Specification of a Reification Extension for SPARQL

Olaf Hartig

May 7, 2012

Abstract

This document defines an extension for SPARQL that adds support for accessing RDF reification based statement identifiers in queries. The definitions in this document are compatible with the latest editors' draft of the SPARQL 1.1 Query Language [HS12].

The document is structured as follows: First, Section 1 extends the grammar of SPARQL. Section 2 specifies how to support the extended grammar during the conversion of query strings into algebra expressions; this specification includes the introduction of a new algebra symbol. Finally, Section 3 defines the evaluation semantics for algebra expressions that use the new symbol.

1 Grammar

A *triple reference pattern* is a new syntax element that conforms to the following, new grammar rule[1]:

TRefPattern ::= '<<' VarOrIRIrefOrLitOrTRefP (VarOrIRIref | 'a') VarOrIRIrefOrLitOrTRefP '>>'

VarOrIRIrefOrLitOrTRefP ::= Var | IRIref | RDFLiteral | NumericLiteral | BooleanLiteral | TRefPattern

As the previous grammar rules indicate, a triple reference pattern may contain other triple reference pattern. Hereafter, such a triple reference pattern is called a *nested triple reference pattern*.

Triple reference patterns (including nested triple reference patterns) may be used in a query in the following two ways: 1.) they are part of a BIND clause, or 2.) they are embedded in a triple pattern or a property path pattern. We specify and discuss each of these two options in the following.

Our extended syntax allows for BIND clauses that use a triple reference pattern (instead of an expression). Accordingly, we redefine the following grammar rule[2]:

Bind ::= 'BIND' '(' **(** Expression **| TRefPattern)** 'AS' Var ')'

A triple reference pattern may not only be used in a BIND clause but it may also be embedded in a property path pattern or in a triple pattern[3]. More precisely, in our extended syntax a property path pattern may begin with and end with a triple reference pattern (instead of an RDF term or a query variable). Similarly, triple patterns (as elements of the extended syntax) may have a triple reference pattern in the subject position and in the object position (notice, this does not hold for triple patterns as elements of the algebra). Accordingly, we introduce a new grammar rule:

VarOrTermOrTRefP ::= Var | GraphTerm | TRefPattern

and redefine the following grammar rules:

TriplesSameSubjectPath ::= **VarOrTermOrTRefP** PropertyListPathNotEmpty | TriplesNode PropertyListPath

Object ::= GraphNode **| TRefPattern**

[1] Elements of the grammar that are not specified explicitly in this document are defined as given in [HS12, Section 19.8]

[2] The adjusted part in which a redefined grammar rule differs from the corresponding rule in [HS12, Section 19.8] is given in bold font.

[3] For a definition of *property path pattern*s and *triple pattern*s we refer to Sections 18.1.7 and 18.1.5 in [HS12], respectively.

2 Translation to the Algebra

Based on the SPARQL grammar the SPARQL specification defines "the process of converting graph patterns and solution modifiers in a SPARQL query string into a SPARQL algebra expression" [HS12, Section 18.2]. This process must be adjusted to consider the extended grammar introduced in the previous section. In the following we discuss all those steps of the conversion process that require adjustment.

2.1 Expand Syntax Forms

The translation process starts with expanding "abbreviations for IRIs and triple patterns" [HS12, Section 18.2.2.1]. This step must be extended in two ways:

1. Abbreviations for triple patterns with embedded triple reference patterns must be expanded as if each embedded triple reference pattern was a variable (or an RDF term). For instance, the following syntax expression

   ```
   <<?c a rdfs:Class>> dct:source ?src ;
                       prov:wasDerivedFrom <<?c a owl:Class>> .
   ```

   must be expanded to

   ```
   <<?c a rdfs:Class>> dct:source ?src .
   <<?c a rdfs:Class>> prov:wasDerivedFrom <<?c a owl:Class>> .
   ```

2. Abbreviations for IRIs in all triple reference patterns must be expanded. For instance, the triple reference pattern

   ```
   <<?c a rdfs:Class>>
   ```

   must be expanded to

   ```
   <<?c <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2000/01/rdf-schema#Class>>>
   ```

2.2 Translate BIND Clauses with a Triple Reference Pattern

The extended grammar in Section 1 allows for BIND clauses with a triple reference pattern. For translating such a BIND clause to an algebra expression we introduce a new algebra symbol:

- TR(variable or IRI or literal , variable or IRI , variable or IRI or literal , variable)

The translation of `BIND ( <<s p o>> AS ?v )` is defined as follows:

```
If s is a triple reference pattern
    Let vs := a fresh variable
Else
    Let vs := s
End

If o is a triple reference pattern
    Let vo := a fresh variable
Else
    Let vo := o
End

Let A := TR(vs,p,vo,?v)

If s is a triple reference pattern
    A: = PrependTR( s, vs, A )
End
```

```
If o is a triple reference pattern
    A: = PrependTR( o, vo, A )
End

The result is A
```

The translation makes use of a function `PrependTR`. This function recursively un-nests nested triple reference patterns. `PrependTR` has three input parameters: a triple reference pattern `<<s p o>>`, a variable `v`, and an algebra expression `A0`. `PrependTR` is defined as follows:

```
Let A := A0

If s is a triple reference pattern
    Let vs := a fresh variable
Else
    Let vs := s
End

If o is a triple reference pattern
    Let vo := a fresh variable
Else
    Let vo := o
End

A := Join( TR(vs,p,vo,v), A )

If s is a triple reference pattern
    A := PrependTR( s, vs, A )
End

If o is a triple reference pattern
    A := PrependTR( o, vo, A )
End

The result is A
```

Notice, the translation of BIND clauses with a triple reference pattern as defined in this section is used during the translation of group graph patterns that is specified in [HS12, Section 18.2.2.6]. The case of BIND clauses with a triple reference pattern is covered in this translation of group graph patterns by the last, "catch all other" `IF` statement (i.e. the `IF` statement with the condition `E is any other form`) and not by the `IF` statement for BIND clauses with an expression.

2.3 Translate Property Path Patterns

Section 18.2.2.3 in [HS12] defines the translation of property path patterns. This translation has to be adjusted because the extended syntax allows for property path patterns that begin with or end with a triple reference pattern (cf. Section 1).

The translation as specified in [HS12, Section 18.2.2.3] distinguishes four cases. The first three of these cases do not require adjustment because the adjusted translation of basic graph patterns (as defined in the following section) shall take care of them. However, the fourth case must be adjusted as follows.

Let `X P Y` be a property path expression that corresponds to the fourth case in [HS12, Section 18.2.2.3]. Given the grammar introduced in Section 1, `X` and `Y` may be an RDF term, a variable, or a triple reference pattern, respectively. The translation of `X P Y` to an algebra expression makes use of function `PrependTR` (cf. Section 2.2) and is defined as follows:

```
If X is a triple reference pattern
    Let vx := a fresh variable
```

```
Else
    Let vx := X
End

If Y is a triple reference pattern
    Let vy := a fresh variable
Else
    Let vy := Y
End

Let A := Path(vx,P,vy)

If X is a triple reference pattern
    A: = PrependTR( X, vx, A )
End

If Y is a triple reference pattern
    A: = PrependTR( Y, vy, A )
End

The result is A
```

2.4 Translate Basic Graph Patterns

After translating property path patterns, [HS12, Section 18.2.2.4] requires to translate each block of adjacent triple patterns into a basic graph pattern. This step has to be adjusted because triple patterns in the extended syntax may have a triple reference pattern in their subject position or in their object position (or in both). Such triple patterns cannot simply be added to basic graph patterns because in algebra expressions we do not allow for triple patterns that contain a triple reference pattern. As a consequence, each block of adjacent triple patterns must be translated as follows.

For a block of adjacent triple patterns let `R` denote the set of all triple reference patterns that occur in a triple pattern of the block. For each triple reference pattern in `R` we introduce a unique, fresh variable. The translation of the block makes use of function `PrependTR` (cf. Section 2.2) and is defined as follows:

```
Let A := the basic graph pattern that can be obtained from the block of adjacent
         triple patterns after replacing each occurrence of any triple reference
         pattern by the variable that has been introduced for the triple reference
         pattern

For each triple reference pattern trp in R
    Let v := the variable that has been introduced for trp
    A := PrependTR( trp, v, A )

The result is A
```

2.5 Variable Scope

Section 18.2.1 in [HS12] introduces a notion of *in-scope variables*. We extend this definition for the new syntax elements introduced in Section 1.

Definition 2.1 A variable `v` is *in-scope* if either:

- it satisfies one of the cases in the table in [HS12, Section 18.2.1], or
- `v` occurs in any triple reference pattern (including in embedded triple reference patterns or triple reference patterns in BIND clauses). □

3 Evaluation Semantics

Section 18.6 in [HS12] defines "eval(D(G), algebra expression) as the evaluation of an algebra expression with respect to a dataset D having active graph G." For algebra expressions that use the new algebra symbol TR, introduced in Section 2.2, the evaluation is defined as follows:

Definition 3.1 Let P be an algebra expression of the form $\mathsf{TR}(s, p, o, v)$. Recall, s is an IRI, an RDF literal, or a variable; p is an IRI or a variable; o is an IRI, an RDF literal, or a variable; and v is a variable. We define:

eval(D(G), P) = multiset Ω of solution mappings

such that for each $\mu \in \Omega$ holds:

1. $\mathrm{dom}(\mu) = \{v\} \cup \big\{x \in \{s, p, o\} \,|\, x \text{ is a variable}\big\}$,
2. RDF triple $\big(\mu(v), \texttt{rdf:type}, \texttt{rdf:Statement}\big)$ is in G,
3. if s is a variable, then RDF triple $\big(\mu(v), \texttt{rdf:subject}, \mu(s)\big)$ is in G,
4. if s is not a variable, then RDF triple $\big(\mu(v), \texttt{rdf:subject}, s\big)$ is in G,
5. if p is a variable, then RDF triple $\big(\mu(v), \texttt{rdf:predicate}, \mu(p)\big)$ is in G,
6. if p is not a variable, then RDF triple $\big(\mu(v), \texttt{rdf:predicate}, p\big)$ is in G,
7. if o is a variable, then RDF triple $\big(\mu(v), \texttt{rdf:object}, \mu(o)\big)$ is in G,
8. if o is not a variable, then RDF triple $\big(\mu(v), \texttt{rdf:object}, o\big)$ is in G, and
9. $\mathrm{card}[\Omega](\mu) = 1$. □

References

[HS12] Steve Harris and Andy Seaborne. SPARQL 1.1 Query Language. Editors' Draft, May 2012. http://www.w3.org/2009/sparql/docs/query-1.1/rq25.xml.