A Demonstration of a Natural Language Query Interface to an Event-Based Semantic Web Triplestore

Richard A. Frost*, Jonathon Donais*, Eric Mathews*, Wale Agboola*, and Rob Stewart**

*School of Computer Science, University of Windsor, Windsor, Canada
**Department of Computer Science, Heriot-Watt University, Edinburgh. U.K.

Abstract. Natural language semantic-web queries can be treated as expressions of the lambda calculus and evaluated directly with respect to an event-based triplestore using only basic triple retrieval operations. This facilitates the accommodation of complex NL constructs.

1 Introduction

Most semantic-web data sources contain sets of "entity-based" triples, e.g.:

```
<dbpedia:Al_Capone>  <dbpedia_owl:spouse>  <dbpedia:Mae_Capone> .
```

Many methods have been developed for querying entity-based triplestores, including: [11], [14], [15], [5], [17], [3], [2], [12], [9] and [6]. A good survey of work up to 2011 is given in [13]. Most of these methods convert the query to the SPARQL query language and then run the SPARQL query against the triplestore.

There are two difficulties with this approach. Firstly, consider a query with a simple prepositional phrase "in 1918":

```
"Who married Al Capone in 1918?"
```

Adding the following triple is insufficient as Capone could have married twice:

```
<..Al_Capone>  <..marriage_year>  <..1918>.
```

There are solutions to this problem which involve various forms of reification. However, most of these solutions appear to complicate translation of the NL query to SPARQL.

Another problem is the apparent difficulty of translating complex NL queries to SPARQL. Consider the following query:

```
"Who joined every gang that was joined by a person who stole a car
in 1899 or 1908 in Brooklyn?"
```

We are not aware of any approach, other than ours, that can accommodate such NL queries, which contain chained complex prepositional phrases containing arbitrarily-nested quantifiifiers (e.g. "a" and "every").

2 Event-based triplestores

Our proposed solution, called DEV-NLQ, is to represent data using a form of reification involving event-based triples, and to treat NL queries as expressions of the lambda calculus which are evaluated directly with respect to event-based triplestores. For example, in the following, `event1030` ties the data together:

```
<...event1030>   <...type>      <...marriage_ev> .
<...event1030>   <...subject>   <...Al_Capone> .
<...event1030>   <...object>    <...Mae_Capone" .
<...event1030>   <...year>      <..."1918"> .
```

The event-based triplestore that we use in our demo can be accessed at: http://speechweb2.cs.uwindsor.ca/ESWC/demo.html

3 Direct evaluation of NL queries

In [7] we describe a denotational semantics for natural-language query interfaces to event-based triplestores. Our semantics is based on an efficient version of Montague Semantics [4]. Our semantics accommodates proper and common nouns, adjectives, intransitive and transitive verbs, negation, and chained complex prepositional phrases containing arbitrarily-nested quantifiers.

The idea is that every word in English (after disambiguation by the parser) denotes a function. For example, in the following, `person`, `capone`, `a` etc. are functions defined in the Haskell programming language. The functions `getts_1` and `getts_3` are basic triple retrieval functions. See [7] for explanations. Note that `e => r` means that `r` is the result of evaluating `e`. Note also that we ignore URIs in the following, but address them in section 4.

```
e.g. getts_1 ("?", "subject", "torrio")   => ["event1009", "event1011"],
     getts_3 ("event1009", "type", "?")   => ["join_ev"]

get members set            = defined in terms of getts_1
get_subjs_of_event_type et = defined in terms of getts_1 and getts_3

gang    = get_members "gang"
          e.g.   gang          => ["fpg", "bowery"]

smoke   = get_subjs_of_event_type  "smoke_ev"
          e.g.   smoke         => ["capone"]

capone    setofents  = member  "capone"  setofents
          e.g.   capone smoke  => True

a       nph vbph     = length (intersect  nph vbph) /= 0
every   nph vbph     = subset nph vbph
no      nph vbph     = length (intersect  nph vbph) = 0
```

```
nounand s t          = intersect s t
that                 = nounand
nounor  s t          = mkset (s ++ t)

-- termand is a higher-order function which creates a new function from
the two functions given as input
termand tmph1 tmph2 = f  where
                    f setofents = (tmph1 setofents) && (tmph2 setofents)
         e.g.   (capone `termand` torrio) person => True

-- steal is a complex function (see Frost et al 2014 for the definition)
steal tmph preps = defined in terms of getts_1 and getts_3
         e.g. steal (a car) [("year", year_1899 `termor` year_1908),
                             ("location", brooklyn)]  => ["capone"]
```

Note that we can define the meaning of words in terms of others, e.g.

```
gangster = join (a gang)
```

4 Interfacing the query processor to the Semantic Web

We use the Haskell package `hsparql` [1] to interface our query processor to an external SPARQL endpoint containing our data. The functions `getts_1` and `getts_3` above are re-defined in terms of `hsparql` functions in a module called `Getts_4V`. All strings, such as `"capone"` in the definitions exemplified in section 3 are modified by a function `gts` to include a URI prefix. The Haskell code is available at the following URL:

`http://speechweb2.cs.uwindsor.ca/ESWC/src1`

Note that we do not translate the *whole* NL query to SPARQL. The `hsparql` functions only issue two types of basic SPARQL SELECT requests:

```
SELECT ?first WHERE {?first, <given_second>, <given-third>} .
SELECT ?third WHERE {<given_first>, <given_second>, ?third} .
```

Note that our semantics could be used with other non-SPARQL-endpoint interfaces to triplestores.

5 The demonstration

Readers can access our query interface as follows: 1) go to the Welcome page `http://speechweb2.cs.uwindsor.ca/ESWC/` which has three links: "Live Demo", "Source Code", and "Haskell Code ..", 2) `->` "Live Demo" `->` "List of triples in the Graph" to see how we represent data such as "Capone stole car_1 in 1908 in Brooklyn", 3) `->` Welcome page `->` "Source Code" `->` `gangster_v4.hs` which contains the Haskell definitions of the denotations of different words, 4)

-> Welcome page -> "Source Code" -> `Getts_v4.hs` which contains the code that links our semantics program to our external tripestore using the `h_sparql` module, 5) -> Welcome page -> "Live Demo" -> "Click here for more examples" which shows how brackets are placed in the queries according to their syntactic structure. Readers can copy and paste some of the examples into the "query" box on the "Live Demo" page and hit the "run query" button. Readers can also experiment with their own bracketed queries.

Queries are evaluated by our Haskell program in the same way as the expression `3 + (2 * 4)`. For example the query "Which gangster who stole a car in 1899 or 1908 in Brooklyn, joined a gang that was joined by Torrio?" can be entered into the query box as the following bracketed expression (note that you cannot cut and paste the expression from this .pdf document as the quotes are different in pdf).

```
which (gangster ‘that‘(steal’(a car)
                            [(gts "year",year_1899 ‘termor‘ year_1908),
                             (gts "location", brooklyn)]))
      (join (a (gang ‘that‘ (joined_by torrio))))
```

the following result is returned by our query interface:

```
["http://richard.myweb.cs.uwindsor.ca/ESWC/gangster_triplestore#capone"]
```

6 Concluding comments

We are currently integrating the semantics with our NL parser [8] which will introduce the brackets according to the syntactic structure of the query.

Our approach assumes that the full URIs are known (and used in the definitions of the denotations of words). We intend to investigate the integration of the method of Walter et al [16] for mapping query words to appropriate URIs and building the denotations of words in real-time when the query is parsed.

Our approach assumes the existence of event-based triplestores. We are currently investigating how to extract sets of event-based triples from conventional triplestores as required. However, we also note that triplestores are being developed to accommodate "richer" contextual data. YAGO2 is an example [10] which uses a simple form of reification to represent temporal and spatial properties. We are developing another denotational semantics so that NL queries can be evaluated directly with respect to YAGO2 data.

Because there is no need to translate NL queries to a formal query language such as SPARQL, F-Logic or SPOTLX, we can concentrate solely on linguistic issues and develop semantics to accommodate highly-complex NL constructs.

7 Acknowledgements

The authors acknowledge the Natural Science and Engineering Council of Canada (NSERC), and the reviewers who provided very useful constructive criticism.

References

1. The hsparql package. http://hackage.haskell.org/package/hsparql-0.1.2 Author: Jeff Wheeler, Maintained by: Rob Stewart.
2. D. Damljanovic, M. Agatonovic, and H. Cunningham. Freya: An interactive way of querying linked data using natural language. In *The Semantic Web: ESWC 2011 Workshops*, pages 125–138. Springer, 2012.
3. M. Damova, D. Dannelles, R. Enache, M. Mateva, and A. Ranta. Natural language interaction with semantic web knowledge bases and lod. In *Towards the Multilingual Semantic Web*. Springer, 2013.
4. D. Dowty, R. Wall, and S. Peters. *Introduction to Montague Semantics*. D. Reidel Publishing Company, Dordrecht, Boston, Lancaster, Yokyo, 1981.
5. S. Ferre. Squall: A controlled natural language for querying and updating rdf graphs. In *Proceedings of CNL 2012*, pages 11–25. LNCS 7427, 2012.
6. A. Freitas, F. F. de Faria, S. O'Riain, and E. Curry. Answering natural language queries over linked data graphs: a distributional semantics approach. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 1107–1108. ACM, 2013.
7. R. A. Frost, W. Agboola, and E. Matthews. Querying graph-structured data using natural language. In *Proc GraphQ Workshop EDBT/ICDT 2104*, pages 192–199, 2014.
8. R. Hafiz and R. Frost. Lazy combinators for executable specifications of general attribute grammars. In *Proceedings of the 12th International Symposium on Practical Aspects of Declarative Languages (PADL)*, pages 167–182. ACM-SIGPLAN, Jan. 2010.
9. S. Hakimov, H. Tunc, M. Akimaliev, and E. Dogdu. Semantic question answering system over linked data using relational patterns. In *Proc. of the Joint EDBT/ICDT 2013 Workshops*, pages 83–88. ACM, 2013.
10. J. Hoffart, F. M. Suchanek, K. Berberich, and G. Weikum. Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial Intelligence*, 194:28–61, 2013.
11. E. Kaufmann and A. Bernstein. Evaluating the usibility of natural language query languages and interfaces to semantic web knowledge bases. *Web Semantics - Science, Services and Agents on the World Wide Web*, 8(4):377–393, Nov 2009.
12. V. Lopez, M. Fernández, E. Motta, and N. Stieler. Poweraqua: Supporting users in querying and exploring the semantic web. *Semantic Web*, 3(3):249–265, 2012.
13. V. Lopez, V. Uren, M. Sabou, and E. Motta. Is question answering fit for the semantic web?: a survey. *Semantic Web*, 2(2):125–155, 2011.
14. A. Ran and R. Lencevicius. Natural language query system for rdf repositories. In *Proceedings of the 7th International Symposium on Natural Language processing*, pages 1–6. SNLP, 2007.
15. C. Unger and P. Cimiano. Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. In *NLDB 2011, LNCS 6716*, pages 153–160, 2011.
16. S. Walter, C. Unger, P. Cimiano, and D. Bär. Evaluation of a layered approach to question answering over linked data. In *The Semantic Web–ISWC 2012*, pages 362–374. Springer, 2012.
17. M. Yahya, K. Berberich, S. Elbassuoni, M. Ramanath, V. Tresp, , and G. Weikum. Natural language questions for the web of data. In *The 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 379–390. ACL, July 2012.