EXISTStential Aspects of SPARQL

David Martin and Peter F. Patel-Schneider

Nuance Communications
david.martin@nuance.com, peter.patel-schneider@nuance.com

The SPARQL 1.1 Query Language [1] permits patterns inside FILTER expressions using the EXISTS construct, specified by using substitution. Substitution destroys some of the aspects of SPARQL that make it suitable as a data access language. As well, substitution causes problems in the SPARQL algebra and produces counterintuitive results. Fixing the problems with EXISTS is best done with a completely different definition that does not use substitution at all.

EXISTS is used inside FILTER constructs to determine whether a pattern matches or does not match. EXISTS can be used to find people who do not have any names, as in this example from the SPARQL specification [1, §8.1.1]:

```
SELECT ?person WHERE { ?person rdf:type foaf:Person .                [1]
  FILTER NOT EXISTS { ?person foaf:name ?name } }
```

Any SPARQL pattern can be used in an EXISTS, including subqueries, as in a query to find people who know someone who knows more than 100 others:

```
SELECT ?person WHERE { ?person foaf:knows ?friend .
  FILTER EXISTS { SELECT ?friend WHERE { ?friend foaf:knows ?y . }
                  GROUP BY ?friend HAVING ( COUNT(*) > 100 ) } }
```

Definition The definition of EXISTS in SPARQL starts with a translation to the SPARQL algebra [1, §18.2.2.2], replacing EXISTS{P} with *exists(translate(*P*))*. The function *exists* is described as "*exists(pattern)* is a function that returns true [iff] the pattern evaluates to a non-empty solution sequence." [1, §18.5] The formal definition of exists uses substitution into its argument

> **Substitute** Let μ be a solution mapping.
> *substitute(pattern,* μ*)* = the pattern formed by replacing every occurrence of a variable v in pattern by $\mu(v)$ for each v in *dom(*μ*)*. [1, §18.6]
>
> **Evaluation of Exists** Let μ be the current solution mapping for a filter and P a graph pattern: The value *exists(P)*, given *D(G)* is true if and only if *eval(D(G), substitute(*P*,*μ*))* is a non-empty sequence. [1, §18.6]

Counterintuitive Results Blank nodes in the graph being queried often produce counterintuitive results with EXISTS. In Query [1] the substitution is performed for every node in the graph that has an rdf:type link to foaf:Person. If one of these nodes is a blank node, say _:Bill, the evaluation ends up matching _:Bill foaf:name ?name against the graph. As _:Bill is a blank node, it acts like a variable and can itself be mapped. As a consequence, no answers will be returned from Query [1] for the graph

```
ex:John rdf:type foaf:Person ; foaf:name "John" .                    [G]
_:Bill rdf:type foaf:Person .
```

This counterintuitive result is particularly pernicious as it can occur any time a FILTER variable is used to query the graph inside an EXISTS.

MINUS is also problematic inside EXISTS. One might expect the result of

```
SELECT ?x WHERE { BIND ( :b AS ?x )
   FILTER EXISTS { ?x :p :b . MINUS { ?x :p :b } } }
```

to be empty because any solution mapping on the left side of the MINUS is eliminated by the same mapping from right side. However, substitution replaces ?x with :b so the two sides don't share a variable and then, because of its definition in SPARQL, the MINUS does not remove any solution mappings.

Substitution also interferes with SPARQL constructs that act like variable bindings. One might expect that

```
SELECT ?x WHERE { ?x :p :b .                                          [2]
   FILTER EXISTS { ?x :p :b . { SELECT ( :d AS ?x ) WHERE { } } } }
```

has no solutions except for ?x mapped to :d because otherwise solutions from ?x :p :b, i.e., with ?x mapped to something other than :d cannot be joined to solutions from SELECT (:d AS ?x) WHERE { }. However, the substitution replaces ?x throughout with :d so the "variable" that comes out of the subquery is not ?x but instead is whatever ?x was mapped to.

Implementations of SPARQL don't produce all these counterintuitive results, instead diverging from the specification, but some do produce some of them.

Semantic Anomalies The substitution for EXISTS does not distinguish between the different uses of variables in SPARQL constructs. It substitutes in triples, as it needs to, and in expressions, which it also needs to, but it also substitutes in other places. In Query [2] this ends up with solution mappings that map non-variables to values, counter to the semantic definitions underlying SPARQL.

This semantic anomaly happens in SELECT clauses, as above, but also in BIND constructs like

```
SELECT ?x WHERE { BIND ( :b AS ?x ) BIND ( :b AS ?y )
   FILTER EXISTS { BIND ( :j AS ?x ) BIND ( :k AS ?y ) } }
```

and in VALUES constructs [4, errata-query-10]. In each case mappings are created that do not conform with the definition of solution mappings in SPARQL.

Other semantic anomalies can also arise, as in

```
SELECT ?x WHERE { :b :p ?x . FILTER EXISTS { FILTER BOUND(?x) } }
```

where the BOUND function is applied to a non-variable, counter to its definition in SPARQL.

Some of these anomalous situations have further problems, as in

```
SELECT ?x WHERE { BIND ( :b AS ?x ) BIND ( :b AS ?y )
  FILTER EXISTS { { SELECT ( :d AS ?x ) WHERE { } }
                  { SELECT ( :e AS ?y ) WHERE { } } } }
```

Here both ?x and ?y end up being substituted as :b so the join between the results from the subqueries ends up being empty, which is counterintuitive as well as being semantically anomalous.

We have not found a SPARQL implementation that reports these semantic anomalies. Either they silently allow illegal internal constructs or they silently do something different from the specification.

Bottom-up Evaluation SPARQL is supposedly designed so that queries can be evaluated "bottom-up [and] subqueries are evaluated logically first, and the results are projected up to the outer query." [1, §12] However, EXISTS is at odds with this design because EXISTS substitution creates altered subqueries that did not exist before the substitution. Because of this, for some queries the results of an evaluation based on substitution will differ dramatically from those produced by bottom-up evaluation, as exhibited by:

```
SELECT ?x WHERE { ?x :b ?y .
   FILTER EXISTS { SELECT ?z WHERE { ?z :f ?y . } } }
```

The problem is particularly acute with uses of variables that would not contribute to solution mappings, such as the nested use of ?y in the following. In this query, the results prescribed by substitution not only differ from bottom-up results, but they are also strongly counterintuitive, as in the query below where no results will be returned because ?y is replaced, even though this variable does not contribute mappings to the results returned from the inner query.

```
SELECT ?x WHERE { ?x :p ?y .
   FILTER EXISTS { SELECT ?x WHERE { ?x :p ?y . }
                   GROUP BY ?x HAVING ( COUNT(*) > 1 ) } }
```

There is a suggested erratum that these variables should not be subject to substitution [4, errata-query-8]. Different implementations of SPARQL produce different answers for these queries, as noticed by Hernndez *et al* [2]. They argue that naïve substitution has many known problems and propose a notion of different kinds of variables that governs when substitution is to be applied. Their solution solves problems related to substitution in subqueries but does not solve the counterintuitive results related to MINUS and blank nodes.

Fixing the Definition To summarize, EXISTS produces counterintuitive results, generates semantic anomalies, hinders bottom-up evaluation, is not implemented as specified, and has divergent implementations. The problem lies firmly with substitute, which naïvely replaces variables with values even in places where this substitution produces undefined algebra constructs or changes their meaning. Making a more nuanced version of substitute that does not substitute in these problematic spots is not possible without extensive changes to the SPARQL algebra, because such approaches would leave variables incorrectly unconstrained as, for example, in Query [1].

Given that a fixed version of *substitute* is not possible, is another kind of definition of EXISTS a solution? Yes, a definition of EXISTS based on solution mappings is possible. The core of this definition basically moves the FILTER solution mapping to the beginning of the argument to EXISTS. For the solution mapping {(?person, _:Bill)} in [1] above, the EXISTS, in effect, evaluates

```
{ VALUES ?person { _:Bill }   ?person foaf:name ?name . }
```

(except that _:Bill is a blank node in graph [G], which is not possible in VALUES).

The technical details of this fix to EXISTS are as follows:

1. Add a new construct, Initial, to the SPARQL syntax and algebra. Initial will be used to set up the initial multiset of solution mappings inside an EXISTS.

It will work much like VALUES except that it will transfer solution mappings through the EXISTS instead of setting up a constant solution mapping.

2. When collecting FILTER elements replace EXISTS{pattern} in the filter expression with *exists(*Initial(t)*,translate(*{Initial(t) pattern'}*))* where t is a fresh token, and similarly for NOT EXISTS{pattern}. If pattern is a *SubSelect* then pattern' is {pattern} otherwise pattern' is just pattern.
3. Translate Initial(t) as itself.
4. Change the definition of the *exists* function to:

   Let μ be the current solution mapping for a filter, t a token, and P a graph pattern: The value *exists(*Initial(t),P*)* given *D(G)* is true iff *eval(D(G),*P'*)* is a non-empty multiset of solution bindings, where P' is P with Initial(t) replaced by $\{\mu\}$.

This definition eliminates all the semantic anomalies resulting from EXISTS. It also fixes the counterintuitive results above. It corresponds better to how EXISTS behaves in SPARQL implementations. As well, under this definition all subqueries in SPARQL queries can then be evaluated completely bottom up and independently of anything outside of the subquery. In particular, subqueries inside EXISTS can be evaluated before the solution mappings going into the FILTER are known.

Variants of this definition are possible, for example, by adding Initial(t) to the beginning of *GroupGraphPattern*s and not just at the top, that modify the behaviour of EXISTS without reintroducing any semantic anomalies.

Pre-binding [3] is an operation provided by many SPARQL implementations that creates a prepared query and then evaluates this query with a given initial solution mapping. SPARQL implementations provide inadequate descriptions of pre-binding so it is very hard to determine how it works in detail.

Pre-binding can be given a formal definition using the technique proposed here for EXISTS. This version of pre-binding pushes a solution mapping into the query by adding Initial(0) to the WHERE clause as in Point 2 evaluates the resulting query in the same way that EXISTS is evaluated.

The definition of EXISTS proposed here eliminates the counterintuitive results of SPARQL, avoids producing any semantic anomalies, and makes SPARQL more suitable as a data access language. It is in all ways better than the current substitution definition.

References

1. Steve Harris and Andy Seaborne. SPARQL 1.1 query language. W3C Rec., https://www.w3.org/TR/2013/REC-sparql11-query-20130321/, 21 March 2013.
2. Daniel Hernndez, Claudio Gutierrez, and Renzo Angles. Correlation and substitution in SPARQL. https://scirate.com/arxiv/1606.01441, 7 June 2016.
3. Jena Class QueryExecutionFactory. https://jena.apache.org/documentation/... /QueryExecutionFactory.html, retrieved 21 June 2016.
4. Errata in SPARQL 1.1. https://www.w3.org/2013/sparql-errata, 30 January 2016.