RSAComb: Combined Approach for CQ Answering in RSA *

Federico Igne[1 [0000−0002−2790−7513]], Stefano Germano[1 [0000−0001−6993−0618]], and Ian Horrocks[1 [0000−0002−2685−7462]]

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

Abstract The combined approach is a well-known technique used to address the problem of conjunctive query (CQ) answering over knowledge bases. Various versions of the approach exist for relevant fragments of OWL 2. In this paper we focus on the combined approach for CQ answering over RSA, a class of ontologies that extends the OWL 2 profiles while maintaining the tractability of standard reasoning tasks. The algorithm was first presented in [4], but to the best of our knowledge a stable implementation is not currently available. We present RSAComb, a novel implementation of the algorithm that introduces several improvements and fixes to the original approach and uses RDFox as the underlying Datalog reasoner. As well as providing high performance, the extended features of RDFox allow for an optimised implementation of the filtration step. We developed the system with modularity and stability in mind so that it can be used as a standalone tool or integrated into other software as a library. We include an extensive evaluation of the system, focusing on performance and scalability.

Keywords: CQ answering · combined approach · RSA· RDFox.

1 Introduction

Conjunctive query (CQ) answering is a primary reasoning task 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*).

In order to achieve tractability and scalability for the problem, one main approach is to limit the expressive power of the class of ontologies considered.

Query answering algorithms have been developed for several fragments of OWL 2 for which CQ answering is tractable with respect to *data complexity*

* This work was supported by the AIDA project (Alan Turing Institute), the SIRIUS Centre for Scalable Data Access (Research Council of Norway), Samsung Research UK, Siemens AG, and the EPSRC projects AnaLOG (EP/P025943/1), OASIS (EP/S032347/1) and UK FIRES (EP/S019111/1).

[1]. Three such fragments have been standardized as *OWL 2 profiles*, and CQ answering techniques for these fragments have been shown to be highly scalable at the expense of expressive power [2,7,8,9,15,16,18,17].

An interesting fragment of OWL 2, tractable for standard reasoning tasks, is RSA, an ontology language that subsumes all the OWL 2 profiles, first presented in [3] and for which a CQ answering algorithm based on the *combined approach* was proposed in [4]. Alas, the proof of concept implementation used for the experimental results showed in [4] is currently not available.

In this paper we introduce RSAComb, an efficient implementation of the combined approach algorithm for RSA. We developed the system with *modularity*, *extensibility*, and *integration* in mind and the software offers both an intuitive command-line interface and a Scala API to be used as a self-contained library.

The new implementation features RDFox [14,13,10,11,12] as an underlying Datalog reasoner and introduces several improvements, both technical and theoretical and a few minor fixes. Among other changes we propose an improved version of the filtering step for the combined approach, optimised for RDFox.

The system accepts *any* OWL 2 ontology and includes a customisable approximation step to RSA. In addition, we streamlined 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. Furthermore, we performed an extensive set of experiments on the system. An analysis of the most relevant results is provided in Section 5.

RSAComb has been released as free and open source software. Source code and documentation are available at `https://github.com/KRR-Oxford/RSAComb`.

2 Preliminaries

We assume familiarity with standard concepts of first-order logic (FO), with OWL 2, the (Horn-)$\mathcal{ALCHOIQ}$ fragment and OWL 2 profiles, and with the definition of CQ, CQ answering and combined approach for CQ answering.

Combined approach in RSA RSA, first introduced in [3], is a class of ontology languages subsuming all OWL 2 profiles, while maintaining tractability of standard reasoning tasks. RSA includes all axioms in Horn-$\mathcal{ALCHOIQ}$ (listed in Table 1), restricting their interaction to ensure a polynomial bound on model size. 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.

The combined approach for CQ answering in RSA consists of two main steps. 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

Table 1. Translation of normalized Horn-$\mathcal{ALCHOIQ}$ axioms into rules. From [4].

Axiom/expr.		Definite rules
(R1)	R^-	$R(x,y) \rightarrow R^-(y,x); R^-(y,x) \rightarrow R(x,y)$
(R2)	$R \sqsubseteq S$	$R(x,y) \rightarrow S(x,y)$
(T1)	$\bigsqcap_{i=1}^{n} A_i \sqsubseteq B$	$\bigwedge_{i=1}^{n} A_i(x) \rightarrow B(x)$
(T2)	$A \sqsubseteq \{a\}$	$A(x) \rightarrow x \approx a$
(T3)	$\exists R.A \sqsubseteq B$	$R(x,y) \wedge A(y) \rightarrow B(x)$
(T4)	$A \sqsubseteq \leq 1R.B$	$A(x) \wedge R(x,y) \wedge B(y) \wedge R(x,z) \wedge B(z) \rightarrow y \approx z$
(T5)	$A \sqsubseteq \exists R.B$	$A(x) \rightarrow R(x, f^A_{R,B}(x)) \wedge B(f^A_{R,B}(x))$
(A1)	$A(a)$	$\rightarrow A(a)$
(A2)	$R(a,b)$	$\rightarrow R(a,b)$

second step of the computation performs a filtration of the computed answers to identify only the *certain answers* to the input query. Both of these steps are offloaded to a Datalog reasoner able to handle *negation as failure* and *function symbols*.

For a definition of the RSA class of ontologies, and a detailed description of the combined approach algorithm, we refer the reader to [3,4].

3 Design and architecture

In this section we give an overview of the system, focusing on the overall architecture and design choices that we had to make along the way.

Figure 1 summarises the workflow of the system: (i) normalisation and customisable approximation steps approximate an unrestricted OWL 2 ontology to RSA; (ii) the canonical model is then computed for the resulting ontology; (iii) a Datalog filtering program is derived from the input query and is combined with the canonical model to produce the set of *certain* answers to the input query over the approximated ontology. Depending on how the approximation is performed, the set of answers returned might have different properties (e.g., the approximation currently provided computes a lower bound of the answers to the query over the original ontology).

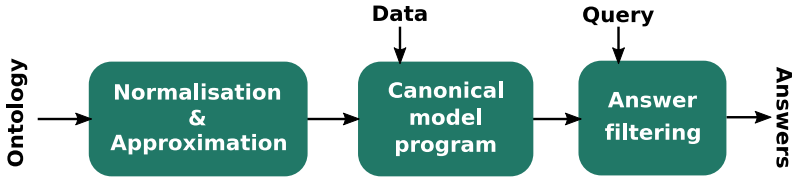


Figure 1. RSAComb architecture and workflow of the system.

It is worth noting that, in this scenario, steps (i),(ii) are *query independent*, while step (iii) is *ontology independent*. As such, when multiple queries are submitted, steps (i-ii) can be performed "offline" and only the third step needs to be performed for each input query.

Our implementation uses RDFox to perform steps (ii) and (iii).

3.1 Principles

We built the system around these general principles:

Modularity the code should be modular and different steps in the algorithm should be as independent of each other as possible. It should be easy to reimplement (or enhance) an intermediate step of the algorithm as long as the *signature* and the *interface* with the system as a whole remain unaltered. We achieved this by an extensive use of Scala *traits*, building a collection of interfaces that describe the behaviour of the different actors that take part in the execution of the combined approach for RSA. As explained in the following sections, the integration with RDFox was also key to provide a good level of modularity to the systems.

Scalability the system has to be able to scale efficiently even for large amounts of data. We factor parts of the computation and reuse previous results whenever possible. A more detailed analysis on the scalability of the system is reported in Section 5.

Integration it should be equally possible to use the system as a self-contained application or integrate it in another system. As such, our software presents a simple but effective command line interface alongside a well-structured set of classes exposing all the necessary tools to work with RSA ontology, while hiding unnecessary implementation details. The different steps can also be disabled for user convenience.

3.2 Normalisation and approximation

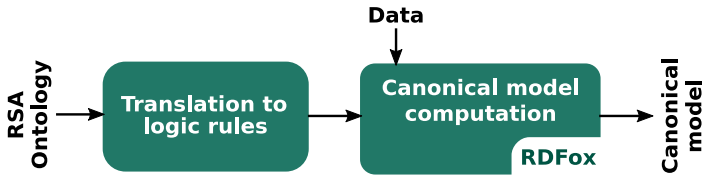
The process of importing the input ontology into the system is performed using the OWLAPI. Note that only the ontology (TBox, RBox) is imported while the data (ABox) provided as separate files is loaded "on demand" directly into RDFox to maximize performance.

Our system performs an optimised normalisation to reduce the input ontology to the syntax introduced in Table 1. The ontology is then approximated to RSA. These steps allow the input of unrestricted OWL 2 ontologies.

Since the approximation of an unrestricted OWL 2 ontology to RSA can be highly application-dependent, users can easily provide their own implementation

or turn it off completely when dealing with RSA ontologies. We provide a sound approximation to RSA, which means that query answers for non-RSA ontologies will be sound but possibly incomplete (i.e., we compute a lower bound answer). The algorithm is based on the idea of deleting nodes to reduce the dependency graph built from the input ontology [4] to a *tree*. The action of deleting 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.

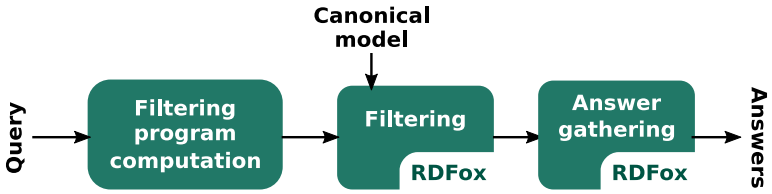
3.3 Canonical model computation



The computation of the canonical model involves the conversion of the input RSA ontology into *logic rules* as described in Section 3 in [4]. The system performs the conversion and then offloads the materialisation of the rules, combined with the input data, to the underlying Datalog reasoner.

Since the canonical model is query independent, this process can be performed once and the result can be cached and reused for every subsequent query over the same input ontology. We achieve this using RDFox's support for RDF named graphs, which enables us to perform operations on specific "named" subsets of the data. Further operations on the graph operate and produce additional data on a different named graph, leaving the materialised canonical model intact.

3.4 Filtering program and answer computation



Answer filtration involves the computation of the filtering program from the input query, the filtering of the materialised canonical model and the final process of gathering the answers.

Our system performs the translation of the query into a set of logic rules. As noted before, this step was reworked to be completely ontology independent[1]. To

[1] See [4] for the definition of the original filtering program, and Section 4.2 for more details on the improvements introduced.

achieve this we moved the process of introducing some ontology-dependent set of facts to the computation of the canonical model. Then, the filtering program is loaded into the Datalog reasoner and the materialisation is *updated* taking into account the newly introduced rules. The triples (ABox facts) produced by this second round of materialisation are stored in a separate named graph to keep the product of filtration separate from the canonical model. This is possible because the signature of the atoms derived in the filtering program is separate from the signature of the canonical model.

Here we can see the real advantage of using named graphs to separate the outcomes of different steps; when processing a new query, the only step we need to take is dropping the named graph associated with the filtration from the previous query, leaving unaltered all other triples. Better yet, here we have the possibility to execute queries in parallel, each one associated with a separate filtering program and hence a different named graph. The materialisation update for each of the queries is isolated and does not interfere with the other processes.

At this point, the task of gathering the answers to the query over the input knowledge base is reduced to querying a materialised named graph for the atoms representing the certain answers, which is straightforward for any materialisation-based Datalog reasoner. Note that this step is slightly different from the original implementation in [4], where the underlying Datalog reasoner most likely returned a (filtered) materialisation instead.

4 Improvements to the combined approach

Here is a description of some of the more significant improvements introduced[2] w.r.t. to the original approach presented in [4].

4.1 RDFox adoption

A main technical difference from the original work on the RSA combined approach is the adoption of RDFox as a Datalog reasoner instead of DLV. This choice has several advantages, but we had to overcome some challenges.

RDFox does not provide *direct* support for function symbols (which are required in the computation of the canonical model), but we can mimic this using the *Skolemization capabilities* offered by the tool. In particular a logic rule with function symbols

```
A(?X) -> R(?X,f(?X)), B(f(?X)) .
```

is turned into

```
A(?X), SKOLEM("f",?X,?Y) -> R(?X,?Y), B(?Y) .
```

[2] For an up-to-date list of changes, fixes, and improvements we refer the reader to the project repository.

where `SKOLEM` is an RDFox built-in atom that computes a unique identifier from a sequence of terms (`"f"` and `?X` in this example), storing it in the last term of the atom (`?Y` in this case).

It is worth noting that the `SKOLEM` functionality is in general more powerful than a *hash function*[3] since RDFox guarantees a bijective relation between a specific sequence and its identifier. In other words, given the special atom `SKOLEM(?X,?Y,?Z,?K)`, we can compute `?K` knowing the sequence of terms `?X`, `?Y`, `?Z`, or alternatively "unpack" `?K` into its original sequence `?X`, `?Y`, `?Z`. We use this property to optimise the filtering step.

Another benefit of the introduction of RDFox is the ability to organize the computation in *named graphs*, as described in Section 3.3, to reuse partial results in the computation when necessary.

In general, the introduction of RDFox, and a clever use of its features, allowed us to keep the system somewhat closer to the realm of description logics[4].

4.2 Improved filtering program

RDFox is primarily an RDF reasoner and its ability to handle (extended) Datalog 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) a first rewriting step gets rid of all atoms with arity greater than 2 through *reification*; b) filtering rules are then greatly simplified by making extensive use of the `SKOLEM` functionality introduced above[5].

Example 1. Consider rule (3c) from the original filtering program in [4] (w.r.t. a query $q(\vec{x}) = \psi(\vec{x}, \vec{y})$ where $\vec{x} = x_1, \ldots, x_m$, $\vec{y} = y_1, \ldots, y_n$). 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.

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

Given 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 using a well-known technique called *reification*. E.g., an atom $P(x, y, z)$ becomes $P_1(k, x), P_2(k, y), P_3(k, z)$, where $k = \texttt{KEY}(x, y, z)$ and P_n, for $1 \leq n \leq arity(P)$, are fresh predicates of arity 2. Rule (1) then becomes

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

[3] By "hash function" we mean any function that can map values of arbitrary size to fixed-size values.

[4] RDF triples are first-class citizens and only atoms with arity ≤ 2 are allowed.

[5] Thus, avoiding some expensive *joins* that would slow down the computation (see Section 5 in [4], especially the results for query q_1).

Using `SKOLEM`[6], we are able to reduce the arity of a predicate P (see predicate id in Rule (3)) without having to introduce $arity(P)$ additional fresh predicates. 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) while the use of `SKOLEM` does not introduce additional join operation[7].

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

□

4.3 Top and equality axiomatisation

RDFox has built-in support for $\top$ (*top*, *truth* or `owl:Thing`) and different kinds of *equality* (using `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.

Unfortunately, in both cases we are not able to use these features directly. We need to import axioms as Datalog rules, and those are not considered when RDFox derives new $\top$ subsumptions. The equality feature cannot be enabled along with *negation-as-failure*, which is required for the combined approach and is extensively used in our system. To work around this, we introduce the axiomatisation for both concepts explicitly as explained in [4].

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`.

For *equality* axiomatisation, we make `owl:sameAs` *reflexive*, *symmetric* and *transitive* as follows:

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

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

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

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

[7] Rule 3 showcases how this functionality can be used in both "directions": given a previously skolemized term, we can *unpack* it in a single operation to retrieve the corresponding sequence of terms and given a sequence of terms, we can *pack* them into a new fresh term.

4.4 Additional fixes

We have also clarified and refined some specific details of the theoretical definitions and their implementation reported in [4].

In the canonical model computation in [4], 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 the `NI` predicate, 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 the `named` atom represents 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 normalized ontology.

5 Evaluation

Implementation details Our implementation is written in Scala and uses RDFox[8] 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 with the OWLAPI [6] for easy ontology manipulation. We communicate with RDFox through the Java wrapper API provided by 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 32 virtual cores, 512 GB of RAM and running Fedora 33, kernel version 5.8.17-300.fc33.x86_64.

The system has been tested against LUBM [5] and Reactome[9]. Both ontologies and associated datasets are taken from the PAGOdA distribution[10]. LUBM

[8] `https://www.oxfordsemantic.tech/product`
[9] `https://elixir-europe.org/platforms/data/core-data-resources`
[10] `https://www.cs.ox.ac.uk/isg/tools/PAGOdA/2015/jair/`

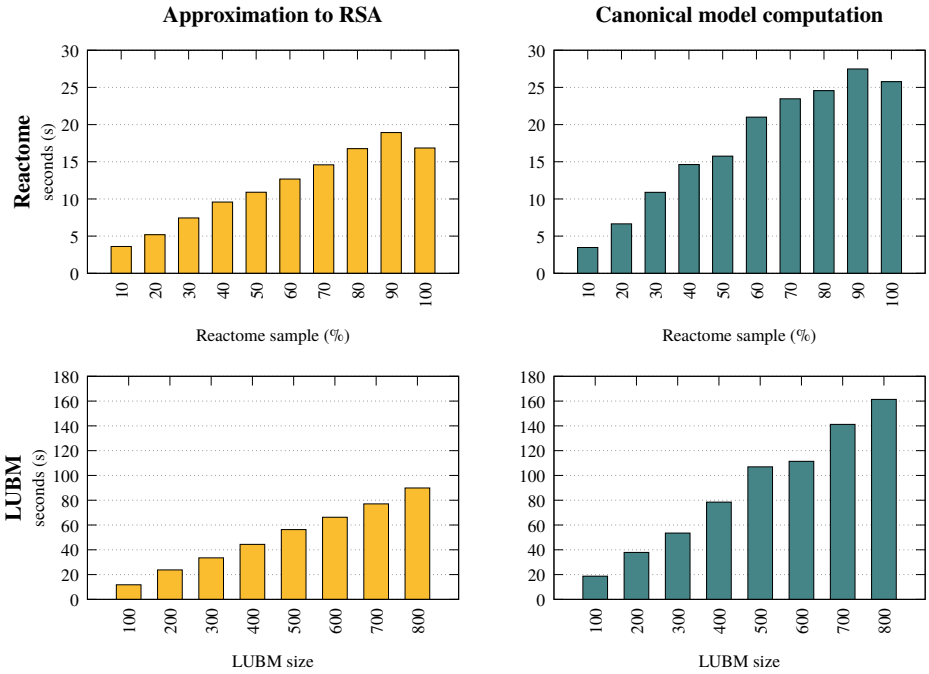


Figure 2. Scalability of approximation to RSA and canonical model computation

datasets come in sizes 100 through 800 universities (in steps of 100). Reactome dataset was sampled from 10% through 100% (in steps of 10%). We used the same set of queries used in [4], which provides 4 queries for LUBM and 3 queries for Reactome. All measurements provided below are averages of at least 3 runs.

Scalability of the system In Figure 2 we show the scalability of our algorithm for the approximation to RSA and the computation of the canonical model for the approximated ontology. The two steps are query independent and present a linear growth w.r.t. the dataset size, both in LUBM and Reactome.

The filtering process is instead less dependent on the size of the data and more dependent on its composition and distribution. As such, a bigger dataset does not necessarily correspond to a greater amount of filtering, as shown in Figure 3, where we reported the execution time for query 1 and 2 in Reactome. This figure also shows how the filtering depends on the data distribution; both queries take longer on size "50" than on other datasets (even larger ones) due to its specific content.

In general, we noticed that the time spent by the system on the filtering step is considerably smaller than the time spent on the canonical model computation (as described below, and shown in Figure 5).

This unpredictability of the filtering step can "backfire" when a huge amount of filtering is involved. In Figure 4 we show the filtering time for query 2 in

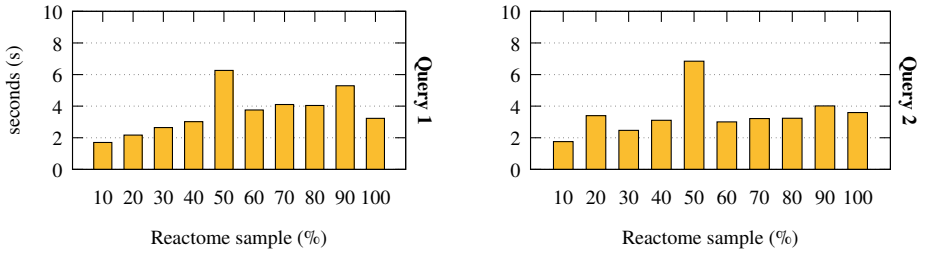


Figure 3. Filtering times in Reactome

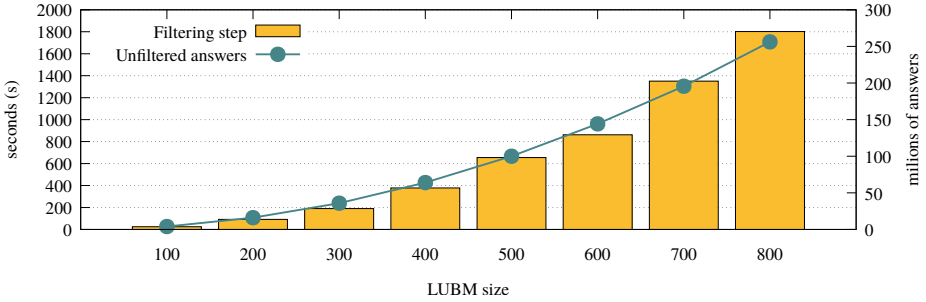


Figure 4. Filtering step with high degree of filtration in Query 2 in LUBM

LUBM along with the amount of *unfiltered answers* that the filtering program needs to process. Of these, less than 1‰ is found to be a certain answers. Figure 4 confirms the previous claims that the filtering step grows proportionally to the amount of filtering that is needed for a particular query. Finally, this figure shows how our system is able to handle a gigantic filtering step, processing hundreds of millions of facts in a reasonable amount of time.

Canonical model computation vs. filtering Finally, Figure 5 shows how execution time is distributed among the two main tasks of the combined approach. Filtering takes consistently less that 20% of the total execution time, when considering bigger datasets. As mentioned before, we can limit the impact of the canonical model computation by factoring out the task and computing it "offline" whenever we found ourselves in a scenario in which we need to perform query answering over a *fixed* ontology.

6 Future development

We presented RSAComb, an alternative and improved implementation of the combined approach for CQ answering in RSA. The software presents a few differences from the original description presented in [4], along with several enhancements. Our system relies on RDFox as a Datalog reasoner to offload part

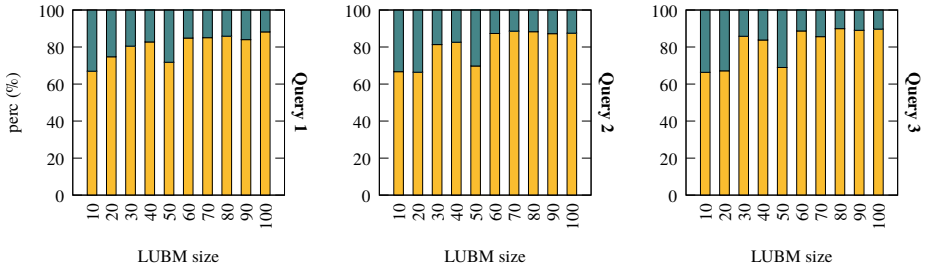


Figure 5. Percent time distribution of canonical model computation (at the bottom, in yellow) and answer filtering (at the top, in blue) in Reactome

of the computation. A customisable approximation step allows handling unrestricted OWL 2 ontologies.

RSAComb is currently used as part of an effort to improve the performance of the OWL 2 reasoner PAGOdA [19]. As such, future development of RSAComb will be mainly influenced by requirements originating from the process of integration of the system in PAGOdA.

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. Journal of Automated Reasoning **39**(3), 385–429 (2007). https://doi.org/10.1007/s10817-007-9078-x
3. Carral, D., Feier, C., Grau, B.C., 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. Feier, C., Carral, D., Stefanoni, G., Grau, B.C., 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)
5. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics **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`
6. 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`

7. 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)
8. Lutz, C.: The complexity of conjunctive query answering in expressive description logics. In: Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings. Lecture Notes in Computer Science, vol. 5195, pp. 179–193. Springer (2008). https://doi.org/10.1007/978-3-540-71070-7_16
9. 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
10. Motik, B., Nenov, Y., Piro, R., Horrocks, I.: Combining rewriting and incremental materialisation maintenance for datalog programs with equality. In: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015. pp. 3127–3133. AAAI Press (2015)
11. 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)
12. 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)
13. 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)
14. 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)
15. Ortiz, M., Rudolph, S., Simkus, M.: Query answering in the horn fragments of the description logics SHOIQ and SROIQ. In: IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011. pp. 1039–1044. IJCAI/AAAI (2011). https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-178
16. 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
17. Stefanoni, G., Motik, B.: Answering conjunctive queries over el knowledge bases with transitive and reflexive roles. CoRR **abs/1411.2516** (2014)
18. 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

19. Zhou, Y., Cuenca Grau, B., Nenov, Y., Kaminski, M., Horrocks, I.: Pagoda: Pay-as-you-go ontology query answering using a datalog reasoner. Journal of Artificial Intelligence Research **54**, 309–367 (2015)