Towards a Systematic Benchmarking of Ontology-Based Query Rewriting Systems

Jose Mora and Oscar Corcho

Ontology Engineering Group, Departamento de Inteligencia Artificial,
Facultad de Informática,Universidad Politécnica de Madrid, Spain
{jmora,ocorcho}@fi.upm.es

Abstract. Query rewriting is one of the fundamental steps in ontology-based data access (OBDA) approaches. It takes as inputs an ontology and a query written according to that ontology, and produces as an output a set of queries that should be evaluated to account for the inferences that should be considered for that query and ontology. Different query rewriting systems give support to different ontology languages with varying expressiveness, and the rewritten queries obtained as an output do also vary in expressiveness. This heterogeneity has traditionally made it difficult to compare different approaches, and the area lacks in general commonly agreed benchmarks that could be used not only for such comparisons but also for improving OBDA support. In this paper we compile data, dimensions and measurements that have been used to evaluate some of the most recent systems, we analyse and characterise these assets, and provide a unified set of them that could be used as a starting point towards a more systematic benchmarking process for such systems. Finally, we apply this initial benchmark with some of the most relevant OBDA approaches in the state of the art.

1 Introduction

Ontology Based Data Access (OBDA) consists on superimposing a conceptual layer as a view to an underlying information system, which abstracts away from how that information is maintained in the data layer and provides inference capabilities [1]. The process of query answering in OBDA systems consists in using ontologies to transform ontology-based queries (e.g., written in SPARQL or in ad-hoc query languages) into queries that the underlying data sources are able to process (e.g., written in SQL, Datalog, etc.). This allows obtaining the answers for the original queries as if they had been posed to an ontology that contains instances, where such instances are obtained from the data available in those data sources.

OBDA has been mainly applied to the combination of description logic TBoxes with relational databases, which are the main type of data sources. However, it is not uncommon to find other works focused on providing OBDA support for other

types of data sources, such as data streams [2], spreadsheets [3], REST APIs, etc. In all these cases, mappings are commonly used to specify the relationships between the ontologies and the schema used by the underlying data sources. These mappings are normally called database-to-ontology mappings, in general, or RDB2RDF mappings, when the resulting instances are transformed (in a materialized or virtual manner) to RDF. In the latter case, R2RML [4] is a W3C recommendation that allows defining such type of mappings.

OBDA query answering approaches normally rely on query rewriting techniques [5]. In that case several stages are commonly considered in the whole OBDA query answering process:

- **Query rewriting**. In this stage, ontology-based queries are rewritten into queries that consider the inferences that can be done with the ontology.
- **Query translation**. In this stage the previous queries are transformed into the query language (or API) and schema of the underlying data sources, so that they can be evaluated by their corresponding query evaluation systems. This stage makes use of the aforementioned mappings.
- **Query execution**. The generated queries or API calls are evaluated or executed, obtaining results according to the underlying data source schema.
- **Result translation**. The results obtained from the evaluation are translated into the original ontology-based schema, so that they can be interpreted by the original issuer of the ontology-based queries.

In this paper we focus on the first stage, that is, on query rewriting. The motivation for our work stems from the fact that query rewriting systems are largely heterogeneous, in terms of the ontology languages that they give support to and the type of output that they produce as a result of the query rewriting process. Besides, there is also a large heterogeneity in the test cases that have been proposed to evaluate these approaches, with different real-world and synthetic ontologies and queries. This heterogeneity, in approaches and test cases, makes it difficult to compare how each of them behaves with respect to the others, and consequently to take decisions on which approach to use for each type of problem to be addressed. For this reason, we propose an integrated set of test cases that considers those used in previous approaches, and apply them to a set of OBDA systems that are being actively maintained nowadays, so that this benchmark can be used (and extended if needed) in the future to help in the continuous improvement of such systems, and hence of the whole OBDA research and development area.

This paper is structured as follows. In section 2 we describe the logics previously mentioned, the different systems in the state of the art that use these logics, and how these systems have been evaluated. In section 3 we analyse (*a priori*) the characteristics of the previously mentioned systems and their outputs, as well as how they relate to each other. In section 4 we present the results obtained from the evaluation of the different systems in the state of the art and analyse them (*a posteriori*). In section 5 we consider the limitations in the current state of the art for evaluation of these systems and the construction of a standard benchmark and hint at the future lines and difficulties to overcome.

2 Background

In this section, we describe briefly the most relevant logics used in recent OBDA systems and review the most relevant implementations that use these logics. We consider the properties of these implementations with respect to the type of input that they accept, the type of output that they generate and the optimizations that they implement in their query rewriting process. We have decided to use a chronological order for the presentation.

2.1 Logics Used in OBDA Approaches

In an OBDA context, query rewriting is a process that uses at least an ontology $\mathcal{O}$ to transform a query q into a rewritten query $q_{\mathcal{O}}$. In principle, the ontology $\mathcal{O}$ may be implemented in any ontology language $\mathcal{L}$. However, there is a tradeoff between the expressiveness of such ontology language $\mathcal{L}$ and the computational cost of processing ontologies in that language to obtain the rewritten queries. In this context, several languages have been proposed as interesting compromise solutions between expressiveness and tractability, among which we can cite: the *DL-Lite* family [6], which includes $DL\text{-}Lite_{core}$, $DL\text{-}Lite_{\mathcal{F}}$ and $DL\text{-}Lite_{\mathcal{R}}$; the QL profile of $OWL2$ [7]; and some families in Datalog± [8]. Along with these logics, we can also find $\mathcal{ELHIO}^{\neg}$ [9], where rewriting also remains tractable (PTIME-complete).

A common property for all these logics, except for $\mathcal{ELHIO}^{\neg}$, is that query rewriting with them is first-order language reducible (also known as first-order rewritable). Intuitively, query rewriting over some ontological language $\mathcal{L}$ is FOL-reducible if any union of conjunctive queries q expressed over a TBox $\mathcal{T}$ in the language $\mathcal{L}$ can be rewritten into a FOL query $q_{\mathcal{T}}$ such that the answers for $q_{\mathcal{T}}$ over the data source are the same that we would expect from q over $\mathcal{T}$ and the data source. In short, FOL-reducibility means that rewritten queries are first-order queries, what allows converting them to languages like SQL without using advanced features like recursion. It has been shown that the combined complexity of satisfiability on these logics is polynomial, while data complexity is AC^0 [10,11].

Now we will describe in more detail these logics:

- The *DL-Lite* family diverged into $DL\text{-}Lite_{\mathcal{R}}$ and $DL\text{-}Lite_{\mathcal{F}}$, both extending $DL\text{-}Lite_{core}$, where concept inclussions are restricted to $A \sqsubseteq B$ and $A \sqsubseteq \neg B$. $DL\text{-}Lite_{\mathcal{R}}$ includes ISA and disjointness assertions between roles and $DL\text{-}Lite_{\mathcal{F}}$ includes functionality restrictions on roles. These logics are first-order reducible with a tractable complexity [6].
- The $OWL2$ QL profile was inspired by the *DL-Lite* family and designed to keep the complexity of rewriting low, considering first-order rewritability. As a summary of a more extensive comparison [10], the main differences with *DL-Lite* are the status of the unique name assumption (UNA) which is adopted in DL-Lite but not in OWL, which uses instead explicit relations `sameAs` and `differentFrom`. OWL2 also lacks constructs that would conflict

with UNA and cause a greater complexity, like number restrictions, functionality constraints and keys. Among the constructs in OWL 2 not supported in *DL-Lite* we can remark nominals, concepts of the form $\{a\}$.

- The $\mathcal{ELHIO}^{\neg}$ logic [9] is more expressive; it extends the expressiveness of *DL-Lite*$_{\mathcal{R}}$ by including basic concepts of the form $\{a\}$, $\top$, and $B_1 \sqcap B_2$, as well as axioms of the form $\exists R.B \sqsubseteq C$. This logic is the only one in this section that does not present the first-order rewritability property, this means that depending on the query and the expressiveness in the ontology, the generated Datalog may contain recursive predicates, thus some queries cannot be unfolded into a union of conjunctive queries (UCQ) and must be rewritten to recursive Datalog when considering $\mathcal{ELHIO}^{\neg}$ ontologies. In spite that, the computational complexity of the rewriting process remains tractable (PTIME-complete).
- Some families in Datalog± preserve the property of first-order rewritability to SQL equivalent languages while offering a greater expressiveness for rewritings to SQL or non-recursive Datalog, mainly because of the fact that Datalog± predicates are n-ary. Some of the Datalog paradigms that ensure decidability are chase termination, guardedness or stickiness, extended to weak-stickiness by Calì [12]. These paradigms limit the loops that can be present in some Datalog to ensure decidability of the unfolding and thus first-order rewritability.
- Finally, Horn-$\mathcal{SHIQ}$ includes the role hierarchies and inverse roles as $\mathcal{ELHIO}$. It does also include universal restrictions and transitive roles ($\mathcal{S}$) axioms of the form $A \sqsubseteq \forall R.B$ and $trans(R)$. This logic does also include qualified cardinality restrictions ($\mathcal{Q}$) axioms of the form $A \sqsubseteq \leq 1R.B$. The Horn prefix refers to the Horn fragment of $\mathcal{SHIQ}$, this means that the axioms in this fragment can be converted to Horn clauses.

2.2 OBDA Query Rewriting Systems

These logics have been used in several approaches, starting with the perfect reformulation [6], which is implemented in **Quonto** and by Pérez-Urbina [9]. This approach accepts ontologies written in the *DL-Lite* family (*DL-Lite*$_{core}$, *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$) and generates a UCQ as a result of the rewriting process. This approach was the first of a series and would inspire many others, usually generating UCQs and optimizing the query rewriting process, for instance by applying optimizations based on query decomposition or identification of connected components.

The **RQR** algorithm in the REQUIEM system [9] accepts $\mathcal{ELHIO}^{\neg}$ ontologies and generates a rewriting using resolution with free selection (RFS) [13]. Both RQR and RFS form the basis for the algorithm presented here. RFS is proven to be correct and complete on Horn clauses [14] however due to space limitations RFS will not be explained, we refer the reader to previous citations. RQR reduces the number of useless factorizations in RFS, queries generated and processing time through several optimisations, the main one being the introduction of Skolem functions when an existential quantification occurs in the head of

a clause, which was handled in previous approaches as a nameless variable. The output generated by this approach is again a UCQ. Resolution in REQUIEM is splitted in two steps, the first one is saturation, which generates a (possibly recursive) Datalog program, and the second one is unfolding, which unfolds this Datalog program to generate a UCQ. REQUIEM may produce a Datalog program for the output by simply skipping the latter stage. If the ontology includes recursion the UCQ cannot be complete, in this case the unfolding generates a (recursive) linear Datalog program (at most one intensional predicate per clause). The number of different head predicates in this Datalog program is reduced in the unfolding stage in this case to reduce the amount of information loss in the case that clauses with head predicates different from the query predicate are dropped by the system that receives the output.

The previous approaches generate a large number of queries in the UCQ, as the generation of this UCQ from Datalog presents a combinatorial blowup that depends on the length of the query. **Presto** [15] addresses this problem on $DL\text{-}Lite_{\mathcal{R}}$, this does not include expressions of the form $\exists R.B$, which eases the search for *most-general subsumees* (MGS). MGS are used to remove existential join variables, to then remove unbound variables and redundant atoms. This way the query is recursively factorized and splitted, depending on the existential joins and the connectivity in the query. Presto obtains results that are several orders of magnitude faster in the query rewriting process than previous approaches. Depending on the ontology and the query, the resulting non-recursive Datalog is also normally briefer in the number of clauses, hiding the combinatorial explosion that would result from unfolding. As a result of the factorization, several parts of the query are rewritten into (potentially equivalent) subqueries.

Stamou [16] takes a different approach in the handling of Skolem functions. This approach has evolved into **Rapid** [17], which handles an expressiveness that falls between REQUIEM and Presto: expressions of the form $\exists R.B$ are allowed in REQUIEM but not in Presto; and in the case of Rapid they are allowed in the right hand side but not in the left hand side of subsumption axioms. Rapid applies two rules alternatively, *query shrinking* and *query unfolding*. *Query shrinking* removes a bound variable by unifying it with a functional term. Skolem functions are internally handled in this resolution rule, so they do not appear after applying it. *Query unfolding* replaces a set of atoms with its unfoldings, preserving the terms in the atoms (no functional terms are used). This strategy generates less subsumed (redundant) queries and it is possible to restrict the search for subsumed queries among subsets of the queries generated. The output is equivalent to REQUIEM as far as the ontology used is contained in the expressiveness that Rapid can handle.

A similar approach using two different resolution steps (factorization and rewriting) in a stratified strategy is the one implemented in **Nyaya** [18]. During the factorization step the query is compacted with unifications that preserve the query semantics, and in the rewriting step the query is unfolded. An optional step removes redundant atoms in the queries. In Nyaya, the expressiveness is greater than in previous cases by allowing the use of n-ary predicates. However

there is no statement about which additional ontological axioms could be covered with this. This expressiveness is limited for efficiency, in this case the body of the clauses is restricted to those that have only one atom. With this it is possible to identify atoms in the body of a query that are implied by some other atom in the body, what means that they can be eliminated, reducing the size of the query, the UCQ and the required processing. This approach is specially tailored at reducing the size of the UCQ that are generated in the process. Depending on the query, this optimisation may provide much smaller queries, in size, width or length, which are respectively, the number of queries in the UCQ, the number of joins to be performed and the number of atoms in the perfect rewriting as explained in the evaluation done for this approach.

Another approach that should be mentioned is the one taken by **Venetis** [19], based on perfect reformulation. This approach is based on the fact that users normally pose a succession of queries, refining an initial conjunctive query by adding or removing atoms. In these cases it is possible to use partial results from previous rewritings in the new rewritings.

Prexto [20] modifies Presto by considering extensional constraints, concept and role disjointness assertions and role functionality assertions, so as to reduce the size of the rewritten UCQ. Disjointness and role functionality assertions are considered when construcing the Datalog program along with subsumption. And in the unfolding stage these assertions are considered again, along with the extensional constraints, to reduce the views by removing the parts considered as unnecessary according to these criteria. These considerations allow reducing the combinatorial explosion usual in the unfolding of these Datalog programs and the size of the rewritten UCQ.

Clipper [21] aims at more expressive logics, more precisely Horn-$\mathcal{SHIQ}$. This approach can rewrite the ontology including TBox and ABox or only the TBox. The ontology is in this case preprocessed and saturated independently of the query, and the query is rewritten into a UCQ with additional Datalog rules that "complete" the ABox, which is comparable to a Datalog program. Despite the high expressiveness, the time to obtain the rewritings and the size of the rewritings are kept low. This is achieved by aiming at this UCQ with rules output and using a Datalog system that can handle it, such as DLV [22] or Clingo [23], instead of obtaining a UCQ as other systems do.

Finally, **kyrie** modifies REQUIEM by adding some engineering optimisations to speed up the process of query rewriting with the side effect of a possible reduction in the Datalog program generated when the output is set to Datalog. The optimisations performed in this case consist on a preprocessing step that materializes some of the conclusions that would be derived later in the query rewriting phase, additional subsumption checks to reduce the size of the rewriting as soon as possible, a more strict ordering strategy for resolutions and data structures to simplify the search for relevant clauses. This approach keeps the resolution with free selection from REQUIEM as well as the possibility to handle $\mathcal{ELHIO}$ expressiveness. Hence it cannot be ensured that the output can be reduced to a (finite) UCQ as the Datalog produced can be recursive. Resolution

is split into a few more stages, more precisely the first resolution stage to produce the Datalog query is split into three stages, one is done in the preprocessing and the other two are performed separately with two different selection functions for efficiency reasons.

2.3 Previous Efforts in Benchmarking Query Rewriting Systems

All previous systems have been evaluated formally and empirically, at least with a set of examples. However, we find the first and most noteworthy example of benchmarking query rewriting systems in the work of Imprialou [24], who proposes an algorithm to generate an automatic benchmark.

This algorithm accepts an ontology as the input and generates a set of queries. There is no statement about how well these queries may represent real queries that could be posed by users. The automatic generation of the queries allows querying for all the concepts in the ontology. This coverage allowed the detection of problems in the soundness and completeness of the benchmarked algorithms. Most of the detected problems were solved in latter versions of these systems. Therefore, a good coverage is key for an appropriate benchmarking and even for the individual testing of the systems.

In this paper we give a more granular and quantitative focus to queries and their results assuming soundness and completeness wrt a given expressiveness. We do also analyse ontologies and the impact that their characteristics can have on the behaviour of query rewriting systems.

3 Analysis of Dimensions and Assets to Be Used for Benchmarking

3.1 Dimensions and Challenges in Query Rewriting Approaches

We will first start analysing the main differences among different query rewriting approaches, so as to be able to obtain some of the dimensions and challenges that should be considered when comparing among them:

Determine the Ontology Language Expressiveness. We have already seen that query rewriting in an OBDA context uses at least an ontology $\mathcal{O}$ to transform a query q into a rewritten query $q_{\mathcal{O}}$. The main characteristics of this process are the computational cost of the rewriting and the complexity of the rewritten query, both of which depend on the expressiveness of the ontology. We have also described briefly the main logics used in the state of the art in section 2. Each logic has a different expressiveness, which determines the coverage of the ontology that can be performed when converting it from a description logic language into a set of clauses in Datalog, Datalog± or FOL. The conversion to FOL happens in Rapid, REQUIEM and kyrie. Given the languages that they give support to, these systems consider expressions of the form $\exists R.A \sqsubseteq B$. These expressions generate Skolem functions, what implies the production of FOL clauses that are

later processed and converted to Datalog. Nyaya considers as well the existential quantifier in Datalog±.

Characterise the Impact of Reduced Expressiveness in Each Approach. The conversion of the ontology $\mathcal{O}$ into a set of clauses Σ that can be handled by the approach is one of the first main differences among systems, as discussed above. In fact, it normally happens that some axioms of the original ontology may be even lost or discarded in this transformation process. This can lead to unnecessarily complex queries, incomplete results and even incorrect results:

- Some systems may not be able to handle the additional expressiveness, which means the loss of completeness wrt that expressiveness and the loss of some answers.
- Non-recursive Datalog may be a precondition for some systems. This systems may enter infinite loops when the axioms in some ontology lead to the production of recursive Datalog.
- The subsumption of some clauses may be implied by some axioms out of the handled expressiveness. Thus clauses that could have been eliminated upon checking that subsumption will be preserved in the rewritten query.
- Negations (or disjunctions) may not be handled by some systems. This means the rewritten query may contain clauses that are contradictory and will not obtain answers (they could be eliminated) or if the data source is not consistent with these negations then incorrect answers may be obtained, according to the semantics of the TBox.

The impact of the lost expressiveness has not been considered in previous evaluations. And to the best of our knowledge, there is no evaluation about which is the usual expressiveness in the ontologies that can be found and reused to enable OBDA.

Determine the Complexity of Rewritten Queries. Evaluation of query rewriting systems has mainly focused so far on the most comparable cases: ontologies whose expressiveness is at the intersection of the expressiveness of all the systems to be compared. Most systems allow choosing whether the output of the query rewriting process should be a Datalog program or a UCQ. If Datalog is produced then this Datalog is always non-recursive for FOL-reducible logics. If the expressiveness is in the aforementioned intersection of all systems, then rewritten queries are ensured to be complete and correct wrt this expressiveness. In this case the only difference that can be found among different UCQs, for the same ontology and query, is in terms of subsumed clauses or atoms that are not removed from the final result. Datalog rewritings may vary more due to different ways in which subqueries can be arranged. This adds a lot of heterogeneity and complexity to the evaluation process.

Furthermore, there is neither standard nor a set of tools to convert the Datalog or the UCQs obtained with these systems to actual queries to perform on a

data source. As we have seen in section 2.2 some systems are integrated in OBDA systems and some are not, but the modular design and implementation does not come with a modular evaluation that can compare them properly. A workaround for this limitation has been proposed in the evaluation of Nyaya by measuring more carefully the UCQs generated (Datalog is not considered in this evaluation). More precisely they propose considering the number of clauses, the number of atoms and the number of joins. However, it is not specified whether the number of atoms is the total number of atoms or the number of distinct atoms. Atom repetition may have a big impact on the execution of the query depending on whether the query translation uses some kind of temporary tables to store intermediate results or repeated atoms are translated to as many queries as times they are related. However the differences among systems are small in the resulting UCQs due to the lack of freedom in this structure: UCQs are flat and unlike Datalog the results are basically equal in all systems, unless some subsumed atoms or clauses are kept in the UCQ or there is some loss of completeness or correctness. This should not be the case for any of these systems if the ontologies used do not exceed the expressiveness that they can handle.

Determine the Impact of the Complexity of Original Queries. The behaviour of query rewriting systems may also vary greatly depending on the original queries that are posed to the systems. Among the main characteristics to be considered for queries we can cite:

- length of the queries, in terms of predicates in the body of the query and variables in the head.
- types of variables and number of variables of each type. Variables may be present in queries in different roles
  - distinguished variables (those in the head of the query).
  - existential variables (non distinguished variables that appear only in one predicate).
  - join variables (non distinguished variables that appear in at least two predicates).
- length of the property paths in the queries.
- hanging or closed property paths. Hanging property paths leads to existential variables, closed property paths lead to distinguished variables.
- separability of some parts of the query as independent subqueries.

A set of queries has traditionally been used for evaluations in the state of the art. However, this set of queries has not been adequately characterised in terms of their realism or the coverage of the casuistry.

Usage of Additional Information for the Query Rewriting Process. Additionally to ontologies and queries, some systems add the possibility to use additional information for the query rewriting process (e.g., the use of an EBox in Prexto). Such additional information may provide potentially further optimisations of the process and results. However, as such additional information may

be very ad-hoc for each system, it is difficult to compare results among systems that use and do not use it. Furthermore, in the case of EBoxes, for instance, their application in realistic scenarios is still largely unknown, what makes it harder to evaluate the impact of EBoxes in the rewriting (process and results).

3.2 Assets Used for Evaluations

Despite the difficulties and challenges presented above, several approaches have been formulated for the comparison of query rewriting approaches. However, there is a large heterogeneity in these approaches, what claims for the need to come up with a common evaluation framework to allow better comparisons and to provide sufficient information as well for system developers to improve their systems, as in any general benchmarking process.

Some of the seminal work can be attributed to the perfect reformulation proposal from Calvanese [6], which is evaluated only in theoretical terms w.r.t. complexity, completeness and correctness. Pérez-Urbina compared his approach with this by using a set of ontologies that have become common across evaluations in this area:

- Adolena (A). Developed to allow OBDA for the South African National Accessibility Portal [25]. This is the largest ontology in this set of ontologies.
- path1 (P1) and path5 (P5). Synthetic ontologies to help understand (and show) the "impact of the reduction step" in REQUIEM. Despite their apparent simplicity, rewriting times for Datalog in these ontologies are significant.
- StockExchange (S). It captures information about European Union financial institutions, Rodríguez-Muro [26] uses this as a driving example to explain OBDA and how users may benefit from it.
- University (U). A DL-Lite$_R$ version of LUBM [27]. LUBM focuses on the ABox more than the TBox. On the one hand this allows systems like Clipper to use the ABox for further evaluation of rewritten queries. On the other hand it has a rather flat TBox [28], which means that the rewritings are simple, it takes a short time to be produce them and the rewritten queries are also short, as we will see in the next section.
- Vicodi (V). An ontology of European history developed in the EU-funded VICODI project [29].

This set of ontologies was expanded with AX, P5X, and UX, where some of the previous ontologies included auxiliary predicates. These auxiliary predicates replace the existential predicates by applying the encoding required by the previous approach [6]. These ontologies are accompanied by a set of five queries for each of them.

For the evaluation of Presto these sets of queries are incremented up to seven queries for each of the ontologies. The ontologies are also incremented with the ontologies from Kontchakov [30], more examples from the LUBM benchmark (besides of U and UX) and a newly created ontology. The ontologies from Kontchakov are:

- Galen-Lite. The *DL-Lite*$_{core}$ approximation of the well-known medical ontology Galen [31]. The interest in this ontology is mainly in its taxonomy, and few axioms involve roles.
- Core. A *DL-Lite*$_{core}$ representation of (a fragment of) a supply-chain management system used by the bookstore chain Ottakar's, now rebranded as Waterstone's. Contrary to Galen, the taxonomy in Core is smaller but it contains a rich set of axioms about roles.

Later approaches like Rapid, Nyaya, Clipper and kyrie inherit these mentioned ontologies and queries, and keep on using them for evaluation purposes. Prexto is only compared with Presto by means of a small unnamed ontology to show the impact that the optimisations in Prexto may have, which is done more as an example than an empirical evaluation. In the case of kyrie some newly created ontologies named AXE, AXEb, P5XE and UXE are used. These ontologies expand AX, P5X and UXE by including additional axioms that fall in the expressiveness of $\mathcal{ELHIO}$, which is not covered in less expressive systems. Again the purpose of this evaluation is showing how these axioms may impact results, regardless of realism or empirical significance.

Upon a closer analysis of the ontologies and the queries we can see that they are fairly heterogeneous, rewritings may vary greatly in time and size, depending on the characteristics of the ontology and the query as we have mentioned in the previous section. Finally, there is not a well-founded analysis of whether these ontologies are representative enough to cover all the challenges and characteristics that we have discussed in the previous section.

4 Experimental Results

We have executed five of the systems described in section 2.2 (REQUIEM with the F mode, Presto, Rapid, Nyaya or Clipper, and kyrie) using the assets (ontologies and queries) described in section 3.2. In this paper we will only show and discuss results for ontologies U and UX, which are representative enough for our analysis and are also some of the most widely used, as they are based on LUBM. The full results for this evaluation are available online[1].

In table 1 we can see the results for the execution of the queries for these two ontologies when the output is set to datalog. In table 2 we can see the analogous results when the output is set to UCQ. Please note that Nyaya and Clipper only produce one type of output, thus one replaces the other on each table.

The results have been obtained on cold runs, by restarting the applications after every query since there was no perceptible difference in the results. The consistency of the results regardless of how the systems are run has been ensured by appropriately measuring the time required by discarding all operations done before and after the query rewriting. The only exception to this rule was Prexto, where the results differed; here we show the results for a second run in Prexto and we refer the reader to the full results to check the times on a cold run.

[1] `http://www.oeg-upm.net/files/jmora/iswc2013/`

Table 1. Results of the execution to obtain datalog (time in ms)

		REQUIEM(F)		Presto		Rapid		Clipper		kyrie	
$\mathcal{O}$	q	size	time	size	time	size	time	size	time	size	time
U	1	19	12	4	7	4	3	2	21	2	0
	2	47	16	2	9	2	9	49	19	47	3
	3	20	9	8	16	8	12	21	24	20	3
	4	64	15	3	12	3	3	63	18	64	3
	5	53	12	8	15	8	12	53	15	53	0
	6	20	12	19	6	21	15	16	18	16	9
	7	49	25	22	15	22	18	44	18	45	12
	8	10	9	13	6	13	9	10	20	10	3
	9	29	15	24	12	24	17	19	20	21	7
UX	1	22	6	7	10	7	3	5	27	5	6
	2	52	15	2	15	2	15	54	27	52	3
	3	24	15	10	28	10	9	25	28	24	3
	4	70	15	6	19	6	12	69	22	70	0
	5	56	15	11	15	11	15	56	21	56	12
	6	24	18	28	15	27	22	20	19	20	3
	7	55	28	29	21	27	21	50	28	51	12
	8	11	6	14	12	14	13	11	26	11	1
	9	32	15	30	21	30	15	22	46	24	4

Table 2. Results of the execution to obtain UCQ (time in ms)

		REQUIEM(F)		Rapid		Prexto		Nyaya		kyrie	
$\mathcal{O}$	q	size	time	size	time	size	time	size	time	size	time
U	1	2	15	2	3	2	9	2	5	2	0
	2	1	103	1	15	1	15	1	1	1	34
	3	4	212	4	9	4	18	4	34	4	18
	4	2	3762	2	12	2	15	2	4	2	50
	5	10	13034	10	18	10	15	10	33	10	37
	6	29	47	29	28	28	12	40	1595	29	28
	7	42	797	42	37	70	18	54	670	42	43
	8	10	15	10	18	10	6	10	63	10	3
	9	960	1893	960	209	960	928	960	75135	960	1107
UX	1	5	15	5	12	5	12	5	22	5	9
	2	1	172	1	12	1	16	1	3	1	37
	3	12	2062	12	15	12	28	12	55	12	21
	4	5	31422	5	15	5	23	5	6	5	47
	5	25	91878	25	27	25	23	25	39	25	46
	6	323	468	323	106	448	178	348	2685	323	187
	7	1456	37212	224	81	280	75	264	852	224	121
	8	20	21	20	23	20	15	20	61	20	9
	9	4200	30506	4200	739	4200	21181	4200	366673	4200	16875

Each query has been run a minimum of five times per system and the results averaged. The hardware used in the evaluation is a Intel®Core™2 6300 @1.86GHz with 2GB of RAM, Windows® XP and Java™version 1.6.0_33. We have measured the systems in the same way they are measured in their respective evaluations, displaying the number of clauses, time for the rewriting, etc.

The times have been measured with the code the systems provide to do this. All systems are developed in Java and for time measurement most of these systems use the difference between two calls to `System.currentTimeMillis()`. The only two exceptions are Nyaya and Presto/Prexto. In the case of Nyaya `System.nanoTime()` is used instead. In the case of Presto and Prexto we did not have access to the source code but the results obtained take measurements of the time with an accuracy of 15 or 16 milliseconds, which suggest that the same method (`System.currentTimeMillis()`) is used. Regardless of the accuracy of different methods for time measurement and the possibilities to improve it, we can consider rewriting times are negligible when they are shorter than 15 milliseconds, specially when compared with the times that may be needed for the actual execution of the rewritten queries in an OBDA context.

Upon closer inspection of the results we can see that a single axiom in an ontology or a single atom in a query can produce very significant variations in the obtained results wrt query rewriting time and wrt size of the rewritten queries. This is the case for query 5 in AX and AXE, differing only in one axiom. We can see a similar example in queries 6 and 7 in ontology UX, differing only in one atom. This impact in' behaviour stresses the relevance of representative test cases, to show and evaluate the behaviour of these systems in realistic scenarios including those that may have a big impact on performance or results.

5 Evaluation and Conclusions

We can group the shortcomings we have seen in section 3 into two main groups, one relating with the input of the systems and the other relating with the output.

On the side of the **input** we need to know how well the tests represent reality, in terms of (1) queries wrt (1a) syntax, (1b) expressiveness, (1c) shape and (1d) size, (2) ontologies in terms of (2a) shape, (2b) size and (2c) expressiveness and whether it is possible to consider (3) additional information such as ABox dependencies or EBoxes.

On the side of the **output** we need to know how well rewritten queries may perform when posed to some other system, and again we should focus on the same details like the (1) shape in terms of (1a) expressiveness (Property paths in SPARQL, subqueries in Datalog, UCQs), (1b) types of clauses (non-recursive Datalog, linear Datalog), (1c) syntax with special characteristics (SPARQL, SQL, Datalog), and (2) size in terms of (2a) number of clauses, (2b) number of atoms and distinct atoms, (2c) number of joins.

In the experimental results we have seen these characteristics of the input of these systems are relevant and may cause the output to vary greatly. Clearly the same principle may apply to underlying systems, which means that the same care should be put on the evaluation of the output.

We have given a first step to the solution of these shortcomings. Nowadays we can make better comparisons thanks to the additional approaches and how they behave in different testcases. We can see that:

- **REQUIEM** was an interesting addition to the state of the art and being among the most expressive systems allows comparisons with systems that handle less expressive logics.
- **Rapid** has improved REQUIEM in every aspect except expressiveness, which is more reduced, if the ontology can be handled by Rapid then this is the most competitive approach on the light of the data.
- **Prexto** is unique among the evaluated approaches in the handling of the EBox. The EBox can produce dramatic improvements by reducing the size of the rewritten query and the time to obtain it.
- **Nyaya** is the only system that paid attention to the size of the queries in terms of atoms and joins. As the area matures and standardization continues we will be able to evaluate the impact of these factors on the underlying systems.
- **Clipper** improves previous approaches in terms of expressiveness, it handles the most expressive logic among the evaluated approaches (Horn-$\mathcal{SHIQ}$) but its output is also the most expressive, only Datalog meant to be evaluated by Datalog engines. It offers the most and it also requires the most.
- **kyrie** has improved REQUIEM in every aspect except expressiveness, which is exactly the same. If the ontology cannot be handled by Rapid then kyrie allows handling this expressiveness more efficiently than REQUIEM.

As more systems are developed and evaluated, we will be able to ascertain with more detail the impact of their differences by analysing how they behave. As more test data is available, we will be able to determine with better accuracy the significance of each test, how representative they are among all the tests. Finally, as the area goes through standardization and the underlying systems are standardized in the area, we will be able to compare how well the output of rewriting systems can be handled by these underlying systems.

In this paper we have shown that it is already possible to do this evaluation despite of current limitations, we have pointed at current limitations and future lines in the area and we have compiled and shared test data (available in the links) to help to address mentioned current limitations and future lines.

Acknowledgements. We would like to kindly thank authors of systems (1) REQUIEM, (2) Rapid, (3) Presto/Prexto, (4) Nyaya and (5) Clipper for their help in the usage of their systems and respectively (1, 2, 5) publishing the code, (3) sharing the binaries and (4) sharing the code.

The work presented in this paper has been funded by an PIF grant (Personal Investigador en Formación) from UPM (Universidad Politécnica de Madrid) (RR01/2008) and by the Spanish national project myBigData (TIN2010-17060).

References

1. Calvanese, D., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Ontology -based database access, tech. rep., CiteSeerX (2007)
2. Calbimonte, J.-P., Corcho, O., Gray, A.J.G.: Enabling ontology-based access to streaming data sources. In: Patel-Schneider, P.F., Pan, Y., Hitzler, P., Mika, P., Zhang, L., Pan, J.Z., Horrocks, I., Glimm, B. (eds.) ISWC 2010, Part I. LNCS, vol. 6496, pp. 96–111. Springer, Heidelberg (2010)
3. Priyatna, F., Buil-Aranda, C., Corcho, O.: Applying SPARQL-DQP for federated SPARQL querying over google fusion tables. In: ESWC 2013 Demo (2013)
4. Das, S., Sundara, S., Cyganiak, R.: R2RML: RDB to RDF mapping language. W3C RDB2RDF Working Group (September 2012)
5. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: What is query rewriting? In: Klusch, M., Kerschberg, L. (eds.) CIA 2000. LNCS (LNAI), vol. 1860, pp. 51–59. Springer, Heidelberg (2000)
6. 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, 385–429 (2007)
7. Cuenca Grau, B., Horrocks, I., Motik, B., Parsia, B., Patel Schneider, P., Sattler, U.: OWL 2: The next step for OWL. Web Semantics: Science, Services and Agents on the World Wide Web 6, 309–322 (2008)
8. Calì, A., Gottlob, G., Pieris, A.: New expressive languages for ontological query answering. In: Burgard, W., Roth, D. (eds.) AAAI. AAAI Press (2011)
9. Pérez-Urbina, H., Horrocks, I., Motik, B.: Efficient query answering for OWL 2. In: Bernstein, A., Karger, D.R., Heath, T., Feigenbaum, L., Maynard, D., Motta, E., Thirunarayan, K. (eds.) ISWC 2009. LNCS, vol. 5823, pp. 489–504. Springer, Heidelberg (2009)
10. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The DL-lite family and relations. J. Artif. Int. Res. 36(1), 1–69 (2009)
11. Gottlob, G., Orsi, G., Pieris, A.: Ontological query answering via rewriting. In: Eder, J., Bielikova, M., Tjoa, A.M. (eds.) ADBIS 2011. LNCS, vol. 6909, pp. 1–18. Springer, Heidelberg (2011)
12. Calì, A., Gottlob, G., Pieris, A.: Query answering under non-guarded rules in datalog+/-. In: Hitzler, P., Lukasiewicz, T. (eds.) RR 2010. LNCS, vol. 6333, pp. 1–17. Springer, Heidelberg (2010)
13. Bachmair, L., Ganzinger, H.: Resolution theorem proving. In: Handbook of Automated Reasoning, vol. 1, pp. 19–99 (2001)
14. Lynch, C.: Oriented equational logic programming is complete. Journal of Symbolic Computation 23, 23–45 (1997)
15. Rosati, R., Almatelli, A.: Improving query answering over DL-Lite ontologies. In: Lin, F., Sattler, U., Truszczynski, M. (eds.) Proceedings of the Twelfth International Conference on the Principles of Knowledge Representation and Reasoning. AAAI Press (2010)
16. Stamou, G., Trivela, D., Chortaras, A.: Progressive semantic query answering. In: The 6th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2010), p. 112 (2010)
17. Chortaras, A., Trivela, D., Stamou, G.: Optimized query rewriting for OWL 2 QL. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) CADE 2011. LNCS, vol. 6803, pp. 192–206. Springer, Heidelberg (2011)

18. Gottlob, G., Orsi, G., Pieris, A.: Ontological queries: Rewriting and optimization (extended version). arXiv:1112.0343 (December 2011)
19. Venetis, T., Stoilos, G., Stamou, G.: Query rewriting under query extensions for OWL 2 QL ontologies. In: The 7th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS 2011), p. 59 (2011)
20. Rosati, R.: Prexto: Query rewriting under extensional constraints in DL-Lite. In: Simperl, E., Cimiano, P., Polleres, A., Corcho, O., Presutti, V. (eds.) ESWC 2012. LNCS, vol. 7295, pp. 360–374. Springer, Heidelberg (2012)
21. Eiter, T., Ortiz, M., Šimkus, M., Tran, T.-K., Xiao, G.: Query rewriting for horn-SHIQ plus rules. In: Proc. of the 26th AAAI Conference on Artificial Intelligence. AAAI (2012)
22. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The DLV system for knowledge representation and reasoning. ACM Trans. Comput. Logic 7, 499–562 (2006)
23. Gebser, M., Kaufmann, B., Kaminski, R., Ostrowski, M., Schaub, T., Schneider, M.: Potassco: The potsdam answer set solving collection. AI Commun. 24, 107–124 (2011)
24. Imprialou, M., Stoilos, G., Grau, B.C.: Benchmarking ontology-based query rewriting systems. In: Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence. AAAI (2012)
25. Keet, C.M., Alberts, R., Gerber, A., Chimamiwa, G.: Enhancing web portals with Ontology-Based Data Access: the case study of South Africa's Accessibility Portal for People with Disabilities. In: OWLED (2008)
26. Rodriguez-Muro, M., Lubyte, L., Calvanese, D.: Realizing ontology based data access: A plug-in for protégé. In: IEEE 24th International Conference on Data Engineering Workshop, ICDEW 2008, pp. 286–289 (2008)
27. Guo, Y., Pan, Z., Heflin, J.: LUBM: a benchmark for OWL knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web 3, 158–182 (2005)
28. Rodriguez-Muro, M., Calvanese, D.: High performance query answering over DL-Lite ontologies. In: Brewka, G., Eiter, T., McIlraith, S.A. (eds.) KR. AAAI Press (2012)
29. Nagypal, G.: History ontology building: The technical view. In: Proceedings of the XVI International Conference of the Association for History and Computing, pp. 207–214. Royal Netherlands Academy of Arts and Sciences, Amsterdam (2005)
30. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: Combined FO rewritability for conjunctive query answering in DL-Lite. In: Grau, B.C., Horrocks, I., Motik, B., Sattler, U. (eds.) Description Logics. CEUR Workshop Proceedings, vol. 477, CEUR-WS.org (2009)
31. Rogers, J., Rector, A.: The GALEN ontology. Medical Informatics Europe MIE 1996, 174–178 (1996)