An Event-based Approach for Querying Graph-Structured Data Using Natural Language

GraphQ 2014

Richard Frost , Wale Agboola, Eric Matthews and Jon Donais
School of Computer Science
University of Windsor

Natural-Language Interfaces to triplestores
Translate NL to SPARQL – Problem 1

“Who married Al Capone in 1918?” prepositional phrase

⇩ translation

SPARQL QUERY

↕ SPARQL endpoint

<http://dbpedia.org/resource#Al_Capone>
<http://dbpedia.org/ontology/spouse
<http://dbpedia.org/resource#Mae_Capone> .

How do we add the data representing the prepositional phrase?

<….Al_Capone> <year_married> <…1918> .
Not adequate as Capone could have married twice.

ANSWER: Use some version of reification
BUT complicates translation from NL to SPARQL

Natural-Language Interfaces to triplestores
Translate NL to SPARQL – Problem 2

"Who joined every gang that was joined by Torrio and stole a car in 1918 or 1920 in a borough of New York?"

⇩

SPARQL QUERY

Too complicated – chained complex prepositional phrases with arbitrarilty-nested quantifiers?

We are unaware of any system that can do this.

ANSWER: Do not use SPARQL

Evaluate the NL queries directly w.r.t. the triplestore

A Solution to Problem 1
Event-Based Triplestores – a form of reification

```
[("event1000","type",      "born_ev"),
 ("event1000", "subject", "capone"),
 ("event1000", "year",     "1899"),
 ("event1000", "location","brooklyn"),
 ("event1001", "type",     "join_ev"),
 ("event1001", "subject", "capone"),
 ("event1001", "object",   "fpg"),
 ("event1002", "year",     "1908" ),
 ("event1004", "type",     "steal_ev"),
 ("event1004", "subject", "capone"),
 ("event1004", "object",   "car_1"),
 ("event1004", "year",     "1908"),
 ("event1004", "location","brooklyn"),
("event1005", ”type",     “smoke_event"),
("event1006", "type",     "membership"),
("event1006", "subject", "car_1"),
("event1006", "object",   "car"),
("event1007", "type",     "membership"),
("event1007", "subject", "fpg"),
("event1007", "object",   "gang"),
("event1005", “subject”, “capone”),
("event1010", "subject", "capone"),
("event1010", "object",   "person"),
("event1010", "type",     "membership"),
("event1011", "subject", "torrio"),
 etc.
```

“Who stole a car in 1918 or 1920 in a borough of New York?”

⇩ parser ⇩

“Who (stole (a car) [(in (1918 or 1920), in (a (borough (located_in New_York)))])?”

⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩ ⇩

λ… (λ… (λ… λ…) [(λ… (λ… λ… λ…), λ …(λ…(λ… (λ… λ…))))])

⇧ ⇧ ⇧ ⇧

TRIPLESTORE

Basic Triplestore Retrieval Functions

getts_1 (ANY, REL “subject”, ENT “capone”)

=> {(1000, REL “subject”, ENT “capone”),

(1001, REL “subject”, ENT “capone”), etc.

getts_3 similar.

getts_1 and getts_3 are used to define other basic retrieval operators.

Defs in the paper..

Example uses of other retrieval operators:

get_subjs_for_events {EV 1000, EV 1009} => {ENT "capone", ENT "torrio"}

get_members “thief_set” => {ENT “capone"}

get_subjs_of_event_type “born_ev” => {ENT “capone”}

We can now define semantics using these basic operators

Definitions of the denotations of Words

We use the notation of set theory in place of lambda expressions

thief = get_members “thief_set"

e.g. **thief** => {ENT “capone”}

smokes = get_subjs_of_event_type “smoke_ev”

e.g. **smokes** => {ENT “capone”}

capone setofents = (ENT "capone") ∈ setofents

e.g. **capone** **smokes** => True

a nph vbph = #(nph ⋂ vbph) ~= 0

term_and tmph1 tmph2 = f where
f setofevs = (tmph1 setofevs) & (tmph2 setofevs)

e.g. **((a thief) $term_and capone) smokes** => True

Our new semantics – major contribution 1
An explicit definition of the denotation of transitive verbs

join   tmph

= { subj | (subj, evs) ∈   (make_image   join_event)

&

tmph (⋃ {map thirds (getts (ev, REL "object", ANY)) | ev ∈ evs})}

where, for example: make_image   "join_ev"

=> {(ENT "capone", {EV 1001, EV 1003}), (ENT "torrio",   {EV 1009})}

e.g.   **join (a gang)** => {ENT "capone", ENT "torrio"}

Major contribution 2: Prepositional phrases

Simplified example – a single prepositional phrase

steal_with_time tmph date
= {subj | (subj, evs) ∈ image_steal &
tmph (⋃ {thirds (getts (ev,REL"object",ANY))
| ev ∈ evs
&
date(thirds (getts (ev,REL "date", ANY)))})}

The date argument is used to “filter” the events.

e.g. steal_with_time (a car) (date_1918) => {ENT "capone"}

The result: A wide range of English NL queries

We have implemented simple case of prepositional phrases in Haskell – with an in-program triplestore.

e.g. “Which gangster who stole a car in 1915 or 1918 joined a gang that was joined by Torrio?”

⇩ **manual insertion of brackets**

which (gangster $that (steal_with_time (a car)
(date_1915 $term_or date_1908))
(join (a (gang $that (joined_by torrio))))

⇩

{ENT “capone”}

Next steps

1. Extend implementation to include chained prepositional phrases such as: "who stole a car in Brooklyn in 1908 or 1915" (our solution is briefly described in the paper) (have done this)

2. Deploy our triples on the semantic web and access them through a SPARQL endpoint using the basic retrieval operators (which only use basic SPARQL SELECT operations (have done this)

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

3.Integrate the semantics with our X-SAIGA NL parser.

(have nearly completed this)

4.Interface our query processor with our speech browser.

(have nearly completed this)

⇩

An NL speech query interface to semantic web data

References to our previous work

SEMANTICS: R. A. Frost, B. S. Amour, and R. Fortier. An event based denotational semantics for natural language queries to data represented in triple stores. *IEEE ICSC, 2013.* IEEE, 142-145.

Frost, R. A. and Fortier, R. (2007) An efficient denotational semantics for natural language database queries, *NLDB 07*, LNCS 4592, 12-24.

Frost, R. A. (2006) Realization of natural language interfaces using lazy functional programming. ACM Comp. Surv. 38 (4) Article 11.

PARSING: Hafiz, R. and Frost, R, (2010) Lazy combinators for executable specifications of general attribute grammars, *Proc. of the 12th International Symposium on Practical aspects of declarative languages (PADL),* LNCS 5937, 167-182.

Frost, R., Hafiz, R., Callaghan, P., (2007) Modular and efficient top-down parsing for ambiguous left-recursive grammars. In 10th ACL, IWPT, 109–120.

SPEECH RECOGNITION: Frost, R. A., Ma, X. and Shi, Y. (2007) A browser for a public-domain SpeechWeb. *WWW 2007*, 1307-1308.

Frost, R. A. (2005). A call for a public-domain SpeechWeb. *CACM* 48 (11) 45-49.

Acknowledgements

Rahmatullah Hafiz

Paul Callaghan

Nabil Abdullah

Ali Karaki

Paul Meyer

Matthew Clifford

Shane Peelar

Stephen Karamatos

Walid Mnaymneh

Rob Mavrinac

Cai Filiault

NSERC – Natural Science and Engineering Council of Canada

Research Services - University of Windsor