Computing CQ lower-bounds over OWL 2 through approximation to RSA ⋆
Extended version

Federico Igne ⬤, Stefano Germano ⬤, and Ian Horrocks ⬤

Department of Computer Science, University of Oxford, Oxford, UK
`firstname.lastname@cs.ox.ac.uk`

Abstract Conjunctive query (CQ) answering over knowledge bases is an important reasoning task. However, with expressive ontology languages such as OWL, query answering is computationally very expensive. The PAGOdA system addresses this issue by using a tractable reasoner to compute lower and upper-bound approximations, falling back to a fully-fledged OWL reasoner only when these bounds don't coincide. The effectiveness of this approach critically depends on the quality of the approximations, and in this paper we explore a technique for computing closer approximations via RSA, an ontology language that subsumes all the OWL 2 profiles while still maintaining tractability. We present a novel approximation of OWL 2 ontologies into RSA, and an algorithm to compute a closer (than PAGOdA) lower bound approximation using the RSA combined approach. We have implemented these algorithms in a prototypical CQ answering system, and we present a preliminary evaluation of our system that shows significant performance improvements w.r.t. PAGOdA.

Keywords: CQ answering · combined approach · ontology approximation · RSA.

1 Introduction

Conjunctive query (CQ) answering is one of the primary reasoning tasks over knowledge bases for many applications. However, when considering expressive description logic languages, query answering is computationally very expensive, even when considering only complexity w.r.t. the size of the data (*data complexity*). Fully-fledged reasoners oriented towards CQ answering over unrestricted OWL 2 ontologies exist but, although heavily optimised, they are only effective on small to medium datasets. In order to achieve tractability and scalability for the problem, two main approaches are often used: either the expressive power of the input ontology or the completeness of the computed answers is sacrificed.

Using the first approach, query answering procedures have been developed for several fragments of OWL 2 for which CQ answering is tractable with respect to data complexity [1]. Three such fragments have been standardised as *OWL 2 profiles*, and CQ answering techniques for these fragments have been shown to be highly scalable at the expense of expressive power [2,11,12,17,19,18]. Using the second approach, several algorithms have been proposed to compute an approximation of the set of answers to a given CQ. This usually results in computing *a sound subset* of the answers, sacrificing completeness. One such technique is to approximate the input ontology to a tractable fragment, e.g., by dropping all those axioms outside the fragment; a tractable algorithm can then be used to answer CQs over the approximated ontology. This process is clearly sound but possibly incomplete, and hence provides a *lower-bound* answer to any given query.

A particularly interesting approach to CQ answering over unrestricted OWL 2 ontologies, using a combination of the aforementioned techniques, is adopted by PAGOdA [20]. Its "pay-as-you-go" approach allows us to use a Datalog reasoner to handle the bulk of the computation, computing lower and upper approximations of the answers to a query, while relying on a fully-fledged OWL 2 reasoner like HermiT only as necessary to fully answer the query.

While PAGOdA is able to avoid the use of a fully-fledged OWL 2 reasoner in some cases, its performance rapidly deteriorates when the input query requires (extensive) use of the underlying OWL 2 reasoner. Results from our tests show that whenever PAGOdA relies on HermiT to compute the bulk of the answers to a query, computation time is usually prohibitive and sometimes unfeasible. The computation of lower and upper bounds is achieved by under- and over-approximating the ontology into OWL 2 RL so that a tractable reasoner can be used for CQ answering. The tractability of OWL 2 RL is achieved in part by avoiding problematic interactions between axioms that can cause an exponential blow-up of the computation (so-called *and-branching*). As it turns out, this elimination of problematic interactions between axioms is rather coarse, and PAGOdA often ends up falling back to the underlying OWL 2 reasoner even when it is not really needed.

This work expands on this "pay-as-you-go" technique; it aims to improve the lower-bound approximation in PAGOdA, tightening the gap between lower and upper bounds and minimising the use of HermiT. We achieve this by (soundly) approximating the input ontology into RSA [3], an ontology language that subsumes all the OWL 2 profiles, for which CQ answering is still tractable, and for which a CQ answering algorithm based on the *combined approach* has been proposed in [6]. We present a novel algorithm for approximating the input ontology into RSA, and an implementation [10] of the combined approach CQ answering algorithm adapted to the use of RDFox [16,15,13,14] as a backend Datalog reasoner; this includes the design of an improved version of the filtering step for the combined approach, optimised for RDFox. In addition, we streamline the execution of the combined approach by factoring out those steps in the combined approach that are *query independent* to make answering multiple queries

over the same knowledge base more efficient. To summarise (Figure 1), given an OWL 2 ontology, we propose an algorithm to approximate it down to RSA, and compute its canonical model as part of the combined approach algorithm for RSA; we then derive an improved filtering program from the input query that, combined with the canonical model produces a lower-bound of the answers to the query over the original ontology.
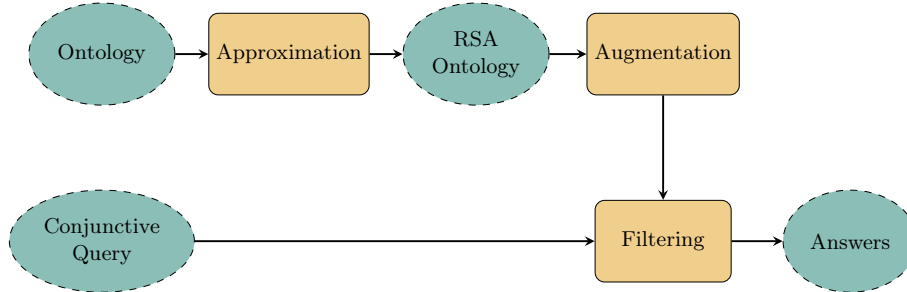

Figure 1. RSAComb Architecture

We have integrated our improved lower bound computation into PAGOdA and carried out a preliminary evaluation to assess its effectiveness. Our experimental results show that the new technique yields significant performance improvements in several important application scenarios.

2   Preliminaries

PAGOdA is a reasoner for sound and complete conjunctive query answering over OWL 2 knowledge bases, adopting a "pay-as-you-go" approach to compute the certain answers to a given query. It uses a combination of a *Datalog reasoner* and a *fully-fledged OWL 2 reasoner*; PAGOdA treats the two systems as black boxes and tries to offload the bulk of the computation to the former and relies on the latter only when necessary. [1]

To achieve this, PAGOdA exploits the Datalog reasoner to compute a lower and upper bound to the certain answers to the input query. If these bounds match, then the query has been fully answered; otherwise the answers in the "gap" between the bounds are further processed and verified against the fully-fledged reasoner. Lower and upper bounds are computed by approximating the input ontology to a logic program and answering the query over the approximations.

---

[1] The capabilities, performance and scalability of PAGOdA inherently depend on the ability of the fully-fledged OWL 2 reasoner in use, and the ability to delegate the workload to a given Datalog reasoner. In the best scenario, with an OWL 2 DL reasoner, PAGOdA is able to answer *internalisable queries* [9].

In the following we provide a brief description of the computation of the lower bound, since some details will be useful later on. See [20] for a more in-depth description of the algorithm and heuristics in use.

Given an *ontology* $\mathcal{O}$ and a CQ q, the *disjunctive Datalog* subset of the input ontology is computed, denoted $\mathcal{O}^{DD}$, by dropping any axiom that does not correspond to a disjunctive Datalog rule. Using a variant of *shifting* [5], $\mathcal{O}^{DD}$ is polynomially transformed in order to eliminate disjunction in the head. The resulting Datalog program $\texttt{shift}(\mathcal{O}^{DD})$ is sound but not necessarily complete for CQ answering. A first materialisation is performed, and the resulting facts are added to the input ontology to obtain $\mathcal{O}'$. Next, the $\mathcal{ELHO}_\perp^r$ [18] subset of $\mathcal{O}'$ is computed[2], denoted $\mathcal{O}'_{\mathcal{EL}}$, by dropping any axiom that is not in $\mathcal{ELHO}_\perp^r$; the final lower bound is then computed by applying the *combined approach* for $\mathcal{ELHO}_\perp^r$ [12,19] to q over $\mathcal{O}'_{\mathcal{EL}}$.

While PAGOdA performs really well on simpler queries over complex OWL 2 ontologies, it can struggle when addressing more complex queries that actually make use of the complexity and expressivity of the underlying ontology language.

To improve PAGOdA's performance and compute a tighter lower-bound we approximate the input ontology to RSA, a tractable ontology language (more expressive than $\mathcal{ELHO}_\perp^r$) based on the Horn-$\mathcal{ALCHOIQ}$ language with additional global restrictions on role interaction. To perform this approximation, we proceed similarly to PAGOdA, by dropping any axiom in the input ontology that is not part of a particular target DL language ($\mathcal{ALCHOIQ}$ in our case) and remove any disjunction in the axioms by means of a shifting step. Finally, we introduce a novel algorithm to approximate the resulting Horn-$\mathcal{ALCHOIQ}$ ontology into RSA by weakening axioms as needed to ensure that the global restrictions on role interactions are satisfied.

Logic programs We assume familiarity with standard concepts of first-order logic (FO) such as term, variable, constant, predicate, atom, literal, logic rule, (stratified) programs. See [6] and Appendix A for a formal introduction to these concepts.

We will call a rule *definite* without negation in its body, and *Datalog* a function-free definite rule. A Datalog rule is *disjunctive* if it admits disjunction in the head. A *fact* is a Datalog rule with an empty body. Given a stratified program $\mathcal{P}$, we denote its *least Herbrand model* (LHM) as $M[\mathcal{P}]$, and define $\mathcal{P}^{\approx,\top}$ the program extended with axiomatisation rules for equality ($\approx$) and truth value ($\top$) in a standard way [6].

Ontologies and conjunctive query answering We define Horn-$\mathcal{ALCHOIQ}$ as the set of axioms that are allowed in the language and specify its semantics by means of translation to definite programs. The definition will fix a *normal form* for this ontology language, and w.l.o.g. we assume any input ontology in Horn-$\mathcal{ALCHOIQ}$ contains only these types of axioms.

---

[2] $\mathcal{ELHO}_\perp^r$ is an OWL 2 EL fragment, for which CQ answering is tractable.

Table 1. Normalised Horn-$\mathcal{ALCHOIQ}$ axioms and their translation in definite rules.

Axioms α	Definite rules $\pi(\alpha)$
(R1) R^-	$R(x,y) \to R^-(y,x); R^-(y,x) \to R(x,y)$
(R2) $R \sqsubseteq S$	$R(x,y) \to S(x,y)$
(T1) $\prod_{i=1}^n A_i \sqsubseteq B$	$\bigwedge_{i=1}^n A_i(x) \to B(x)$
(T2) $A \sqsubseteq \{a\}$	$A(x) \to x \approx a$
(T3) $\exists R.A \sqsubseteq B$	$R(x,y) \wedge A(y) \to B(x)$
(T4) $A \sqsubseteq\, \leq 1R.B$	$A(x) \wedge R(x,y) \wedge B(y) \wedge R(x,z) \wedge B(z) \to y \approx z$
(T5) $A \sqsubseteq \exists R.B$	$A(x) \to R(x, f_{R,B}^A(x)) \wedge B(f_{R,B}^A(x))$
(A1) $A(a)$	$\to A(a)$
(A2) $R(a,b)$	$\to R(a,b)$

Let N_C, N_R, N_I be countable disjoint sets of concepts names, role names and individuals respectively. We define a *role* as an element of $N_R \cup \{R^- \mid R \in N_R\}$, where R^- is called *inverse role*. We also introduce a function $Inv(\cdot)$ closed for roles s.t. $\forall R \in N_R : Inv(R) = R^-, Inv(R^-) = R$. An *RBox* $\mathcal{R}$ is a finite set of axioms of type (R2) in Table 1 where R, S are roles. We denote $\sqsubseteq_\mathcal{R}^*$ as a minimal relation over roles closed by reflexivity and transitivity s.t. $R \sqsubseteq_\mathcal{R}^* S$, $Inv(R) \sqsubseteq_\mathcal{R}^* Inv(S)$ hold if $R \sqsubseteq S \in \mathcal{R}$. A *TBox* $\mathcal{T}$ is a set of axioms of type (T1-5) where $A, B \in N_C$, $a \in N_I$ and R is a role. An *ABox* $\mathcal{A}$ is a finite set of axiom of type (A1-2) with $A \in N_C$, $a, b \in N_I$ and $R \in N_R$. An *ontology* is a set of axioms $\mathcal{O} = \mathcal{A} \cup \mathcal{T} \cup \mathcal{R}$. Finally, if we consider $\mathcal{ALCHOIQ}$, the TBox is further extended with an additional axiom type $A \sqsubseteq \bigsqcup_{i=1}^n B_i$ allowing disjunction on the right-hand side.

A *conjunctive query (CQ)* q is a formula $\exists \vec{y}.\psi(\vec{x}, \vec{y})$ with $\psi(\vec{x}, \vec{y})$ a *conjunction* of function–free atoms over $\vec{x} \cup \vec{y}$, and $\vec{x}, \vec{y}$ are called *answer variables* and *bounded variables* respectively. Queries with an empty set of answer variables are called *boolean conjunctive queries (BCQ)*. Let π be the translation of axioms into definite rules defined in Table 1; by extension we write $\pi(\mathcal{O}) = \{\pi(\alpha) \mid \alpha \in \mathcal{O}\}$. An ontology $\mathcal{O}$ is satisfiable if $\pi(\mathcal{O}^{\approx, \top}) \not\models \exists y.\bot(y)$. A tuple of constants $\vec{c}$ is an *answer* to q if $\mathcal{O}$ is *unsatisfiable* or $\pi(\mathcal{O}^{\approx, \top}) \models \exists \vec{y}.\psi(\vec{c}, \vec{y})$. The set of answers to a query q is written $cert(q, \mathcal{O})$.

3 Combined approach for CQ answering in RSA

RSA is a class of ontology languages designed to subsume all OWL 2 profiles, while maintaining tractability of standard reasoning tasks like CQ answering. The RSA ontology language is designed to avoid interactions between axioms that can result in the ontology being satisfied only by exponentially large (and potentially infinite) models. This problem is often called *and-branching* and can be caused by interactions between axioms of type (T5) with either axioms (T3) and (R1), or axioms (T4), in Table 1.

RSA includes all axioms in Table 1, restricting their interaction to ensure a polynomial bound on model size [3].

Definition 1. *A role R in $\mathcal{O}$ is* unsafe *if it occurs in axioms (T5), and there is a role S s.t. either of the following holds:*

1. *$R \sqsubseteq_{\mathcal{R}}^{*} Inv(S)$ and S occurs in an axiom (T3) with left-hand side concept $\exists S.A$ where $A \neq \top$;*
2. *S is in an axiom (T4) and $R \sqsubseteq_{\mathcal{R}}^{*} S$ or $R \sqsubseteq_{\mathcal{R}}^{*} Inv(S)$.*

A role R in $\mathcal{O}$ is safe *if it is not unsafe.*

Note that, by definition all OWL 2 profiles ($\mathcal{RL}$, $\mathcal{EL}$ and $\mathcal{QL}$) contain only *safe* roles.

Definition 2. *Let PE and E be fresh binary predicates, let U be a fresh unary predicate, and let $u_{R,B}^{A}$ be a fresh constant for each concept $A, B \in N_C$ and each role $R \in N_R$. A function π_{RSA} maps each (T5) axiom $\alpha \in \mathcal{O}$ to $A(x) \rightarrow R(x, u_{R,B}^{A}) \wedge B(u_{R,B}^{A}) \wedge PE(x, u_{R,B}^{A})$ and $\pi(\alpha)$ otherwise. The program $\mathcal{P}_{RSA}$ consists of $\pi_{RSA}(\alpha)$ for each $\alpha \in \mathcal{O}$, rule $U(x) \wedge PE(x, y) \wedge U(y) \rightarrow E(x, y)$ and facts $U(u_{R,B}^{A})$ for each $u_{R,B}^{A}$, with R unsafe.*

Let M_{RSA} be the LHM of $\mathcal{P}_{RSA}^{\approx, \top}$. Then, $G_{\mathcal{O}}$ is the digraph with an edge (c, d) for each $E(c, d)$ in M_{RSA}. Ontology $\mathcal{O}$ is equality-safe *if for each pair of atoms $w \approx t$ (with w and y distinct) and $R(t, u_{R,B}^{A})$ in M_{RSA} and each role S s.t. $R \sqsubseteq Inv(S)$, it holds that S does not occur in an axiom (T4) and for each pair of atoms $R(a, u_{R,B}^{A}), S(u_{R,B}^{A}, a)$ in M_{RSA} with $a \in N_I$, there is no role T such that both $R \sqsubseteq_{\mathcal{R}}^{*} T$ and $S \sqsubseteq_{\mathcal{R}}^{*} Inv(T)$ hold.*

We say that $\mathcal{O}$ is RSA *if it is* equality-safe *and $G_{\mathcal{O}}$ is an oriented forest.*

The fact that $G_{\mathcal{O}}$ is a DAG ensures that the LHM $M[\mathcal{P}_{\mathcal{O}}]$ is finite, whereas the lack of "diamond-shaped" subgraphs in $G_{\mathcal{O}}$ guarantees polynomiality of $M[\mathcal{P}_{\mathcal{O}}]$. The definition gives us a programmatic procedure to determine whether an Horn-$\mathcal{ALCHOIQ}$ ontology is RSA.

Theorem 1 ([6], Theorem 2). *If $\mathcal{O}$ is RSA, then $|M[\mathcal{P}_{\mathcal{O}}]|$ is polynomial in $|\mathcal{O}|$.*

3.1   RSA combined approach

Following is a summary of the combined approach (with filtration) for conjunctive query answering for RSA presented in [6]. This consists of two main steps to be offloaded to a Datalog reasoner able to handle *negation* and *function symbols*.

The first step computes the canonical model of an RSA ontology over an extended signature (introduced to deal with *inverse roles* and *directionality* of newly generated binary atoms). The computed canonical model is not universal and, as such, might lead to spurious answers in the evaluation of CQs.

The second step of the computation performs a filtration of the computed answers to identify only the *certain answers* to the input query.

Canonical model computation The computation of the canonical model for an ontology $\mathcal{O}$ is performed by computing the LHM of a translation of the ontology into definite rules. The translation for each axiom type is given in [6] and is an enhanced version of the translation given in Table 1 where axioms of type (T5) are *skolemised* if the role involved is unsafe, and *constant skolemised* otherwise[3]. We call this translation $E_{\mathcal{O}}$ and denote the computed canonical model as $M[E_O]$. $M[E_{\mathcal{O}}]$ is polynomial in $|\mathcal{O}|$ and if $\mathcal{O}$ is satisfiable; $\mathcal{O} \models A(c)$ iff $A(c) \in M[E_{\mathcal{O}}]$ (see [6], Theorem 3).

Filtering spurious answers For the filtering step, a *query dependent* logic program $\mathcal{P}_q$ is introduced to filter out all spurious answers to an input query q over the extended canonical model $M[E_{\mathcal{O}}]$ computed in the previous section.

The program identifies and discards any match with a *fork/cycle* involving *anonymous terms*, scenarios that cannot be possibly enforced by a TBox alone and hence correspond to spurious answers induced by the canonical model. For more details on the construction of $\mathcal{P}_q$, please refer to Appendix B and [6], Section 4.

Let $\mathcal{P}_q$ be the filtering program for q, and $\mathcal{P}_{\mathcal{O},q} = E_{\mathcal{O}} \cup \mathcal{P}_q$, then we know that $M[\mathcal{P}_{\mathcal{O},q}]$ is polynomial in $|\mathcal{O}|$ and exponential in $|q|$ (see [6], Theorem 4). We obtain a "guess and check" algorithm that leads to an NP-completeness result for BCQs [6]. The algorithm first materialises $E_{\mathcal{O}}$ in polynomial time and then guesses a match σ to q over the materialisation; finally it materialises $(\mathcal{P}_{\mathcal{O},q})\sigma$.

Theorem 2 ([6], Theorem 5). *Checking whether $\mathcal{O} \models q$ with $\mathcal{O}$ an RSA ontology and q a BCQ is* NP-*complete in combined complexity.*

3.2   Improvements to the combined approach

RDFox adoption One first technical difference from the original work on the RSA combined approach is the adoption of RDFox as a Datalog reasoner instead of DLV. RDFox provides support stratified negation but does not provide *direct* support for function symbols. We simulate function symbols using the Skolemisation feature, making it possible to associate a unique term to a unique tuple of terms. Doing so, we keep somewhat closer to the realm of description logics since RDF triples are a first-class citizen and only atoms with arity ≤ 2 are allowed.

Improved filtering program RDFox is primarily an RDF reasoner and its ability to handle Datalog (with a set of useful extension) makes it able to capture the entire $\mathcal{RL}$ profile. We were able to partially rewrite and simplify the filtering step in the RSA combined approach: a first rewriting step gets rid of all atoms with arity greater than 2; filtering rules are then greatly simplified by making extensive use of the Skolemisation function provided by RDFox, hence avoiding

---

[3] A more detailed description of this step is described in Appendix B.

some expensive *joins* that would slow down the computation (see [6], Section 5, especially the results for query q_1).

Example 1. We show rule (3c) in the original filtering program (w.r.t. a query $q(\vec{x}) = \psi(\vec{x}, \vec{y})$ where $\vec{x} = x_1, \ldots, x_m$, $\vec{y} = y_1, \ldots, y_n$), along with its simplification steps. Rule (3c) computes the transitive closure of a predicate *id*, keeping track of identity between anonymous terms w.r.t. a specific match for the input query.

$$id(\vec{x}, \vec{y}, u, v), id(\vec{x}, \vec{y}, v, w) \rightarrow id(\vec{x}, \vec{y}, u, w) \tag{1}$$

Provided we have access to a function KEY to compute a new term that uniquely identifies a tuple of terms, we can turn any n-ary atom into a set of n atoms of arity 2. E.g., an atom $P(x, y, z)$ becomes $P_1(k, x), P_2(k, y), P_3(k, z)$, where $k = \text{KEY}(x, y, z)$ and P_n, for $1 \leq n \leq arity(P)$, are fresh predicates of arity 2. Rule (1) then becomes

$$\begin{aligned}
id_1(k, x_1), \ldots, &id_{m+n}(k, y_n), id_{m+n+1}(k, u), id_{m+n+2}(k, v), \\
id_1(j, x_1), \ldots, &id_{m+n}(j, y_n), id_{m+n+1}(j, v), id_{m+n+2}(j, w), \\
l := \text{KEY}(\vec{x}, \vec{y}, u, w) &\rightarrow id_1(l, x_1), \ldots, id_{m+n}(l, y_n), \\
&id_{m+n+1}(l, v), id_{m+n+2}(l, w)
\end{aligned} \tag{2}$$

Using the SKOLEM function[4] in RDFox, we are able to reduce the arity of a predicate P (see predicate *id* in Rule (3)) without having to introduce $arity(P)$ fresh predicates. Also note how joins over multiple terms (*id* joining over $(\vec{x}, \vec{y})$ in (1)) can now be rewritten into simpler joins (*id* joining over a single term k)[5].

$$\begin{aligned}
id(k, j), \text{SKOLEM}(\vec{x}, \vec{y}, u, v, j), id(k, l), \text{SKOLEM}(\vec{x}, \vec{y}, v, w, l), \\
\text{SKOLEM}(\vec{x}, \vec{y}, u, w, t) \rightarrow id(k, t)
\end{aligned} \tag{3}$$

$\square$

Query independent computation One of the main features of the combined approach for conjunctive query answering over knowledge bases is its two-stage process. The first step, i.e., the computation of the canonical model, is notably dependent solely on the input knowledge base; similarly the filtration step is only dependent on the query.

The two-stage nature of the approach can be implemented directly in RDFox using different *named graphs* to store the materialisation of the combined approach and the filtering step respectively. Assigning different named graphs (here essentially used as *namespaces*) to different parts of the computation allows us to treat them independently, managing partial results of a computation, dropping or preserving them. This means that for every new query over the same

---

[4] `https://docs.oxfordsemantic.tech/tuple-tables.html#rdfox-skolem`

[5] Rule 3 showcases how the SKOLEM function can be used in both directions: given a sequence of terms, we can *pack* them into a single fresh term; give a previously skolemised term, we can *unpack* it to retrieve the corresponding sequence of terms.

knowledge base we only need to perform the filtering step. Once the answers to a particular query are computed we can simply drop the named graph corresponding to the filtering step for that query and start fresh for the next one.

Note that RDFox supports parallel computation as well, and since the filtering steps for a set of queries are independent of each other we can execute multiple filtering steps in parallel to take advantage of hardware parallelisation (see Section 7).

Top and equality axiomatisation RDFox has built-in support for $\top$ (*top*, *truth* or `owl:Thing`) and *equality* (`owl:sameAs`), so that $\top$ automatically *subsumes* any new class introduced within an RDF triple, and equality between terms is always consistent with its semantics.

In both cases we are not able to use these features directly: in the case of top axiomatisation, we import axioms as Datalog rules, which are not taken into consideration when RDFox derives new $\top$ subsumptions; in the case of equality axiomatisation, the feature cannot be enabled along other features like *aggregates* and *negation-as-failure*, which are extensively used in our system.

To work around this, we introduce the axiomatisation for both predicates explicitly. For more details on the set of rules used for this, we refer the reader to Appendix C.

3.3   Additional fixes

Our work also includes a few clarifications on theoretical definitions and their implementation.

In the canonical model computation in [6], the `notIn` predicate is introduced to simulate the semantics of set membership and in particular the meaning of `notIn[a, b]` is "a is not in set b". During the computation of the canonical model program we have complete knowledge of any set that might be used in a `notIn` atom. For each such set S, and for each element $a \in S$, we introduce the fact `in[`a`,`S`]` in the canonical model. We then replace any occurrence of `notIn[?X, ?Y]` in the original program $E_{\mathcal{O}}$ with `NOT in[?X, ?Y]`, where `NOT` is the operator for *negation-as-failure* in RDFox.

A similar approach has been used to redefine and implement predicate `NI`, representing the set of *non-anonymous* terms in the materialised canonical model. We enumerate the elements of this set introducing the following rule:

```
NI[?Y] :- named[?X], owl:sameAs[?X, ?Y] .
```

where `named` is a predicate representing the set of constants in the original ontology.

A final improvement has been made on the computation of the `cycle` function during the canonical model computation. The original definition involved a search over all possible triples (A, R, B) where $A, B \in N_C$, $R \in N_R$ in the original ontology. We realised that traversing the whole space would significantly slow down the computation, and is *not* necessary; we instead restrict our search

over all (A, R, B) triples that appear in a (T5) axiom $A \sqsubseteq \exists R.B$ in the original normalised ontology.

4   Integration of RSA into PAGOdA

As described in Section 2 and in [20], the process of computing the lower-bound of the answers to an input query involves (1) approximating the input ontology to *disjunctive Datalog* and further processing the rules to obtain a Datalog program; (2) approximating the input ontology to $\mathcal{ELHO}^r_\perp$ and applying the corresponding combined approach presented in [18].

These two approximations are handled independently, by means of materialisation in the first case, and the combined approach in the second; this allows PAGOdA to avoid having to deal with *and-branching* and the resulting intractability of most reasoning problems (see Definition 1). The RL and $\mathcal{ELHO}^r_\perp$ approximations used by PAGOdA eliminate *all* interactions between axioms (T5) and either axioms (T4) or axioms (T3) and (R1)[6]. However, not all such interactions cause an exponential jump in complexity, and PAGOdA's filtering of such cases is unnecessarily coarse. In RSA, interactions between these types of axioms are allowed but limited, and the filtering of those cases that may lead to and-branching is based on a fine-grained analysis of *role safety*; hence the lower-bound produced by the RSA combined approach is often larger than the one computed by PAGOdA.

In the following we show how to integrate the aforementioned combined approach for RSA into the lower-bound computation procedure.

4.1   Lower-bound computation

We take different steps depending on how the input ontology can be classified. We assume w.l.o.g. that the input ontology is consistent and normalised.

If the input ontology is inside one of the OWL 2 profiles, we simply use the standard PAGOdA algorithm to compute the answers to the query. Note that this check is purely syntactic over the normalised ontology.

If the first check fails (i.e., the ontology is not in any of the profiles), we check whether the ontology is in RSA. This can be done using the polynomial algorithm presented in [6] and reimplemented in our system (Section 3). If the input ontology is inside RSA we are able to apply the combined approach for query answering directly and collect the sound and complete set of answers to the input query. Efficiency of the RSA combined approach, compared to PAGOdA, mainly depends on the input ontology and the type of query; as explained earlier, this new approach is particularly effective when query answers depend on interactions between axioms that belong to different profiles. Based on our tests (Section 6), if PAGOdA is not able to compute the complete set of answers by means of

---

[6] Note that OWL 2 RL does not allow axioms (T5) and OWL 2 EL (which contains $\mathcal{ELHO}^r_\perp$) does not allow axioms (T4) or inverse roles (R1).

computing its lower and upper-bounds and instead relies on HermiT to finalise the computation, then the RSA approach can be up to 2 orders of magnitude faster in returning the complete set of answers.

If the input ontology is not RSA, we approximate it to $\mathcal{ALCHOIQ}$. The approximation is carried out by removing any axiom in the normalised ontology that is not part of $\mathcal{ALCHOIQ}$. We then eliminate any axiom involving *disjunction* on the right-hand side using a *program shifting* technique. Note that this approach is the same used by PAGOdA to handle disjunctive rules in the original lower-bound computation. This procedure guarantees to produce a sound (but not necessarily complete) approximation w.r.t. CQ answering. The resulting ontology is in Horn-$\mathcal{ALCHOIQ}$.

The next step involves the approximation from Horn-$\mathcal{ALCHOIQ}$ to RSA. We achieve this using a novel algorithm to approximate an Horn-$\mathcal{ALCHOIQ}$ ontology to RSA in polynomial time (Section 5). Then, we can apply the RSA combined approach to the resulting approximated ontology.

We can then summarise the overall procedure in the following steps:

1. If the input ontology is inside one of the OWL 2 profiles, we run the standard PAGOdA algorithm. In this scenario, PAGOdA is able to compute complete query answers using a tractable procedure for the relevant profile.
2. If the input ontology is in RSA, we run the combined approach algorithm described in Section 3.1. This will return the complete set of answers to the input query.
3. If the ontology is not RSA we substitute the lower-bound computation process in PAGOdA with the following steps:
   (a) We approximate the input ontology to Horn-$\mathcal{ALCHOIQ}$ by first discarding any non-$\mathcal{ALCHOIQ}$ axioms, and then using a *shifting technique* to eliminate disjunction on the right-hand side of axioms.
   (b) We use a novel algorithm to approximate the Horn-$\mathcal{ALCHOIQ}$ ontology to RSA (see Section 5).
   (c) We apply the RSA combined approach to obtain a lower-bound of the answers to the query.
   (d) We continue with the standard PAGOdA procedure to compute the complete set of answers.

The approximation algorithm guarantees that the combined approach applied over the approximated RSA ontology will return a subset (lower-bound) of the answers to the query over the original ontology, i.e., $cert(q, \mathcal{O}_{RSA}) \subseteq cert(q, \mathcal{O})$, where q is the input CQ, $\mathcal{O}$ is the original ontology and $\mathcal{O}_{RSA}$ is its RSA approximation. Let ℓ_P be the lower-bound computed by PAGOdA, and ℓ_R be the lower-bound computed by our procedure; then we have in general that $l_P \subseteq l_R$.

5   Horn-$\mathcal{ALCHOIQ}$ to RSA approximation

One of the steps involved in the process of integrating the RSA combined approach in PAGOdA is the approximation of the input ontology to RSA. In the

original algorithm, PAGOdA would approximate the ontology by removing most of the out-of-profile axioms and deal in a more fine-grained manner with *existential quantification* and *union*.

Note that we can't directly apply this approach to the new system since the definition of RSA is not purely syntactical and an approximation to RSA by removing out-of-language axioms is not possible. Instead, we propose an algorithm that first approximates the input ontology to an Horn-$\mathcal{ALCHOIQ}$ ontology $\mathcal{O}$ and then further approximates $\mathcal{O}$ to RSA using a novel technique acting on the custom dependency graph $G_{\mathcal{O}}$ presented in Definition 2.

In the following we provide a description of the algorithm to approximate a Horn-$\mathcal{ALCHOIQ}$ ontology $\mathcal{O}_S$ into an RSA ontology $\mathcal{O}_T$ such that $cert(q, \mathcal{O}_T) \subseteq cert(q, \mathcal{O}_S)$.

Given an Horn-$\mathcal{ALCHOIQ}$ ontology $\mathcal{O}$, checking if $\mathcal{O}$ is RSA consists of:

1. checking whether $G_{\mathcal{O}}$ is an *oriented forest*;
2. checking whether $\mathcal{O}$ is *equality safe*.

We first consider (1). If $\mathcal{O}$ is not RSA, then it presents at least one cycle in $G_{\mathcal{O}}$. The idea is to disconnect the graph and propagate the changes into the original ontology. A way of doing this is to delete some nodes $u^A_{R,B}$ from the graph to break the cycles. By definition of $u^A_{R,B}$, the node uniquely identifies an axiom $A \sqsubseteq \exists R.B$ of type (T5) in $\mathcal{O}$ and hence, removing the axiom will break the cycle in $G_{\mathcal{O}}$. We can gather a possible set of nodes that disconnect the graph by using a slightly modified version of a BFS visit. The action of deleting the nodes from the graph can be then propagated to the ontology by removing the corresponding T5 axioms. Due to monotonicity of first order logic, deleting axioms from the ontology clearly produces a lower-bound approximation of the ontology w.r.t. conjunctive query answering.

Next, we need to deal with *equality safety* (2). The following step can be performed to ensure this property:

– delete any T4 axiom that involves a role S such that there exists $w \approx t$ (with w and y distinct) and $R(t, u^A_{R,B})$ in M_{RSA} and $R \sqsubseteq Inv(S)$;
– if there is a pair of atoms $R(a, u^A_{R,B}), S(u^A_{R,B}, a)$ in M_{RSA} with $a \in N_I$ and a role T such that both $R \sqsubseteq^*_{\mathcal{R}} T$ and $S \sqsubseteq^*_{\mathcal{R}} Inv(T)$ hold, then remove some axiom (R2) to break the derivation chain that deduces either $R \sqsubseteq^*_{\mathcal{R}} T$ or $S \sqsubseteq^*_{\mathcal{R}} Inv(T)$.

Note that the set of nodes that are computed by the graph visit to disconnect all cycles in a graph is not, in general, unique, and hence might not guarantee the tightest lower-bound on the answers to a given query. On the other hand this gives us a simple way of determining whether the approximation will affect the resulting answer computation. It is easy to see that if the deleted axioms are not involved in the computation of the answers to the input query, the set of answers will be left unaltered and will correspond to the set of answers to the query w.r.t. to the original ontology.

Algorithm 1: Approximate an Horn-$\mathcal{ALCHOIQ}$ ontology to RSA

Input: Ontology dependency graph G

1 let N be the set of nodes in G;

2 let C be an empty set;

3 **foreach** *node n in N* **do**

4   **if** *n is not* discovered **then**

5   let S be an empty stack;

6   push n in S;

7   **while** S *is not* empty **do**

8   pop v from S;

9   **if** v *is not* discovered **then**

10   label v as *discovered*;

11   let *adj* be the set of nodes adjacent to v;

12   **if** *any node in adj is* discovered **then**

13   push v in C;

14   **else**

15   **foreach** *node w in adj* **do**

16   push w in S;

17 remove C from G;

With reference to the PAGOdA approach, $cert(q, O_P) \subseteq cert(q, O_T)$ for both approximations O_P to Datalog and $\mathcal{ELHO}^r_\perp$ used by PAGOdA for the lower-bound computation.

6   Evaluation

Implementation details As discussed above, we provide our own implementation of the combined approach algorithm for RSA (called RSAComb) [10]: on the one hand, the implementation presented in [6] is not available, and on the other hand we wanted to take advantage of a tight integration with RDFox and simplify the subsequent integration with PAGOdA.

Our implementation is written in Scala and uses RDFox[7] as the underlying Datalog reasoner. At the time of writing, development and testing have been carried out using Scala v2.13.5 and RDFox v4.1. Scala allows us to easily interface with Java libraries and in particular the OWLAPI [8] for easy ontology manipulation. We communicate with RDFox through the Java wrapper API provided with the distribution.

Testing environment All experiments were performed on an Intel(R) Xeon(R) CPU E5-2640 v3 @ 2.60GHz with 16 real cores, extended via hyper-threading to

---

[7] https://www.oxfordsemantic.tech/product

Table 2. Comparison of answering time for PAGOdA and our system with multiple queries over LUBM

ABox size	Query ID	Answers	PAGOdA preprocessing (s)	PAGOdA answering (s)	RSAComb preprocessing (s)	RSAComb answering (s)
100	34	4	196	109	41	2
	31	18		159		3
	36	72927		219		154
200	34	4	461	2303	78	5
	31	18		7535		5
	36	145279		-		613
300	34	4	824	10563	112	7
	31	18		23309		7
	36	217375		-		1227
400	34	4	1023	14527	153	10
	31	18		-		11
	36	290516		-		2593
500	34	4	1317	23855	206	12
	31	18		-		13
	36	363890		-		4174
600	34	4	1738	33322	210	16
	31	18		-		15
	36	436961		-		4302
700	34	4	2390	-	252	19
	31	18		-		21
	36	509401		-		4667
800	34	4	3619	-	260	22
	31	18		-		21
	36	582658		-		6105

32 virtual cores, 512 GB of RAM and running Fedora 33, kernel version 5.8.17-300.fc33.x86_64. While PAGOdA is inherently single core, we were able to make use of the multicore CPU and distribute the computation across cores, especially for intensive tasks offloaded to RDFox.

Comparison with PAGOdA To compare our system against PAGOdA, we performed our tests on the LUBM ontology [7], using the queries and datasets provided by the PAGOdA distribution[8], plus an additional query to test performance with large answer sets. LUBM is not in Horn-$\mathcal{ALCHOIQ}$ (because of some *role transitivity* axiom) but contains only safe roles. Datasets from the PAGOdA distribution are automatically generated with the LUBM data generator[9], with the parameter indicating the number of universities ranging from 100 up to 800, with steps of 100.

---

[8] https://www.cs.ox.ac.uk/isg/tools/PAGOdA/
[9] http://swat.cse.lehigh.edu/projects/lubm/uba1.7.zip

For queries where PAGOdA does not require HermiT, the performance of PAGOdA and RSAComb is very similar. In Table 2, we show the results for three queries where PAGOdA does require HermiT to complete the computation (i.e. the query is classified as "FullReasoning"): query 31 and 34 are queries provided by the PAGOdA distribution and query 36 is an additional query that we introduced to test the system on a query with a much higher number of answers. We provide these queries in Appendix D. For each query we provide in order: the size of the ABox, the number of answers to the query, preprocessing and answering time in PAGOdA, preprocessing time in our system (including approximation to RSA and computation of the canonical model), answering time for RSAComb (including filtering program computation and filtering step, answers gathering). Execution time had a timeout set to 10h and timed-out computation is indicated in the tables with a hyphen "-".

The results clearly show how our system is able to compute the complete set of answers to the queries in considerably less time and without the need of a fully-fledged reasoner like HermiT. For larger datasets, the introduction of our system makes the difference between feasibility and unfeasibility. Focusing on query 36, we are able to limit the impact that a high number of answers to a query has on performance.

Another important aspect shown here is that, even when factoring out the preprocessing time for both systems (we can argue that this step can be pre-computed offline when the ontology is fixed), we still achieve considerably faster results, especially when it comes to datasets of larger size.

7   Discussion and Future Work

We presented a novel algorithm to approximate an OWL 2 ontology into RSA, and an algorithm to compute a lower-bound approximation of the answers to a CQ using the RSA combined approach. We showed that this lower-bound is stricter than the one computed by PAGOdA and provided an implementation of the algorithms in a prototypical CQ answering system.

We are already working on additional improvements to the approximation algorithm to RSA; the current visit of the dependency graph to detect the axioms to delete might be improved with different heuristics and might in some cases take into account the input query (deleting axioms that are not necessarily involved in the computation of the answers). A similar approach could be introduced to integrate RSA in the upper-bound of the answers to a query, with the ultimate goal of improving this step in PAGOdA as well.

On a different note, we hope to obtain additional improvements in performance in the current implementation of the RSA combined approach by introducing parallel execution of filtering steps for different input queries, using the *named graph* functionality provided by RDFox.

Finally, we would like to explore the possibility to avoid the conversion of axioms into Datalog overall and come up with a different encoding of the RSA

combined approach that would make use of the built-in support for OWL 2 $\mathcal{RL}$ currently present in RDFox.

References

1. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. In: Proceedings, Tenth International Conference on Principles of Knowledge Representation and Reasoning, Lake District of the United Kingdom, June 2-5, 2006. pp. 260–270. AAAI Press (2006)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. J. Autom. Reasoning **39**(3), 385–429 (2007). https://doi.org/10.1007/s10817-007-9078-x
3. Carral, D., Feier, C., Cuenca Grau, B., Hitzler, P., Horrocks, I.: Pushing the boundaries of tractable ontology reasoning. In: The Semantic Web - ISWC 2014 - 13th International Semantic Web Conference, Riva del Garda, Italy, October 19-23, 2014. Proceedings, Part II. Lecture Notes in Computer Science, vol. 8797, pp. 148–163. Springer (2014). https://doi.org/10.1007/978-3-319-11915-1_10
4. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Ma, L.: Scalable semantic retrieval through summarization and refinement. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, July 22-26, 2007, Vancouver, British Columbia, Canada. pp. 299–304. AAAI Press (2007), `http://www.aaai.org/Library/AAAI/2007/aaai07-046.php`
5. Eiter, T., Fink, M., Tompits, H., Woltran, S.: On eliminating disjunctions in stable logic programming. In: Dubois, D., Welty, C.A., Williams, M. (eds.) Principles of Knowledge Representation and Reasoning: Proceedings of the Ninth International Conference (KR2004), Whistler, Canada, June 2-5, 2004. pp. 447–458. AAAI Press (2004), `http://www.aaai.org/Library/KR/2004/kr04-047.php`
6. Feier, C., Carral, D., Stefanoni, G., Cuenca Grau, B., Horrocks, I.: The combined approach to query answering beyond the OWL 2 profiles. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 2971–2977. AAAI Press (2015)
7. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. J. Web Semant. **3**(2-3), 158–182 (2005). https://doi.org/10.1016/j.websem.2005.06.005, `https://doi.org/10.1016/j.websem.2005.06.005`
8. Horridge, M., Bechhofer, S.: The OWL API: A java API for OWL ontologies. Semantic Web **2**(1), 11–21 (2011). https://doi.org/10.3233/SW-2011-0025, `https://doi.org/10.3233/SW-2011-0025`
9. Horrocks, I., Tessaris, S.: A conjunctive query language for description logic aboxes. In: Kautz, H.A., Porter, B.W. (eds.) Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on on Innovative Applications of Artificial Intelligence, July 30 - August 3, 2000, Austin, Texas, USA. pp. 399–404. AAAI Press / The MIT Press (2000), `http://www.aaai.org/Library/AAAI/2000/aaai00-061.php`
10. Igne, F., Germano, S., Horrocks, I.: RSAComb - Combined approach for Conjunctive Query answering in RSA (Jun 2021). https://doi.org/10.5281/zenodo.5047811, `https://doi.org/10.5281/zenodo.5047811`

11. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to query answering in dl-lite. In: Principles of Knowledge Representation and Reasoning: Proceedings of the Twelfth International Conference, KR 2010, Toronto, Ontario, Canada, May 9-13, 2010. AAAI Press (2010)
12. Lutz, C., Toman, D., Wolter, F.: Conjunctive query answering in the description logic EL using a relational database system. In: Boutilier, C. (ed.) IJCAI 2009, Proceedings of the 21st International Joint Conference on Artificial Intelligence, Pasadena, California, USA, July 11-17, 2009. pp. 2070–2075 (2009), `http://ijcai.org/Proceedings/09/Papers/341.pdf`
13. Motik, B., Nenov, Y., Piro, R., Horrocks, I.: Handling owl: sameas via rewriting. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 231–237. AAAI Press (2015)
14. Motik, B., Nenov, Y., Piro, R., Horrocks, I.: Incremental update of datalog materialisation: the backward/forward algorithm. In: Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, January 25-30, 2015, Austin, Texas, USA. pp. 1560–1568. AAAI Press (2015)
15. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. pp. 129–137. AAAI Press (2014)
16. Nenov, Y., Piro, R., Motik, B., Horrocks, I., Wu, Z., Banerjee, J.: Rdfox: A highly-scalable RDF store. In: The Semantic Web - ISWC 2015 - 14th International Semantic Web Conference, Bethlehem, PA, USA, October 11-15, 2015, Proceedings, Part II. Lecture Notes in Computer Science, vol. 9367, pp. 3–20. Springer (2015)
17. Ren, Y., Pan, J.Z., Guclu, I., Kollingbaum, M.J.: A combined approach to incremental reasoning for EL ontologies. In: Web Reasoning and Rule Systems - 10th International Conference, RR 2016, Aberdeen, UK, September 9-11, 2016, Proceedings. Lecture Notes in Computer Science, vol. 9898, pp. 167–183. Springer (2016). https://doi.org/10.1007/978-3-319-45276-0_13
18. Stefanoni, G., Motik, B.: Answering conjunctive queries over EL knowledge bases with transitive and reflexive roles. CoRR **abs/1411.2516** (2014)
19. Stefanoni, G., Motik, B., Horrocks, I.: Introducing nominals to the combined query answering approaches for EL. In: desJardins, M., Littman, M.L. (eds.) Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence, July 14-18, 2013, Bellevue, Washington, USA. AAAI Press (2013), `http://www.aaai.org/ocs/index.php/AAAI/AAAI13/paper/view/6156`
20. Zhou, Y., Cuenca Grau, B., Nenov, Y., Kaminski, M., Horrocks, I.: Pagoda: Pay-as-you-go ontology query answering using a datalog reasoner. J. Artif. Intell. Res. **54**, 309–367 (2015)

Appendices

A Additional preliminaries

Logic programs We define a *rule* as an expression of the form $\varphi(\vec{x}, \vec{y}) \rightarrow \psi(\vec{x})$, with $\varphi(\vec{x}, \vec{y})$ a conjunction of literals over variables $\vec{x} \cup \vec{y}$ and $\psi(\vec{x})$ a non-empty conjunction of atoms over $\vec{x}$. Given a role r, we denote $head(r)$ the set of atoms in $\psi(\vec{x})$, and $body^+(r)$ ($body^-(r)$) the set of *positive* (*negative*) literals in $\varphi(\vec{x}, \vec{y})$. We will call *definite* a rule without negation in its body, and *Datalog* a function-free definite rule. The definition can be trivially extended to sets of rules. A *fact* is a Datalog rule with an empty body.

A *program* $\mathcal{P}$ is a set of rules. Let $pred(X)$ be the set of predicate in X (either a set of atoms, a rule or a program). A *stratification* of a program $\mathcal{P}$ is a function $\delta : pred(\mathcal{P}) \rightarrow \{1, \ldots, k\}$ with $k \leq |pred(\mathcal{P})|$, s.t. for every rule $r \in \mathcal{P}$ and $P \in pred(head(r))$ it holds:

- for every $Q \in pred(body^+(r))$, $\delta(Q) \leq \delta(P)$;
- for every $Q \in pred(body^-(r))$, $\delta(Q) < \delta(P)$;

The *stratification partition* of $\mathcal{P}$ induced by δ is the sequence $(\mathcal{P}_1, \ldots, \mathcal{P}_k)$ with each $\mathcal{P}_i$ be the set of rules $r \in \mathcal{P}$ s.t. $\max_{a \in head(r)}(\delta(pred(a))) = i$. Programs $\mathcal{P}_i$ are called *strata* of $\mathcal{P}$. A program is *stratified* if it admits a stratification. All definite programs are stratified.

PAGOdA Following is a slightly more detailed description of the procedure adopted by PAGOdA to compute the answers to a query. See [20] for a more in-depth description of the algorithm and heuristics in use.

Given an *ontology* $\mathcal{O} = (\mathcal{A}, \mathcal{T}, \mathcal{R})$ [10] and a query q, PAGOdA executes the following steps in order to compute the answers to q w.r.t. $\mathcal{O}$:

1. the Datalog reasoner is exploited to compute a *lower bound L^q* and an *upper bound U^q* to the answers to the query q. This is achieved by approximating the input ontology $\mathcal{O}$ into a tractable language to be handled by the Datalog reasoner. Depending on the approximation procedure, running the query over the approximated ontology will result in either a lower or an upper bound of the certain answers to the query. We mainly focus on the steps taken to compute the lower bound since some details will be useful later on in the paper:

   (a) the *disjunctive Datalog* subset of the input ontology is computed, in symbols $\mathcal{O}^{DD}$, dropping any axiom that does not correspond to a disjunctive Datalog rule;

   (b) using a variant of *shifting*[5], $\mathcal{O}_{DD}$ is polynomially transformed in order to eliminate disjunction in the head. The resulting ontology $\mathtt{shift}(\mathcal{O}_{DD})$ is sound but not necessarily complete for CQ answering;

---

[10] In the following we consider the input knowledge base to be *consistent* and *normalised*. This is ensured by PAGOdA, preprocessing the input ontology and checking for consistency.

(c) a first materialization is performed, i.e. $M_1 = M[\texttt{shift}(\mathcal{O}_{DD})]$. The resulting facts are added to the input ontology to obtain $\mathcal{O}' = (\mathcal{A} \cup M_1, \mathcal{T}, \mathcal{R})$;

(d) the $\mathcal{ELHO}_\perp^r$ [18] subset of $\mathcal{O}'$ is computed, in symbols $\mathcal{O}'_{EL}$, dropping any axiom that is not in $\mathcal{ELHO}_\perp^r$;

(e) the *combined approach* for $\mathcal{ELHO}_\perp^r$ [12,19] is used to compute the answers to the query q over $\mathcal{O}'_{\mathcal{EL}}$.

2. if lower and upper bound coincide (i.e. $L^q = U^q$) then the Datalog reasoner was able to provide a sound and complete set of answers to the input query. The computation terminates;

3. otherwise, the "gap" between the upper and lower bound (i.e., $G^q = U^q \setminus L^q$) is a set of answers that need to be verified against the knowledge base using a fully fledged OWL 2 reasoner. The Datalog reasoner is again exploited for this step to compute a *subset* $\mathcal{K}^q$ of the knowledge base $\mathcal{K}$ that is enough to check whether the answers in G^q are certain or spurious;

4. for each $\vec{a} \in G^q$, the fully fledged reasoner is used to check whether $\mathcal{K}^q \models q(\vec{a})$. This process is further optimized by reducing the number of answers in G^q that need to be checked; a *summarization* technique[4] is used for this, along with the use of algorithm to keep track of the dependency between answers;

5. once all spurious answers have been removed from G^q, $L^q \cup G^q$ is returned.

B Combined Approach for RSA

Here we provide a more detailed definition of the canonical model computation in RSA. First we define the Datalog program $E_{\mathcal{O}}$ used to compute the canonical model for $\mathcal{O}$.

Table 3. Translation of Horn-$\mathcal{ALCHOIQ}$ axioms to build $E_{\mathcal{O}}$

Axioms in $\mathcal{O}$	LP rules
non-(T5) axiom α	$\pi(\alpha)$
$R \sqsubseteq S, * \in \{f, b\}$	$R^*(x, y) \to S^*(x, y)$
R role, $* \in \{f, b\}$	$R^*(x, y) \to R(x, y)$ $R^f(x, y) \to Inv(R)^b(y, x)$ $R^b(x, y) \to Inv(R)^f(y, x)$
(T5) axiom, R unsafe	$A(x) \to R^f(x, f_{R,B}^A(x)) \wedge B(f_{R,B}^A(x))$
(T5) axiom, R safe	$A(x) \wedge \texttt{notIn}(x, \texttt{unfold}(A, R, B)) \to R^f(x, v_{R,B}^{A,0}) \wedge B(v_{R,B}^{A,0})$ if $R \in \texttt{confl}(R)$, for every $i = 0, 1$: $A(v_{R,B}^{A,i}) \to R^f(v_{R,B}^{A,i}, v_{R,B}^{A,i+1}) \wedge B(v_{R,B}^{A,i+1})$ for every $x \in \texttt{cycle}(A, R, B)$: $A(x) \to R^f(x, v_{R,B}^{A,1}) \wedge B(v_{R,B}^{A,1})$

Definition 3. *Let* $\mathtt{confl}(R)$ *be the set of roles S s.t. $R \sqsubseteq_{\mathcal{R}}^* T$ and $S \sqsubseteq_{\mathcal{R}}^* Inv(T)$ for some T. Let prec be a strict total order on triples (A, R, B), with R safe and A, B concept names in $\mathcal{O}$. For each (A, R, B), let $v_{R,B}^{A,0}$, $v_{R,B}^{A,1}$ and $v_{R,B}^{A,2}$ be fresh constants; let $\mathtt{self}(A, R, B)$ be the smallest set containing $v_{R,B}^{A,0}$ and $v_{R,B}^{A,1}$ if $R \in \mathtt{confl}(R)$; and let $\mathtt{cycle}(A, R, B)$ be the smallest set of terms containing, for each $S \in \mathtt{confl}(R)$,*

 - *$v_{S,C}^{D,0}$ if $(A, R, B) \prec (D, S, C)$;*
 - *$v_{S,C}^{D,1}$ if $(D, S, C) \prec (A, R, B)$;*
 - *$f_{S,C}^D(v_{S,C}^{D,0})$ and each $f_{T,E}^F(v_{S,C}^{D,0})$ s.t. $u_{S,C}^D \approx u_{T,E}^F$ is in M_{RSA}, if S is unsafe.*

Finally, $\mathtt{unfold}(A, R, B) = \mathtt{self}(A, R, B) \cup \mathtt{cycle}(A, R, B)$.

Let R^f and R^b be fresh binary predicates for each role R in $\mathcal{O}$, let $\mathtt{NI}$ be a fresh unary predicate, and $\mathtt{notIn}$ be a built-in predicate which holds when the first argument is not *an element of the set given as the second element. Let $\mathcal{P}$ be the smallest program with a rule $\rightarrow \mathtt{NI}(a)$ for each constant a and all rules in Table 3. We define $E_{\mathcal{O}} = \mathcal{P}^{\approx,\top}$.*

The canonical model for an RSA input ontology is defined as $M[E_{\mathcal{O}}]$.

Theorem 3 (from [6], Theorem 3). *The following holds:*

 (i) *$M[E_{\mathcal{O}}]$ is polynomial in $|\mathcal{O}|$;*
 (ii) *$\mathcal{O}$ is satisfiable iff $E_{\mathcal{O}} \not\models \exists y.\bot(y)$;*
 (iii) *if $\mathcal{O}$ is satisfiable, $\mathcal{O} \models A(c)$ iff $A(c) \in M[E_{\mathcal{O}}]$;*
 (iv) *there are no terms s, t and role R s.t. $E_{\mathcal{O}} \models R^f(s, t) \wedge R^b(s, t)$.*

Given a query $q = \exists \vec{y}.\psi(\vec{x}, \vec{y})$, the original filtering program introduced in [6] is presented in Table 4

Following is the definition of $\mathcal{P}_q$ and its extension $\mathcal{P}_{\mathcal{O},q}$ with $E_{\mathcal{O}}$ from Def. 3, which can then be used to compute the set of certain answers to q w.r.t. $\mathcal{O}$.

Definition 4. *Let $q = \exists \vec{y}.\psi(\vec{x}, \vec{y})$ be a CQ, let $\mathtt{QM}$, $\mathtt{sp}$, and $\mathtt{fk}$ be fresh predicates of arity $|\vec{x}| + |\vec{y}|$, let $\mathtt{id}$, $\mathtt{AQ}^*$, $\mathtt{TQ}^*$ with $* \in \{f, b\}$ be fresh predicates of arity $|\vec{x}| + |\vec{y}| + 2$, let $\mathtt{Ans}$ be a fresh predicate of arity $|\vec{x}|$, let $\mathtt{named}$ be a fresh unary predicate, and let U be a set of fresh variables s.t. $|U| \geq |\vec{y}|$. Then, $\mathcal{P}_q$ is the smallest program with all rules in Table 4, and $\mathcal{P}_{\mathcal{O},q}$ is defined as $E_{\mathcal{O}} \cup \mathcal{P}_q$.*

Let $\mathcal{P}_q$ be the filtering program for q, and $\mathcal{P}_{\mathcal{O},q} = E_{\mathcal{O}} \cup \mathcal{P}_q$, then we know that $M[\mathcal{P}_{\mathcal{O},q}]$ is polynomial in $|\mathcal{O}|$ and exponential in $|q|$ (see Theorem 4 in [6]).

Theorem 4. *Let $\mathcal{P}_q$ be the filtering program for q, and $\mathcal{P}_{\mathcal{O},q} = E_{\mathcal{O}} \cup \mathcal{P}_q$. It holds that [6]: (i) $\mathcal{P}_{\mathcal{O},q}$ is* stratified*; (ii) $M[\mathcal{P}_{\mathcal{O},q}]$ is polynomial in $|\mathcal{O}|$ and exponential in $|q|$; (iii) if $\mathcal{O}$ is satisfiable, $\vec{x} \in cert(q, \mathcal{O})$ iff $\mathcal{P}_{\mathcal{O},q} \models \mathtt{Ans}(\vec{x})$.*

Table 4. Rules in $\mathcal{P}_Q$. Variables u, v, w from U are distinct.

(1) $\psi(\vec{x}, \vec{y}) \to \texttt{QM}(\vec{x}, \vec{y})$
(2) $\to \texttt{named}(a)$ for each constant a in $\mathcal{O}$
(3a) $\texttt{QM}(\vec{x}, \vec{y}), not\ \texttt{NI}(y_i) \to id(\vec{x}, \vec{y}, i, i)$ for each $1 \le i \le \|\vec{y}\|$
(3b) $id(\vec{x}, \vec{y}, u, v) \to id(\vec{x}, \vec{y}, v, u)$
(3c) $id(\vec{x}, \vec{y}, u, v), id(\vec{x}, \vec{y}, v, w) \to id(\vec{x}, \vec{y}, u, w)$
(4a) for all $R(s, y_i)$, $S(t, y_j)$ in q with $y_i, y_j \in \vec{y}$
$\quad R^f(s, y_i) \wedge S^f(t, y_j) \wedge id(\vec{x}, \vec{y}, i, j) \wedge not\ s \approx t \to \texttt{fk}(\vec{x}, \vec{y})$
(4b) for all $R(s, y_i)$, $S(y_j, t)$ in q with $y_i, y_j \in \vec{y}$
$\quad R^f(s, y_i) \wedge S^b(y_j, t) \wedge id(\vec{x}, \vec{y}, i, j) \wedge not\ s \approx t \to \texttt{fk}(\vec{x}, \vec{y})$
(4c) for all $R(y_i, s)$, $S(y_j, t)$ in q with $y_i, y_j \in \vec{y}$
$\quad R^b(y_i, s) \wedge S^b(y_j, t) \wedge id(\vec{x}, \vec{y}, i, j) \wedge not\ s \approx t \to \texttt{fk}(\vec{x}, \vec{y})$
for all $R(y_i, y_j)$, $S(y_k, y_l)$ in q with $y_i, y_j, y_k, y_l \in \vec{y}$
(5a) $R^f(y_i, y_j) \wedge S^f(y_k, y_l) \wedge id(\vec{x}, \vec{y}, j, l) \wedge y_i \approx y_k \wedge not\ \texttt{NI}(y_i) \to id(\vec{x}, \vec{y}, i, k)$
(5b) $R^f(y_i, y_j) \wedge S^b(y_k, y_l) \wedge id(\vec{x}, \vec{y}, j, k) \wedge y_i \approx y_l \wedge not\ \texttt{NI}(y_i) \to id(\vec{x}, \vec{y}, i, l)$
(5c) $R^b(y_i, y_j) \wedge S^b(y_k, y_l) \wedge id(\vec{x}, \vec{y}, i, k) \wedge y_j \approx y_l \wedge not\ \texttt{NI}(y_j) \to id(\vec{x}, \vec{y}, j, l)$
(6) for each $R(y_i, y_j)$ in q with $y_i, y_j \in \vec{y}$ and $* \in \{f, b\}$
$\quad R^*(y_i, y_j) \wedge id(\vec{x}, \vec{y}, i, v) \wedge id(\vec{x}, \vec{y}, j, w) \to \texttt{AQ}^*(\vec{x}, \vec{y}, v, w)$
for each $* \in \{f, b\}$
(7a) $\texttt{AQ}^*(\vec{x}, \vec{y}, u, v) \to \texttt{TQ}^*(\vec{x}, \vec{y}, u, v)$
(7a) $\texttt{AQ}^*(\vec{x}, \vec{y}, u, v) \wedge \texttt{TQ}^*(\vec{x}, \vec{y}, v, w) \to \texttt{TQ}^*(\vec{x}, \vec{y}, u, w)$
(8a) $\texttt{QM}(\vec{x}, \vec{y}) \wedge not\ \texttt{named}(x) \to \texttt{sp}(\vec{x}, \vec{y})$ for each $x \in \vec{x}$
(8b) $\texttt{fk}(\vec{x}, \vec{y}) \to \texttt{sp}(\vec{x}, \vec{y})$
(8c) $\texttt{TQ}^*(\vec{x}, \vec{y}, v, v) \to \texttt{sp}(\vec{x}, \vec{y})$ for each $* \in \{f, b\}$
(9) $\texttt{QM}(\vec{x}, \vec{y}) \wedge not\ \texttt{sp}(\vec{x}, \vec{y}) \to \texttt{Ans}(\vec{x})$

We can then build a worst-case exponential algorithm that, given an ontology $\mathcal{O}$ and a CQ q, it materialises $\mathcal{P}_{\mathcal{O},q}$ and returns all instances of predicate $\texttt{Ans}$. This procedure can be adapted to obtain a "guess and check" algorithm that leads to an NP-completeness result for BCQs [6]. The algorithm first materialises $E_{\mathcal{O}}$ in polynomial time and then guesses a match σ to q over the materialization; finally it materialises $(\mathcal{P}_{\mathcal{O},q})\sigma$.

Theorem 5 (from [6]). *Checking whether $\mathcal{O} \models q$ with $\mathcal{O}$ a RSA ontology and q a BCQ is* NP*-complete in combined complexity.*

C   Improvements to the combined approach

Top and equality axiomatisation are performed as follows. For every *concept name* $C \in N_C$ and for every *role name* $R \in N_R$ in the input ontology, we add the following rules to RDFox:

```
owl:Thing[?X] :- C[?X] .
owl:Thing[?X], owl:Thing[?Y] :- R[?X, ?Y] .
```

This gives us the correct semantics for `owl:Thing`.

To axiomatise equality we introduce a new role `congruent` that represents equality between two terms, to avoid unwanted interactions with RDFox's own built-in predicate `owl:sameAs`.

We make the role *reflexive, symmetric* and *transitive*:

```
congruent[?X, ?X] :- owl:Thing[?X] .
congruent[?Y, ?X] :- congruent[?X, ?Y] .
congruent[?X, ?Z] :- congruent[?X, ?Y], congruent[?Y, ?Z] .
```

and introduce substitution rules to complete the axiomatization. For every *concept name* $C \in N_C$ and for every *role name* $R \in N_R$ in the input ontology, we add:

```
C[?Y] :- C[?X], congruent[?X, ?Y] .
R[?Z, ?Y] :- R[?X, ?Y], congruent[?X, ?Z] .
R[?X, ?Z] :- R[?X, ?Y], congruent[?Z, ?Z] .
```

D   Evaluation

We provide below the queries used for the comparison between RSAComb and PAGOdA. Prefixes for the queries are the following

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
```

Query 31 is:

```
SELECT ?X
WHERE {
    ?X ub:publicationAuthor ?Z .
    ?X ub:publicationAuthor <http://www.Department0.University0.edu/FullProfessor0> .
    ?Y ub:member ?Z .
    ?Y rdf:type ub:ResearchGroup
}
```

Query 34 is:

```
SELECT ?X
WHERE {
    <http://www.Department0.University0.edu> ub:member ?X .
    ?W ub:member ?X .
    ?W rdf:type ub:ResearchGroup .
    ?X ub:takesCourse ?Y .
    ?Z ub:teacherOf ?Y .
    ?Z rdf:type ub:FullProfessor
}
```

Query 36 is:

```
SELECT ?X
WHERE {
    ?Y ub:member ?X .
    ?Y rdf:type ub:ResearchGroup .
}
```