COMP3220: Document Processing and Semantic Technologies SPARQL

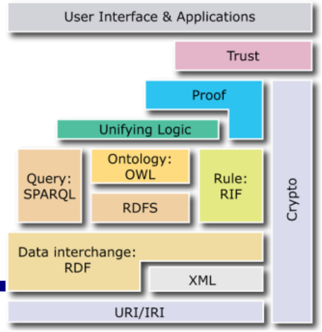
Rolf Schwitter
Rolf.Schwitter@mq.edu.au

Today’s Agenda

- What is SPARQL?
- Structure of a SPARQL Query
- Architecture and Endpoints
- SPARQL Queries
- SPARQL Queries in Python
- More on SPARQL

Examples taken from: http://www.w3.org/TR/sparql11-overview/

What is SPARQL?



- SPARQL is the standardised query language for RDF:
  http://www.w3.org/TR/sparql11-overview/
- SPARQL stands for Simple Protocol And RDF Query Language.
- SPARQL allows us to:
  - pull values from RDF data
  - explore RDF data by querying unknown relationships
  - perform complex joins of disparate RDF databases
  - transform RDF data from one vocabulary to another.

Structure of a SPARQL Query

```
# prefix declaration: for abbreviating URIs
PREFIX foo: <http://example.com/resources/ … >
# dataset definition: which RDF graph(s) are being queried
FROM …
# result clause: what information to return from the query
SELECT …
# query pattern: what to query for in the underlying dataset
WHERE { ... }
# query modifiers: slicing, ordering, rearranging query results
ORDER BY …
```

SPARQL Architecture and Endpoints

- SPARQL queries are executed against RDF datasets:
  - stored natively as RDF or
  - viewed as RDF via middleware (RDB2RDF mapping software).
- A SPARQL endpoint accepts queries and returns results via HTTP.
- The result can be returned in a variety of formats:
  - XML: for returning tables of results
  - JSON: useful for web applications
  - CSV or TSV: for importing into spreadsheets.

Data in RDF (Turtle)

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

:book1 dc:title "SPARQL Tutorial" .
```

SPARQL Query

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

Same SPARQL Query

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX :   <http://example.org/book/>

SELECT ?title
WHERE
{
  :book1 dc:title ?title .
}
```

Result

title
"SPARQL Tutorial"

NOTE on Prefix Directive

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

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

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

SPARQL in Python

```
# pip install rdflib
import rdflib

graph = rdflib.Graph()
graph.parse("example.rdf", format="turtle")
res = graph.query(
    """ PREFIX dc: <http://purl.org/dc/elements/1.1/>
        PREFIX :   <http://example.org/book/>

        SELECT ?title
        WHERE { :book1 dc:title ?title . }
    """)

for row in res:
    print("Book title: %s" % row)

# Book title: SPARQL Tutorial
```

SPARQL in Python: XML Serialisation

```
print(res.serialize(format = "xml"))
```

```
<?xml version="1.0" encoding="utf-8"?>
<sparql:sparql
   xmlns:sparql="http://www.w3.org/2005/sparql-results#"
   xmlns:xml="http://www.w3.org/XML/1998/namespace">
  <sparql:head>
    <sparql:variable name="title"> </sparql:variable>
  </sparql:head>
  <sparql:results>
    <sparql:result>
      <sparql:binding name="title">
        <sparql:literal>SPARQL Tutorial</sparql:literal>
      </sparql:binding>
    </sparql:result>
  </sparql:results>
</sparql:sparql>
```

SPARQL in Python: JSON Serialization

```
print(res.serialize(format = "json"))

{"results":
  {"bindings":
     [{"title":
        {"type": "literal",
         "value": "SPARQL Tutorial"}}]},
 "head": {"vars": ["title"]}}
```

SPARQL in Python: DBpedia Endpoint

```
# pip install SPARQLWrapper

from SPARQLWrapper import SPARQLWrapper, JSON

sparql = SPARQLWrapper("http://dbpedia.org/sparql")
sparql.setQuery("""
    PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    PREFIX dbo: <http://dbpedia.org/ontology/>
    SELECT ?name
    WHERE {
      <http://dbpedia.org/resource/Nicole_Kidman> dbo:spouse ?name .
    }
""")
```

SPARQL in Python: DBpedia Endpoint

```
sparql.setReturnFormat(JSON)
results = sparql.query().convert()

for result in results["results"]["bindings"]:
    print(result["name"]["value"])

# http://dbpedia.org/resource/Tom_Cruise
# http://dbpedia.org/resource/Keith_Urban
```

Constraints

- SPARQL FILTERs restrict solutions to those for which the filter expression evaluates to TRUE.
- FILTER functions like `regex` can test RDF literals against regular expressions.
- In SPARQL, `regex` matches only string literals.
- However, `regex` can be used to match the lexical forms of other literals by using the `str` function.

Data in RDF (Turtle)

```
@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 .
```

SPARQL 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"

Optional Pattern Matching

- In basic graph patterns, the entire query pattern must match in order to count as a solution.
- However, not all RDF graphs are complete.
- Sometimes not the entire query pattern does match.
- Optional matching provides a solution to this problem.
- If the optional part does not match, then no bindings are created, but the solution for that part is not eliminated.

Data in RDF (Turtle)

```
@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" .
```

SPARQL 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"	

Matching Alternatives

- SPARQL uses the keyword UNION for combining graph patterns so that one of several alternative patterns may match.
- If more than one of the alternatives matches, then all the possible pattern solutions are returned.
- UNION is useful for concatenating the solutions from two possibilities.

Data in RDF (Turtle)

```
@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)" .
```

SPARQL 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"

SPARQL Query

```
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"	

Negation

- SPARQL supports two styles of negation:
  - via testing of the absence of a pattern
  - via removing solutions related to a second pattern.

Test for the Absence of a Pattern

- Testing the absence of a pattern is done with a filtering expression.
- The NOT EXISTS filter expression tests whether a graph pattern does not match the dataset.
- It does not generate any additional bindings.
- There is also an EXISTS filter available that tests for the presence of a pattern.

Data in RDF (Turtle)

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

:alice  rdf:type   foaf:Person .
:alice  foaf:name  "Alice" .
:bob    rdf:type   foaf:Person .
```

SPARQL Query

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

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

Result:

person
<http://example/bob>

Removing Possible Solutions

- Removing possible solutions is done with the MINUS keyword.
- In this case, two arguments are evaluated: one on the left-hand side of the MINUS keyword and one on the right-hand side of the MINUS.
- The result consists of solutions on the left-hand side that are not compatible with solutions on the right-hand side.

Data in RDF (Turtle)

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

:alice  foaf:givenName "Alice" ;
        foaf:familyName "Smith" .

:bob    foaf:givenName "Bob" ;
        foaf:familyName "Jones" .

:carol  foaf:givenName "Carol" ;
        foaf:familyName "Smith" .
```

SPARQL Query

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

SELECT DISTINCT ?s
WHERE {
   ?s ?p ?o .
   MINUS { ?s foaf:givenName "Bob" . }
}
```

Result:

s
<http://example/carol>
<http://example/alice>

NOT EXIST versus MINUS

- NOT EXIST and MINUS can produce in some cases different solutions.

```
@prefix : http://example/> .            # Turtle Data
 :a :b :c .
```

```
 SELECT *                               # SPARQL Query
 {
   ?s ?p ?o
   FILTER NOT EXISTS { ?x ?y ?z } # eliminates any solutions
 }
```

Evaluates to a result with no solutions.

NOT EXIST versus MINUS

- NOT EXIST and MINUS can produce in some cases different solutions.

```
@prefix : http://example/> .         # Turtle Data
:a :b :c .
```

```
SELECT *                             # SPARQL Query
{
  ?s ?p ?o             # no shared variables between (?s ?p ?o)
  MINUS                # and (?x ?y ?z) so no bindings are
  { ?x ?y ?z }         # eliminated
}
```

Evaluates to a result with all solutions.

SPARQL Updates

```
# Data before the update
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix ns: <http://example.org/ns#> .

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


# Data after the update
@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" .
```

SPARQL Updates

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

SPARQL: Federated Queries (Data)

```
# Data at remote endpoint: http://people.example.org/sparql

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

:people15  foaf:name     "Alice" .
:people16  foaf:name     "Bob" .
:people17  foaf:name     "Charles" .
:people17  foaf:interest <http://www.w3.org/2001/sw/rdb2rdf/> .
```

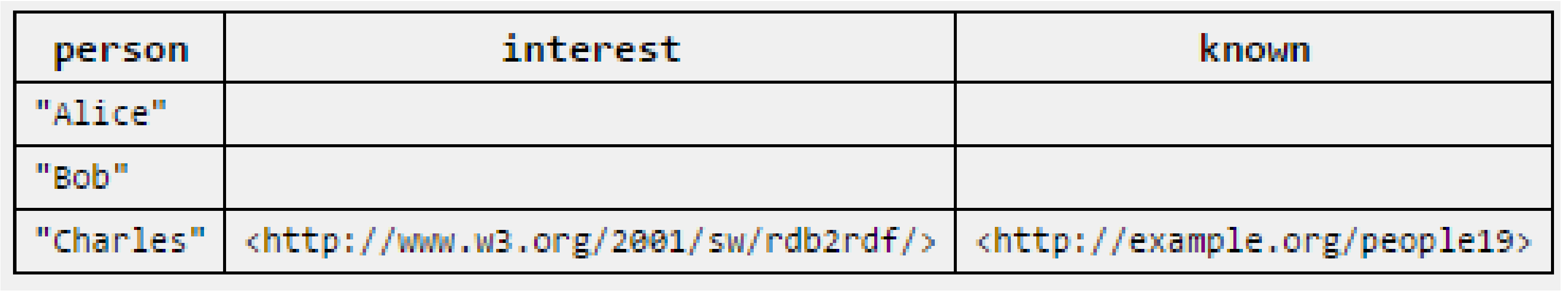
SPARQL: Federated Queries (Data)

```
# Data at remote endpoint: http://people2.example.org/sparql

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

:people15  foaf:knows    :people18 .
:people18  foaf:name     "Mike" .
:people17  foaf:knows    :people19 .
:people19  foaf:name     "Daisy" .
```

SPARQL: Federated Queries

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?person ?interest ?known
WHERE
{
  SERVICE <http://people.example.org/sparql> {
    ?person foaf:name ?name .
    OPTIONAL {
      ?person foaf:interest ?interest .
      SERVICE <http://people2.example.org/sparql> {
        ?person foaf:knows ?known . } }
  }
}
```

SPARQL: Federated Queries (Answer)



person	interest	known
"Alice"		
"Bob"		
"Charles"	<http://www.w3.org/2001/sw/rdb2rdf/>	<http://example.org/people19>

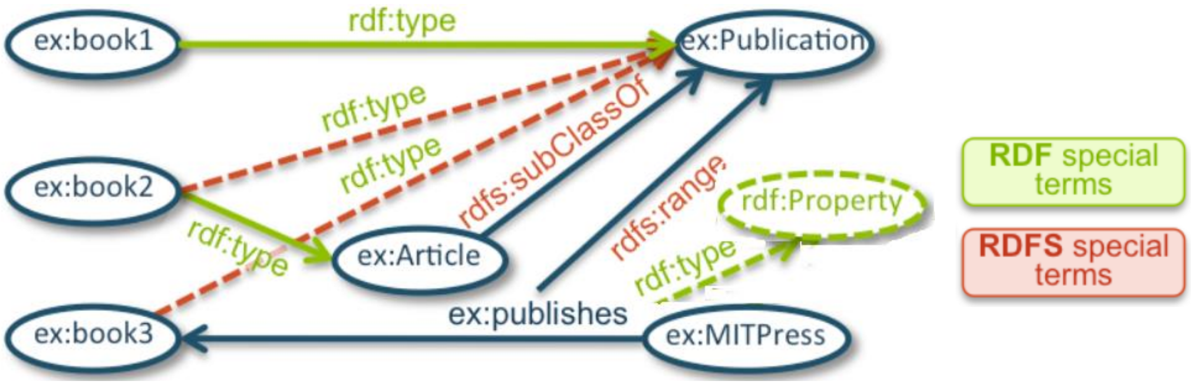
Entailment Regimes

- SPARQL defines the evaluation of a basic pattern by means of subgraph matching.
- This form of pattern evaluation is also called simple entailment.
- Simple entailment can be recognised by relatively simple syntactic comparisons.
- But the SPARQL specification discusses also extensions to other entailment relations, such as RDF and RDFS entailment.

Entailment Regimes

- Simple, RDF, and RDFS entailment:

(1) ex:book1 rdf:type ex:Publication .
(2) ex:book2 rdf:type ex:Article .
(3) ex:Article rdfs:subClassOf ex:Publication .
(4) ex:publishes rdfs:range ex:Publication .
(5) ex:MITPress ex:publishes ex:book3 .

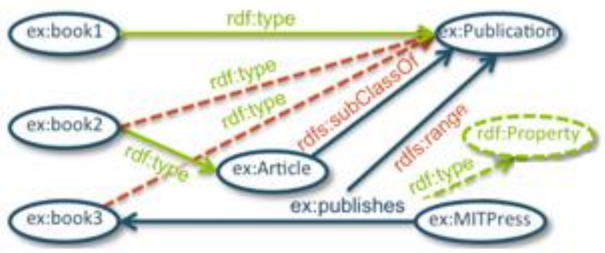


Green dashed line = RDF entailed triple

Red dashed lines = RDFS entailed triples

https://www.w3.org/TR/sparql11-entailment/

Entailment Regimes



- Given the following query:

  `SELECT ?prop WHERE { ?prop rdf:type rdf:Property }`

- Under simple entailment, the answer is empty.
- Under RDF entailment `ex:publishes` is a valid binding for `?prop`.
- Given the following query:

  `SELECT ?pub WHERE { ?pub rdf:type ex:Publication }`

- Under RDFS entailment, we can derive:

  `ex:book3 rdf:type ex:Publication .`

Take-Home Messages

- SPARQL is a query language to query and manipulate RDF graphs on the Web or in an RDF store.
- SPARQL supports different result formats.
- SPARQL can be used to update RDF graphs.
- SPARQL can execute queries over distributed endpoints.
- Extensions of the SPARQL semantics (= entailment regimes) can be defined for various semantic web languages.