SPARQL 1.1

Peter Fischer
DMQL

SPARQL 1.0 limitations

- Limited graphs operations: How to compute connectedness?
- No updates
- No aggregates
- No explicit negation
- No subqueries
- …

Property Paths – Motivation

- RDF is a graph data model, expressed as set of node-edge-node triples
- SPARQL allows us to ask queries on these graphs.
- Basic primitve: selecting *individual* triples using patterns
- Combinations of triples need to be stated explicitly

Property Paths – Motivation (2)

- Many interesting graph algorithms need a more general way to select triples or „paths“ between nodes:
  - In a social network, is there a connection between me and Kevin Bacon
    (and if yes, is it really 6 degrees of separation)
  - What is my complete list of ancestors?
  - How can I retrieve the entire graph?
- What can we do in SPARQL 1.0?
  - Fixed-length paths via BGP, UNION, OPTIONAL
  - No Recursion (as in XQuery, modern SQL)
  - No arbitrary graph paths

Property Paths - Idea

- Permit paths (=sequence of triples) with possibly unbounded length
- Describe properties of this path
- Trivial case: single triple pattern
- Complex paths:
  - Extend triple pattern syntax in to include a more powerful „middle part“
  - borrow regular expression syntax
  - Variables possible at the start and end
  - Allow cycles

Property Paths - Syntax

- `elt`: any path element (recursively defined)
- `IRI`: single “step” (like a predicate)
- `^elt`: inverse direction (object->predicate)
- `!IRI`: negated property
- `(elt)`: group (for precedence)
- `elt1/elt2`: sequence of elt1 followed by elt2
- `elt1|elt2`: alternative, either elt1 or elt2 possible
- `elt*, elt+, elt?`: zero or more, one or more, one or zero elt
- `elt{n,m}`: between n and m occurences of elt
- `elt{n}, elt{n,}, elt{,n}`: exactly n, at least n, at most n

Property Paths - Examples

- Alternative
```
{ :book1 dc:title|rdfs:label ?displayString }
```
- Sequence: name of people that Alice knows
```
{
?x foaf:mbox <mailto:alice@example> .
?x foaf:knows/foaf:name ?name .
}
```
- Same as above, but two steps away
```
{
?x foaf:mbox <mailto:alice@example> .
?x foaf:knows{2}/foaf:name ?name .
}
```
- Arbitrary distance
```
{ ?x foaf:mbox <mailto:alice@example> .
?x foaf:knows+/foaf:name ?name . }
```

Property Paths – More examples

- Negated Property Paths: Find nodes connected but not by rdf:type (either way round)

```
{ ?x !(rdf:type|^rdf:type) ?y }
```

- Multiple paths

```
@prefix : <http://example/> .
:x :p :z1 .
:x :p :z2 .
:z1 :q :y .
:z2 :q :y .

PREFIX : <http://example/>
SELECT * { ?s :p/:q ?o . }
```

What should be the expected result?

Property Paths –Semantics

- All duplicates are being returned/counted
- Is this a good idea?

- Consider a fully connected graph with N nodes, same predicate p (clique)
- How many results are there for {?a (p*)* ?)
- N = 1: 1 N = 3: 6 N=4: 305 N=5: 418657
  N= 8: 79 x 10^{24} (Yottabytes!)
- WWW12 Best Paper by Arenas, Conca, Perez
- Existential semantics do scale, however!

Extended operations with solutions

- SPARQL 1.0 only allows limited operations on matching results/solutions
  - Filter/Duplicate elimination/Ordering
  - Projection
  - Triple construction (CONSTRUCT)
- Need to provide more flexible operations
  - Aggregates
  - Grouping
  - Assignment
  - Select expressions

SELECT expressions

- More flexible rules on SELECT
  - Bind new variables
  - Perform operations on variables

```
PREFIX dc:
<http://purl.org/dc/elements/1.1/> PREFIX
ns: <http://example.org/ns#>
SELECT ?title (?p*(1-?discount) AS ?price)
{ ?x ns:price ?p .
?x dc:title ?title .
?x ns:discount ?discount }
```

Aggregates

- Provide the usual suspects:
  - COUNT, SUM, MIN, MAX, AVG
  - SUM, AVG working on numeric values
- Slightly more unusual
  - GROUP_CONCAT: Concatenate all values to a string
  - SAMPLE: Return arbitrary value from set
  - DISTINCT can be used for all arguments
- Compute results over a group of bindings

GROUP BY

- Usual Syntax: GROUP BY Expression+
- Can bind new variables
- Further restrict using HAVING
- Projection list can only contain group variables and aggregates

Aggregate+Group Example

```
PREFIX : <http://data.example/>
SELECT (AVG(?size) AS ?asize)
WHERE { ?x :size ?size }
GROUP BY ?x
HAVING(AVG(?size) > 10)
```

Subqueries

- Embed a SPARQL query into another
- Possible use cases: complex correlations
  „ Return a name (the one with the lowest sort order) for all the people that know Alice and have a name.”

```
PREFIX : <http://people.example/>
SELECT ?y ?minName
WHERE {
     :alice :knows ?y .
   {
     SELECT ?y (MIN(?name) AS ?minName)
       WHERE { ?y :name ?name . }
      GROUP BY ?y
    }
}
```

„Negation“ in 1.0

```
# Names of people who have not stated that they know anyone
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:givenName ?name .
    OPTIONAL { ?x foaf:knows ?who } .
FILTER (!BOUND(?who)) }
```

What are we doing here?

$\Rightarrow$ Not very intuitive

Two solutions in 1.1

1. NOT EXISTS
2. MINUS

Negation via NOT EXISTS

```
PREFIX foaf:
<http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:givenName ?name .
FILTER (NOT EXISTS
          {?x foaf:knows ?who })
}
```

- NOT EXISTS is a filter function that yields true of a binding does not exists
- There is now also a EXISTS

Negation via MINUS

```
PREFIX foaf:
<http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { ?x foaf:givenName ?name .
     MINUS { ?x foaf:knows ?who } .
}
```

- MINUS is a graph Pattern Match (like UNION, OPTIONAL)
- Removes Bindings that match

Entailment

- Recall entailment? Adding semantics and metadata, we can generate new triples/facts
- Entailment affects triple matching: we may find additional triples which were not present in the original (axiomatic) triples
- SPARQL 1.0 only considered simple entailment
- SPARQL 1.1 provides
  - Detailed rules how entailment should work
  - Descriptions for different entailment standards (RDF, RDFS, OWL, …)

Some entailment effects

- RDF entailment
  - blank nodes (consistent in answers)
  - XML Literals
  - Properties
- RDFS entailment
  - Can lead to inconsistencies (fewer answers!) Here only due to invalied XML Literals
  - Derived results due to new tuples

Entailment example

ex:book1 a ex:Publication .
ex:book2 a ex:Article .
ex:Article rdfs:subClassOf ex:Publication .
ex:publishes rdfs:range ex:Publication .
ex:MITPress ex:publishes ex:book3 .

SELECT ?pub WHERE { ?pub a ex:Publication }

What are the results under

- Simple entailment ?
- RDF entailment ?
- RDFS entailment ?

Updates

- SPARQL 1.0 is read-only
- Changes to graphs need to be done using other languages or proprietary extensions
- SQL and XQuery have update languages

- SPARQL 1.1 has two update mechanism:
  1. Language-based updates (like SQL, XQuery)
  2. REST API: Graph Store operations via HTTP

Update - Concepts

Graph Store

- Collection of graphs, default+named
- Does not need to be authoritative (Cache!)
- Local operations should be atomic

- Two classes of operations:
  1. Modifying triples in graphs
  2. Managing complete graphs

INSERT into a graph

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA {
<http://example/book1> dc:title "A new book";

dc:creator "A.N.Other" .
}
```

@prefix dc: <http://purl.org/dc/elements/1.1/> .

@prefix ns: <http://example.org/ns#> .

<http://example/book1> ns:price 42 .

<http://example/book1> dc:title "A new book" . <http://example/book1> dc:creator "A.N.Other" .

- Optionally a graph name
- Triples must not contain variables
- What happens if a triple with the same values is already present?

DELETE from a graph

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
DELETE DATA {
<http://example/book2> dc:title "David
     Copperfield" ;
            dc:creator "Edmund Wells" .
}
```

@prefix dc: <http://purl.org/dc/elements/1.1/> .

@prefix ns: <http://example.org/ns#> .

<http://example/book2> ns:price 42 .

~~<http://example/book2> dc:title "David Copperfield" .~~
~~<http://example/book2> dc:creator "Edmund Wells" .~~

- No variables or blank nodes
- Entailed triples will not be deleted

Parameterized Delete/Insert

(**WITH** *IRIref*)?
((*DeleteClause* *InsertClause*?) | *InsertClause*)
(**USING (NAMED)?** *IRIref*)*
WHERE *GroupGraphPattern*

DeleteClause ::= **DELETE** *QuadPattern*
InsertClause ::= **INSERT** *QuadPattern*

- Match triples in WHERE,
  perform delete, then insert with bindings
  (Why?)
- Triples in WHERE can be from a different store/graph (USING) than updated graph (WITH)
- Shorthands for DELETE only/INSERT only

Update Example

Rename all “Bills” to “William”

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
WITH http://example/addresses
DELETE { ?person foaf:givenName 'Bill' }
INSERT { ?person foaf:givenName 'William' }
USING  http://example/addresses
WHERE { ?person foaf:givenName 'Bill' }
```

Complex Filter+Moving Example

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
INSERT
{ GRAPH <http://example/bookStore2>
  { ?book ?p ?v }
}
WHERE
{ GRAPH <http://example/bookStore>
 { ?book dc:date ?date .
  FILTER (
  ?date > "1970-01-01T00:00:0002:00
           ^^xsd:dateTime)
  ?book ?p ?v
 }
}
```

Copy all book published from 1970 onwards into bookstore2

Bulk operations

- LOAD *uri* [INTO GRAPH *uri*]
  Load all triples from uri into the graph
- CLEAR [GRAPH *uri* | DEFAULT | NAMED | ALL]
  Delete all triples from the graph(s)

Graph Management

- **CREATE** (**SILENT**)? **GRAPH** *IRIref*
- **DROP** (**SILENT**)? (**GRAPH** *IRIref* | **DEFAULT** | **NAMED** | **ALL**)
- **COPY** (**SILENT**)? ((**GRAPH**)? *IRIref_from* | **DEFAULT**) **TO** ((**GRAPH**)? *IRIref_to* | **DEFAULT**)
  copy all triples from *IRIref_from* to *IRIref_to*, *overwrite all contents of* *IRIref_to*
- **MOVE** (**SILENT**)? ((**GRAPH**)? *IRIref_from* | **DEFAULT**) **TO** ((**GRAPH**)? *IRIref_to* | **DEFAULT**)
  as COPY, just delete the source
- **ADD** (**SILENT**)? ((**GRAPH**)? *IRIref_from* | **DEFAULT**) **TO** ((**GRAPH**)? *IRIref_to* | **DEFAULT**)
  as COPY, keep contents of *IRIref_to*

Graph Store Protocol

- SPARQL 1.0 already defined a protocol to query RDF data over the network (=> Linked Open Data)
- Extend Protocol to manage Graph Stores
- Use REST vocabulary
  - Use Graph URI/IRI as location (directly or as parameter)
  - PUT to COPY a graph
  - DELETE to DROP a graph
  - POST to ADD triples to a graph
  - GET to return a entire graph (CONSTRUCT)

Other new features

- Explicit support for federated data: SERVICE keyword to invoke parts of a query at remote endpoints
- Service description: provide capabilities, vocabulary of a SPARQL endpoint
- Short form for CONSTRUCT (state graph and bindings only once)
- Many new functions:
  - EXISTS/NOT EXISTS, IN/NOT IN
  - String manipulation
  - Math
  - Date/Time accessors, current dateTime
  - Hashing

Summary

- SPARQL 1.1 fixes many shortcomings of 1.0
- Feature set closer to other classical query languages
- Introduction of significant complexity (property paths, subqueries)
- What is still missing?
  - Fulltext operations
  - Integration with application development