

Accommodating Negation in an Efficient Event-based Natural Language Query Interface to the Semantic Web

Shane Peelar^a and Richard A. Frost^b

School of Computer Science, University of Windsor, 401 Sunset Avenue, Windsor, Ontario, Canada

Keywords: Semantic Web, Compositional Semantics, Event-based Triplestores, Natural Language Processing, Natural Language Query Interfaces, Quantification.

Abstract: Although The Semantic Web was built with the Open World Assumption in mind, there are many cases where the Closed World Assumption would be a better fit. This is unfortunate because the OWA prevents rich queries involving negation from taking place, even in contexts where it would be appropriate. In this paper we present an English Natural Language Query Interface to event-based triplestores based on Compositional Semantics that can support both open and closed world semantics with “drop-in” denotations for “no”, “not”, “non”, and “the least”. Where closed world semantics are not appropriate, omitting these denotations is sufficient to restore the OWA. The result is a highly expressive compositional semantics supporting complex linguistic constructs such as chained prepositional phrases, n -ary transitive verbs, superlative phrases, and negation, suitable for expert systems and knowledge bases.

1 INTRODUCTION


The Semantic Web consists of a collection of triplestores accessible via *endpoints* that process queries using various query languages. Widely used methods for querying triplestores include using SPARQL (Harris et al., 2013) and Linked Data Fragments (LDF) (Verborgh et al., 2014). These query languages, while powerful, are not designed with end-users in mind, with their primary use cases aimed towards databases rather than user-facing applications. An alternative approach to using a database querying language directly is to use a Natural Language Query Interface (NLQI). NLQIs have a number of benefits including being accessible through both text and speech modalities.


There are two main approaches used by NLQIs. Machine Learning (ML) can be used to attempt to determine the user’s intent and retrieve corresponding relevant information. This has the advantage of being able to support a wide variety of queries, with the risk that returned information may not truly satisfy the user’s intent. The second type of approach is to use a Compositional Semantics (CS) to directly answer the query with regards to a knowledge base. CS is pred-

icated on the notion that the meaning of a sentence can be derived from the meaning of its parts (Dowty et al., 1981). This has the advantage that the answer to a query is as correct as the information in the knowledge base itself. As a result, CS-based NLQIs are able to express highly sophisticated “narrow” queries using complex linguistic constructs including superlatives and chained prepositional phrases. For example, it is possible for the NLQI presented in this paper to evaluate, with respect to a knowledge base consisting of facts about the solar system, the query:

```
which vacuumous moon that orbits the
planet that is orbited by the most moons
was discovered by nicholson or pickering
with a telescope in 1898 at not mt_wilson
or not mt_hopkins
```

However, CS approaches have drawn a lot of criticism. They have been characterized as being rigid, and therefore not sufficiently able to handle complex queries in real world applications. Recent work has addressed a number of these issues, including accommodating chained complex prepositional phrases (Peelar, 2016), n -ary transitive verbs (Peelar and Frost, 2020b) and superlative phrases (Frost and Peelar, 2019). It has also been shown that CS can be memoized for efficient evaluation (Peelar and Frost, 2020a), which also enables offline pre-computation of query results.

^a  <https://orcid.org/0000-0001-7391-0951>

^b  <https://orcid.org/0000-0001-7083-5060>

One criticism of our previous approaches was that they relied on the Open World Assumption (OWA) and hence could not support negation in queries. While the Resource Description Framework (RDF) ((W3C), 2014) underlying the Semantic Web itself is predicated on the OWA, there exist triplestores where the Closed World Assumption (CWA) holds, particularly in knowledge bases for expert systems. It would be ideal to support negation in queries to these triplestores.

In this paper, we show that it is possible to accommodate negation in an English NLQI to an event-based triplestore where the CWA holds. In particular, we describe an English NLQI to an event-based triplestore using a CS that supports arbitrary quantification including negation, complex linguistic constructs including chained prepositional phrases with superlatives and n -ary transitive verbs. The approach is an extension of Montague’s approach (Dowty et al., 1981). Readers are directed towards (Frost and Peelar, 2019) and (Peelar and Frost, 2020a) for an introduction to the work that this paper builds on.

In Section 2 we describe previous work on NLQIs to the Semantic Web that support negation. In Section 3 we describe how to access a live demonstration of our NLQI that can accommodate the queries presented in this paper along with some other example queries. In Section 4 we describe our event-based semantics and in Section 5 we describe how to accommodate negation where the CWA holds. In Section 6 we provide a list of examples queries and explain how they are processed. Finally, we conclude in Section 7 and Section 8.

2 PREVIOUS WORK

In 2002, Frost and Boulos introduced the notion of “complementary sets” as a way of accommodating negation in FLMS, a set-theoretic version of Montague Semantics (Frost and Boulos, 2002). Their approach has two drawbacks: first, that a separate denotation for transitive verbs had to be created for handling queries such as “discover no moon”. In practice, this meant that 4 denotations of transitive verbs were required: for active and passive tense verbs, and a corresponding “no” query (as in “discover no moon” or “discovered by no person”). The approach presented in this paper requires only one denotation for transitive verbs for all cases, including with the presence of chained prepositional phrases and superlatives. Second, their approach required that the cardinality of the set of entities in the database be a known constant. The denotations presented in this pa-

per receive the cardinality of the set of entities as an argument instead. A query is made to the triplestore itself to retrieve the cardinality of the set of entities, removing the need for computing it locally.

Champollion showed that that negation in event-based CS can be accommodated using “negative events” (Champollion, 2011). These appear to be similar to the ideas expressed in (Frost and Boulos, 2002), although both approaches were developed independently. Where Frost and Boulos discuss representing the result of a “negative” query by implicitly enumerating the complement of a set of entities, Champollion describes events that preclude other events from occurring. This gives some confidence about the nature of the approach taken towards accommodating negation in CS. Our own approach to negation in this paper is based in part on (Frost and Boulos, 2002), but is event-based rather than entity-based and therefore suitable for event-based triplestores.

SQUALL (Ferre, 2012) has limited support for negation in queries, mapping negation onto the “NOT EXISTS” construct of SPARQL. In particular, SQUALL has a denotation for the adverb “not”, where its presence removes triples from the result set. This implies closed-world semantics for the query (Darari et al., 2014), although this is not discussed by the authors. SQUALL is unable to accommodate negation in noun-phrases (such as “which non-moon spins”). SQUALL is also unable to negate termphrases (for example “not Hall or not Galileo”). The reading of SQUALL queries is also not as natural as our semantics. The example given in (Ferré, 2014), “Which author of Paper42 has not affiliation Salford.University?” could be expressed in the semantics of this paper as “Which author of Paper42 is not affiliated with Salford.University?” Also, where SQUALL depends strictly on translation to SPARQL, the approach described in this paper is not tied to any particular database query language or interface and could readily be adapted to relational databases.

3 HOW TO ACCESS OUR NLQI

A live demonstration of our NLQI is accessible via the following URL:

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

In addition to accepting textual input, it also can be interacted with speech on browsers that support the WebSpeech API (W3C et al., 2018). Currently, this includes Google Chrome-based browsers and Firefox.

3.1 System Overview

The approach presented in this paper is based on Richard Montague’s denotational semantics (Dowty et al., 1981). In particular, our system derives the meaning of a query from the meaning of its parts. A query is evaluated with respect to a triplestore as though it were a formal mathematical expression using an Executable Attribute Grammar (Hafiz, 2011). For example, the query “ganymede discovered no moons” is evaluated as the expression:

$$\|phobos\| (\|discovered\| (\|no\| \|moons\|))$$

where $\|x\|$ represents the denotation (meaning) of x .

3.2 Supported Features

The following are a list of example queries that demonstrate features supported by the interface.

n-ary Transitive verbs:

who used a telescope to discover a moon

Quantification:

who used two telescopes to discover one moon \Rightarrow science_team_2

Chained prepositional phrases:

which telescope was used by a person in 1877 \Rightarrow refractor_telescope_1

Superlatives:

hall discovered the most moons that orbit mars \Rightarrow True

Negated noun-phrases:

a non-planet was discovered \Rightarrow True

Negated verb-phrases:

allen did not discover anything \Rightarrow True

Negated term-phrases, including conjunction:

not hall and galileo discovered phobos \Rightarrow False

Adjectives:

enceladus is a vacuumous moon \Rightarrow True

The above features can be combined arbitrarily to form rich queries. For example, adjectives can be combined with negation:

mars is a non-blue planet \Rightarrow True

In all cases, the query processor returns the syntax tree of the query to help the user understand how the query was evaluated (Peelar and Frost, 2020b). A list of example queries and a discussion of how they are evaluated can be found in Section 6.

4 EVENT-BASED DENOTATIONAL SEMANTICS

The approach described in this paper builds upon FLMS (Frost and Launchbury, 1989), EV-FLMS (Frost et al., 2014), UEV-FLMS (Peelar, 2016), and most recently Memoized UEV-FLMS (Peelar and Frost, 2020a). Notably, our semantics are event-based rather than entity-based. The fundamental data structure underlying our semantics is called the Function defined by a Relation, or *FDBR*, described in Section 4.2. This data structure has been shown to be useful in answering a wide variety of Natural Language queries (Frost and Peelar, 2019). In this paper, we show how the FDBR can be used to answer queries involving negation in event-based databases where the CWA holds.

4.1 Event-based Triplestores

A conventional triplestore is a database of triples that have the form (*Subject, Predicate, Object*). An event-based triplestore is a triplestore where the *Subject* of a triple denotes an event (Frost et al., 2013)(Frost et al., 2014). The main advantage event-based triplestores offer is that it is straightforward to add additional information to an event by simply adding more triples referencing that event. It is less straightforward to do the same in a triplestore where the *Subject* denotes an entity. Such an approach in a conventional triplestore requires *reification* and involves using ontological information to link multiple triples together.

As an example, consider a triple that describes the statement “Jane bought a pencil”:

```
<ent:Jane> <act:purchase> <ent:pencil_1> .
```

Without reification, there is no way to add other information about the purchase to the triplestore, such as the price, or the time or location that the transaction took place. In an event-based triplestore, this is straightforward:

```
<event:1> <type> <type:purchase_ev> .
<event:1> <subject> <ent:Jane> .
<event:1> <object> <ent:pencil_1> .
```

Since the triples directly reference the event itself, adding more information about the event simply involves adding more triples to the triplestore with the *Subject* matching the event.

4.2 The Function Defined by a Relation (FDBR)

The notion of a *Function defined by a Relation* (FDBR) was first described in (Peelar, 2016) as use-

ful datastructure for accommodating chained prepositional phrases in Natural Language Queries. It was shown that the word “by”, as in “discovered by”, could be treated as a “virtual preposition” under this approach. In (Frost and Peelar, 2019) it was shown that the FDBR can be used to answer many kinds of NL queries including superlatives (including those that occur in a prepositional phrase), and as a useful datastructure for memoizing the results of queries performed in the denotations. This vastly improved query execution time and opened the door for offline computation of results. The definition of the FDBR is as follows:

$$\text{FDBR}(rel) = \{(x, image_x) \mid (\exists e) (x, e) \in rel \\ \& image_x = \{y \mid (x, y) \in rel\}\}$$

Where *rel* is the name of a binary relation. The FDBR has been shown to be useful for the denotation of transitive verbs. Consider the denotation for the active voice of “discover” given in (Peelar and Frost, 2020b), for example:

$$\|discover\| = \\ \lambda t \{(s, relevs) \mid (s, evs) \in \text{FDBR}(discover_rel) \\ \& (t \text{ obj_fdbr}(evs) \neq \emptyset) \\ \& relevs = \text{gather}(\text{obj_fdbr}(evs))\}$$

where *obj_fdb*(*evs*) is the FDBR from the objects in the events of the set *evs* to the events they participate in within *evs*. “discover phobos”, where *phobos* is a proper noun, results in the FDBR:

$$\{(e_{hall}, \{ev_{1045}, ev_{1046}\})\}$$

The FDBR can be readily extended to *n*-ary relations (and hence *n*-ary transitive verbs) (Peelar and Frost, 2020b). In this paper we show that with some small modifications, the FDBR can be used to answer NL queries with negation as well, in cases where the CWA holds.

5 ACCOMMODATING NEGATION

Negation in NL queries is only possible if the CWA holds for a database. We modify the semantics presented in (Frost and Peelar, 2019), (Frost and Peelar, 2018) and (Peelar and Frost, 2020b) such that the results of denotations may return the complement of an FDBR in addition to an FDBR, adopting a similar approach to (Frost and Boulos, 2002). We define this type as follows:

```
type Result = FDBR fdb | ComplementFDBR fdb
```

We then define the intersection of two Result types as follows:

$$\begin{aligned} \text{intersect_result} (\text{FDBR } a) (\text{FDBR } b) &= \text{FDBR } \$ \text{intersect_fdb} a b \\ \text{intersect_result} (\text{FDBR } a) (\text{ComplementFDBR } b) &= \text{FDBR } \$ \text{difference_fdb} a b \\ \text{intersect_result} (\text{ComplementFDBR } a) (\text{FDBR } b) &= \text{FDBR } \$ \text{difference_fdb} b a \\ \text{intersect_result} (\text{ComplementFDBR } a) &(\text{ComplementFDBR } b) \\ &= \text{ComplementFDBR } \$ \text{union_fdb} a b \end{aligned}$$

Where *intersect_fdb* operates as it did previously, and a new function *difference_fdb* is introduced as follows:

$$\text{difference_fdb} = \lambda m \lambda s. \{(e_1, evs_2) \mid (e_1, evs_1) \in m \\ \& \forall (e_2, evs_2) ((e_2, evs_2) \in s \Rightarrow e_1 \neq e_2)\}$$

That is, *difference_fdb* removes all entities found in the left column of the second FDBR from the first FDBR. This is a key function for performing negation, and plays a similar role to the NOT EXISTS operator in SPARQL. A function is introduced for computing the union of Results as well:

$$\begin{aligned} \text{union_result} (\text{FDBR } a) (\text{FDBR } b) &= \text{FDBR } \$ \text{union_fdb} a b \\ \text{union_result} (\text{FDBR } a) (\text{ComplementFDBR } b) &= \text{ComplementFDBR } \$ b \text{'difference_fdb' } a \\ \text{union_result} (\text{ComplementFDBR } a) (\text{FDBR } b) &= \text{ComplementFDBR } \$ a \text{'difference_fdb' } b \\ \text{union_result} (\text{ComplementFDBR } a) &(\text{ComplementFDBR } b) \\ &= \text{ComplementFDBR } \$ a \text{'intersect_fdb' } b \end{aligned}$$

This is used in the denotation of “and” and “or” as used with termphrases. Next, we introduce a function to obtain the cardinality of a Result:

$$\begin{aligned} \text{cardinality_} (\text{FDBR } np) &= \text{List.length } np \\ \text{cardinality} (\text{Just num_ents}) (\text{ComplementFDBR } np) &= \text{num_ents} - \text{length } np \end{aligned}$$

The first argument to this function is passed in from the query pipeline described in (Peelar and Frost, 2020a), and is either *Nothing* or *Just num_ents*, where *num_ents* is the cardinality of the set of entities in the triplestore. It will only be retrieved if the query has any denotations involving negation in it.

Our approach maintains leftmost-outermost scoping of quantifiers including negation, which enables a natural reading of the query.

5.1 Quantifiers

We modify the denotations of all quantifiers to be characterized in terms of the cardinality:

```

a' = intersect_result
every'' cardinality nph vbph =
  if cardinality result == cardinality nph
  then result else FDBR []
one'' cardinality nph vbph =
  if cardinality result == 1
  then result
  else FDBR []
two'' cardinality nph vbph =
  if cardinality result == 2
  then result
  else FDBR []
most'' cardinality nph vbph =
  if n_nph /= 0 && (n_nph_v / n_nph) > 0.5
  then result else FDBR []
where
  n_nph = fromIntegral $ cardinality nph
  n_nph_v = fromIntegral $ cardinality res

```

where in the above denotations, `result = intersect_result'' nph vbph`. Curiously, the function `cardinality` appears as the first argument to these quantifiers, giving them three arguments in total. This function is passed in from the caller as a function that can be used to obtain the cardinality of a FDBR. A function, “`apply_card`” is used to automatically apply the cardinality function to the denotations. For example:

```

every' = applyCard every''
>|< GettsIntersect GI_Every
every = wrapS2 every'

```

The `>|<` operator is described in more detail in (Peelar and Frost, 2020a). It is used to assign a unique name to the denotations according to the syntax tree of the query. This is useful for memoization and query optimization. “no” is denoted as follows:

```

no'' cardinality nph (FDBR []) =
  ComplementFDBR []
no'' cardinality nph vbph =
  if cardinality result == 0 then vbph
  else FDBR []

```

where in the above denotations, `result = intersect_result'' nph vbph`. This is a departure from the denotation of “no” given in (Frost and Boulos, 2002). Namely, the complement of the empty FDBR (denoting “everything”) is returned when an empty FDBR is passed as the second argument to “no”. This is critical in handling “no” in the denotation of transitive verbs as discussed later in Section 5.4.

5.2 Negating Noun- and Verb-Phrases

We denote “not”, when applied to a verb-phrase (such as “not spins”) as follows:

```

not (FDBR vbph) = ComplementFDBR vbph
not (ComplementFDBR vbph) = FDBR vbph

```

“non” plays a similar role as a prefix to a noun-phrase, and can be denoted as:

```
non = not
```

5.3 Negating Term-Phrases

One aspect missing from both (Ferré, 2014) and (Frost and Boulos, 2002) is the notion of negating termphrases, such as “hall”, “a moon”, “one moon”, and “no moon”, which exhibits double negation. Negating termphrases provides more flexibility to the query interface, making it possible to express the query:

```

who discovered in 1877 not one moon that
  orbits mars

```

Where “one” denotes “exactly one”. This query explicitly is excluding any discoverers that discovered exactly one moon that orbits mars. It results in `hall`, because `hall` discovered two moons that orbit mars in 1877. “not” when applied to a term-phrase, such as “hall” or “a moon” is denoted as follows:

```

termnot tmpv vbph
  = intersect_result (not (tmpv vbph)) vbph

```

Therefore `not hall spins` is evaluated as follows:

```

(not hall) spins
=> intersect_result (not (hall spins)) spins
=> intersect_result (not (FDBR [])) spins
=> intersect_result (ComplementFDBR []) spins
=> spins
=> True (because spins is not empty)

```

Negating term-phrases was not discussed in (Frost and Boulos, 2002), and it offers more flexibility in the nature of queries that can be performed (see Section 6)

5.4 A Denotation for Transitive Verbs That Accommodates Superlatives, Prepositional Phrases, and Negation

Transitive verbs are less straightforward to accommodate with negation. Consider the following query:

```
ganymede discovered no moons
```

This query should evaluate to *True*, as `ganymede`, a moon, was not the subject of any discovery events – however, `ganymede` is not the *subject* of any events of type *discover*. Therefore, it is missing from `FDBR(discover_rel)`.

A denotation is given in (Frost and Boulos, 2002) that accommodates this usage of transitive verbs; however it requires syntactic disambiguation at the grammar level to apply correctly. The approach also

does not scale well when other linguistic constructs are introduced, such as chained prepositional phrases and superlatives, requiring a new denotation to support each usage. The examples given in that paper required 4 denotations for transitive verb depending on the context.

The denotation we introduce expands on the denotation introduced in (Frost and Peelar, 2019), where we described how superlative phrases can also be accommodated. This new denotation evaluates the list of prepositional phrases (including superlatives) in leftmost-outermost order, which is consistent with other work in the area (Champollion, 2010), (Ferré, 2014). For example, the query “discovered a moon in 1877 with a telescope” would be evaluated with scoping as though it were as follows: “discovered (a moon (in 1877 (with a telescope))) – that is, “with a telescope” takes precedence over “in 1877, which in turn takes precedence over a moon”.

Only one denotation for transitive verbs is required for all cases (rather than 4 as in (Frost and Boulos, 2002)). In particular, the word “no” can be handled compositionally rather than syntactically in the query.

We modify the denotation for transitive verbs given in (Peelar and Frost, 2020a) to evaluate the list of prepositional phrases in leftmost-outermost order. The `filter_ev` function, described in (Peelar, 2016), is modified to operate on one prepositional phrase at a time: a new FDBR is computed for each prepositional phrase applied. This allows superlatives to be neatly evaluated in the order they appear rather than in a separate stage after the prepositions are evaluated as denoted in (Frost and Peelar, 2019). `filter_ev` also is modified to account for negation in the query: first, the current prepositional phrase is evaluated against the empty FDBR (`FDBR[]`). If the result is not an FDBR, then it is a no termphrase:

```
in no place (FDBR [])
=> ComplementFDBR []
```

This is owing to the denotation of `no` used in Section 5.1. Indeed, the only way to obtain a non-empty FDBR from applying an empty FDBR is through negation. When this is the case, `filter_ev` returns a complement in the same fashion as (Frost and Boulos, 2002).

Since `filter_ev` can also receive the complement of an FDBR, as in the case when negation is present in the query, applying term-phrases can be difficult. The complement operation is reversed by taking the FDBR of the transitive verb itself and performing the intersection of it with the complement passed into `filter_ev`. Whether negation is present in the cur-

rent prepositional phrase or not, this FDBR is passed in to the termphrase of that preposition. If no negation is present in the current preposition, the result is returned as-is, unless it contains a superlative. Otherwise, if negation is present, then if a complement of an FDBR was passed into `filter_ev`, the FDBR used in the denotation of the transitive verb itself is used to compute the complement, otherwise the FDBR passed into `filter_ev` is used directly. This allows for `discover no moon in 1948` to work as expected. This can neatly handle the following cases:

```
discover no moons in no places with no
telescopes
```

The result is the complement of the FDBR of those that discovered a moon in a place with a telescope

```
discover no moons in 1877
```

The result is the complement of the FDBR of those that discovered a moon in 1877

```
discover a moon in 1877
```

The result is the FDBR of the people that discovered a moon in 1877.

5.5 Obtaining the Cardinality of the Entities of the Triplestore

In systems where the Open World Assumption holds, obtaining the cardinality of the set of entities may not be possible, as the cardinality of that set may be infinite. Attempting to obtain that set at all may not be practical. Even in systems where the CWA holds, obtaining the set of all entities in the database may not be feasible. Fortunately, only the cardinality is required to start answering queries.

A new querying primitive is introduced from (Peelar and Frost, 2020a) that queries the remote triplestore itself for the cardinality of the set of entities in the triplestore:

```
getts_cardinality_allents ev_data props
```

Here, `ev_data` represents the URL of the triplestore itself (in the case of SPARQL, a SPARQL endpoint URL), and `props` is the set of properties of the events contained in the triplestore whose entities should be counted towards the cardinality. In the example queries given in this paper, the properties listed for cardinality are `subject`, `object`, `location`, and `implement`. We exclude the `year` property as all entities must exist both physically and temporally ((W3C), 2014). This function, like the other `getts*` family functions described in (Frost and Peelar, 2019), can be specialized for different types of databases, including relational triplestores.

This alleviates having to send the full set of entities to the semantics in order to answer a query that

uses negation. Note that the cardinality of the set of entities of the triplestore is only ever required in queries that have negation present. Using the memoization and triplestore querying framework described in (Peelar and Frost, 2020a), a guarantee is made that if no negation is present in the query, the cardinality query will never be performed. Therefore, our denotations for negation in queries, including “not”, “non”, “no” and “the least”, are drop-in enhancements to NLQIs built using our framework: if the CWA holds for the application, all one needs to do is add these denotations in. Otherwise, the NLQI will operate with the open world semantics described in (Frost and Peelar, 2019).

In some cases, the user may want to force evaluation of the complement. It is possible to introduce a special denotation, “force_eval”, will obtain all triples and force retrieval of all entities. This may be cached on the interface to alleviate the load against the remote triplestore using the memoization framework in (Frost and Peelar, 2019). It may be appropriate to evaluate the complement if its cardinality is under a certain threshold as well, triggering “force_eval” automatically – this could be customized on a per-application basis.

5.6 Accommodating “The Least”

In (Frost and Peelar, 2019) we described how to accommodate superlative phrases compositionally by delegating their evaluation to the transitive verb they are arguments of. This allows them to appear in chained prepositional phrases.

One problem described with that approach was answering queries with superlatives such as “the least” or “the lowest number of”. The main reason for this was owing to the OWA underlying the semantics. Under that approach, “which planets are orbited by the least number of moons” would return earth, despite both venus and mercury having a lower number of moons than Earth. The semantics had no concept of *zero* and could only report about what was observable. Since there were no events explicitly stating that venus and mercury had no moons, it could not assume that it was not the case.

We propose an alternative approach in this paper, where “the least” is handled similarly to the word “no”. The denotation for “the least” first checks that the complement of the FDBR is non-empty. If so, “the least” returns the complement of that FDBR – this allows for “venus” and “mercury” to appear in the result set while removing all non-candidates. If the complement of the FDBR is empty, however, then

it performs the same cardinality partitioning that the “the most” does (Frost and Peelar, 2019), except it chooses the lowest *object cardinality* entities to form the result rather than the greatest.

6 EXAMPLE QUERIES

The following are some example queries that can be handled by our NLQI. With each query we explain the result and how it was evaluated.

no people spin \Rightarrow *True*

The intersection of the people and spins FDBRs is empty, therefore no returns spin, which is non-empty and therefore *True*.

a non person exists \Rightarrow *True*

“non person” is the complement of the person set, and the intersection of this complement with the exists set (which is the complement of the empty set) is the same as the complement of the union of the person set with the empty set. The answer is characterized in terms of the cardinality, which for the complement of an FDBR is defined as the cardinality of the number of set of entities in the triplestore minus the cardinality of the FDBR itself. This is greater than 0, and therefore there is at least one entity that is both a non-person and exists.

a person does not exist \Rightarrow *False*

This computes the intersection of the person FDBR with the complement of the exists FDBR, which is just the empty FDBR. Therefore, the result is empty, and the result is *False*.

what discovered no moon in 1877 \Rightarrow
everything except: hall

This sentence is treated similarly to what did not discover a moon in 1877. The result is the complement of the set of entities that discovered a moon in 1877, in this case, hall.

what discovered a non moon \Rightarrow nothing.

This query is specifically asking about entities that discovered non-moons – the entities that did not discover anything are not included in this set and therefore an FDBR is returned. Since that FDBR is empty, the result is that nothing in our triplestore discovered any non-moons.

allen discovered no moon at no places \Rightarrow *True*

The result of “discovered no moon at no places” is the complement of the FDBR returned from “discovered a moon at a place”. This includes entities that either discovered a moon at no known location, or discovered a non-moon at a known location. Since allen does not appear in

the FDBR returned by “discovered a moon at a place”, the result is *True*.

what discovered the most moons using no telescopes ⇒ *voyager_science_team*

This query combines both a superlative phrase with negation. The query is asking “out of the events where entities discovered something without using a telescope, which ones discovered the most moons”. Since *voyager_science_team* used no telescopes at all to discover 22 moons, more than any other entities that discovered using no telescopes, they are in the result set.

what was discovered by no team in 1877 ⇒ *everything*.

This query is handled the same as “what was not discovered by a team in 1877”, which returns the complement of the empty FDBR, since no teams discovered anything in 1877.

how was something discovered using no telescope ⇒ I can't perform this query because I would need to enumerate the entire triplestore.

This query is asking about which implements that are not telescopes were used in a discovery event. However, something is defined as the complement of the empty FDBR, and discovered using no telescope is the complement of the FDBR of discovered using a telescope. Since the intersection of the two complements is itself a complement, how receives the complement of an FDBR and is unable to enumerate the events to retrieve implements from directly. Although it is possible to answer the query by fully evaluating the complement, we have not implemented this behaviour in our NLQI at this time. However, a similar query, “which non telescope was used to discover something” is able to yield the result “*cassini, voyager_1, voyager_2*”.

not hall discovered *ganymede* ⇒ *True*
“not hall” is a negated term-phrase. The result is *True* because *galileo* discovered *ganymede*, not *hall*.

which person that does not spin discovered no planet in 1877 using a telescope and is a discoverer ⇒ *bernard, bond, cassini, christy, dollfus, galileo, hall, herschel, holman, huygens, karkoschka, kowal, kuiper, lassell, melotte, nicholson, perrine, pickering, sheppard, showalter*

The result is all of the people that are discoverers, since none of the members of *person spin*, and none of them discovered a planet in 1877 using a telescope. It may be helpful to examine the scoping of this query:

which (person 'that' (does not spin)) ((discovered (no planet) [in 1877, using (a telescope)]) and (is a discoverer)) nothing exists ⇒ *False*

This is *False* because the intersection of *thing* and *exists* is non-empty.

everything exists ⇒ *True*

This is *True* because *thing* and *exists* are both the complement of the empty FDBR, and the intersection of those results in the same. Therefore “*thing*” is a subset of “*exists*”.

what was not discovered by *hall* ⇒ everything except: *deimos, phobos*

This is the complement of the FDBR “discovered by *hall*”. The answer is the set of all things excluding those that *hall* discovered.

phobos and *deimos* were not discovered by not *hall* ⇒ *True*

This query features double negation and is equivalent to asking “*phobos* and *deimos* were discovered by *hall*”. The result is *True*.

not not *kuiper* discovered not not *nereid*

This query also features double negation on the termphrases *kuiper* and *nereid*. This is equivalent to the query *kuiper* discovered *nereid*.

which non vacuumous moon that orbits most planets that spin was not discovered by *kuiper* at two places using the most telescopes in 1942 ⇒ *none*.

This query features a variety of complex linguistic constructs, including nested *n*-ary transitive verbs, adjectives, negation, chained prepositional phrases, quantification and superlative phrases. The result is “*none*” because “non vacuumous moon that orbits most planets that spin” returns the empty FDBR, and the intersection of the empty FDBR with any set is also the empty FDBR.

not no moon orbits *mars* ⇒ *True*

This query features a negated termphrase, which itself consists of the word “no” (entailing negation itself). “*orbits mars*” is the FDBR from the entities *phobos* and *deimos* to their orbit events, and “not no moon orbits *mars*” evaluates to “*orbits mars*” with the entities of “no moon orbits *mars*” removed. Since “no moon orbits *mars*” is *False*, it returns the empty FDBR, which is then removed from “*orbits mars*”, giving a non-empty result. Therefore, the query returns *True*. This provides evidence that our NLQI correctly handles negation as a compositional construct.

who discovered no moons at no places ⇒ *allen, baum, buie, burns ... (full results omitted here) ... weaver, young_e_f,*

young_1_a

The result is everyone that is not known to have discovered a moon at a place. This includes the discoverers that discovered a moon at no known location, or whose location property is not listed in the event, or discoverers that discovered a non moon at a known location.

7 FUTURE WORK

Our next efforts will be focused on creating an NLQI to DBPedia using the approaches described here and in (Peelar and Frost, 2020a). Specifically, we plan to use Timbr.ai (Timbr, 2020) to provide a relational view of DBPedia, targeting SQL as the query language. Once this is done, we plan to test our NLQI using well-known benchmarks such as QALD (Usbeck et al., 2018).

We also plan to explore interfacing with non-event based triplestores in general. ML approaches may be useful in contexts where ontological information is not available for reification.

8 CONCLUSIONS

We have shown that it is possible to accommodate negation in our event-based CS efficiently. We have shown that our approach to negation is powerful, able to be applied to noun-phrases, verb-phrases, and term-phrases. We presented a denotation for “no” that enables it to be treated as a quantifier that can be compositionally used in conjunction with transitive verbs, either as an argument to the verb or as a preposition. We improved on (Frost and Boulos, 2002) by maintaining only one denotation for transitive verbs throughout the semantics rather than requiring different denotations depending on the context. Notably, our approach to negation seems to be consistent with other work in event semantics (Champollion, 2011). We improved on (Ferré, 2013) by enabling the negation of term-phrases, and also enabling our approach to be used with other query languages than SPARQL. We discussed the necessity of the Closed World Assumption for queries involving negation and described how to extend the CS in (Frost and Peelar, 2019) to accommodate negation in queries. Where the CWA is not appropriate, leaving out the denotations for “not”, “non”, and “the least” is sufficient to restore the Open World Assumption in the semantics. Our approach also fits within the memoization framework in (Frost and Peelar, 2019). We also discussed example queries that are supported with our NLQI and ex-

plained how the results are formed. We believe now that our semantics is ready to be benchmarked directly against other systems on large knowledge bases using, for example, QALD-9 (Usbeck et al., 2018).

ACKNOWLEDGEMENTS

This research was supported by NSERC of Canada.

REFERENCES

- Champollion, L. (2010). Quantification in event semantics. In *Talk given at the 6th int. symp. of cogn., Logic and commun.*
- Champollion, L. (2011). Quantification and negation in event semantics.
- Darari, F., Razniewski, S., and Nutt, W. (2014). Bridging the semantic gap between RDF and SPARQL using completeness statements [extended version]. *CoRR*, abs/1408.6395.
- Dowty, D., Wall, R., and Peters, S. (1981). *Introduction to Montague Semantics*. D. Reidel Publishing Company, Dordrecht, Boston, Lancaster, Tokyo.
- 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.
- Ferré, S. (2014). Squall: The expressiveness of sparql 1.1 made available as a controlled natural language. *Data & Knowledge Engineering*, 94:163–188.
- 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., Amour, B. S., and Fortier, R. (2013). An event based denotational semantics for natural language queries to data represented in triple stores. In *ICSC, 2013 IEEE Seventh International Conference on Semantic Computing*, pages 142–145. IEEE.
- Frost, R. A. and Boulos, P. (2002). An efficient compositional semantics for natural-language database queries with arbitrarily-nested quantification and negation. In Cohen, R. and Spencer, B., editors, *Advances in Artificial Intelligence, 15th Conference of the Canadian Society for Computational Studies of Intelligence, AI 2002, Calgary, Canada, May 27-29, 2002, Proceedings*, volume 2338 of *Lecture Notes in Computer Science*, pages 252–267. Springer.
- Frost, R. A., Donais, J., Matthews, E., and Agboola, W. (2014). A demonstration of a natural language query interface to an event-based semantic web triplestore. In *ESWC*, pages 343–348. Springer LNCS Volume 8798.

- Frost, R. A. and Peelar, S. M. (2018). An extensible natural-language query interface to an event-based semantic web triplestore. In Choi, K., Anke, L. E., Declerck, T., Gromann, D., Kim, J., Ngomo, A. N., Saleem, M., and Usbeck, R., editors, *Joint proceedings of the 4th Workshop on Semantic Deep Learning (SemDeep-4) and NLIWoD4: Natural Language Interfaces for the Web of Data (NLIWOD-4) and 9th Question Answering over Linked Data challenge (QALD-9) co-located with 17th International Semantic Web Conference (ISWC 2018), Monterey, California, United States of America, October 8th - 9th, 2018*, volume 2241 of *CEUR Workshop Proceedings*, pages 1–16. CEUR-WS.org.
- Frost, R. A. and Peelar, S. M. (2019). A new data structure for processing natural language database queries. In *Proceedings of the 15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019*, pages 80–87.
- Hafiz, R. (2011). Executable attribute grammars for modular and efficient natural language processing.
- Hafiz, R., Frost, R., Peelar, S., Callaghan, P., and Matthews, E. (2020). The xsaiga package.
- Harris, S., Seaborne, A., and Prud'hommeaux, E. (2013). Sparql 1.1 query language. *W3C recommendation*, 21(10).
- Peelar, S. (2016). Accommodating prepositional phrases in a highly modular natural language query interface to semantic web triplestores using a novel event-based denotational semantics for english and a set of functional parser combinators. Master's thesis, University of Windsor (Canada).
- Peelar, S. and Frost, R. A. (2020a). A new approach for processing natural-language queries to semantic web triplestores. In *15th International Conference on Web Information Systems and Technologies, WEBIST 2019, Vienna, Austria, September 18-20, 2019, Revised Selected Papers*, Lecture Notes in Business Information Processing. Springer.
- Peelar, S. M. and Frost, R. A. (2020b). A compositional semantics for a wide-coverage natural-language query interface to a semantic web triplestore. In *IEEE 14th International Conference on Semantic Computing, ICSC 2020, San Diego, CA, USA, February 3-5, 2020*, pages 257–262. IEEE.
- Timbr (2020). Timbr.io. <https://wiki.dbpedia.org/timbr>.
- Usbeck, R., Gusmita, R. H., Ngomo, A.-C. N., and Saleem, M. (2018). 9th challenge on question answering over linked data (qald-9). In *Semdeep/NLIWoD@ ISWC*, pages 58–64.
- Verborgh, R., Vander Sande, M., Colpaert, P., Coppens, S., Mannens, E., and Van de Walle, R. (2014). Web-scale querying through linked data fragments. In *LDOW*. Citeseer.
- W3C et al. (2018). Web speech api. Retrieved from *World Wide Web Consortium*: <https://w3c.github.io/speech-api>.
- (W3C), T. W. W. W. C. (2014). RDF 1.1 Semantics. <https://www.w3.org/TR/rdf11-mt/>. [Online; accessed 06-September-2016].

APPENDIX

The complete source code for the demonstration, including the semantics and parsing framework, can be found online at the Hackage Haskell package repository under the XSaiga project (Hafiz et al., 2020):

<https://hackage.haskell.org/package/XSaiga>