Semantic Web

Lecture 11

Dr. Knarig Arabshian
Knarig.arabshian@hofstra.edu

Reference

All slides are taken from the following text:

Hitzler, Krotzch, Rudolph, Foundations of Semantic Web Technologies

Query Language: SPARQL

- SPARQL Protocol and RDF Query Language
- Query language for RDF-based information
- Core of SPARQL are simple queries in the form of simple graph patterns
- Provides advanced functions for constructing advanced query patterns
- Only covering the query language, not the protocol or result format
- Similar to SQL but fundamentally different

SPARQL

- Simple RDF graphs are used as fundamental query pattern
- Graph is represented using Turtle syntax
- Uses query variables to specify parts of a query pattern that should be retrieved
- Each query specifies how results should be formatted

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
WHERE { ?book     ex:publishedBy     <http://crc-press.com/uri> .
        ?book     ex:title           ?title .
        ?book     ex:author          ?author }
```

- Consists of three major parts
  - PREFIX: declares namespace prefix, similar to Turtle notation
  - SELECT: determines the general result format
    - similar as in SQL.
    - listed names are identifiers of variables that need to be retrieved
    - return all values for the variables ?title and ?author
  - WHERE: actual query clause

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
WHERE { ?book      ex:publishedBy     <http://crc-press.com/uri> .
            ?book       ex:title             ?title .
            ?book       ex:author            ?author }
```

- WHERE:
  - Simple graph patter in Turtle notation
  - Difference from Turtle is that triples may contain variables in addition to URIs and Qnames, such as ?book
  - Identifiers represent possible concrete values obtained in the process of answering the query
  - Variables can be used in multiple places so same value must be used in those positions

SPARQL

```
PREFIX ex: <http://example.org/>
SELECT ?title ?author
WHERE { ?book      ex:publishedBy      <http://crc-press.com/uri> .
               ?book      ex:title            ?title .
               ?book      ex:author           ?author }
```

- This query retrieves all things that have been published by CRC Press where title and author are known
- Result will show pairs of title and author

SPARQL

@prefix ex: <http://example.org/> .

@prefix book: <http://semantic-web-book.org/uri/> .

ex:SemanticWeb ex:publishedBy <http://crc-press.com/uri> ;

ex:title "Foundations of SWT" ;
ex:author book:Hitzler, book:Krotzsche, book:Rudolph .

- Result of the query on this RDF document is a table
- Values are only for those variables that have been mentioned explicitly in the SELECT line
- ?book although part of the query has not been selected fro the result

title	author
"Foundations of SWT"	http://semantic-web-book.org/uri/Hitzler
"Foundations of SWT"	http://semantic-web-book.org/uri/Krotzsche
"Foundations of SWT"	http://semantic-web-book.org/uri/Rudolph

SPARQL: Simple Graph Patterns

- Simple graph patterns can represent arbitrary RDF graphs that should be searched in the given data set
- SPARQL has a weak assumption of semantics in the knowledge base
- It takes only simple RDF inferences into account
- RDFS or OWL are not directly supported
- Mainly uses Turtle syntax and abbreviations for creating graph patters

SPARQL: Simple Graph Patterns

- Variables are distinguished by the initial symbol ? Or $ followed by sequence of numbers, letters and some special symbols like underscore
- ?author and $author refer to the same variable
- Choice of variable names has no impact on meaning of query but meaningful names are helpful to understand the query
- Variables may appear as the subject and object in a triple as well as in the predicate
- Example:
  - Retrieve all known relations between the given URIs

  BASE <http://semantic-web-book.org/>

  SELECT ?relation

  WHERE {<uri> ?relation <http://crc-press.com/uri> }

  - Result: http://example.org/publishedBy

SPARQL: Simple Graph Patterns

- Variables are distinguished by the initial symbol ? Or $ followed by sequence of numbers, letters and some special symbols like underscore
- ?author and $author refer to the same variable
- Choice of variable names has no impact on meaning of query but meaningful names are helpful to understand the query
- Variables may appear as the subject and object in a triple as well as in the predicate
- Example:
  - Retrieve all known relations between the given URIs

  BASE <http://semantic-web-book.org/>

  SELECT ?relation

  WHERE {<uri> ?relation <http://crc-press.com/uri> }

  - Result: http://example.org/publishedBy

SPARQL: Complex Graph Patterns

- Groups
  - Group patterns can be used to restrict scope of query conditions to certain parts of the pattern
  - Possible to define sub-patterns as optional
  - Provide multiple alternative patterns
- Separate multiple simple patterns from each other using curly braces
- Example:

```
SELECT ?title ?author
WHERE { {   ?book ex:publishedBy <uri> .
             ?book  ex:title ?title }
             { }
             ?book ex:author ?author
      }
```

OPTIONAL Pattern

- Optional pattern indicated by key word OPTIONAL
- Not required to occur in all results
- If found, may produce bindings of variables
- Example:

```
{ ?book  ex:publishedBy    <uri> .
  ?book  ex:title          ?title .
   OPTIONAL { ?book  ex:author      ?author }
```

- Result: matches all books of <uri> for which a title is provided. If authors of each book are given, then these are retrieved, but not all results need to have specified authors.

OPTIONAL Pattern

- OPTIONAL refers to subsequent group pattern
- Every occurrence of OPTIONAL must be followed by a group pattern
- Value is assigned to variable ?author if pattern is found
- Multiple optional patterns can also be specified

OPTIONAL Pattern

- Example:

```
{?book          ex:publishedBy          <uri> .
  OPTIONAL { ?book   ex:title          ?title      }
  OPTIONAL { ?book   ex:author         ?author }
```

RESULT:

book	title	author
uri1	title1	author1
uri2	title2	
uri3	title3	author3
uri4		author4
uri5		

Alternative Patterns: Union

- Alternative patterns can also be specified by key word UNION
- Related to logical disjunction:
  - every result must match at least one of the provided alternative patterns
  - might match more than one
- Example

```
{ ?book ex:publishedBy  <uri>      .
   { ?book        ex:author        ?author . } UNION
   { ?book        ex:writer        ?author . }
}
```

UNION

- Individual parts are processed independently of each other
- Both alternative patterns refer to the same variable ?author
- Results of the query are obtained by taking union of the results of two separate queries:

```
{ ?book ex:publishedBy <uri> .
  { ?book ex:author ?author . }
}
{ ?book ex:publishedBy <uri> .
  { ?book ex:writer ?author . }
}
```

Combination of Group Patterns

- OPTIONAL and UNION can be used more than once to combine results of multiple alternative patterns.
- Need to know how individual patterns are grouped

```
{ ?book  ex:publishedBy    <uri> .
  { ?book      ex:author      ?author . } UNION
  { ?book      ex:writer ?author . } OPTIONAL
  { ?author    ex:lastName  ?name  . }
}
```

Equivalent to:

```
{?book   ex:publishedBy    <uri> . }
{   { ?book          ex:author      ?author  . } UNION
    { ?book          ex:writer ?author  . }
} OPTIONAL { ?author       ex:lastName  ?name . }
}
```

Combination of Group Patterns

- UNION and OPTIONAL refer to two patterns: preceding and succeeding one.
- Both key words are binary operators
- OPTIONAL always refers to exactly one group graph pattern immediately to its right
- OPTIONAL and UNION are left-associative and none of the operators has precedence over the other

Combination of Group Patterns

- Example

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

Equivalent To:

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

Queries with Data Values

Example Turtle:

ex:s1 ex:p "test" .

ex:s2 ex:p "test"^^xsd:string .

ex:s3 ex:p "test"@en .

ex:s4 ex:p "42"^^xsd:integer .

ex:s5 ex:p "test"^^<http://example.org/datatype1>.

Query:

{?subject ex:p “test” . }

Result: ex:s1

Input data for ex:s1, ex:s2 and ex:s3 seem to be matching patterns but result will only be ex:s1 because RDF strictly distinguishes typed and untyped literals.

To obtain ex:s2 and ex:s3 different queries need to be formulated specifying the datatype

Queries with Data Values

- SPARQL provides syntactic abbreviations for common datatypes
- Numerical inputs are interpreted based on their syntactic form to refer to literals of type xsd:integer, xsd:decimal, xsd:double

{subject <http://example.org/p> 42 . }

- Returns ex:s4

Filters

- Query not just for exact data value but
  - for values within a range
  - search for literals within a certain word
- Filters are additional conditions in a query that restrict set of matching results

```
select ?book where
{ ?book       ex:publishedBy    <uri> .
  ?book       ex:price          ?price
   FILTER (?price < 100)
}
```

Filters

- Comparison Operators
  - =, >, >=, !=
  - Types compared are boolean, string, dates, numerical datatypes and untyped RDF literals with language settings.
  - Natural order is used for comparing literals
  - = and != can be used for all RDF elements but produce error if two lexically different values with unknown datatypes are given (impossible to determine whether both literals describe the same value)

Filters

- Special Operators for accessing RDF-specific information
- Example:

```
select ?book where
{?book          ex:publishedBy    <uri> .
 OPTIONAL {?book           ex:author      ?author .}
FILTER (DATATYPE(?author)=<http://www.w3.org/2001/XMLSchema#string>)
}
```

Filters

- Special Operators
  - BOUND(A) – true if A is bound variable
  - isURI(A) – true if A is a URI
  - isBLANK(A) – true if A is a blank node
  - isLITERAL(A) – true if A is an RDF literal
  - STR(A) – maps RDF literals or URIs to corresponding lexical representation of type xsd:string
  - LANG(A) – returns language code of an RDF literal as xsd:string
  - DATATYPE(A) – returns URI of an RDF literal's datatype or the value “xsd:string” for untyped literals without language settings

Filters

- sameTerm(A, B) – true if A and B are the same RDF terms
- langMatches(A, B) – true if the literal A is a language tag that matches the pattern B
- REGEX(A,B) – true if the regular expression B can be matched to the string A
- sameTERM and equality symbol are different
  - SameTERM performs direct term comparison on RDF level where datatypes are not taken into account
  - Allows for comparison of different literals of unknown datatypes
- langMatches
  - Language settings may have hierarchical forms
    - LANG(“Test”@en-GB) = “en” is not satisfied
    - langMATCHES(LANG(“Test@en-GB), “en”) is
    - LangMATCHES is interpreted in a general way

OWL Formal Semantics

- OWL DL can be identified with a decidable fragment of first-order predicate logic
- Draws on history of philosophical and mathematical logic
- OWL DL can be traced back to semantic networks
  - Used for modeling simple relationships between individuals and classes via roles
  - Comparable to RDFS
- Since meaning of semantic networks was vague, formalization was necessary
  - Led to development of description logics
  - Designed to achieve trade-offs between expressivity and scalability
  - Usually decidable

ALC

- ALC: Attributive Language with Complement
  - Most fundamental description logic
  - Formally defined as follows
    - The following are *concepts*:
      - $\top$ (*top* is a *concept*)
      - $\bot$ (*bottom* is a *concept*)
      - Every $A \in N_C$ (all *atomic concepts* are *concepts*)
    - If C and D are *concepts* and $R \in N_R$ then the following are *concepts*:
      - $C \sqcap D$ (the intersection of two *concepts* is a *concept*)
      - $C \sqcup D$ (the union of two *concepts* is a *concept*)
      - $\neg C$ (the complement of a *concept* is a *concept*)
      - $\forall R.C$ (the universal restriction of a *concept* by a *role* is a *concept*)
      - $\exists R.C$ (the existential restriction of a *concept* by a *role* is a *concept*)

ALC

- ALC is a subset of OWL DL
- Classes, roles, individuals
- Class membership and role instances
- owl:Thing and owl:Nothing
- Class inclusion, equivalence and disjointness
- Conjunction, disjunction and negation of classes
- Role restrictions using owl:allValuesFrom and owl:someValuesFrom
- rdfs:domain and rdfs:range

ALC

- TBox: contains terminological or schema knowledge (classes, properties)
  - TBox contains statements for class relations
  - C=D, C[D
- ABox contains assertional knowledge about instances (individuals)
  - Abox consists of statements of the form:
    - C(a), where C is a class expression,
    - R(a, b), where R is a role, and a, b are individuals
- Statements of either kind are called axioms
- Knowledge base consists of an ABox and a TBox

SHOIN(D)

- OWL DL can not be expressed fully in ALC
- Need to extend ALC to the description logic SHOIN (D) which encompasses ALC and further expressions
- Letters representing different types of expressions
  - S – ALC plus role transitivity
  - H – role hierarchies
  - O – nominals (closed classes with one element)
  - I – inverse roles
  - N – cardinality restrictions
  - D – datatypes

SHOIN(D)

- Expresses:
  - All language constructs from ALC
  - Equality and inequality between individuals
  - Closed classes (disjunctions of nominals)
  - Cardinality restrictions
  - Role inclusion axioms and rule equivalences (role hierarchies)
  - Inverse roles
  - Transitivity, symmetry, functionality and inverse functionality of roles
  - Datatypes