PAGOdA: Pay-as-you-go ABox Reasoning

Yujiao Zhou, Yavor Nenov, Bernardo Cuenca Grau, and Ian Horrocks

Department of Computer Science, University of Oxford, UK

1 Introduction

Ontologies are increasingly used to provide structure and enhance access to large datasets. In such applications the ontology can be seen as a TBox, and the dataset as an ABox, with the key reasoning problem being conjunctive query (CQ) answering. Unfortunately, for OWL 2 this problem is known to be of high worst case complexity, even when complexity is measured with respect to the size of the data, and in realistic settings datasets may be very large.

One way to address this issue is to restrict the ontology to a fragment with better computational properties, and this is the motivation behind the OWL 2 profiles. Another approach is to optimise reasoning for arbitrary OWL 2 ontologies. This latter approach has proved very successful for TBox reasoning, with systems such as Konclude, HermiT, Pellet and Racer being widely used to reason over large-scale ontologies. Up to now, however, reasoning with large ABoxes has, in practice, largely been restricted to the OWL 2 profiles.

In this paper we describe PAGOdA: a highly optimised reasoning system that supports CQ answering with respect to an arbitrary OWL 2 ontology and an RDF dataset (roughly equivalent to a $\mathcal{SROIQ}$ TBox and ABox). It uses a novel approach to query answering that combines a datalog (or OWL 2 RL) reasoner (currently RDFox [11]) with a fully-fledged OWL 2 reasoner (currently HermiT [5]) to provide scalable performance while still guaranteeing sound and complete answers.[1] PAGOdA delegates the bulk of the computational workload to the datalog reasoner, with the extent to which the fully-fledged reasoner is needed depending on interactions between the ontology, the dataset and the query. This approach is 'pay-as-you-go' in the sense that query answering is fully delegated to the datalog reasoner whenever the input ontology is expressed in any of the OWL 2 profies; furthermore, even when using an out-of-profile ontology, queries can often be fully answered using only the datalog reasoner; and even when the fully-fledged reasoner is required, PAGOdA employs a range of optimisations, including relevant subset extraction, summarisation and dependency analysis, to reduce the number and size of the relevant reasoning problems.

This approach has proved to be very effective in practice: in our tests of more than 4,000 queries over 8 ontologies, none of which is contained within any of the OWL profiles, more than 99% of queries were fully answered without

[1] In practice we are limited by the capabilities of OWL 2 reasoners, which typically restrict the structure of the ontology and/or query in order to ensure decidability (which is open for CQ answering over unrestricted OWL 2 ontologies).

resorting to the fully-fledged reasoner. Moreover, even when the fully-fledged reasoner was used, the above mentioned optimisations were highly effective: the size of the dataset was typically reduced by an order magnitude, and often by several orders of magnitude, and it seldom required more than a single test to resolve the status of all potential answer tuples. Taken together, our experiments demonstrate that PAGOdA can provide an efficient conjunctive query answering service in real-world scenarios requiring both expressive ontologies and datasets containing hundreds of millions of facts.

The basic approach implemented in PAGOdA has been described in [13–15], and full details about the algorithms currently implemented can be found in a an accompanying technical report.[2] Here, we provide an overview of the system and summarise the results of an extensive evaluation.

2 The PAGOdA System

PAGOdA is written in Java and it is available under an academic license.[3] As well as RDFox and HermiT, PAGOdA also exploits the combined approach for $\mathcal{ELHO}_{\bot}^{r}$ implemented in KARMA.[4]

PAGOdA accepts as input arbitrary OWL 2 DL ontologies, datasets in turtle format and CQs in SPARQL. Queries can be interpreted under ground or certain answer semantics. In the former case, PAGOdA is sound and complete. In the latter case, however, PAGOdA is limited by the capabilities of HermiT, which can only check entailment of ground or DL concept queries; hence, PAGOdA can guarantee completeness only if the lower and upper bounds match, or if the query can be transformed into a DL concept query via rolling-up.[5] Otherwise, PAGOdA returns a sound (but possibly incomplete) set of answers, along with a bound on the incompleteness of the computed answer set.

The architecture of PAGOdA is depicted in Figure 1. We could, in principle, use any materialisation-based datalog reasoner that supports CQ evaluation and the incremental addition of facts, and any fully-fledged OWL 2 DL reasoner that supports fact entailment.

PAGOdA uses four instances of RDFox (one in each of the lower and upper bound and subset extractor components) and two instances of HermiT (one in each of the summary filter and dependency graph components).

The process of fully answering a query can be divided into several steps. Here, we distinguish between query independent steps and query dependent ones. As we can see in Figure 1, the 'loading ontology' and 'materialisation' steps are query independent. Therefore, both of them are counted as *pre-processing* steps. 'Computing query bounds', 'extracting subset' and 'full reasoning' are query dependent, and are called *query processing* steps.

We next describe each component, following the process flow of PAGOdA.

[2] `http://www.cs.ox.ac.uk/isg/tools/PAGOdA/pagoda-tr.pdf`
[3] `http://www.cs.ox.ac.uk/isg/tools/PAGOdA/`
[4] `http://www.cs.ox.ac.uk/isg/tools/KARMA/`.
[5] PAGOdA implements an extension of the well-known rollig-up technique.

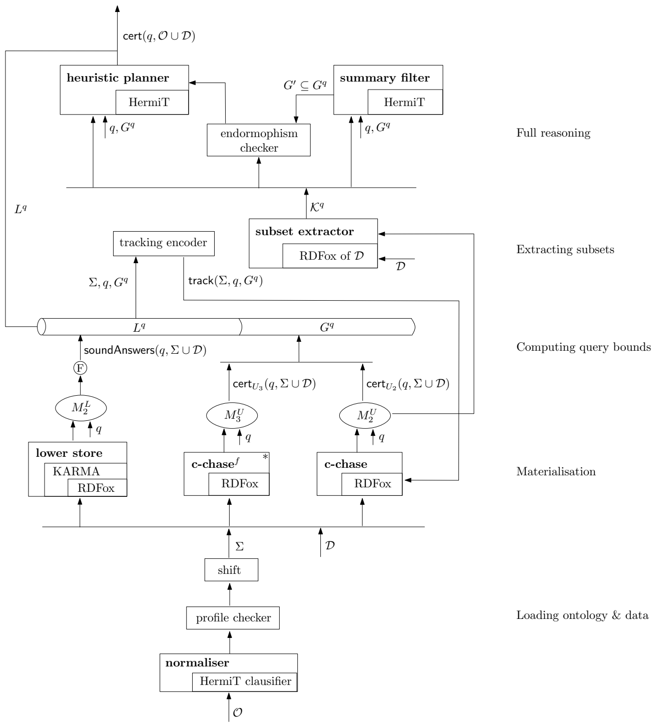


Fig. 1: The architecture of PAGOdA

Loading ontology and data. PAGOdA uses the OWL API to parse the input ontology $\mathcal{O}$. The dataset $\mathcal{D}$ is given separately in turtle format. The normaliser then transforms the ontology into a set of rules corresponding to the axioms in $\mathcal{O}$. PAGOdA's normaliser is an extension of HermiT's clausification component [5], which transforms axioms into so-called DL-clauses [12]. The dataset $\mathcal{D}$ is loaded directly into (the four instances of) RDFox.

After normalisation, the ontology is checked to determine if it is inside OWL 2 RL (resp. $\mathcal{ELHO}^r_\bot$); if so, then RDFox (resp. KARMA) is already sound and complete, and PAGOdA simply processes $\mathcal{O}$, $\mathcal{D}$ and subsequent queries using the relevant component. Otherwise, PAGOdA uses a variant of *shifting*—a polynomial program transformation commonly used in Answer Set Programming [4]—to enrich the deterministic part of the ontology with some additional information from disjunctive rules, resulting in a rule set Σ.

Materialisation. There are three components involved in this step, namely lower bound, c-chase and c-chase^f. Each of these takes as input Σ and $\mathcal{D}$, and each computes a materialisation (shown in Figure 1 as ellipses). The lower bound component first uses RDFox to compute a materialisation of $\mathcal{D}$ using the datalog subset of Σ; it then uses the materialised dataset as input to KARMA, which computes the materialisation M_2^L using the $\mathcal{ELHO}^r_\bot$ subset of Σ. The c-chase and c-chase^f components compute the M_2^U and M_3^U upper bound materialisations using chase-like procedures [1]. The former (M_2^U) is computed by over-approximating Σ into datalog; this involves, roughly speaking, transforming disjunctions into conjunctions, and replacing existentially quantified variables with fresh constants [15]. However, PAGOdA optimises the treatment of "Skolemised" existential rules by not applying them if the existential restriction is already satisfied in the data. In M_3^U, PAGOdA further optimises the treatment of disjunctions by selecting a single disjunct in the head of disjunctive rules, using a heuristic choice function that tries to select disjuncts that will not (eventually) lead to a contradiction.

If $\perp$ is derived while computing M_2^L, then the input ontology and dataset is unsatisfiable, and PAGOdA simply reports this and terminates. If $\perp$ is derived while computing M_3^U, then the computation is aborted and M_3^U is no longer used. If $\perp$ is derived while computing M_2^U, then PAGOdA checks the satisfiability of $\Sigma \cup \mathcal{D}$ (using the optimised query answering procedure described below). If $\Sigma \cup \mathcal{D}$ is unsatisfiable, then PAGOdA reports this and terminates; otherwise the input ontology and dataset is satisfiable, and PAGOdA is able to answer queries.

Computing query bounds. Given a query q, PAGOdA uses the M_2^L lower bound materialisation to compute the lower bound answer L^q, exploiting the filtration procedure in KARMA to eliminate spurious answer tuples (shown as a circle with an "F" in it in Figure 1). If $\perp$ was not derived when computing the M_3^U materialisation, then the upper bound answer U^q is the intersection of the query answers w.r.t. M_3^U and M_2^U; otherwise U^q is computed using only M_2^U.

Extracting subsets. If $L^q = U^q$, then PAGOdA simply returns L^q; otherwise it must determine the status of the tuples in the "gap" $G^q = U^q \setminus L^q$. To do this, PAGOdA extracts subsets of Σ and $\mathcal{D}$ that are sufficient to check the entailment of each such tuple. First, the tracking encoder component is used to compute a datalog program that tracks rule applications that led to the derivation of the tuples in G^q. This program is then added to the rules and data in the c-chase component, and RDFox is used to extend the c-chase materialisation accordingly. The freshly derived facts (over the tracking predicates introduced by the tracking encoder) are then passed to the subset extractor component, which identifies the relevant facts and rules in Σ and $\mathcal{D}$.

Full reasoning. PAGOdA uses HermiT to verify answers in G^q. As HermiT only accepts queries given either as facts or DL concepts, we have implemented the standard rolling-up technique to transform CQs into concepts [7]. In the summary filter component, PAGOdA uses summarisation techniques inspired by the SHER system to quickly identify spurious gap tuples [2, 3]. The remaining gap answers $G' \subseteq G^q$ are then passed to the endormorphism checker, which exploits a greedy algorithm to compute a (incomplete) dependency graph between answers in G'. An edge $\mathbf{a} \to \mathbf{b}$ in such graph between gap answers $\mathbf{a}$ and $\mathbf{b}$ encodes the following dependency: $\mathbf{b}$ is a spurious answer whenever $\mathbf{a}$ is, in which case it makes sense to check $\mathbf{a}$ using the fully-fledged reasoner before checking $\mathbf{b}$. This information is used by the heuristic planner to optimise the order in which the answers in G' are checked using HermiT. Finally, verified answers from G' are combined with the lower bound L^q.

	♯axioms	♯rules	♯∃-rules	♯∨-rules	♯facts
LUBM(n)	93	133	15	0	$n \times 10^5$
UOBM(n)	186	234	23	6	$2.6n \times 10^5$
FLY	14,447	18,013	8396	0	8×10^3
NPD	771	778	128	14	3.8×10^6
DBPedia$^+$	1,716	1,744	11	5	2.9×10^7
ChEMBL	2,593	2,960	426	73	2.9×10^8
Reactome	559	575	13	23	1.2×10^7
Uniprot	442	459	20	43	1.2×10^8

Table 1: Statistics for test datasets

3 Evaluation

We have evaluated our PAGOdA on a range of realistic and benchmark ontologies, datasets and queries. Experiments were conducted on a 32 core 2.60GHz Intel Xeon E5-2670 with 250GB of RAM, and running Fedora 20. All test ontologies, queries, and results are available online.[6]

3.1 Test Setting

Table 1 summarises our test data. Each column from left to right indicates the number of DL axioms, the number of rules after normalisation, the number of rules containing ∃, the number of rules containing ∨ in each ontology and the number of facts in each dataset.

LUBM and UOBM are widely-used reasoning benchmarks [6, 10]. To make the tests on LUBM more challenging, we extended the benchmark with 10 additional queries for which datalog lower-bound answers are not guaranteed to be complete (as is the case for the standard queries).

FLY is an ontology used in the Virtual Fly Brain tool.[7] Although the dataset is small, the ontology is rich in existentially quantified rules, which makes query answering challenging. We tested 6 CQs provided by the ontology developers.

NPD FactPages is an ontology describing petroleum activities in the Norwegian continental shelf. The ontology comes with a dataset containing 3.8 million facts. We tested all atomic queries over the signature of the ontology.

DBPedia contains information about Wikipedia entries. Although the dataset is rather large, the ontology axioms are simple and can be captured by OWL 2 RL. To provide a more challenging test, we have used LogMap [8] to extend DBPedia with a tourism ontology containing both existential and disjunctive rules. We again focused on atomic queries.

ChEMBL, Reactome, and Uniprot are ontologies that are available from the European Bioinformatics Institute (EBI) linked data platform.[8] They are

[6] `http://www.cs.ox.ac.uk/isg/tools/PAGOdA/2015/jair/`
[7] http://www.virtualflybrain.org/site/vfb_site/overview.htm
[8] http://www.ebi.ac.uk/rdf/platform

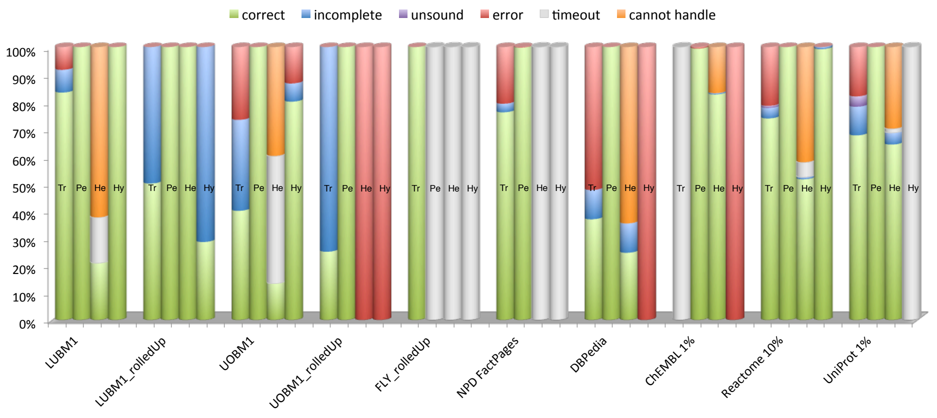


Fig. 2: Quality of the answers computed by each system. The four bars for each ontology represent Trowl, Pellet, HermiT and Hydrowl respectively.

rich in both existential and disjunctive rules, and the datasets are large. In order to test scalability, we computed subsets of the data using a data sampling algorithm based on random walks [9]. We tested example queries provided on the EBI website as well as all atomic queries over the relevant signatures.

3.2 Experiments and Results

Comparison with Other Systems We compared PAGOdA with HermiT (v.1.3.8), Pellet (v.2.3.1), TrOWL-BGP (v.1.2), and Hydrowl (v.0.2). Although TrOWL is incomplete for OWL 2, it has been included in the evaluation because, like PAGOdA, it exploits ontology approximation techniques.

In this test we used LUBM(1) and UOBM(1), 1% of the dataset for ChEMBL and UniProt, and 10% for Reactome; these are already hard for some systems, but can be processed by most. We rolled up all 6 queries into concepts wherever possible to get LUBM_rolledUp, UOBM_rolledUp and FLY_rolledUp. Since the answers to the FLY queries under SPARQL semantics are all empty, we only present results for FLY_rolledUp. We set timeouts of 20min for each individual query, and 5h for all the queries over a given ontology.

In Figure 2, each bar represents the performance of a particular reasoner w.r.t. a given ontology and set of test queries. We use green to indicate the percentage of queries for which the reasoner computed all the correct answers, where correctness was determined by majority voting, and blue (resp. purple) to indicate the percentage of queries for which the reasoner was incomplete (resp. unsound). Red, orange and grey indicate, respectively, the percentage of queries for which the reasoner reported an exception during execution, did not accept the input query, or exceeded the timeout. PAGOdA is not represented in the figure as it was able to correctly compute all answers for every query and test ontology within the given timeouts.

Figure 3 summarises the performance of each system relative to PAGOdA, but in this case we considered only those queries for which the relevant system

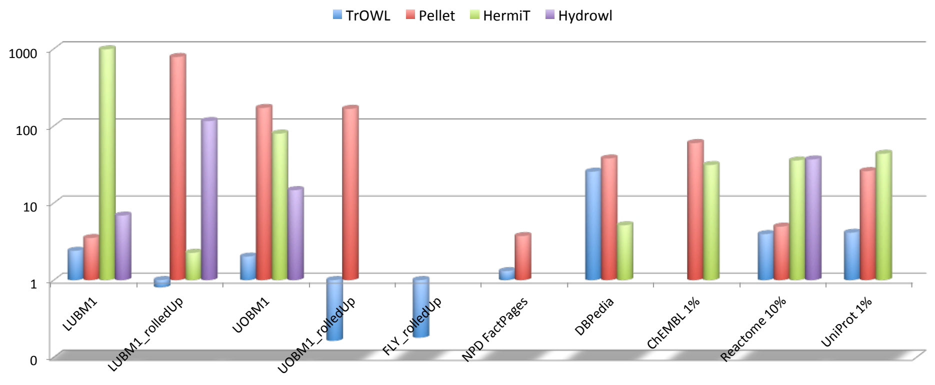


Fig. 3: Performance comparison with other systems.

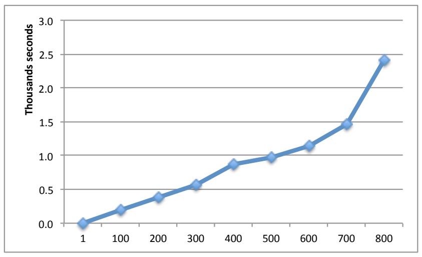
yields an answer (even if unsound and/or incomplete). This is not ideal, but we chose to consider all such queries (rather than only the queries for which the relevant system yields the correct answer) because *(i)* the resulting time measurement is obviously closer to the time that would be required to correctly answer all queries; and *(ii)* correctness is only relative as we do not have a gold standard. For each ontology and reasoner, the corresponding bar shows t_2/t_1 (on a logarithmic scale), where t_1 (resp. t_2) is the total time required by PAGOdA (resp. the compared system) to compute the answers to the queries under consideration; a missing bar indicates that the comparison system failed to answer any queries within the given timeout. Please note that two different bars for the same ontology are not comparable as they may refer to different sets of queries, so each bar needs to be considered in isolation.

TrOWL is faster than PAGOdA on LUBM_rolledUp, UOBM_rolledUp and FLY_rolledUp, but it is incomplete for 7 out of 14 LUBM queries and 3 out of 4 UOBM queries. For ChEMBL, TrOWL exceeds the timeout while performing the satisfiability check. For the remaining ontologies, PAGOdA is more efficient in spite of the fact that TrOWL is incomplete for some queries, and even unsound for several UniProt queries.

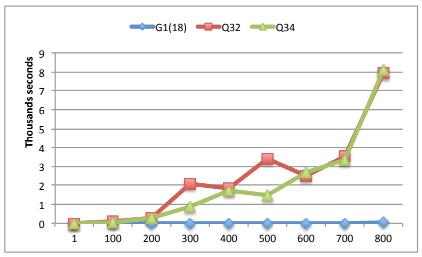
Pellet times out for the FLY ontology, but it succeeds in computing all answers in the remaining cases. We can observe, however, that in all cases Pellet is significantly slower than PAGOdA, sometimes by more than two orders of magnitude.

HermiT can only answer queries with one distinguished variable, so we could not evaluate binary queries. HermiT exceeds the timeout in many cases, and in the tests where it succeeds, it is significantly slower than PAGOdA.

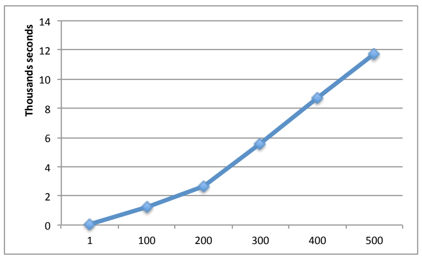
Hydrowl is based on a theoretically sound and complete algorithm, but it was found to be incomplete in some of our tests. It also exceeded the timeout for three of the ontologies, ran out of memory for another two of the ontologies, and reported an exception for ChEMBL 1%. In the remaining cases, it was significantly slower than PAGOdA.



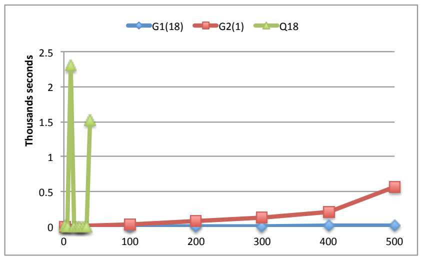
(a) LUBM pre-processing



(b) LUBM query processing



(c) UOBM pre-processing



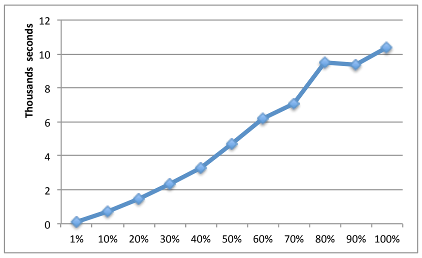
(d) UOBM query processing

Fig. 4: Scalability tests on benchmarks

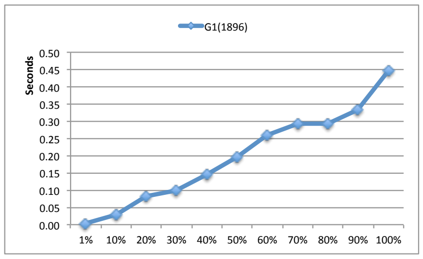
Scalability Tests We tested the scalability of PAGOdA on LUBM, UOBM and the ontologies from the EBI platform. For LUBM we used datasets of increasing size with a step of $n = 100$. For UOBM we also used increasingly large datasets with step $n = 100$ and we also considered a smaller step of $n = 5$ for hard queries. Finally, in the case of EBI's datasets, we computed subsets of the data of increasing sizes from 1% of the original dataset up to 100% in steps of 10%. In each case we used the test queries described in Section 3.1. For each test ontology we measured *pre-processing time* and *query processing time* as described in Section 2. We organise the test queries into three groups: **G1**: queries for which the lower and upper bounds coincide; **G2**: queries with a non-empty gap, but for which summarisation is able to filter out all remaining candidate answers; and **G3**: queries where the fully-fledged reasoner is called over an ontology subset on at least one of the test datasets. We set a timeout of 2.5h for each individual query and 5h for all queries.

We also tested Pellet (the only other system found to be sound and complete for our tests) on Reactome, the only case were Pellet managed to process at least two datasets.

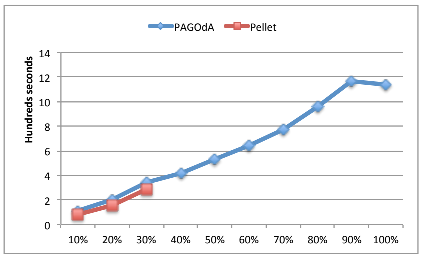
Our results are summarised in Figures 4 and 5. For each ontology, we plot time against the size of the input dataset, and for query processing we distinguish different groups of queries as discussed above. PAGOdA behaves relatively uniformly for queries in **G1** and **G2**, so we plot only the average time per query for these groups. In contrast, PAGOdA's behaviour for queries in **G3** is quite variable, so we plot the time for each individual query.



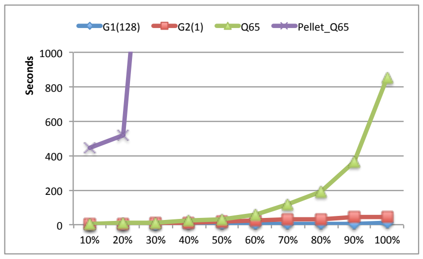
(a) ChEMBL pre-processing



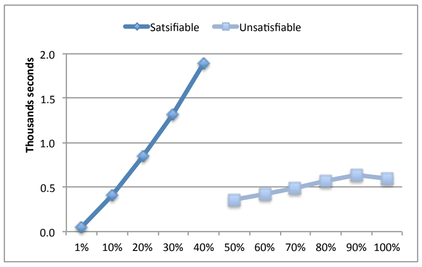
(b) ChEMBL query processing



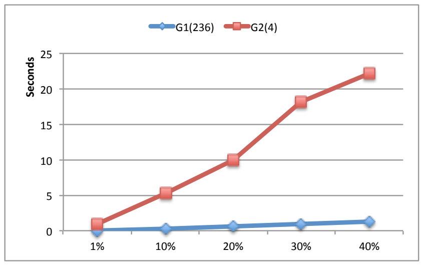
(c) Reactome pre-processing



(d) Reactome query processing



(e) Uniprot pre-processing



(f) Uniprot query processing

Fig. 5: Scalability tests on EBI linked data platform

LUBM(n) All LUBM queries belongs to either **G1** or **G3** with the latter group containing two queries. The average query processing time for queries in **G1** never exceeds 13s; for the two queries in **G3** (Q32 and Q34), this reaches 8,000s for LUBM(800), most of which is accounted for by HermiT.

UOBM(n) As with LUBM, most test queries were contained in **G1**, and their processing times never exceeded 8 seconds. We found one query in **G2**, and PAGOdA took 569s to answer this query for UOBM(500). UOBM's randomised data generation led to the highly variable behaviour of Q18: it was in **G3** for UOBM(1) UOBM(10) and UOBM(50), causing PAGOdA to time out in the last case; it was in **G2** for UOBM(40); and it was in **G1** in all other cases.

ChEMBL All test queries were contained in **G1**, and average processing time was less than 0.5s in all cases.

	LUBM	UOBM	FLY	NPD	DBPedia	ChEMBL	Reactome	UniProt
Total	35	20	6	478	1247	1896	130	240
$L_1 + U_1$	26	4	0	442	1240	1883	82	204
$L_2 + U_1$	33	4	5	442	1241	1883	82	204
$L_2 + U_2$	33	12	5	442	1241	1883	98	204
$L_2 + U_{2\|3}$	33	16	5	473	1246	1896	128	236

Table 2: ♯Queries answered by different bounds

Reactome Groups **G2** and **G3** each contained one query, with all the remaining queries belonging to **G1**. Query processing time for queries in **G1** never exceeded 10 seconds; for **G2** processing time appeared to grow linearly in the size of datasets, and average time never exceeded 10 seconds; the **G3** query (Q65) is much more challenging, but it could still be answered in less than 900 seconds, even for the largest dataset.

On Reactome, Pellet is able to process the samples of size $10\%, 20\%$ and 30%, with pre-processing times comparable to PAGOdA. Average query-processing times for queries in **G1** and **G2** are slightly higher than those of PAGOdA, but times for query Q65 were significantly higher. This was due to PAGOdA's subset extraction technique, which is able to keep the input to the fully-fledged reasoner small, even for the largest datasets.

Uniprot In contrast to the other cases, Uniprot as a whole is unsatisfiable; however, our sampling technique can produce a satisfiable subset up to 40%. For larger subsets, pre-processing times drop abruptly as unsatisfiability can be efficiently detected in the lower bound. Query processing times were only considered for satisfiable samples. There were no queries in **G3**, and only four in **G2**, all of which were efficiently handled.

Effectiveness of Different Techniques We have evaluated the effectiveness of the various reasoning techniques implemented in PAGOdA by comparing the numbers of test queries that can be fully answered using the relevant technique.

Query bounds Table 2 illustrates the effectiveness of different combinations of upper and lower bounds in terms of the number of queries for which the bounds coincided for each test ontology and its smallest test datasets. In the table, we refer to the lower bound computed w.r.t. the datalog subset of the input knowledge base as L_1 and to the combined lower bound computed by PAGOdA as L_2. Similarly, we refer the naive upper bound computed using a datalog over-approximation of Σ as U_1; the upper bound computed w.r.t. M_2^U and M_3^U as U_2 and U_3; and the combined upper bound as $U_{2|3}$.

It can be seen that L_1 and U_1 suffice to answer most of the queries in many test ontologies. L_2 was very effective in the case of FLY, where the basic bounds did not match for any query, and also useful for LUBM, yielding matching bounds for 7 more queries. U_2, was especially effective for UOBM and Reactome, where many existentially quantified rules were already satisfied by the lower bound materialisation. Finally, the refined treatment of disjunctive rules in $U_{2|3}$ was instrumental in obtaining additional matching bounds for non-Horn ontologies.

	LUBM	UOBM	Fly	NPD	DBPedia	Reactome	Uniprot
Facts	0.5%	10.4%	7.3%	16.5%	9×10^{-5}%	5.2%	4×10^{-4}%
Rules	3.7%	10.9%	0.9%	18.4%	2.4%	5.3%	1.1%

Table 3: Size of the largest subsets given as percentage over input rules and facts.

	LUBM		UOBM				FLY	DBPedia	NPD	Reactome		UniProt
$L_2 + U_{2\|3}$	26	14	264	112	1470	264	344	10	326	18	52	168
+ Sum	26	14	264	0	1444	264	344	0	0	0	52	0
+ Dep	1	1	1	0	1	1	7	0	0	0	37	0

Table 4: The number of hard calls to HermiT to fully answer each query

Subset extraction Table 3 shows, for each dataset, the maximum percentage of facts and rules that are included in the relevant subset over all test queries with non-matching bounds. We can observe that subset extraction is effective in all cases in terms of both facts and rules.

Summarisation and Dependencies The effectiveness of these techniques was measured by the number of 'hard' calls to HermiT that were required to fully answer each query, where a call is considered hard if the knowledge base passed to HermiT is not a summary. The first row of Table 4 shows the number of gap answers for each query where the L_2 and $U_{2|3}$ bounds don't match. Without optimisation, we would have to call HermiT this number of times to fully answer each query. Row 2 (resp. row 3) shows the number of hard calls to HermiT after applying summarisation (resp. summarisation plus dependency analysis).

4 Discussion

The reasoning techniques we have proposed here are very general and are applicable to a wide range of knowledge representation languages. Our main goal in practice, however, has been to realise our approach in a highly scalable and robust query answering system for OWL 2 DL ontologies, which we have called PAGOdA. Our extensive evaluation has not only confirmed the feasibility of our approach in practice, but also that our system PAGOdA significantly ourperforms state-of-the art reasoning systems in terms of both robustness and scalability. In particular, our experiments using the ontologies in the EBI linked data platform have shown that PAGOdA is capable of fully answering queries over highly complex and expressive ontologies and realistic datasets containing hundreds of millions of facts.

Acknowledgements. This work has been supported by the Royal Society under a Royal Society Research Fellowship, by the EPSRC projects MaSI^3, Score! and DBOnto, and by the EU FP7 project Optique.

References

1. Calì, A., Gottlob, G., Kifer, M.: Taming the infinite chase: Query answering under expressive relational constraints. Journal of Artificial Intelligence Research 48, 115–174 (2013), `http://dx.doi.org/10.1613/jair.3873`
2. Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Schonberg, E., Srinivas, K., Ma, L.: Scalable semantic retrieval through summarization and refinement. In: AAAI 2007, 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`
3. Dolby, J., Fokoue, A., Kalyanpur, A., Schonberg, E., Srinivas, K.: Scalable highly expressive reasoner (SHER). Journal of Web Semantics 7(4), 357–361 (2009)
4. Eiter, T., Fink, M., Tompits, H., Woltran, S.: Simplifying logic programs under uniform and strong equivalence. In: LPNMR 2004, Proceedings of Logic Programming and Nonmonotonic Reasoning - 7th International Conference, Fort Lauderdale, FL, USA, January 6-8, 2004, Proceedings. pp. 87–99 (2004)
5. Glimm, B., Horrocks, I., Motik, B., Stoilos, G., Wang, Z.: HermiT: An OWL 2 reasoner. Journal of Automated Reasoning 53(3), 245–269 (2014)
6. Guo, Y., Pan, Z., Heflin, J.: LUBM: A benchmark for OWL knowledge base systems. Journal of Web Semantics 3(2-3), 158–182 (2005)
7. Horrocks, I., Tessaris, S.: A conjunctive query language for description logic aboxes. In: Kautz, H.A., Porter, B.W. (eds.) AAAI/IAAI 2000, 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`
8. Jiménez-Ruiz, E., Cuenca Grau, B.: LogMap: Logic-based and scalable ontology matching. In: Aroyo, L., Welty, C., Alani, H., Taylor, J., Bernstein, A., Kagal, L., Noy, N.F., Blomqvist, E. (eds.) ISWC 2011, The Semantic Web - 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I. Lecture Notes in Computer Science, vol. 7031, pp. 273–288. Springer (2011), `http://dx.doi.org/10.1007/978-3-642-25073-6_18`
9. Leskovec, J., Faloutsos, C.: Sampling from large graphs. In: KDD 2006, Proceedings of the Twelfth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Philadelphia, PA, USA, August 20-23, 2006. pp. 631–636 (2006)
10. Ma, L., Yang, Y., Qiu, Z., Xie, G.T., Pan, Y., Liu, S.: Towards a complete OWL ontology benchmark. In: Sure, Y., Domingue, J. (eds.) ESWC 2006, The Semantic Web: Research and Applications, 3rd European Semantic Web Conference, Budva, Montenegro, June 11-14, 2006, Proceedings. Lecture Notes in Computer Science, vol. 4011, pp. 125–139. Springer (2006), `http://dx.doi.org/10.1007/11762256_12`
11. Motik, B., Nenov, Y., Piro, R., Horrocks, I., Olteanu, D.: Parallel materialisation of datalog programs in centralised, main-memory RDF systems. In: Brodley, C.E., Stone, P. (eds.) AAAI 2014, Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, July 27 -31, 2014, Québec City, Québec, Canada. pp. 129–137. AAAI Press (2014), `http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8505`
12. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. Journal of Artificial Intelligence Research 36, 165–228 (2009), `http://dx.doi.org/10.1613/jair.2811`

13. Zhou, Y., Nenov, Y., Cuenca Grau, B., Horrocks, I.: Complete query answering over horn ontologies using a triple store. In: The Semantic Web - ISWC 2013 - 12th International Semantic Web Conference, Sydney, NSW, Australia, October 21-25, 2013, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8218, pp. 720–736. Springer (2013)
14. Zhou, Y., Nenov, Y., Cuenca Grau, B., Horrocks, I.: Pay-as-you-go ontology query answering using a datalog reasoner. In: Informal Proceedings of the 27th International Workshop on Description Logics, Vienna, Austria, July 17-20, 2014. CEUR Workshop Proceedings, vol. 1193, pp. 352–364. CEUR-WS.org (2014)
15. Zhou, Y., Nenov, Y., Cuenca Grau, B., Horrocks, I.: Pay-as-you-go OWL query answering using a triple store. In: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence (2014), http://www.aaai.org/ocs/index.php/AAAI/AAAI14/paper/view/8232