1, \mathcal{T})$- and $(\mathcal{Q}_2, \mathcal{T})$-complete, while it can also be verified that for $\mathcal{Q} = S(x, y) \wedge D(x)$ it is $(\mathcal{Q}, \mathcal{T})$-complete since for every ABox $\mathcal{A}$ we have $\mathsf{cert}(\mathcal{Q}, \mathcal{T}_{\mathcal{L}} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$.

As it can be seen, $(\mathcal{Q}, \mathcal{T})$-completeness of ans is rather expected since $\mathcal{Q}$ is formed by atoms $S(x, y)$ and $D(x)$ which precisely correspond to queries $\mathcal{Q}_1$ and $\mathcal{Q}_2$, and that we have already established that ans is $(\mathcal{Q}_1, \mathcal{T})$- and $(\mathcal{Q}_2, \mathcal{T})$-complete. ♢

The above example suggests that given a set of atomic queries over which ans is known to be complete, then we can deduce the completeness of ans w.r.t. an arbitrary SPARQL query $\mathcal{Q}$ by checking if for each of its atoms there is a "matching" query in the set. We call such set of queries a query base.

Definition 3 *Let $\mathcal{T}$ be an OWL 2 DL-TBox and let* ans *be a system. A* query base *(QB) of* ans *for $\mathcal{T}$ is a finite set of constant-free atomic queries $\mathcal{U}$ built from the symbols in* $\mathsf{Sig}(\mathcal{T})$ *such that if $\mathcal{Q} \in \mathcal{U}$, then* ans *is $(\mathcal{Q}, \mathcal{T})$-complete.*

Without loss of generality and to simplify the presentation we often say "an atom α of a query *appears in* $\mathcal{U}$" meaning that "there is a query $\mathcal{Q}_1 \in \mathcal{U}$ and an isomorphism σ from the terms of $\mathcal{Q}_1$ to those of α such that $\mathcal{Q}_1\sigma = \alpha$".

Towards identifying a condition for deducing completeness of a system ans for a query $\mathcal{Q}$ given its QB, the following example shows that even if there exists an atom in $\mathcal{Q}$ that is not in the given QB, we might still be able to correctly recognise that ans is $(\mathcal{Q}, \mathcal{T})$-complete.

Example 4 Let the following TBox $\mathcal{T}$ and query $\mathcal{Q}$:

$$\mathcal{T} = \{\exists R.C \sqsubseteq A, B \sqsubseteq A\} \qquad \mathcal{Q} = A(x) \wedge B(x)$$

and consider again the system ans from Example 2.

First, note that ans is $(\mathcal{Q}, \mathcal{T})$-complete: for any ABox $\mathcal{A}$ such that for some individual a we have $\mathcal{T} \cup \mathcal{A} \models \mathcal{Q}(a)$, $\mathcal{A}$ must contain the assertion $B(a)$; but then, ans is characterised by $\mathcal{T}_{\mathcal{L}} = \{B \sqsubseteq A\}$, hence also $\mathcal{T}_{\mathcal{L}} \cup \mathcal{A} \models A(a)$. Consequently, for any $\mathcal{A}$ we have $\mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) = \mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$.

Second, assume that a QB of ans for $\mathcal{T}$ is given that contains only the query $\mathcal{Q}_1 = B(x)$. Even though $\mathcal{Q}$ contains an atom $A(x)$ that does not appear in $\mathcal{U}$ it is still possible to identify that ans is $(\mathcal{Q}, \mathcal{T})$-complete as follows: first, we can note that for the other atom of the query (i.e., $B(x)$) we have $\mathcal{T}_{\mathcal{L}} \models B(x) \rightarrow A(x)$ and, second, that $B(x)$ is in $\mathcal{U}$. ♢

[2] http://www.w3.org/TR/rdf-sparql-query/

[3] In Semantic Web terms ans is complete for RDFS.

The previous example suggests that, it is sufficient that some of the atoms of $\mathcal{Q}$ are only "covered" by the presence of other atoms which appear in the given QB. Even more, it is interesting to note that ans might even be incomplete for the atomic queries that correspond to the atoms that need to be covered. In the previous example, although for $\mathcal{Q}_2 = A(x)$ ans is $(\mathcal{Q}_2, \mathcal{T})$-*incomplete* and $A(x)$ appears in $\mathcal{Q}$, ans is, however, $(\mathcal{Q}, \mathcal{T})$-complete. The notion of covering is formalised next.

Definition 5 *Let* ans *be a system complete for a DL $\mathcal{L}$, let $\mathcal{Q}$ be a CQ, let $\mathcal{U}$ be a QB of* ans *for an OWL 2 DL-TBox $\mathcal{T}$, and let $\mathcal{T}_\mathcal{L}$ be the TBox that characterises* ans *over $\mathcal{T}$. Let also $\mathcal{B}$ be all atoms in $\mathcal{Q}$ which appear in $\mathcal{U}$. We say that an atom α in $\mathcal{Q}$ is* covered *by $\mathcal{U}$ if either α appears in $\mathcal{U}$ or $\mathcal{T}_\mathcal{L} \models \bigwedge \mathcal{B} \rightarrow \alpha$.*[4]

Using the notion of covering we can show the following result.

Theorem 6 *Let $\mathcal{T}$ be an OWL 2 DL-TBox, let* ans *be a system complete for a DL $\mathcal{L}$, let $\mathcal{U}$ be a QB of* ans *for $\mathcal{T}$, and let $\mathcal{Q}$ be a SPARQL query. If each atom α in $\mathcal{Q}$ is covered by $\mathcal{U}$, then* ans *is $(\mathcal{Q}, \mathcal{T})$-complete.*

Proof. Let $\mathcal{Q}$ be a SPARQL CQ with $\vec{x}$ its answer variables, let $\mathcal{A}$ be an arbitrary ABox such that for some tuple of individuals $\vec{a}$ we have $\mathcal{T} \cup \mathcal{A} \models \mathcal{Q}(\vec{a})$ and assume that each atom of $\mathcal{Q}$ is covered by $\mathcal{U}$. Since $\mathcal{Q}$ is SPARQL, we have that $\mathcal{T} \cup \mathcal{A} \models \alpha_i(\vec{a}_i)$ for each atom $\alpha_i(\vec{x}_i)$ in $\mathcal{Q}$, where $\vec{x}_i$ are variables from $\vec{x}$. Let also $\mathcal{T}_\mathcal{L}$ be the TBox that characterises ans over $\mathcal{T}$.

Now, consider an arbitrary atom $\alpha_k(\vec{x}_k)$ in $\mathcal{Q}$ such that $\mathcal{T} \cup \mathcal{A} \models \alpha_k(\vec{a}_k)$. If $\alpha_k(\vec{x}_k)$ appears in $\mathcal{U}$ then we clearly have $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \models \alpha_k(\vec{a}_k)$. The interesting case is if for the set $\mathcal{B}$ defined as in Definition 5 we have $\mathcal{T}_\mathcal{L} \models \bigwedge \mathcal{B} \rightarrow \alpha_k(\vec{x}_k)$. Since $\mathcal{B}$ are atoms of $\mathcal{Q}$ we must have $\mathcal{T} \cup \mathcal{A} \models \mathcal{B}(\vec{c})$ where $\vec{c} \subseteq \vec{a}$. Moreover, again by assumption, we have that each $\beta_i \in \mathcal{B}$ appears in $\mathcal{U}$; hence we must also have $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \models \bigwedge \mathcal{B}(\vec{c})$. Thus, it also follows that $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \models \alpha_k(\vec{a}_k)$.

Consequently, since $\alpha_k(\vec{x}_k)$ was arbitrarily chosen we must have $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \models \alpha_i(\vec{a}_i)$ for each $\alpha_i(\vec{a}_i) \in \mathcal{Q}(\vec{a})$ and hence also $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \models \mathcal{Q}(\vec{a})$. Moreover, also $\vec{a}$ and $\mathcal{A}$ were arbitrary, hence for each $\mathcal{A}$ we must have $\mathsf{cert}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A}) \subseteq \mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$. □

Ideally, $\mathcal{U}$ should contain all atomic queries from $\mathsf{Sig}(\mathcal{T})$ for which ans is complete. Unfortunately, as the following example shows, even in this case covering as defined previously provides a sufficient but not necessary condition for $(\mathcal{Q}, \mathcal{T})$-completeness.

Example 7 Consider the following TBox $\mathcal{T}$:

$$B_1 \sqsubseteq \exists S.\top \quad \exists S.\top \sqsubseteq A_1 \quad B_1 \sqcap A \sqsubseteq A_1$$

and consider also an OWL 2 RL system ans. Then, over $\mathcal{T}$, ans is characterised by $\mathcal{T}_\mathcal{L} = \{\exists S.\top \sqsubseteq A_1, B_1 \sqcap A \sqsubseteq A_1\}$. Clearly, for the atomic query $\mathcal{Q}_1 = A_1(x)$, ans is $(\mathcal{Q}_1, \mathcal{T})$-incomplete as witnessed by the ABox $\mathcal{A}_1 = \{B_1(a)\}$. Hence, $\mathcal{Q}_1$ cannot be in any QB $\mathcal{U}$ of ans for $\mathcal{T}$. More precisely, $\mathcal{U}$ can consist at most of the queries $\mathcal{Q}_2 = S(x, y)$, $\mathcal{Q}_3 = A(x)$, and $\mathcal{Q}_4 = B_1(x)$.

Consider now the query $\mathcal{Q} = A_1(x) \wedge A(x)$. As can be seen, the atom $A_1(x)$ of $\mathcal{Q}$ is not covered by $\mathcal{U}$: first, $A_1(x)$ cannot be in $\mathcal{U}$ and, second, for the only other atom of $\mathcal{Q}$ (i.e., $A(x)$) we have $\mathcal{T}_\mathcal{L} \not\models A(x) \rightarrow A_1(x)$.

However, it can be seen that ans is $(\mathcal{Q}, \mathcal{T})$-complete. First, we observe that any ABox $\mathcal{A}$ that provides an answer to $\mathcal{Q}$, i.e., $\mathcal{T} \cup \mathcal{A} \models \mathcal{Q}(a)$, must contain an assertion of the form $A(a)$. Moreover, $\mathcal{A}$ must be such that $\mathcal{T} \cup \mathcal{A} \models A_1(a)$. Due to the latter, $\mathcal{A}$ must contain one of the assertions $A_1(a)$, or $S(a, b)$, or $B_1(a)$. In all cases we can also see that $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \models A(a) \wedge A_1(a)$. Especially for $\mathcal{A} = \{A(a), B_1(a)\}$, i.e., the ABox that contains the witness for the incompleteness of ans for $\mathcal{Q}_1, \mathcal{T}$, we have $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \models A_1(a)$ since $B_1 \sqcap A \sqsubseteq A_1 \in \mathcal{T}_\mathcal{L}$. ♢

Intuitively, the issue highlighted in the previous example is that although the ABox $\mathcal{A}_1 = \{B_1(a)\}$ witnesses the incompleteness of ans w.r.t. $\mathcal{Q}_1$, when $\mathcal{A}_1$ is taken together with additional assertions that provide an answer to the second atom of the query (i.e., $A(x)$) it ceases from being a witness of incompleteness. This suggests that to obtain a sufficient and necessary condition we additionally need to pre-compute all atomic queries $\mathcal{Q} = \alpha$ for which ans is incomplete, as well as, all witnesses $\mathcal{A}_\mathcal{Q}$ of this incompleteness. Then, at run-time, if atom α appears in some user query we need to check if for every witness $\mathcal{A}_\mathcal{Q}$ we have $\mathcal{T}_\mathcal{L} \cup \mathcal{A}_\mathcal{Q} \models (\bigwedge \mathcal{B} \rightarrow \alpha)\pi$, where $\mathcal{B}$ is as in Definition 5 and π is a mapping related to the construction of $\mathcal{A}_Q$. In Example 7, for the witness $\mathcal{A}_\mathcal{Q} = \{B_1(a)\}$ we have $\mathcal{T}_\mathcal{L} \cup \mathcal{A}_\mathcal{Q} \models (A(x) \rightarrow A_1(x))\{x \mapsto a\}$. Unfortunately we know from theory that this set of witnesses can be infinite [2, 3] (see also Example 10 next) and moreover, even if it is finite, checking a condition like the one above at run-time will not be very practical as it requires performing a (possibly) very large number of entailment tests.

To alleviate the above issue, in the next section, we design a condition that can be easily checked in practice and which implies incompleteness of ans. Since real-world ontologies are expected to rarely contain combinations of axioms like the ones depicted in Example 7, in practical scenarios this condition combined with the techniques presented in this section are expected to leave only very few unknown cases.

4 CHECKING INCOMPLETENESS

In the current section, we provide a condition which is necessary in order to have $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \models \bigwedge \mathcal{B} \rightarrow \alpha$ for every $\mathcal{A}$. This implies that, if the condition does not hold, then we can deduce that the system under consideration is incomplete.

Our syntactic condition is based on the notion of reachability between the symbols of a TBox $\mathcal{T}$, which is a well-known notion that has been used extensively in the past in other contexts [14, 11].

Definition 8 *Let $\mathcal{T}$ be an OWL 2 DL-TBox and $\boldsymbol{S} \subseteq \mathsf{Sig}(\mathcal{T})$ a signature. The set of $\boldsymbol{S}$-reachable names in $\mathcal{T}$ is defined inductively as follows: (i) x is $\boldsymbol{S}$-reachable in $\mathcal{T}$, for every $x \in \boldsymbol{S}$; and (ii) for all inclusion axioms $C_L \sqsubseteq C_R$, if for some $x \in \mathsf{Sig}(C_L)$ x is $\boldsymbol{S}$-reachable in $\mathcal{T}$, then y is $\boldsymbol{S}$-reachable in $\mathcal{T}$ for every $y \in \mathsf{Sig}(C_R)$.*

Reachability provides a necessary condition for entailment. For example, if $\mathcal{T} \models A \sqsubseteq B$ then B must be $\{A\}$-reachable in $\mathcal{T}$. Hence, non-reachability will guarantee that $\mathcal{T}_\mathcal{L} \cup \mathcal{A} \not\models \bigwedge \mathcal{B} \rightarrow \alpha$.

Finally, the following property (♯) on CQs $\mathcal{Q} = \alpha_1(\vec{x}_1) \wedge \ldots \wedge \alpha_n(\vec{x}_n)$ is additionally required to prove our next result: for every $\mathcal{A}$ and $1 \leq i \leq n$ with $\mathcal{T} \cup \mathcal{A} \models \alpha_i(\vec{a})$, the following ontology is consistent $\mathcal{T} \cup \mathcal{A} \cup \{\alpha_j(\vec{x}_j)\iota \mid \alpha_j \in \mathsf{bd}(\mathcal{Q}), \alpha_j \neq \alpha_i, \iota$ injective mapping from each variable of $\mathcal{Q}$ different than $\vec{x}_i$ to a fresh individual$\}$.

Theorem 9 *Let $\mathcal{L}_\mathcal{H}$ be a Horn DL, let* ans *be a system that is complete for $\mathcal{L}_\mathcal{H}$, let $\mathcal{Q}$ be a CQ satisfying (♯), let $\mathcal{T}$ be an OWL 2 DL-TBox, let $\mathcal{U}$ be a QB of* ans *for $\mathcal{T}$, and let $\mathcal{B}$ be all the atoms in $\mathcal{Q}$*

[4] The reader is referred to [13] for details about how the entailment relation $\mathcal{T}_\mathcal{L} \models \bigwedge \mathcal{B} \rightarrow \alpha$ can be checked in practice by treating ans as a black box.

which appear in $\mathcal{U}$. Finally, let α be an atom of $\mathcal{Q}$ that is not covered by $\mathcal{U}$. If α is not $\mathsf{Sig}(\mathcal{B})$-reachable in $\mathcal{T}_{\mathcal{L}_\mathcal{H}}$, then ans *is $(\mathcal{Q},\mathcal{T})$-incomplete.*

Proof. Consider the query $\mathcal{Q}_\alpha = \alpha(\vec{x})$. By assumption $\mathcal{Q}_\alpha$ is not in $\mathcal{U}$ hence ans is $(\mathcal{Q}_\alpha,\mathcal{T})$-incomplete. This implies that there exists some ABox $\mathcal{A}_\alpha$ and tuple of individuals $\vec{a}$ from $\mathcal{A}_\alpha$ such that $\vec{a} \in \mathsf{cert}(\mathcal{Q}_\alpha,\mathcal{T}\cup\mathcal{A}_\alpha)$ but $\vec{a} \notin \mathsf{ans}(\mathcal{Q}_\alpha,\mathcal{T}\cup\mathcal{A}_\alpha)$. Since ans is complete for $\mathcal{L}_\mathcal{H}$ then its behaviour is characterised by $\mathcal{T}_{\mathcal{L}_\mathcal{H}}$; hence, $\mathcal{T}_{\mathcal{L}_\mathcal{H}} \cup \mathcal{A} \models \mathcal{Q}_\alpha(\vec{a})$ iff $\vec{a} \in \mathsf{ans}(\mathcal{Q}_\alpha,\mathcal{T}\cup\mathcal{A})$ and, in the following, we can use $\mathcal{T}_{\mathcal{L}_\mathcal{H}}$ and $\models$ instead of ans. Next, we will extend $\mathcal{A}_\alpha$ to $\mathcal{A}$ such that for $\vec{a} \subseteq \vec{c}$, we have $\mathcal{T}\cup\mathcal{A} \models \mathcal{Q}(\vec{c})$ but $\mathcal{T}_{\mathcal{L}_\mathcal{H}} \cup \mathcal{A} \not\models \mathcal{Q}(\vec{c})$ which will imply that $\vec{c} \notin \mathsf{ans}(\mathcal{Q},\mathcal{T}\cup\mathcal{A})$.

Assume that $\sigma = \{\vec{x} \mapsto \vec{a}\}$ and let ι be an injective mapping from all variables of $\mathcal{Q}$ that do not appear in $\vec{x}$ to fresh individuals. Then, for $\pi = \sigma \cup \iota$, let $\mathcal{A} := \mathcal{A}_\alpha \cup \{\alpha_i(\vec{x}_i)\pi \mid \alpha_i \text{ in } \mathcal{Q} \text{ different than } \alpha\}$. Clearly, $\mathcal{A} \supseteq \mathcal{A}_\alpha$ and by property (♯), $\mathcal{T}\cup\mathcal{A}$ is consistent. Moreover, for some tuple of individuals $\vec{c}$ from the range of π we have $\mathcal{T}\cup\mathcal{A} \models \mathcal{Q}(\vec{c})$. Moreover, for each atom α_i different than α we clearly have $\mathcal{T}_{\mathcal{L}_\mathcal{H}} \cup \mathcal{A} \models \alpha_i(\vec{x}_i)\pi$.

Assume in contrast that $\mathcal{T}_{\mathcal{L}_\mathcal{H}} \cup \mathcal{A} \models \mathcal{Q}(\vec{c})$. Since $\mathcal{T}_{\mathcal{L}_\mathcal{H}} \cup \mathcal{A}_\alpha \not\models \alpha(\vec{a})$ and $\mathcal{A} \supseteq \mathcal{A}_\alpha$, then $\mathcal{T}_{\mathcal{L}_\mathcal{H}} \cup \mathcal{A} \models \mathcal{Q}(\vec{c})$ can only be the case if $\mathcal{T}_{\mathcal{L}_\mathcal{H}} \cup \mathcal{A} \models \alpha(\vec{a})$. By definition, $\mathcal{L}_\mathcal{H}$ is logically equivalent to a set of Horn clauses (possibly with equality) $\mathcal{H}$. For this set we have $\mathcal{H} \cup \mathcal{A} \models \alpha(\vec{a})$ and we can use SLD-resolution with backwards chaining starting with the goal $\alpha(\vec{a})$ in order to derive the empty clause (equality can be axiomatised and treated as a regular predicate and, moreover, since $\mathcal{T}\cup\mathcal{A}$ is consistent the $\bot$ predicate is not involved). The derivation will use as side premises Horn clauses from $\mathcal{H}$ of the form $B_1 \wedge \ldots \wedge B_n \rightarrow H$, where some goal will unify with the atom H and it will create a new goal that contains all atoms B_i. Finally, to derive the empty clause the atoms that are introduced (the B_i's) should be eliminated by using as side premises facts from $\mathcal{A}$ which, by construction, contains assertions for all atoms in $\mathsf{Sig}(\mathcal{B})$. It follows that these atoms are introduced as goals due to some rule in which they appear in the body. Hence, $\mathcal{T}_{\mathcal{L}_\mathcal{H}}$ must also contain axioms $C_L \sqsubseteq C_R$ where atoms from $\mathcal{B}$ appear in C_L which implies that all atoms in $\mathcal{B}$ reach α in $\mathcal{T}_{\mathcal{L}_\mathcal{H}}$. □

5 COMPUTING QUERY BASES IN PRACTICE

Since a TBox $\mathcal{T}$ has a finite signature and for every atomic query $\mathcal{Q}$ a system ans is either complete or not, it follows trivially that the query base always exists. In general, one could construct it by first extracting a (hopefully) small subset of $\mathcal{T}$ that is relevant to answer $\mathcal{Q}$ over any ABox (e.g., a module [11]) and then contrast its constructors with the language that ans supports. However, this process can be very labour intensive, hence, in the following we will show how we can reuse existing technology to assist and speed up this process.

Clearly, the main problem is to use (semi-)automatic methods to check (in)completeness of a system w.r.t. every (atomic) query $\mathcal{Q}$ built from the signature of a TBox. Checking (in)completeness of a system w.r.t. an arbitrary query has been studied before in the literature [2]. For a given query $\mathcal{Q}$ and TBox $\mathcal{T}$ it has been shown how to devise a set of tests (called *test suite*) of the form $\Im = \{\langle\mathcal{A}_1,\mathcal{Y}_1\rangle,\ldots,\langle\mathcal{A}_n,\mathcal{Y}_n\rangle\}$, where each $\mathcal{A}_i$ is a small ABox and each $\mathcal{Y}_i$ is a query (possibly different from $\mathcal{Q}$). In addition, the test suite should satisfy the following desirable property:

(♦): if for each $\langle\mathcal{A}_i,\mathcal{Y}_i\rangle \in \Im$ we have $\mathsf{cert}(\mathcal{Y}_i,\mathcal{T}\cup\mathcal{A}_i) \subseteq \mathsf{ans}(\mathcal{Y}_i,\mathcal{T}\cup\mathcal{A}_i)$, then ans is $(\mathcal{Q},\mathcal{T})$-complete.

The problem has been studied for various classes of query answering systems, including systems complete for Horn DLs, providing sufficient conditions for the existence of $\Im$ and practical algorithms for computing it. Unfortunately, there exist TBoxes where the test suite can satisfy property (♦) only if it is infinite. However, as the following example shows, we might still be able to compute a QB.

Example 10 Consider the following TBox $\mathcal{T}$:

$$\top \sqsubseteq G \sqcup B \quad \exists E.G \sqsubseteq B \quad \exists E.B \sqsubseteq G$$

and let ans be an OWL 2 RL system. Then, the set $\mathcal{T}_\mathcal{L}$ that characterises ans over $\mathcal{T}$ consists of the last two axioms of $\mathcal{T}$.

Assume next that we want to determine using the techniques in [2] whether ans is $(\mathcal{Q}_1,\mathcal{T})$-complete for $\mathcal{Q}_1 = G(x)$. It is not hard to see that, for any odd integer $i \geq 2$, the ABox $\mathcal{A}_i = \{E(a_1,a_2),\ldots,E(a_{i-1},a_i),E(a_1,a_i)\}$ provides an answer to $\mathcal{Q}_1$—that is, $\mathsf{cert}(\mathcal{Q}_1,\mathcal{T}\cup\mathcal{A}_i) = \{a_1\}$. Hence, according to [2] a test suite $\Im$ satisfying property (♦) must contain an infinite umber of tests of the form $\langle\mathcal{A}_i,\mathcal{Q}_1\rangle$.

However, for $i = 3$ it can be verified that $\mathsf{cert}(\mathcal{Q},\mathcal{T}_\mathcal{L}\cup\mathcal{A}_3) = \emptyset$ since to compute the answer a_1 a system needs to be able to reason over the disjunctive axiom of $\mathcal{T}$. Hence, we can conclude that ans is $(\mathcal{Q}_1,\mathcal{T})$-incomplete, without needing to consider additional tests.

In a similar way, for $\mathcal{Q}_2 = B(x)$ we can identify that ans is $(\mathcal{Q}_2,\mathcal{T})$-incomplete, while finally, that for $\mathcal{Q}_3 = E(x,y)$ it is $(\mathcal{Q}_3,\mathcal{T})$-complete. Consequently, $\mathcal{U}$ can only contain $\mathcal{Q}_3$. ♢

In the previous example, the situation would have been different if ans was complete for the test $\langle\mathcal{A}_3,\mathcal{Q}_1\rangle$. Then, we would need to try (possibly) all tests for every odd $i \geq 2$ and hence we would have never been able to say with certainty whether ans is $(\mathcal{Q}_1,\mathcal{T})$-complete. However, since in Horn TBoxes the test suite always exists (by the results in [4] and [2] it follows that it is always finite), then the source of infiniteness in the non-Horn case is related to the interaction of the disjunctive axiom with other axioms of the TBox. Since systems complete for one of the tractable profiles of OWL 2 DL are inherently incomplete for inferences involving such constructors, then in most practical cases they are expected to fail already for the smallest possible test.

6 HYBRID QUERY ANSWERING

The straightforward approach to exploit our proposed techniques is to use them at running time to decide if a given query $\mathcal{Q}$ can be evaluated using some very scalable system ans or we need to resort to an OWL 2 DL reasoner. In the current section, we take this step further and we show that even in the latter case ans can still be used to (possibly) speed up query evaluation significantly.

Our idea is based on the fact that to compute the answers of a SPARQL query $\mathcal{Q}$ one can compute the answers of each atomic query $\mathcal{Q}_\alpha = \alpha$ with $\alpha \in \mathsf{bd}(\mathcal{Q})$ and then construct the answer by joining all the results. This implies that the evaluation of $\mathcal{Q}$ can be split into the part for which the system ans is known to be complete and the one that is not. Then, the OWL 2 DL reasoner is only applied on the second part, which is perhaps easier to evaluate than the whole query. Moreover, ans can be used to restrict the search space of the OWL 2 DL reasoner.

Our hybrid query answering algorithm is presented in Algorithm 1. Internally it is using a system ans complete for some fragment $\mathcal{L}$ of OWL 2 DL and (possibly) an OWL 2 DL reasoner (function getInstances). It accepts as input a SPARQL CQ $\mathcal{Q}$, a TBox $\mathcal{T}$,

Algorithm 1 HYBRIDQA$(\mathcal{Q}, \mathcal{T}, \mathcal{A}, \mathcal{U})$

Input: A SPARQL CQ $\mathcal{Q}$ with $\vec{x}$ its answer variables, an OWL 2 DL-TBox $\mathcal{T}$, an ABox $\mathcal{A}$, and a QB $\mathcal{U}$ of the system ans used internally below for $\mathcal{T}$.

$\mathcal{B}$:= atoms of $\mathcal{Q}$ that appear in $\mathcal{U}$
$\mathcal{C} := \{\alpha \in \mathcal{Q} \mid \alpha$ neither cov. by $\mathcal{U}$ nor $\mathsf{Sig}(\mathcal{B})$-reach. in $\mathcal{T}_\mathcal{L}\}$
if $\mathcal{C} = \emptyset$ **then**
  return $\mathsf{ans}(\mathcal{Q}, \mathcal{T} \cup \mathcal{A})$
else
  $\mathsf{UpBnd} := \{a \mid a$ appears in $\mathcal{A}\}^n$ where n is the arity of $\vec{x}$
  $\mathcal{Q}'$:= new CQ that contains all $\alpha \in \mathsf{bd}(\mathcal{Q})$ s.t. $\alpha \notin \mathcal{C}$
  if $\mathcal{Q}' \neq \emptyset$ **then** $\mathsf{UpBnd} := \mathsf{ans}(\mathcal{Q}', \mathcal{T} \cup \mathcal{A})$
  for all $\alpha \in \mathcal{C}$ **do**
    $\mathsf{PAns} := \mathsf{ans}(\alpha, \mathcal{T} \cup \mathcal{A})$
    $\mathsf{Ans}_\alpha := \mathsf{getInstances}(\alpha, \mathcal{T} \cup \mathcal{A}, \mathsf{UpBnd}, \mathsf{PAns})$
    $\mathsf{UpBnd} := \{\vec{a} \in \mathsf{UpBnd} \mid \exists \vec{c} \in \mathsf{Ans}_\alpha \cup \mathsf{PAns}, \vec{c} \subseteq \vec{a}\}$
  end for
  return UpBnd
end if

an ABox $\mathcal{A}$, and a QB $\mathcal{U}$ for the system ans that has been computed previously. It then proceeds as follows. First, it collects all atoms of $\mathcal{Q}$ that are neither covered by $\mathcal{U}$ according to Definition 5 nor reachable according to Theorem 9 (set $\mathcal{C}$). If $\mathcal{C}$ is empty, then ans can be used to evaluate $\mathcal{Q}$ over $\mathcal{T} \cup \mathcal{A}$; otherwise the algorithm enters the else-block where it also uses an OWL 2 DL reasoner. More precisely, it extracts from $\mathcal{Q}$ the part for which ans is complete and evaluates it over $\mathcal{T} \cup \mathcal{A}$. This provides an upper bound of the answer (UpBnd) since the conjuncts in $\mathcal{C}$ have not been considered. Then, for each atom $\alpha \in \mathcal{C}$ it uses the OWL 2 DL reasoner to retrieve its instances. To further speed up this procedure, two additional parameters are passed to the function getInstances. The first one is the upper bound and the second one are (some) known instances of α computed again using ans (PAns). Then, the search of the OWL 2 DL reasoner is restricted to only those individuals that are in UpBnd and not in PAns. Finally, tuples in UpBnd for which α does not hold are pruned.

Correctness of HYBRIDQA follows by our previous results as well as the use of an OWL 2 DL reasoner for the rest of the atoms of $\mathcal{Q}$.

Proposition 11 HYBRIDQA$(\mathcal{Q}, \mathcal{T}, \mathcal{A}, \mathcal{U})$ *returns the certain answers of a SPARQL CQ $\mathcal{Q}$ over $\mathcal{T} \cup \mathcal{A}$.*

7 EVALUATION

We have implemented a prototype tool, called Hydrowl,[5] which can be used to extract query bases for incomplete systems and check their (in)completeness over a given query using the notions of covering and reachability. If none of the techniques apply then the tool replies "unknown". Our current implementation supports the well-known incomplete system OWLim and it is internally using the system SyGENiA[6] to check completeness w.r.t. atomic queries and construct the QB. However, note that other systems can be easily supported.

We used Hydrowl to compute a query base of OWLim for the two well-known ontology benchmarks LUBM[7] and UOBM.[8] For LUBM the tool required 14.5 seconds and returned a QB containing 40 atomic queries (LUBM has 43 concept names). We have verified that the computed QB contains all atomic queries for which OWLim is complete. For UOBM an initial QB was computed in 48.7 seconds but due to expressivity restrictions of SyGENiA two atomic queries had to be added manually. In total the computed QB contained 59 atomic queries (UOBM has 69 concept names). Subsequently, we used our implementation to check completeness of OWLim for all the test queries of LUBM and UOBM.

Regarding LUBM, all atoms of the queries 1–5 and 11–14 appear in the computed QB and hence according to Theorem 6 OWLim is complete for them (indeed this was also veryfied using SyGENiA). Moreover, for these queries our tool replied "complete" almost instantaneously (less than 5ms), hence we do not report the times in detail. All other queries contain the atom $\mathsf{Student}(x)$ and it is well-known that OWLim is (generally) not complete for this atomic query over LUBM since it contains the axioms $\mathsf{GraduateStudent} \sqsubseteq \exists \mathsf{takesCourse}.\mathsf{Course}$ and $\exists \mathsf{takesCourse}.\mathsf{Course} \sqsubseteq \mathsf{Student}$. Consequently, in these queries the algorithm proceeds in checking if this atom can be covered or is not reachable. The results are depicted in Table 1, where we give the time (in milliseconds) required by our tool (row t), whether the atom is reachable (row $\rightsquigarrow$), whether it is covered, and finally whether OWLim is actually complete or not for the query (checked again using SyGENiA).

Table 1. Results for the LUBM TBox.

	6	7	8	9	10
t	0	47	1	47	43
$\rightsquigarrow$	×	✓	×	✓	✓
Covered	-	×	-	✓	×
Complete	no	no	no	yes	no

First, we can see that in all cases Hydrowl required less than 50ms to reply. Second, for queries 6 and 8 the atom $\mathsf{Student}(x)$ is not reachable by other atoms and hence we can immediately conclude its incompleteness without checking covering. Since this can be done very efficiently the tool again replied almost instantaneously. Third, in query 9 the atom is covered due to the following implication, for $\mathcal{T}_\mathcal{L}$ the set that characterises OWLim over LUBM:

$$\mathcal{T}_\mathcal{L} \models \mathsf{advisor}(x, y) \wedge \mathsf{takesCourse}(x, z) \wedge \mathsf{Course}(z) \rightarrow \mathsf{Student}(x)$$

Hence, by Theorem 6 OWLim is complete for query 9. However, in queries 7 and 10 (highlighted by gray in the table) the atom $\mathsf{Student}(x)$ is reachable but not covered; hence, our tool replied "unknown". Interestingly, we can see that for these queries OWLim is incomplete, hence using in addition a complete reasoner in query answering would not introduce an unnecessary overhead.

Regarding UOBM, again for the queries where all atoms have an exact match in the computed QB the tool replied "complete" almost instantaneously. For the rest, the results are depicted in Table 2. Like in LUBM we can see that in all cases the tool replied very quickly. In particular, for the queries were there are atoms than are not reachable, i.e., queries 2, 3, 11, 13, 14, and 15, the tool replied "incomplete" almost instantaneously. Next, queries 6, 7, and 12, contain atoms that are reachable and also covered, hence the tool correctly replied "complete". The relevant entailments for these queries are the following:

query 6 $\quad \mathcal{T}_\mathcal{L} \models \mathsf{hasAlumnus}(y, x) \rightarrow \mathsf{Person}(x)$
query 7 $\quad \mathcal{T}_\mathcal{L} \models \mathsf{hasSameHomeTownWith}(x, y) \rightarrow \mathsf{Person}(x)$
query 12 $\quad \mathcal{T}_\mathcal{L} \models \mathsf{takesCourse}(x, y) \rightarrow \mathsf{Student}(x)$

[5] http://www.image.ece.ntua.gr/ gstoil/hydrowl/
[6] http://code.google.com/p/sygenia/
[7] http://swat.cse.lehigh.edu/projects/lubm/
[8] http://www.cs.ox.ac.uk/isg/tools/UOBMGenerator/

The only query that our tool replied "unknown" was query 8, where again OWLim is incomplete.

Table 2. Results for the UOBM TBox

	2	3	6	7	8	11	12	13	14	15
t	0	0	36	37	44	0	39	1	0	0
$\rightsquigarrow$	×	×	✓	✓	✓	×	✓	×	×	×
Cov	-	-	✓	✓	×	-	✓	-	-	-
Com	no	no	yes	yes	no	no	yes	no	no	no

Finally, we have also implemented Algorithm 1 in Hydrowl (using OWLim and the standard HermiT reasoner [10]) and we have used it to answer all test queries of LUBM and UOBM. We have created datasets for 5 and 10 universities for LUBM and for 1 department and 1 university for UOBM, and we have compared against the HermiT-SPARQL system (H-QL) [7], an implementation of the SPARQL OWL-DL entailment regime in HermiT.

Table 3 presents the results for some interesting queries. The behaviour of H-QL can be highly non-deterministic, hence we have taken an average over several runs. With grey color we have marked the queries where Algorithm 1 enters the else-block and hence uses both OWLim and HermiT. As can be seen, in all queries H-QL requires several seconds (even up to minutes) to compute the answers. In contrast Hydrowl computed the correct answers within milliseconds in all but query 14. Moreover, H-QL could also not manage to load any of the large datasets (10 universities for LUBM and 1 university for UOBM) after 1 hour. This is mostly because H-QL uses HermiT at pre-processing to materialise many entailments. In contrast, loading in Hydrowl always took less than 5 minutes and query answering was again quite efficient (apart from query 14 where we aborted after 15 minutes).

Finally, note that for the queries we have not reported times H-QL and Hydrowl have similar response times. In some of them, however, Hydrowl was a few milliseconds slower than H-QL (less than 100ms) due to the additional overhead introduced by checking covering, splitting the query, and joining the results. However, as shown in the table the benefits when it comes to the hard queries are much more significant compared to this minor overhead.

Table 3. Query Answering Times

	LUBM			UOBM				
	3	8	9	3	4	9	11	14
	5 universities			**1 department**				
H-QL	1.4	1.4	105	204	5.8	21.6	1.7	48.7
Hydrowl	.07	.24	.13	.02	.01	.01	.07	35.3
	10 universities			**1 university**				
Hydrowl	.9	6.7	.4	.3	.09	.05	2.6	t/o

8 CONCLUSIONS

In this paper we have investigated whether given a (SPARQL) query $\mathcal{Q}$ over an OWL 2 DL TBox $\mathcal{T}$ and a system ans complete for a fragment $\mathcal{L}$ of OWL 2 DL it is possible to identify in an efficient way if ans is complete for $\mathcal{Q}, \mathcal{T}$. We have provided with a sufficient condition for checking completeness and shown that there are theoretical limitations for also devising a necessary condition. Nevertheless, for the latter case we have provided a syntactic condition for checking incompleteness of ans. Our techniques have important applications in query answering. More precisely, we have devised an algorithm that decides whether a user query $\mathcal{Q}$ can be evaluated using a highly-scalable system ans or a fully-fledged OWL 2 DL reasoner needs to be employed. Even if ans cannot be used in general, our algorithm can still exploit it to a large extent in order to speed up the evaluation of $\mathcal{Q}$ by the OWL 2 DL reasoner. Our experiments have provided with very encouraging results showing that our hybrid algorithm can answer queries that are hard for a state-of-the-art OWL 2 DL system in a matter of milliseconds.

Regarding directions for future work we would like to clarify the issue of the sufficient and necessary condition, apply our framework using different combinations of systems, and lift the restriction to SPARQL CQs. Further tests and optimisations are also envisioned.

ACKNOWLEDGEMENTS

Giorgos Stoilos was funded by a Marie Curie Career Reintegration Grant within European Union's 7th Framework Programme (FP7/2007-2013) under REA grant agreement 303914.

REFERENCES

[1] F. Baader, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, *The Description Logic Handbook: Theory, implementation and applications*, Cambridge University Press, 2002.
[2] Bernardo Cuenca Grau, Boris Motik, Giorgos Stoilos, and Ian Horrocks, 'Completeness guarantees for incomplete ontology reasoners: Theory and practice', *J. of Artif. Intell. Res.*, **43**, 419–476, (2012).
[3] Bernardo Cuenca Grau, Boris Motik, Giorgos Stoilos, and Ian Horrocks, 'Computing datalog rewritings beyond horn ontologies', in *Proc. of IJCAI 2013*, (2013).
[4] Thomas Eiter, Magdalena Ortiz, Mantas Simkus, Trung-Kien Tran, and Guohui Xiao, 'Query rewriting for Horn-$\mathcal{SHIQ}$ plus rules', in *AAAI*, (2012).
[5] Birte Glimm, Carsten Lutz, Ian Horrocks, and Ulrike Sattler, 'Conjunctive query answering for the description logic $\mathcal{SHIQ}$', *J. of Artif. Intel. Res. (JAIR)*, **31**, 157–204, (2008).
[6] Atanas Kiryakov, Barry Bishoa, Damyan Ognyanoff, Ivan Peikov, Zdravko Tashev, and Ruslan Velkov, 'The Features of BigOWLIM that Enabled the BBCs World Cup Website', in *Proc. Semantic Data Management (SemData)*, (2010).
[7] Ilianna Kollia and Birte Glimm, 'Optimizing SPARQL query answering over owl ontologies', *J. Artif. Intell. Res. (JAIR)*, **48**, 253–303, (2013).
[8] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz (Editors), 'OWL 2 Web Ontology Language Profiles', *W3C Recommendation*, (2009).
[9] Boris Motik, Ian Horrocks, and Su Myeon Kim, 'Delta-Reasoner: A Semantic Web Reasoner for an Intelligent Mobile Platform', in *Proc. of WWW 2012*, pp. 63–72, (2012).
[10] Boris Motik, Rob Shearer, and Ian Horrocks, 'Hypertableau Reasoning for Description Logics', *J. of Artif. Intell. Res.*, **36**, 165–228, (2009).
[11] Riku Nortje, Katarina Britz, and Thomas Meyer, 'Reachability modules for the description logic $\mathcal{SRIQ}$', in *Proc. of LPAR 19*, (2013).
[12] Magdalena Ortiz, Diego Calvanese, and Thomas Eiter, 'Data complexity of query answering in expressive description logics via tableaux', *J. of Autom. Reas.*, **41**(1), 61–98, (2008).
[13] Giorgos Stoilos, Bernardo Cuenca Grau, Boris Motik, and Ian Horrocks, 'Repairing ontologies for incomplete reasoners', in *Proc. of ISWC-11*, pp. 681–696, (2011).
[14] Boontawee Suntisrivaraporn, 'Module extraction and incremental classification: A pragmatic approach for ontologies', in *Proc. of ESWC 2008*, pp. 230–244, (2008).
[15] Zhe Wu, George Eadon, Souripriya Das, Eugene Inseok Chong, Vladimir Kolovski, Melliyal Annamalai, and Jagannathan Srinivasan, 'Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle', in *Proc. of ICDE*, pp. 1239–1248, (2008).
[16] Yujiao Zhou, Yavor Nenov, Bernardo Cuenca Grau, and Ian Horrocks, 'Complete query answering over horn ontologies using a triple store.', in *Proc. of ISWC 2013*, (2013).