Efficient Query Rewriting for RDB2RDF Systems based on mapping coverage

Jose Mora[1] and Oscar Corcho[1]

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

Abstract. RDB2RDF systems generate RDF from relational databases, either by syntactically transforming the relational database schema or using mapping languages with a semantically richer correspondence between the database schema and existing ontologies, to which the generated RDF data can be related. Besides, these systems operate in two different manners: materializing the database content into RDF or acting as virtual RDF datastores that transform SPARQL into SQL queries. In the former, query answering is handled by the RDF triple store where the RDF data is materialized, and inferences could be handled by this triple store, even if not natively considered in the SPARQL semantics. In the latter, existing RDB2RDF systems do not normally perform inferences that could be done with the corresponding ontologies. This paper shows how the algorithm used in the REQUIEM system can be adapted to handle run-time inferences in query answering with RDB2RDF systems that use rich mapping languages, such as ODEMapster.

Keywords: RDB2RDF, query rewriting, reasoning

1 Introduction

Organisations store ever growing amounts of information and a big part of that information is stored in relational databases (RDBs). In the context of the Semantic Web, and specially related to the Linked Data initiative, RDB2RDF tools are being used to generate RDF from a large number of these databases.

These RDB2RDF tools may use mappings for a merely syntactical conversion from the database schema, usually performed automatically, or may use mappings with existing ontology, relating the generated RDF with their classes and properties and thus considering semantics for this conversion to RDF.

Their mapping implementation may be either a static Extract Transform Load (ETL) implementation or a query-driven dynamic implementation, depending on how the generation of the RDF code is performed, either in a batch process for the total of the information contained in the database or as an on-demand process for the answering of each query posed to the system [1].

Although RDB2RDF tools have attracted more attention recently due to the emergence of the Linked Data initiative, where efforts are mainly focused on the publication of RDF data according to the Linked Data guidelines, some of these tools originated earlier as a means to create wrappers in ontology-based information integration architectures [2]. This will be the focus of this paper.

Finding answers in information integration scenarios requires a logical and physical path search. The former consists on finding rewritings of the query that keep the semantics. The latter consists in finding the physical information sources that provide the relevant information for the rewritten query [3]. Thus in information integration the rewriting consists on these two aspects. On the one hand the concepts involved in the query can be replaced by the combination of other concepts according to the inference enabled by the TBox. On the other hand, once the relevant concepts have been found it is necessary to pose the queries to the original sources of information, for which the mappings, either GAV, LAV or GLAV [4], between this TBox and the local schemas of the sources are used.

In information integration scenarios using ontologies [2,5] description logic (DL) has been applied extensively. The TBox normally takes the role of the global schema [6] in the description of the contents of the integrated ABoxes, which are constructed from the distributed data sources.

This knowledge base provides a more expressive description of the information contained in these ABoxes and allows the use of reasoning for query answering [7]. The reasoning procedure uses the TBox for the rewriting of the original query according to the rules defined for the ontology language used. As the expressiveness of the ontology language increases so do the number of rules, and thus the complexity of this process. After the rewriting, a greater number of simpler queries are generated, whose combination will provide the results for the original query. However, performing this reasoning introduces an efficiency penalty in the query answering process, and thus most RDB2RDF systems with run-time query answering approach do not use any reasoning [1].

Moreover, when the ABox is generated from a RDB by the use of an RDB2RDF tool and a TBox is added for the provision of reasoning capabilities, the presence of concepts and relations in the TBox that are not covered in the RDB2RDF mappings is not uncommon. This is even mroe common in information integration scenarios, where the same TBox, as a global schema, covers several ABoxes. In these scenarios, the previously stated logical path search corresponds with the reasoning performed to deduce predicates that contain answers that are correct for the predicates in the original query. Reasoning in those cases can be seen as a navigation through the clauses in the TBox, from the predicates in the original query to those offered by the RDB2RDF mappings. If a predicate is the head in several clauses in the TBox, all of them will provide correct answers for this predicate, but when only some of them lead to RDB2RDF mappings and the remaining clauses do not provide answers, these remaining clauses can be removed from the obtained query plan.

Pruning these clauses that do not provide any solution leads to a reduction in the number of clauses that the system has to handle, which causes an increase in the efficiency of the whole process. As usual, the sooner the information that is being processed is pruned, the better the consequences in the load reduction for the following steps of the process are. With this rationale, in this paper a method to prune the number of clauses involved in the process of query rewriting is presented. The method, as will be explained later, has two working modes and acts at two different phases of the algorithm.

This paper is structured as follows: section 2 will provide some background about the RDF2RDF tools and their mappings, as well as the use description logics used to describe the global schema in information integration and the reasoning methods used with these logics for query rewriting. Section 3 will explain he method used to prune the inference to avoid the generation of queries rendered as futile for the mismatch between RDF2RDF mappings and the description logics global schema. Section 4 presents the evaluation and comparison with the results obtained, where the efficiency gain can be checked. Finally, section 5 contains the conclusions of the paper.

2 Background

The contribution presented in this paper starts from the use of reasoning to rewrite queries posed in a global schema and the use of RDB2RDF tools to obtain from a relational database the RDF individuals that populate that schema.

On the one hand, it is important to know the common capabilities, limitations and the differences between RDB2RDF tools, as the information provided by them through the use of their mappings is the base on which the query answering system lies. Some details will be provided in section 2.1 in addition to the brief description already provided in section 1.

On the other hand, the rules in the corresponding DL family in which the ontology is implemented define the rules that will allow the rewriting of the query as they are unified with the axioms provided by the TBox. The corresponding reasoning algorithm describes the way in which these rules are going to be applied and the solutions that will be found through the inference steps given through the application of these rules. Since this is a very broad area, the focus in this paper will be set on the use of reasoning for query rewriting. Thus, it is also important to know current state of the art in the relationship between query rewriting algorithms and DL families for a better comprehension of the remainder of the paper.

2.1 RDB2RDF tools

The importance of offering the information available in relational databases (RDBs) as RDF is reflected in the number of approaches taken to produce RDF data from relational databases, which are classified into three main categories [1]: proof of concept projects, domain-specific projects and tools/applications. All RDB2RDF tools take some information from a RDB and offer some RDF as output. The representation of the correspondence between the information in the DB and the code generated in RDF is done by means of mappings. There are important differences in the expressiveness of these mappings [8], for instance in how the selection of the data from the DB is performed, in whether it is possible to select elements just from tables or whether filter conditions can be applied to rows and columns, etc. The transformation operations to generate URIs and derived data from raw data are also a significant source of differences among systems.

Those differences will lead to the generation of RDF code with different degrees of quality. For instance, it is specially relevant if there is some "information gain" in the conversion, in comparison to a pure syntactical conversion, which is often found in automatic tools [1]. However the information gain is still limited in systems with run-time query answering approach, since the reasoning capabilities that the use of an ontology may provide are often omitted due to the efficiency penalty that reasoning implies in the query rewriting process.

A compromise solution consists in performing some efficient reasoning, in polynomial time, with limited expressiveness and reasoning capabilities, as explained in the next section.

2.2 Query rewriting on description logics

Query rewriting produces different a number of rewritings of a query that preserve the desired results that are to be obtained from it. Query rewriting can be seen as a search for solutions, in this case rewritings, which may be logical (among classes or properties) or physical (among sources of information) [3]. In an information integration scenario, where there may be several distributed sources with different capabilities in the information that they provide, physical search will be crucial. Besides, the logical search is focused on selecting the concepts and properties that provide useful answers for the query that is posed to the system.

The efficiency of the query rewriting process is influenced by two factors: firsts, the number of axioms in the ontology and the specific way in which these are combined in it; second, the expressiveness of the ontology language that is used to describe the global schema. From these two factors, the most relevant one in terms of determining the efficiency of the process is the latter. Several alternatives have been proposed in the search of this efficiency while covering the highest expressiveness that is possible at an adequate level of complexity.

With this motivation *DL-Lite* was created as a family of description logics that keeps efficient reasoning. Specifically, the complexity of reasoning in *DL-Lite* is polynomial in time and size of the TBox and LogSpace with respect to the size of the ABox [7]. The *DL-Lite* family includes *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$, the former includes the possibility to express functional restrictions on roles while the latter includes ISA and disjointness assertions between roles.

As a desired consequence of this limited expressiveness for a more efficient handling, these logics ease the representation of ABox assertions as relations managed in a secondary storage in a database, which is consequent with the RDB2RDF approach. This allows to get the most from both approaches, the efficiency in management of large quantities of data of databases and the expressive power of the explicit schema in description logics. These capabilities and possibilities have led to the creation of the *QL* profile in *OWL 2* [9], tailored for the query rewriting to SQL storages for ABoxes. In order to ensure that a query can be rewritten into a union of conjunctive queries, *OWL 2 QL* forbids the use of disjunction and AllValuesFrom restrictions, as well as certain other features that require recursive query evaluation. The profile *EL* in *OWL 2* has also a special relevance, since it is based on the $\mathcal{EL}++$ family of description logics and it is designed to reason with large terminologies. The central modeling features of this profile are class conjunction and SomeValuesFrom restrictions. In order to achieve tractability, the use of negation, disjunction, AllValuesFrom restrictions, and cardinality restrictions is disallowed.

As previously stated, there are different families of logics with different expressivenesses that allow the rewriting of the query to SQL, and provide similar expressivenesses since even slight increases in expressiveness would prevent the queries from being evaluable by a SQL engine over the ABox [7]. In this paper the focus will be set on $\mathcal{ELHIO}^{\neg}$ DL, which is one of the most expressive Horn logics for which query answering is polynomial with respect to data complexity [10]. Beside of $\mathcal{EL}$, as in the previously described *EL* profile of *OWL 2*, it provides basic concepts of the form $\{a\}$, inverse roles and role inclusions. The reasoning in this logic can be carried out by the REQUIEM algorithm, presented in [11], which performs saturation on the query and the relevant part of the *DL-Lite* ontology to produce all the rewritings. The whole process can be divided in four main phases performed over the ontology and query:

- In the first phase, the ontology and the query are parsed and clausified. (The clausification process for $\mathcal{ELHIO}^{\neg}$ *KBs* is explained in [10]). The statements that fall out of the expressiveness of $\mathcal{ELHIO}^{\neg}$ are discarded.
- In the second phase, the ontology is pruned and the part of it that is useful for the generation of rewritings is taken for the following phases, since saturation performs all the possible inferences. This is how the inferences are limited to those useful for the current query.
- In the third phase, saturation is performed over the ontology and query, generating a datalog program as query rewriting. The datalog program is pruned, removing clauses that contain equality, functional terms or nominal terms.
- In the fourth phase, the datalog program is resolved, generating the set of conjunctive queries that is the rewriting of the original query and that can be posed to the database or set of databases that are abstracted with the schema used by the algorithm.

After that process the queries that should be posed to the DB according to the ontology and the original query are generated, without considering the possible query containment (and therefore redundancy) or the availability or the information in the DB that is to be queried.

3 Improvements over the REQUIEM query rewriting process

The REQUIEM query rewriting process, described in section 2.2, is pruned at two specific points. First, the ontology is pruned for the inference; second, the datalog program generated by the saturation process that continues with this pruned ontology is pruned again, after its generation, so that it only contains predicates that can be obtained from the declared RDB2RDF mappings. The resulting queries generated from the datalog program are noticeably reduced after both prunes. Our proposal consists on pruning the information that is handled by the process at two specific points and according to two different methods that will be useful in different situations.

The first of these two points is the pruning process after the clausification of the ontology, which can be more restrictive, including only the clauses that are relevant and can provide results according to the declared RDB2RDF mappings. The prune at this stage can be performed according to two methods, depending on the inference that the RDB2RDF system may perform, if the system is able to perform some inference the prune can be more restrictive, otherwise, this part of the logical path search will need to be performed by the rewriting algorithm, some clauses will be required for that inference and shall not be pruned. This will be further explained in section 3.1.

The second of these two points is immediately after the generation of the datalog program. The generation of the datalog program may require the use of some clauses for the inference, however, since the datalog program may serve as an input for other processes and algorithms, it is pruned for a better efficiency, the prune removes the clauses that contain predicates that are not included in the declared RDB2RDF mappings without losing relevant answers to the query. This process is explained in more detail in section 3.2.

3.1 Pruning the ontology

The parsing and clausification processes, which define the ontology expressiveness that can be handled, have been reused from REQUIEM [10]. This means the axioms in the KB that fall in the expressiveness of $\mathcal{ELHIO}^{\neg}$ will be clausified and the remainder will be discarded, which may lead to the loss of some answers for the axioms that cannot be clausified.

We start from the assumption that our global schema will be described in the $\mathcal{ELHIO}^{\neg}$ DL [10]. In the usual case that our schema is described in OWL, we will consider only the axioms that fall under the $\mathcal{ELHIO}^{\neg}$ DL, as it is done in other works (e.g, [10]). It is important to note that we are aware of the fact that this axiom exclusion may have some relevant implications in the results obtained from the query rewriting process, and this as a part of our future work we will aim at providing user support tools to inform about those consequences.

This $\mathcal{ELHIO}^{\neg}$ projection of the original ontology is clausified as described in [10]. Once the ontology has been clausified, finding the rewritings is analogous to searching through a tree in this context. In this case the head of the query would act as the root of the tree, the children of the root would be the predicates in the body of the query, and as far as these predicates are in the head of other clauses the process can be repeated, conforming a tree. The leaves of the tree will be a set of predicates that are not in the head of any clause. This tree, avoiding loops, is the result of the prune process in REQUIEM, containing only clauses relevant for the current query. Table 1 contains a fragment of an hydrology ontology and its translation to clauses that will be used as an example for further clarification. nombres traducidos de Hydrontology, al ser lenguaje tcnico pueden contener errores, pendiente de revision

$\mathcal{ELHIO}$ clause	$\mathcal{ELHIO}$ axiom
1 $Water(x) \leftarrow DrainsAt(x, y)$	$\exists drainsAt \sqsubseteq Water$
1 $DrainsAt(x, drains(x)) \leftarrow RunningWater(x)$	$RunningWater \sqsubseteq \exists drainsAt$
2 $RunningWater(x) \leftarrow River(x)$	$River \sqsubseteq RunningWater$
3 $River(x) \leftarrow Tributary(x)$	$Tributary \sqsubseteq River$
4 $RunningWater(x) \leftarrow Stream(x)$	$Stream \sqsubseteq RunningWater$
5 $Water(x) \leftarrow StillWater(x)$	$StillWater \sqsubseteq Water$
6 $StillWater(x) \leftarrow Enclosure(x)$	$Enclosure \sqsubseteq StillWater$
7 $Enclosure(x) \leftarrow SaltMarsh(x)$	$SaltMarsh \sqsubseteq Enclosure$
8 $SalineGround(x) \leftarrow SaltMarsh(x)$	$SalineGround \sqsubseteq SaltMarsh$
9 $Morphology(x) \leftarrow FloodableArea(x)$	$FloodableArea \sqsubseteq Morphology$

Table 1. Axioms in the example ontology and the corresponding clauses

In the example the query posed to the ontology in table 1 is $Q(x) \leftarrow Water(x)$. In this case, the clauses relevant for the query will provide information about $Water$ and thus the starting point are clauses whose head is the "Water" predicate, this is clauses 1 and 5. The predicates referred in the body of clause 1, DrainsAt lead to clause 2, and so we can continue the process, conforming the tree of the clauses that may be relevant for the rewriting of the query. In this case the only clause pruned is the last one, since it is not connected to any other by this procedure, and since the head present in this clause is not present in the tree it cannot be unified in the resolution. Notice that if some clause could be unified with a predicate in the body of this discarded clause that would still be irrelevant for the resolution of the query, in that case either directly or transitively Q could imply $Morphology$, but the implication of Q by $Morphology$ is impossible even transitively given the set of clauses in the example. se puede hacer una demostracion mas teorica y estricta, pero es un poco evidente

Notice that a simple fragment of the ontology has been chosen for the example, most of the predicates are unary and all the bodies in the rules are composed by one single atom, the focus in the example is on the additional prune that can be done when considering that only some predicates are mapped by a RDB2RDF tool, but $\mathcal{ELHIO}$ *DL* has a much greater expressiveness as shown in [10]. That expressiveness has no impact on the prune presented here, though.

When considering that only some predicates are mapped by some RDB2RDF mappings and thus only some predicates can provide valid answers, the prune can be more strict. This prune can be performed according to two different methods, depending on whether the RDB2RDF system performs any inference and for instance the answers of subconcepts are included in their respective concepts or not. These two methods replace the original pruning step performed in REQUIEM and generate a number of clauses that is smaller or equal, depending on the mapped predicates and the method used. In the example the mapped predicates will be $River, Enclosure$ and $SalineGround$.

If the inclusion relies on inference and retrieving the instances from one concept requires using the mappings for the subconcepts, then the prune cannot be stopped as soon as a retrievable concept is found and has to continue to the lowest retrievable concept in this search tree, possibly including not retrievable gaps that will have to be pruned in the next pruning step. This is specially relevant in the cases where the ontology is used for querying but not for the mappings. Since the taxonomy is known at query time these relations are not reflected in the mappings, it is also important in any case in which there are no underlying systems that would take care of this inference. In this case, *Enclosure* would be relevant and so would *SalineGround*, since the information of *SalineGround* being included in *Enclosure* is only known at query time, with the ontology, or for any other reason the RDB2RDF system does not guarantee that the information about salines is included when retrieving the values for enclosures.

On the contrary, if querying a concept or property retrieves all the values from the subconcepts or subproperties, then all the lower concepts or properties, the datalog predicates that imply the current one, can be pruned, since they will not provide additional values. In the previous example, once the mapping with *Enclosure* has been found, the predicates in its body would not be inspected, since the RDB2RDF mappings return all the instances for *Enclosure*, including *SalineGround*, whose mapping is rendered superfluous in this case.

This defines respectively two pruning methods: "global" for all retrievable concepts and properties and "first" that prunes the clauses that imply a specific concept or property if this concept or property is retrievable, since that predicate does provide all the information and no alternative rewritings are necessary. In table 2 the results of applying the original prune method 'N', the global 'G' or the first 'F' on the example are presented.

3.2 Pruning the datalog program

The generation of the datalog program in REQUIEM is performed through saturation. In the saturation phase some clauses that will need to be pruned later are still kept so that they can be used in the inferences. For instance, this happens to the clauses containing functions. Similarly, in our case, non retrievable predicates are still kept for the inference, and are removed after the useful clauses depending on them have been generated. In order to get a datalog program that

	'N'	'G'	'F'
1 $Water(x) \leftarrow DrainsAt(x, y)$	Yes	Yes	Yes
2 $DrainsAt(x, drains(x)) \leftarrow RunningWater(x)$	Yes	Yes	Yes
3 $RunningWater(x) \leftarrow River(x)$	Yes	Yes	Yes
4 $River(x) \leftarrow Tributary(x)$	Yes	No	No
5 $RunningWater(x) \leftarrow Stream(x)$	Yes	No	No
6 $Water(x) \leftarrow StillWater(x)$	Yes	Yes	Yes
7 $StillWater(x) \leftarrow Enclosure(x)$	Yes	Yes	Yes
8 $Enclosure(x) \leftarrow SaltMarsh(x)$	Yes	Yes	No
9 $SaltMarsh(x) \leftarrow SalineGround(x)$	Yes	Yes	No
10 $Morphology(x) \leftarrow FloodableArea(x)$	Yes	Yes	No

Table 2. Clauses kept after pruning

does not contain non retrievable predicates, but keeps the capability of retrieving all the answers, some resolution steps that would rely on non-mapped predicates have to be performed.

Once that the saturated datalog program is obtained as described in [11], we propose the execution of a new stage, focused on the prevention of infinite loops. To achieve this, we apply resolution to the clauses that contain non retrievable predicates so the resolution that would be done with clauses that contain non-mapped predicates in the next phase is done in this one. This way, all the inferences enabled by these clauses are already done at this phase and the corresponding clauses can be safely removed from the datalog program, pruning it without loss of generality.

The selection function in this case will select the unmapped atoms in the body of the clauses, if there are any. If not, the head will be selected in case that it is not mapped. This way, every inference made will remove one unmapped atom in the body of a clause, unifying it with the head of another clause, whose body will contain only mapped predicates, thus generating a clause that has at most one unmapped atom less than the base clause of the inference. After saturation has been performed using this selection function and no new clauses can be generated the clauses that contain unmapped predicates can be removed safely, without removing any valid answer with them.

Resuming the example from section 3.1 the saturation process would execute iterations and generate new clauses used for the inference in the next iteration, until there are no new clauses to generate and the saturation process finishes. This is explained in context with the previous example next. Before the saturation starts we have the ontology pruned, we will use the "global" strategy which is more convenient for the explanatory purposes of the example, and the contents are the following:

$$Q(x) \leftarrow Water(x) \tag{1}$$

$$Water(x) \leftarrow DrainsAt(x, y) \tag{2}$$

$$DrainsAt(x, drains(x)) \leftarrow RunningWater(x) \quad (3)$$

$$RunningWater(x) \leftarrow River(x) \quad (4)$$

$$Water(x) \leftarrow StillWater(x) \quad (5)$$

$$StillWater(x) \leftarrow Enclosure(x) \quad (6)$$

$$Enclosure(x) \leftarrow SaltMarsh(x) \quad (7)$$

$$SaltMarsh(x) \leftarrow SalineGround(x) \quad (8)$$

$$Morphology(x) \leftarrow FloodableArea(x) \quad (9)$$

In this case the saturation process does not perform any inference, so these are the clauses that serve as input in the added stage for the removal of non-mapped predicates. The resolution performed takes clauses two by two, first trying with all the combinations of the initial knowledge base and then, as new clauses are generated, with those clauses and all the previous clauses. This procedure always takes a non-mapped predicate in the body of a clause and unifies this predicate with the head of other clause that does not contain any non-mapped predicates in the body, thus every inference step generates a clause with at least one non-mapped predicate less than the clauses used to infer it, proving convergence. In our example the mapped predicates are $River, Enclosure$ and $SalineGround$. In the first step of the resolution we get:

$$from(3)and(4) : DrainsAt(x, drains(x)) \leftarrow River(x) \quad (10)$$

$$from(5)and(6) : Water(x) \leftarrow Enclosure(x) \quad (11)$$

$$from(7)and(8) : Enclosure(x) \leftarrow SalineGround(x) \quad (12)$$

In the second step of the resolution we get:

$$from(2)and(10) : Water(x) \leftarrow River(x) \quad (13)$$

$$from(1)and(11) : Q(x) \leftarrow Enclosure(x) \quad (14)$$

Similarly, in the third step we get:

$$from(2)and(10) : Water(x) \leftarrow River(x) \quad (15)$$

And finally in the fourth step we get:

$$from(2)and(10) : Q(x) \leftarrow River(x) \quad (16)$$

After pruning the clauses that contain non-mapped predicates we get the following datalog program:

$$Q(x) \leftarrow Enclosure(x) \quad (17)$$

$$Q(x) \leftarrow River(x) \quad (18)$$

$$Enclosure(x) \leftarrow SalineGround(x) \quad (19)$$

This datalog program generated with the 'g' variant of the algorithm contains all the mapped predicates that can provide useful answers, with a reduced number of clauses. In the next section the evaluation and quantification of this reduction is performed.

4 Evaluation

For the evaluation of the system two ontologies have been used, both developed and used in the context of independent projects. The first ontology is *hydrOntology* [12], with 155 concepts and expressiveness $\mathcal{SHIN}(\mathcal{D})$. *HydrOntology* has been used in several projects and mapped with several gepgraphical databases of the Instituto Geogrfico Nacional (IGN)[1], generating several RDB2RDF mapping files, which, being available, have been used for the tests. More precisely, the databases mapped here are three:

– Atlas which counts with X mappings.
– BCN200 providing X mappings.
– EGM that has X mappings.

The second ontology is *PhenomenOntology* [13], also developed in the context of cartography. In this case among the phenomena that could be covered only the module regarding transportation networks has been used, which provides 66 concepts, having a common ancestor in *"Red" ("Network")* which is the concept used for the test queries whose results are shown in table 3. In the case of *PhenomenOntology*, only one file with X mappings is used.

Both of these two ontologies are expressed in OWL, without imposing any kind of restriction, thus some of the axioms have to be discarded in the beginning of the process, as described in section 3.1 as they do not fall into the expressiveness of $\mathcal{ELHIO}^{\neg}$ *DL*. The mapping files are R2O mappings [8].

8	1	6
3	5	7
4	9	2

Table 3. Number of clauses in each stage and time in the process

The results, presented in table 3 vary depending on the method used. The first letter is for the methods in the original REQUIEM, 'N' for "naive", 'F' for "full forwarding" and 'G' for "greedy", the second letter stands for the modification applied as described in this paper, in this case 'N' stands for "none". 'F' will stop the prune on the first predicate that is mapped, as all the values that are correct for that predicate are assumed to be retrievable from the RDB2RDF mappings,

[1] http://www.ign.es

as described in section 3.1. Finally 'G' will be used for the "global" approach, which will keep all the mapped predicates in the ontology after pruning it, since they could be complementary to some extent, again as described in section 3.1.

The query posed to the system for comparison purposes is a general query covering a good part of the ontology, in the case of *hydrOntology* the query is simply $Q(x) \leftarrow Aguas(x)$, which covers as a superclass most of the taxonomy present in *hydrOntology*. Similarly in the case of *PhenomenOntology* the query used is $Q(x) \leftarrow Red(x)$, being a module about transportation networks, the concept "Red" is the most general one. Both queries cover most of the mappings and ontologies used for the tests, providing a good approximation of what could be expected in a different context.

5 Conclusions

As a first consequence the number of queries and information in several steps of the process can be reduced when considering the information provided by the RDB2RDF mappings, this is the information that can be retrieved from one single source, either a physical source or an abstraction of several sources. Because of this reduction there is a gain in efficiency, taking less time to rewrite the query. Therefore, an algorithm that takes into account the capabilities of the source of information can make a better use of this source queried, even when there is one single source. The rewriting of the query in logical terms, logical path search, is powered with this information, leading to better results.

There is still a big room for improvement. The redundancy of information in related concepts is not considered, and it is responsibility of the user to specify if there is no overlap or if there is a complete containment of subclasses by their respective classes, however, when the overlap is partial and multiple, checking the minimum set of predicates to retrieve the maximum number of answers to the original query is not trivial. Thus, the description of the information provided by the source, in logical terms, can be improved in this sense. Similarly, the information regarding the sources of information available and the information they provide is not considered here. The same way that concept inclusion can be considered, these concepts could keep information of the provenance, allowing the algorithm to differentiate among the concepts that are provided by different sources and again the overlapping that may exist among them.

Finally, the datalog program is heavily pruned immediately after its generation. Keeping a more descriptive datalog program and using a distributed query processor capable of handling this expressiveness would allow posing more complex queries to the sources. Albeit this is not a trivial task either. posiblemente ampliar, explicar que se diferencia de perez-urbina en la medida en que hay conceptos que no se han mapeado

6 Acknoledgements

gracias a la UPM por la beca, y gracias a Luis, Freddy y Miguel ngel por los casos de prueba. Gracias especiales a Hctor por compartir el cdigo y permitir su modificacin.

References

1. J. Sequeda, “A Survey of Current Approaches for Mapping of Relational Databases to RDF,” 2009.
2. H. Wache, T. Voegele, U. Visser, and H, “Ontology-based integration of information-a survey of existing approaches,” *IJCAI-01 Workshop:*, vol. 2001, pp. 108–117, 2001.
3. J. Bleiholder, S. Khuller, F. Naumann, and L. Raschid, “Query planning in the presence of overlapping sources,” *Advances in Database*, pp. 811–828, 2006.
4. M. Friedman, A. Levy, and T. Millstein, “Navigational plans for data integration,” *sixteenth national conference on Artificial intelligence and the eleventh Innovative applications of artificial intelligence conference*, pp. 67–73, 1999.
5. Paton N.W., Goble C.A., and Bechhofer S., “Knowledge based information integration systems,” *Information and Software Technology*, vol. 42, pp. 299–312, Apr. 2000.
6. A. Halevy, A. Rajaraman, and J. Ordille, “Data integration: The teenage years,” *conference on Very Large Data Bases (VLDB)*, pp. 9–16, 2006.
7. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, “Tractable Reasoning and Efficient Query Answering in Description Logics: The DL-Lite Family,” *Journal of Automated Reasoning*, vol. 39, pp. 385–429, Oct. 2007.
8. J. Barrasa, O. Corcho, and A. Gómez-Pérez, “R2O, an Extensible and Semantically based Database-to-Ontology Mapping Language,” *2nd Workshop on Semantic Web and Databases (SWDB2004)*, vol. 3372, pp. 1069—-1070, 2004.
9. B. C. Grau, I. Horrocks, B. Motik, B. Parsia, P. Patel-Schneider, and U. Sattler, “OWL 2: The next step for OWL,” *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 6, pp. 309–322, Nov. 2008.
10. H. Pérez-Urbina, B. Motik, and I. Horrocks, “Tractable query answering and rewriting under description logic constraints,” *Journal of Applied Logic*, vol. 8, pp. 186–209, June 2010.
11. H. Pérez-Urbina, I. Horrocks, and B. Motik, “Efficient Query Answering for OWL 2,” *The Semantic Web-ISWC 2009*, 2009.
12. L. Blázquez, M. Poveda, M. Suárez-Figueroa, and A, “Towntology &amp; hydrOntology: Relationship between Urban and Hydrographic Features in the Geographic Information Domain,” *Ontologies for Urban Development*, pp. 73–84, 2007.
13. A. Gómez-Pérez, J. Ramos, and A. Rodríguez, “The IGN-E Case: Integrating Through a Hidden Ontology,” *Headway in Spatial Data Handling*, pp. 417–435, 2008.