Ontological Conjunctive Query Answering over Semi-Structured KBs

Bruno Paiva Lima da Silva, Jean-François Baget, Madalina Croitoru

Ontological Conjunctive Query Answering over Semi-Structured KBs

Bruno Paiva Lima da Silva #, Jean Francois Baget #, Madalina Croitoru #

#*EPI GraphIK*
LIRMM (CNRS - Université Montpellier II)
INRIA Sophia Antipolis
161 Rue Ada, F-34392 Montpellier Cedex 5, France
{bplsilva,baget,croitoru}@lirmm.fr

Abstract—In the context of ontological conjunctive query answering different paradigms for representation and their subsequent manipulation by dedicated reasoning systems have been successfully studied in the past. However, new challenges, problems and issues have appeared in the context of knowledge representation in AI that involve the logical manipulation of increasingly large information sets (see for example the Semantic Web). In this paper we explain these challenges by the means of an example and try to further identify the difficulties ahead of our goal.

I. INTRODUCTION

The purpose of this paper is to present our current research question: *"How to store large knowledge bases in order to be able to scale up ontological conjunctive query answering?"*

Let us further describe the context of this research question. Knowledge Representation and Reasoning (KRR) studies computational models for building explicit representations of knowledge processed by reasoning engines. Systems based upon such computational models are called knowledge-based systems (KBS). Their fundamental components are a knowledge base (KB) containing different kinds of knowledge and a reasoning engine which performs the inferences. First Order Logic (FOL) is the reference logic in KRR and most formalisms in this area can be translated into fragments (i.e., particular subsets) of FOL. A large part of research in this domain can be seen as studying trade-off between the expressivity of languages and the complexity of (sound and complete) reasoning in these languages. The fundamental problem in KRR formal languages is deduction (or consequence, entailment) checking: *"can a given piece of knowledge be deduced from other pieces of knowledge (for instance the KB)?"* Another important problem is consistency checking: *"is a set of knowledge pieces (for instance the KB itself) consistent, i.e., is it sure that nothing absurd can be deduced from it?"* Here we are interested in the ontological conjunctive query answering problem. In its decision form, it asks *if the KB contains an answer to the query*, and is equivalent to deduction in the special case of boolean queries (i.e., queries with a yes/no answer). Queries must be at least as expressive as conjunctive queries in databases (eg. a conjunction of positive atoms in logical form); and the knowledge base is split into a factual component (that can be seen as a database or as a

conjunction of positive atoms) and an ontological component, that is often expressed by formulas of a specific subset of FOL.

Nevertheless, in this context, different paradigms for representation and their subsequent manipulation by dedicated reasoning systems have been successfully studied in the past. However, new challenges, problems and issues have appeared in the context of knowledge representation in AI that involve the logical manipulation of increasingly large **knowledge** sets (see for example the Semantic Web). Improvements in storage capacity and performance also affect the nature of KRR systems. Their focus has now shifted towards representational power and execution performance. Therefore, research into KRR must move towards investigating structures for representation optimally manipulated to perform large scale reasoning, given very new and different constraints to those existing only few years ago. While a plethora of systems dedicated to non relational structures (such as NoSQL[1], which means "Not Only SQL") for **information** storing and querying have lately received much attention, we are interested in the representation and reasoning with **knowledge**. Our querying mechanisms are more expressive than typical queries done in the above mentioned systems. On the other hand, a discussion solely on expressivity / reasoning and how it relates to other NoSQL systems might turn abstract very fast, with the storage issues hidden away. This is why we chose to present this paper solely by the means of a running example where we identify informally the problems we are facing.

Our research interests[2] follow the computational and logic-oriented approach of KRR: the different kinds of knowledge have a logical semantics and the reasoning corresponds to inferences in this logic, at least for the kernels of studied languages. The fundamental decision problem we want to address is (boolean) ontological conjunctive query answering, which can be expressed as a deduction problem: *"is a (boolean) conjunctive query deducible from a KB?"* The afore mentioned KB queries are supposed to be at least as expressive as the basic queries in databases, i.e., conjunctive queries, which can be seen as existentially closed conjunctions of atoms.

A knowledge base is composed of a set of facts, and a set

---

[1] http://nosql-database.org/
[2] http://www.lirmm.fr/graphik/index.html

of rules (ontological knowledge). In this paper rules will be expressed using ∀∃-rules, which have the same logical form as TGDs in databases, and which forms the core of Datalog$^\pm$ family of languages. ∀∃-rules form an abstraction particularly well-suited to the representation of ontological knowledge since they generalize several specific knowledge representation languages adapted to query answering: RDFS [1] (the basic semantic web language), constraints in F-logic-Lite [2], [3] (a powerful subset of F-logic, a formalism for object-oriented deductive databases), as well as the core of new families of description logics tailored for conjunctive query answering [4], [5], [6], [7].

II. EXAMPLE

In this paper, we deliberately chose to present our work informally, by the means of an example. Through this example, we highlight some issues and the limits of current methods used in KRR. For the theoretical foundation of the formalisms detailed below please see: [8] for relational databases and Datalog evaluation techniques, [9] for our graph based representation and [10] for Datalog$^\pm$, that corresponds to the graph based rules as discussed in [11].

Let us consider a knowledge base representing the companies people work in and their practised sport. The knowledge base is composed of the following facts:

- (1) "There is a person, named *Bob*, working for some Organisation (u). This person plays for some football team (p) which has "$RueAda, Montpellier$" as address. There is another person, named *Tom*, working for another Organisation (v), which is part of u, and who plays for some rugby team (q), which also has "$RueAda, Montpellier$" as address."

The knowledge base is also composed of the following rules:

- (2) "If x works for y, and y is part of z, then x also works for z"
- (3) "If x and z work for y, then x and z are co-workers"
- (4) "If x and z share the same address y, then there is a club t, which contains x and z and has y as address."
- (5) "If x plays in y, which is contained in z, then x is a member of z."
- (6) "If x and z are members of y, then they are in the same club."

Being given the above mentioned facts and rules we want to be able to answer the two following queries:

- (7) "Is there a person which works in an organisation and plays in a football team?"
- (8) "Are there two persons who are co-workers and share the same sports club?"

Please note that query (7) can be answered directly on the facts in (1) i.e. without the application of any of the rules (2) – (6). However, query (8) can only be answered by applying the rules (2) – (6). Also, please note that rules (2), (3), (5) and (6) intuitively only add relations between existing entities, while rule (4) generates a new entity (club) that is added to the knowledge base[3].

The next section (Section II-A) shows the logical translation of the knowledge base. Then in Section II-B we show the equivalent representation using relational databases. Sections II-C and II-D explain how query answering takes place (without and with rule application). In Section III we then illustrate how the knowledge base can be expressed using a graph based formalism and how query answering (without rules) is equivalent to a labelled graph homomorphism.

A. First Order Logic translation

In order to lay the logical foundations for the different models used later in this paper please find below the First Order Logic translation of the facts in the knowledge base mentioned in Section II.

- (1) $\exists x, y, u, v, p, q$ ($is\text{-}a(x, \text{"}Person\text{"})$ $\wedge name(x, \text{"}Bob\text{"}) \wedge works\text{-}for(x, u)$ $\wedge is\text{-}a(u, \text{"}Organisation\text{"})$ $\wedge is\text{-}a(y, \text{"}Person''\text{"}) \wedge name(y, \text{"}Tom\text{"})$ $\wedge works\text{-}for(y, v) \wedge is\text{-}a(v, \text{"}Organisation\text{"})$ $\wedge part\text{-}of(v, u)$ $\wedge is\text{-}a(p, \text{"}FootballTeam\text{"}) \wedge plays\text{-}in(x, p)$ $\wedge is\text{-}a(q, \text{"}RugbyTeam\text{"}) \wedge plays\text{-}in(y, q)$ $\wedge address(p, \text{"}RueAda, Montpellier\text{"})$ $\wedge address(q, \text{"}RueAda, Montpellier\text{"})$)

The rule set containing ∀∃-rules:

- (2) $\forall x, y, z$ $works\text{-}for(x, y) \wedge part\text{-}of(y, z) \rightarrow works\text{-}for(x, z)$
- (3) $\forall x, y, z$ $works\text{-}for(x, y) \wedge works\text{-}for(z, y) \rightarrow co\text{-}worker(x, z)$
- (4) $\forall x, y, z$ $\wedge address(x, y) \wedge address(z, y) \rightarrow \exists t$ $is\text{-}a(t, \text{"}Club\text{"}) \wedge contains(t, x) \wedge contains(t, z) \wedge address(t, y)$
- (5) $\forall x, y, z$ $plays\text{-}in(x, y) \wedge contains(z, y) \rightarrow member(x, z)$
- (6) $\forall x, y, z$ $\wedge member(x, y) \wedge member(z, y) \rightarrow same\text{-}club(x, z)$

And queries:

- (7) $\exists x, y, z$ $works\text{-}for(x, y) \wedge is\text{-}a(y, \text{"}Organisation\text{"}) \wedge plays\text{-}in(x, z) \wedge is\text{-}a(z, \text{"}FootballTeam\text{"})$

---

[3]One has to be careful with rules adding new constants or variables to the knowledge base since some of them could be fired forever (for example "Every person has another person as its parent".) For more information please see [12].

- (8) $\exists x, y$ $co\text{-}worker(x, y) \wedge same\text{-}club(y, x)$

B. Equivalence with a Relational Database

This section shows an encoding of the example formulas using a relational database containing the facts in the knowledge base introduced in Section II. For every n-ary predicate P in our facts, we create a new table P with n columns. Variables in the knowledge base are then frozen and replaced by fresh constants, since there are no variables in a relational database. After that, for every atom in the facts, a row is added to the table corresponding to the name of the predicate.

Figures 1 to 4 represent the relational database for facts (1).

name	
1	2
$<x>$	"Bob"
$<y>$	"Tom"

works-for	
1	2
$<x>$	$<u>$
$<y>$	$<v>$

Fig. 1.   *name* and *works-for* relations

is-a	
1	2
$<x>$	"Person"
$<u>$	"Organisation"
$<y>$	"Person"
$<v>$	"Organisation"
$<p>$	"FootballTeam"
$<q>$	"RugbyTeam"

Fig. 2.   *is-a* relation

part-of	
1	2
$<v>$	$<u>$

plays-in	
1	2
$<x>$	$<p>$
$<y>$	$<q>$

Fig. 3.   *part-of* and *plays-in* relations

address	
1	2
$<p>$	"RueAda, Montpellier"
$<q>$	"RueAda, Montpellier"

Fig. 4.   *address* relation

C. Rule Application

Rule application allows to enrich the KB with new facts deduced from existing ones. In Datalog$^\pm$, a rule is an expression of form $\forall x_1, ..., x_k$ $C \rightarrow C'$ where x_1 to x_k are the variables present in the conjunction of atoms C and C'. C is called the head of the rule, while C' is the tail.

In Datalog$^\pm$, the *chase* algorithm consists in enriching the the database (directly, or just a local copy) with inferred information. It queries the head of the rule in the database using a $SELECT\text{-}FROM\text{-}WHERE$ call, and if there is a positive answer to the query, a specification of the tail of the rule is then inserted to the base with an $INSERT$ call. In the case of our example, rule (2) will fire since (1) $<v>$ is *part-of* $<u>$ and (2) $<y>$ *works-for* $<v>$. Therefore, the new row $<y>,<u>$ will be added to the table *works-for*. Similarly, the rules (3), (5) and (6) will add new rows in corresponding tables. Please note though that the expressivity of the rules we are interested in is that of $\forall\exists$-rules as explained in the introduction. This means that we want to be able to add not only new rows but also new relations between pieces of information, which in this case would correspond to adding new tables or changing the relational schema of the database. This cannot be dealt with by Datalog itself. We point out that TGDs can add rows that use newly generated constants, which why the *chase* may not halt (the deduction problem is indeed undecidable). However, decidable subclasses of the problem have been studied (eg. [3], [11], [12]).

D. Querying a Knowledge Base

The SQL translation of the queries in the example is as follows:

- (7) $SELECT$ $*$ $FROM$ $works\text{-}for, plays\text{-}in, is\text{-}a$ $isa1, is\text{-}a$ $isa2$ $WHERE$ $works\text{-}for.2 = isa1.1$ AND $isa1.2 = "Organisation"$ AND $plays\text{-}in.2 = isa2.1$ AND $isa2.2 = "FootballTeam"$

- (8) $SELECT$ $*$ $FROM$ $co\text{-}worker, same\text{-}club$ $WHERE$ $co\text{-}worker.1 = same\text{-}club.2$ AND $co\text{-}worker.2 = same\text{-}club.1$

No rules need to be applied to answer the first query, however, 4 tables (the same table twice), need to be joined. More importantly, the second query can only be answered after the addition of new pieces of knowledge to the base. Consistency becomes a problem in this case since it is initially maintained by the fact that the data stored is supposed to be organized into normalized tables, in which independent sets of data are related by a key and in which redundancy is avoided. While this model seems appopriate when dealing with sets of structured data, the semi-structured nature of our data makes the relational model difficult to fit into. We can distinguish two different methods for rule application over a KB: forward-chaining and backwards-chaining.

The forward-chaining method fires all rules looking for new pieces of information to be added to the base until an answer to the query is found. The large amount of joins when performing a request in the KB becomes quickly very costly, and things get even more difficult when new information has to be introduced to the base, since indexes and the database schema have to

be updated several times. On the other hand, the backwards-chaining method does not add any new information to the KB, but has a major drawback as it can possibly create a huge number of queries.

Scaling is also one of the real problems of those approaches since joins usually create new temporary tables in main memory. One should not forget our initial intentions to answer conjunctive queries when the knowledge base itself cannot be held any more in main memory.

So far we have presented the relational approach and its drawbacks when scaling up. These drawbacks led us to look into graph appropriate storage for solving our problem. Let us further detail the graph model and highlight **our** specific needs for the conjunctive query answering problem.

III. GRAPH-BASED REPRESENTATION

In this section we represent the knowledge base using the graph based formalism detailed in [9]. In this representation, constants and variables are vertices in our (hyper)graph, while (hyper)edges between those vertices represent the atoms in our KB[4].

$$name(x, \text{``Bob''}) \qquad x \bullet \xrightarrow{\ name\ } \bullet \text{``Bob''}$$
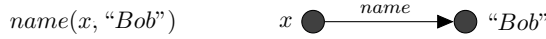
Fig. 5. A fact in the knowledge base and its graph representation on the right.

Figure 5 shows the transformation of the facts of the KB in a graph, and Figure 6 represents the graph containing the facts in the KB for the given example.
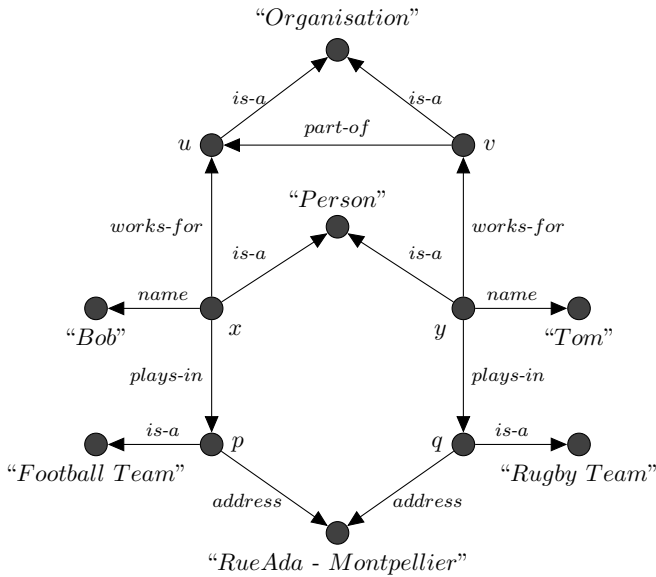
Fig. 6. Graph containing the facts (1) from Section II.

[4]If the predicate in the knowledge base is n-ary, we represent it as a hyperedge. In our example, we have only binary predicates, thus simple edges.

A. Querying

In a graph-based KB, queries are represented as graphs. Figures 7 and 8 show the graphs of both queries from Section II.
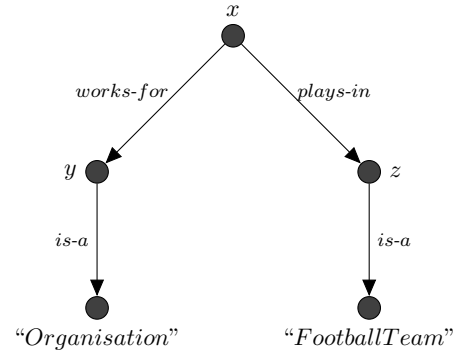
Fig. 7. Graph representing the query (7) from Section II.
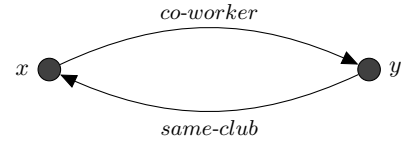
Fig. 8. Graph representing the query (8) from Section II.

B. Homomorphism

Queries in a graph-based KB are answered by computing a labelled graph homomorphism from the graph representing the query Q to F, the graph of facts. The fundamental theorem says that a logical formula Q can be deduced from another formula F iff there is a labelled graph homomorphism from the graph representing Q to the graph representing F.

Basically, a labelled graph homomorphism is a mapping, say π, from the vertices of the graph Q (the query), to those of the graph F (the facts) that preserves both the constants labelling the nodes and the hyperedges and their type. More formally:

- if x is a vertex of Q labelled by a constant c, then the label of $\pi(x)$ is also c;
- if there is a hyperedge labelled p between vertices $(x_1, ..., x_q)$ in Q, then there must also be a hyperedge labelled p between vertices $(\pi(x_1), ..., \pi(x_q))$ in F.

Since the labelled graph homomorphism problem is NP-Complete, a backtracking algorithm is used to enumerate all the possible matchings that answer the query.

Let us now detail the execution of a backtracking algorithm for homomorphism finding. We consider a recursive algorithm that, at every loop, extends the current matching algorithm. We name the algorithm *Extend()*. The algorithm takes as

parameters the set of already matched nodes couples, and the graphs representing query and facts.

Extend(MatchedNodes,Query,Facts) returns true if the number of couples in the matched nodes set is equal to the number of nodes in the query graph. If not, it selects a node n that has not yet been matched, and returns a list of **relevant** possible node matchings i of the node n according to MatchedNodes, Query and Facts. Then, for each of the possible matchings, the algorithm is recursively called with the couple (n,i) added to the matched nodes set.

Let us consider the first query that can be answered without rule application (detailed in the next section). To find a homomorphism from this query to the facts graph, the algorithm starts matching all the constant vertices from the query graph to the facts one. As we have seen on Figure 7, the constants in the query, "Organisation" and "FootballTeam", are also present in the facts, and are added to the matched nodes set. After that, the algorithm will try to match the remaining variable nodes of the query graph. Node x has two outgoing relations to another variable: $works\text{-}for$ and $plays\text{-}in$. Possible matchings for x then are nodes x and y from the graph of facts. When node x in query is matched with y in facts, y will then be then paired to v since it is an Organisation, and then z will not have any matching possibility since it there is no node in the facts connected to y via $plays\text{-}in$ relation and to $"FootballTeam"$ via $is\text{-}a$ relation. This branch will then be left and other branches will be explored in order to find a positive answer. On the other side, when x is paired with x, y will then be paired to u, and z to p, which is a Football Team. True is then returned by the algorithm, meaning that there is in fact an homomorphism from Q to F, and the matched nodes set $\{(x,x),(y,u),(z,p)\}$ is the answer of our query.

There are thus two elementary operations needed in order to calculate a homomorphism from the query graph to the facts:

- Matching a vertex from the query graph with another in the facts graph [5]
- Accessing the neighborhood of both matched vertices to check whether their edges and adjacent vertices are compatible or not.

While matching vertices from two different graphs into a pair can be easily done in constant time, efficient access to the neighbourhood of a vertex is primordial in order to have some real advantage over traditional relational database systems.

For instance, for a n-ary predicate p of arity 4, the complexity of searching for the presence of an atom $p(x, <c1>, y, <c2>)$ in a KB using a relational database is $O(m_p)$, m_p being the size of the table p. Using a graph-based KB instead, this complexity can be improved to $O(d_{<c1>})$, with d being the size of the neighbourhood of a given vertex, in this case $<c1>$, if there are optimized data structures and algorithms to quickly access the neighbourhood of a vertex. This improvement is due to the fact that the encoding of semi-structured data into a graph does generate a graph with a very small density, in which good efficiency

[5]We do not make the assumption that the facts graph is connected.

is obtained when accessing the neighbourhood of a vertex. However, there is still no efficient method for doing it when the graph-based KB is stored in secondary memory.

It is however impossible to have an answer for the second query without enriching the KB with new facts (or rewriting the query). Let us detail in the next section how rule application takes place.

C. Rule application in a graph-based KB

The forward-chaining method for rule application in a graph-based KB that will be described more precisely in this part of the paper mimics exactly the *chase* method mentioned in Section II-C.
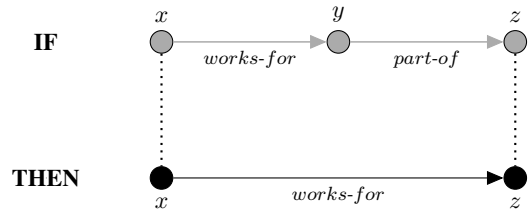
Fig. 9. Graph representing the Rule (2) from Section II.

In a graph-based KB, rules are represented using two pieces of graph: head (hypothesis) and tail (conclusion). Figure 9 and 10 show the graph representation of the Rules (2) and (4) from the example in Section II.
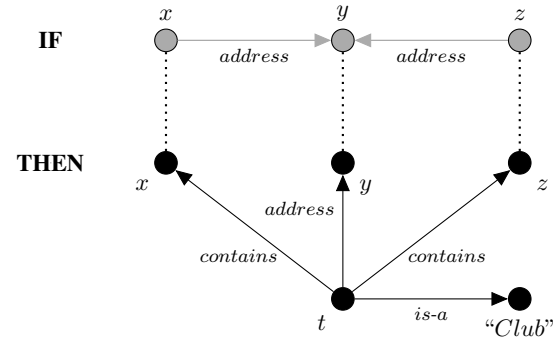
Fig. 10. Graph representing the Rule (4) from Section II.

Rule application works in two steps: first, the head of the rule needs to be deducted from the facts (using homomorphism). Then, if there is a homomorphism, the conclusion gets added to the graph respecting links between vertices in head and tail parts of the rule. The method we use to enrich the graph-based KB is called forward-chaining, which consists in applying all rules of the rule set until there is no more new information to be added to the KB. Note that while rule (2) only adds edges to the facts graph rule (4) can also introduce new variables to the KB. This is the reason why, apart from the efficient operations mentioned above, we need a flexible

data structure that **also** allows quickly adding edges and nodes to the graph. Figure 11 shows the facts graph after applying rule (4). Once the graph saturated with all the rules, the second query can be answered in the same way as described before.

Fig. 11. Graph after rule (4) application. Please note that the old graph vertices and edges are colored in grey. Old edges are also now dotted. The newly introduced vertices and edges are in black.

IV. CONCLUSION

Knowledge Representation and Reasoning (KRR) studies computational models for building explicit representations of knowledge processed by reasoning engines. Given the need for increasingly large knowledge, research into KRR must move now towards investigating structures for representation, optimally manipulated to perform large scale reasoning.

In this work, we have compared two different approaches for the ontological conjunctive query answering. We have first introduced the method for performing reasoning over a KB using a relational database with a set rules. This method has already proved that it works very well when dealing with large sets of structured data, stored whether in main or in secondary memory. Then we have presented another approach for solving our problem using a graph-based KB with graph rules. As we have showed before, this approach ensures an unicity between the KB and the query languages and has theoretically better results than the first method when reasoning over semi-structured data.

Nevertheless, our graph based approach for representation[6] and reasoning[7] only works well when the facts hold in main memory. This is why we are now interested in finding adequate

storage solutions suitable for our graphs reasoning manipulations. More precisely, as shown in the previous sections, the reasoning mechanism we use on graphs is labelled graph homomorphism. When detailing our backtracking algorithm we identified two important qualities of the desired graph based storage: a good indexing on nodes (that facilitates the identification of potential candidates for matching) and an efficient structure of access to the neighbors of a given node, and, for these neighbors, its subsequent neighbors. Moreover, rule application will mean that our structure also needs to be flexible for the insertion of new nodes and edges.

After having set the operations we need in order to perform reasoning over graphs, our future work will consist in evaluating current solutions to the ontological conjunctive query problem (such as AllegroGraph[8], Neo4J[9] or HyperGraphDB[10]) and testing their large scale efficiency. In parallel, we will also investigate different graph paradigms for storage that allow efficient manipulation for the operations previously described.

REFERENCES

[1] P. Hayes, Ed., *RDF Semantics*, ser. W3C Recommendation. W3C, 2004, http://www.w3.org/TR/rdf-mt/.

[2] A. Calì and M. Kifer, "Containment of conjunctive object meta-queries," in *Proceedings of the Thirty-Second International Conference on Very Large Data Bases (VLDB 2006), Seoul, Korea*, 2006, pp. 942–952.

[3] A. Calì, G. Gottlob, and M. Kifer, "Taming the infinite chase: Query answering under expressive relational constraints," in *Proceedings of the Eleventh International Conference on the Principles of Knowledge Representation and Reasoning (KR 2008), Sydney, Australia*, 2008, pp. 70–80.

[4] D. Calvanese, G. D. Giacomo, D. Lembo, M. Lenzerini, and R. Rosati, "Tractable reasoning and efficient query answering in description logics: The DL-Lite family," *J. Autom. Reasoning*, vol. 39, no. 3, pp. 385–429, 2007.

[5] C. Lutz, D. Toman, and F. Wolter, "Conjunctive query answering in the description logic el using a relational database system," in *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, California, USA,*, 2009, pp. 2070–2075.

[6] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat, "DL-$\mathcal{SR}$: a lite DL with expressive rules: Preliminary results," in *Proceedings of the Twenty-First International Workshop on Description Logics (DL2008), Dresden, Germany*, 2008.

[7] A. Calì, G. Gottlob, and T. Lukasiewicz, "A general datalog-based framework for tractable query answering over ontologies," in *Proceedings of the Twenty-Eigth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS 2009), Providence, Rhode Island, USA*, 2009, pp. 77–86.

[8] S. Abiteboul, R. Hull, and V. Vianu, *Foundations of Databases*. Addison-Wesley, 1995.

[9] M. Chein and M.-L. Mugnier, *Graph-based Knowledge Representation and Reasoning—Computational Foundations of Conceptual Graphs*, ser. Advanced Information and Knowledge Processing. Springer, 2009.

[10] A. Calì, G. Gottlob, T. Lukasiewicz, B. Marnette, and A. Pieris, "Datalog+/-: A family of logical knowledge representation and query languages for new applications," in *LICS*. IEEE Computer Society, 2010, pp. 228–242.

[11] J.-F. Baget, M. Leclère, M.-L. Mugnier, and E. Salvat, "Extending decidable cases for rules with existential variables," in *Proceedings of the Twenty-First International Joint Conference on Artificial Intelligence (IJCAI 2009), Pasadena, California, USA,*, 2009, pp. 677–682.

[12] J.-F. Baget, M. Leclère, and M.-L. Mugnier, "Walking the decidability line for rules with existential variables," in *Proceedings of the Twelfth International Conference on Principles of Knowledge Representation and Reasoning, Toronto, Canada*. AAAI Press, 2010.

---

[6] http://www.lirmm.fr/cogui/
[7] http://cogitant.sourceforge.net/

[8] http://www.franz.com/agraph/allegrograph/
[9] http://neo4j.org/
[10] http://www.kobrix.com/hgdb.jsp