NSWI144 – Linked Data – Lecture 5 – 5 November 2012

SPARQL

Martin Svoboda

Faculty of Mathematics and Physics

Charles University in Prague

XRG
XML Web Engineering
research group

Outline

- **SPARQL**
  - Introduction
  - Constructs
    - Graph patterns
    - Term constraints
    - Solution modifiers
    - Query forms

SPARQL

- **SPARQL**
  - SPARQL = **Query Language for RDF**
  - W3C
  - Versions
    - **1.0** – accepted standard (2008)
      - Language, protocol and result serialization
      - http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
    - **1.1** – working draft (2012)

Introduction

- Data

  -
    ```
    @prefix is: <http://is.cuni.cz/studium/is#> .
    @prefix foaf: <http://xmlns.com/foaf/0.1/> .
    is:s1 rdf:type is:Student ;
          is:name "Thomas" ; is:age "26" .
    is:s2 rdf:type is:Student ;
          is:name "Peter" .
    is:s3 rdf:type is:Student ;
          is:name "John" ; is:age "30" .
    is:s1 foaf:knows is:s2 .
    is:s2 foaf:knows is:s3 .
    ```

Introduction

- Query

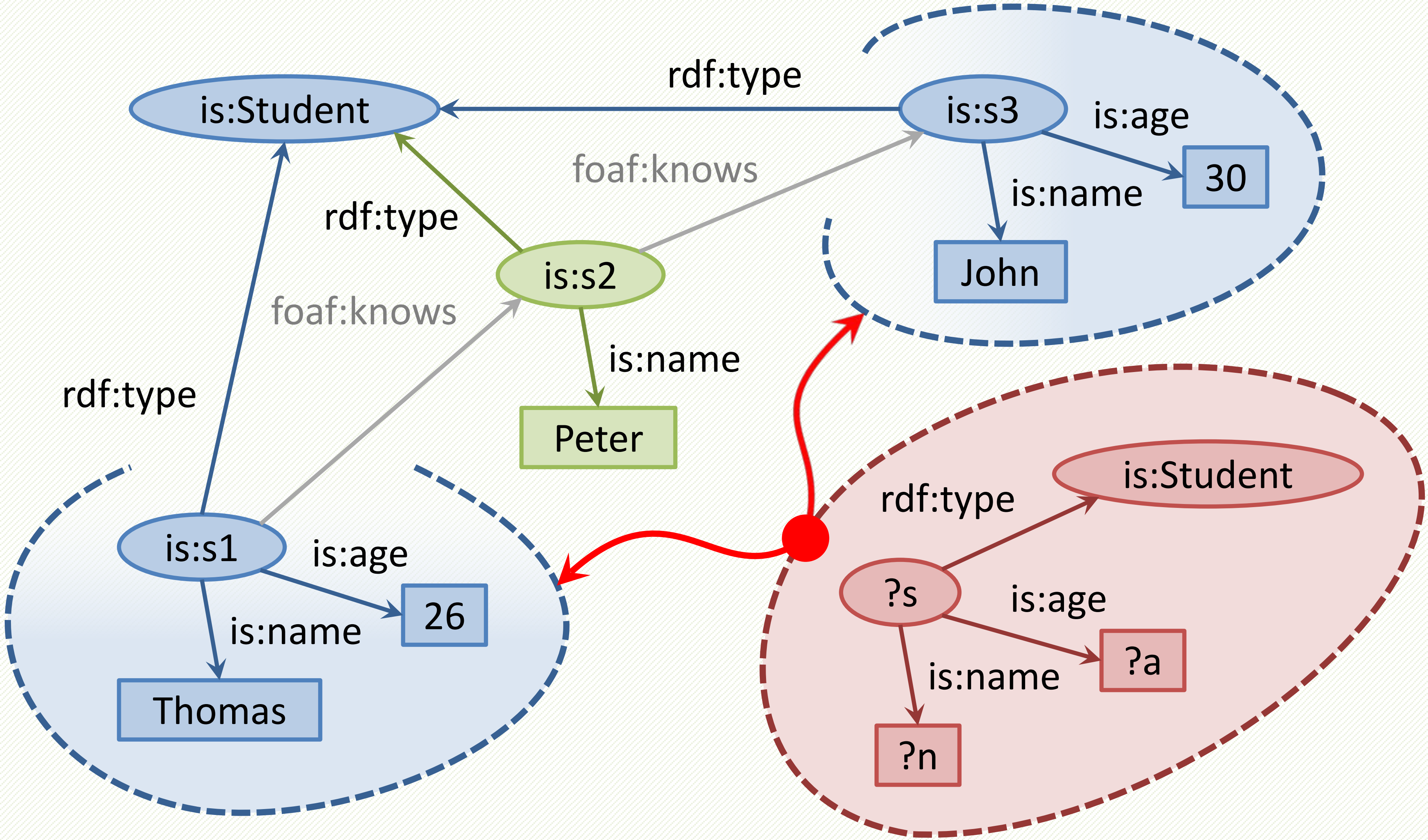
  -
    ```
    PREFIX is: <http://is.cuni.cz/studium/is#>
    SELECT ?n ?a
    WHERE {
      ?s rdf:type is:Student ;
         is:name ?n ;
         is:age ?a .
    }
    ```
- Result

?n	?a
"Thomas"	"26"
"John"	"30"

Querying Idea

- **Graph patterns**
  - Based on ordinary triples
    - Subject, predicate and object
    - URI references, blank nodes, literals and **variables**
      - `?name` or `$name`
  - We are attempting to find **subgraphs of the data graph that are matched by the query patterns**
    - This matching is based on substitution of variables
    - However, SPARQL is not just a simple graph matching!

Querying Idea



Matching

- Graph patterns
  - **Triple pattern** as a triple with variables
  - **Basic graph pattern** as a set of triple patterns
  - … and other more complex patterns
- How the matching works?
  - **Basic graph pattern matches a subgraph** of the RDF data graph **when terms** from that subgraph **may be substituted for the variables** and the result is RDF graph equivalent to the subgraph

Matching

- Equivalency of **literals**
  - **Language tags**
    - Evaluated as different literals!
      - `"Praha"`
      - `"Praha"@cs`
      - `"Prague"@en`
  - **Typed literals**
    - Shortcuts available for common typed literals…
      - `1 = "1"^^xsd:integer`
      - `1.5 = "1.5"^^xsd:decimal`
      - `true = "true"^^xsd:boolean`

Matching

- Equivalency of **blank nodes**
  - … in a data graph
    - Distinct nodes within the document scope
  - … in a query pattern
    - Blank nodes act as non-selectable variables
      - **Blank node labels in the query cannot be expected to correspond to blank nodes in the source data graph!**
  - … in a query result
    - Distinct nodes within the result scope
      - **Blank node labels in the query result may not correspond to blank nodes from the source graph and nor the query!**

Query Results

- Results
  - **Variable binding**
    - `(?n, "Thomas")`
  - **Solution** = set of variable bindings
    - Represents one possible way of variables substitution
      - Note that not all variables need to be bound!
    - Corresponds to one row of the result table
    - `{ (?n, "Thomas"), (?a, "26") }`
  - **Solution sequence** = ordered multiset of solutions

    -
      ```
      { (?n, "Thomas"), (?a, "26") },
      { (?n, "John"), (?a, "30") }
      ```

Query Structure

- Syntax

  -
    ```
    PREFIX …
    SELECT …
    FROM …
    WHERE { … }
    ORDER BY … LIMIT … OFFSET …
    ```

Prologue

- **PREFIX**
  - Definition of prefix labels for URIs
  - Example
    - `PREFIX my: <http://www.my.cz/>`
    - … then `my:x` corresponds to `<http://www.my.cz/x>`
- **BASE**
  - Usage of relative URIs
  - Example
    - `BASE <http://www.my.cz/>`
    - … then `<x>` corresponds to `<http://www.my.cz/x>`

Graph Patterns

- **WHERE**
  - **Graph patterns**
    - Triple
    - Basic
    - Group
    - Optional
    - Alternative
    - Named graphs
  - Inductive construction
    - Combining smaller patterns into more complex ones

Graph Patterns

- **Basic** graph pattern
  - **… when a set of triple patterns must all match**
  - Syntax
    - Ordinary triple patterns…
    - … and abbreviated forms inspired by Turtle
      - Object lists using `,` and predicate-object lists using `;`
      - Blank nodes using `[ ]` and collections using `( )`
  - Examples
    - `s p1 o1 . s p1 o2 . s p2 o3 .`
    - `s p1 o1 , o2 ; p2 o3 .`

Graph Patterns

- **Basic** graph pattern
  - Interpretation
    - **All involved triple patterns must match**
    - I.e. we combine them using **conjunction**
    - Note that all variables need to be bound

Graph Patterns

- **Group** graph pattern
    - **… when a set of graph patterns must all match**
  - Syntax
    - **{** *Pattern1 Pattern2* **… }**
    - Empty group patterns are also allowed
  - Interpretation
    - **All involved graph patterns must match**
    - I.e. we combine them using **conjunction**

Graph Patterns

- **Optional** graph pattern
    - **… when additional patterns may extend the solution**
  - Syntax
    - *Pattern1* **`OPTIONAL`** `{` *Pattern2* `}`
  - Interpretation
    - **If the optional part does not match, it creates no bindings but does not eliminate the solution**

Graph Patterns

- **Optional** graph pattern
  - Example

    -
      ```
      PREFIX is: <http://is.cuni.cz/studium/is#>
      SELECT ?n ?a
      WHERE {
        ?s rdf:type is:Student ; is:name ?n .
        OPTIONAL { ?s is:age ?a . }
      }
      ```

?n	?a
"Thomas"	"26"
"Peter"	
"John"	"30"

Graph Patterns

- **Optional** graph pattern
  - Left-associativity
    - { OPTIONAL { *P1* } }
{ { } OPTIONAL { *P1* } }
    - *P1* OPTIONAL { *P2* } OPTIONAL { *P3* }
{ *P1* OPTIONAL { *P2* } } OPTIONAL { *P3* }

Graph Patterns

- **Alternative** graph pattern
  - **… when two or more possible patterns are tried**
  - Syntax
    - `{` *Pattern1* `}` **`UNION`** `{` *Pattern2* `}`
  - Interpretation
    - Traditional union of sets of solutions

Graph Patterns

- **Named graphs**
  - Motivation
    - **Dataset** = collection of…
      - … one **default graph**
      - … and zero or more **named graphs**
      - Each of these graphs is indentified by a URI
    - **Active graph** = graph used for evaluation
      - We can switch the default graph to another named graph
  - Syntax
    - **`FROM`** `<http://...>`
    - **`FROM NAMED`** `<http://...>`

Graph Patterns

- **Named graphs**
  - Default graph
    - If there are more FROM definitions…
      - We use merge of all these graphs
    - If these is no FROM definition…
      - We use an empty graph
  - Usage
    - **`GRAPH`** `<http://...> { … }`
      - Sets the specified named graph as the active one
    - **`GRAPH`** `?g { … }`
      - Ranges over all named graphs defined in the dataset

Term Constraints

- **FILTER**
  - Motivation
    - Impose constraints on variables and their values
    - Cause filtering of solutions when not satisfied
  - Example
    - `FILTER (?age < 20)`
  - Usage
    - Expressions with operators and functions
    - **Filters are applied on entire group graph patterns**

Term Constraints

- **Functions**
  - Arithmetic operators
    - Unary + –
    - Binary + – * /
  - Term accessors
    - STR – lexical form of URI or literal
    - LANG – language tag of a literal
    - DATATYPE – type of a literal

Term Constraints

- **Predicates**
  - Comparison operators
    - `<  <=  >=  >`
      - Unbound variable < blank node < URI < literal
    - `=  !=`
  - Variable tests
    - `BOUND` – whether a variable is assigned a value
    - `isURI`, `isBLANK`, `isLITERAL`

Term Constraints

- **Connectives**
  - Logical connectives
    - ! && ||
- **Semantics**
  - 3 value logic
    - True, false, error

Solution Modifiers

- Query structure
  - PREFIX …

    SELECT DISTINCT | REDUCED …

    FROM …

    WHERE { … }

    ORDER BY … LIMIT … OFFSET …
- Motivation
  - Modify the entire sequence of solutions
    - Only allowed in SELECT queries

Solution Modifiers

- **DISTINCT**
  - Removes duplicates from the solution sequence
- **REDUCED**
  - Permits elimination of some non-unique solutions

Solution Modifiers

- **ORDER BY**
  - Motivation
    - Orders solutions in the solutions sequence
    - This ordering can be hierarchical
  - Behavior
    - **ASC** = ascending (default), **DESC** = descending
    - Unbound variable < blank node < URI < literal
  - Example
    - `ORDER BY ?name, DESC(?age)`

Solution Modifiers

- **LIMIT**
  - Limits the number of solutions in the result
    - (Always) should be preceded by ORDER BY modifier
    - Otherwise the order of solutions is not defined
  - Example
    - `ORDER BY ?name **LIMIT** 10`
- **OFFSET**
  - Index of the first reported item from the sequence
  - Example
    - `ORDER BY ?name LIMIT 10 **OFFSET** 20`

Query Forms

- Query structure

  -
    ```
    PREFIX …
    SELECT | DESCRIBE | ASK | CONSTRUCT …
    FROM …
    WHERE { … }
    ORDER BY … LIMIT … OFFSET …
    ```

Query Forms

- **SELECT**
    - SPARQL querying considered so far…
  - Result
    - **Solutions sequence** as an ordered multiset of solutions
  - Syntax
    - **SELECT** *variables* …
      - Variables are separated by spaces
      - Asterisk * selects all variables

Query Forms

- **ASK**
  - Checks whether at least one solution exists
  - Result
    - `true` or `false`

Query Forms

- **DESCRIBE**
  - Result
    - RDF graph with data about resources
    - **Non-deterministic** behavior
  - Examples
    - `DESCRIBE <http://www.my.cz/>`

    -
      ```
      DESCRIBE ?s
      FROM <http://is.cuni.cz/studium>
      WHERE { ?s rdf:type is:Student . }
      ```

Query Forms

- **CONSTRUCT**
  - Construction of new graphs from solutions
- Result
  - RDF graph constructed from a template
  - Illegal triples (unbound or invalid) are thrown away
- Example
  -
```
CONSTRUCT
 { ?s is:name concat(?n1, " ", ?n2) . }
FROM <http://is.cuni.cz/studium>
WHERE
  { ?s is:firstName ?n1 ; is:lastName ?n2 . }
```

Conclusion

- **SPARQL**
  - Model
    - **Matching subgraphs** and **substitution of variables**
    - Result as an **ordered multiset of solutions**
    - Solution as a **set of variable bindings**
  - Syntax

    -
      ```
      PREFIX …
      SELECT …
      FROM …
      WHERE { … }
      ORDER BY … LIMIT … OFFSET …
      ```