Pay-as-you-go OWL Query Answering Using a Triple Store

Yujiao Zhou and **Yavor Nenov** and **Bernardo Cuenca Grau** and **Ian Horrocks**
Department of Computer Science, University of Oxford, UK

Abstract

We present an enhanced hybrid approach to OWL query answering that combines an RDF triple-store with an OWL reasoner in order to provide scalable pay-as-you-go performance. The enhancements presented here include an extension to deal with arbitrary OWL ontologies, and optimisations that significantly improve scalability. We have implemented these techniques in a prototype system, a preliminary evaluation of which has produced very encouraging results.

1 Introduction

The use of RDF (Manola and Miller 2004) and SPARQL (W3C SPARQL Working Group 2013) to store and access semi-structured data is increasingly widespread. In many cases, an OWL ontology is used to formally specify the meaning of data (Motik, Patel-Schneider, and Parsia 2009), as well as to enhance query answers with tuples that are only entailed by the combination of the ontology and data.

Unfortunately, computing such answers is of high worst case complexity, and although heavily optimised, existing systems for query answering w.r.t. RDF data and an unrestricted OWL 2 ontology can process only small to medium size datasets (Möller et al. 2013; Wandelt, Möller, and Wessel 2010; Kollia and Glimm 2013). This has led to the development of query answering procedures that are more scalable, but that can (fully) process only fragments of OWL 2, and several prominent fragments have now been standardised as OWL 2 profiles (Motik et al. 2009). Such systems have been shown to be (potentially) highly scalable (Urbani et al. 2012; Bishop et al. 2011; Urbani et al. 2011; Wu et al. 2008), but if the ontology falls outside the relevant profile, then the answers computed by such a system may be incomplete: if it returns an answer, then all tuples in the answer are (usually) valid, but some valid tuples may be missing from the answer. When used with out-of-profile ontologies, a query answer computed by such a system can thus be understood as providing a *lower-bound* on the correct answer; however, they cannot in general provide any upper bound or even any indication as to how complete the computed answer is (Cuenca Grau et al. 2012).

More recently, we have proposed a hybrid approach that addresses some of these issues. This approach uses a triple store with OWL 2 RL reasoning capabilities to compute not only the usual lower bound answer, but also an upper bound answer, in the latter case by rewriting the input ontology into a strictly stronger OWL 2 RL ontology (Zhou et al. 2013b). If lower and upper bound answers coincide, they obviously provide a sound and complete answer. Otherwise, *relevant fragments* of the ontology and data can be extracted that are guaranteed to be sufficient to test the validity of tuples in the "gap" between the two answers; these fragments are typically much smaller than the input ontology and data, and may thus allow for checking the gap tuples using an OWL 2 reasoner such as HermiT (Motik, Shearer, and Horrocks 2009) or Pellet (Sirin et al. 2007). This extraction and checking step was, however, only proven to be correct for Horn ontologies, and also suffered from scalability issues both w.r.t. the extraction itself and the subsequent checking.

In this paper, we present several important enhancements to this hybrid approach. First, we show how the lower bound can be tightened by integrating scalable reasoning techniques for other OWL 2 profiles. Second, we show how to extend the relevant fragment extraction procedure so that it is correct for arbitrary OWL 2 ontologies, and how to use the triple store itself to compute these fragments. Finally, we show how summarisation techniques inspired by the SHER system (Dolby et al. 2007; 2009) can be used to tighten the upper bound on query answers, thus further reducing the requirement for fully-fledged OWL 2 reasoning.

We have implemented our procedure in a prototypical system using the RDFox triple store (Motik et al. 2014) and we present a preliminary evaluation over both benchmark and realistic data which suggests that the system can provide scalable pay-as-you-go query answering for a wide range of OWL 2 ontologies, RDF data and queries. In almost all cases, the system is able to completely answer queries without resorting to fully-fledged OWL 2 reasoning, and even when this is not the case, relevant fragment extraction and summarisation are effective in reducing the size of the problem to manageable proportions. This paper comes with an online technical report with all proofs.[1]

[1] http://www.cs.ox.ac.uk/isg/people/yujiao.zhou/#publications

2 Preliminaries

We adopt standard first order logic notions, such as variables, constants, atoms, formulas, clauses, substitutions, satisfiability, and entailment (Bachmair and Ganzinger 2001). We also assume basic familarity with OWL 2 (Motik, Patel-Schneider, and Parsia 2009) and its profiles (Motik et al. 2009).

Datalog Languages Extended datalog languages are well-known KR formalisms based on rules, and they have many connections with OWL 2. A generalised rule (or just a *rule*) is a function-free first order sentence of the form

$$\forall \mathbf{x} \, (\bigwedge_{j=0}^{n} B_j(\mathbf{x}) \rightarrow \bigvee_{i=0}^{m} \exists \mathbf{y}_i \, \varphi_i(\mathbf{x}, \mathbf{y}_i))$$

where $B_j(\mathbf{x})$ are *body* atoms and φ_i are conjunctions of *head* atoms. The universal quantifiers are left implicit from now on. A rule is *Horn* if $m \leq 1$, and it is *datalog* if it is Horn and does not contain existential quantifiers. A *fact* is a ground atom and a *dataset* is a finite set of facts. A *knowledge base* $\mathcal{K}$ consists of a finite set of rules and a dataset. We treat equality ($\approx$) in knowledge bases as an ordinary predicate, but assume that every knowledge base in which equality occurs contains the axioms of equality for its signature. Each OWL 2 ontology can be normalised as one such knowledge base using the correspondence of OWL and first order logic and a variant of the structural transformation (see (Motik, Shearer, and Horrocks 2009) for detais); furthermore, each OWL 2 RL ontology corresponds to a datalog knowledge base. From now on, we interchangeably use ontologies and the knowledge bases they correspond to; in particular, we use OWL 2 RL and datalog interchangeably.

Queries We focus on *conjunctive query answering* as the key reasoning problem. A *query* is a formula of the form $q(\mathbf{x}) = \exists \mathbf{y} \, \varphi(\mathbf{x}, \mathbf{y})$ with $\varphi(\mathbf{x}, \mathbf{y})$ a conjunction of atoms. Usually, we omit the *distinguished* variables $\mathbf{x}$ of queries and write just q. The query is *atomic* if $\varphi(\mathbf{x}, \mathbf{y})$ is a single atom. A tuple of individuals $\mathbf{a}$ is a *(certain) answer* to q w.r.t. a set of sentences $\mathcal{F}$ iff $\mathcal{F} \models q(\mathbf{a})$. The set of all answers to $q(\mathbf{x})$ w.r.t. $\mathcal{F}$ is denoted by $\mathsf{cert}(q, \mathcal{F})$.

Datalog Reasoning There are two main techniques for query answering over a datalog knowledge base $\mathcal{K}$. *Forward chaining* computes the set $Mat(\mathcal{K})$ of facts entailed by $\mathcal{K}$, called the *materialisation* of $\mathcal{K}$. A query q over $\mathcal{K}$ can be answered directly over the materialisation. *Backward chaining* treats a query as a conjunction of atoms (a *goal*). An *SLD (Selective Linear Definite) resolvent* of a goal $A \wedge \psi$ with a datalog rule $\varphi \rightarrow C_1 \wedge \cdots \wedge C_n$ is a goal $\psi\theta \wedge \varphi\theta$, with θ the MGU (Most General Unifier) of A and C_j, for some $1 \leq j \leq n$. An *SLD proof* of a goal G_0 in $\mathcal{K}$ is a sequence of goals $(G_0, \ldots, G_n)$ with G_n the empty goal ($\square$), and each G_{i+1} a resolvent of G_i and a rule in $\mathcal{K}$.

3 Overview

Background The main idea behind our hybrid approach to query answering is to exploit a datalog-based triple store to compute both lower bound (sound but possibly incomplete) and upper bound (complete but possibly unsound) query answers (Zhou et al. 2013a; 2013b). Given a knowledge base $\mathcal{K}$ and a query q, this is achieved by computing datalog knowledge bases $\mathcal{L}(\mathcal{K})$ and $\mathcal{U}(\mathcal{K})$ s.t. $\mathsf{cert}(q, \mathcal{L}(\mathcal{K})) \subseteq \mathsf{cert}(q, \mathcal{K}) \subseteq \mathsf{cert}(q, \mathcal{U}(\mathcal{K}))$. In (Zhou et al. 2013a), $\mathcal{L}(\mathcal{K})$ was simply the subset of datalog rules in $\mathcal{K}$ and hence $\mathcal{K} \models \mathcal{L}(\mathcal{K})$. In turn, $\mathcal{U}(\mathcal{K})$ is the result of consecutively applying the transformations Σ, Ξ and Ψ defined next. Σ renders a knowledge base into a set of clauses via standard Skolemisation. Ξ transforms a set of clauses into a satisfiable set of Horn clauses, by first adding a nullary atom $\bot$ to each negative clause, and then splitting each non-Horn clause into several clauses (one for each positive literal). Finally, Ψ transforms the output of Ξ by replacing each functional term by a fresh constant and replacing clauses by their equivalent datalog rules. Both, $\mathcal{L}(\mathcal{K})$ and $\mathcal{U}(\mathcal{K})$ are independent from query and data (they depend on the rules in $\mathcal{K}$).

$$\mathsf{Foreman}(x) \rightarrow \mathsf{Manag}(x) \quad (T_1)$$
$$\mathsf{Superv}(x) \rightarrow \mathsf{Manag}(x) \quad (T_2)$$
$$\mathsf{Manag}(x) \rightarrow \mathsf{Superv}(x) \vee \exists y.(\mathsf{boss}(x, y) \wedge \mathsf{Manag}(y)) \quad (T_3)$$
$$\mathsf{Superv}(x) \wedge \mathsf{boss}(x, y) \rightarrow \mathsf{Worker}(y) \quad (T_4)$$
$$\mathsf{TeamLead}(x) \wedge \mathsf{boss}(x, y) \wedge \mathsf{Manag}(y) \rightarrow \quad (T_5)$$
$$\mathsf{Manag}(x) \rightarrow \exists y.(\mathsf{boss}(x, y)) \quad (T_6)$$

$$\mathsf{Manag}(Sue) \quad (D_1)$$
$$\mathsf{Superv}(Dan) \quad (D_2)$$
$$\mathsf{Superv}(Rob) \quad (D_3)$$
$$\mathsf{boss}(Dan, Ben) \quad (D_4)$$
$$\mathsf{Manag}(Jo) \quad (D_5)$$
$$\mathsf{TeamLead}(Jo) \quad (D_6)$$
$$\mathsf{boss}(Jane, Rob) \quad (D_7)$$

Figure 1: Example knowledge base $\mathcal{K}^{ex}$.

We demonstrate these basic ideas using the knowledge base $\mathcal{K}^{ex}$ in Figure 1, which we use as a running example. The query q^{ex} asks for individuals that manage a worker.

$$q^{ex}(x) = \exists y(\mathsf{boss}(x, y) \wedge \mathsf{Worker}(y))$$

We have that $\mathsf{cert}(q^{ex}, \mathcal{K}^{ex}) = \{Dan, Rob, Jo\}$. The lower bound knowledge base $\mathcal{L}(\mathcal{K}^{ex})$ consists of facts D_1-D_7 and the following datalog rules.

$$\mathsf{Foreman}(x) \rightarrow \mathsf{Manag}(x) \quad (L_1)$$
$$\mathsf{Superv}(x) \rightarrow \mathsf{Manag}(x) \quad (L_2)$$
$$\mathsf{Superv}(x) \wedge \mathsf{boss}(x, y) \rightarrow \mathsf{Worker}(y) \quad (L_4)$$
$$\mathsf{TeamLead}(x) \wedge \mathsf{boss}(x, y) \wedge \mathsf{Manag}(y) \rightarrow \quad (L_5)$$

The upper bound $\mathcal{U}(\mathcal{K}^{ex})$ extends $\mathcal{L}(\mathcal{K}^{ex})$ with the rules given next, where c_1 and c_2 are fresh constants:

$$\mathsf{Manag}(x) \rightarrow \mathsf{Superv}(x) \quad (U_3^1)$$
$$\mathsf{Manag}(x) \rightarrow \mathsf{boss}(x, c_1) \quad (U_3^2)$$
$$\mathsf{Manag}(x) \rightarrow \mathsf{Manag}(c_1) \quad (U_3^3)$$
$$\mathsf{Manag}(x) \rightarrow \mathsf{boss}(x, c_2) \quad (U_6)$$

It is easy to verify that $\mathsf{cert}(q^{ex}, \mathcal{L}(\mathcal{K}^{ex})) = \{Dan\}$ and that $\mathsf{cert}(q^{ex}, \mathcal{U}(\mathcal{K}^{ex})) = \{Sue, Dan, Rob, Jo\}$.

For a knowledge base $\mathcal{K}$ like the one above, for which $\mathsf{cert}(q, \mathcal{L}(\mathcal{K})) \subsetneq \mathsf{cert}(q, \mathcal{U}(\mathcal{K}))$, we check for each "gap"

tuple $\mathbf{a} \in \mathsf{cert}(q, \mathcal{U}(\mathcal{K})) \setminus \mathsf{cert}(q, \mathcal{L}(\mathcal{K}))$ whether $\mathbf{a} \in \mathsf{cert}(q, \mathcal{K})$. This could be achieved simply by using a fully-fledged OWL 2 reasoner to check if $\mathcal{K} \models q(\mathbf{a})$, but this is typically not feasible for large datasets (Zhou et al. 2013a). In order to address this issue, we use backwards chaining reasoning over $\mathcal{U}(\mathcal{K})$ to extract a (typically small) relevant subset $\mathcal{K}_f$ of the original knowledge base $\mathcal{K}$ such that $\mathbf{a} \in \mathsf{cert}(q, \mathcal{K})$ iff $\mathbf{a} \in \mathsf{cert}(q, \mathcal{K}_f)$; a fully-fledged OWL 2 reasoner is then used to compute $\mathsf{cert}(q, \mathcal{K}_f)$ (Zhou et al. 2013b). Unfortunately, this fragment extraction technique is only answer preserving for Horn ontologies, and thus the technique as a whole can only guarantee to compute sound and complete answers when $\mathcal{K}$ is Horn. Moreover, the extraction technique can lead to memory exhaustion in practice, and even when successful, computing $\mathsf{cert}(q, \mathcal{K}_f)$ can still be challenging for fully-fledged OWL 2 reasoners (Zhou et al. 2013a).

New Contributions In the following sections we show how our technique can be extended so as to deal with arbitrary ontologies, and to improve the quality and scalability of lower and upper bound computations. In Section 4, we show how the lower bound answer can be tightened by additionally applying other scalable query answering procedures that also rely on datalog reasoning. In Section 5, we show how our fragment extraction technique can be extended to deal with non-Horn ontologies, and how the datalog engine itself can be used to perform the extraction. Finally, in Section 6, we show how a summarisation technique inspired by the SHER system can be used to tighten the upper bound by ruling out "obvious" non-answers, and how we can further exploit this idea to reduce the number of calls to the fully-fledged OWL 2 reasoner.

Our approach is *pay-as-you-go* in the sense that the bulk of the computation is delegated to a highly scalable datalog engine. Although our main goal is to answer queries over OWL 2 ontologies efficiently, our technical results are very general and hence our approach is not restricted to OWL. More precisely, given a first-order KR language L that can be captured by generalised rules and over which we want to answer conjunctive queries, our only assumption is the availability of a fully-fledged reasoner for L and a datalog reasoner, which are both used as a "black box".

Related techniques. The SCREECH system (Tserendorj et al. 2008) first exploits the KAON2 reasoner (Hustadt, Motik, and Sattler 2007) to rewrite a $\mathcal{SHIQ}$ ontology into disjunctive datalog while preserving atomic queries, and then transforms $\vee$ into $\wedge$; the resulting over-approximation can be used to compute upper bound query answers. However, this technique is restricted to $\mathcal{SHIQ}$ ontologies and atomic queries; furthermore, the set of rules obtained from KAON2 can be expensive to compute, as well as of exponential size. Both the QUILL system (Pan, Thomas, and Zhao 2009) and the work of (Wandelt, Möller, and Wessel 2010) under-approximate the ontology into OWL 2 QL; however, neither approximation is independent of both query and data, and using OWL 2 QL increases the chances that the approximated ontology will be unsatisfiable.

The SHER system uses summarisation (see Section 6 for details) to efficiently compute an upper bound answer, with exact answers then being computed via successive relaxations (Dolby et al. 2007; 2009). The technique has been shown to be scalable in practice, but it is only known to be applicable to $\mathcal{SHIN}$ and atomic queries, and is less modular than our approach. In contrast, our approach can profitably exploit the summarisation technique, and could even improve scalability for the hardest queries by replacing HermiT with SHER when the extracted fragment is $\mathcal{SHIN}$.

4 Tightening Lower Bounds

A direct way to compute lower bound answers given $\mathcal{K}$ and q is to select the datalog fragment $\mathcal{L}(\mathcal{K})$ of $\mathcal{K}$ and compute $\mathsf{cert}(q, \mathcal{L}(\mathcal{K}))$ using the datalog engine. If $\mathcal{K}$ is an OWL 2 knowledge base, however, it is possible to further exploit the datalog engine to compute a larger set of lower bound answers. To this end, of particular interest is the *combined approach* introduced to handle query answering in $\mathcal{ELHO}^r_\bot$ (Stefanoni, Motik, and Horrocks 2013)—a logic that captures most of OWL 2 EL. Given an $\mathcal{ELHO}^r_\bot$ knowledge base $\mathcal{K}'$ and a query q, the combined approach first computes the upper bound datalog program $\mathcal{U}(\mathcal{K}')$ and the corresponding answers $\mathsf{cert}(q, \mathcal{U}(\mathcal{K}'))$. A subsequent filtering step Φ, which is efficiently implementable, guarantees to eliminate all spurious tuples; the resulting answer $\Phi(\mathsf{cert}(q, \mathcal{U}(\mathcal{K}')))$ is thus sound and complete w.r.t. q and $\mathcal{K}'$.

The combined approach is clearly compatible with ours. Given an OWL 2 knowledge base $\mathcal{K}$ and query q, the procedure we use consists of the following steps. First, as in our earlier work, we select the datalog fragment $\mathcal{K}_1 = \mathcal{L}(\mathcal{K})$, and compute the materialisation $Mat(\mathcal{K}_1)$ using the datalog engine. Second, we select the subset $\mathcal{K}_2$ of $\mathcal{K}$ corresponding to $\mathcal{ELHO}^r_\bot$ axioms and Skolemise existential quantifiers to constants to obtain $\mathcal{U}(\mathcal{K}_2)$. Then, we further compute the answers $\mathsf{cert}(q, \mathcal{U}(\mathcal{K}_2) \cup Mat(\mathcal{K}_1))$. Finally, we apply the aforementioned filtering step Φ to obtain the final set of lower bound answers. Note that $\mathcal{K}_1$ and $\mathcal{K}_2$ are, in general, neither disjoint nor contained within each other.

The $\mathcal{ELHO}^r_\bot$ fragment for our running example $\mathcal{K}^{ex}$ consists of axioms T_1, T_2, T_4 and T_6, and the resulting new lower bound answer of q^{ex} is the set $\{Dan, Rob\}$.

5 Computing Relevant Fragments

If lower and upper bound answers coincide, and $\mathcal{U}(\mathcal{K})$ does not entail the nullary atom $\bot$, then we have fully answered the query q. Otherwise, we consider the remaining candidate answers S and compute a (hopefully small) fragment of $\mathcal{K}$ that is sufficient to determine whether $\mathcal{K}$ is satisfiable, as well as whether each tuple in S is indeed an answer to q.

The Horn Case In (Zhou et al. 2013b) we proposed an algorithm for computing such a fragment for the specific case where $\mathcal{K}$ is Horn. The algorithm proceeds in two steps. First, if $\mathcal{U}(\mathcal{K})$ is found to entail $\bot$, the algorithm computes a fragment $\mathcal{K}_\bot$ of $\mathcal{K}$. If $\mathcal{K}$ is found to be satisfiable, then the algorithm computes $\mathcal{K}_{[q,S]}$ for the relevant candidate answers S and checks each of them using the fully-fledged reasoner. The relevant fragment extraction is done by an inspection of all SLD proofs in $\mathcal{U}(\mathcal{K})$ of $\bot$ (for the first step) and each

$b(R, y) \wedge W(y)$	
$M(R) \wedge W(y)$	by U_6
$S(R) \wedge W(c_1)$	by U_3^3
$W(c_1)$	by D_2
$S(x) \wedge m(x, c_1)$	by L_4
$b(R, c_2)$	by D_2
$M(R)$	by U_6
$S(R)$	by L_2
$\square$	by D_2

Table 1: An SLD proof of $q^{ex}(Rob)$ in $\mathcal{U}(\mathcal{K}^{ex})$

$\bot$		$b(J, y) \wedge W(y)$	
$T(x) \wedge b(x, y) \wedge M(y)$	by L_5	$M(J) \wedge W(c_2)$	by U_6
$b(J, y) \wedge M(y)$	by D_6	$W(c_2)$	by D_5
$M(J) \wedge M(c_2)$	by U_6	$S(x) \wedge m(x, c_2)$	by L_4
$M(c_2)$	by D_5	$M(x) \wedge m(x, c_2)$	by U_3^1
$M(x)$	by U_3^3	$b(J, c_2)$	by D_5
$\square$	by D_5	$M(J)$	by U_6
		$\square$	by D_5

Table 2: SLD proofs in $\mathcal{U}(\mathcal{K}^{ex})$ of $\bot$ and $q^{ex}(Jo)$

answer in S (for the second step). The two fragments are defined as follows.

Definition 1. *Let $\mathcal{K}$ be a knowledge base, $q(\mathbf{x})$ be a query, and S be a set of tuples. Then $\mathcal{K}_\bot$ (resp. $\mathcal{K}_{[q,S]}$) is the set of all $\alpha \in \mathcal{K}$ for which there exists $\beta \in \mathcal{U}(\alpha)$ involved in an SLD proof of $\bot$ (resp. $Q(\mathbf{a})$, for some $\mathbf{a} \in S$) in $\mathcal{U}(\mathcal{K})$.*

Consider the SLD proof of $q^{ex}(Rob)$ in $\mathcal{U}(\mathcal{K}^{ex})$ presented in Table 1, where predicates and constants are abbreviated to their first letters. By Definition 1, T_2, T_4, T_6, and D_2 are included in ${\mathcal{K}^{ex}}_{[q^{ex},\{Rob\}]}$, which will entail $q^{ex}(Rob)$. Note that, in general we need to consider *all* proofs of $q^{ex}(Rob)$ in $\mathcal{U}(\mathcal{K}^{ex})$, since $\mathcal{U}(\mathcal{K}^{ex})$ overapproximates $\mathcal{K}^{ex}$.

This technique is, however, only complete for Horn knowledge bases. Indeed, recall that $Jo \in \mathsf{cert}(q^{ex}, \mathcal{K}^{ex})$, and note that every fragment of $\mathcal{K}^{ex}$ that entails $q^{ex}(Jo)$ must include rule T_5. According to Definition 1, $\mathcal{K}_{[q,\{Jo\}]}$ will include T_5 if and only if L_5 is used in an SLD proof of $q^{ex}(Jo)$ in $\mathcal{U}(\mathcal{K}^{ex})$. However, no such proof will involve L_5, since the goal $q^{ex}(Jo)$ does not involve $\bot$, and there is no way of eliminating $\bot$ from a goal using the rules in $\mathcal{U}(\mathcal{K}^{ex})$ since they do not contain $\bot$ in their bodies.

The General Case This technique can be extended to the general case by taking into account interactions between non-Horn rules and rules with $\bot$ in the head. In particular, we show that extending $\mathcal{K}_{[q,S]}$ with $\mathcal{K}_\bot$ when checking candidate answers suffices to ensure completeness.

Theorem 1. *Let $\mathcal{K}$ be a knowledge base, $q(\mathbf{x})$ a conjunctive query, and S a set of tuples. Then, $\mathcal{K}$ is satisfiable iff $\mathcal{K}_\bot$ is satisfiable; furthermore, if $\mathcal{K}$ is satisfiable, then,*

$$\mathcal{K} \models q(\mathbf{a}) \text{ iff } \mathcal{K}_{[q,S]} \cup \mathcal{K}_\bot \models q(\mathbf{a}) \quad \text{for each } \mathbf{a} \in S.$$

Consider the proofs of $\bot$ and $q^{ex}(Jo)$ presented in Table 2. According to Definition 1, $\{T_3, \ldots, T_6, D_5, D_6\}$ is a subset of $\mathcal{K}_\bot \cup \mathcal{K}_{[q^{ex},\{Jo\}]}$, and, hence, one can readily check that $q^{ex}(Jo)$ is entailed by $\mathcal{K}_\bot \cup \mathcal{K}_{[q^{ex},\{Jo\}]}$.

The proof of Theorem 1 is involved, and details are deferred to our online appendix. Nonetheless, we sketch the arguments behind the proof. A first observation is that, w.l.o.g., we can restrict ourselves to the case where q is atomic.

Lemma 1. *Let $\mathcal{K}$ be a knowledge base, $q(\mathbf{x}) = \exists \mathbf{y}\, \varphi(\mathbf{x}, \mathbf{y})$ be a conjunctive query, S be a set of tuples, Q be a fresh predicate, and let $\mathcal{K}' = \mathcal{K}_{[q,S]} \cup \mathcal{K}_\bot$, then*

$$\mathcal{K}' \models q(\mathbf{a}) \text{ iff } \mathcal{K}' \cup \{\varphi(\mathbf{x}, \mathbf{y}) \rightarrow Q(\mathbf{x})\} \models Q(\mathbf{a})$$

The crux of the proof relies on the following properties of the transformation Ξ (the step in the definition of $\mathcal{U}$ which splits each non-Horn clause C into different Horn clauses).

Lemma 2. *Let $\mathcal{N}$ be a set of first-order clauses. Then:*

- *if $C \in \mathcal{N}$ participates in a refutation in $\mathcal{N}$, then every $C' \in \Xi(C)$ is part of an SLD proof of $\bot$ in $\Xi(\mathcal{N})$;*
- *if $C \in \mathcal{N}$ participates in a resolution proof in $\mathcal{N}$ of an atomic query $Q(\mathbf{a})$, then each $C' \in \Xi(C)$ participates in an SLD proof of $\bot$ or $Q(\mathbf{a})$ in $\Xi(\mathcal{N})$.*

Thus, by Lemma 2, each resolution proof in a set of clauses $\mathcal{N}$ can be mapped to SLD proofs in $\Xi(\mathcal{N})$ that "preserve" the participating clauses. The following lemma, which recapitulates results shown in (Zhou et al. 2013a), allows us to restate Lemma 2 for $\Psi \circ \Xi$ instead of Ξ.

Lemma 3. *Let $\mathcal{H}$ be a set of first-order Horn clauses, $Q(\mathbf{x})$ be an atomic query, and $\mathbf{a}$ be a tuple of constants. If a clause C participates in an SLD proof of $Q(\mathbf{a})$ in $\mathcal{H}$, then $\Psi(C)$ participates in an SLD proof of $Q(\mathbf{a})$ in $\Psi(\mathcal{H})$.*

With these Lemmas in hand, we can exploit the refutational completeness of resolution and the entailment preservation properties of Skolemisation to show Theorem 1.

Fragment Computation The computation of the relevant fragments requires a scalable algorithm for "tracking" all rules and facts involved in SLD proofs for datalog programs. We present a novel technique that delegates this task to the datalog engine itself. The main idea is to extend the datalog program with additional rules that are responsible for the tracking; in this way, relevant rules and facts can be obtained directly from the materialisation of the modified program.

Definition 2. *Let $\mathcal{K}$ be a datalog knowledge base and let F be a set of facts in $Mat(\mathcal{K})$. Then, $\Delta(\mathcal{K}, F)$ is the datalog program containing the rules and facts given next:*

- *each rule and fact in $\mathcal{K}$;*
- *a fact $\bar{P}(\mathbf{a})$ for each fact $P(\vec{a})$ in F;*
- *the following rules for each rule $r \in \mathcal{K}$ of the form $B_1(\mathbf{x}_1), \ldots, B_m(\mathbf{x}_m) \rightarrow H(\mathbf{x})$, and every $1 \leq i \leq m$:*

$$\bar{H}(\mathbf{x}) \wedge B_1(\mathbf{x}_1) \wedge \ldots, B_m(\mathbf{x}_m) \rightarrow \mathbf{S}(\mathbf{c_r}) \qquad (1)$$

$$\bar{H}(\mathbf{x}) \wedge B_1(\mathbf{x}_1), \ldots \wedge B_m(\mathbf{x}_m) \rightarrow \bar{B}_i(\mathbf{x}_i) \qquad (2)$$

where $\mathbf{c_r}$ is a fresh constant for each rule r, and $\mathbf{S}$ is a globally fresh predicate.

The auxiliary predicates $\bar{P}$ are used to record facts involved in proofs; in particular, if $\bar{P}(\vec{c})$ is contained in

$Mat(\Delta(\mathcal{K}, F))$, then we can conclude that $P(\vec{c})$ participates in an SLD proof in $\mathcal{K}$ of a fact in F. Furthermore, each rule $r \in \mathcal{K}$ is represented by a fresh constant $\mathbf{c_r}$, and $\mathbf{S}$ is a fresh predicate that is used to record rules of $\mathcal{K}$ involved in proofs. In particular, if $\mathbf{S}(\mathbf{c_r})$ is contained in $Mat(\Delta(\mathcal{K}, F))$, then we can conclude that rule r participates in an SLD proof in $\mathcal{K}$ of a fact in F. The additional rules (1) and (2) are responsible for the tracking and make sure that the materialisation of $\Delta(\mathcal{K}, F)$ contains the required information. Indeed, if there is an instantiation $B_1(\mathbf{a}_1) \wedge \ldots \wedge B_m(\mathbf{a}_m) \rightarrow H(\mathbf{a})$ of a rule $r \in \Delta$, then, by virtue of (1), $\mathbf{c_r}$ will be added to $\mathbf{S}$, and, by virtue of (2), each $\bar{B}_i(\mathbf{a}_i)$, for $1 \leq i \leq m$, will be derived. Correctness is established as follows.

Theorem 2. *Let $\mathcal{K}$ be a datalog knowledge base and let F be a set of facts in $Mat(\mathcal{K})$. Then, a fact $P(\mathbf{a})$ (resp. a rule r) in $\mathcal{K}$ participates in an SLD proof of some fact in F iff $\bar{P}(\mathbf{a})$ (resp. $\mathbf{S}(\mathbf{c_r})$) is in $Mat(\Delta(\mathcal{K}, F))$.*

6 Summarisation and Answer Dependencies

Once the relevant fragment has been computed, we still need to check, using the fully-fledged reasoner, whether it entails each candidate answer. This can be computationally expensive if either the fragment is large and complex, or there are many candidate answers to verify.

Data Summarisation To address these issues, we first exploit summarisation techniques (Dolby et al. 2007) to efficiently prune candidate answers. The main idea behind summarisation is to "shrink" the data in a knowledge base by merging all constants that instantiate the same unary predicates. Since summarisation is equivalent to extending the knowledge base with equality assertions between constants, the summarised knowledge base entails the original one by the monotonicity of first-order logic.

Definition 3. *Let $\mathcal{K}$ be a knowledge base. A* type *T is a set of unary predicates; for a constant a in $\mathcal{K}$, we say that $T = \{A \mid A(a) \in \mathcal{K}\}$ is the type for a. Furthermore, for each type T, let c_T be a globally fresh constant uniquely associated with T. The* summary function *over $\mathcal{K}$ is the substitution σ mapping each constant a in $\mathcal{K}$ to c_T, where T is the type for a. Finally, the knowledge base $\sigma(\mathcal{K})$ obtained by replacing each constant a in $\mathcal{K}$ with $\sigma(a)$ is called the* summary *of $\mathcal{K}$.*

By summarising a knowledge base in this way, we thus overestimate the answers to queries (Dolby et al. 2007).

Proposition 3. *Let $\mathcal{K}$ be a knowledge base, and let σ be the summary function over $\mathcal{K}$. Then, for every conjunctive query q we have $\sigma(\mathsf{cert}(q, \mathcal{K})) \subseteq \mathsf{cert}(\sigma(q), \sigma(\mathcal{K}))$.*

Summarisation can be exploited to detect spurious answers in S: if a candidate answer is not in $\mathsf{cert}(\sigma(q), \sigma(\mathcal{K}))$, then it is not in $\mathsf{cert}(q, \mathcal{K})$. Since summarisation can significantly reduce the size of a knowledge base, we can efficiently detect non-answers even if checking each of them over the summary requires calling the OWL reasoner.

Corollary 1. *Let $\mathcal{K}$ be a knowledge base, let q be a query, let S be a set of tuples, and let $\mathcal{K}' = \mathcal{K}_{[q,S]} \cup \mathcal{K}_\perp$. Furthermore, let σ be the summary function over $\mathcal{K}'$. Then, $\sigma(\mathcal{K}') \not\models \sigma(q(\mathbf{a}))$ implies $\mathcal{K} \not\models q(\mathbf{a})$ for each $\mathbf{a} \in S$.*

Data	DL	Axioms	Facts
LUBM(n)	$\mathcal{SHI}$	93	$10^5 n$
UOBM$^-$(n)	$\mathcal{SHIN}$	314	$2 \times 10^5 n$
FLY	$\mathcal{SRI}$	144,407	6,308
DBPedia$^+$	$\mathcal{SHOIN}$	1,757	12,119,662
NPD	$\mathcal{SHIF}$	819	3,817,079

Table 3: Statistics for test data

Strategy	Solved	$\lvert$Univ$\rvert$	$t_{\text{avg.}}$
RL Bounds	14	1000	10.7
+ EL Lower Bound	22	1000	7.0
+ Sum, Dep	24	100	41.9

Table 4: Result for LUBM

Exploiting dependencies between answers In this last step, we try to further reduce the calls to the fully-fledged reasoner by exploiting dependencies between the candidate answers in S. Consider tuples $\mathbf{a}$ and $\mathbf{b}$ in S and the dataset $\mathcal{D}$ in the relevant fragment $\mathcal{K}_{[q,S]} \cup \mathcal{K}_\perp$; furthermore, suppose we can find an endomorphism h of $\mathcal{D}$ in which $h(\mathbf{a}) = \mathbf{b}$. If we can determine (by calling the fully-fledged reasoner) that $\mathbf{b}$ is a spurious answer, then so must $\mathbf{a}$ be; as a result, we no longer need to call the reasoner to check $\mathbf{a}$. We exploit this idea to compute a dependency graph having candidate answer tuples as nodes and an edge $(\mathbf{a}, \mathbf{b})$ whenever an endomorphism in $\mathcal{D}$ exists mapping $\mathbf{a}$ to $\mathbf{b}$. Computing endomorphisms is, however, a computationally hard problem, so we have implemented a sound (but incomplete) greedy algorithm that allows us to approximate the dependency graph.

7 Evaluation

We have implemented a prototype based on RDFox and HermiT (v. 1.3.8). In our experiments we used the LUBM and UOBM benchmarks, as well as the Fly Anatomy ontology, DBPedia and NPD FactPages (their features are summarised in Table 3). Data, ontologies, and queries are available online.[2] We compared our system with Pellet (v. 2.3.1) and TrOWL (Thomas, Pan, and Ren 2010) on all datasets. While Pellet is sound and complete for OWL 2, TrOWL relies on approximate reasoning and does not provide correctness guarantees. Tests were performed on a 16 core 3.30GHz Intel Xeon E5-2643 with 125GB of RAM, and running Linux 2.6.32. For each test, we measured materialisation times for upper and lower bound, the time to answer each query, and the number of queries that can be answered using different techniques. All times are in seconds.

LUBM Materialisation is fast on LUBM (Guo, Pan, and Heflin 2005): it takes 274s (286s) to materialise the basic lower (upper) bound entailments for LUBM(1000). These bounds match for all 14 standard LUBM queries, and we have used 10 additional queries for which this is not the case; we tested our system on all 24 queries (see Table 4 for a summary of the results). The refined lower bound was

[2] http://www.cs.ox.ac.uk/isg/people/yujiao.zhou/#resources

Strategy	Solved	|Univ|	$t_{\text{avg.}}$
RL Bounds	9	500	5.8
+ EL Lower Bound	12	500	9.7
+ Summarisation	14	60	40.2
+ Dependencies	15	1	2.7

Table 5: Result for UOBM$^-$

materialised in 307s, and it matches the upper bound for 8 of the 10 additional queries; thus, our system could answer 22 of the 24 queries over LUBM(1000) efficiently in 11s on average.[3] For the remaining 2 queries, we could scale to LUBM(100) in reasonable time. On LUBM(100) the gaps contain 29 and 14 tuples respectively, none of which were eliminated by summarisation; however, exploiting dependencies between gap tuples reduced the calls to HermiT to only 3 and 1 respectively, with the majority of time taken in extraction (avg. 86s) and HermiT calls (avg. 384.7s). On LUBM(1000), Pellet ran out of memory. For LUBM(100), it took on average 8.2s to answer the standard queries with an initialisation overhead of 388s. TrOWL timed out after 1h on LUBM(100).

UOBM UOBM is an extension of LUBM (Ma et al. 2006). Query answering over UOBM requires equality reasoning (e.g., to deal with cardinality constraints), which is not natively supported by RDFox,[4] so we have used a slightly weakened ontology UOBM$^-$ for which equality is not required. Materialisation is still fast on UOBM$^-(500)$: it takes 305s (356s) to materialise the basic lower (upper) bound entailments. We have tested the 15 standard queries (see Table 5). The basic lower and upper bounds match for 9 queries, and this increases to 12 when using our refined lower bound (the latter took 312s to materialise); our system is efficient for these queries, with an avg. query answering time of less than 10s over UOBM$^-$(500). For 2 of the remaining queries, summarisation prunes all candidate answers. Avg. times for these queries were under 40s for UOBM$^-$(60). For the one remaining query, summarisation rules out 6245 among 6509 answers in the gap, and the dependency analysis groups all the remaining individuals. HermiT, however, takes 20s to check the representative answer for UOBM$^-$(1), and 2000s for UOBM$^-$(10). Pellet times out even on UOBM$^-(1)$. TrOWL took 237s on average to answer 14 out of the 15 queries over UOBM$^-$(60);[5] however, comparison with our system reveals that TrOWL answers may be neither sound nor complete for most test queries.

Fly Anatomy Ontology Fly Anatomy is a complex ontology, rich in existential axioms, and including a dataset with over 6,000 facts. We tested it with five queries provided by the developers of the ontology. It took 1.7s (5.9s) to materialise lower (upper) bound entailments. The basic lower bounds for all queries are empty, whereas the refined lower bounds (which take 5.7s to materialise) match with the upper bound in all cases; as a result, we can answer the queries in 0.1s on average. Pellet fails to answer queries given a 1h timeout, and TrOWL returns only empty answers.

DBPedia$^+$ In contrast to Fly, the DBPedia dataset is relatively large, but the ontology is simple. To provide a more challenging test, we have used the LogMap ontology matching system (Jiménez-Ruiz et al. 2012) to extend DBPedia with a tourism ontology which contains both disjunctive and existential axioms. Since the tested systems report errors on datatypes, we have removed all axioms and facts involving datatypes. It takes 37.2s (40.7s) to materialise the basic lower (upper) bound entailments. The upper bound was unsatisfiable and it took 55.2s to check satisfiability of the $\mathcal{K}_\bot$ fragment. We queried for instances of all 441 atomic concepts. Bounds matched in 439 cases (using the refined lower bound), and these queries were answered in 0.3s on average. Summarisation filtered out all gap tuples for the other 2 queries; the answer time for these was 58s. Pellet takes 280.9s to initialise and answers each query in an average time of 16.2s. TrOWL times out after 1h.

NPD FactPages The NPD FactPages ontology describes petroleum activities on the Norwegian continental shelf. The ontology is not Horn, and it includes existential axioms. As in the case of DBPedia, we removed axioms involving datatypes. Its dataset has about 4 million triples; it takes 8s (10s) to materialise the lower (upper) bound entailments. The upper bound is unsatisfiable, and it took 60s to check satisfiability of $\mathcal{K}_\bot$. We queried for the instances of the 329 atomic concepts, and could answer all queries using a combination of lower and upper bounds and summarisation in 5s on average. Queries with matching bounds (294 out of 329) could be answered within 0.035s. Pellet took 127s to initialise, and average query answering time was 3s. TrOWL took 1.3s to answer queries on average, and its answers were complete for 320 out of the 329 queries.

8 Discussion

We have proposed an enhanced hybrid approach for query answering over arbitrary OWL 2 ontologies. The approach integrates scalable and complete reasoners to provide pay-as-you-go performance: 772 of the 814 test queries could be answered using highly scalable lower and upper bound computations, 39 of the remaining 42 queries yielded to scalable extraction and summarisation techniques, and even for the remaining 3 queries our fragment extraction and dependency techniques greatly improved scalability. Our approach is complementary to other optimisation efforts, and could immediately benefit from alternative techniques for efficiently computing lower bounds and/or a more efficient OWL reasoner. Furthermore, our technical results are very general, and hold for any language L captured by generalised rules and over which we want to answer queries; our only assumption is the availability of a fully-fledged query engine for L and one for datalog, both used as a "black box".

There are still many possibilities for future work. For the immediate future, our main focus will be improving the fragment extraction and checking techniques so as to improve scalability for the hardest queries.

[3] Avg. query answering times measured after materialisation.

[4] RDFox axiomatises equality as a congruence relation.

[5] An exception is reported for the remaining query.

Acknowledgements

This work was supported by the Royal Society, the EPSRC projects Score!, Exoda, and MaSI^3, and the FP7 project OPTIQUE.

References

Bachmair, L., and Ganzinger, H. 2001. Resolution Theorem Proving. In *Handbook of Automated Reasoning*. Elsevier. 19–99.

Bishop, B.; Kiryakov, A.; Ognyanoff, D.; Peikov, I.; Tashev, Z.; and Velkov, R. 2011. OWLIM: A family of scalable semantic repositories. *Semantic Web* 2(1):33–42.

Cuenca Grau, B.; Motik, B.; Stoilos, G.; and Horrocks, I. 2012. Completeness guarantees for incomplete ontology reasoners: Theory and practice. *J. Artif. Intell. Res. (JAIR)* 43:419–476.

Dolby, J.; Fokoue, A.; Kalyanpur, A.; Kershenbaum, A.; Schonberg, E.; Srinivas, K.; and Ma, L. 2007. Scalable semantic retrieval through summarization and refinement. In *AAAI*, 299–304. AAAI Press.

Dolby, J.; Fokoue, A.; Kalyanpur, A.; Schonberg, E.; and Srinivas, K. 2009. Scalable highly expressive reasoner (SHER). *J. Web Sem.* 7(4):357–361.

Guo, Y.; Pan, Z.; and Heflin, J. 2005. LUBM: A benchmark for OWL knowledge base systems. *J. Web Sem.* 3(2-3):158–182.

Hustadt, U.; Motik, B.; and Sattler, U. 2007. Reasoning in description logics by a reduction to disjunctive datalog. *J. Autom. Reasoning* 39(3):351–384.

Jiménez-Ruiz, E.; Cuenca Grau, B.; Zhou, Y.; and Horrocks, I. 2012. Large-scale interactive ontology matching: Algorithms and implementation. In *ECAI*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 444–449. IOS Press.

Kollia, I., and Glimm, B. 2013. Optimizing SPARQL query answering over OWL ontologies. *J. Artif. Intell. Res. (JAIR)* 48:253–303.

Ma, L.; Yang, Y.; Qiu, Z.; Xie, G. T.; Pan, Y.; and Liu, S. 2006. Towards a complete OWL ontology benchmark. In *ESWC*, volume 4011 of *Lecture Notes in Computer Science*, 125–139. Springer.

Manola, F., and Miller, E. 2004. RDF primer. W3C Recommendation. Available at `http://www.w3.org/TR/rdf-primer/`.

Möller, R.; Neuenstadt, C.; Özçep, Ö. L.; and Wandelt, S. 2013. Advances in accessing big data with expressive ontologies. In *Description Logics*, volume 1014 of *CEUR Workshop Proceedings*, 842–853. CEUR-WS.org.

Motik, B.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; Fokoue, A.; and Lutz, C. 2009. OWL 2 Web Ontology Language Profiles. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-profiles/`.

Motik, B.; Nenov, Y.; Piro, R.; Horrocks, I.; and Olteanu, D. 2014. Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In *AAAI*. AAAI Press.

Motik, B.; Patel-Schneider, P. F.; and Parsia, B. 2009. OWL 2 Web Ontology Language Structural Specification and Functional-style Syntax. W3C Recommendation. Available at `http://www.w3.org/TR/owl2-syntax/`.

Motik, B.; Shearer, R.; and Horrocks, I. 2009. Hypertableau reasoning for description logics. *J. Artif. Intell. Res. (JAIR)* 36:165–228.

Pan, J. Z.; Thomas, E.; and Zhao, Y. 2009. Completeness guaranteed approximations for OWL-DL query answering. In *Description Logics*, volume 477 of *CEUR Workshop Proceedings*. CEUR-WS.org.

Sirin, E.; Parsia, B.; Cuenca Grau, B.; Kalyanpur, A.; and Katz, Y. 2007. Pellet: A practical OWL-DL reasoner. *J. Web Sem.* 5(2):51–53.

Stefanoni, G.; Motik, B.; and Horrocks, I. 2013. Introducing nominals to the combined query answering approaches for EL. In *AAAI*. AAAI Press.

Thomas, E.; Pan, J. Z.; and Ren, Y. 2010. TrOWL: Tractable OWL 2 reasoning infrastructure. In *ESWC (2)*, volume 6089 of *Lecture Notes in Computer Science*, 431–435. Springer.

Tserendorj, T.; Rudolph, S.; Krötzsch, M.; and Hitzler, P. 2008. Approximate OWL-reasoning with Screech. In *RR*, volume 5341 of *Lecture Notes in Computer Science*, 165–180. Springer.

Urbani, J.; van Harmelen, F.; Schlobach, S.; and Bal, H. E. 2011. QueryPIE: Backward reasoning for OWL Horst over very large knowledge bases. In *International Semantic Web Conference (1)*, volume 7031 of *Lecture Notes in Computer Science*, 730–745. Springer.

Urbani, J.; Kotoulas, S.; Maassen, J.; van Harmelen, F.; and Bal, H. E. 2012. WebPIE: A web-scale parallel inference engine using MapReduce. *J. Web Sem.* 10:59–75.

W3C SPARQL Working Group. 2013. SPARQL 1.1 Overview. W3C Recommendation. Available at `http://www.w3.org/TR/sparql11-overview/`.

Wandelt, S.; Möller, R.; and Wessel, M. 2010. Towards scalable instance retrieval over ontologies. *Int. J. Software and Informatics* 4(3):201–218.

Wu, Z.; Eadon, G.; Das, S.; Chong, E. I.; Kolovski, V.; Annamalai, M.; and Srinivasan, J. 2008. Implementing an inference engine for RDFS/OWL constructs and user-defined rules in oracle. In *ICDE*, 1239–1248. IEEE.

Zhou, Y.; Cuenca Grau, B.; Horrocks, I.; Wu, Z.; and Banerjee, J. 2013a. Making the most of your triple store: query answering in OWL 2 using an RL reasoner. In *WWW*, 1569–1580. International World Wide Web Conferences Steering Committee / ACM.

Zhou, Y.; Nenov, Y.; Cuenca Grau, B.; and Horrocks, I. 2013b. Complete query answering over Horn ontologies using a triple store. In *ISWC (1)*, volume 8218 of *Lecture Notes in Computer Science*, 720–736. Springer.