SPARQL 1.1

Werner Nutt

Acknowledgment

These slides are essentially identical with those by Sebastian Rudolph for his course on Semantic Web Technologies at TU Dresden

Expressions in the Selection and Bindings

Solutions can be extended by evaluated expressions with

```
(expression AS ?var)
```

used for the assignment:

- in the `SELECT` clause
- in the `GROUP BY` clause
- within `BIND` in a group graph pattern

Solutions from a group can further be joined with solutions given via a `VALUES` pattern

Example BIND (without Prefix Declarations)

Data

```
ex:Book       ex:title      "SPARQL Tutorial" ;
              ex:price      42 ;
              ex:discount   10 .
```

Query

```
SELECT ?title ?price WHERE
{ ?b   ex:title     ?title;
       ex:price     ?p ;
       ex:discount ?r
BIND ((?p-?r) AS ?price) }
```

Result

```
?title → "SPARQL Tutorial", ?price → 32
```

➔ Algebra: Extend(Bgp(...), ?price, (?p-?r))

Example SELECT Expressions (without Prefix Declarations)

Data

```
ex:Book       ex:title      "SPARQL Tutorial" ;
              ex:price      42 ;
              ex:discount   10 .
```

Query

```
SELECT ?title ((?p-?r) AS ?price WHERE
{ ?b   ex:title     ?title;
       ex:price     ?p ;
       ex:discount  ?r
}
```

Result

```
?title → "SPARQL Tutorial", ?price → 32
```

➔ Algebra: Extend(Bgp(...), ?price, (?p-?r))

Example VALUES

Data

```
ex:Book1      ex:title      "SPARQL Tutorial".
ex:Book2      ex:title      "SemWeb".
```

Query

```
SELECT ?title WHERE {
 ?b ex:title ?title
 VALUES ?b { ex:Book1 }
}
```

Result

```
?title → "SPARQL Tutorial"
```

➔ Bindings are conjunctively joined

Aggregates

Aggregates allow for

- the grouping of solutions and
- the computation of values over the groups

Example Query

```
SELECT ?lecture (COUNT(?student) AS ?c)
WHERE  { ?student ex:attends ?lecture }
GROUP BY ?lecture
HAVING COUNT(?student) > 5
```

- `GROUP BY` groups the solutions
  (here into students who attend the same lecture)
- `COUNT` is an aggregate function that counts the solutions within a group (here the number of students in the lecture)
- `HAVING` filters aggregated values

Aggregates in SPARQL 1.1

SPARQL 1.1 supports the following aggregate functions, which are evaluated over the values in a group:

- `COUNT` – counts the solutions
- `MIN` – finds the minimal value
- `MAX` – finds the maximal value
- `SUM` – sums up the values
- `AVG` – computes the average
- `SAMPLE` – picks a random value
- `GROUP_CONCAT` – string concatenation,

  Example: `GROUP_CONCAT(?x ; separator = ",")`

Note: Most SPARQL 1.1 implementations only concatenate strings!

Exercise Aggregates

Data

```
ex:Paul   ex:hasMark   28.0 .
ex:Paul   ex:hasMark   24.0 .
ex:Mary   ex:hasMark   25.0 .
ex:Peter  ex:hasMark   22.0 .
```

- Return those students (with their average marks) that have an average mark > 25.
- Return the subgraph of the data consisting of students with an average mark > 25 and their marks.
- Return those that have an average mark > 25 together with a list of their marks, separated by “|”.

Show also what you expect to be the solution.

Subqueries

Example Query

```
SELECT ?name WHERE {
 ?x foaf:name ?name .
 { SELECT ?x (COUNT(*) AS ?count)
   WHERE { ?x foaf:knows ?y . }
   GROUP BY ?x
   HAVING (?count >= 3)
 }
}
```

- Results for the inner query are conjunctively joined with the results of the outer query

Negation in Queries

Two forms of negation with conceptual and small semantic differences

① Test for non-matches for a pattern

② Removal of matching solutions

Filter

```
SELECT ?x WHERE {
 ?x  rdf:type  foaf:Person .
 FILTER NOT EXISTS { ?x foaf:name ?name }
}
```

Minus

```
SELECT ?x WHERE {
 ?x  rdf:type  foaf:Person .
 MINUS { ?x foaf:name ?name }
}
```

What are the corresponding constructs in SQL?
What is the difference between them in SQL?

Evaluation of Negation via Filter

Data

```
_:x  rdf:type   foaf:Person .
_:x  foaf:name  "Peter" .
_:y  rdf:type   foaf:Person .
```

Query Pattern

```
{ ?x rdf:type foaf:Person .
  FILTER NOT EXISTS { ?x foaf:name ?name } }
```

① [[Bgp(1st Pattern)]]$_G$: μ_1: ?x → _:x, μ_2: ?x → _:y

② For each solution, we instantiate the second pattern

- Solution is removed if the instantiated pattern matches (μ_1)
- otherwise we keep the solution (μ_2)

Evaluation of Negation via Minus

Data

```
_:x  rdf:type   foaf:Person .
_:x  foaf:name  "Peter" .
_:y  rdf:type   foaf:Person .
```

Query Pattern

```
{ ?x rdf:type foaf:Person .
  MINUS { ?x foaf:name ?name } }
```

[[Bgp(1st Pattern)]]$_G$: $\Omega_1 = \{ \mu_1$: ?x → _:x, μ_2: ?x → _:y $\}$

[[Bgp(2nd Pattern)]]$_G$: $\Omega_2 = \{ \mu_3$: ?x → _:x, ?name → "Peter" $\}$

$$\text{Minus}(\Omega_1, \Omega_2) = \{\mu \mid \mu \in \Omega_1 \text{ and } \forall \mu' \in \Omega_2 : \mu \text{ and } \mu' \text{ incompatible or } \text{dom}(\mu) \cap \text{dom}(\mu') = \varnothing\}$$

$\mu_1 \notin \Omega$: μ_1 compatible with μ_3 and has non-disjoint domain

$\mu_2 \in \Omega$: μ_2 incompatible with μ_3

Minus and Filter Negation: Differences

Data

```
ex:a ex:b ex:c .
```

Query Pattern

```
{ ?s ?p ?o FILTER NOT EXISTS { ?x ?y ?z } }
```

- Filter pattern matches always (variables disjoint)

➔ every solution is removed

Query Pattern

```
{ ?s ?p ?o MINUS { ?x ?y ?z } }
```

- Minus does not remove any solutions
  since the domain of the solutions is disjoint

Regular Expressions in Patterns

Property paths are constructed using
regular expressions over predicates

- Sequence of paths: `?s ex:p1/ex:p2 ?o`
- Optional path: `?s ex:p? ?o`
- Paths with arbitrary length: `?s ex:p+ ?o`, `?s ex:p* ?o`
- Alternative paths: `?s (ex:p1|ex:p2) ?o`
- Inverse paths: `?s ^ex:p ?o`
  same as `?o ex:p ?s`
- Negation of paths: `?s !ex:p ?o`

A negation of path expression `!ex:p` matches a path
that does *not have* predicate `ex:p`

Regular Expressions in Patterns

- Property paths are, where possible, translated into standard SPARQL constructs

  Examples?

- Some new operators are still necessary

Property Path Examples

Example Query 1

```
PREFIX ...
SELECT ?xName WHERE {
   ?x  rdf:type   foaf:Person .
   ?x  foaf:name  ?xName
   ?x  foaf:knows/foaf:knows/foaf:name  "Bill Gates" .
}
```

Example Query 2

```
PREFIX ...
SELECT ?s WHERE {
   ?s    rdf:type    ?type .
   ?type rdfs:subClassOf*  ex:SomeClass .
}
```