Introduction to SPARQL

Acknowledgements

- This presentation is based on the W3C Candidate Recommendation “SPARQL Query Language for RDF” from http://www.w3.org/TR/rdf-sparql-query/

- Some of the material in this presentation is verbatim from the above Web site.

Presentation Outline

- Query languages for RDF and RDFS
- SPARQL: A Query Language for RDF
- Semantics of SPARQL

Query Languages for RDF and RDFS

- There have been many proposals for RDF and RDFS query languages:
  - RDQL (http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/)
  - ICS-FORTH RQL (http://139.91.183.30:9090/RDF/RQL/) and SeRQL (http://www.openrdf.org/doc/sesame/users/ch06.html)
  - SPARQL (http://www.w3.org/TR/rdf-sparql-query/)
  - …

In this course we will only cover **SPARQL which is the current W3C recommendation for querying RDF data.**

SPARQL

- SPARQL stands for “SPARQL Protocol and RDF Query Language”.

- In addition to the language, W3C has also defined:
  – The **SPARQL Protocol** for RDF specification: it defines the remote protocol for issuing SPARQL queries and receiving the results.
  – The **SPARQL Query Results** XML Format specification: it defines an XML document format for representing the results of SPARQL queries.

SPARQL 1.1

- In this lecture we will cover **the SPARQL standard as of 2008.**
- The standardization of SPARQL is carried out under the auspices of the W3C by the **SPARQL working group**.
- More information about ongoing work by this working group can be found at
  - http://www.w3.org/2009/sparql/wiki/Main_Page
- See http://www.w3.org/TR/sparql11-query/ for the new version of the SPARQL language (SPARQL 1.1).

SPARQL Basics

- SPARQL is based on **matching graph patterns against RDF graphs.**

- What is a graph pattern?

- To define graph patterns, we must first define triple patterns:
  - A **triple pattern** is like an RDF triple, but with the option of a variable in place of RDF terms (i.e., IRIs, literals or blank nodes) in the subject, predicate or object positions.

  - Example:

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title> ?title .
```

  - `?title` is a **variable.**

SPARQL Graph Patterns

- We can distinguish the following kinds of graph patterns:
  - **Group** graph patterns. These are the more general case of graph pattern. They are build out of:
    - **Basic** graph patterns
    - **Filter** conditions
    - **Optional** graph patterns
    - **Alternative** graph patterns
  - Patterns on **named graphs**

Basic Graph Patterns

- A **basic graph pattern** (BGP) is a **set of triple patterns** written as a sequence of triple patterns (separated by a period if necessary).
- A BGP should be understood as the **conjunction** of its triple patterns.

- Example:

```
?x foaf:name ?name . ?x foaf:mbox ?mbox
```

Group Graph Patterns

- A **group graph pattern** is a **set** of graph patterns delimited with braces { }.

- **Simple examples:**

```
{ ?x foaf:name ?name . ?x foaf:mbox ?mbox  }

{ ?x foaf:name ?name . ?x foaf:mbox ?mbox . }

{ { ?x foaf:name ?name . }
  { ?x foaf:mbox ?mbox . } }
```

- The above group graph patterns are **equivalent**. In general:
  - When a group graph pattern consists only of triple patterns or only of BGPs, these patterns are interpreted **conjunctively**, and the group graph pattern is equivalent to the corresponding set of triple patterns.

Group Graph Patterns (cont'd)

- `{ }` is the **empty** group graph pattern.

- Group graph patterns are the **most general** kind of graph patterns; they can involve **other constructs** to be defined below. These constructs are introduced by certain **keywords**.

- **Important**: There is **no keyword for conjunction** (e.g., AND) in SPARQL. Conjunctive triple patterns or BGPs are simply juxtaposed and then enclosed in { and } to form a group graph pattern.

A Simple SPARQL Query

- **Data:**

```
<http://example.org/book/book1>
<http://purl.org/dc/elements/1.1/title>
"SPARQL Tutorial" .
```

- **Query:**

```
SELECT ?title
WHERE { <http://example.org/book/book1>
        <http://purl.org/dc/elements/1.1/title>
        ?title . }
```

- **Result:**

title
"SPARQL Tutorial"

Comments

- Data will be presented using **Turtle**. The Turtle syntax is also utilized in SPARQL so it is useful to know it well.
- SELECT and WHERE clauses are like in SQL. But be careful: SPARQL and SQL are very different languages in general.
- Variables are like in Prolog or Datalog.
- Variables can also be written as `$x` instead of `?x`.
- We can write `SELECT *` **like in SQL.**
- The result of a query is a set of **bindings** for the variables appearing in the SELECT clause. Bindings will be shown in tabular form.

Another Example

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Johnny Lee Outlaw" .
_:a foaf:mbox <mailto:jlow@example.com> .

_:b foaf:name "Peter Goodguy" .
_:b foaf:mbox <mailto:peter@example.org> .

_:c foaf:mbox <mailto:carol@example.org> .
```

Example (cont'd)

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name . ?x foaf:mbox ?mbox }
```

- **Result:**

name	**mbox**
“Peter Goodguy"	<mailto:peter@example.org>
"Johnny Lee Outlaw”	<mailto:jlow@example.com>

Queries with RDF Literals

- We have to be careful when matching RDF literals (see the SPARQL specification for all the details). For example:

- **Data:**

```
@prefix dt: <http://example.org/datatype#> .
@prefix ns: <http://example.org/ns#> .
@prefix : <http://example.org/ns#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

:x ns:p "cat"@en .
:y ns:p "42"^^xsd:integer .
:z ns:p "abc"^^dt:specialDatatype .
```

Matching RDF Literals (cont'd)

- The queries

  `SELECT ?v WHERE { ?v ?p "cat" }`

  and

  `SELECT ?v WHERE { ?v ?p "cat"@en }`

  have different results.

- Only the second one finds a matching triple and returns:

v
<http://example.org/ns#x>

Blank Nodes in Query Results

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
```

- **Query:**

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

- **Result:**

x	name
_:c	"Alice"
_:d	"Bob"

Blank Nodes in Query Results (cont'd)

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?x ?name1 ?y ?name2
WHERE { ?x foaf:name ?name1 . ?y foaf:name ?name2 .
        ?x foaf:knows ?y }
```

- **Result:**

?x	name1	?y	name2
_:c	"Alice"	_:d	"Bob"
_:d	"Bob"	_:c	"Alice"

Comments

- SPARQL does not consider blank nodes to be something like existentially quantified variables in FOL as semantics of RDF do!

- SPARQL considers blank nodes to be **distinct constants scoped to the graph where they appear.**

- **Example:** If we ask in the previous graph "How many resources with a name do we have?", the answer is 2.

- See the paper

  A. Mallea, M. Arenas, A. Hogan and A. Polleres. On Blank Nodes.
  Proc. of ISWC 2011.
  Available from http://axel.deri.ie/publications.html for a comprehensive discussion of issues relating to blank nodes in the theory and practice of RDF and SPARQL.

Blank Nodes in Graph Patterns

- Blank nodes in graph patterns act as **variables**, not as references to specific blank nodes in the data being queried.
- Blank nodes **cannot appear** in a SELECT clause.
- The **scope** of blank node is the BGP in which it appears. A blank node which **appears more than once** in the same BGP stands for the same RDF term.
- The same blank node is **not** allowed to appear in two BGPs of the same query.
- **Important:** there is no reason to use blank nodes in a query; you can get the same functionality using variables.

Example

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE { _:z foaf:name ?name . }
```

- **Result:**

name
"Alice"
“Bob”

Example (cont'd)

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1 ?name2
WHERE { _:z foaf:name ?name1 . _:v foaf:name ?name2 .
        _:z foaf:knows _:v }
```

- **Result:**

name1	**name2**
"Alice"	"Bob"
“Bob”	“Alice”

Example (cont’d)

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:b foaf:name "Bob" .
_:a foaf:knows _:b .
_:b foaf:knows _:a .
```

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name1 ?name2
WHERE { {_:z foaf:name ?name1} {_:z foaf:name ?name2} }
```

- **Result:** Error (blank node reused across basic graph patterns).

Query Forms

- The SELECT query form returns **variable bindings.**
- The CONSTRUCT query form returns an **RDF graph** specified by a graph template.
- The ASK query form can be used to **test whether or not a graph pattern has a solution.** No information is returned about the possible query solutions, just whether or not a solution exists.
- There is also a DESCRIBE query form which is not important and SPARQL does not prescribe any semantics for it.

Example - CONSTRUCT

- **Data:**

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

_:a org:employeeName "Alice" .
_:a org:employeeId 12345 .
_:b org:employeeName "Bob" .
_:b org:employeeId 67890 .
```

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX org: <http://example.com/ns#>

CONSTRUCT { ?x foaf:name ?name }
WHERE { ?x org:employeeName ?name }
```

Example (cont’d)

- The **result** now is a **graph**:

```
@prefix foaf: <http://xmlns.com/foaf/0.1/>
_:c foaf:name "Alice" .
_:d foaf:name "Bob" .
```

Examples - ASK

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:homepage <http://work.example.org/alice/> .

_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@work.example> .
```

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK { ?x foaf:name "Alice" }
```

- **Answer:**

```
yes
```

Examples (cont'd)

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
ASK { ?x foaf:name "Alice" ;
         foaf:mbox <mailto:alice@work.example> }
```

- **Answer:**

```
no
```

- **Note:** The answer should be understood as saying: **I couldn't find bindings** to compute a solution to the given graph pattern.

Constraints on Variables

- The FILTER construct **restricts variable bindings** to those for which the filter expression evaluates to TRUE.

Example: Arithmetic Filters

- **Data:**

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .

:book1 dc:title "SPARQL Tutorial" .
:book1 ns:price 42 .

:book2 dc:title "The Semantic Web" .
:book2 ns:price 23 .
```

Example (cont'd)

- **Query:**

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x ns:price ?price .
        FILTER (?price < 30.5)
        ?x dc:title ?title . }
```

- **Result:**

title	**price**
"The Semantic Web"	23

Example: String Filters

- **Query:**

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT  ?title
WHERE   { ?x dc:title ?title
          FILTER regex(?title, "^SPARQL")
        }
```

- **Result:**

title
“SPARQL Tutorial"

Scope of Filters

- **Group graph patterns** are used to restrict the scope of FILTER conditions.

- A FILTER condition is **a restriction on solutions over the whole group** in which the filter appears.

Example

- The following graph patterns all have the same set of solutions:

  - ```
    {  ?x foaf:name ?name .
       ?x foaf:mbox ?mbox .
       FILTER regex(?name, "Smith")   }
    ```
  - ```
    {  FILTER regex(?name, "Smith")
       ?x foaf:name ?name .
       ?x foaf:mbox ?mbox .   }
    ```
  - ```
    {  ?x foaf:name ?name .
       FILTER regex(?name, "Smith")
       ?x foaf:mbox ?mbox .   }
    ```
    ```

Comments

- We can have **multiple FILTERs** in a group graph pattern. They are equivalent to a single filter with conjoined filter conditions.
- FILTERs can be very complex Boolean conditions (see the SPARQL specification for details http://www.w3.org/TR/rdf-sparql-query/).
- The regular expression language used by `regex` is defined in XQuery 1.0 and XPath 2.0.

Optional Graph Patterns

- Regular, complete structures **cannot be assumed** in all RDF graphs.
- It is useful to have queries **that allow information to be added to the answer where the information is available**, but do not reject the answer because some part of the query pattern does not match.
- **Optional graph pattern matching** provides this facility: if the optional part does not match, it creates no bindings but does not eliminate the solution.

Example

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-
  ns#> .

_:a rdf:type foaf:Person .
_:a foaf:name "Alice" .
_:a foaf:mbox <mailto:alice@example.com> .
_:a foaf:mbox <mailto:alice@work.example> .

_:b rdf:type foaf:Person .
_:b foaf:name "Bob" .
```

Example (cont'd)

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        OPTIONAL { ?x foaf:mbox ?mbox }
      }
```

- **Result:**

name	**mbox**
"Alice"	<mailto:alice@example.com>
"Alice"	<mailto:alice@work.example>
"Bob"	

Semantics of Answers

- We can now see that answers to a SPARQL query can be formalized as sets of **mappings** i.e., **partial functions** from the set of variables to the set of RDF terms (URIs, literals and blank nodes).

- Later on we will give a complete formal semantics of SPARQL queries.

Example

- The answer of the previous query can be formalized by the following set of mappings:

```
{
 { ?name → "Alice", ?mbox → <mailto:alice@example.com> }
 { ?name → "Alice", ?mbox → <mailto:alice@work.example> }
 { ?name → "Bob" }
}
```

Optional Graph Patterns (cont’d)

- **Optional parts** of a graph pattern that we are trying to compute may be specified by starting with a graph pattern `P1` and then applying the **keyword** OPTIONAL to another graph pattern `P2` that follows it:

```
P1 OPTIONAL { P2 }
```

Properties of OPTIONAL

- OPTIONAL is a **binary operator.**

- OPTIONAL is **left-associative:**

```
P1 OPTIONAL { P2 } OPTIONAL { P3 }
```

is equivalent to

```
{ P1 OPTIONAL { P2 } } OPTIONAL { P3 }
```

Properties of OPTIONAL (cont'd)

- The syntactic form

  `{ OPTIONAL { P } }`

  is equivalent to

  `{ { } OPTIONAL { P } }.`

- In general

  `{ P1 OPTIONAL P2 } OPTIONAL P3`

  is **not** equivalent to

  `P1 OPTIONAL { P2 OPTIONAL P3 }.`

FILTERs in Optional Pattern Matching

- The group graph pattern following a keyword OPTIONAL can of course be as complex as possible e.g., it can contain a FILTER.

Example

- **Data:**

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix : <http://example.org/book/> .
@prefix ns: <http://example.org/ns#> .

:book1 dc:title "SPARQL Tutorial" .

:book2 dc:title “A New SPARQL Tutorial" .
:book2 ns:price 42 .

:book3 dc:title "The Semantic Web" .
:book3 ns:price 23 .
```

Example (cont'd)

- **Query:**

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?title ?price
WHERE { ?x dc:title ?title .
        OPTIONAL { ?x ns:price ?price .
                   FILTER (?price < 30) }
      }
```

- **Result:**

title	Price
"SPARQL Tutorial"	
"A New SPARQL Tutorial"	
"The Semantic Web"	23

Comments

- Note that the OPTIONAL pattern in the previous query does not generate bindings in the following two cases:
  - There is no `ns:price` property for `?x` (e.g., when `?x=book1`).
  - There is an `ns:price` property for `?x` but its value is greater than or equal to 30 (e.g., when `?x=book2`).

Example with Multiple OPTIONALs

- **Data:**

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

_:a foaf:name "Alice" .
_:a foaf:homepage <http://work.example.org/alice/> .

_:b foaf:name "Bob" .
_:b foaf:mbox <mailto:bob@work.example> .
```

Example (cont'd)

- **Query:**

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox ?hpage
WHERE { ?x foaf:name ?name .
        OPTIONAL { ?x foaf:mbox ?mbox . }
        OPTIONAL { ?x foaf:homepage ?hpage . }
      }
```

- **Result:**

name	**mbox**	**hpage**
"Alice"		<http://work.example.org/alice/>
"Bob"	<mailto:bob@work.example>	

Properties of OPTIONAL (cont’d)

- The operator OPTIONAL **has higher precedence than conjunction** (remember: conjunction is encoded as juxtaposition of graph patterns).

Example

- **Data**

```
@prefix ex:  <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .

ex:book1 dc:creator ex:Smith .
ex:book1 dc:title "Semantic Web" .
ex:book1 ns:price 30 .

ex:book2 dc:creator ex:Jones .
ex:book2 dc:title "SPARQL" .

ex:book3 dc:creator ex:Doyle.
ex:book3 ns:price 34 .

ex:book4 dc:title "RDF" .
ex:book4 ns:price 50 .
```

Example (cont'd)

- **Query 1:**

```
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?book ?title
WHERE  { ?book dc:creator ?author .
         OPTIONAL { ?book dc:title ?title .}
        { ?book ns:price ?price .}
       }
```

- **Answer:**

book	**title**
<http://example.org/book3>	
<http://example.org/book1>	"Semantic Web"

Example (cont’d)

- Give the precedence and associativity of OPTIONAL and conjunction, Query 1 is equivalent to the following query:

```
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?book ?title
WHERE  { { ?book dc:creator ?author .
           OPTIONAL { ?book dc:title ?title .} }
         { ?book ns:price ?price .}
       }
```

- It is interesting to also see Query 2 below which has results different than Query 1.

Example (cont'd)

- **Query 2:**

```
PREFIX ex: <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?book ?title
WHERE  { ?book dc:creator ?author .
         OPTIONAL { { ?book dc:title ?title .}
                    { ?book ns:price ?price .} }
       }
```

- **Answer:**

book	**title**
<http://example.org/book3>	
<http://example.org/book2>	
<http://example.org/book1>	"Semantic Web"

Alternative Patterns (Disjunction)

- SPARQL provides a means of forming the **disjunction of graph patterns** so that one of several alternative graph patterns may match. If more than one of the alternatives match, all the possible pattern solutions are found.

- Pattern alternatives are syntactically specified with the keyword UNION.

Example

- **Data:**

```
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .

_:a dc10:title "SPARQL Query Language Tutorial" .
_:a dc10:creator "Alice" .

_:b dc11:title "SPARQL Protocol Tutorial" .
_:b dc11:creator "Bob" .

_:c dc10:title "SPARQL" .
_:c dc11:title "SPARQL (updated)" .
```

Example (cont'd)

- **Query:**

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { { ?book dc10:title ?title }
        UNION
        { ?book dc11:title ?title }
      }
```

- **Result:**

title
"SPARQL Protocol Tutorial"
"SPARQL"
"SPARQL (updated)"
"SPARQL Query Language Tutorial"

Example (cont'd)

- **Query**:

```
 PREFIX dc10:  <http://purl.org/dc/elements/1.0/>
PREFIX dc11:  <http://purl.org/dc/elements/1.1/>

SELECT ?author ?title
WHERE  { { ?book dc10:title ?title .  ?book dc10:creator ?author . }
         UNION
         { ?book dc11:title ?title .  ?book dc11:creator ?author . }
       }
```

- **Result:**

author	**title**
"Alice"	"SPARQL Query Language Tutorial"
"Bob"	"SPARQL Protocol Tutorial"

Semantics of UNION

- UNION is a **binary operator**.
- Alternative graph patterns that are combined by UNION are processed **independently** of each other and the results are combined using (set-theoretic) union.

Semantics of UNION (cont'd)

- The query

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { { ?book dc10:title ?title }
        UNION
        { ?book dc11:title ?title }
      }
```

gives a result that is the same as the set-theoretic union of the results of the following two queries:

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
SELECT ?title
WHERE { ?book dc10:title ?title }

PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?book dc11:title ?title }
```

Semantics of UNION (cont'd)

- We have to be careful whether or not to use the **same variables** in each alternative (as we did in the previous query). This decision depends on what we want to compute.

Example

- Consider now the following query where different variables are used for title:

```
PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>
SELECT ?x ?y
WHERE { {?book dc10:title ?x} UNION {?book dc11:title ?y} }
```

- **Result:**

x	y
	"SPARQL (Updated)"
	"SPARQL Protocol Tutorial"
"SPARQL"	
"SPARQL Query Language Tutorial"	

Properties of UNION

- **Precedence and associativity:**
  - UNION is **left-associative**.
  - UNION and OPTIONAL have the **same precedence**.
  - UNION has **higher precedence than conjunction** (i.e., juxtaposition of patterns).

- **Commutativity**:
  - `P UNION Q` is equivalent to `Q UNION P`
- **Associativity property**:
  - `{P UNION Q} UNION R` is equivalent to
    `P UNION {Q UNION R}`

Examples of Combining UNION and OPTIONAL

```
{ {s1 p1 o1} UNION {s2 p2 o1}
  OPTIONAL {s3 p3 o3}
}
```

is equivalent to

```
{ { {s1 p1 o1} UNION {s2 p2 o1}
  } OPTIONAL {s3 p3 o3}
}
```

Examples (cont'd)

```
{   {s1 p1 o1} OPTIONAL {s2 p2 o1}
    UNION {s3 p3 o3} OPTIONAL
    {s4 p4 o4} OPTIONAL {s5 p5 o5}
}
```

is equivalent to

```
{ { { { {s1 p1 o1} OPTIONAL {s2 p2 o1}
      } UNION {s3 p3 o3}
    } OPTIONAL {s4 p4 o4}
  } OPTIONAL {s5 p5 o5}
}
```

Examples of Combining UNION and conjunction

```
{ {s1 p1 o1} UNION {s2 p2 o1}
  {s3 p3 o3}
}
```

is equivalent to

```
{ { {s1 p1 o1} UNION {s2 p2 o1}
  }
  {s3 p3 o3}
}
```

- See the difference in the results of Queries 1 and 2 below.

Example

- **Data:**

```
@prefix ex:  <http://example.org/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .

ex:book1 dc:creator ex:Smith .
ex:book1 dc:title "Semantic Web" .

ex:book2 dc:creator ex:Jones .
ex:book2 dc:title "SPARQL" .
ex:book2 ns:price 30 .

ex:book3 dc:creator ex:Jones.
ex:book3 dc:title "RDF" .
ex:book3 ns:price 35 .
```

Example (cont'd)

- **Query 1:**

```
PREFIX ex:  <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?book ?title ?price
WHERE {
      { ?book dc:creator ex:Smith . ?book dc:title ?title . }
      UNION
      { ?book dc:creator ex:Jones . ?book ns:price ?price . }
      }
```

- **Answer:**

book	**title**	**price**
<http://example.org/book1>	"Semantic Web"	
<http://example.org/book3>		35
<http://example.org/book2>		30

Example (cont'd)

- **Query 2:**

```
PREFIX ex:  <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>
SELECT ?book ?title ?price
WHERE {
      { ?book dc:creator ex:Smith . ?book dc:title ?title . }
      UNION
      { ?book dc:creator ex:Jones .} { ?book ns:price ?price . }
      }
```

- **Answer:**

book	title	price
<http://example.org/book3>		35
<http://example.org/book2>		30

Semantics of SPARQL

- The formal semantics of SPARQL can be found in the W3C specification (http://www.w3.org/TR/rdf-sparql-query/#sparqlDefinition).

- We prefer to discuss the semantics and expressive power of SPARQL following the papers

  Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. *Semantics and Complexity of SPARQL*. Proc. of ISWC 2006. Long version in ACM Transactions on Database Systems, 34(3), 2009.

  Renzo Angles, Claudio Gutierrez. *The Expressive Power of SPARQL.* Proc. of ISWC 2008.

  Available from http://www.dcc.uchile.cl/~cgutierr/papers/

- We will use the presentation from the tutorial
  - **SPARQL - Where are we? Current state, theory and practice.** Tutorial given at ESWC 2007, Innsbruck, Austria, June 2007.
    Unit-2: **SPARQL Formalization**. Available from http://axel.deri.ie/%7Eaxepol/sparqltutorial/ .

Evaluation of SPARQL queries

- We can evaluate SPARQL queries by **translating** them into the algebraic language of Perez et al. that we have just presented.

Example Query

```
PREFIX ex:  <http://example.org/>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX ns: <http://example.org/ns#>

SELECT ?book ?title ?price
WHERE  { ?book ns:price ?price .
         FILTER (?price < 30)
         OPTIONAL { ?book dc:title ?title .}
         { ?book dc:creator ex:Smith . } UNION
         { ?book dc:creator ex:Jones . }
       }
```

Translation into Algebra

```
( ( ( {(?book, ns:price, ?price)}
           FILTER (?price < 30)
    )
    OPT
    {(?book, dc:title, ?title)}
  )
  AND
  ( {(?book, dc:creator, ex:Smith)}
    UNION
    {(?book, dc:creator, ex:Jones)}
   )
)
```

General Method of Translation

- **Identify the BGPs.** These are the atomic operands (the leafs of the corresponding parse tree of the algebra expression).

- Proceeding from the innermost to the outermost patterns, **use the precedence and associativity of operators** to obtain the algebra expression.

Readings

- Chapter 7 of the book “Foundations of Semantic Web Technologies”.
- The W3C Candidate Recommendation “SPARQL Query Language for RDF” from http://www.w3.org/TR/rdf-sparql-query/ .
- The SPARQL tutorial given at ESWC 2007 available from http://axel.deri.ie/%7Eaxepol/sparqltutorial/ especially Unit 2 (SPARQL formalization).
- The two papers on the semantics of SPARQL cited earlier.