RDF y SPARQL: Dos componentes básicos para la Web de datos

Marcelo Arenas

PUC Chile & University of Oxford

Semantic Web

"The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation."

[Tim Berners-Lee et al. 2001.]

Specific goals:

- Build a description language with standard semantics
  - Make semantics machine-processable and understandable
- Incorporate logical infrastructure to reason about resources
- W3C proposals: Resource Description Framework (RDF) and SPARQL

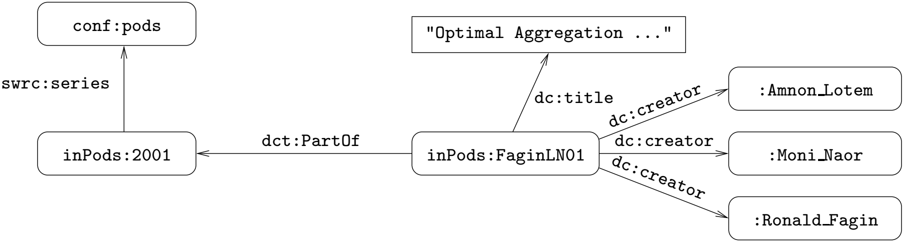
RDF in a nutshell

RDF is the framework proposed by the W3C to represent information in the Web:

- URI vocabulary
  - A URI is an atomic piece of data, and it identifies an abstract resource
- Syntax based on directed labeled graphs
  - URIs are used as node labels and edge labels
- Schema definition language (RDFS): Define new vocabulary
  - Typing, inheritance of classes and properties, ...
- Formal semantics

An example of an RDF graph: DBLP

```
      : <http://dblp.l3s.de/d2r/resource/authors/>
  conf: <http://dblp.l3s.de/d2r/resource/conferences/>
inPods: <http://dblp.l3s.de/d2r/resource/publications/conf/pods/>
  swrc: <http://swrc.ontoware.org/ontology#>
    dc: <http://purl.org/dc/elements/1.1/>
   dct: <http://purl.org/dc/terms/>
```



An example of a URI

`http://dblp.l3s.de/d2r/resource/conferences/pods`

URI can be used for any abstract resource

`http://dblp.l3s.de/d2r/page/authors/Ronald_Fagin`

Querying RDF

Why is this an interesting problem? Why is it challenging?

- RDF graphs can be interconnected
  - URIs should be dereferenceable
- Semantics of RDF is open world
  - RDF graphs are inherently incomplete
  - The possibility of adding optional information if present is an important feature
- Vocabulary with predefined semantics
- ...

Querying RDF: SPARQL

- SPARQL is the W3C recommendation query language for RDF (January 2008).
  - SPARQL is a recursive acronym that stands for *SPARQL Protocol and RDF Query Language*
- SPARQL is a graph-matching query language.
- A SPARQL query consists of three parts:
  - Pattern matching: optional, union, filtering, ...
  - Solution modifiers: projection, distinct, order, limit, offset, ...
  - Output part: construction of new triples, ...

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series     conf:pods .
}
```

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series     conf:pods .
}
```

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:PartOf     ?Conf .
  ?Conf       swrc:series    conf:pods .
}
```

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series     conf:pods .
}
```

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series     conf:pods .
}
```

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:PartOf     ?Conf .
  ?Conf       swrc:series    conf:pods .
}
```

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series     conf:pods .
}
```

A SPARQL query consists of a:

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:PartOf     ?Conf .
  ?Conf       swrc:series    conf:pods .
}
```

A SPARQL query consists of a:

Body: Pattern matching expression

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:PartOf     ?Conf .
  ?Conf       swrc:series    conf:pods .
}
```

A SPARQL query consists of a:

Body: Pattern matching expression

Head: Processing of the variables

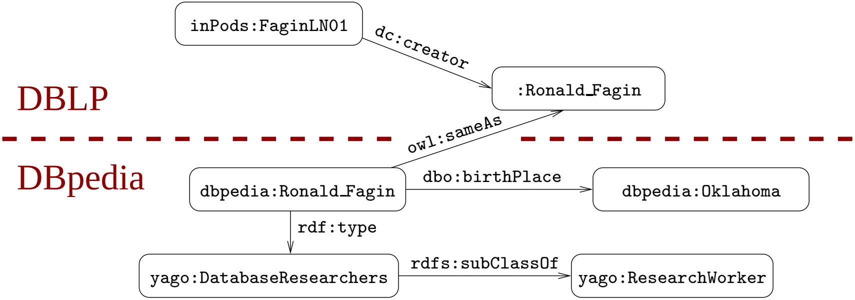
What are the challenges in implementing SPARQL?

SPARQL has to take into account the distinctive features of RDF:

- Should be able to extract information from interconnected RDF graphs
- Should be consistent with the open-world semantics of RDF
  - Should offer the possibility of adding optional information if present
- Should be able to properly interpret RDF graphs with a vocabulary with predefined semantics

Extracting information from interconnected RDF graphs

```
        : <http://dblp.l3s.de/d2r/resource/authors/>
dbpedia: <http://dbpedia.org/resource/>
    rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
   rdfs: <http://www.w3.org/2000/01/rdf-schema#>
    owl: <http://www.w3.org/2002/07/owl#>
   yago: <http://dbpedia.org/class/yago>
    dbo: <http://dbpedia.org/ontology/>
```



Dereferenceable URIs are the glue

`http://dbpedia.org/resource/Ronald_Fagin`

Querying interconnected RDF graphs

Retrieve the authors that have published in PODS and were born in Oklahoma:

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series     conf:pods .
  ?Person     owl:sameAs      ?Author .
  ?Person     dbo:birthPlace dbpedia:Oklahoma .
}
```

Retrieving optional information

Retrieve the authors that have published in PODS, and their Web pages if this information is available:

```
SELECT ?Author ?WebPage
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:PartOf     ?Conf .
  ?Conf       swrc:series    conf:pods .
  OPTIONAL { ?Author  foaf:homePage  ?WebPage . }
}
```

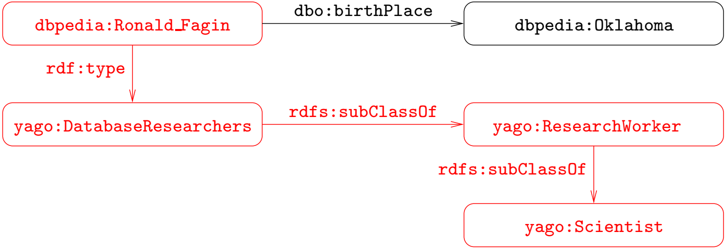
Taking into account vocabularies with predefined semantics

Retrieve the scientists that were born in Oklahoma and that have published in PODS:

```
SELECT ?Author
WHERE
{
  ?Author     rdf:type        yago:Scientist .
  ?Author     dbo:birthPlace dbpedia:Oklahoma .
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series     conf:pods .
}
```

Taking into account vocabularies with predefined semantics

Retrieve the scientists that were born in Oklahoma and that have published in PODS:



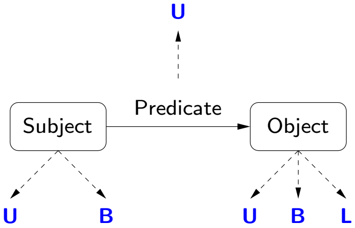
Outline of the talk

- RDF
- SPARQL: Syntax and semantics
- RDFS: RDF Schema
- Some new features in SPARQL 1.1
- Concluding remarks

Outline of the talk

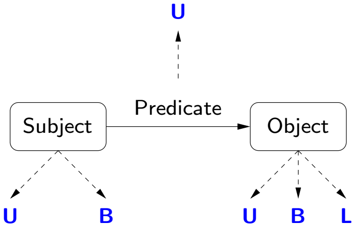
- RDF
- SPARQL: Syntax and semantics
- RDFS: RDF Schema
- Some new features in SPARQL 1.1
- Concluding remarks

RDF formal model



U	:	set of URIs
B	:	set of blank nodes
L	:	set of literals

RDF formal model



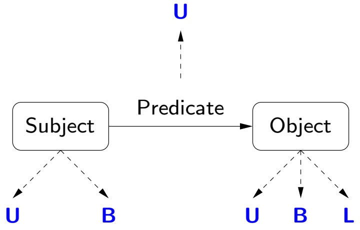
U : set of URIs

B : set of blank nodes

L : set of literals

$(s, p, o) \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$ is called an RDF triple

RDF formal model



U : set of URIs

B : set of blank nodes

L : set of literals

$(s, p, o) \in (\mathbf{U} \cup \mathbf{B}) \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{B} \cup \mathbf{L})$ is called an RDF triple

A set of RDF triples is called an RDF graph

RDF formal model

Proviso

In this talk, we do not consider blank nodes

- $(s, p, o) \in \mathbf{U} \times \mathbf{U} \times (\mathbf{U} \cup \mathbf{L})$ is called an RDF triple

Outline of the talk

- RDF
- SPARQL: Syntax and semantics
- RDFS: RDF Schema
- Some new features in SPARQL 1.1
- Concluding remarks

SPARQL queries can be complex

Interesting features:

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering

```
{ P1
  P2 }
```

SPARQL queries can be complex

Interesting features:

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering

```
{ { P1
    P2 }


  { P3
    P4 }


}
```

SPARQL queries can be complex

Interesting features:

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7 }  }

}
```

SPARQL queries can be complex

Interesting features:

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
```

SPARQL queries can be complex

Interesting features:

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9 }
```

SPARQL queries can be complex

Interesting features:

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering

```
{ { P1
    P2
    OPTIONAL { P5 }  }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9
  FILTER ( R ) }
```

SPARQL: An algebraic syntax

- Graph pattern:

`?X name ?Y`	$(?X, \text{name}, ?Y)$
`{ P1 . P2 }`	$(P_1 \text{ AND } P_2)$
`{ P1 OPTIONAL { P2 }}`	$(P_1 \text{ OPT } P_2)$
`{ P1 } UNION { P2 }`	$(P_1 \text{ UNION } P_2)$
`{ P1 FILTER ( R ) }`	$(P_1 \text{ FILTER } R)$

- SPARQL query:

`SELECT ?X ?Y ... { P }`	$(\text{SELECT } \{?X, ?Y, \ldots\} \; P)$

SPARQL: An algebraic syntax (cont'd)

- Explicit precedence/association

Example

```
{ t1
  t2
  OPTIONAL { t3 }
  OPTIONAL { t4 }
  t5
}
```

$$((((t_1 \text{ AND } t_2) \text{ OPT } t_3) \text{ OPT } t_4) \text{ AND } t_5)$$

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \; : \; \mathbf{V} \longrightarrow (\mathbf{U} \cup \mathbf{L})$$

The evaluation of a graph pattern results in a set of mappings.

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \quad : \quad \mathbf{V} \longrightarrow (\mathbf{U} \cup \mathbf{L})$$

The evaluation of a graph pattern results in a set of mappings.

The semantics of triple patterns

Given an RDF graph G and a triple pattern t.

Definition

The evaluation of t over G is the set of mappings μ such that:

- μ has as domain the variables in t: $\mathrm{dom}(\mu) = \mathrm{var}(t)$
- μ makes t to match the graph: $\mu(t) \in G$

The semantics of triple patterns

Given an RDF graph G and a triple pattern t.

Definition

The evaluation of t over G is the set of mappings μ such that:

- μ has as domain the variables in t: $\text{dom}(\mu) = \text{var}(t)$
- μ makes t to match the graph: $\mu(t) \in G$

Example

graph	triple	evaluation
$(R_1$, name, john$)$ $(R_1$, email, J@ed.ex$)$ $(R_2$, name, paul$)$	$(?X$, name, $?Y)$	see below

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph G and a triple pattern t.

Definition

The evaluation of t over G is the set of mappings μ such that:

- μ has as domain the variables in t: $\text{dom}(\mu) = \text{var}(t)$
- μ makes t to match the graph: $\mu(t) \in G$

Example

graph	triple	evaluation
(R_1, name, john)		
(R_1, email, J@ed.ex)	(?X, name, ?Y)	
(R_2, name, paul)		

	?X	?Y
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph G and a triple pattern t.

Definition

The evaluation of t over G is the set of mappings μ such that:

- μ has as domain the variables in t: $\text{dom}(\mu) = \text{var}(t)$
- μ makes t to match the graph: $\mu(t) \in G$

Example

graph	triple	evaluation
(R_1, name, john) (R_1, email, J@ed.ex) (R_2, name, paul)	(?X, name, ?Y)	

	?X	?Y
μ_1:	R_1	john
μ_2:	R_2	paul

Compatible mappings

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2

Compatible mappings

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2

Compatible mappings

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	

Compatible mappings

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	

Compatible mappings

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

Compatible mappings

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

- μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let Ω_1 and Ω_2 be sets of mappings.

Definition

Join: extends mappings in Ω_1 with compatible mappings in Ω_2

- $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1,\ \mu_2 \in \Omega_2 \text{ and } \mu_1,\ \mu_2 \text{ are compatible}\}$

Difference: selects mappings in Ω_1 that cannot be extended with mappings in Ω_2

- $\Omega_1 \smallsetminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \text{there is no mapping in } \Omega_2 \text{ compatible with } \mu_1\}$

Sets of mappings and operations

Definition

Union: includes mappings in Ω_1 and in Ω_2

- $\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}$

Left Outer Join: extends mappings in Ω_1 with compatible mappings in Ω_2 **if possible**

- $\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \smallsetminus \Omega_2)$

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{aligned}
[\![t]\!]_G &= \\
[\![(P_1 \text{ AND } P_2)]\!]_G &= \\
[\![(P_1 \text{ UNION } P_2)]\!]_G &= \\
[\![(P_1 \text{ OPT } P_2)]\!]_G &= \\
[\![(\text{SELECT } W \ P)]\!]_G &=
\end{aligned}
$$

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{aligned}
[\![t]\!]_G &= \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\} \\
[\![(P_1 \text{ AND } P_2)]\!]_G &= [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \\
[\![(P_1 \text{ UNION } P_2)]\!]_G &= [\![P_1]\!]_G \cup [\![P_2]\!]_G \\
[\![(P_1 \text{ OPT } P_2)]\!]_G &= [\![P_1]\!]_G ⟕ [\![P_2]\!]_G \\
[\![(\text{SELECT } W \ P)]\!]_G &= \{\mu_{|W} \mid \mu \in [\![P]\!]_G\}
\end{aligned}
$$

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{aligned}
[\![t]\!]_G &= \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\} \\
[\![(P_1 \text{ AND } P_2)]\!]_G &= [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \\
[\![(P_1 \text{ UNION } P_2)]\!]_G &= [\![P_1]\!]_G \cup [\![P_2]\!]_G \\
[\![(P_1 \text{ OPT } P_2)]\!]_G &= [\![P_1]\!]_G ⟕ [\![P_2]\!]_G \\
[\![(\text{SELECT } W \; P)]\!]_G &= \{\mu_{|W} \mid \mu \in [\![P]\!]_G\}
\end{aligned}
$$

$\mathrm{dom}(\mu_{|W}) = \mathrm{dom}(\mu) \cap W$ and

$\mu_{|W}(?X) = \mu(?X)$ for every $?X \in \mathrm{dom}(\mu_{|W})$

Semantics of SPARQL: An example

Example

$$(R_1, \text{name}, \text{john})$$
$$(R_1, \text{email}, \text{J@ed.ex})$$
$$(R_2, \text{name}, \text{paul})$$

$$((?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E))$$

Semantics of SPARQL: An example

Example

$(R_1$, name, john)
$(R_1$, email, J@ed.ex)
$(R_2$, name, paul)

$((?X,\ \text{name},\ ?Y)$ OPT $(?X,\ \text{email},\ ?E))$

Semantics of SPARQL: An example

Example

$(R_1$, name, john$)$
$(R_1$, email, J@ed.ex$)$
$(R_2$, name, paul$)$

$((?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{email},\ ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

Semantics of SPARQL: An example

Example

$$(R_1, \text{name}, \text{john})$$
$$(R_1, \text{email}, \text{J@ed.ex})$$
$$(R_2, \text{name}, \text{paul})$$

$$((?X, \text{ name}, ?Y) \text{ OPT } (?X, \text{ email}, ?E))$$

$?X$	$?Y$
R_1	john
R_2	paul

Semantics of SPARQL: An example

Example

$(R_1$, name, john$)$
$(R_1$, email, J@ed.ex$)$
$(R_2$, name, paul$)$

$((?X,$ name, $?Y)$ OPT $(?X,$ email, $?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?E$
R_1	J@ed.ex

Semantics of SPARQL: An example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{ name}, ?Y) \text{ OPT } (?X, \text{ email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?E$
R_1	J@ed.ex

Semantics of SPARQL: An example

Example

$(R_1$, name, john$)$
$(R_1$, email, J@ed.ex$)$
$(R_2$, name, paul$)$

$((?X,$ name, $?Y)$ OPT $(?X,$ email, $?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

Semantics of SPARQL: An example

Example

$(R_1$, name, john$)$
$(R_1$, email, J@ed.ex$)$
$(R_2$, name, paul$)$

$((?X,$ name, $?Y)$ OPT $(?X,$ email, $?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join

Semantics of SPARQL: An example

Example

$(R_1, \text{name}, \text{john})$
$(R_1, \text{email}, \text{J@ed.ex})$
$(R_2, \text{name}, \text{paul})$

$((?X, \text{ name}, ?Y) \text{ OPT } (?X, \text{ email}, ?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Difference

Semantics of SPARQL: An example

Example

$(R_1$, name, john$)$
$(R_1$, email, J@ed.ex$)$
$(R_2$, name, paul$)$

$((?X,$ name, $?Y)$ OPT $(?X,$ email, $?E))$

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Union

Filter expressions (value constraints)

Filter expression: P FILTER R

- P is a graph pattern
- R is a built-in condition

We consider in R:

- equality $=$ among variables and RDF terms
- unary predicate bound
- boolean combinations ($\wedge$, $\vee$, $\neg$)

Filter expressions (value constraints)

Example

Some filter expressions:

$$(?X = \texttt{conf:pods})$$
$$\neg(?X = \texttt{conf:pods})$$
$$(?X = \texttt{conf:pods}) \vee (?Y = \texttt{conf:sigmod})$$
$$(?X = \texttt{conf:pods}) \wedge \neg(?Y = \texttt{conf:sigmod})$$

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$
- R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$
- R is bound$(?X)$ and $?X \in \text{dom}(\mu)$

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$
- R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$
- R is bound$(?X)$ and $?X \in \text{dom}(\mu)$
- R is $\neg R_1$ and $\mu \models R_1$ does not hold
- R is $(R_1 \vee R_2)$, and $\mu \models R_1$ or $\mu \models R_2$
- R is $(R_1 \wedge R_2)$, $\mu \models R_1$ and $\mu \models R_2$

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$
- R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$
- R is $\text{bound}(?X)$ and $?X \in \text{dom}(\mu)$
- R is $\neg R_1$ and $\mu \models R_1$ does not hold
- R is $(R_1 \vee R_2)$, and $\mu \models R_1$ or $\mu \models R_2$
- R is $(R_1 \wedge R_2)$, $\mu \models R_1$ and $\mu \models R_2$

Definition

FILTER : selects mappings that satisfy a condition

$$[\![(P \text{ FILTER } R)]\!]_G \quad = \quad \{\mu \in [\![P]\!]_G \mid \mu \models R\}$$

Outline of the talk

- RDF
- SPARQL: Syntax and semantics
- RDFS: RDF Schema
- Some new features in SPARQL 1.1
- Concluding remarks

Syntax of RDFS

RDFS extends RDF with a schema vocabulary: subPropertyOf (`rdf:sp`), subClassOf (`rdf:sc`), domain (`rdf:dom`), range (`rdf:range`), type (`rdf:type`).

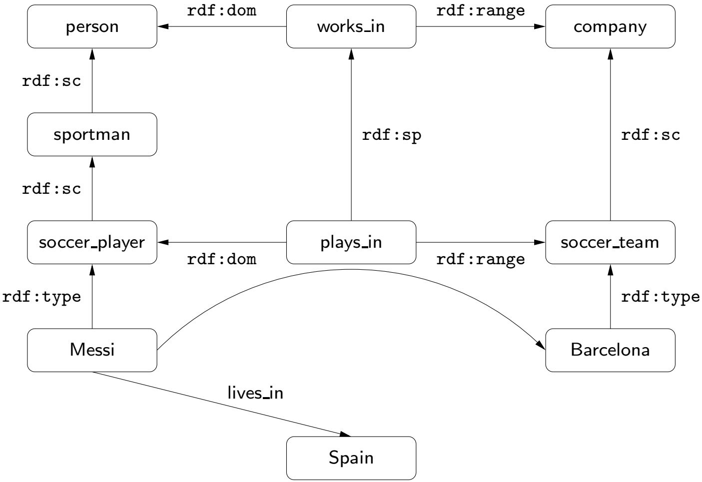
Syntax of RDFS

RDFS extends RDF with a schema vocabulary: subPropertyOf (`rdf:sp`), subClassOf (`rdf:sc`), domain (`rdf:dom`), range (`rdf:range`), type (`rdf:type`).

How can one query RDFS data?

- Evaluating queries which involve this vocabulary is challenging

A simple SPARQL query: (Messi, rdf:type, person)



Semantics of RDFS

Checking whether a triple t is in a graph G is the basic step when answering queries over RDF.

- For the case of RDFS, we need to check whether t is implied by G

The notion of entailment in RDFS can be defined in terms of classical notions such as model, interpretation, etc.

- As for the case of first-order logic

This notion can also be characterized by a set of inference rules.

An inference system for RDFS

Inference rule: $\frac{R}{R'}$

- R and R' are sequences of RDF triples including symbols $\mathcal{A}$, $\mathcal{X}$, …, to be replaced by elements from $(\mathbf{U} \cup \mathbf{L})$

Instantiation of a rule: $\frac{\sigma(R)}{\sigma(R')}$

- $\sigma : \{\mathcal{A}, \mathcal{X}, \ldots\} \to (\mathbf{U} \cup \mathbf{L})$

Application of a rule $\frac{R}{R'}$ to an RDF graph G:

- Select an assignment $\sigma : \{\mathcal{A}, \mathcal{X}, \ldots\} \to (\mathbf{U} \cup \mathbf{L})$
- if $\sigma(R) \subseteq G$, then obtain $G \cup \sigma(R')$

An inference system for RDFS (cont'd)

Sub-property :

$$\frac{(\mathcal{A},\ \texttt{rdf:sp},\ \mathcal{B})\ (\mathcal{B},\ \texttt{rdf:sp},\ \mathcal{C})}{(\mathcal{A},\ \texttt{rdf:sp},\ \mathcal{C})}$$

$$\frac{(\mathcal{A},\ \texttt{rdf:sp},\ \mathcal{B})\ (\mathcal{X},\ \mathcal{A},\ \mathcal{Y})}{(\mathcal{X},\ \mathcal{B},\ \mathcal{Y})}$$

Subclass :

$$\frac{(\mathcal{A},\ \texttt{rdf:sc},\ \mathcal{B})\ (\mathcal{B},\ \texttt{rdf:sc},\ \mathcal{C})}{(\mathcal{A},\ \texttt{rdf:sc},\ \mathcal{C})}$$

$$\frac{(\mathcal{A},\ \texttt{rdf:sc},\ \mathcal{B})\ (\mathcal{X},\ \texttt{rdf:type},\ \mathcal{A})}{(\mathcal{X},\ \texttt{rdf:type},\ \mathcal{B})}$$

Typing :

$$\frac{(\mathcal{A},\ \texttt{rdf:dom},\ \mathcal{B})\ (\mathcal{X},\ \mathcal{A},\ \mathcal{Y})}{(\mathcal{X},\ \texttt{rdf:type},\ \mathcal{B})}$$

$$\frac{(\mathcal{A},\ \texttt{rdf:range},\ \mathcal{B})\ (\mathcal{X},\ \mathcal{A},\ \mathcal{Y})}{(\mathcal{Y},\ \texttt{rdf:type},\ \mathcal{B})}$$

Entailment in RDFS

The previous system of inference rules characterize the notion of entailment in RDFS (without blank nodes).

Thus, a triple t can be deduced from an RDF graph G ($G \models t$) if there exists an RDF G' such that:

- $t \in G'$
- G' can be obtained from G by successively applying the rules in the previous system.

Entailment in RDFS: Closure of a graph

Definition

The closure of an RDFS graph G ($\text{cl}(G)$) is the graph obtained by adding to G all the triples that are implied by G.

A basic property of the closure:

- $G \models t$ iff $t \in \text{cl}(G)$

Querying RDFS data

Basic step for answering queries over RDFS:

- Checking whether a triple t is in $\text{cl}(G)$

Querying RDFS data

Basic step for answering queries over RDFS:

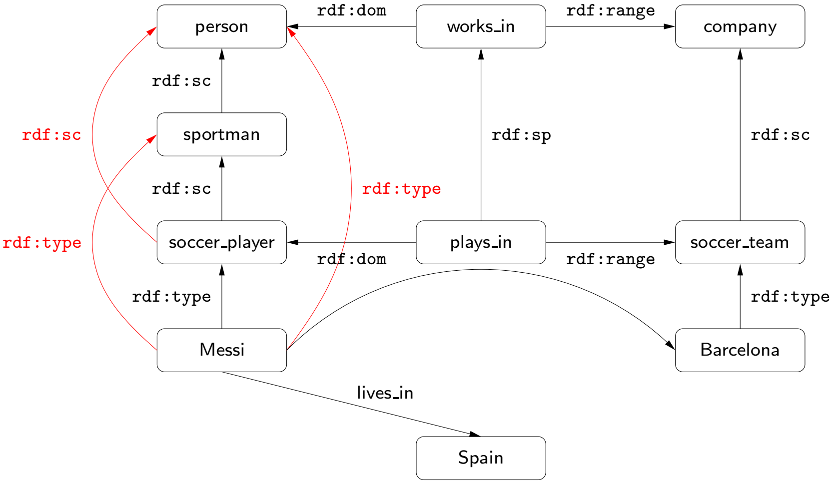
- Checking whether a triple t is in $\mathrm{cl}(G)$

Definition

The *RDFS-evaluation of a graph pattern* P *over an RDFS graph* G is defined as the evaluation of P over $\mathrm{cl}(G)$:

$$[\![P]\!]_G^{\mathrm{rdfs}} = [\![P]\!]_{\mathrm{cl}(G)}$$

Example: (Messi, rdf:type, person) over the closure



Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDF graph G:

- Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDF graph G:

- Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF

This approach has some drawbacks:

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDF graph G:

- Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF

This approach has some drawbacks:

- The size of the closure of G can be quadratic in the size of G

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDF graph G:

- Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF

This approach has some drawbacks:

- The size of the closure of G can be quadratic in the size of G
- Once the closure has been computed, all the queries are evaluated over a graph which can be much larger than the original graph

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDF graph G:

- Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF

This approach has some drawbacks:

- The size of the closure of G can be quadratic in the size of G
- Once the closure has been computed, all the queries are evaluated over a graph which can be much larger than the original graph
- The approach is not goal-oriented

  When evaluating $(a, \texttt{rdf:sc}, b)$, a goal-oriented approach should not compute $\text{cl}(G)$:

  - It should just verify whether there exists a path from a to b in G where every edge has label `rdf:sc`

Extending SPARQL with navigational capabilities

The example (a, `rdf:sc`, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Extending SPARQL with navigational capabilities

The example (a, `rdf:sc`, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

Extending SPARQL with navigational capabilities

The example (a, `rdf:sc`, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDF graph G is answered by navigating G (cl(G) is not computed)

Extending SPARQL with navigational capabilities

The example $(a, \texttt{rdf:sc}, b)$ suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDF graph G is answered by navigating G ($\text{cl}(G)$ is not computed)

This approach has some advantages:

Extending SPARQL with navigational capabilities

The example (a, rdf:sc, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDF graph G is answered by navigating G (cl(G) is not computed)

This approach has some advantages:

- It is goal-oriented

Extending SPARQL with navigational capabilities

The example (a, `rdf:sc`, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDF graph G is answered by navigating G (cl(G) is not computed)

This approach has some advantages:

- It is goal-oriented
- It has been used to design query languages for XML (e.g., XPath and XQuery). The results for these languages can be used here

Extending SPARQL with navigational capabilities

The example (a, `rdf:sc`, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDF graph G is answered by navigating G (cl(G) is not computed)

This approach has some advantages:

- It is goal-oriented
- It has been used to design query languages for XML (e.g., XPath and XQuery). The results for these languages can be used here
- Navigational operators allow to express natural queries that are not expressible in SPARQL over RDFS

Outline of the talk

- RDF
- SPARQL: Syntax and semantics
- RDFS: RDF Schema
- Some new features in SPARQL 1.1
- Concluding remarks

SPARQL 1.1

A new version of SPARQL has just been released (March 2013): SPARQL 1.1

Some new features in SPARQL 1.1:

SPARQL 1.1

A new version of SPARQL has just been released (March 2013): SPARQL 1.1

Some new features in SPARQL 1.1:

- Nesting of SELECT expressions

SPARQL 1.1

A new version of SPARQL has just been released (March 2013): SPARQL 1.1

Some new features in SPARQL 1.1:

- Nesting of SELECT expressions
- Entailment regime for RDFS

SPARQL 1.1

A new version of SPARQL has just been released (March 2013): SPARQL 1.1

Some new features in SPARQL 1.1:

- Nesting of SELECT expressions
- Entailment regime for RDFS
- Navigational capabilities: Property paths

SPARQL 1.1

A new version of SPARQL has just been released (March 2013): SPARQL 1.1

Some new features in SPARQL 1.1:

- Nesting of SELECT expressions
- Entailment regime for RDFS
- Navigational capabilities: Property paths
- Aggregates

SPARQL 1.1

A new version of SPARQL has just been released (March 2013): SPARQL 1.1

Some new features in SPARQL 1.1:

- Nesting of SELECT expressions
- Entailment regime for RDFS
- Navigational capabilities: Property paths
- Aggregates
- An operator SERVICE to distribute the execution of a query
  - (SERVICE $?X$ P)

SPARQL 1.1

A new version of SPARQL has just been released (March 2013): SPARQL 1.1

Some new features in SPARQL 1.1:

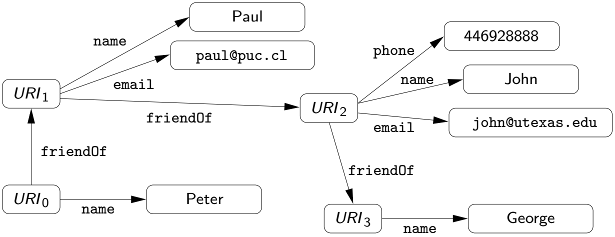
- Nesting of SELECT expressions
- Entailment regime for RDFS
- Navigational capabilities: Property paths
- Aggregates
- An operator SERVICE to distribute the execution of a query
  - (SERVICE ?X P)

Aggregates in SPARQL 1.1

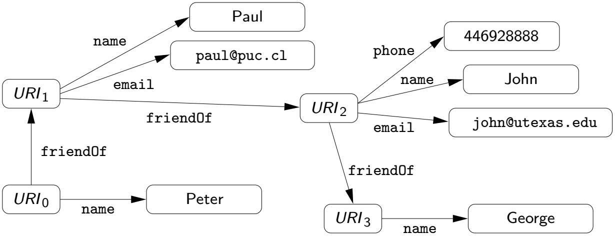
```
SELECT COUNT(DISTINCT ?Author)
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series     conf:pods .
}
```

This query can be executed in the DBLP SPARQL endpoint.

SPARQL provides limited navigational capabilities

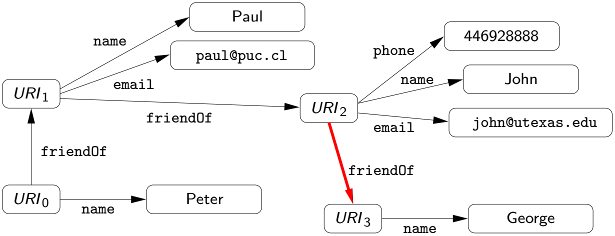


SPARQL provides limited navigational capabilities



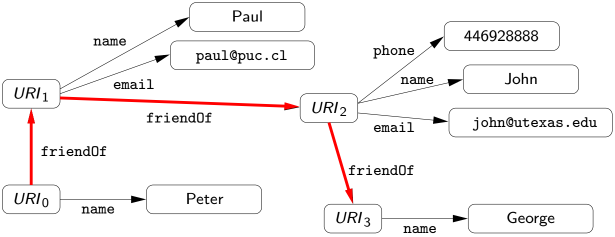
(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))

SPARQL provides limited navigational capabilities



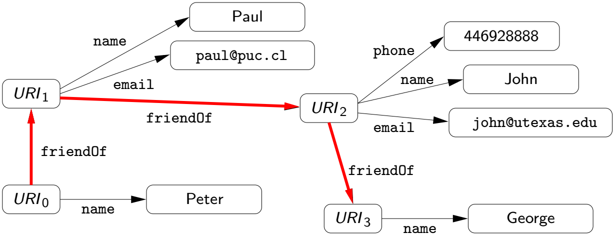
(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))

SPARQL provides limited navigational capabilities

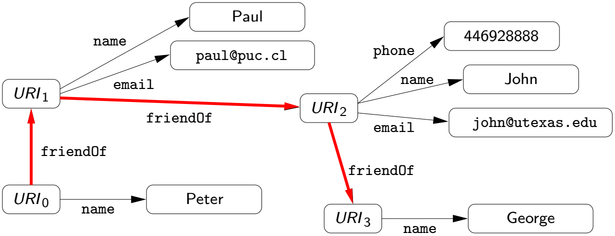


(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))

A possible solution: Property paths



A possible solution: Property paths



(SELECT $?X$ (($?X$, (friendOf)*, $?Y$) AND ($?Y$, name, George)))

Navigational capabilities in SPARQL 1.1: Property paths

Syntax of property paths:

$$exp \quad := \quad a \;\mid\; exp/exp \;\mid\; exp|exp \;\mid\; exp^*$$

where $a \in \mathbf{U}$

Evaluating property paths

The evaluation of a property path over an RDF graph G is defined as follows:

Evaluating property paths

The evaluation of a property path over an RDF graph G is defined as follows:

$$[\![a]\!]_G \quad = \quad \{(x, y) \mid (x, a, y) \in G\}$$

Evaluating property paths

The evaluation of a property path over an RDF graph G is defined as follows:

$$
\begin{aligned}
[\![a]\!]_G &= \{(x,y) \mid (x,a,y) \in G\} \\
[\![exp_1/exp_2]\!]_G &= \{(x,y) \mid \exists z\ (x,z) \in [\![exp_1]\!]_G \text{ and } \\
&\qquad (z,y) \in [\![exp_2]\!]_G\}
\end{aligned}
$$

Evaluating property paths

The evaluation of a property path over an RDF graph G is defined as follows:

$$
\begin{aligned}
[\![a]\!]_G &= \{(x, y) \mid (x, a, y) \in G\} \\
[\![exp_1/exp_2]\!]_G &= \{(x, y) \mid \exists z\ (x, z) \in [\![exp_1]\!]_G \text{ and} \\
&\qquad\qquad\qquad (z, y) \in [\![exp_2]\!]_G\} \\
[\![exp_1|exp_2]\!]_G &= [\![exp_1]\!]_G \cup [\![exp_2]\!]_G
\end{aligned}
$$

Evaluating property paths

The evaluation of a property path over an RDF graph G is defined as follows:

$$
\begin{aligned}
[\![a]\!]_G &= \{(x,y) \mid (x,a,y) \in G\} \\
[\![exp_1/exp_2]\!]_G &= \{(x,y) \mid \exists z\ (x,z) \in [\![exp_1]\!]_G \text{ and } \\
&\qquad (z,y) \in [\![exp_2]\!]_G\} \\
[\![exp_1|exp_2]\!]_G &= [\![exp_1]\!]_G \cup [\![exp_2]\!]_G \\
[\![exp^*]\!]_G &= \{(a,a) \mid a \text{ is a URI in } G\} \cup [\![exp]\!]_G \cup \\
&\qquad [\![exp/exp]\!]_G \cup [\![exp/exp/exp]\!]_G \cup \cdots
\end{aligned}
$$

Property paths in SPARQL 1.1

New element in SPARQL 1.1: A triple of the form (x, exp, y)

- *exp* is a property path
- x (resp. y) is either an element from **U** or a variable

Property paths in SPARQL 1.1

New element in SPARQL 1.1: A triple of the form (x, exp, y)

- *exp* is a property path
- x (resp. y) is either an element from **U** or a variable

Example

- $(?X, (\texttt{rdf:sc})^*, ?Y)$: Verifies whether $?X$ is a subclass of $?Y$
- $(?X, (\texttt{rdf:sp})^*, ?Y)$: Verifies whether $?X$ is a subproperty of $?Y$

Semantics of property paths

Evaluation of $t = (?X, exp, ?Y)$ over an RDF graph G is the set of mappings μ such that:

Semantics of property paths

Evaluation of $t = (?X, exp, ?Y)$ over an RDF graph G is the set of mappings μ such that:

- The domain of μ is $\{?X, ?Y\}$, and
- $(\mu(?X), \mu(?Y)) \in [\![exp]\!]_G$

Semantics of property paths

Evaluation of $t = (?X, exp, ?Y)$ over an RDF graph G is the set of mappings μ such that:

- The domain of μ is $\{?X, ?Y\}$, and
- $(\mu(?X), \mu(?Y)) \in [\![exp]\!]_G$

Example

- $(?X$, KLM/(KLM)*, $?Y)$: It is possible to go from $?X$ to $?Y$ by using the airline KLM
- $((?X$, KLM/(KLM)*, $?Y)$ FILTER $\neg(?X = ?Y))$: Same as before, but now $?X$, $?Y$ must be different

SPARQL 1.1 and RDFS

Property paths can help in encoding the semantics of RDFS.

- Given a SPARQL graph pattern P, we would like to find a SPARQL 1.1 graph pattern Q such that:

  For every RDF graph G: $[\![P]\!]_G^{\text{rdfs}} = [\![Q]\!]_G$

SPARQL 1.1 and RDFS

Property paths can help in encoding the semantics of RDFS.

- Given a SPARQL graph pattern P, we would like to find a SPARQL 1.1 graph pattern Q such that:

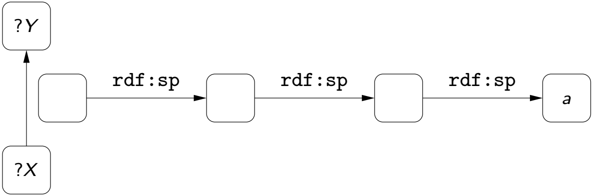
  For every RDF graph G: $[\![P]\!]^{\text{rdfs}}_G = [\![Q]\!]_G$

We already saw how to encode rdf:sc and rdf:sp.

- We will consider the example $P = (?X, a, ?Y)$, where $a \in \mathbf{U} \setminus \{\texttt{rdf:sc}, \texttt{rdf:sp}, \texttt{rdf:type}, \texttt{rdf:dom}, \texttt{rdf:range}\}$

The case of $P = (?X, a, ?Y)$

What are the difficulties in this case?



The case of $P = (?X, a, ?Y)$ (cont'd)

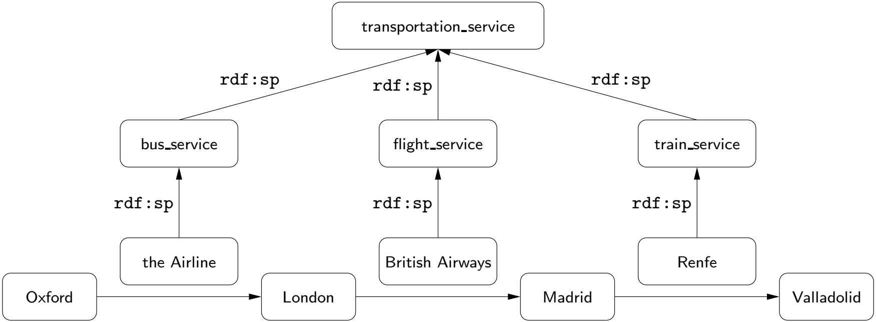
Let Q be:

$$\Big(\text{SELECT } \{?X, ?Y\} \quad ((?X, ?Z, ?Y) \text{ AND } (?Z, (\texttt{rdf:sp})^*, a))\Big)$$

Then for every RDF graph G: $[\![P]\!]_G^{\text{rdfs}} = [\![Q]\!]_G$

Are we done?

List the pairs *a*, *b* of cities such that there is a way to travel from *a* to *b*.



Concluding remarks

- We have witnessed a constant growth in the amount of RDF data available on the Web
- Two fundamental components of the Semantic Web: RDF and SPARQL
- Some of the distinctive features of RDF have made the study and implementation of SPARQL challenging
- SPARQL is still under development: SPARQL 1.1, ...

Thank you!