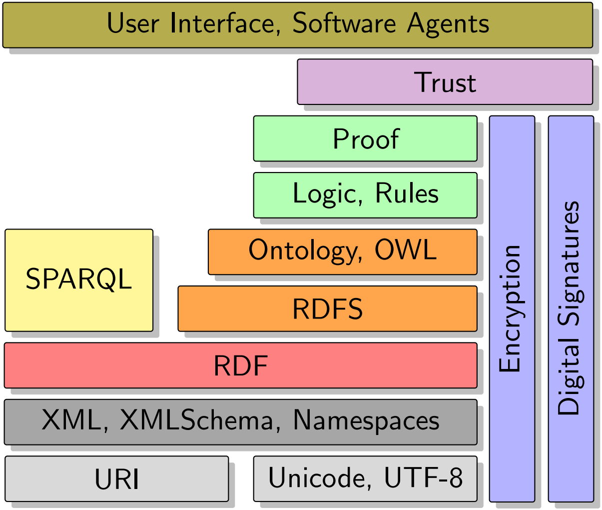
SPARQL: An RDF Query Language

Wiltrud Kessler

Institut für Maschinelle Sprachverarbeitung
Universität Stuttgart

Semantic Web
Winter 2014/15

The Semantic Web Stack [W3C, Tim Berners-Lee]



Outline

Outline

Basic SPARQL

Complex Patterns

Filter

Explicit and Inferred Information

Summary

References

Accessing Information from OWL Documents

- We have a lot of information encoded in OWL or RDF/S.
- How do we access it?
- We can use reasoning to get information from an OWL ontology:
  - What are the instances of class X?
  - Which pizza have Y as a topping?
  - Which classes are a subclass of Z?
- But sometimes we need more information than OWL and the reasoner can give us:
  - Which German labels are contained in the ontology?
  - Which property connects the individuals x and y?
  - Which two pizzas have a common topping?

SPARQL: SPARQL Protocol and RDF Query Language

- SPARQL is a recursive acronym for "SPARQL Protocol and RDF Query Language".
- SPARQL is pronounced like "sparkle".
- SPARQL is a W3C recommendation since 2008.
- Apart from the query language we are presenting here, SPARQL is also a protocol for the transmission of queries and a way of encoding the results in XML.
- SPARQL is intended for querying RDF, but OWL is based on RDF, so we can use SPARQL to query our ontologies.

SPARQL Basics (1)

- SPARQL matches graph patterns in the query with a RDF graph (remember that RDF consists of triples).
- Every triple in the graph that matches the query pattern is returned.
- If the query contains variables, the variables get "bound" to the parts of the triple that correspond to them.
- Several patterns are regarded as a conjunction (as if connected by `and`).

SPARQL Basics (2)

- Example query pattern:
  `?pizza rdfs:subClassOf pz:NamedPizza .`
- Matching RDF triple:
  `pz:PolloAdAstra rdfs:subClassOf pz:NamedPizza .`
- `?pizza` gets bound to `pz:PolloAdAstra`.

SPARQL Basic Syntax – Very Simple Example

Query:

```
PREFIX pz: <http://www.co-ode.org/ontologies/pizza/pizza.owl#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?pizza
WHERE {
   ?pizza rdfs:subClassOf pz:NamedPizza .
}
```

Result:

?pizza
pz:Margherita
pz:Napoletana
pz:PolloAdAstra
...

SPARQL Basic Syntax – Example 2

Query:

```
PREFIX pz: <http://www.co-ode.org/ontologies/pizza/pizza.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?pizzaInstance ?pizzaClass ?toppingInstance
WHERE {
   ?pizzaClass rdfs:subClassOf pz:NamedPizza .
   ?pizzaInstance rdf:type ?pizzaClass .
   ?pizzaInstance pz:hasTopping ?toppingInstance .
}
```

Result:

?pizzaInstance	?pizzaClass	?toppingInstance
pz:Margherita_Inst1	pz:Margherita	pz:SlicedTomatoTopping_Inst1
pz:Margherita_Inst1	pz:Margherita	pz:MozarellaTopping_Inst1
pz:Margherita_Inst2	pz:Margherita	pz:SundriedTomatoTopping_Inst1
pz:Margherita_Inst2	pz:Margherita	pz:MozarellaTopping_Inst2

SPARQL Syntax – Prefix

```
PREFIX pz: <http://www.co-ode.org/ontologies/pizza/pizza.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
```

- `PREFIX` declares a namespace and the abbreviating prefix that can be used to access it.

SPARQL Syntax – Result Clause

```
SELECT ?pizzaInstance ?pizzaClass ?toppingInstance
```

- The result clause identifies what information to return from the query.
- Variables are denoted by `?`.
- Only the information bound to variables in the result clause is returned.
- The number of lines returned is depending on the number of triples found, each line gives one possible way the variables can be bound.

SPARQL Syntax – Query Patterns

```
WHERE {
   ?pizzaClass rdfs:subClassOf pz:NamedPizza .
   ?pizzaInstance rdf:type ?pizzaClass .
   ?pizzaInstance pz:hasTopping ?toppingInstance .
}
```

- The query pattern is the main part of the SPARQL query.
- Query patterns are written in Turtle-Syntax for RDF.
- Query patterns are graph patterns for statements with subject, predicate and object.
- Patterns may contain variables replacing any part of the statement.
- Every pattern is terminated by a dot.
- A sequence of patterns is interpreted as a conjunction.

Querying Restrictions

- We don't only want to list instances, we want to know what pizzas are made of, i.e., query the restrictions on classes that define the possible toppings for a specific class of pizza.
- This is simple enough, we just have to know how it's defined:

```
<owl:Class rdf:about="#Margherita">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasTopping"/>
      <owl:someValuesFrom rdf:resource="#TomatoTopping"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

```
pz:Margherita rdfs:subClassOf _:node1 .
_:node1 rdf:type owl:Restriction .
_:node1 owl:onProperty pz:hasTopping .
_:node1 owl:someValuesFrom pz:TomatoTopping
```

Querying Restrictions – Example Query

Query:

```
PREFIX pz: <http://www.co-ode.org/ontologies/pizza/pizza.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX owl: <http://www.w3.org/2002/07/owl#>

SELECT ?pizza ?topping
WHERE {
   ?pizza rdfs:subClassOf pz:NamedPizza .
   ?pizza rdfs:subClassOf ?restriction .
   ?restriction owl:onProperty pz:hasTopping .
   ?restriction owl:someValuesFrom ?topping .
}
```

Querying Restrictions – Example Result

?pizza	?topping
pz:American	pz:PeperoniSausageTopping
pz:American	pz:TomatoTopping
pz:American	pz:MozzarellaTopping
pz:Margherita	pz:TomatoTopping
pz:Margherita	pz:MozzarellaTopping
pz:Rosa	pz:GorgonzolaTopping
pz:Rosa	pz:TomatoTopping
pz:Rosa	pz:MozzarellaTopping
pz:Napoletana	pz:TomatoTopping
pz:Napoletana	pz:MozzarellaTopping
pz:Napoletana	pz:CaperTopping
...	...

SPARQL and Data Types

```
ex:bsp1 ex:p "test" .
ex:bsp2 ex:p "test"^^xsd:string .
ex:bsp3 ex:p "test"@de .
ex:bsp4 ex:p "42"^^xsd:integer .
```

- Remember: Literals are typed in RDF.
- SPARQL queries are sensitive to data types, exact matches are necessary.
- The query `?subject ex:p "test".` will give `ex:bsp1` as the only result.
- To get `ex:bsp3`, the query `?subject ex:p "test"@de .` must be used.
- For numbers a short form `?subject ex:p 42 .` is possible, the number is interpreted as `xsd:integer`, `xsd:decimal`, or `xsd:double` according to the surface form.

Quiz: Basic SPARQL Queries

Which query pattern retrieves all vegetable toppings that have medium spiciness?

```
SELECT ?topping
WHERE {
   ...
}
```

A
```
?topping rdfs:subClassOf pz:VegetableTopping .
?topping rdfs:subClassOf ?restriction .
?restriction owl:onProperty pz:hasSpiciness .
?restriction owl:someValuesFrom pz:Medium .
```

B
```
?topping rdfs:subClassOf pz:VegetableTopping .
?topping rdfs:subClassOf ?restriction .
?restriction owl:onProperty pz:hasSpiciness .
?restriction owl:someValuesFrom ?spiciness .
?spiciness rdf:type pz:Medium .
```

C
```
?topping rdfs:subClassOf pz:VegetableTopping .
?topping pz:hasSpiciness pz:Medium .
```

Outline

OPTIONAL

- The first query listed only pizza instances that had an object property `pz:hasTopping`.
- To list all pizza instances and their toppings (if they have one), we can use `OPTIONAL`.
- Pizzas that don't have a topping are returned, but the variable `?topping` is unbound.
- Several optional parts can be included in one query, they are evaluated separately.
- Several statement can occur after `OPTIONAL`, they are read as a conjunction.

OPTIONAL Example (1)

Query

```
SELECT ?pizza ?topping ?base
WHERE {
   ?pizzaClass rdfs:subClassOf pz:NamedPizza .
   ?pizza rdf:type ?pizzaClass .
   OPTIONAL { ?pizza pz:hasTopping ?topping .}
   OPTIONAL { ?pizza pz:hasBase ?base .}
}
```

Result:

?pizza	?topping	?base
pz:Margherita_Inst1	pz:SlicedTomatoTopping_Inst1	
pz:Margherita_Inst1	pz:MozarellaTopping_Inst1	
pz:Margherita_Inst2	pz:SundriedTomatoTopping_Inst1	pz:ThinAndCrispyBase_Inst1
pz:Margherita_Inst2	pz:MozarellaTopping_Inst2	pz:ThinAndCrispyBase_Inst1
pz:Giardiniera_Inst1		
pz:Parmense_Inst1		pz:DeepPanBase_Inst1

OPTIONAL Example (2)

Query

```
SELECT ?pizza ?topping ?base
WHERE {
   ?pizzaClass rdfs:subClassOf pz:NamedPizza .
   ?pizza rdf:type ?pizzaClass .
   OPTIONAL {
      ?pizza pz:hasTopping ?topping .
      ?pizza pz:hasBase ?base . }
}
```

Result:

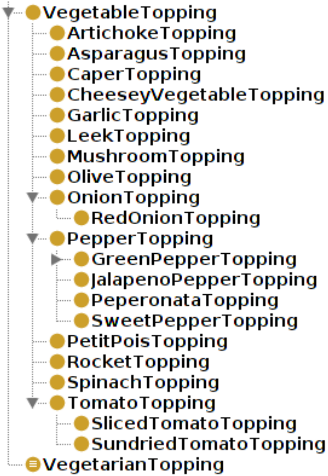
?pizza	?topping	?base
pz:Margherita_Inst1		
pz:Margherita_Inst2	pz:SundriedTomatoTopping_Inst1	pz:ThinAndCrispyBase_Inst1
pz:Margherita_Inst2	pz:MozarellaTopping_Inst2	pz:ThinAndCrispyBase_Inst1
pz:Giardiniera_Inst1		
pz:Parmense_Inst1		

UNION

- A sequence of patterns in a SPARQL query is always interpreted as a conjunction (logical `and`).
- To use the logical `or` for parts of the pattern, pattern alternatives must be used.
- Pattern alternatives are specified with the `UNION` keyword.

UNION Example Task

- Example: Return all instances that are instances of a direct subclass of `VegetableTopping` or instances of a subclass of a subclass of `VegetableTopping`.



UNION Example Query

```
SELECT ?topping ?spiciness
WHERE {
   {
      ?topping rdfs:subClassOf pz:VegetableTopping .
   }
   UNION
   {
      ?toppingClass rdfs:subClassOf pz:VegetableTopping .
      ?topping rdfs:subClassOf ?toppingClass .
   }
   ?topping rdfs:subClassOf ?restriction .
   ?restriction owl:onProperty pz:hasSpiciness .
   ?restriction owl:someValuesFrom ?spiciness .
}
```

UNION Example Result

?topping	?spiciness
pz:TomatoTopping	pz:Mild
pz:MushroomTopping	pz:Mild
pz:CaperTopping	pz:Mild
pz:LeekTopping	pz:Mild
pz:ArtichokeTopping	pz:Mild
pz:PetitPoisTopping	pz:Mild
pz:GarlicTopping	pz:Medium
pz:RocketTopping	pz:Medium
pz:OliveTopping	pz:Mild
pz:OnionTopping	pz:Medium
pz:SpinachTopping	pz:Mild
pz:AsparagusTopping	pz:Mild
pz:SundriedTomatoTopping	pz:Mild
pz:SlicedTomatoTopping	pz:Mild
pz:SweetPepperTopping	pz:Mild
pz:JalapenoPepperTopping	pz:Hot
pz:PeperonataTopping	pz:Medium

Combining OPTIONAL and UNION

- `OPTIONAL` and `UNION` are left-associative:

  ```
  pattern OPTIONAL { pattern } OPTIONAL { pattern }
  ```

  is equivalent to

  ```
  { pattern OPTIONAL { pattern } } OPTIONAL { pattern }
  ```

- Both operators have the same precedence:

  ```
  {s1 p1 o1} OPTIONAL {s2 p2 o2} UNION {s3 p3 o3}
  OPTIONAL {s4 p4 o4} OPTIONAL {s5 p5 o5}
  ```

  is equivalent to

  ```
  { { { {s1 p1 o1} OPTIONAL {s2 p2 o2} } UNION {s3 p3 o3}
    } OPTIONAL {s4 p4 o4} } OPTIONAL {s5 p5 o5}
  ```

- Additional `{}` may always be used to group patterns.

Quiz: OPTIONAL and UNION

```
SELECT ?pizza ?topping ?spiciness
WHERE {
   ?pizzaClass rdfs:subClassOf
                    pz:NamedPizza .
   ?pizza rdf:type ?pizzaClass .
   ?pizza pz:hasTopping ?topping .
   ...
}
```

Complete the above SPARQL query to retrieve all toppings of pizza instances and their spiciness, if available.

A

```
OPTIONAL {
   ?topping rdfs:subClassOf ?restriction .
   ?restriction owl:onProperty pz:hasSpiciness .
   ?restriction owl:someValuesFrom ?spiciness .
}
```

B

```
OPTIONAL {
   ?topping rdf:type ?toppingClass .
   ?toppingClass rdfs:subClassOf ?restriction .
   ?restriction owl:onProperty pz:hasSpiciness .
   ?restriction owl:someValuesFrom ?spiciness .
}
```

C

```
?topping rdf:type ?toppingClass .
OPTIONAL {
   ?toppingClass rdfs:subClassOf ?restriction .
   ?restriction owl:onProperty pz:hasSpiciness .
   ?restriction owl:someValuesFrom ?spiciness .
}
```

Outline

FILTER

- Sometimes it is useful to restrict the values of a variable.
- Example: List all pizzas with a price between 6 and 9 euro.
- `FILTER` can be used to filter out lines in the result for which the filter expression evaluates to `false`.
- A filter is always applied to the whole pattern group in which the filter appears.
- Filters can be connected with boolean operators: `&&`, `||`, `!`

FILTER Syntax

- For numerical types, different operators for comparison are defined: `<, =, >, <=, >=, !=`
- For other types, only = and `!=` are defined.
- Literals of incompatible types cannot be compared.
- Arithmetic operators can be used in filters:
  `FILTER (?weight / (?height * ?height) >= 25 )`
- There are several other predefined filters (see literature), e.g.:

`BOUND(A)`	true if A is a bound variable
`isURI(A)`	true if A is a URI
`isLITERAL(A)`	true if A is a literal
`DATATYPE(A)`	returns the URI of the datatype of A
`REGEX(A,B)`	true if regular expression B can be found in A

FILTER Example (1)

```
SELECT ?pizza ?price
WHERE {
   ?pizzaClass rdfs:subClassOf pz:NamedPizza .
   ?pizza rdf:type ?pizzaClass .
   ?pizza pz:hasPrice ?price.
   FILTER (?price < 9)
   FILTER (?price > 6)
}
```

Result:

?pizza	?price
pz:Parmense_Inst1	7.80

FILTER Example (2)

```
SELECT ?pizza ?price
WHERE {
   ?pizzaClass rdfs:subClassOf pz:NamedPizza .
   ?pizza rdf:type ?pizzaClass .
   ?pizza pz:hasPrice ?price.
   FILTER (?price > 9 || !((?price + 5) >= 11))
}
```

Result:

?pizza	?price
pz:Margherita_Inst1	5.60
pz:Margherita_Inst2	5.60
pz:Giardiniera_Inst1	9.95

Quiz: FILTER

Determine at which point the line `FILTER (?price <9)` has to be inserted in the SPARQL query, so that only pizza instances (and their toppings if available) with a price less than 9 € are retrieved.

```
1  SELECT ?pizza ?price ?topping
2  WHERE {
3     ?pizzaClass rdfs:subClassOf pz:NamedPizza .
4     ?pizza rdf:type ?pizzaClass .
5     OPTIONAL {
6        ?pizza pz:hasTopping ?topping .
7     }
8     OPTIONAL {
9        ?pizza pz:hasPrice ?price.
10    }
11 }
```

1. Between lines 9 and 10.
2. Between lines 6 and 7.
3. Between lines 3 and 4.

More . . .

- Modifiers to influence the presentation:

  ORDER BY establishes the ordering of the resulting lines by some variable, ascending or descending.

  LIMIT limits the number of solutions returned.

  OFFSET causes the solutions generated to start after the specified number of solutions (only predictable in combination with ORDER BY).

  DISTINCT eliminates duplicate solutions (a solution that binds the same variables to the same RDF terms as another solution).

- Different query forms, besides SELECT covered here, there is ASK, DESCRIBE and CONSTRUCT.

Outline

Explicit and Inferred Information (1)

```
SELECT ?pizza ?price
WHERE {
   ?pizza rdf:type pz:NamedPizza .
   ?pizza pz:hasPrice ?price .
}
```

- In the ontology we have several subclasses of `NamedPizza` (e.g. `Caprina` or `Parmense`) and some instances of these.
- What is the result of this query?
- We would expect it to be all instances, because if they are an instance of a subclass of `NamedPizza`, they are also an instance of `NamedPizza`.
- Actually the result of this query is empty.
- Why? Because this information is not explicitly contained in the graph, but needs to be inferred.

Explicit and Inferred Information (2)

```
SELECT ?pizza ?price
WHERE {
   ?pizzaClass rdfs:subClassOf pz:NamedPizza .
   ?pizza rdf:type ?pizzaClass .
   ?pizza pz:hasPrice ?price.
}
```

- The above query gets all instances of subclasses of `NamedPizza`.
- So this results in what we want:

?pizza	?price
pz:Margherita_Inst1	5.60
pz:Margherita_Inst2	5.60
pz:Giardiniera_Inst1	9.95
pz:Parmense_Inst1	7.80

- But what about subclasses of subclasses?

Explicit and Inferred Information (3)

- We cannot (or don't want to) enumerate all posible levels of subclasses of subclasses of ...
- We want to query inferred information!
- But SPARQL is only a query language, there is no inference in the query language itself.
- Fortunately, some implementations (e.g., Jena) have a reasoner included.
- Otherwise, it is possible to let a reasoner work on the ontology and then export the ontology including inferred statements (in Protégé this should work with "Menu – Export inferred axioms as ontology").

Exercise: SPARQL

- Write some SPARQL queries that your application might use.
- Discuss your solution with your neighbour.

- Example: List the title and year of writing (if available) of all songs written by Michael Jackson.

```
SELECT ?title ?year
WHERE {
   ?song rdf:type ex:Song .
   ?song ex:writtenBy ex:MichaelJackson .
   ?song ex:hasTitle ?title .
   OPTIONAL {
      ?song ex:writtenIn ?date .
      ?date ex:hasYear ?year .
   }
}
```

Outline

Summary:

- SPARQL matches **graph patterns** in the query with an RDF graph, if the query contains **variables**, the variables get "bound" to the parts of the triple that correspond to them.
- Several patterns are regarded as a **conjunction**, to express a **disjunction** ("or"), `UNION` can be used.
- `OPTIONAL` can be used to include the information **if available**, otherwise the result is returned with the variable unbound.
- `FILTER` can be used to **filter** out lines in the result for which the filter expression evaluates to `false`.
- We can only query **explicit information**, to access inferred information, a reasoner must be run on the ontology first.

Outline

Suggested Reading

[HKRS08] Pascal Hitzler, Markus Krötzsch, Sebastian Rudolph and York Sure. Semantic Web. Grundlagen. Springer textbook, 2008. (Chapter 7)

[HKR09] Pascal Hitzler, Markus Krötzsch and Sebastian Rudolph. Foundations of Semantic Web Technologies. Chapman & Hall/CRC, 2009. (Chapter 7)