Preliminary results of a conceptual graph query language

David Genest, Marc Legeay, Stéphane Loiseau
Univ Angers, LERIA, SFR MATHSTIC, F-49000 Angers, France
david.genest@univ-angers.fr, marc.legeay@univ-angers.fr, stephane.loiseau@univ-angers.fr

Abstract

The contribution of this article is to bring the interesting features of SPARQL to the conceptual graph model while preserving its distinctive features and especially the graphical ones. Therefore this article presents a general conceptual graph query language, called CGQL. This article also provides an experimental comparison between CGQL and SPARQL engines: it shows the interest of our implementation for querying knowledge.

Introduction

The conceptual graph model (Sowa 1984; 1992) provides a way to represent knowledge using labelled graphs (Chein and Mugnier 2009) called *conceptual graphs*. The querying method of this model is based on the main operation of conceptual graphs, a graph homomorphism called *projection*: this operation determines if knowledge in a conceptual graph called *query graph* can be deduced from the knowledge base which is a conceptual graph called *factual graph*.

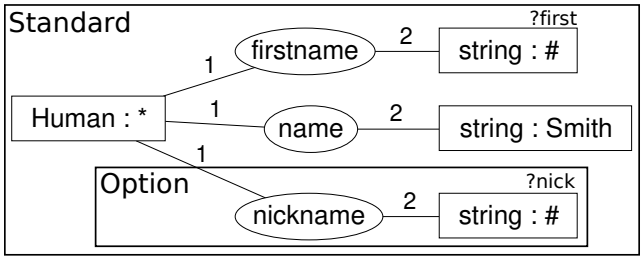
Semantic Web and conceptual graphs have similarities. Both aim at representing knowledge. In the Semantic Web, the query language (SPARQL) is based on the language for representing factual knowledge (RDF), whereas in conceptual graphs, graphs are used to represent queries and factual knowledge.

The availability of a simple but powerful query language, and its implementation in various tools is a part of the success of Semantic Web approaches. This paper proposes to combine the simplicity of visual representation from conceptual graphs with the powerful querying model of Semantic Web to improve conceptual graph query expressiveness. This paper proposes a query language for conceptual graphs, named *Conceptual Graph Query Language (CGQL)* and based on our paper (Genest et al. 2014), to answer the three lacks of conceptual graph querying. On one hand, CGQL is based on *expressive graph* to express query : it allows to express *disjunction* conditions (an "or" between two of its subgraphs) and *option* conditions (a subgraph is preferred but not necessarily required). On the other hand, an expressive graph enables to express constraints on values and extract relevant knowledge with a *filter* (an expression on values that have to be true). Our language is inspired by SPARQL, hence CGQL is composed of four query forms: `ask` query, `select` query, `describe` query and `construct` query.

In our paper, we briefly present the query language with one example, and we present the experiments performed to compare implementations of SPARQL and CGQL.

Query language

CGQL is a query language for conceptual graphs with four kind of queries, inspired by the ones of SPARQL: `ask`, `select`, `describe` and `construct`.



$sel = \{$?first, ?nick $\}$

Figure 1: `Select` query.

The `select` query of Figure 1 extracts the first name and the nickname, if it exists, of humans named Smith that have a first name and may have a nickname. The result of this query will be an association of named nodes (`?first` and `?nick`) of the query graph with the nodes of a factual graph *via* the projection of the query graph in the factual graph.

```
SELECT ?first ?nick
WHERE {
 ?h rdf:type ?th .
 ?th rdfs:subClassOf ns:Human .
 ?h ns:firstname ?first .
 ?h ns:name "Smith"^^xsd:string .
 OPTIONAL {
   ?h ns:nickname ?nick .
 }
}
```

Figure 2: Translation into SPARL of the CGQL `select` query of Figure 1

Figure 2 shows that the translation between CGQL and SPARQL can be automatized: a block in CGQL is translated into a section in SPARQL, a type of a concept node can be specified in SPARQL with the `rdf:type` and `rdfs:subClassOf` relations, and a relation in CGQL is a relation in SPARQL.

Experimentation

We developed an implementation of CGQL to demonstrate the viability of the approach. The implementation is based on classes and algorithms of Cogitant, an existing conceptual graph engine. Cogitant is a free (GPL license) C++ library (Genest 2018). It provides a complete implementation of the conceptual graph model, including the projection operation and some extensions such as datatypes. Since we got inspired by SPARQL, we compare ourselves with three SPARQL engines: Apache Fuseki[1]; Openlink Virtuoso[2] and Corese[3] (Corby and Faron-Zucker 2007).

We chose the SPARQL benchmark SP^{2}Bench (Schmidt et al. 2010) ; it is a well-known benchmark that has two interesting features: First, it comes with a free (BSD license) data generator which can produce set of RDF triples of an arbitrary size under the Turtle[4] syntax. Second, a set of 17 queries is provided, 14 of them are `select` queries, 3 are `ask` queries. The set of queries does not contain construct nor describe queries because they are build upon the core evaluation of select queries with a post-processing step (Schmidt et al. 2010). Some of them are simple (*q1, q2, q3a, q3b, q3c, q8, q9, q10, q11, q12a, q12b, q12c*) and some can be considered as more complex (*q4, q5a, q5b, q6, q7*) because they use various expressions of SPARQL such as union, option and filters. Our implementation parses these SPARQL queries and the RDF files, then translates them into our model. We generated a database with 2 million triples.

Tests have been done on an Intel Core i7-1165G7, 16 GB RAM under GNU/Linux Debian 11.6 with: Apache Fuseki version 4.7.0, as a standalone server, with OpenJDK Runtime Environment 11.0.18 ; Openlink Virtuoso (Open-source edition) version 7.2.8.3235 ; Corese version 4.3.0 (core), with OpenJDK 11.0.18 ; Our implementation based on Cogitant version 5.3.2, compiled with GNU C++ 10.2.1.

On the 2 million triples database (Table 1), our prototype competes with Virtuoso even if the execution times may vary, sometimes in favor of Virtuoso, sometimes in favor of our CGQL implementation and for most queries, our implementation is faster than Fuseki and Corese. For complex queries, our implementation is the fastest. For simple queries, the efficiency of indexes is the key. Also on this point Virtuoso is very efficient. The execution times are extremely variable depending on the queries, with a ratio of 24 to one (q12b: 0.03 vs 0.73). However the results of our CGQL implementation are good and it is the only tool without any execution time that exceeds 15 seconds.

[1]https://jena.apache.org/documentation/fuseki2/index.html

[2]https://virtuoso.openlinksw.com/

[3]https://project.inria.fr/corese/

[4]https://www.w3.org/TR/turtle/

	Results	Fuseki	Virtuoso	Corese	**CGQL**
q1	1	○ 0.10	● 0.00	0.07	0.05
q2	78651	7.25	○ 41.49	0.93	● 0.49
q3a	93973	0.59	○ 0.66	0.25	● 0.20
q3b	636	0.22	● 0.01	○ 0.26	0.04
q3c	0	0.20	● 0.00	○ 0.26	0.04
q4 ∗	5993155	○ –	61.77	○ –	● 12.31
q5a ∗	80740	○ –	1.10	○ –	● 0.70
q5b ∗	80740	○ 1.77	0.97	0.68	● 0.31
q6 ∗	142041	11.44	1.82	○ –	● 1.18
q7 ∗	565	○ –	2.32	○ –	● 0.78
q8	432	0.16	● 0.04	0.18	○ 0.72
q9	4	○ 1.14	● 0.35	0.98	0.69
q10	632	0.13	● 0.01	0.09	○ 0.30
q11	10	0.19	● 0.05	0.24	○ 0.44
q12a	1	○ 0.38	● 0.01	0.03	0.05
q12b	1	0.11	● 0.03	○ 0.10	○ 0.73
q12c	0	○ 0.11	● 0.00	● 0.00	● 0.00

Table 1: Execution times (in seconds) for 2 million triples. ● denotes the best result and ○ denotes the worst result. ∗ denotes a complex query.

The results of this experiment look rather encouraging. However, all these tools have been used in their "default" configuration, and especially for Virtuoso, the fine tuning of the engine can lead to far better performance. Moreover, Virtuoso can handle tens or hundreds of billions triples, because it has an efficient storage mechanism and powerful indexes, but our implementation of CGQL cannot handle such large bases. The goal of our implementation is not to compete with these tools, but to evaluate if the approach of this paper and its implementation based on graph projection is relevant in terms of efficiency.

Conclusion

We presented in this paper a new way to query conceptual graphs: CGQL. It is a language inspired by SPARQL but remains in the graph model since the query is graphical, the result is graphical and the query operations are performed thanks to graph operations. CGQL answers to three main lacks in the conceptual graph querying method: it is possible to express a "or", to filter values, and to extract specific knowledge from the answer of the query. It is, to our knowledge, the first general query language in the conceptual model that is powerful enough to be compared to SPARQL performances.

A formal comparison between SPARQL and CGQL is relevant. This comparison requires first to study similarities and differences between conceptual graphs and RDF, and then to compare SPARQL queries with CGQL queries. Our prototype provides a transformation of a subset of SPARQL into CGQL queries. We will pursue work in that direction to perform a detailed comparison between Semantic Web and conceptual graphs, both on a semantic level and on an power of interrogation level (i.e. SPARQL vs CGQL).

References

Chein, M., and Mugnier, M.-L. 2009. *Graph-based knowledge representation: computational foundations of conceptual graphs*. Springer.

Corby, O., and Faron-Zucker, C. 2007. Implementation of SPARQL query language based on graph homomorphism. In Priss, U.; Polovina, S.; and Hill, R., eds., *Conceptual Structures: Knowledge Architectures for Smart Applications, 15th International Conference on Conceptual Structures, ICCS 2007*, volume 4604 of *Lecture Notes in Artificial Intelligence*, 472–475. Springer.

Genest, D.; Legeay, M.; Loiseau, S.; and Béchade, C. 2014. A graphical language to query conceptual graphs. In *2014 IEEE 26th International Conference on Tools with Artificial Intelligence*, 304–308.

Genest, D. 2018. Cogitant 5.3.2 - reference manual. https://cogitant.sourceforge.io/files/cogitant.pdf.

Schmidt, M.; Hornung, T.; Meier, M.; Pinkel, C.; and Lausen, G. 2010. *SP2Bench: A SPARQL Performance Benchmark*. Springer Berlin Heidelberg. 371–393.

Sowa, J. F. 1984. *Conceptual Structures: Information Processing in Mind and Machine*. Addison-Wesley.

Sowa, J. F. 1992. Conceptual graphs. *Knowledge-Based Systems* 5(3):171 – 172. Conceptual Graphs Special Issue.