Grouping Answers in Ontology-Based Query Answering

Maxime Buron

▶ **To cite this version:**

Maxime Buron. Grouping Answers in Ontology-Based Query Answering. Logic in Computer Science [cs.LO]. 2017. hal-01622564v1

HAL Id: hal-01622564

https://inria.hal.science/hal-01622564v1

Submitted on 10 Nov 2017 (v1), last revised 27 Nov 2017 (v2)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Grouping Answers in Ontology-Based Query Answering

Maxime Buron
Supervised by Michaël Thomazo

September 4, 2017

Contents

Abstract

We aim at easing the use of semantic and structured data by improving the restitution of answers. Instead of providing a set/ ranked list of answers, we devise techniques to group answers and to semantically label the group in an informative way, easing further exploration of the answers.

1 Introduction

1.1 Context and Motivation

Among the wealth of resources that are currently available on the Internet, there exist some that have very interesting features for an automatized processing: those that are both *structured* and endowed with a *semantics*. Structured data allows for formal querying, whose precision is much higher than that of an intuitive but rather vague keyword querying. Semantics ease the formulation of queries, allowing the user to use a vocabulary with which he/she may be familiar with, ignoring the implementation details.

Such semantic and structured data has been widely made available, often under open licenses. Some of the flagship so-called knowledge base include DBPedia [**?**], Wikidata [**?**] or YAGO [**?**]. While the potential use cases of such knowledge bases are far reaching, their exploitation is still only starting. We believe that such is the case for several reasons, two of which we shortly elaborate on. First, writing queries in a structured query language, allowing to fully exploit the potential of structured data, is

a task that requires some training. Second, while semantic querying may theoretically provide more complete answers, this is hardly the case with current knowledge base, as it is still usually faster to fulfill an information need with classical keyword search on well-known web search engines than through a SPARQL query. The payoff for learning to write structured queries is thus too small to motivate a large number of users to spend the time learning a new way to query knowledge bases.

In this paper, we propose a novel way to exploit structured and semantic querying, providing an added value that is not only completeness of the results, but also an enhanced restitution of them. To illustrate our contribution, let us consider the case of a user willing to know more about sport leagues around the world. A first approach would be to make a keyword query on a web search engine. Current engines typically refer to Wikipedia pages, providing a list of sport leagues, manually curated. An alternative approach would be to issue a query on a knowledge base, say DBPedia. Such a query would get an unordered list of more than 3000 results. This already provides more data than the similar keyword search, but more answers *alone* does not mean that it fulfills in a *better* way the information need of the user: a few thousands answers is already too much to be efficiently grasped by a human user.

To alleviate this problem, several methods have been introduced. First, a list of answers may be returned not in a random order, but in an order that may reflect the user interest, and potentially only return the top-k answers [**?**]. This assumes however a prior knowledge (or estimation) or the user interest, and hides (in practice) a very large part of the results. Another method is based on clustering method [**?**]: using some features, answers are clustered and an element of each cluster is returned to the user. A shortcoming of such methods is that clusters built in such way are not easily described in a human understandable way, making the identification of interesting subparts of the result harder.

In this paper, we propose novel methods to organize the set of answers in such a way that the user get a full overview of the result set, while having sufficiently few results to consider not to be overwhelmed. In the case of sport leagues, we may decide to group the answers by region of the world, by sport, by average revenue of players within the sports leagues, or by any other criterion that the knowledge base and/or the query may allow us to use.

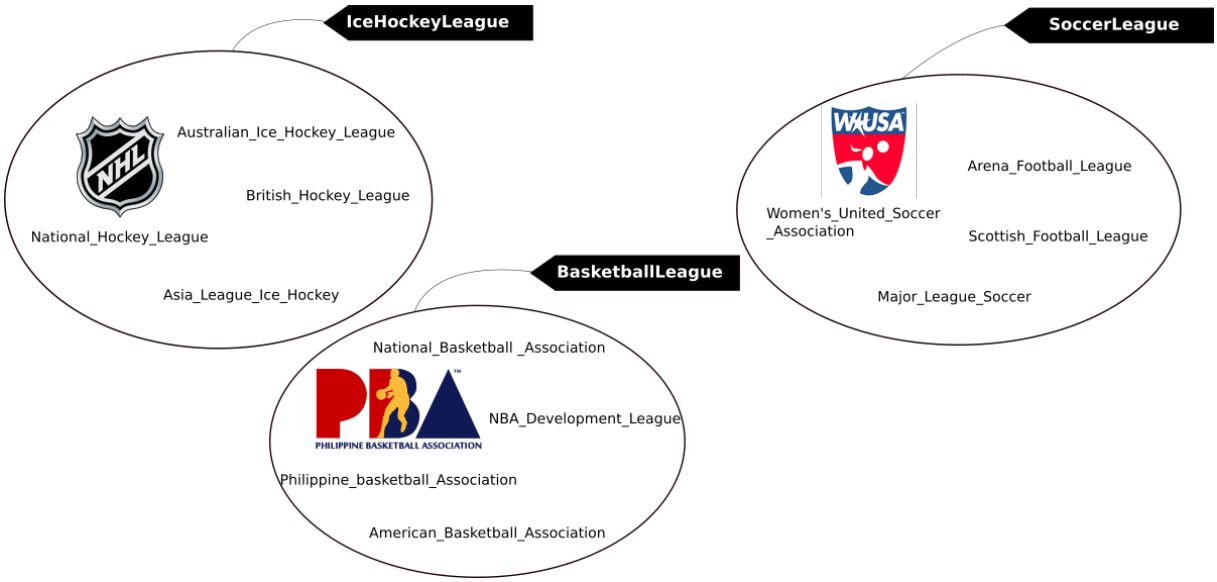

Figure 1: Answers of the query to sports leagues group by type of sport.

In the remaining of this paper, we first recall the necessary background about semantic and structured querying (Section 2), formally state the problem (Section 3), introduce the notion of a rewriting graph which is at the core of our approach and of its own interest (Section 4), define a novel notion of labellisation (Section 5), and experimentally evaluate our approach (Section 6). We conclude by presenting possible future work.

2 Recall of OBQA Problem & Approches

We assume the reader to be familiar with first-order logic, and recall some basic notions that will be used to describe queries and knowledge, re-using the notations from [**?**]. The interested reader can consult [**?**].

2.1 Basic Logical Notations

We consider a **vocabulary** as a pair $\mathscr{V} = (\mathscr{P}, \mathscr{C})$, where $\mathscr{C}$ is a (possibly infinite) set of constants and $\mathscr{P}$ is a finite set of predicates. Each predicate has an arity, which is a natural integer. Unary predicates (with arity of 1) are named *classes* and binary predicates (with arity of 2) are named *properties*. A *term* on $\mathscr{V}$ is either a constant of $\mathscr{C}$ or a variable. An **atom** on $\mathscr{V}$ is of form $p(t_1, t_2, \ldots, t_k)$ where p is a predicate of $\mathscr{P}$ of arity k and each t_i is a term.

We consider the restriction of first-order logic $FOL(\wedge, \exists)$, which contains exactly the formulas built with atoms, the "and" connector $\wedge$ and the existential quantifier $\exists$. We denote the set of variables of a formula F by vars(F). In a formula, a variable which is not in the scope of a quantifier is called a *free* variable. A formula is *closed* if it has no free variable.

Example 2.1. *We consider two properties p and r, a class A and a constant c. With this vocabulary, we can build the following formula:*

$$F = \exists y \exists z, p(x, y) \wedge r(c, z) \wedge A(z).$$

The variable x is the free variable of the formula F.

In the following lines, we will just consider closed formulas. So without loss of generality, we can take all formulas of $FOL(\wedge, \exists)$ in prenex form (all quantifier $\exists$ are at the beginning of the formula) and consider it as a set of atoms. Our approach heavily rely on the notion of *homomorphism*.

Definition 2.2. *Given two closed formulas F and G considered as sets of atoms, a* homomorphism h *from F to G is a substitution from the variables of F to terms of G (and which acts as identity on constants of F) such as for all atoms $p(t_1, t_2, \ldots, t_n) \in F$, it holds that $p(h(t_1), h(t_2), \ldots, h(t_n)) \in G$; in others words $h(F) \subset G$.*

Example 2.3. *If we consider the two closed formulas $F = \{p(x_1, y_1), r(y_1, z_1), A(z_1)\}$ and $G = \{p(x_2, y_2), r(y_2, x_2), A(x_2)\}$, the substitution $h = \{x_1 \mapsto x_2, y_1 \mapsto y_2, z_1 \mapsto x_2\}$ is a homomorphism from F to G.*

An important notion in logic is the notion of entailment (for which we refer the reader to [] MB: need citation), which is classicaly denoted by the symbol $\models$. The following results states the link between entailment in $FOL(\wedge, \exists)$ and homomorphisms.

Theorem 2.4 ([**?**])**.** *Let F and G two formulas of $FOL(\wedge, \exists)$. It holds that $F \models G$ if and only if there exists a homomorphism from G to F.*

2.2 Ontology-Based Query Answering Problem

We call *fact* a closed conjunction of atoms. A *conjunctive query* is a formula of $FOL(\wedge, \exists)$, in case the formula is closed, we called it *Boolean conjunctive query*. Free variables of the query are called the *answer variables*.

Example 2.5. *If we consider the classes $Professor$, $Teacher$ and $Reviewer$, the property $reviews$ and the constant $alice$, we can define the following fact D and queries q_1 and q_2:*

- $D = Professor(alice) \wedge Reviewer(alice)$
- $q_1 = \exists v \exists w Teacher(v) \wedge reviews(v, w)$
- $q_2(x) = Professor(x)$

q_1 is a Boolean conjunctive query and q_2 conjunctive query, which has a single answer variable x.

Given a fact D, an *answer* to a query q with free variables $\{x_1, x_2, \ldots, x_n\}$ is the restriction to $\{x_1, x_2, \ldots, x_n\}$ of a homomorphism h from q to D. In particular, we have $D \models h(q)$.

The behavior in this example is not fully satisfying, as it does not take the additional knowledge that we have on professors and teachers into account: indeed, we know that all professors are actually teachers. We tackle this by introducing existential rules [**?**], which are a formalism to represent formal definitions of domain knowledge, also known *ontologies*. Note that we could have alternatively used Description Logics [**?**].

Definition 2.6 (Existential Rule). *An* existential rule *R is a first-order formula with the following form:*

$$R = \forall \boldsymbol{x} \forall \boldsymbol{y}\ B[\boldsymbol{x}, \boldsymbol{y}] \rightarrow \exists \boldsymbol{z} H[\boldsymbol{y}, \boldsymbol{z}]$$

where $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ are sets of variables and B, H are conjunctions of atoms whose variables belong to respectively $\mathbf{x} \cup \mathbf{y}$ and $\mathbf{y} \cup \mathbf{z}$. We call B the body *of the rule R, denoted* $\text{body}(R)$*, H the* head *of R, denoted* $\text{head}(R)$*. The* frontier *of R, denoted* $\text{fr}(R)$*, is the set of variables $\boldsymbol{y}$ and $\boldsymbol{z}$ are call the existential variables of R.*

Example 2.7 (Existential Rules). *Using the same vocabulary as in Example 2.5, we can define the following existential rules:*

- $R_1 = \forall x\ Reviewer(x) \Rightarrow \exists y\ reviews(x, y)$,
- $R_2 = \forall x\ Professor(x) \Rightarrow Teacher(x)$.

R_1 says that each reviewer reviews something. R_2 says that each professor is also a teacher. We notice that in rule R_1 there are no existential variables.

We call a set of rules $\mathcal{R}$ an ontology and a pair $(D, \mathcal{R})$, where D is a fact and $\mathcal{R}$ is an ontology, a knowledge base. We denoted a knowledge base by $\mathscr{K} = (D, \mathcal{R})$. Ontology-based query answering is the problem of finding the answers of a query on a knowledge base. Given a knowledge base and a (Boolean) query, an important reasoning task is to determine whether the query is entailed by the knowledge base. Several methods have been designed to this purpose. To ease the understanding, we first introduce *saturation*, which intuitively adds atoms that are entailed by the knowledge base to the data. We focus in the next section on an alternative approach called *query rewriting*, which is at the core of our approach.

Definition 2.8 (Rule Application). *A rule R is applicable to a fact D if there exists a homomorphism h from the body of R to D. The* result *of the application of R with respect to h is $\pi(D, R, h^{safe}) = F \cup h^{safe}(\text{head}(R))$, where h^{safe} is a substitution on variables of* $\text{head}(R)$ *that replaces a variable x by $h(x)$ if $x \in \text{fr}(R)$ and by a fresh variable otherwise.*

One can apply a rule on a fact that results from a rule application, hence leading to the notion of *derivation sequence*.

Definition 2.9 (Derivation Sequence). *Let $\mathscr{K} = (D, \mathcal{R})$ be a knowledge base. An $\mathcal{R}$* -derivation sequence *of D is a finite sequence $(D_0 = D), D_1, \ldots, D_n$ such as for $0 \leq i \leq n$, there is $R_i \in \mathcal{R}$ and a homomorphism h_i from* $\text{body}(R_i)$ *to D_i such as $F_{i+1} = \pi(D_i, R_i, h_i)$.*

Example 2.10. *We can apply the rule $R_1 = \forall x\ Reviewer(x) \Rightarrow \exists y\ reviews(x, y)$ to $D = Professor(alice) \wedge Reviewer(alice)$ and get the fact $D_1 = Professor(alice) \wedge Reviewer(alice) \wedge \exists x\ reviews(alice, x)$. We can apply the rule $R_2 = \forall x\ Professor(x) \Rightarrow Teacher(x)$ to D_1 and we get the fact $D_2 = Professor(alice) \wedge Reviewer(alice) \wedge \exists x\ reviews(alice, x) \wedge Teacher(alice)$. Let $\mathcal{R} = \{R_1, R_2\}$, the sequence D, D_1, D_2 is an $\mathcal{R}$-derivation sequence of D. We notice that D_2 satifies q_2, when D does not.*

The following result states that entailment can be characterized through derivation sequences.

Theorem 2.11 ([?]). *Let $\mathscr{K} = (D, \mathcal{R})$ be a knowledge base and q a Boolean conjunctive query, we know that $\mathscr{K} \models q$ if and only if there exists an $\mathcal{R}$ -derivation $(D_0 = D), D_1, \ldots, D_k$ such that $D_k \models q$.*

2.3 Rewriting Approach with Existential Rules

While saturation is a very intuitive approach to OBQA, an alternative approach that we will exploit in our approach is the so-called *query rewriting approach*. Instead of modifying the data in such a way that queries can be evaluated directly over the modified data, the query is modified in such a way that the modified query can be evaluated directly over the original data. We need a few technical tools to define this approach. For the ease of understanding, we only present the rewriting method on Boolean conjunctive queries, but it can be generalised for conjunctive queries.

Definition 2.12 (Separating Variables). *Let q be a Boolean conjunctive query and be q' a subset of q, the set of* separating variables *of q' is denoted by* $\text{sep}_q(q')$ *and is defined as:*

$$\text{sep}_q(q') = \text{vars}(q \setminus q') \cap \text{vars}(q')$$

Definition 2.13 (Piece Unifier). *Let q be a Boolean conjunctive query and R be an existential rule. A* ***piece unifier*** *$\mu = (q', u)$ of q with R is such as $q' \subset q$ and u is a function from* $\text{vars}(q') \cup \text{fr}(R)$ *to* $\text{const}(q') \cup \text{terms}(\text{head}(R))$ *such as:*

- *for all* $y \in \text{fr}(R), u(y) \in \text{fr}(R)$ *or* $u(y)$ *is a constant;*
- *for all* $x \in \text{sep}_q(q'), u(x) \in \text{fr}(R)$ *or* $u(x)$ *is a constant;*
- $u(q') \subset u(\text{head}(R))$.

Using piece unifiers, the notion of direct rewriting of a query can be defined.

Definition 2.14 (Direct Rewriting)**.** *Given* q *a Boolean conjunctive query,* R *an existential rule and a piece unifier* $\mu = (q', u)$ *of* q *with* R*, the* direct rewriting *of* q *with* μ*, denoted* $\beta(q, R, \mu)$ *is* $u(\text{body}(R) \cup (q \setminus q'))$.

Example 2.15 (Direct Rewriting)**.** *If we use the notation from Example 2.5 and 2.7, we can find two direct rewritings of* q_1 *by applying either* R_1 *or* R_2*. Considering* R_1*, we find the following piece unifier* $\mu_1 = (q_1, u_1)$ *where* $q'_1 = \{Teacher(v)\}$*,* $u_1 = \{v \mapsto x\}$ *and the direct rewriting of* q_1 *with* μ *is* $q_3 = \exists x \exists w\ Professor(x) \wedge reviews(x, w)$*. Considering* R_2 *on* q_3*, we find the following piece unifier* $\mu_2 = (q'_3, u_2)$*, where* $q'_3 = \{reviews(x, w)\}$*, this time the separing variables is not the empty set, but contains* x*. The direct rewriting of* q_3 *by* μ_2 *is the query* $q_4 = \exists x \exists y,\ Professor(x) \wedge reviews(x, y)$*. We notice that* D *entails* q_4 *from the Example 2.5 – hence this is an example of how rewriting the query enables to find more results than the original query.*

As in the saturation case, it is possible to apply a rewriting step on a query that has been generated by a first rewriting step. This leads to the notion of rewriting sequence.

Definition 2.16 (Rewriting Sequence)**.** *A sequence of queries* $q_0, q_1, \ldots, q_k$ *is a* $\mathcal{R}$*-rewriting sequence of* q_0 *if for each* $1 \leq i \leq k$*, it exists a rule* $R_i \in \mathcal{R}$ *and a piece of unifier* μ_i *of* q_{i-1} *with* R *such as* $q_i = \beta(q_{i-1}, R_i, \mu_i)$.

Definition 2.17 ($\mathcal{R}$-rewriting)**.** *Let* q *be a Boolean conjunctive query and* $\mathcal{R}$ *be a set of existential rules,* q_k *is a* $\mathcal{R}$-rewriting *of* q *if there exists* $q_0 = q, q_1, \ldots, q_k$ *a* $\mathcal{R}$*-rewriting sequence of* q.

We recall the following theorem, proving that the above notion of query rewriting is adequate.

Theorem 2.18 ([**?**])**.** *Let* D *be a fact,* $\mathcal{R}$ *a set of existential rules and* q *be a Boolean conjunctive query,* $D, \mathcal{R} \models q$ *if and only if there exists* q' *a* $\mathcal{R}$*-rewriting of* q *such that* $D \models q'$.

In our approach, we will often exploit the following corollary of Theorem 2.18.

Corollary 2.19. *Let* q *be a Boolean conjunctive query,* $\mathcal{R}$ *a set of existential rules and* q' *a* $\mathcal{R}$*-rewriting of* q*, then* $q', \mathcal{R} \models q$.

Proof. We knows that q' is a $\mathcal{R}$-rewriting of q such as $q' \models q'$, using Theorem 2.18 and since q' can be seen as a fact, we know that $q', \mathcal{R} \models q$. □

Let us notice that the set of rewritings of a query with respect to a set of rules $\mathcal{R}$ is not necessarily finite, as witnessed by the following example.

Example 2.20. *Let us consider the query* $q = r(a, b)$ *and the rule* $\forall x\ \forall y\ r(x, y) \wedge r(y, z) \rightarrow r(x, z)$*. Any query of the shape* $r(a, x_1) \wedge r(x_1, x_2) \wedge \ldots \wedge r(x_n, b)$*, for any* n*, is a rewriting of* q*. There exists thus an infinity of (non-equivalent)* $\mathcal{R}$*-rewriting of* q.

Of special interest to our approach are sets of rules whose set of rewritings admits a *finite* covering. In other words, there exists a finite set of query $\mathcal{Q}$ such that for any fact D, it holds that $D, \mathcal{R} \models q$ if and only if there exists $q' \in \mathcal{Q}$ such that $D \models q'$. Sets of rules ensuring that there exists such $\mathcal{Q}$ are called *finite unification sets* [**?**], and prominent exemple of such sets of rules are aGRD, linear [**?**] or sticky rules [**?**].

3 Problem Statement

Given a query q and a knowledge base $(F, \mathcal{R})$, we want to find interesting semantically defined subsets of the answers of q. We formalize this through the notion of *labellisation* of a query. For the sake of generality, we define this notion for first-order queries and not only formulas of $FOL(\wedge, \exists)$.

Definition 3.1. *A* labellisation $\mathscr{L}$ *of a query* q *with respect to a set of rules* $\mathcal{R}$ *is a set of queries such as each* $q' \in \mathscr{L}$ *semantically implies* q*, formally:*

$$q', \mathcal{R} \models q$$

We expect that a labellisation of a query has at least two properties: covering and simplicity. Covering states that any answer of the original query is an answer of some query of the labellisation. Simplicity states that the labellisation does not contain redundant queries.

Definition 3.2 (Covering). *A labellisation $\mathscr{L}$ of a query q with respect to a set of rules $\mathcal{R}$ is called* covering *if :*

$$q, \mathcal{R} \models \bigvee_{q' \in \mathscr{L}} q'.$$

Definition 3.3 (Simplicity). *A labellisation $\mathscr{L}$ of a query q with respect to a set of rules $\mathcal{R}$ is called* simple *if :*

$$\forall q_1, q_2 \in \mathscr{L}, q_1, \mathcal{R} \not\models q_2.$$

Let us notice that this notion of labellisation defines a set acceptable objects. However, there exists trivial labellisations, such as $\{q\}$. Our goal is to find more interesting ones – and we will especially consider the cardinality of answers of the queries appearing in the labellisation.

4 Rewritings Graphs

As we will from now on exclusively consider conjunctive queries, we simply the terminology and use the term "query" for conjunctive queries. We introduce the notion of a *rewriting graph*, which is at the core of the method presented in the next section, which returns a labellisation. The main idea behind the rewriting graph is to have a graph such as its vertices are the rewritings of the original query q and the edges contain semantic structure (entailment) relations between rewriting queries. We choose to find a labellisation in the set of rewriting queries of q, because they naturally are more precise queries than q by Proposition 2.19. We also use the way rewriting queries are build using rules which provide their semantic structure. The first Subsection 4.1 defines the general notion of the rewriting graph for existential rules. After a quick recall of RDFS and of a simpler rewriting procedure for that ontology language, Subsection 4.3 defines *product graphs* which can be viewed as an approximation of the rewriting graph for RDFS rules.

4.1 Definition & Properties

In this part, we define a graph of rewritings of the initial query q which contains in its edges information about entailments which exist between rewritings. In the first part, we define theoretically this graph and some of this properties, in the second part, we define the same graph using a more constructive approach.

4.1.1 Definition of the Rewriting Graph

In order to get a semantic definition of subset of answers, we define the set of all the rewritings of a query.

Notation 1. *Let q be a query and $\mathcal{R}$ a set of existential rules, we denote the set of $\mathcal{R}$-rewritings of q by $\mathcal{N}_{q,\mathcal{R}}$. Whenever there is no ambiguity, we will abusively simplify this notation in $\mathcal{N}_q$.*

As it is convenient to abstract away from the syntax of rewritings, we will consider classes of queries up to semantic equivalence, as formalized in the next definition.

Definition 4.1 (Class of Semantic Equivalence). *Let $\mathcal{R}$ be a set of existential rules and S a set of q, the set of* classes of semantic equivalence *of S with is:*

$$\{\tilde{q}_{\mathcal{R},S} / q \in S\},$$

where $\tilde{q}_{\mathcal{R},S} = \{q' \in S / q', \mathcal{R} \models q \text{ and } q', \mathcal{R} \models q\}$.

Whenever there is no ambiguity, we will denote $\tilde{q}_{\mathcal{R},S}$ by $\tilde{q}$. We define a graph that contains all rewritings of a query with the respect of a set of existential rules and also all implication links between rewritings.

Definition 4.2 (Saturated Graph). *The* saturated graph *of $\mathcal{R}$-rewriting classes of q is the graph $G_q = (V, E)$ defined by:*

- *V is the set of classes of semantic equivalence of $\mathcal{N}_q$;*
- *E is the set of edges $(\tilde{q_1}, \tilde{q_2})$ such as $\tilde{q_1} \neq \tilde{q_2}$ and the class $\tilde{q_2}$ semantically implies the class $\tilde{q_1}$, which means there exists $q_1 \in \tilde{q_1}$ and $q_2 \in \tilde{q_2}$ such as $q_2, \mathcal{R} \models q_1$.*

Proposition 4.3. *Let q be a query and $\mathcal{R}$ be a set of existential rules, the saturated graph G of $\mathcal{R}$-rewriting classes of q is an acyclic graph.*

Proof. Suppose there is a cycle $\tilde{q_0}, \tilde{q_2}, \ldots, \tilde{q_k}$ in G, we know that for $1 \leq i \leq k$, $q_{i-1}, \mathcal{R} \models q_i$ and $\tilde{q_0} = \tilde{q_k}$, So finally for $1 \leq i, j \leq k, \tilde{q_i} = \tilde{q_j}$ and there is not cycle. □

Let us notice that the saturated graph of $\mathcal{R}$-rewriting classes of q may be infinite, but is finite whenever $\mathcal{R}$ is a finite unification set. In the remaining of this paper, we assume $\mathcal{R}$ to be a finite unification set. We can now define the rewriting graph of a query q, which is the version of the graph we use to extract from a labellisation. Note that this notion is well-defined, as the transitive reduction of an acyclic graph is uniquely defined. We define also this graph as a transitive reduction, because we want that all oriented paths in this graph are as detailed as possible. In other words, by definition of the transitive reduction, if we have the path $\tilde{q_0}, \tilde{q_1}, \ldots, \tilde{q_k}$ in the rewriting graph, we know that for all $1 \leq i \leq k$, there is no rewriting q'_i of q such as $q'_i \notin \tilde{q_{i-1}} \cap \tilde{q_i}$ and such as $q_{i-1}, \mathcal{R} \models q'_i$ and $q'_i, \mathcal{R} \models q_i$.

Definition 4.4 (Rewriting Graph)**.** *The* rewriting graph *of a query q with respect to a set of existential rules $\mathcal{R}$ is the transitive reduction of the saturated graph of $\mathcal{R}$-rewriting classes of q.*

4.1.2 Construction of the Rewriting Graph

In this subsection, we prove that Algorithm 1 builds the rewriting graph of a query q with respect to a set of existential rules $\mathcal{R}$. Algorithm 1 works as follows: Lines 4 to 14, the graph of rewriting process is built. Lines 15 to 18, all homomorphisms links are added to the edges of the graph. Strongly connected components are then merged (Line 21, by MergeStronglyConnectedComponents(G)) and the transitive reduction is computed by TransitiveReduction(G) and output. We do not provide more details about the done by TransitiveReduction and MergeStronglyConnectedComponents, as these operations are rather classical.

Input: A conjunctive query q, a finite unification set $\mathcal{R}$
Output: The rewriting graph of q w.r.t. $\mathcal{R}$

```
1  V = {q}
2  E = ∅
3  N = {q}
4  while N ≠ ∅ do
5  |  Pick q' ∈ N
6  |  for any direct R-rewriting q'' of q' do
7  |  |  if there is no q̂ ∈ V s.t. q̂ ⊨ q'' then
8  |  |  |  V = V ∪ {q''}
9  |  |  |  E = E ∪ {(q', q'')}
10 |  |  |  N = N ∪ {q''}
11 |  |  end
12 |  end
13 |  N = N \ {q'}
14 end
15 for (q, q') ∈ V × V do
16 |  if q ⊨ q' then
17 |  |  E = E ∪ {(q', q)}
18 |  end
19 end
20 G = (V, E)
21 G = MergeStronglyConnectedComponents(G)
22 G = TransitiveReduction(G)
23 return G
```

Algorithm 1: ComputeRewritingGraph($q, \mathcal{R}$)

First we use the process of rewriting to build a graph of queries that contains the rewritings of a initial query.

Definition 4.5 (Graph of Rewriting Process)**.** *The* graph of rewriting process *of a query q with respect to a set of existential rules $\mathcal{R}$ is the graph $G = (V, E)$ where $V = \cup_{k=1}^{+\infty} V_k$ and $E = \cup_{k=1}^{+\infty} E_k$ such as $V_1 = \{q\}$, $E_1 = \{\}$ and for strict positive integer k sets V_{k+1} and E_{k+1} are defined by(the function β is defined in Definition 2.14):*

- $V_{k+1} = V_k \cup \{\beta(q_1, R, \mu) | \exists q_1 \in V_k, \exists R \in \mathcal{R}, \mu \text{ is a piece of unifier of } q_1 \text{ with } R\}$,
- $E_{k+1} = E_k \cup \{(q_1, \beta(q_1, R, \mu)) | \exists q_1 \in V_k, \exists R \in \mathcal{R}, \mu \text{ is a piece of unifier of } q_1 \text{ with } R\}$.

Proposition 4.6. *The vertices of the* graph of rewriting process *of a query q with respect to a set of existential rules $\mathcal{R}$ is the set of rewriting of q with the respect of $\mathcal{R}$.*

Proof. We prove by induction that for all strict positive integer k the graph $G_k = (V_k, E_k)$ where V_k and E_k are defined as in Definition 4.5, V_k is the set of queries of the $\mathcal{R}$-rewriting sequences of q of length at most k.

The initial case where $k = 1$ is trivial.

We suppose that the property is true for a strict positive integer k. Let q' be such as it exists an $\mathcal{R}$-rewriting sequence of q of length at most $k+1$ ended by q'. If the length is $k+1$, then we will prove that $q' \in V_{k+1}$, else it is trivial. The subsequence from q to q_1 the predecessor of q' is of length k and is also an $\mathcal{R}$-rewriting sequence of q, so q_1 is in V_k. By definition of a rewriting sequence, there exists $R \in \mathcal{R}$ and μ a piece of unifier of q_1 with R such as q' is the rewriting of q_1 by μ, in other words $q' = \beta(q_1, R, \mu)$. Hence $q' \in V_{k+1}$.

Let $q' \in V_{k+1} \setminus V_k$, it exists $q_1 \in V_k$, $R \in \mathcal{R}$ and μ a piece of unifier such of q_1 with R such as $q' = \beta(q_1, R, \mu)$. Since $q_1 \in V_k$, there exists by hypothesis an $\mathcal{R}$-rewriting sequence of q ended by q_1, so adding q' to the end of this sequence give us a new $\mathcal{R}$-rewriting sequence of q while last element is q', which concludes the proof.

□

Example 4.7. *Suppose that we define A and B as classes and r and s as properties. With this vocabulary, we consider the rules $R_1 = \forall x\ A(x) \Rightarrow B(x)$ and $R_2 = \forall x\ r(x, y) \Rightarrow s(x, y)$. With these rules, we can build the process rewriting graph of the query $q(x, y) = A(x) \wedge B(x) \wedge s(x, y)$ represented by Figure 2.*

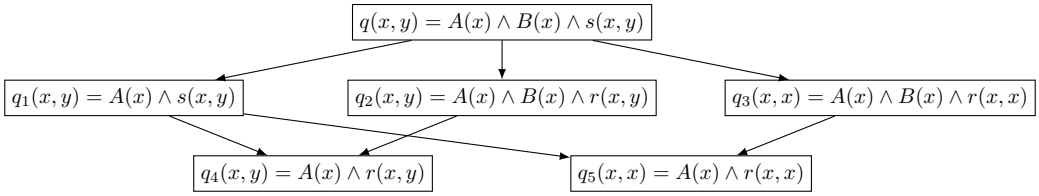

Figure 2: The process rewriting graph of the query $q(x, y) = A(x) \wedge B(x) \wedge s(x, y)$.

The next step is to saturate the graph of rewriting process with all existing homomorphism.

Definition 4.8 (Homomorphical Saturation)**.** *The* homomorphical saturation *of a graph $G = (V, E)$ whose vertices are queries, is the graph $G' = (V', E')$ defined by:*

- $V' = V$
- $E' = E \cup \{(q_1, q_2) \in V^2 | q_1 \neq q_2$ *and there exists a homomorphism from q_1 to q_2}*

Example 4.9. *Figure 3 represents the graph of classes of the homomorphical saturation of rewriting process graph of $q(x, y) = A(x) \wedge B(x) \wedge s(x, y)$.*

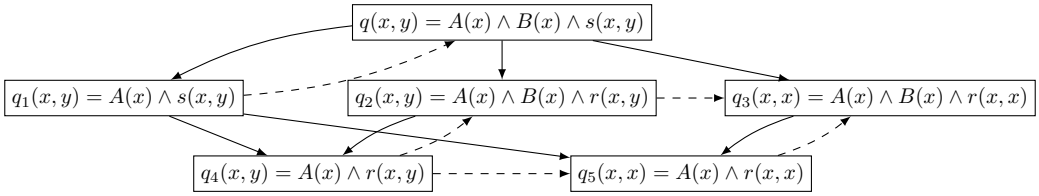

Figure 3: The homomorphical saturation of process rewriting graph of the query $q(x, y) = A(x) \wedge B(x) \wedge s(x, y)$. Here, homorphisms are represented by dashed edges.

The important property of this saturation is that is allows to directly read on the obtained graph entailment (with respect to $\mathcal{R}$) through reachability. This is the topic of Lemma 4.10.

Lemma 4.10. *Let be $G = (V, E)$ the homomorphical saturation of the graph of rewriting process of a conjunctive query q w.r.t. a set of existential rules $\mathcal{R}$ and let q_1, q_2 two queries in V, if $q_2, \mathcal{R} \models q_1$, then there exists an oriented path in G from q_1 to q_2.*

Proof. We know that $q_2, \mathcal{R} \models q_1$, so using the Theorem 2.18 there exists $q_1' \in V$ an $\mathcal{R}$-rewriting of q_1 such that $q_2 \models q_1'$. By definition since q_1' is an $\mathcal{R}$-rewriting of q_1, there exists an $\mathcal{R}$-rewriting sequence $q_1^0 = q_1, q_1^1, \ldots, q_1^{k-1}, q_1^k = q_1'$ and this sequence is an oriented path from q_1 to q_1' in the graph of rewriting process of q. Moreover, because $q_2 \models q_1'$, there exists an edge from q_1' to q_2 in G, since G is homomorphically saturated. This proves that there is a path from q_1 to q_2 in G. □

We now want to simplify as much as possible the obtained graph. Especially, we do not want two distinct queries of the graph to be semantically equivalent.

Definition 4.11 (Graph of Classes)**.** *Let $G = (V, E)$ be a graph whose vertices are a subset of rewritings of the query q, $\tilde{G} = (\tilde{V}, \tilde{E})$ the* graph of classes *of G is defined by :*

- *$\tilde{V}$ is the set of semantic equivalence classes of V;*
- *$\tilde{E} = \{(\tilde{q_1}, \tilde{q_2}) \in \tilde{V}^2 / \exists (q_1, q_2) \in E^2, q_1 \in \tilde{q_1}, q_2 \in \tilde{q_2}\}$*

Computing the graph of classes on the homomorphical saturation amouns to merge strongly connected components, as state by the following proposition.

Proposition 4.12. *Let be $G = (V, E)$ the homorphical saturation of the graph of rewriting process of a conjunctive query q w.r.t. a set of existential rules $\mathcal{R}$, the semantic equivalence class of a query $q_1 \in V$ is the strong connected component of q_1 in the graph G.*

Proof. We have to prove the equality between S_{q_1} the strong connected component of a query q_1 and $\tilde{q_1}$ the semantic equivalence class of q_1 in V w.r.t. $\mathcal{R}$.

Let be $q_2 \in S_{q_1}$, so there is an oriented path from q_1 to q_2 and an oriented path from q_2 to q_1 in G. By definition of edges of G and using the Corollary 2.19, we know respectively that $q_2, \mathcal{R} \models q_1$ and $q_1, \mathcal{R} \models q_2$, so $q_2 \in \tilde{q_1}$.

Let be $q_2 \in \tilde{q_1}$, so using two times Lemma 4.10 we know that there is an oriented path from q_1 to q_2 and the inverse path, therefore $q_2 \in S_{q_1}$. □

Example 4.13. *Using the preceding Property 4.12, we find the graph of classes of the homomorphical saturated graph from Example 4.9 reprented in Figure 4.*

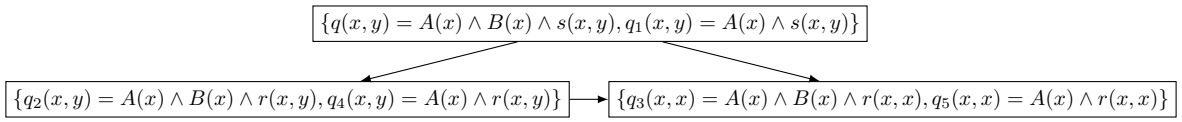

Figure 4: The graph of classes of the homomorphical saturation of rewriting process graph of $q(x, y) = A(x) \wedge B(x) \wedge s(x, y)$

Last, we show that the saturated graph of $\mathcal{R}$-rewriting of q can be obtained by taking the transitive reduction of the graph constructed so far. For technical simplicity, we first show that their transitive saturation are equal.

Proposition 4.14. *The transitive saturation of graph of classes of the homomorphical saturation of graph of rewriting process of a query q and of a set of existential rules $\mathcal{R}$ is the saturated graph of $\mathcal{R}$ -rewriting classes of q.*

Proof. Let q a query and $\mathcal{R}$ a set of existential rules. We denote the transitive saturation of graph of classes of the homomorphical saturation of graph of rewriting process of the query q and a set of existential rules $\mathcal{R}$ by $G_P = (V_P, E_P)$ and we denote the saturated graph of $\mathcal{R}$-rewriting classes of q by $G_S = (V_S, E_S)$.

We only need to prove that sets of edges E_P and E_S are equals, because by the Proposition 4.6 that V_P and V_S are equals to the semantic classes of the set of classes of the rewritings of q with the respect of $\mathcal{R}$.

First, we prove that $E_P \subset E_S$, indeed if we have $(C_1, C_2) \in E_P$ then there exists an oriented path in G_P from C_1 to C_2 such as for each consecutive vertices C_a, C_b of this path, there exists $q_a \in C_a$ and $q_b \in C_b$ such that either:

1. q_b is a $\mathcal{R}$ -rewriting of q_a;
2. or there exists a homomorphism from q_a to q_b.

In the two cases, we know that $q_b, \mathcal{R} \models q_a$, so by definition of G_S, we know that $(C_a, C_b) \in E_S$. So the same path from C_1 to C_2 also exists in G_S and prove that $C_1 \models C_2$, then $(C_1, C_2) \in E_S$.

We want to prove the inclusion in the other way. Let $(C_1, C_2) \in E_S$, then there exists $q_1 \in C_1$ and $q_2 \in C_2$ such as $q_2, \mathcal{R} \models q_1$, so using Lemma 4.10, there is an oriented path from q_1 to q_2 in the homomorphical saturation of the rewriting process graph of q. So by definition of edges of G_P, there is also an oriented path from C_1 to C_2 in G_P. Finally, G_P is a transitive saturated graph, hence $(C_1, C_2) \in E_P$, so we prove that $E_S \subset E_P$. □

Corollary 4.15. *The transitive reduction of graph of classes of the homomorphical saturation of graph of rewriting process of a query q and of a set of existential rules $\mathcal{R}$ is the rewriting graph of $\mathcal{R}$ -rewriting classes of q.*

Example 4.16. *We conclude the first part of this running example by presenting the transitive reduction of Figure 4 in Figure 5. One edge has been deleted.*

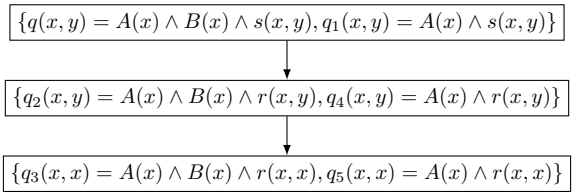

Figure 5: The rewriting graph of $q(x,y) = A(x) \wedge B(x) \wedge s(x,y)$ w.r.t. rules $R_1 = \forall x, A(x) \Rightarrow B(x)$ and $R_2 = \forall x, r(x,y) \Rightarrow s(x,y)$.

4.2 RDFS rewriting

As the case of RDFS ontologies is especially important from a practical point of view, we introduce a method that is specifically tailored towards knowledge bases having this ontology. After a quick recall of what kind of rules are allowed within an RDFS ontology, we introduce an alterative rewriting mechanism that will be used to speed up the computation.

RDFS is a W3C standard [**?**] that provides a data-modelling vocabulary for RDF. More specifically, it provides axioms to describe subclass, subproperty, domain and range restrictions. Expressed in a logical way, ontologies expressible in RDFS contains rules of the shape described in Table 1.

Axiom type	Logical Formula
Subclass	$p(x) \rightarrow q(x)$
Subproperty	$r(x,y) \rightarrow s(x,y)$
Domain Restriction	$r(x,y) \rightarrow p(x)$
Range Restriction	$r(x,y) \rightarrow p(y)$

Table 1: Axioms appearing in RDFS ontologies

The specificity of RDFS rules allows for a more simple rewriting algorithm, such as the one presented in [**?**]. A similar rewriting method could be used as long as rules do not contain existential variables, and no variable is repeated within the head. Under these conditions, the query can be rewritten atom by atom. The algorithm for RDFS performs as well a breadth-first traversal of rewritings, but considers only unifiers $\mu = (q', u)$ such that q' is a single atom and where u is injective. Moreover, one can choose the names of variables introduced by the rewriting in such a way that if μ_1 is applied on some atom, then μ_2 on another atom, then the rewritings generated by this sequence of rewritings is *syntactically* the same as the one generated when rewriting first through μ_2 then μ_1. This can be done by assigning to each atom an integer i, and whenever an atom is rewritten, the generated atom is assigned the same integer[1] and if a fresh variable is introduced in that atom, one can name it FV_i, where $\{FV_i\}_{i \in \mathbb{N}}$ is a set of variables disjoint from the query variables.

Example 4.17. *Note that the above rewriting algorithm, despite being classically equivalent to the general rewriting algorithm, does not generate syntactically equals rewriting sets. For instance, the query $q(x,y) = s(x,y)$, which can be rewritten by the rule $r(x,y) \Rightarrow s(x,y)$, leads to different queries from a rewriting system to another:*

[1]Note that this may require to consider queries as multisets of atoms.

- *RDFS rewriting system allows one rewriting* $q'(x,y) = r(x,y)$,
- *existential rewriting system allows two rewriting* $q_1(x,y) = r(x,y)$ *and* $q_2(x,x) = r(x,x)$.

4.3 Product of Graph

The algorithm presented in the previous section can also be seen as follows: one compute rewritings atom by atom and then perform a cartesian product to get a rewriting of the original query. We exploit this fact in order to compute a graph that we will use as the rewriting graph of q with respect to $\mathcal{R}$. In this section, we explain Algorithm 2: we first define the output of the subroutine ComputeProductGraph (Definition 4.20), and the relationships that exist between the output of Algorithm 2 and the rewriting graph of q with respect to $\mathcal{R}$.

Input: A conjunctive query q, a finite unification set $\mathcal{R}$
Output: The rewriting graph of q w.r.t. $\mathcal{R}$
for *all atom* $a_i \in q$ **do**
  $G_i = \text{ComputeRewritingGraph}(a_i, \mathcal{R})$
end
return $\Pi_i(G_i)$

Algorithm 2: ComputeProductGraph$(q, \mathcal{R})$

Let us first introduce a generic notion of product graph.

Definition 4.18 (Product Graph)**.** *Let* $G_1, G_2, \ldots, G_n$ *be graphs with* V_i *is the set of vertices of* G_i *and* E_i *the set of edges of* G_i, $G = (V, E)$ *the* ***product graph*** *of* $G_1, G_2, \ldots, G_n$ *is defined by :*

- $V = \prod_{i=1}^{n} V_i$;
- $E = \{((t_1, t_2, \ldots, t_n), (u_1, u_2, \ldots, u_n)) / \exists 1 \leq k \leq n, (t_k, u_k) \in E_k \text{ and } \forall i \neq k, t_i = u_i\}$

We directly apply this notion to the rewriting graph of atomic queries to define the product graph of a query.

Definition 4.19 (Product Graph of a Query)**.** *Let* q *a RDFS query defined by the atoms :* $a_1, a_2, \ldots, a_n$ *and* $G_1, G_2, \ldots, G_n$ *be graphs such as* G_i *is the rewriting graphs of* a_i*. We define the* ***product graph*** *of* q *as the product graph of* $G_1, G_2, \ldots, G_n$.

With each node of the product graph of a query, we naturally associate a query.

Definition 4.20. *Let* $G = (V, E)$ *a product graph of a query* q *and* $(a_1, a_2, \ldots, a_n) \in V$ *a vertex of* G, *we define a* query interpretation *of this vertex defined as the query composed by* $\{a_1, a_2, \ldots, a_n\}$ *and the answers variables of* q.

Example 4.21. *Suppose that we define* A *and* B *as classes and* r *and* s *as properties. With this vocabulary, we consider the rules* $R_1 = \forall x\ A(x) \Rightarrow B(x)$ *and* $R_2 = \forall x\ r(x,y) \Rightarrow s(x,y)$*. With these rules, we want to build the product graph of the query* $q(x,y) = A(x) \wedge B(x) \wedge s(x,y)$*. We first build the three rewriting graphs of the atoms of* q*, as shown in Figures 6, 7 and 8.*

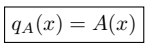

Figure 6: Rewriting graph of the atomic query $q_A(x) = A(x)$

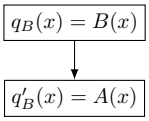

Figure 7: Rewriting graph of the atomic query $q_B(x) = B(x)$

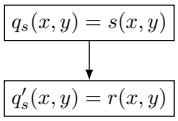

Figure 8: Rewriting graph of the atomic query $q_s(x,y) = s(x,y)$

Using these rewriting graphs, we build the product graph of q *and keep the repetition of atoms in vertices labels for a better understanding, in Figure 9. We notice that* q *and* q_1 *are equivalentes with respect to the rules as* q_2 *and* q_4.

We expect some properties of the product graph of rewriting graph of queries.

Proposition 4.22. *If* $G_1, G_2, \ldots, G_n$ *are acyclic, transitively rewriting graphs, then the product graph* G *of* $G_1, G_2, \ldots, G_n$ *is also acyclic and transitively reduced.*

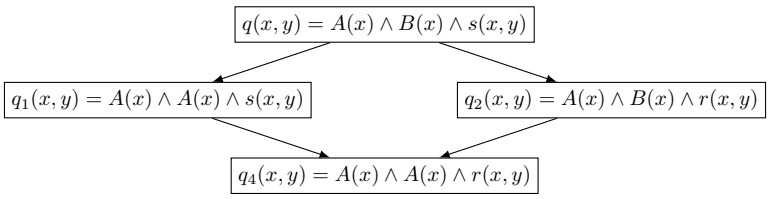

Figure 9: The product graph of the query $q(x,y) = A(x) \wedge B(x) \wedge s(x,y)$.

Proof. First, let $v^1, v^2, \ldots, v^l$ be an oriented path in G, we denote each $v^i = (v^i_1, v^i_2, \ldots, v^i_n)$ for $1 \leq i \leq l$. We define a *projection of this path* on G_k for $1 \leq k \leq n$ as the oriented path in G_k defined by the sequence of edges of G_k which is the maximum sub-sequence of $(v^1_k, v^2_k), (v^2_k, v^3_k), \ldots, (v^{n-1}_k, v^n_k)$ containing only edges of G_k. In other words, we keep from the sequence $(v^1_k, v^2_k), (v^2_k, v^3_k), \ldots, (v^{n-1}_k, v^n_k)$ only couples (v^i_k, v^{i+1}_k) such as $v^i_k \neq v^{i+1}_k$. We notice that the projection of the path $v_1, v_2, \ldots, v_l$ on G_k is an empty path if and only if for all $1 \leq i, j \leq l$ we have $v^i_k = v^j_k$.

We want to prove that the product graph G is acyclic. Suppose that $v^1, v^2, \ldots, v^l$ is a cycle of G. The projection of this path on G_k for $1 \leq k \leq n$ is not empty if and only if it is a cycle in G_k. Since for all $1 \leq k \leq n$, there is no cycle in each G_k, we know that for all $1 \leq i, j \leq l$, we have $v^i_k = v^j_k$. So $v^1, v^2, \ldots, v^l$ is a constant sequence, but G does not contain loop by definition of these edges, absurd.

We want to prove that G is transitively reduced. We suppose the contrary by assuming the existence of an oriented path $v^1, v^2, \ldots, v^l$ in G containing at least two edges such as (v^1, v^l) is an edge of G. We use the following notations $v^1 = (v^1_1, v^1_2, \ldots, v^1_n)$ and $v^l = (v^l_1, v^l_2, \ldots, v^l_n)$. By definition of the edges of G, there exists $1 \leq k \leq n$ such as (v^1_k, v^l_k) is an edge of G_k and $\forall 1 \leq h \leq n$ and $i \neq k, v^1_h = v^l_h$. With this last property, we know that $\forall 1 \leq h \leq n$ and $h \neq k$, the projection of the path $v^1, v^2, \ldots, v^l$ on G_h is empty, because otherwise there is a cycle in G_h. So for all $1 \leq i, j \leq l$ we have $v^i_h = v^j_h$. Since the path $v^1, v^2, \ldots, v^l$ contains at least two edges, the projection of this path has to contain also at least two edges. But this projection is an oriented path in G_k from v^1_k to v^l_k and G_k contains the edge (v^1_k, v^l_k), then G_k is not transitively reduced, absurd. □

Corollary 4.23. *The product graph of a query with respect to an RDFS ontology is acyclic and transitive reduced.*

Proof. It follows directly from the definition of the product graph of a query. □

Let us notice that the product graph thus generated is *not* always equal to the rewriting graph of $\mathcal{R}$-rewriting classes of q, as witnessed by the Figure 10, which queries q and q_1 are equivalente.

We conjecture the following proposition, which intuitively states the graph output by our optimized algorithm for RDFS is "not too far" from being the $\mathcal{R}$-rewriting graph of queries of the original query.

Conjecture 4.24. *Let $G_1, G_2, \ldots, G_n$ be $\mathcal{R}$-rewriting graph of queries of one atom and let G be the product graph of $G_1, G_2, \ldots, G_n$. If there is t, u two vertices of G such as $t, \mathcal{R} \models u$, then it exists an oriented path in the graphs of classes of G from the class of u to the class of t.*

Example 4.25. *A simple way to prove way to prove the Conjecture 4.24 could have to prove the following proposition. We could expect that in the product graph when there is a homomorphism from a query q_1 to another q_2, then there exists a homomorphism from q_2 to q_1. Unfortunately, this proposition is false and Figure 10 is a counter-example:*

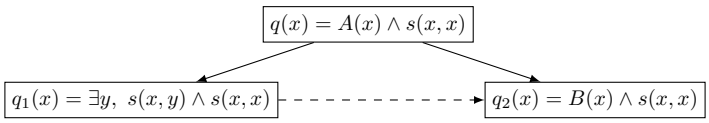

Figure 10: The product graph of the query $q(x,y) = A(x) \wedge s(x,x)$ w.r.t. rules $R_1 = \forall x, y\ r(x,y) \Rightarrow A(x)$ and $R_2 = \forall x,\ B(x) \Rightarrow A(x)$. Dashed arrow represents an homomorphism from q_1 to q_2

In this example, there is a homomorphism from q_1 to q_2, but there is not homomorphism in the other direction.

While the conjecture is still open, it is known to be valid in specific cases that are practically relevant. Actually, a stronger statement is true, as shown in Proposition 4.28.

Definition 4.26 (Folding Free Query)**.** *A query q is* folding free *if for any two rewritings q_1 and q_2 of q (according to some rewriting mechanism), if q_1 and q_2 are equivalent then $q_1 = q_2$.*

Proposition 4.27. *Let $\mathcal{R}$ be an RDFS ontology, q be a query such that for any pair of atoms (a_1, a_2), the set of predicates appearing in the rewritings of a_1 is disjoint from the set of predicates appearing in rewritings of a_2. Then q is a folding free query for the RDFS rewriting mechanism.*

Proposition 4.28. *The product graph of a folding free query q is the rewriting graph of $\mathcal{R}$-rewriting classes of q.*

5 Labellisation

We want to use the structure of the rewriting graph of a query q to extract from its rewritings, a interesting labellisation of q.

First we want to extend by *set graph* definition, the rewriting graph definition of a query q such as each answer of q is contained in at least one leaf of the graph. We also replace semantic classes of equivalence from the vertices by one representative picked from a given function *Repr* which chooses a reprensentative from a class and such as if q is the original query, we have $\text{Repr}(\tilde{q}) = q$.

Definition 5.1 (Set Graph)**.** *Let $G_r = (V_r, E_r)$ the rewriting graph of a conjunctive query q with respect to a set of existential rules $\mathcal{R}$, the* set graph *of q with respect of $\mathcal{R}$ is the graph $G_S = (V_S, E_S)$ defined by the following equalities, where P_r is the set of vertices of G_r which have at least one child:*

- $V_S = \{\text{Repr}(C) |\ C \in V_r\} \cup \{\textit{selfLeaf}(C) |\ C \in P_r\}$
- $E_S = \{(\text{Repr}(C_1), \text{Repr}(C_2)) |\ (C_1, C_2) \in E_r\} \cup \{(\text{Repr}(C), \textit{selfLeaf}(C)) |\ C \in P_r\}$.

where for a vertex C of G_r, we define selfLeaf(C) by:

$$\textit{selfLeaf}(C) = \text{Repr}(C) \wedge \bigwedge_{C_i \text{ child of } C} \neg \text{Repr}(C_i).$$

Proposition 5.2. *Let $G = (V, E)$ be a set graph of a conjunctive query q w.r.t. a set of existential rules $\mathcal{R}$ and q_1, q_2 two queries in V. We know that $q_2, \mathcal{R} \models q_1$ if and only if there exists an oriented path from q_1 to q_2 in the set graph G.*

Proof. Actually, the proposition is trivial when q_1 and q_2 are not selfLeaves of the set graph G. We just need to use Definition 4.2 of saturated graph and check the correctness of the proposition after each transformation of graph.

Otherwise, if there exists an oriented path from q_1 to q_2 in G, then q_2 is a selfLeaf and have a unique parent q_2'. By definition of selfLeaf, $q_2 \models q_2'$, so by using the proposition for the restricted path from q_1 to q_2', we get the result. MB: to be continued □

We use the set graph to reduce the problem of finding a covering labellisation to the problem of finding a set of nodes such as each leaf has an ancestor or is itself into this set.

Proposition 5.3. *Let $G = (V, E)$ be the set graph of a conjunctive query q with respect to a set of rules $\mathcal{R}$. If F is a subset of vertices of G such as for each l leaves of G it exists oriented path in G from a element of F to l, then we have the following equivalence:*

$$q, \mathcal{R} \models \bigvee_{q_f \in F} q_f$$

Proof. We denote the set of leaves of G by L_G. We have the following equations where we use Property 2.19 to go from the first to the second line and the last is obtained by applying the reasoning of the first two lines to each child of q in G.

$$\begin{aligned} q &\equiv_{\mathcal{R}} \bigvee_{(q,q') \in E} (q \wedge q') \vee \left(q \wedge \bigwedge_{(q,q') \in E} \neg q' \right) \\ &\equiv_{\mathcal{R}} \bigvee_{(q,q') \in E} q' \vee \text{selfLeaf}(q) \\ \dots & \\ &\equiv_{\mathcal{R}} \bigvee_{q_l \in L_G} q_l \vee \bigvee_{q_p \in V \setminus L_G} \text{selfLeaf}(q_p) \end{aligned}$$

And because L_G already contains the self leaves, we have $q \equiv_{\mathcal{R}} \bigvee_{q_l \in L_G} q_l$.

Now, for all $q_l \in L_G$, there exists $q_f \in F$ such as there is an oriented path from q_f to q_l. Two consecutive queries q_1, q_2 of this path are such as $q_2, \mathcal{R} \models q_1$, by definition of set graph. Finally, we know that $q_l, \mathcal{R} \models q_f$, so:

$$\begin{aligned} \bigvee_{q_l \in L_G} q_l, \mathcal{R} &\models \bigvee_{q_f \in F} q_f \\ q, \mathcal{R} &\models \bigvee_{q_f \in F} q_f \end{aligned}$$

□

Now, we have to introduce the fact denoted by D which forms with a set $\mathcal{R}$ of existential rules a knowledge base defined as $\mathscr{K} = (D, \mathcal{R})$. We use the fact trough the function $\text{answer}_{D,\mathcal{R}}()$. Let q be a non Boolean conjunctive query, $\text{answer}_{\mathscr{K}}(q)$ returns the set of answers of q on D using $\mathcal{R}$. In order to find a labellisation different from the set containing the initial query, we decide to set a goal t for the number of answers for labellisation queries. We want find a labellisation which contains queries which have a number of answers as close as possible to t.

Definition 5.4 (Small Query)**.** *Let $G = (V, E)$ a set graph of a conjunctive query q w.r.t. a set of rules $\mathcal{R}$ of a knowledge base $\mathscr{K}$ and let be t a positive integer. The* small queries $S^t_{q,\mathscr{K}}$ *of q and $\mathscr{K}$ with respect to an integer t is the subset of queries of V defined by:*

$$S^t_{q,\mathscr{K}} = \{q' \in V | \#\text{answer}_{\mathscr{K}}(q') < t\}.$$

Proposition 5.5. *Let $G = (V, E)$ a set graph of a conjunctive query q w.r.t. a set of rules $\mathcal{R}$ of a knowledge base $\mathscr{K}$ and let be t a positive integer. If a vertex of G, q_s is in $S^t_{q,\mathscr{K}}$ (resp. q_b is not in $S^t_{q,\mathscr{K}}$), then all descendants of q_s (resp. all predecessors of q_b) in G are in S^t_G (resp. not in S^t_G).*

Proof. Suppose that $q_s \in S^t_{q,\mathscr{K}}$ and $q_k \in V$ is a descendant of q_s such as $q_0 = q_s, q_1, \ldots, q_k$ is an oriented path in G. According to Property 5.2, we know that $q_k, \mathcal{R} \models q_s$. So we have the following implications:

$$\begin{aligned} q_k, \mathcal{R} \models q_s &\Rightarrow \text{answer}_{\mathscr{K}}(q_k) \subset \text{answer}_{\mathscr{K}}(q_s) \\ &\Rightarrow \#\text{answer}_{\mathscr{K}}(q_k) \leq \#\text{answer}_{\mathscr{K}}(q_s) \\ &\Rightarrow \#\text{answer}_{\mathscr{K}}(q_k) < t \\ &\Rightarrow q_k \in S^t_{q,\mathscr{K}} \end{aligned}$$

We can prove respectively that the case for $q_b \notin S^t_{q,\mathscr{K}}$ in the same way. □

We define the notion of border in a set graph in order to get a such labellisation.

Definition 5.6 (Frontier)**.** *Let $G = (V, E)$ a set graph of a conjunctive query q w.r.t. a set of rules $\mathcal{R}$ of a knowledge base $\mathscr{K}$ and be a positive integer such as $t \leq \#\text{answer}_{\mathscr{K}}(q)$. The* frontier $F^t_{q,\mathscr{K}}$ *of q and $\mathscr{K}$ with respect to t is the set of queries defined by:*

$$F^t_{q,\mathscr{K}} = V_s \cup V_b,$$

where

$$V_s = \{q_s | q_s \in S^t_{q,\mathscr{K}} \text{ and } \forall (q', q_s) \in E, q' \notin S^t_{q,\mathscr{K}}\}$$

and

$$V_b = \{q_b | q_b \notin S^t_{q,\mathscr{K}} \text{ and } \forall (q_s, q') \in E, q' \in S^t_{q,\mathscr{K}}\}$$

Definition 5.7 (Big Frontier)**.** *Let $G = (V, E)$ a set graph of a conjunctive query q w.r.t. a set of rules $\mathcal{R}$ of a knowledge base $\mathscr{K}$ and be a positive integer such as $t \leq \#\text{answer}_{\mathscr{K}}(q)$. The* big frontier $BF^t_{q,\mathscr{K}}$ *of q and $\mathscr{K}$ w.r.t. t is the set defined by:*

$$BF^t_{q,\mathscr{K}} = F^t_{q,\mathscr{K}} \setminus \{q_s | (q_s, q_b) \in (F^t_{q,\mathscr{K}})^2 \cap E\}$$

Example 5.8. *We illustrate the big frontier on the set graph drawn on Figure 11. In this figure, red vertices are representing small vertices, black vertices the rest of them. Vertices that are gray filled are the vertices of the big frontier.*

Definition 5.9 (Small Frontier)**.** *Let $G = (V, E)$ a set graph of a conjunctive query q w.r.t. a set of rules $\mathcal{R}$ of a knowledge base $\mathscr{K}$ and be a positive integer such as $t \leq \#\text{answer}_{\mathscr{K}}(q)$. The* small frontier *of q and $\mathscr{K}$ with respect to a integer t is the set defined by:*

$$F^t_{q,\mathscr{K}} \setminus \{q_b | (q_s, q_b) \in (F^t_{q,\mathscr{K}})^2 \cap E\}$$

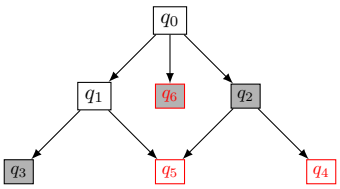

Figure 11: Example of big frontier defined vertices filled in grey, where small queries are in red and others queries in black.

Proposition 5.10. *Let $G = (V, E)$ a set graph of a conjunctive query q w.r.t. a set of rules $\mathcal{R}$ of a knowledge base $\mathscr{K}$ and be a positive integer such as $t \leq \#\text{answer}_{\mathscr{K}}(q)$. The small frontier and big frontier $BF^t_{q,\mathscr{K}}$ of q and $\mathscr{K}$ with respect to t is a **simple** labellisation of G.*

Proof. Suppose that we have q_1 and q_2 two vertices of the big (or small) frontier of G, such as $q_2, \mathcal{R} \models q_1$. According to Property 5.2, we know that there exists an oriented path from q_1 to q_2 in G. If $q_1 \in S^t_{q,\mathscr{K}}$, then we know that all descendants of q_1 in G are in $S^t_{q,\mathscr{K}}$ by Property 5.5, in particular $q_2 \in S^t_{q,\mathscr{K}}$. Also if $q_1 \notin S^t_{q,\mathscr{K}}$, then since $q_1 \in F^t_{q,\mathscr{K}}$, all children of q_1 are in $S^t_{q,\mathscr{K}}$, by extension all descendants of q_1 and in particular $q_2 \in S^t_{q,\mathscr{K}}$. In all cases $q_2 \in S^t_{q,\mathscr{K}}$, since q_2 is also in $q_2 \in F^t_{q,\mathscr{K}}$, so for same reasons as preceding, all predecessors of q_2 are not in $S^t_{q,\mathscr{K}}$ and $q_2 \notin S^t_{q,\mathscr{K}}$. We know that all descendants of q_1 are in $S^t_{q,\mathscr{K}}$ and all predecessors of q_2 are not in $S^t_{q,\mathscr{K}}$, so $(q_1, q_2) \in E \cap (F^t_{q,\mathscr{K}})^2$. By definition of the big (or small) frontier, it is impossible. □

Proposition 5.11. *Let $G = (V, E)$ a set graph of a conjunctive query q and t be an integer such as $t \leq \#\text{answer}_{\mathscr{K}}(q)$. The small frontier and the big frontier of q and $\mathscr{K}$ with respect to t are **covering** labellisations of G.*

Proof. We expose a proof for the big frontier denoted by $F^t_{q,\mathscr{K}}$, which it is also correct for the small frontier, until we do a difference between the two cases.

According to the Proposition 5.3, we just need to prove that for each leaf q_l of the set of leaves L_G of G, there exists an ancestors of q_l in $F^t_{q,\mathscr{K}}$. After that we have that $F^t_{q,\mathscr{K}}$ is a covering labellisation of G.

Let q_l be a leaf of G, we know that if $q_l \notin S^t_{q,\mathscr{K}}$, then $q_l \in F^t_{q,\mathscr{K}}$, because q_l has no child. In the other case $q_l \in S^t_{q,\mathscr{K}}$ then it exists an ancestor $q_a \in S^t_{q,\mathscr{K}}$ of q_l such as q_a has no parent in $S^t_{q,\mathscr{K}}$, because $q \notin S^t_{q,\mathscr{K}}$ and G is acyclic. By definition, we know that q_a is in the border of G w.r.t t.

If $F^t_{q,\mathscr{K}}$ is the small frontier, then $q_a \in F^t_{q,\mathscr{K}}$, else $F^t_{q,\mathscr{K}}$ is the big frontier and we know that either:

- q_a has no parent in the frontier, hence $q_a \in F^t_{q,\mathscr{K}}$
- or q_a has a parent in the frontier, hence this parent is in $F^t_{q,\mathscr{K}}$.

In all cases, we have fund a ancestor of q_l in $F^t_{q,\mathscr{K}}$. □

The building of the set graph from a product graph can be done by reuse Definition 5.1 in which we does not use the Repr() function, because set graph already use queries instead of classes of queries. With a set graph based on a product graph each result of this section holds, if Conjecture 4.24 is true. But the fact that the big (or small) frontier is covering hold anyway in both cases.

6 Implementation & Experiment

We describe in this section the implementation and experiments that have been led to validate our approach. We first tried a direct implementation of Algorithm 1, on which we intented to run our labellisation technique. As will be clear from the experiments, this approach proved to be too slow to be relevant. This motivated the optimization we developed for RDFS, as our test ontologies are expressed in this ontology language. The first set of experiments we present is thus focused on the computation of the rewriting graph of $\mathcal{R}$-rewritings, showing that the product graph approach leads to improved performance. Evaluating the quality of the output labellisations is a task that is much more complicated, and out of the scope of this paper (more on this in the future work section). We still performed a "soundness check", by running our algorithm on several queries on DBPedia and checking the output seemed "reasonable".

6.1 Resources

6.1.1 Datasets

We conducted experiments on two datasets:

- LUBM, which is classically used in the evaluation of ontology-based query answering. LUBM is a synthetic dataset, that is equipped with an ontology. The ontology exists in several flavors, and we consider here the RDFS version. A set of queries is also provided.
- DBPedia [2] is a knowledge base which is built from the information created in the Wikipedia project. The ontology is much richer than the LUBM one, while being still expressible in RDFS. We handcrafted a set of queries, presented in Annex A.

While LUBM is the classical experiment dataset in our setting, we judged it was not very interesting from a point of view of query answers exploration, as the ontology is rather flat and the data is synthetic. DBPedia avoids these two shortcomings, and will allow for better future integration with other semantic web resources.

6.1.2 Implementation

Our prototype relies on RDF-Commons [3], which is a library that has been developed within the Cedar (and previously OAK) team. More specifically, this library provides us with data storage, query representation, query reformulation and query evaluation capabilities.

6.2 Computation of the Rewriting Graph

In this experiment, we aim at comparing the time needed to compute the rewriting graph with Algorithm 1, or its approximation with the optimization for RDFS. We also evaluate the *quality* of the approximation. We define dedicated a set of queries on DBpedia accessible in Table 2 in annex.

First, we want to evaluate and compare the effectiveness of the two methods we propose for building a graph containing the rewritings. Figure 12 shows that the execution time to build exactly rewriting graph is slower than product graph, except for query Q04. For some queries like Q02, Q07, Q08, Q09, Q10, Q11, the rewriting building is not finished after 30 minutes. The execution of the both methods on query Q11 fails to finish before this timeout. Times of executions for rewriting graph building is splited into times of the three following operations:

1. *Rewritings & morphisms* corresponds to Lines 4 to 18 of Algorithm 1 and to the building of homomorphical saturation of the process graph,
2. *Graph of classes* corresponds strongly connected components merging at Line 21, by MergeStronglyConnectedComponents of Algorithm 1,
3. *Transitive reduction* corresponds to transitive reduction computed by TransitiveReduction function of the same algorithm.

In order to compare also the quality of the output of the product graph approximation to the exact rewriting graph, we show the number of vertices in Figure 13 and the number of edges in Figure 14 of both methods. We also add to each figure the output value after each steps of the rewriting graph method, if the value changes during this step. We notice that when the both do not fail with a timeout, the number of vertices and edges of product graph is quite comparable with those of the rewriting graph. The small size of the output for query Q04 may be explain the fact that product graph building is slower than that of the rewriting graph.

6.3 Building Labellisations

Algorithm 3 defines how we build the set graph from the product graph of a query and a set of rules and how we extract of it the big frontier w.r.t. of an integer. From Line 2 to 6, we increase the product graph with selfLeaf to construct a set graph. We build a topological ordered list of the vertices of set graph at Line 8. Then we browse vertices of set graph to extract the big frontier between Line 11 to Line 17.

We display at Figure 15 the set graph and its labellisation. MB: to be continued.

We also run labellisation algorithm on the query $q(x, y) = x$ rdf:type :Politician, x :spouse y, y rdf:type :Artist.

[2] http://wiki.dbpedia.org/downloads-2016-10

[3] https://bil.inria.fr/fr/software/view/2903/tab

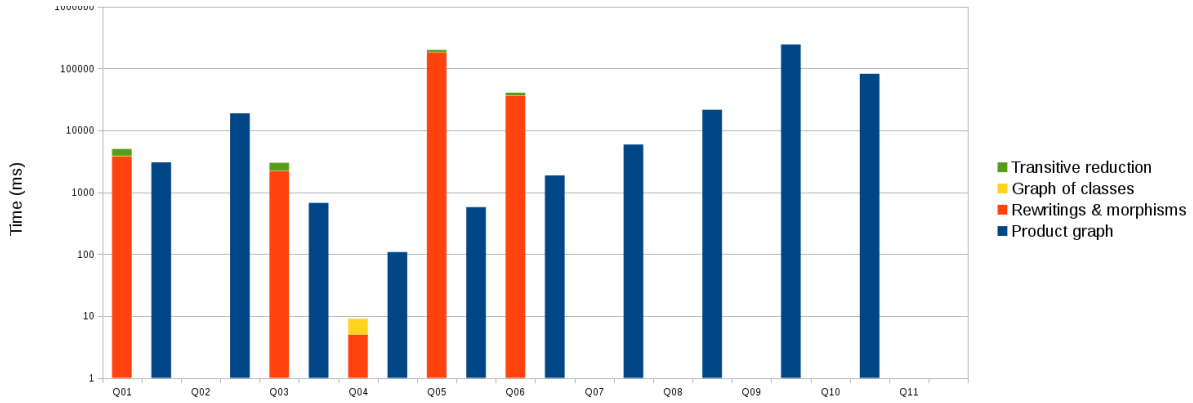

Figure 12: Comparaison of building times of the rewriting graph and product graph. Rewriting graph times are splited into each of step of the algorithm.

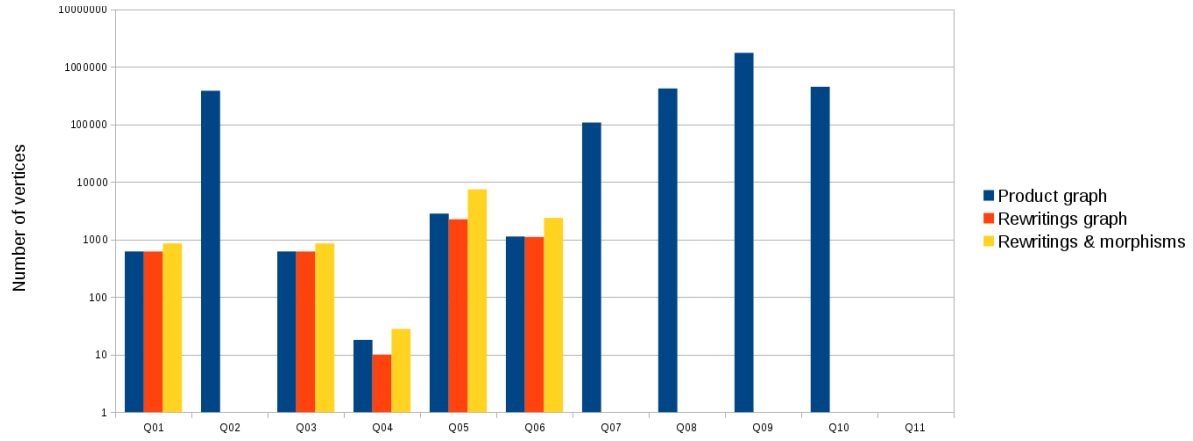

Figure 13: Comparaison of number of vertices of rewriting graph and product graph.

7 Future Work

There are several ways to extend this work. We sketch here a few of the possibilities.

Evaluation As already mentioned, evaluating the quality of the labelisation algorithm is an important and non-trivial task, as the aim is to ease the grasp of answer sets by human users. Setting such an experiment is a non-trivial task and was out of the scope of this internship. A lighter way of evaluating the interest of such an approach would be to disseminate (an upgraded version) of the tool developed in the internship to get feedback on the interest of the proposed labellisations.

Exploring the Space of Labellisation Strategies The algorithm that we propose is in some sense *data oblivious*: data is only used to chose labels among queries that are generated, but it has no influence on the set of labels that are considered. While this has some advantages with respect to the running time of the algorithm, it significantly reduces the insights that such a technique can provide. A systematic description of labellisation strategies together with algorithms illustrating each strategy would be very beneficial.

Control of the labellisation size The initial idea behind the labellisation was to sum up the answers by using semantics labels defining interesting subsets of answers. But our method has no way to control the number of labels contained in the labellisation. In order to return an apprehendable labellisation, it could be relevant to control or limit the number of labels. One way to achieve this is to consider a distance on the labels for clustering the actual labellisation.

Optimization A substantial part of the presented work consists in optimizing the creation of the rewriting graph of $\mathcal{R}$-rewritings, exploiting the fact that there are interesting ontologies available whose expressivity is rather weak. As seen in the experiments, even optimized some realistic queries do not

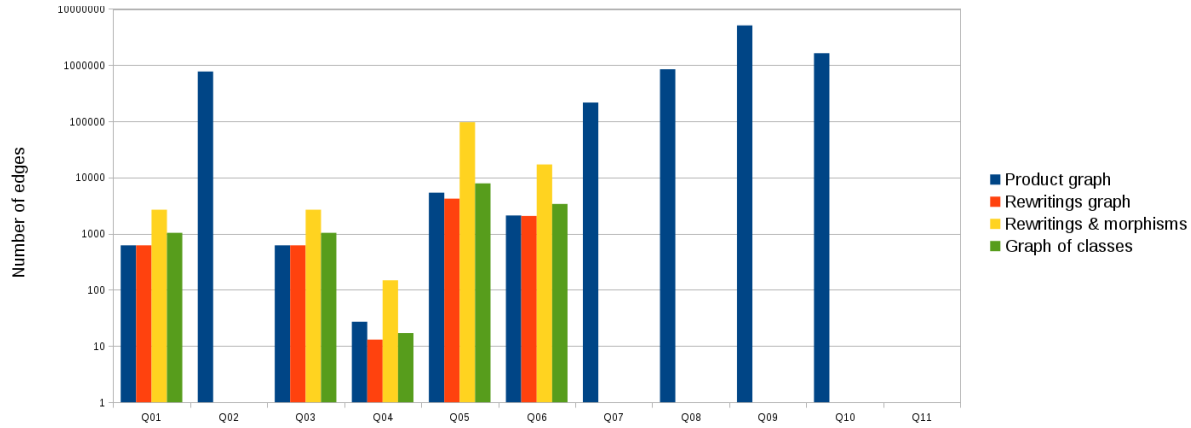

Figure 14: Comparaison of number of vertices of rewriting graph and product graph.

Input: A conjunctive query q, a finite unification set $\mathcal{R}$ and t an integer
Output: The big frontier of the set graph of q w.r.t. $\mathcal{R}$ and t build from product graph

$G = (E, V) = \text{ComputeProductGraph}(q, \mathcal{R})$
for $q' \in V$ **do**
  if q' *is not a leaf of* G **then**
    $V = V \cup \{SelfLeaf(q')\}$
    $E = E \cup \{(q', SelfLeaf(q'))\}$
  end
end
$I = \text{TopologicalOrderedVertices}(G)$
$S = \text{SmallQueries}(V, t)$
$L = \{\}$
for $q' \in I$ **do**
  if $q' \in S$ *and* $Parents(q') \cap L = \emptyset$ *and* $Parents(q') \cap S = \emptyset$ **then**
    $L = L \cup \{q'\}$
  end
  if $q' \notin S$ *and* $Children(q) \cap (V \setminus S) = \emptyset$ **then**
    $L = L \cup \{q'\}$
  end
end
return L

Algorithm 3: ComputeBigFrontier($q, \mathcal{R}, t$)

allow for a quick computation of that graph. A way to avoid that problem would be to compute only parts of the graph, refining only the parts that are necessary when considering the query answers. The second step is also prone to optimization, as queries that are evaluated usually share high commonalities, leaving the space for the use multi query optimization techniques.

Rules Generalization We define a method to get a labellisation from a set of rules and a query, such as we know that the set of rewriting queries is finite. Some sets of rules are such as for each queries there exists a finite subset of its rewritings satisfying Theorem 2.18 as the all set of its rewritings. Even if a covering and simple labellisation can be extracted from rewritings in this context, the building of the rewriting graph can be infinite. It will be interesting to generalize the rewriting graph in order to cope with this context.

Annex A

We used the following set of queries for our experiments on DBPedia.

Q01(x,y)	x :spouse y, x rdf:type :President, y rdf:type :Person;
Q02(x,y)	x :spouse y, x rdf:type :Person, y rdf:type :Person;
Q03(x,y)	x :spouse y, x rdf:type :Person;
Q04(x,y)	x :targetAirport y, x rdf:type :Airline, y rdf:type :Airport;
Q05(x,y)	x :capital y, x rdf:type :PopulatedPlace, y rdf:type :City;
Q06(x)	x :champion y, x rdf:type :SportEvent, y rdf:type :Athlete;
Q07(x)	x :author y, x rdf:type :Person, y rdf:type :Work;
Q08(x)	x :birthPlace y, x rdf:type :Person, y rdf:type :Place;
Q09(x,z)	x rdf:type :Person, x:birthPlace y, y rdf:type :PopulatedPlace, y :capital z, z rdf:type :City ;
Q10(w,x,y,z)	w rdf:type :FilmFestival, w :closingFilm x, x rdf:type :Film, x :directory y, y rdf:type :Person, x:starring z, z rdf:type :Actor ;
Q11(x,y,z)	x :rdf:type :SportsTeam, x :club y, y rdf:type :Athlete, z :champion y, z rdf:type :SportEvent, y rdf:type :Athlete;

Table 2: Tables of queries used for evaluation

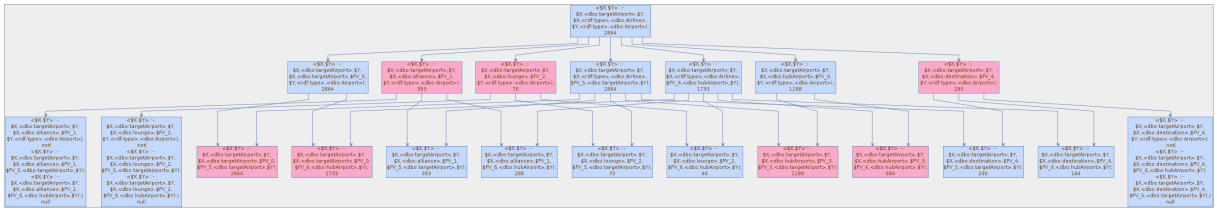

Figure 15: Set graph of query Q04 (Table2) during the labellisation process. The vertices of the labellisation appear in pink.