How Incomplete is Your Semantic Web Reasoner?

Systematic Analysis of the Completeness of Query Answering Systems

Giorgos Stoilos and **Bernardo Cuenca Grau** and **Ian Horrocks**
Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford, UK

Abstract

Conjunctive query answering is a key reasoning service for many ontology-based applications. In order to improve scalability, many Semantic Web query answering systems give up completeness (i.e., they do not guarantee to return all query answers). It may be useful or even critical to the designers and users of such systems to understand how much and what kind of information is (potentially) being lost. We present a method for generating test data that can be used to provide at least partial answers to these questions, a purpose for which existing benchmarks are not well suited. In addition to developing a general framework that formalises the problem, we describe practical data generation algorithms for some popular ontology languages, and present some very encouraging results from our preliminary evaluation.

Introduction

Ontologies expressed in the W3C's Web Ontology Language (OWL) and its revision OWL 2 are playing a key role in the development of the Semantic Web. An important application of OWL is *data access* (Poggi et al. 2008), where an ontology is used to describe the meaning of the data stored in various sources, and query answers reflect both the data and the knowledge captured in the ontology. In this setting, query languages are often based on conjunctive queries (CQs) (Glimm et al. 2007; Ortiz, Calvanese, and Eiter 2008; Calvanese et al. 2007; Lutz, Toman, and Wolter 2009; Pérez-Urbina, Motik, and Horrocks 2008), with the ontology providing the vocabulary used in queries.

Unfortunately, the worst-case complexity of CQ answering for OWL and OWL 2 is very high—in fact the decidability of general CQ answering for OWL is still open—and even for decidable cases, existing algorithms for sound and complete query answering may not scale well in practice. As a result, those attempting to develop scalable systems often choose to restrict the expressive power of the ontology language or to give up completeness (or both).

In Semantic Web applications, completeness is often not strictly required; consequently, many systems have chosen to give up completeness. This allows them to support a large fragment of OWL (2), and also to base their implementations on scalable database and RDF triple store technologies. Examples of such systems are Oracle's Semantic Data Store, Sesame, OWLim, Minerva, HAWK, and Virtuoso.

The tradeoff between completeness and efficiency is obviously a continuum. At one extreme, a very efficient (but highly incomplete) CQ answering system could be implemented by ignoring the ontology and matching the query against the data using a database or triple store. At the other extreme lie complete but comparatively inefficient systems based on theorem proving techniques. Incomplete systems lie somewhere between these two extremes, finding some but not all the answers implied by the ontology and the data.

A major difficulty with incomplete CQ answering systems is determining "how incomplete" they are. Thus, application developers may find it difficult to assess whether a system meets their needs. In practice, such systems are often evaluated using standard benchmarks, such as the Lehigh University Benchmark (LUBM) (Guo, Pan, and Heflin 2005) and the University Ontology Benchmark (UOBM) (Ma et al. 2006). LUBM consists of an ontology describing an academic domain, a collection of Java classes for automatically creating "realistic" data, and a set of 14 test queries; UOBM is an extension of LUBM with a richer ontology.

This kind of evaluation is of limited utility for assessing completeness: the degree of completeness exhibited by a system is likely to be highly dependent on the test ontology, query and data, and good behaviour on LUBM or UOBM does not guarantee good behaviour in general. Moreover, the data generated in LUBM and UOBM exhibits several limitations concerning the evaluation of completeness:

1. The data is not *generic*. The generation scripts have been crafted for the specific ontology in the benchmark, and cannot be reused to generate data for other ontologies.
2. The data is not *exhaustive*. Completeness can be measured only w.r.t. a fixed number of relational structures hard-coded into the generation scripts, and not w.r.t. other relational structures that may occur in "real data".
3. Query answers may not be *verifiable*. Since the correct answers are not known a-priori, completeness can only be measured by comparison with the output of a complete system, and hence cannot be verified at all for large datasets that no complete system can handle.

The first limitation could be addressed by using randomly generated data. However, such data typically bears little resemblance to real data, and so tells us little about a system's likely behaviour in practice. Moreover, it is unlikely to be exhaustive (the number of possible relational structures may be infinite), and query answers may not be verifiable.

In this paper, we present a novel framework for the automatic generation of ontology data that addresses all of the above problems. The idea behind this framework is that, given an ontology $\mathcal{T}$ (with no data attached to it) and a CQ q, we would like to generate an *exhaustive* (and hopefully small) collection of datasets $\mathbf{D}$ such that, if a system S is complete for $\mathcal{T}$, q and each of the datasets $d \in \mathbf{D}$, then S is complete for $\mathcal{T}$, q and *any* dataset. Furthermore, for each $d \in \mathbf{D}$, the correct answer to q w.r.t. $\mathcal{T}$ and d should be known; we can then check the answers given by S and provide a quantitative measure of its "completeness degree" w.r.t. $\mathcal{T}$ and q. Although a favourable result with such a $\mathbf{D}$ does not guarantee the completeness of S in general, it does allow application developers to assess with a much higher degree of confidence whether S meets their needs, particularly if the ontology and the kinds of query to be answered are largely determined by the application.

In order to be exhaustive, $\mathbf{D}$ would need to include instances of every kind of relational structure that can produce an answer to q. Unfortunately, we show that there exist OWL ontologies and CQs for which such a $\mathbf{D}$ would necessarily be infinite. However, we go on to show that exhaustive collections of datasets do exist for ontologies and CQs satisfying certain restrictions, and we investigate the design of an algorithm for generating them. Furthermore, we show that by fixing an upper bound on the number of individuals that can occur in a dataset, an exhaustive $\mathbf{D}$ always exists (even for arbitrary OWL and OWL 2 ontologies), and we also examine how to generate such collections in practice.

We have implemented a dataset generator and evaluated it on the LUBM ontology and queries using several CQ answering systems based on incomplete algorithms. Our results show that systems which are complete w.r.t. the LUBM suite for many (if not all) of the test queries, prove to be incomplete w.r.t. the same ontology and queries when the datasets produced by our generator are substituted for the LUBM data. Analysis of failed tests provides useful insights into the causes of incompleteness, and shows that such cases are not unrealistic. Finally, our techniques can be used to generate datasets that are harder than those in LUBM, and thus could be useful for performance analysis.

The proofs of all theorems can be found online[1].

Preliminaries

Description Logics The formal underpinning of OWL(2) is based on *description logics* (DLs) (Baader et al. 2002). We next recapitulate the syntax of the DLs used in this paper.

Let $\mathbf{C}$, $\mathbf{R}$, and $\mathbf{I}$ be countable, disjoint sets of *atomic concepts*, *atomic roles*, and *individuals*. A *role* is either atomic or an *inverse role* R^- for $R \in \mathbf{R}$. For R, R' roles, a *RIA* has the form $R \sqsubseteq R'$ and a *transitivity axiom* $\mathsf{Trans}(R)$.

[1]http://aaai2010web.tripod.com/syntheticgeneration.pdf

Let $\mathcal{DL}$ be either DL-Lite$_{core}$ (Calvanese et al. 2007), $\mathcal{EL}$ (Baader, Brandt, and Lutz 2005), or $\mathcal{FL}_0$ (Baader et al. 2002). For $A \in \mathbf{C}$, R a role, and C_1, C_2, C $\mathcal{DL}$-concepts, the set of $\mathcal{DL}$-concepts is the smallest containing: *(i)* A, $\exists R.\top$, if $\mathcal{DL}$ = DL-Lite$_{core}$; *(ii)* $\top$, A, $C_1 \sqcap C_2$, $\exists R.C$, if $\mathcal{DL} = \mathcal{EL}$; and *(iii)* $\top$, A, $C_1 \sqcap C_2, \forall R.C$, if $\mathcal{DL} = \mathcal{FL}_0$.

A *$\mathcal{DL}$-GCI* has the form $C_1 \sqsubseteq C_2$ for C_1, C_2 $\mathcal{DL}$-concepts; if $\mathcal{DL}$ = DL-Lite$_{core}$, then C_2 can also be the negation of a DL-Lite$_{core}$-concept. A $\mathcal{DL}$-TBox $\mathcal{T}$ is a finite set of $\mathcal{DL}$-GCIs. An ABox $\mathcal{A}$ is a finite set of assertions $A(a)$ or $R(a, b)$, for $A \in \mathbf{C}$, $R \in \mathbf{R}$, and $a, b \in \mathbf{I}$. A $\mathcal{DL}$-ontology $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ consists of a $\mathcal{DL}$-TBox $\mathcal{T}$ and an ABox $\mathcal{A}$. The semantics and definitions of various reasoning problems are standard.

Conjunctive and Datalog Queries We use the standard notions of first-order term, atom and variable. A *datalog clause* is an expression $H \leftarrow B_1 \wedge \ldots \wedge B_n$ where H (the *head*) is a (possibly empty) function-free atom, $\{B_i\}$ (the *body*) is a set of function-free atoms, and each variable in the head also occurs in the body. A *datalog program* P is a finite set of datalog clauses. A *datalog query* q is a tuple $\langle Q_P, P\rangle$ with Q_P a query predicate and P a datalog program. We denote with $\mathsf{var}(q)$ the set of variables occurring in q. A query q is a union of conjunctive queries (UCQ) if Q_P is the only predicate occurring in head position in P and the body of each clause in P does not contain Q_P ; finally, q is a conjunctive query (CQ) if it is a UCQ and P has exactly one clause. If $q = \langle Q_P, P\rangle$ is a CQ we often abuse notation and write $q = P$; if q is a UCQ with $P = \{P_1, \ldots, P_n\}$, we often write $q = \{q_1, \ldots, q_n\}$ with $q_i = P_i$ a CQ. A variable in a CQ is *distinguished* if it appears in the head.

A tuple of constants $\vec{a}$ is a certain answer of a datalog query $q = \langle Q_P, P\rangle$ over $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ iff $\mathcal{O} \cup P \models Q_P(\vec{a})$, where P is seen as a set of universally quantified implications with first-order semantics; the set of certain answers of q on $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ is (equivalently) denoted as either $\mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ or $\mathsf{cert}(q, \mathcal{O})$. A *CQ answering algorithm* ans for $\mathcal{DL}$ is a procedure that, for each $\mathcal{DL}$-ontology $\mathcal{O}$ and CQ q computes in a finite number of steps a set $\mathsf{ans}(q, \mathcal{O})$ of tuples of constants. It is *sound* if $\mathsf{ans}(q, \mathcal{O}) \subseteq \mathsf{cert}(q, \mathcal{O})$ for each $\mathcal{O}$ and q. It is *complete* if $\mathsf{cert}(q, \mathcal{O}) \subseteq \mathsf{ans}(q, \mathcal{O})$ for each $\mathcal{O}$,q. It is *monotonic* if $\mathsf{ans}(q, \mathcal{O}) \subseteq \mathsf{ans}(q, \mathcal{O}')$ for each $\mathcal{O}$, $\mathcal{O}'$ and q with $\mathcal{O} \subseteq \mathcal{O}'$. Finally, it is *invariant under isomorphisms* if for each pair of *isomorphic ABoxes* $\mathcal{A}$ and $\mathcal{A}'$ (i.e., identical modulo renaming of individuals), $\mathsf{ans}(q, \mathcal{T}, \mathcal{A})$ and $\mathsf{ans}(q, \mathcal{T}, A')$ are also identical modulo the same renaming.

Justifications We will often use the notion of a *justification* for an entailment (see e.g., (Kalyanpur et al. 2007)), which we define next for the case of CQ answering.

Definition 1 *Let $\mathcal{O}$ be consistent, q a CQ and $\vec{a} \in$ $\mathsf{cert}(q, \mathcal{O})$. An ontology $J \subseteq \mathcal{O}$ is a justification for $q, \vec{a}$ in $\mathcal{O}$ if $\vec{a} \in \mathsf{cert}(q, J)$ and $\vec{a} \notin \mathsf{cert}(q, J')$ for each $J' \subset J$.*

A Framework for ABox Generation

In this section, we present a framework for the automatic generation of ontology data. The central notion is that of a *testing base*: a collection of ABoxes for a TBox $\mathcal{T}$ and CQ q. Each element of a testing base (called a *testing unit*) represents a minimal ABox which can produce an answer to q. To check completeness, a testing base must be *exhaustive*, i.e., it must contain all "relevant" testing units.

Definition 2 *An ABox $\mathcal{A}$ is a* testing unit *for a CQ q and TBox $\mathcal{T}$ if $\mathcal{T} \cup \mathcal{A}$ is consistent and there exists a tuple $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ such that $\mathcal{A}$ is the ABox part of some justification for q, $\vec{a}$ in $\mathcal{T} \cup \mathcal{A}$. A* testing base $\mathbf{B}$ *is a finite set of testing units for q, $\mathcal{T}$. It is* exhaustive *if, for each testing unit $\mathcal{A}$ for q and $\mathcal{T}$, it contains an ABox that is isomorphic to $\mathcal{A}$, and it is* minimal *if no two ABoxes in it are isomorphic.*

As an example, consider $\mathcal{T} = \{\exists R.\top \sqsubseteq A, S \sqsubseteq R\}$ and $q = \langle Q_P, P\rangle$ where $P = \{Q_P(x) \leftarrow A(x) \wedge R(x, y)\}$. The following ABoxes are testing units for q and $\mathcal{T}$, and the testing base $\mathbf{B} = \{A_1, \ldots, A_8\}$ is exhaustive and minimal:

$$\begin{array}{ll}
\mathcal{A}_1 = \{R(a,b)\} & \mathcal{A}_2 = \{A(a), R(a,b)\} \\
\mathcal{A}_3 = \{R(a,a)\} & \mathcal{A}_4 = \{A(a), R(a,a)\} \\
\mathcal{A}_5 = \{S(a,b)\} & \mathcal{A}_6 = \{A(a), S(a,b)\} \\
\mathcal{A}_7 = \{S(a,a)\} & \mathcal{A}_8 = \{A(a), S(a,a)\}
\end{array}$$

Intuitively, given an ABox $\mathcal{A}$, any $\vec{a} \in \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ must be entailed by some subset of $\mathcal{A}$ that is isomorphic to an ABox in $\mathbf{B}$, and thus a CQ answering algorithm that correctly computes $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}_i)$ for each $\mathcal{A}_i \in \mathbf{B}$ should correctly compute $\mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ for *any* ABox $\mathcal{A}$. This intuition is formalised in the following theorem.

Theorem 3 *Let* ans *be a sound, monotonic and invariant under isomorphisms CQ answering algorithm for $\mathcal{DL}$. Let q be a CQ, $\mathcal{T}$ a $\mathcal{DL}$-TBox and $\mathbf{B}$ an exhaustive testing base for q and $\mathcal{T}$. Then, the following property ($\Diamond$) holds for any $\mathcal{A}'$ s.t. $\mathcal{T} \cup \mathcal{A}'$ is consistent: If $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}) = \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ for each $\mathcal{A} \in \mathbf{B}$, then $\mathsf{ans}(q, \mathcal{T}, \mathcal{A}') = \mathsf{cert}(q, \mathcal{T}, \mathcal{A}')$.*

Furthermore, given a CQ answering algorithm ans, we can quantify the completeness of ans for a CQ q and TBox $\mathcal{T}$ by comparing $\mathsf{ans}(q, \mathcal{T}, \mathcal{A})$ with $\mathsf{cert}(q, \mathcal{T}, \mathcal{A})$ for each ABox $\mathcal{A}$ in a minimal exhaustive testing base for q and $\mathcal{T}$.

Definition 4 *Let* ans*, q, $\mathcal{T}$ and $\mathbf{B}$ be as in Theorem 3 with $\mathbf{B}$ being also minimal. The* completeness degree *δ of the algorithm* ans *for q and $\mathcal{T}$ is defined as follows:*

$$\delta(\mathsf{ans}, q, \mathcal{T}) = \frac{\sharp\{\mathcal{A} \in \mathbf{B} \mid \mathsf{cert}(q, \mathcal{T}, \mathcal{A}) = \mathsf{ans}(q, \mathcal{T}, \mathcal{A})\}}{\sharp\mathbf{B}}$$

Consider our example CQ q, TBox $\mathcal{T}$ and testing base $\mathbf{B} = \{A_1, \ldots, A_8\}$ as given above. An algorithm a_1 that ignores $\mathcal{T}$ and simply matches q to the data would only compute the correct answers for $\mathcal{A}_2$ and $\mathcal{A}_4$; hence, $\delta(\mathsf{a}_1, q, \mathcal{T}) = 0.25$. An algorithm a_2 that handles RIAs but not existential quantification would only compute the correct answers for $\mathcal{A}_2, \mathcal{A}_4, \mathcal{A}_6$ and $\mathcal{A}_8$; thus, $\delta(\mathsf{a}_2, q, \mathcal{T}) = 0.5$. Finally, a complete algorithm a_3 would compute the correct answers for all ABoxes; hence, $\delta(\mathsf{a}_3, q, \mathcal{T}) = 1$.

Intuitively, given a CQ q, a TBox $\mathcal{T}$, and CQ answering algorithms ans and ans', if there is some ABox $\mathcal{A}$ s.t. ans does not compute the correct answer to q w.r.t. $\mathcal{T}$ and $\mathcal{A}$, then $\delta(\mathsf{ans}, q, \mathcal{T})$ should be less than one; furthermore, if ans' computes the correct answer to q w.r.t. $\mathcal{T}$ and $\mathcal{A}$ whenever ans does, then $\delta(\mathsf{ans}', q, \mathcal{T})$ should be greater than $\delta(\mathsf{ans}, q, \mathcal{T})$. Our notion of completeness degree conforms to these intuitions, as shown in the following proposition:

Proposition 5 *Consider the notation in Theorem 3. The following properties hold for any $\mathcal{A}$ s.t. $\mathcal{T} \cup \mathcal{A}$ is consistent:*

1. *If $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) \neq \mathsf{ans}(q, \mathcal{T}, \mathcal{A})$, then $\delta(\mathsf{ans}, q, \mathcal{T}) < 1$.*
2. *Let ans' be a CQ answering algorithm with the same properties as* ans *in Theorem 3. If, for each $\mathcal{A}$, $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) = \mathsf{ans}(q, \mathcal{T}, \mathcal{A})$ implies $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) = \mathsf{ans}'(q, \mathcal{T}, \mathcal{A})$, then $\delta(\mathsf{ans}, q, \mathcal{T}) \leq \delta(\mathsf{ans}', q, \mathcal{T})$.*

Unfortunately, an exhaustive testing base might not exist, even for rather simple TBoxes and queries.

Theorem 6 *Let $\mathcal{DL}$ be $\mathcal{EL}$, or $\mathcal{FL}_0$, or a DL allowing for transitivity axioms. There is a CQ q and a $\mathcal{DL}$-TBox $\mathcal{T}$ for which no exhaustive testing base exists.*

The proof of Theorem 6 exploits the fact that we can have infinitely many non-isomorphic ways to obtain a certain answer by considering ABoxes with chains $R(a_1, a_2), \ldots R(a_{n-1}, a_n)$ of increasing length and using either transitive roles, or cyclic axioms involving existential (universal) quantification (note that a testing base must be a *finite* set). In practice, however, it may be reasonable to impose an upper bound on the length of such chains (e.g., by limiting the number of individuals in an ABox). We can thus obtain a weaker notion of exhaustiveness.

Definition 7 *A testing base for q and $\mathcal{T}$ is n-exhaustive if, for each testing unit $\mathcal{A}$ for q and $\mathcal{T}$ with at most n individuals, it contains an ABox that is isomorphic to $\mathcal{A}$.*

For any positive integer n, an n-exhaustive testing base always exists, as established in the following proposition.

Proposition 8 *For any TBox $\mathcal{T}$, CQ q and positive integer n, there is a minimal, n-exhaustive testing base.*

Furthermore, n-exhaustive testing bases enjoy analogous properties to exhaustive ones. If $\mathbf{B}$ is n-exhaustive, Property ($\Diamond$) from Theorem 3 and Properties 1 and 2 from Proposition 5 hold for any ABox with at most n individuals.

Computing Testing Bases

In this section, we propose techniques for computing testing bases. We start by identifying sufficient conditions for an exhaustive testing base to exist. Then, we investigate the design of practical algorithms for computing testing bases.

Existance of an Exhaustive Testing Base

By Theorem 6, there are CQs and TBoxes given in rather inexpressive DLs for which an exhaustive testing base does not exist. In this section, however, we identify a family of TBoxes and queries that are relevant in practice and for which exhaustive testing bases can be computed.

To this end, we establish a connection between the existence for $\mathcal{T}$, q of an exhaustive testing base and the existence of a *UCQ rewriting*; that is, a UCQ that "compiles" all the information in $\mathcal{T}$ that is relevant to q for the purpose of query answering, as formalised next.

Definition 9 *A* UCQ rewriting *for a TBox $\mathcal{T}$ and a CQ q is a UCQ $\mathsf{rew}(\mathcal{T}, q)$ such that, for each ABox $\mathcal{A}$*

1. $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) \subseteq \bigcup_{q' \in \mathsf{rew}(\mathcal{T}, q)} \mathsf{cert}(q', \emptyset, \mathcal{A})$*; and*

2. *for each $q' \in \mathsf{rew}(\mathcal{T}, q)$, $\mathsf{cert}(q', \emptyset, \mathcal{A}) \subseteq \mathsf{cert}(q, \mathcal{T}, \mathcal{A})$.*

Several algorithms for computing UCQ rewritings have been proposed in the literature. In (Calvanese et al. 2007), it was shown that a UCQ rewriting always exists if $\mathcal{T}$ is expressed in certain extensions of DL-Lite$_{core}$, including the one underlying the QL profile of OWL 2 (Motik et al. 2009). Furthermore, (Pérez-Urbina, Motik, and Horrocks 2008) showed that UCQ rewritings may exist even if $\mathcal{T}$ is expressed in a logic of the $\mathcal{EL}$ family and proposed an algorithm that computes a UCQ rewriting whenever one exists.

In what follows, we show that it is always possible to construct an exhaustive testing base for $\mathcal{T}$,q from a UCQ rewriting for $\mathcal{T}$, q. Our technique is based on the intuition that testing units can be computed by mapping the variables from a CQ in $\mathsf{rew}(\mathcal{T}, q)$ to individuals, as formalised next.

Definition 10 *Let q be a CQ and π a mapping from all the variables of q to individuals. The following ABox is an* instantiation *of q:*

$$\begin{aligned} \mathcal{A}^q_\pi \quad := \quad & \{A(\pi(x)) \mid A(x) \text{ body atom } \} \cup \\ & \{R(\pi(x), \pi(y)) \mid R(x, y) \text{ body atom } \} \end{aligned}$$

Not all instantiations of a CQ in a rewriting, however, lead to a testing unit, as discussed in the following examples.

Example 11 *Consider the following TBox and CQ:*

$$\mathcal{T} = \{\exists R^-.\top \sqsubseteq A\} \quad q = \{Q_P(x) \leftarrow R(x, y) \wedge A(y)\}.$$

The UCQ $\{q, q_1, q_2\}$ with $q_1 = \{Q_P(x) \leftarrow R(x, y)\}$, and $q_2 = \{Q_P(x) \leftarrow R(x, y) \wedge R(z, y)\}$, is a UCQ rewriting. However, the ABox $\mathcal{A}^{q_2}_\pi = \{R(a, b), R(c, b)\}$, where $\pi = \{x \mapsto a, y \mapsto b, z \mapsto c\}$, is not a testing unit. This is because there is a different instantiation of q_2, namely $\mathcal{A}^{q_2}_{\pi'}$ with $\pi' = \{x \mapsto a, y \mapsto b, z \mapsto a\}$, s.t. $\mathcal{A}^{q_2}_\pi$ and $\mathcal{A}^{q_2}_{\pi'}$ lead to the same certain answer and $\mathcal{A}^{q_2}_{\pi'} \subset \mathcal{A}^{q_2}_\pi$.

Example 12 *Consider the following TBox and CQ:*

$$\mathcal{T} = \{C \sqcap B \sqsubseteq B\} \quad q = \{Q_P(x) \leftarrow B(x)\}$$

The UCQ $\{q, q_1\}$ with $q_1 = \{Q_P(x) \leftarrow C(x) \wedge B(x)\}$ is a UCQ rewriting, but $\mathcal{A}^{q_1}_{\pi_1} = \{C(a), B(a)\}$ with $\pi_1 = \{x \mapsto a\}$ is not a testing unit. This is because there is an instantiation $\mathcal{A}^q_\pi$ of q, where $\pi = \{x \mapsto a\}$, s.t. $\mathcal{A}^q_\pi$ and $\mathcal{A}^{q_1}_{\pi_1}$ lead to the same answer and $\mathcal{A}^q_\pi \subset \mathcal{A}^{q_1}_{\pi_1}$.

These examples suggest that, when instantiating a CQ, we need to make sure that there is no "smaller" instantiation of a (possibly different) CQ in the rewriting.

Definition 13 *Let $u = \{q_1, \ldots, q_n\}$ be a UCQ and assume w.l.o.g. that $\mathsf{var}(q_i) \cap \mathsf{var}(q_j) = \emptyset$ for $i \neq j$. Let* $\mathsf{ind} = \{a_1, \ldots, a_m\}$ *be a set of individuals s.t. $m = \sharp\mathsf{var}(u)$, Π_u a subset of the mappings from $\mathsf{var}(u)$ to ind and Π_q the restriction of Π_u to the variables in $q \in u$. The set Π_u is a* valid instantiation *of u if it is a maximal subset of the mappings from $\mathsf{var}(u)$ to ind with the following property:*

> $(*)$*: for each $q_i, q_j \in u$ and $\pi \in \Pi_{q_i}$, there is no mapping π' from $\mathsf{var}(q_j)$ to ind s.t. π and π' map the distinguished variables in q_i and q_j identically and $\mathcal{A}^{q_j}_{\pi'} \subset \mathcal{A}^{q_i}_\pi$.*

Algorithm 1 Compute a non-exhaustive testing base

```
Algorithm: $\mathsf{tb}(u)$
Input: a UCQ rewriting $u$ for $\mathcal{T}$, $q$
1 Initialize $\mathsf{Out} := \emptyset$
2 Compute $u' := \mathsf{sub}(\mathsf{cond}(u))$
3 Construct $\mathsf{ind} := \{a_1, \ldots, a_n\}$ for $n = \sharp\mathsf{var}(u')$
4 For each $q_i \in u'$
      For each injective mapping $\pi$ from $\mathsf{var}(q_i)$ to $\mathsf{ind}$
        $\mathsf{Out} := \mathsf{Out} \cup \{\mathcal{A}^{q_i}_\pi\}$
5 Return Out
```

It is easy to see that a (finite) valid instantiation for a given UCQ always exists. Furthermore, valid instantiations of a UCQ rewriting always produce testing units, as shown next.

Lemma 14 *Let $\mathcal{T}' \subseteq \mathcal{T}$ and $u = \mathsf{rew}(\mathcal{T}', q)$ a UCQ rewriting for $\mathcal{T}'$, q. Let Π_u be a valid instantiation. Then, for each $q_i \in u$ and each $\pi \in \Pi_{q_i}$ such that $\mathcal{T}' \cup \mathcal{A}^{q_i}_\pi$ is consistent, we have that $\mathcal{A}^{q_i}_\pi$ is a testing unit for $\mathcal{T}$, q.*

In order to compute exhaustive testing bases, we need to consider the rewritings of all possible subsets of the input TBox, as shown by the following theorem.

Theorem 15 *Let $\wp(\mathcal{T})$ be the powerset of $\mathcal{T}$. For each $\mathcal{T}_i \in \wp(\mathcal{T})$, let $\mathsf{rew}(\mathcal{T}_i, q)$ be a UCQ rewriting. Finally, let*

$$\mathsf{rew}^{\wp}(\mathcal{T}, q) = \bigcup_{\mathcal{T}_i \in \wp(\mathcal{T})} \mathsf{rew}(\mathcal{T}_i, q)$$

Then, for Π_u a valid instantiation of $\mathsf{rew}^{\wp}(\mathcal{T}, q)$, the following set is an exhaustive testing base for $\mathcal{T}$,q:

$$\mathbf{B} = \{\mathcal{A}^{q_j}_\pi \mid q_j \in \mathsf{rew}^{\wp}(\mathcal{T}, q), \pi \in \Pi_{q_j}, \mathcal{T} \cup \mathcal{A}^{q_j}_\pi \text{ consistent}\}$$

The main result in this section easily follows.

Corollary 16 *Let q be a CQ and $\mathcal{T}$ a TBox for which there is a UCQ rewriting. Then, there exists a testing base for q, $\mathcal{T}$ that is both exhaustive and minimal.*

Practical Algorithms

According to Theorem 15, an algorithm for computing an exhaustive testing base for $\mathcal{T}$, q needs to first compute the UCQ $\mathsf{rew}^{\wp}(\mathcal{T}, q)$ and then compute testing units by using only valid instantiations. Such an algorithm is not practical: first, it must examine exponentially many subsets of $\mathcal{T}$; second, it needs to check property $(*)$ from Definition 13, which potentially involves exponentially many ABox containment tests in the number of query variables.

In this section, we present a practical algorithm (see Algorithm 1) for computing a possibly non-exhaustive testing base, which we have implemented. As we will discuss later on, our empirical results suggest that the output of our algorithm can be successfully used in practice to approximate the completeness degree of several Semantic Web reasoners.

Algorithm 1 takes a UCQ rewriting u for a TBox $\mathcal{T}$ and CQ q, which can be computed using any state-of-the-art rewriting algorithm (Calvanese et al. 2007; Pérez-Urbina, Motik, and Horrocks 2008), and computes a testing base for $\mathcal{T}$,q. To achieve practicality, our algorithm does not consider rewritings for the subsets of $\mathcal{T}$ and hence the output testing base might not be exhaustive.

In Line 3, Algorithm 1 adapts well-known techniques to reduce the size of the input rewriting, as described next.

Definition 17 *A CQ q is* reduceable *if there exist distinct atoms in its body that are unifiable with θ being their* most general unifier. *A reduction q' of q is obtained by applying θ to the body of q. A* condensation reduction $\mathsf{cond}(u)$ *of a UCQ u is a UCQ in which there are no two queries q, q' such that q' subsumes q and q' is a reduction of q.*

A subsumption reduction *of u is a UCQ* $\mathsf{sub}(u)$ *in which there are no two queries q, q' such that q' subsumes q.*

Intuitively, these reductions avoid many tests when checking property $(*)$ from Definition 13. For example, applying cond to the UCQ in Example 11 would eliminate the query q_2, so $\mathcal{A}^{q_2}_{\pi}$ would not be computed. Similarly, applying sub to the UCQ in Example 12 would eliminate the query q_1 thus discarding the instantiation $\mathcal{A}^{q_1}_{\pi_1}$. These reductions, however, do not completely eliminate the need for checking $(*)$.

Example 18 *Consider the following TBox and CQ:*

$$\mathcal{T} = \{C \sqsubseteq \exists R.\top\} \quad q = \{Q_P(x) \leftarrow R(x, y) \wedge R(y, z)\}$$

Given the rewriting $\{q, q_1\}$ with $q_1 = \{Q_P(x) \leftarrow R(x, y) \wedge C(y)\}$, neither cond *nor* sub *removes any query. However, the ABox $\mathcal{A}^q_\pi = \{R(a, a), C(a)\}$ with $\pi = \{x \mapsto a, y \mapsto a\}$ is not a testing unit since a subset of it leading to the same certain answer ($\{R(a, a)\}$) is already a testing unit.*

To remedy this issue, our algorithm (see loop from Line 4) considers only instantiations of CQs in the rewriting that are injective (i.e. map distinct variables to distinct individuals). Using this restriction, the ABox $\mathcal{A}^q_\pi$ from Example 18 would not be computed, and we can show the following result:

Theorem 19 *Algorithm 1 computes a testing base for q,$\mathcal{T}$.*

We conclude this section by considering the case of TBoxes $\mathcal{T}$ and CQs q to which the negative results from Theorem 6 apply. By Corollary 16, no UCQ rewriting exists for such $\mathcal{T}$ and q. However, it might be possible to "compile" the relevant information for q in $\mathcal{T}$ into a Datalog query.

Definition 20 *A* Datalog rewriting *for a TBox $\mathcal{T}$ and a CQ q is a Datalog query $d = \langle Q_P, P\rangle$ such that* $\mathsf{cert}(q, \mathcal{T}, \mathcal{A}) = \mathsf{cert}(d, \emptyset, \mathcal{A})$ *for any ABox $\mathcal{A}$.*

In (Pérez-Urbina, Motik, and Horrocks 2008) it was shown that Datalog rewritings always exist if $\mathcal{T}$ is expressed in certain DLs of the $\mathcal{EL}$ family. For example, given $q = \{Q_P(x) \leftarrow B(x)\rangle\}$ and $\mathcal{T} = \{\exists R.B \sqsubseteq B\}$, the rewriting algorithm from (Pérez-Urbina, Motik, and Horrocks 2008) computes the (recursive) Datalog query $q = \langle Q_P, P\rangle$ where

$$P = \{Q_P(x) \leftarrow B(x), B(x) \leftarrow R(x, y) \wedge B(y)\}$$

Algorithm 2 exploits Datalog rewritings for computing a subset of an n-exhaustive testing base. The algorithm unfolds the Datalog program using SLD-resolution with backward-chaining and instantiates the body of each of the resolved clauses using an injective mapping. Cycles due to recursion are only unfolded up to a finite depth because the possible number of injective mappings using n individuals is finite. Analogously to Algorithm 1, the output of Algorithm 2 contains only testing units, but it does not contain all the testing units of an n-exhaustive testing base.

Algorithm 2 Approximate an n-exhaustive testing base

```
Algorithm: ntb($d, n$)
Inputs: a Datalog rewriting $d = \langle Q_P, P\rangle$ for $\mathcal{T}$,$q$
  a non-negative integer $n$
1 Construct ind := $\{a_1, \ldots, a_n\}$
2 Initialize Out := $\emptyset$
3 Initialize a tree with a root node $R = Q_P$
4 Repeat
      For each node $N = A_1 \wedge \ldots\ldots \wedge A_m$ with depth $i \leq n$
        For each clause $C \leftarrow B_1 \wedge \ldots \wedge B_\ell$ in P
          If $A_k$ and $C$ are unifiable with mgu $\theta$ for some $k \leq m$
            Create a child $M = (A_1 \wedge \ldots \wedge A_{k-1} \wedge B_1 \wedge \ldots \wedge$
                $B_\ell \wedge A_{k+1}, \ldots, A_m)_\theta$ of $N$
            For each injective $\pi$ from vars in $M$ to ind
              Out := Out $\cup$ $\{\mathcal{A}^M_\pi\}$
  Until no new ABox is added to Out
5 Return Out
```

Implementation and Evaluation

We have implemented Algorithm 1 in a prototype tool called SyGENiA, which uses REQUIEM[2] for the computation of UCQ rewritings. We have run SyGENiA over the LUBM TBox and queries to compute testing bases, which we then used to evaluate the completeness of four systems: Sesame 2.3-pr1,[3] OWLim 2.9.1.,[4] Minerva v1.5,[5] and HAWK v3.1.[6] Since REQUIEM does not currently support individuals in queries or transitivity in the TBox, we have replaced the individuals in queries by distinguished variables and removed the only transitivity axiom in the LUBM TBox. In this setting, all queries lead to UCQ rewritings w.r.t. the LUBM TBox, as there is no recursion involved in the rewriting process. Thus, exhaustive testing bases exist for all queries.

Table 1 summarises the results w.r.t. the LUBM queries for which the LUBM and the SyGENiA datasets lead to different completeness results. Our datasets reveal many sources of incompleteness in the evaluated systems and provide a clearer picture of their behaviour w.r.t. the LUBM ontology and queries. Interestingly, OWLim and Minerva, which are 100% complete w.r.t. the LUBM benchmark (and which are reported to be 100% complete even for the UOBM benchmark) were found to be incomplete for several queries. Similarly, Sesame and HAWK were found incomplete for many queries for which they were complete according to the LUBM benchmark; furthermore, for certain queries for which Sesame and HAWK were 0% complete w.r.t. LUBM, we found that these systems retrieve the correct answers for some testing units and therefore are not totally incomplete.

[2]http://www.comlab.ox.ac.uk/projects/requiem/home.html

[3]http://www.openrdf.org/

[4]http://www.ontotext.com/owlim/

[5]http://www.alphaworks.ibm.com/tech/semanticstk

[6]http://swat.cse.lehigh.edu/downloads/index.html♯hawk

HAWK													OWLim		
Query	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11	Q12	Q13	Q6	Q8	Q10
LUBM	1	1	1	1	1	1	1	1	1	1	0	0	1	1	1
SyGENiA	.13	.66	.2	.3	.05	.01	.07	.18	.05	.5	.04	.1	.96	.93	.96

Sesame											Minerva						
Query	Q2	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q12	Q13	Q5	Q6	Q7	Q8	Q10	Q12	Q13
LUBM	1	1	1	.83	.87	.83	.64	.83	0	0	1	1	1	1	1	1	1
SyGENiA	.75	.2	.43	.01	.02	.02	.09	.017	.05	.24	.89	.87	.9	.76	.87	.66	.24

Table 1: Completeness degree measured on the LUBM and SyGENiA datasets respectively

Our techniques provide a detailed diagnosis of the sources of incompleteness since we can point out the testing units for which a system fails and quantify how "severe" the incompleteness is for a given CQ. For example, according to our framework, Sesame is incomplete for queries Q2, Q4 and Q5. This is not surprising, as these queries use inverse roles (e.g., $\mathsf{memberOf}$ and member), which Sesame does not handle. The resulting incompleteness, however, is rather severe (e.g. Sesame is only 20% complete for Q4). However, the LUBM generator does not create assertions involving the role member and hence Sesame is complete for these queries w.r.t. the LUBM data. For Q6 OWLim fails to entail $\mathcal{T} \cup \{\mathsf{GraduateStudent}(a)\} \models \mathsf{Student}(a)$ since it is not complete for inferences involving existential quantification and $\mathsf{GraduateStudent} \sqsubseteq \mathsf{Student}$ follows from

$$\begin{array}{rcl} \mathsf{GraduateStudent} & \sqsubseteq & \exists \mathsf{takesCourse}.\mathsf{GraduateCourse}, \\ \mathsf{GraduateCourse} & \sqsubseteq & \mathsf{Course}, \\ \mathsf{Student} & \equiv & \exists \mathsf{takesCourse}.\mathsf{Course} \end{array}$$

Minerva and HAWK use a DL reasoner to classify the ontology and make explicit subsumptions between atomic concepts. These two systems fail to entail $Q5(a)$ from $\mathcal{T} \cup \{\mathsf{headOf}(a,b)\}$, which follows from the entailment $\mathcal{T} \models \exists \mathsf{headOf}.\top \sqsubseteq \mathsf{Person}$ involving a complex concept. These examples suggest that the identified sources of incompleteness could easily occur in realistic data.

Finally, we have started exploring the use of our techniques for performance evaluation. We generated a pool of individuals and then created an ABox by duplicating testing units using individuals from the pool. The generated ABoxes had the same number of individuals and assertions as LUBM's dataset Univ-1. Our first experiments show that SyGENiA's datasets are approximately 2-4 times harder to load and reason with. However, further research is required to interpret these results, and we leave this for future work.

Conclusions and Future Work

This paper addresses the problem of generating ontology data for evaluating the completeness of Semantic Web reasoners. We have developed a general framework formalising our problem and studied theoretical and practical difficulties. We have provided practical algorithms for some popular DLs and evaluated them. Our first results provide a clear picture of the completeness of several systems w.r.t. the LUBM ontology and queries. There are many interesting challenges for future work, such as the design of practical data generation algorithms for expressive DLs and their evaluation on expressive ontologies, and the application of our techniques to performance analysis of CQ answering.

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

References

Baader, F.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. 2002. *The Description Logic Handbook: Theory, implementation and applications*. Cambridge University Press.

Baader, F.; Brandt, S.; and Lutz, C. 2005. Pushing the $\mathcal{EL}$ envelope. In *Proc. of IJCAI 05*.

Calvanese, D.; De Giacomo, G.; Lembo, D.; Lenzerini, M.; and Rosati, R. 2007. Tractable reasoning and efficient query answering in description logics: The DL-Lite family. *J. of Automated Reasoning* 39(3):385–429.

Glimm, B.; Horrocks, I.; Lutz, C.; and Sattler, U. 2007. Conjunctive query answering for the description logic $\mathcal{SHIQ}$. In *Proc. of IJCAI 2007*.

Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics* 3(2):158–182.

Kalyanpur, A.; Parsia, B.; Horridge, M.; and Sirin, E. 2007. Finding all justifications of OWL DL entailments. In *ISWC/ASWC*, 267–280.

Lutz, C.; Toman, D.; and Wolter, F. 2009. Conjunctive query answering in the description logic el using a relational database system. In *Proc. of IJCAI 09*.

Ma, L.; Yang, Y.; Qiu, Z.; Xie, G. T.; Pan, Y.; and Liu, S. 2006. Towards a complete OWL ontology benchmark. In *ESWC*, 125–139.

Motik, B.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; Fokoue, A.; and (Editors), C. L. 2009. OWL 2 Web Ontology Language Profiles. *W3C Recommendation*.

Ortiz, M.; Calvanese, D.; and Eiter, T. 2008. Data complexity of query answering in expressive description logics via tableaux. *J. Autom. Reasoning* 41(1):61–98.

Pérez-Urbina, H.; Motik, B.; and Horrocks, I. 2008. Rewriting Conjunctive Queries under Description Logic Constraints. In *Proc. of LID 2008*.

Poggi, A.; Lembo, D.; Calvanese, D.; Giacomo, G. D.; Lenzerini, M.; and Rosati, R. 2008. Linking data to ontologies. *J. Data Semantics* 10:133–173.