A New Data Structure for Processing Natural Language Database Queries

Richard A. Frost[a] and Shane Peelar[b]
School of Computer Science, University of Windsor, 401 Sunset Avenue, Windsor, Ontario, Canada

Keywords: Natural Language Processing, Natural Language Query Interfaces, Compositional Semantics, Event Semantics, Quantification.

Abstract: Natural Language Query Interfaces (NLQIs) have once again captured the public imagination, but developing them has proven to be non-trivial. One way is by using a Compositional Semantics (CS) to directly compute the answer to a query from the meanings of its parts. The query is treated as an expression of a formal language, and interpreted directly with respect to a database which provides meanings for words, which are the basic components. The meanings of compound phrases in the query, and the answer to the query itself, are computed from their constituent words and phrases using semantic rules that are applied according to the query's syntactic structure. Montague Semantics (MS), which is a type of CS, has been used in various NLQIs previously. MS accommodates common and proper nouns, adjectives, conjunction and disjunction, intransitive and binary transitive verbs, quantifiers, and intentional and modal constructs. MS does not provide an explicit denotation for *n*-ary transitive verbs nor does it provide an explanation of how to handle prepositional phrases. By adding events to MS and by introducing a new data structure, transitive verbs and prepositional phrases can be accommodated as well as other NL features that are often considered to be non-compositional.

1 INTRODUCTION

We begin by describing a Natural Language Query Interface (NLQI) that we have built. We hope that the interface will motivate readers to look into our modifications to MS. In Section 2, we explain how our NL Query interface (NLQI) can be accessed through the Web. In Sub-section Section 2.1, we describe the Semantic Web triplestore. In Section 3 we discuss example queries and their results: in Section 3.2, we provide examples of what are often referred to as "non-compositional" features of NL that our NLQI can handle, and in Section 3.3 we give examples of NL structures that could be accommodated by extensions to our approach. With each of the examples we provide an informal explanation of how the answer is, or could be, computed.

In Section 4, we describe the new FDBR data structure which is central to our approach, and which can be created from an event-based triplestore (as we do in our online NLQI), or from a relational database.

Much of our semantics is based on MS. We differ in these ways:

1. We add events to the basic ontological concepts of entities and truth values.
2. Each event has a number of roles associated with it. Each role has an entity as a value.
3. For efficiency, we use sets of entities rather than characteristic functions of those sets as is the case in MS.
4. We define transitive *n*-ary verbs in terms of sets of events, each with *n* roles.
5. We compute FDBRs, the novel datastructure presented in this paper, from sets of events (could be computed from relations), and use them in the denotations of transitive verbs, and in computing results of queries containing prepositional phrases. Although not referred to as an FDBR, the use of relational images in denotations of verbs was first proposed by (Frost and Launchbury, 1989).

We hope that this paper reawakens an interest in Compositional Semantics, in particular for NL query processing.

[a] https://orcid.org/0000-0001-7083-5060
[b] https://orcid.org/0000-0001-7391-0951

Frost, R. and Peelar, S.
A New Data Structure for Processing Natural Language Database Queries.
DOI: 10.5220/0008124300800087
In *Proceedings of the 15th International Conference on Web Information Systems and Technologies (WEBIST 2019)*, pages 80-87
ISBN: 978-989-758-386-5

Table 1: Events of type "Discover". The full URIs of the events, properties, and entities have been omitted here.

Event	Property	Entity
event1045	subject	hall
event1045	object	phobos
event1045	type	discover_ev
event1045	year	1877
event1045	location	us_naval_observatory
event1045	implement	refractor_telescope_1

Table 2: Events of type "Membership".

Event	Property	Entity
event1128	subject	galileo
event1128	object	person
event1128	type	membership

2 HOW TO ACCESS OUR NLQI

Our NL interface can be accessed at the following web site:

http://speechweb2.cs.uwindsor.ca/solarman4/demo_sparql.html

2.1 The Triplestore that is Queried

Our NLQI computes answers with respect to a triplestore containing data about the planets, the moons that orbit them, and the people who discovered those moons, and when, where and with what implement they were discovered. Note that each triple could be equally well be represented by a row in a relational database.

The triplestore contains triples such as the following which represent the event #1045 in which `hall` (in the role of *subject*) discovered `phobos` (in the role of *object*) in 1877 (in the role of *year*) with the `refractor_telescope_1` (in the role of *implement*) at the `us_naval_observatory` (in the role of *location*).

Events representing set membership are represented as follows:

The complete triplestore, which contains tens of thousands of triples, is hosted on a remote compute server using the Virtuoso software (Erling and Mikhailov, 2010) and can be accessed by following the link at the beginning of Section 2.

3 EXAMPLE QUERIES

Our NLQI can answer millions of queries with respect to the triplestore discussed above. The NLQI can accommodate queries containing common and proper nouns, adjectives, conjunction and disjunction, intransitive and transitive verbs, nested quantification, superlatives, chained prepositional phrases containing quantifiers, comparatives and polysemantic words. In the following we provide an informal explanation of how the answer is computed.

3.1 Queries Demonstrating the Range of NL Features that our NLQI can Accommodate

`phobos spins` ⇒ *True*
`phobos is a moon` ⇒ *True*

The function denoted by `phobos` checks to see if e_{phobos} is a member of the *spin* set, and secondly if e_{phobos} is a member of the *moon* set.

`a moon spins` ⇒ *True*
`every moon spins` ⇒ *True*
`an atmospheric moon exists` ⇒ *True*

The function denoted by `a` checks to see if the intersection of the set of *moons* and the set of *spins* is non-empty. The function denoted by `every` checks to see if the set of moons is a subset of the *spins* set. The denotations of `a` and `every` that we use are set-theoretic event-based versions of the denotations from MS which uses characteristic functions.

The answer to the third query is obtained by checking if the intersection of the *atmospheric* set and the *moon* set is non-empty.

`hall discovered` ⇒ *True*

All of the events of type "discovered" are collected together and are checked to see if e_{hall} is found as the subject role value of any of them. If so, *True* is returned.

`when did hall discover` ⇒ *1877*

The "year" property of the events returned by "hall discover" (treated as "hall discovered") are returned.

`phobos was discovered` ⇒ *True*

All of the events of type "discovered" are collected together and are checked to see if e_{phobos} is found as the object role value of any of them. If so *True* is returned.

`earth was discovered` ⇒ *False*

Earth was not discovered by anyone, according to our data.

`did hall discover phobos` ⇒ *True*

All of the events of type "discover" are collected together and are checked to create a pair (s, evs) for each value of the subject attribute found in the set of events. *evs* is the set of events to which the subject attribute is related through a discovery event. Each pair is then examined to see if the function denoted by the object termphrase (in this case `phobos`) returns a non-empty set when applied to a set (called an *FDBR*, which is described later) generated from the set of *evs* in the pair, and if so the subject of the pair is added to the set which is returned as the denotation of the verbphrase part of the query. The denotation of the termphrase at the beginning of the query is then applied to the denotation of the verbphrase to obtain the answer to the query.

Owing to the fact that our semantics is compositional the *subject* and *object* termphrases of the query above can be replaced by any termphrases. For example:

`a person or a team discovered every moon that orbits mars` ⇒ *True*
`who discovered 2 moons that orbit mars` ⇒ *hall*

"who", "what", "where", "when" and "how" can be used in place of the *subject* termphrase. Different role values are returned depending on which "wh.." word is used in the query.

`where discovered by galileo` ⇒ *padua*
`when discovered by galileo` ⇒ *1610*
`every telescope was used to discover a moon` ⇒ *True* (w.r.t.our data)
`a moon was discovered by every telescope` ⇒ *False*
`a telescope was used by hall to discover two moons` ⇒ *True*
`which moons were discovered with two telescopes` ⇒ *halimede laomedeia sao themisto*
`who discovered deimos with a telescope that was used to discover` ⇒ *hall*
`every moon that orbits mars` ⇒ *nicholson science_team_18 science_team_2*
`who discovered a moon with two telescopes` ⇒ *blanco_telescope canada-france-hawaii_telescope*
`how was sao discovered` ⇒ *refractor_telescope_1*
`how discovered in 1877` ⇒ 2
`how many telescopes were used to discover sao` ⇒ *science_team_18*
`who discovered sao` ⇒ *blanco_telescope canada-france-hawaii_telescope*
`how did science_team_18 discover sao` ⇒ *saturn; none*
`which planet is orbited by every moon that was discovered by two people` ⇒ *hall; none*
`which person discovered a moon in 1877 with every telescope that was used to discover phobos` ⇒ *kuiper*
`who discovered in 1948 and 1949 with a telescope`

3.2 Queries with "Non-compositional" Structures

We agree that natural language has non-compositional features but believe that the non-compositionality is mostly problematic when the objective is to give a meaning to an NL expression (without a context). It is less problematic when answering NL queries. As illustrated below, the person posing the query, or the database or triplestore can provide contexts that help resolve much of the ambiguity resulting from non-compositional features.

The advantages of a using a compositional semantics include 1) the answer to a query is as correct as the data from which it is derived, 2) the meaning of sub phrases within a query can be discussed formally, 3) the query language can be extended such that all existing phrases maintain their original meanings, 4) the definition of syntax and semantics in the compositional semantics can be used as a blueprint for the implementation of the query processor.

Some researchers have provided examples of what they claim to be non-compositional structures in NL. For example, Hirst (Hirst, 1992) gives the example of the verb "depart" which he states is not compositional because its meaning changes with the prepositional phrase(s) which follow it, and that the definition of compositionality needs to be modified to include the requirement that the function used to compose the meaning of parts must be systematic. We claim that our semantics for verbs is systematic as the denotations of subject and object termphrases,

and the possibly empty list of prepositional phrases following the verb are treated equally and are all used in the same way to filter the set of events of the type associated with the verb, before that set is returned as the denotation of the verb phrase. This is illustrated in the following queries:

`who discovered` ⇒ *bernard bond cassini cassini_imaging_science_team christy dollfus galileo etc...*

No *subject*, *object* or prepositional phrase is given in the query, and so all events of type "discover" are returned by the verbphrase and the denotation of the word `who` picks out the *subjects* from those events.

`where discovered io` ⇒ *padua*

No *subject*, or prepositional phrase is given in the query, and so all events of type "discover" are considered and filtered by the denotation of the *object* termphrase `io` and then, those that pass the filter are returned by the verbphrase and the word `where` picks out the location from those events.
`who discovered in 1610` ⇒ *galileo*

No *subject* or *object* is in the query so all events of type "discover" are considered and only those with attribute "date" equal to 1610 pass the filter and then the denotation of the word `who` selects the subject which is returned.

In our semantics, the *subject* and *object* termphrases are treated as filters as are all prepositional phrases, as shown in the following example:

`who discovered every moon that orbits mars with one telescope or a moon that orbits Jupiter with a telescope` ⇒ *one. ; none. ; none. ; bernard galileo kowal melotte nicholson perrine science_team_1 science_team_2 ; hall ; hall ; none.*

Several results are returned because the query is syntactically ambiguous.

`where discovered in 1610` ⇒ *padua*
`how discovered in padua` ⇒ *galilean_telescope_1*

3.3 Extensions to the Semantics

Some phrases containing nested quantifiers are given by some researchers, as examples of non-compositionality. For example: "a US diplomat was sent to every capital" is often read as having two meanings which can only be disambiguated by additional knowledge. We argue that the person posing a query can express the query unambiguously if they are familiar with quantifier scoping conventions used by our processor, as illustrated in the following:

`christy or science_team_19 or science_team_20 or science_team_21 discovered every moon that orbits pluto` ⇒ *False*

In our semantics, quantifier scoping is always leftmost/outermost, and an unambiguous query can be formulated as follows:

`every moon that orbits pluto was discovered by christy or science_team_19 or science_team_20 or science_team_21` ⇒ *True*

Some examples of non-compositionality involve polysemantic superlative words such as "most" in, for example:

"Who discovered most moons that orbit P. Where P is a planet."

If "most" is treated as "more than half" then:

`who discovered most moons that orbit mars` ⇒ *hall*

Because our semantics currently allows only this reading. However, the answer to the alternate reading "who discovered the most moons that orbit *P*" – i.e. more than anyone else who discovered a moon that orbits *P*. Could be obtained in our semantics by comparing all of the (ent, evs) pairs returned by the verbphrase to see which *subject* is paired with most *objects*. We are currently working on this and other extensions to our semantics.

`how was every moon that orbits saturn discovered` ⇒ *cassini reflector_telescope_1 aerial_telescope_1 refractor_telescope_4 etc...*

It may be surprising that *cassini* is returned in the answer since it is not a `telescope`, but is instead a `spacecraft`. However, since it was used to discover at least one `moon` that orbits `saturn`, it is considered to have fulfilled the *implement* role and is encoded as such in the triplestore.

4 THE FDBR: A NOVEL DATA STRUCTURE FOR NATURAL LANGUAGE QUERIES

4.1 Montague Semantics

All quantifiers, such as "a", "every" and "more than two" are treated in MS as functions which take two characteristic functions of sets as arguments and return a Boolean value as result. Our modifications to MS are to use sets of entities instead of predicates/characteristic functions of those sets, and to pair sets of events with each entity; the set of events paired with an entity justify the entity's inclusion in the denotation. For example:

$$\|propernoun\| = \lambda p.\{(e,evs) \mid (e,evs) \in p \;\&\; e = \text{the entity associated with the proper noun}\}$$

$$\|spins\| = \{(e_{phobos},\{ev_{1360}\}),(e_{deimos},\{ev_{1332}\}), etc\ldots\}$$

Therefore,

$$\begin{aligned}\|phobos\ spins\| &= \|phobos\|\ \|spins\| \\ &= \lambda s.\ \{(e,evs)|(e,evs) \in s \;\&\; e = e_{phobos}\} \\ &\quad \{(e_{phobos},ev_{1360}),(e_{deimos},ev_{1332}),\ldots\} \\ &= \{(e_{phobos},ev_{1360})\}\end{aligned}$$

$$\|a\| = \lambda m \lambda s.\ \{(e_1,evs_2) \mid (e_1,evs_1)) \in m \;\&\; (e_2,evs_2) \in s \;\&\; e_1 = e_2\})$$

$$\|a\ moon\ spins\| = \{(e_{phobos},ev_{1360}), (e_{deimos},ev_{1332}),\ etc\ldots\}$$

Note that the events *evs* paired with the entities returned in the denotation of "was every moon that orbits Saturn discovered" are the events representing membership of those entities of type moon in the object value of events of type discovery. This enables additional data to be accessed from those events, as illustrated in the last example query in the previous section.

4.2 The FDBR

In order to generate the answer to "hall discovered every moon that orbits mars", $\|every\|$ is applied to $\|moon\ that\ orbits\ mars\|$ (i.e. the set of `moons` that orbit `mars`), as first argument, and the set of entities that were discovered by `hall`, as the second argument. Our semantics generates this set from the set of events of type "discover" where the subject role is the entity associated with `hall`, as discussed below: Every set of n-ary events (i.e. events with n roles) of a given type, e.g. discovery, defines $n^2 - n$ binary relations. For example, for discovery events:

$discover\_rel_{subject \to object}$ $discover\_rel_{subject \to year}$
$discover\_rel_{subject \to implement} \ldots$
$discover\_rel_{object \to subject}$ $discover\_rel_{object \to year}$
$discover\_rel_{object \to implement} \ldots$
$discover\_rel_{year \to subject}$ $discover\_rel_{year \to object}$
$discover\_rel_{year \to implement} \ldots$

$etc\ldots$ to 20 binary relations for the set of discovery events or an 5-ary discovery relation. For example:

$$discover\_rel_{subject \to object} = \{(ev_{1045},e_{hall},e_{phobos}),(ev_{1046},e_{hall},e_{deimos}),etc\ldots\}$$

If we collect all of the values from the range of a relation that are mapped to by each value v from the domain (i.e. the image of v under the relation r) and create the set of all pairs $(v, image\_of\_v)$, we obtain a function defined by the relation r, i.e. the FDBR. For example:

$$\text{FDBR}(discover\_rel_{subject \to object}) = \{(e_{hall},\{(e_{phobos},\{ev_{1045}\}),(e_{deimos},\{ev_{1046}\})\}), etc\ldots\}$$

It is these functions that are created, and used, by the denotation of the transitive verb associated with the type of the events. For example in calculating the value of $\|who\ discovered\ every\ moon\ that\ orbits\ mars\|$, $\|every\|$ is applied to the set of entities which is the denotation of "moon that orbits mars" (i.e $\{(e_{phobos},\{ev_{1045}\}),(e_{deimos},\{ev_{1046}\})\}$) and all of the images that are in the second field of the pairs in $\text{FDBR}(discover\_rel_{subject \to object})$. For the pair $(e_{hall},\{(e_{phobos},\{ev_{1045}\}),(e_{deimos},\{ev_{1046}\})\})$, $\|every\|$ returns the non-empty set $\{(e_{phobos},\{ev_{1045}\}),(e_{deimos},\{ev_{1046}\})\}$, and the value in the first field, i.e. e_{hall}, is subsequently returned with the answer to the query.

The various FDBRs are used to answer different types of queries. For example:

`who discovered phobos and deimos` $\Rightarrow$ *hall*
uses $\text{FDBR}(discover\_rel_{subject \to object})$
`where discovered by galileo` $\Rightarrow$ *padua*
uses $\text{FDBR}(discover\_rel_{location \to subject})$
`how discovered in 1610 or 1855`
$\Rightarrow$ *galilean_telescope_1*
uses $\text{FDBR}(discover\_rel_{implement \to year})$

5 HANDLING PREPOSITIONAL PHRASES

Prepositional phrases (PPs) such as "with a telescope" are treated similarly to the method above, except that the termphrase following the preposition is applied to the set of entities that are extracted from the set of events in the FDBR function, according to the role associated with the preposition. The result is a "filtered" FDBR which is further filtered by subsequent PPs.

6 QUANTIFIERS AND EVENTS

In 2015, Champollion (Champollion, 2015) stated that, at that time, it was generally thought by linguists that integration of Montagovian-style compositional semantics and Davidsonian–style event semantics (Parsons, 1990; Davidson, 1967) was problematic, particularly with respect to quantifiers. Champollion did not agree with that analysis and presented an integration which he called "quantificational event semantics" which he claimed solved the difficulties of integration by assuming that verbs and their projections denote existential quantifiers over events and that these quantifiers always take lowest possible scope.

In this paper, we borrow much from Montague Semantics (MS), Davidsonian Event Semantics, and Champollion's Quantificational Event Semantics. However, we provide definitions of our denotations in the notation of set theory, which improves computational efficiency and, we believe, simplifies understanding of our denotations. We also believe that our semantics is intuitive, systematic, and compositional.

7 OUR APPROACH WITH RELATIONAL DATABASES

Our NLQI could be easily adapted for use with conventional relational databases. Each row in a relation *Rel* can be thought of as representing an event of type *Rel*, and each column name can be thought of as a role name. The event itself would serve as the primary key, and only the triple retrieval function would need to be modified. This architecture allows the denotations to remain unchanged and yet still work with different types of databases.

8 IMPLEMENTATION OF OUR NLQI

We built our query processor as an executable attribute grammar using the X-SAIGA Haskell parser-combinator library package (Frost et al., 2008). The *collect* function which converts a binary relation to an FDBR is one of the most compute intensive parts of our implementation of the semantics. However, in Haskell, once a value is computed, it can be made available for future use. We have developed an algorithm to compute FDBR(rel) in $O(n\ log\ n)$ time, where n is the number of pairs in *rel*. Alternatively, the FDBR functions can be computed and stored in a cache when the NLQI is offline. Our implementation is amenable to running on low power devices, enabling it for use with the Internet of Things. A version of our query processor exists that can run on a common consumer network router as a proof of concept for this application. The use of Haskell for the implementation of our NLQI has many advantages, including:

1. Haskell's "lazy" evaluation strategy only computes values when they are required, enabling parser combinator libraries to be built that can handle highly ambiguous left-recursive grammars in polynomial time.
2. The higher-order functional capability of Haskell allows the direct definition of higher-order functions that are the denotations of some English words and phrases.
3. The ability to partially apply functions of n arguments to 1 to n arguments allows the definition and manipulation of denotation of phrases such as "every moon", and "discover phobos".
4. The availability of the *hsparql* (Wheeler, 2009) Haskell package enables a simple interface between our semantic processor and SPARQL endpoints to our triplestores.

9 RELATED WORK

Orakel (Cimiano et al., 2007) is a portable NLQI which uses a Montague-like grammar and a lambda calculus semantics. Our approach is similar in this respect. Queries are translated to an expression of first order logic enriched with predicates for query and numerical operators. These expressions are translated to SPARQL or F-Logic. Orakel supports negation, limited quantification, and simple prepositional phrases.

YAGO2 (Hoffart et al., 2013) is a semantic knowledge base containing reified triples extracted from

Wikipedia, WordNet and GeoNames, representing nearly 0.5 billion facts. Reification is achieved by tagging each triple with an identifier. However, this is hidden from the user who views the knowledge base as a set of "SPOTL" quintuples, where T is for time and L for location. The SPOTLX query language is used to access YAGO2. SPOTLX can handle queries with prepositional aspects involving time and location. However, no mention is made of chained complex PPs.

Alexandria (Wendt et al., 2012) is an event-based triplestore, with 160 million triples (representing 13 million n-ary relationships), derived from FreeBase. Alexandria uses a neo-Davidsonian (Parsons, 1990) event-based semantics. In Alexandria, queries are parsed to a syntactic dependency graph, mapped to a semantic description, and translated to SPARQL queries containing named graphs. Queries with simple PPs are accommodated. However, no mention is made of negation, nested quantification, or chained complex PPs.

The systems referred to above have made substantial progress in handling ambiguity and matching NL query words to URIs. However, they appear to have hit a roadblock with respect to natural-language coverage. Most can handle simple PPs such as in "who was born in 1918" but none can handle chained complex PPs, containing quantifiers, such as "in us_naval_observatory in 1877 or 1860".

Blackburn and Bos (Blackburn and Bos, 2005) implemented lambda calculus with respect to natural language, in Prolog, and (Van Eijck and Unger, 2010) have extensively discussed such implementation in Haskell. Implementation of the lambda calculus for open-domain question answering has been investigated by (Ahn et al., 2005). The SQUALL query language (Ferre, 2012; Ferré, 2013) is a controlled natural language (CNL) for querying and updating triplestores represented as RDF graphs. SQUALL can return answers directly from remote triplestores, as we do, using simple SPARQL-endpoint triple retrieval commands. It can also be translated to SPARQL queries which can be processed by SPARQL endpoints for faster computation of answers. SQUALL can handle quantification, aggregation, some forms of negation, and chained complex prepositional phrases. It is also written in a functional language. However, some queries in SQUALL require the use of variables and low-level relational algebraic operators (see for example, the queries on page 118 of (Ferré, 2013)).

10 CONCLUDING COMMENTS

We are confident that, after we accommodate negation, our compositional semantics is appropriate for most queries that are likely to be asked of data stores containing everyday knowledge. The FDBR datastructure presented in this paper can be used to handle many kinds of complex language features, including chained prepositional phrases and superlatives. The way quantification is handled within the semantics is consistent with other work in this area, as discussed in Section 6. The approach chosen is flexible enough that it can accommodate queries to both relational and non-relational types of databases, including Semantic Web triplestores. It is also suitable for use on low power devices, which may be useful for applications on the Internet of Things (IoT).

In the future, we plan to scale up the capability of our NLQI further to access massive data stores such as DBpedia. To achieve this goal, we plan to accelerate the FDBR generation process using specialized acceleration hardware, such as FPGAs and GPUs.

REFERENCES

Ahn, K., Bos, J., Kor, D., Nissim, M., Webber, B. L., and Curran, J. R. (2005). Question answering with qed at trec 2005. In *TREC*.

Blackburn, P. and Bos, J. (2005). Representation and inference for natural language. *A first course in computational semantics. CSLI*.

Champollion, L. (2015). The interaction of compositional semantics and event semantics. *Linguistics and Philosophy*, 38(1):31–66.

Cimiano, P., Haase, P., Heizmann, J., and Mantel, M. (2007). Orakel: A portable natural language interface to knowledge bases. Technical report, Technical report, Institute AIFB, University of Karlsruhe.

Davidson, D. (1967). The logical form of action sentences.

Erling, O. and Mikhailov, I. (2010). Virtuoso: Rdf support in a native rdbms. In *Semantic Web Information Management*, pages 501–519. Springer.

Ferre, S. (2012). Squall: A controlled natural language for querying and updating rdf graphs. In *proc. of CNL 2012*, pages 11–25. LNCS 7427.

Ferré, S. (2013). Squall: a controlled natural language as expressive as sparql 1.1. In *International Conference on Application of Natural Language to Information Systems*, pages 114–125. Springer.

Frost, R. and Launchbury, J. (1989). Constructing natural language interpreters in a lazy functional language. *The Computer Journal*, 32(2):108–121.

Frost, R. A., Hafiz, R., and Callaghan, P. (2008). Parser combinators for ambiguous left-recursive grammars. In *International Symposium on Practical Aspects*

of Declarative Languages, pages 167–181. Springer LNCS Volume 4902.

Hirst, G. (1992). *Semantic interpretation and the resolution of ambiguity*. Cambridge University Press.

Hoffart, J., Suchanek, F. M., Berberich, K., and Weikum, G. (2013). Yago2: A spatially and temporally enhanced knowledge base from wikipedia. *Artificial intell.*, 194:28–61.

Parsons, T. (1990). *Events in the Semantics of English*, volume 5. Cambridge, Ma: MIT Press.

Van Eijck, J. and Unger, C. (2010). *Computational semantics with functional programming*. Cambridge University Press.

Wendt, M., Gerlach, M., and Düwiger, H. (2012). Linguistic modeling of linked open data for question answering. *proc. of interact. with Linked Data (ILD 2012)[37]*, pages 75–86.

Wheeler, J. (2009). The hsparql package. In *The Haskell Hackage Repository*. http://hackage.haskell.org/package/hsparql-0.1.2.