Implementation of SPARQL Query Language Based on Graph Homomorphism

Olivier Corby, Catherine Faron Zucker

Implementation of SPARQL Query Language based on Graph Homomorphism

Olivier Corby[1] and Catherine Faron-Zucker[2]

[1] INRIA 2004 route des lucioles - BP 93
FR-06902 Sophia Antipolis cedex
Olivier.Corby@sophia.inria.fr
[2] I3S, Université de Nice Sophia Antipolis, CNRS
930 route des Colles - BP 145
FR-06903 Sophia Antipolis cedex
Catherine.Faron-Zucker@unice.fr

Abstract. The SPARQL query language is a W3C candidate recommendation for asking and answering queries against RDF data. It offers capabilities for querying by *graph patterns* and retrieval of solutions is based on *graph pattern matching*. This paper is dedicated to the implementation of the SPARQL query language and its pattern matching mechanism which is reformulated into a graph homomorphism checking constrained by filter evaluation[3].

1 Introduction

The SPARQL[4] query language is a W3C candidate recommendation for asking and answering queries against RDF[5] data. It offers capabilities for querying by *graph patterns* and retrieval of solutions is based on *graph pattern matching*. This paper is dedicated to the implementation of the SPARQL query language and its pattern matching mechanism in the CORESE[6] semantic search engine.

Intuitively, a SPARQL basic graph pattern P is an RDF graph whose some terms are replaced by variables; the basic graph pattern P of a query Q answered against an RDF graph G matches with pattern solution S if G entails $S(P)$, with $S(P)$ the replacement of every variable v in P by $S(v)$. This lead us to RDF graph entailment. The early development of CORESE relies on the reformulation of RDFS-entailment as graph homomorphism [2]. In CORESE last versions, SPARQL pattern matching is also reformulated as a graph homomorphism: it answers a SPARQL query by searching all the existing projections of the conceptual graph P representing the query pattern into the conceptual graph G representing the RDF(S) dataset.

[3] Published in the proceedings of the International Conference on Conceptual Structures, ICCS, Sheffield, July 2007. Springer Verlag.
[4] http://www.w3.org/TR/rdf-sparql-query/
[5] http://www.w3.org/TR/rdf-mt/
[6] http://www-sop.inria.fr/acacia/corese/

We propose an efficient algorithm relying on two main principles. A first principle comes from our choice to represent the SPARQL query pattern and the RDF graph by conceptual graphs: we take advantage of their structure to limit the search space for node projections by dealing with relations first and ordering them so as to force node projections. Second, our algorithm integrates value constraints in the search for graph homomorphisms: SPARQL is provided with a wide range of functions and expressions to filter solutions to queries and our algorithm integrates these value constraints *during* the search process to efficiently reduce the search space. We further detail both principles in the following.

2 Solution Filtering while Pattern Matching

Value constraints and solution modifiers allow to filter solutions retrieved by pattern matching. However a sequential algorithm where filtering would succeed pattern matching would be quite unefficient. For instance, let us consider a query asking for research reports and their authors, members of the INRIA institute, after 2002. The process of retrieving *all* the reports *before* filtering them to keep the only few ones written after 2002 would be unnecessarily expensive.

Consequently, our algorithm takes into account value constraints *during* the search for a graph projection: while searching for a projection of a SPARQL query graph into an RDF graph, as soon as for instance a date in the RDF graph is rejected because it does not pass a SPARQL filter, the projection as a whole which involves this date can be rejected.

Moreover, the sooner value constraints are taken into account the smaller the search space becomes. Therefore our algorithm handles SPARQL filters as soon as they are *evaluable*, which may depend on several graph nodes. For instance, let us consider the following SPARQL query asking for the research reports written before the graduating dates of their authors.

```
SELECT ?doc ?a ?d1 ?d2 WHERE {
   ?doc rdf:type ex:ResearchReport . ?doc ex:date ?d1 .
   ?doc ex:createdBy ?a . ?a ex:graduationDate ?d2 .
   FILTER (xsd:date(?d1) >= xsd:date(?d2)) }
```

Before its filter can be evaluated, both variables $?d1$ and $?d2$ occuring in it must be projected into RDF terms.

This "as early as possible" constraint evaluation principle implicitly defines an ordering of query graph nodes and characterizes an incremental process for the construction of a projection: the set of evaluable constraints increases as fast as possible, depending on the chosen current node of the query for which a projection is searched and those for which a projection has already been found.

3 Highest Precedence for Relations in Conceptual Graphs

Our algorihm takes advantage of the hypergraph structure of our representation of RDF graphs as conceptual graphs to limit the search space for node projections. We view conceptual graphs as hypergraphs where relation nodes have become hyperarcs, while concepts nodes remain the only nodes [1]. As a result, when searching for homomorphisms, relations no more are nodes: in our algorithm they are viewed as constraints for (concept) node projection. Nodes no more are projected in isolation but each one is projected at the same time as the other arguments of a chosen relation to which it participates; relations thus are constraints which reduce the search space of possible projections of nodes. This principle is close to the one described in [4].

Formally, we choose a first relation $r = (x_1, ..., x_i) \in U(P)$, such that $\forall t \in type(r), \exists r' = (x'_1, ..., x'_i) \in U(G)$ such that $\exists t' \in type(r')$ with $t' \leq t$. This choice determines the projections $\pi(x_1) = x'_1, ..., \pi(x_i) = x'_i$ of $x_1, ..., x_i$. While doing so the theoretical search space $V(G) \times ... \times V(G)$ has become the extension of t'. Moreover, when dealing with the next chosen relations, some of their arguments will already have projections previously chosen and the search space for the remaining arguments will even more decrease.

4 Algorithm

Ordering Relations in the Query Graph Relations in the query graph P are heuristically ordered to constrain at best the search space. Heuristics are based on both the structure of query graph P and the RDF graph G.

Regarding the query graph structure, the ordering depends on both the connexity of relations on their arguments and the occurence of value constraints associated to relation arguments. By choosing a relation connected by the greatest number of arguments to previously chosen relations of P, these arguments already have projections which diminish the search space for the remaining arguments. Furthermore, the more value constraints on nodes are evaluated, the more the search space will diminish. At each step of the search we chose to handle the relation for which the greatest number of constraints are evaluable.

Regarding graph G against which the query is asked, the ordering depends on relation types and on how often relations of a given type (or subtype of it) occur in G. The early choice of the relations whose type occur the least in G will significantly reduce the search space.

Graph Indexing and Candidate Relations Graph G against which the SPARQL query is asked is indexed by relation types and by each argument of the relations. Hence there is a direct access to the list of relations of a given type which involve a given node. This graph indexing is a preliminary step of our algorithm; it is preprocessed and statically stored.

Based on this static index of G, we associate to each relation $r \in U(P)$ a set $candidates(r)$ of relations of $U(G)$ candidates for arguments of r to be projected on theirs: $candidate(r) = \{s \in U(G), type(s) \leq type(r)\}$. When a candidate s is elected, each i^{th} argument of r is projected on the i^{th} argument of s.

The backbone of our algorithm is the stack of the ordered relations of $U(P)$ associated to their candidate lists. Candidate lists initially correspond to the static index of G; we incrementally reduce their sizes as we pile them up according to the heuristic criterions described above. Their decreasing is as follows. Let r the current relation elected to be piled up. If it is connected to some relation r' previously piled up with the i^{th} argument of r being the j^{th} argument of r', then relations in $candidate(r)$ can be eliminated whose i^{th} argument does not appear as j^{th} argument in $candidate(r')$. Moreover, if some value constraint is evaluable once r is piled up, $candidate(r)$ is further decreased by eliminating candidates for which the constraint evaluates to false.

As a result, let $stack(P)$ the stack where all relations of $U(P)$ are piled up; it constitutes the search space for graph projection search.

Backjump Our algorithm incrementally search for a partial projection for nested subgraphs of P. To build these subgraphs we consider relations as they are ordered in $stack(P)$. This static ordering enables the handling of constraints during the projection search without ever and ever testing their evaluable status at each step of the algorithm, which would be too time consuming.

Based on this static ordering of relations defined by $stack(P)$, in case of failure of a partial projection search, our algorithm does not just systematically backtrack to the preceding relation in the stack but possibly goes to a deeper relation. It directly *backjumps* to the relation which solves the failure: the latest relation which binds (for the first time) one of the variables in the failing relation or the failing filter.

5 Conclusion

In this paper, we have presented the CORESE implementation of the SPARQL query language and its pattern matching mechanism. We reformulated the problem of answering SPARQL queries against RDF(S) data into a graph homomorphism checking and the CORESE algorithm takes advantage of the structure of graphs translating RDF(S) and SPARQL data and constrains graph homomorphism checking by SPARQL value constraints. Corese has proven its usability in a wide range of real world applications since 2000 [3]. Its implementation has widely evolved and it is now compliant with the core of SPARQL query language.

References

1. J.F. Baget, Simple Conceptual Graphs Revisited: Hypergraphs and Conjunctive Types for Efficient Projection Algorithms, In Proc. of the 11th ICCS, Dresden, Germany, 2003, LNCS 2746, Springer Verlag, pages 229-242.

2. O. Corby, R. Dieng, C. Faron, F. Gandon. Searching the Semantic Web: Approximate Query Processing based on Ontologies, IEEE Intelligent Systems 21(1), 2006.
3. R. Dieng-Kuntz, O. Corby. Conceptual Graphs for Semantic Web Applications, In Proc. of the 13th ICCS, Kassel, Germany, 2005, LNCS 3596, Springer-Verlag.
4. M. Croitoru, E. Compatangelo. A combinatorial approach to conceptual graph projection checking, In Proc. of the 24th International Conference of Specialist Group on Artificial Intelligence (AI'2004), Springer SBM, pages 130-143.