SPARQL Algebra

Andy Seaborne

SPARQL Algebra

- SPARQL Syntax => SPARQL Algebra
- Algebra works on
  - MultiSets (= bags) for pattern matching
  - sequences for solution modifiers
- Spec defines the algorithm for translating syntax to algebra
- Spec defines the correct results for evaluation of an algebra expression
  - Implementations are free to choose any way that gives the same results

SPARQL Pattern Semantics

- Bottom-up evaluation
  - Meaning for all queries
  - All syntactically correct queries have defined semantics
- Optional/LeftJoin is conditional
  - Makes queries more natural to write
- Account of pattern matching and solution modifiers

Basic Graph Patterns

- Set of triple patterns
- Building block in SPARQL
- Extension point for other levels of entailment
  - Blank nodes are extensional variables

Compatible Solutions

- Solutions
  - set of variable/value pairs
  - each variable name occurs at most once.
- Two solutions are *compatible* if variables of the same name are associated with the same RDF term
  - "same" means term-equals

```
S0: { }
S1: { ?x/"foo" , ?y/<http://example/y> }
S2: { ?y/<http://example/y> }
S3: { ?x/"bar" }

S0, S1 and S2 are compatible
S0 and S3 are compatible
S2 and S3 are compatible
S1 and S3 are not compatible
```


SPARQL Algebra

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT *
WHERE
  { ?x foaf:name ?name }
```

```
(tolist
  (BGP [triple ?x foaf:name ?name]))
```

- BGP of one triple pattern
- Group of one element
- Removed: Join({}, P) ➔ P

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  *
WHERE
  { ?x  foaf:mbox  ?mbox ;
        foaf:name  ?name .
  }
```

```
(tolist
  (BGP
    [triple ?x foaf:mbox ?mbox]
    [triple ?x foaf:name ?name]
  ))
```

- BGP of two triple patterns
- Group of one element
- BGP triples are not joined together : BGP is the fundamental unit

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  ?name ?mbox
WHERE
  { ?x  foaf:mbox  ?mbox ;
        foaf:name  ?name .
  }
```

```
(project (?mbox ?name)
  (tolist
    (BGP
      [triple ?x foaf:mbox ?mbox]
      [triple ?x foaf:name ?name]
    )))
```

- Project modifier

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  ?name
WHERE
  { ?x  foaf:name  ?name .
    FILTER regex(?name, "^Smith")
  }
```

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  ?name
WHERE
  { FILTER regex(?name, "^Smith")
    ?x  foaf:name  ?name .
  }
```

```
(project (?name)
  (tolist
    (filter (regex ?name "^Smith")
      (BGP [triple ?x foaf:name ?name]))))
```

- FILTER
- Filter over the whole group

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT  *
WHERE
  { ?x  foaf:name  ?name .
    ?x  foaf:mbox  <mailto:xyz> .
    OPTIONAL
      { ?x  foaf:nick  ?nick }
  }
```

```
(tolist
  (leftjoin
    (BGP
      [triple ?x foaf:name ?name]
      [triple ?x foaf:mbox <mailto:xyz>]
    )
    (BGP [triple ?x foaf:nick ?nick])
    true
  ))
```

- OPTIONAL => LeftJoin
- No explicit null – variables left unbound

SPARQL Algebra

```
PREFIX  foaf: <http://xmlns.com/foaf/0.1/>

SELECT *
WHERE
  { ?x  foaf:name  ?name .
    ?x  foaf:mbox  <mailto:xyz> .
    OPTIONAL
      { ?x  foaf:nick  ?nick
        FILTER regex(?name, "^Smith")
      }
  }
```

```
(tolist
  (leftjoin
    (BGP
      [triple ?x foaf:name ?name]
      [triple ?x foaf:mbox <mailto:xyz>]
    )
    (BGP [triple ?x foaf:nick ?nick])
    (regex ?name "^Smith")))
```

- Conditional LeftJoin
- filter scope includes OPTIONAL left-hand side (fixed part)
  - Here, ?name is available to the regex()

SPARQL Algebra

BasicGraphPattern (BGP)
Filter
Join
LeftJoin
Union

ToList
OrderBy
Distinct
Reduced
Project
Slice

SPARQL Algebra

```
Step 0 : Expand abbreviations for IRIs and triple patterns.

Step 1 : BasicGraphPatterns

Replace all BasicGraphPattern elements by BGP(list of triple patterns)

Step 2 : GroupOrUnionGraphPattern

Replace any GroupOrUnionGraphPattern elements:

    * If the element consists of a single GroupGraphPattern,
         replace with the GroupGraphPattern.
    * If the element consists of multiple GroupGraphPatterns,
         connected with 'UNION' terminals, then
         replace with a sequence of nested union operators:
      e.g. Union(Union(GroupGraphPattern, GroupGraphPattern), GroupGraphPattern).

Step 3 : GraphGraphPattern

Map GRAPH IRI GroupGraphPattern to Graph(IRI, GroupGraphPattern)

Map GRAPH Var GroupGraphPattern to Graph(var, GroupGraphPattern)

Step 4 : . . .
```

SPARQL Algebra

```
Step 4 : GroupGraphPattern

Map all sub-patterns contained in this group
Let SP := List of algebra expressions for sub-patterns
Let F := all filters in the group (not in sub-patterns)
Let G := the empty pattern, {}

for i := 0 ; i < length(SP); i++
   If SP[i] is an OPTIONAL,
      If SP[i] is of the form OPTIONAL(Filter(F, A))
           G := LeftJoin(G, A, F)
      else
           G := LeftJoin(G , A, true)
   Otherwise for expression SP[i], G := Join(G, SP[i])

If F is not empty:
  If G = empty pattern then G := Filter(F, empty pattern)
  If G = LeftJoin(A1, A2, true) then G := LeftJoin(A1, A2, F)
  If G = Join(A1, A2) then G := Filter(F, Join(A1, A2)
  If G = Union(A1, A2) then G := Filter(F, Union(A1, A2))
  If G = Graph(x, A) then G := Filter(F, Graph(x, A))
         where x is a variable or IRI.
The result is G

Step 5 : ...
```


SPARQL Algebra

```
Step 5 : Simplification

Groups of one graph pattern (not a filter) become join({}, A) and can be replaced by A

Replace join({}, A) by A
Replace join(A, {}) by A

Solution Modifiers

Step 1 : ToList
    Let M := ToList(Pattern)
Step 2 : Order By
    M := OrderBy(M, list of order comparators)
Step 3 : Projection
    M := Project(M, vars)
Step 4 : Distinct
    M := Distinct(M)
Step 5 : Reduced
    M := Reduced(M)
Step 5 : OFFSET and LIMIT
    M := Slice(M, start, length)
Result is M
```