Semantic Web Data/RDF/SPARQL

Relational	Semantic Web
Tables	
SQL	

Relational	Semantic Web
Tables	*RDF Graphs*
SQL	

Relational	Semantic Web
Tables	*RDF Graphs*
SQL	*SPARQL*

Relational	Semantic Web
Tables	*RDF Graphs*
SQL	*SPARQL*
Closed Data (inside an organization)	

Relational	Semantic Web
Tables	*RDF Graphs*
SQL	*SPARQL*
Closed Data (inside an organization)	*Open Data* (available on the Web)

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.]

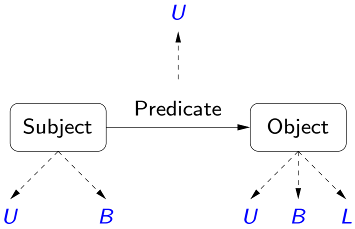
Specific Goals:

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

RDF in a nutshell

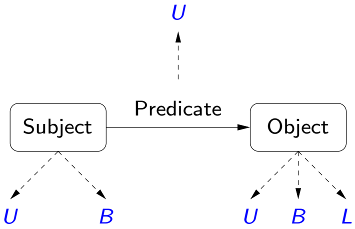
- RDF is the W3C proposal framework for representing information in the Web
- Abstract syntax based on directed labeled graph
- Schema definition language (RDFS): Define new vocabulary (typing, inheritance of classes and properties)
- Extensible URI-based vocabulary
- Formal semantics

RDF formal model



$$
\begin{aligned}
U &= \text{set of Uris} \\
B &= \text{set of Blank nodes} \\
L &= \text{set of Literals}
\end{aligned}
$$

RDF formal model



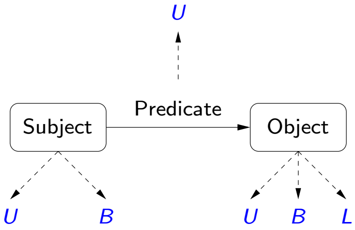
U = set of Uris

B = set of Blank nodes

L = set of Literals

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

RDF formal model



U = set of Uris

B = set of Blank nodes

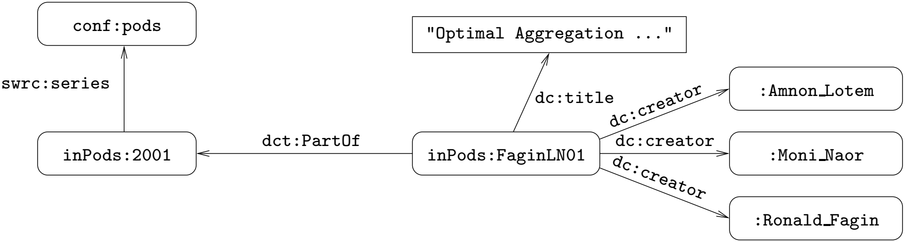
L = set of Literals

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

A set of RDF triples is called an RDF graph

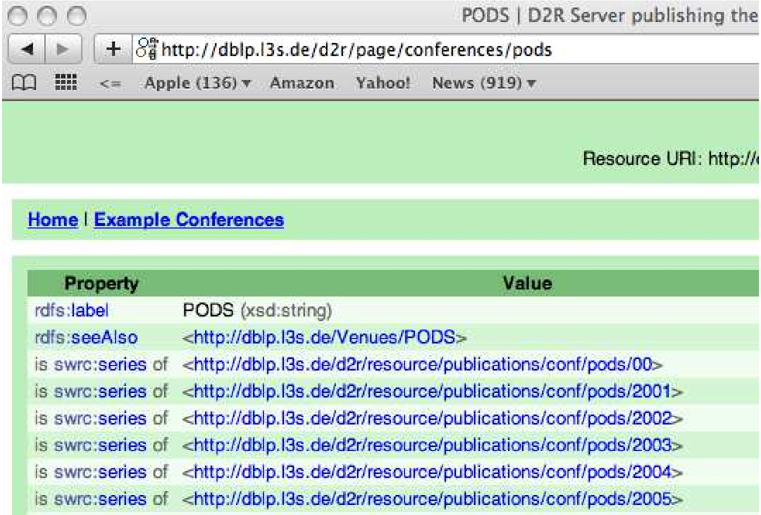
An example of an RDF graph: DBLP

:	<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`



PODS | D2R Server publishing the
http://dblp.l3s.de/d2r/page/conferences/pods
Apple (136) ▾
Amazon
Yahoo!
News (919) ▾

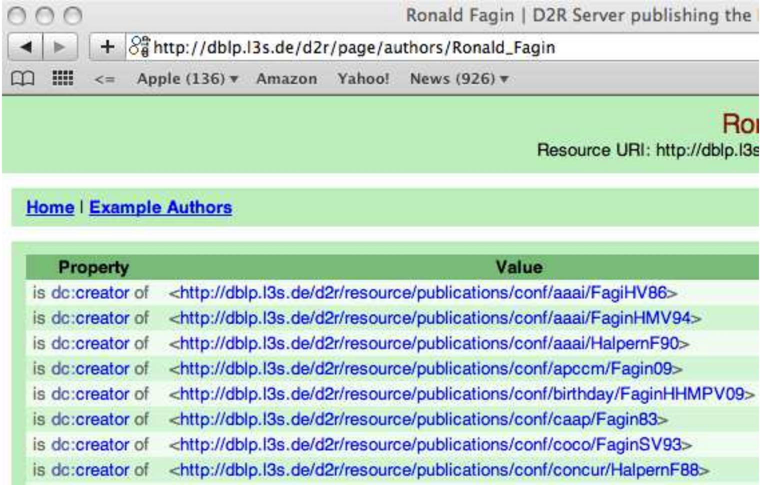
Resource URI: http://

Home | Example Conferences

Property	Value
rdfs:label	PODS (xsd:string)
rdfs:seeAlso	<http://dblp.l3s.de/Venues/PODS>
is swrc:series of	<http://dblp.l3s.de/d2r/resource/publications/conf/pods/00>
is swrc:series of	<http://dblp.l3s.de/d2r/resource/publications/conf/pods/2001>
is swrc:series of	<http://dblp.l3s.de/d2r/resource/publications/conf/pods/2002>
is swrc:series of	<http://dblp.l3s.de/d2r/resource/publications/conf/pods/2003>
is swrc:series of	<http://dblp.l3s.de/d2r/resource/publications/conf/pods/2004>
is swrc:series of	<http://dblp.l3s.de/d2r/resource/publications/conf/pods/2005>

URI can be used for any abstract resource

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



Ronald Fagin | D2R Server publishing the
http://dblp.l3s.de/d2r/page/authors/Ronald_Fagin
<=
Apple (136) ▾
Amazon
Yahoo!
News (926) ▾

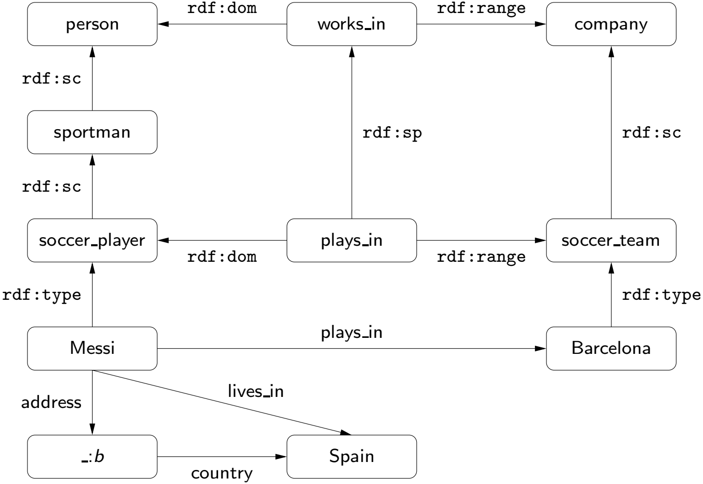
Ro

Resource URI: http://dblp.l3s

Home | Example Authors

Property	Value
is dc:creator of	<http://dblp.l3s.de/d2r/resource/publications/conf/aaai/FagiHV86>
is dc:creator of	<http://dblp.l3s.de/d2r/resource/publications/conf/aaai/FaginHMV94>
is dc:creator of	<http://dblp.l3s.de/d2r/resource/publications/conf/aaai/HalpernF90>
is dc:creator of	<http://dblp.l3s.de/d2r/resource/publications/conf/apccm/Fagin09>
is dc:creator of	<http://dblp.l3s.de/d2r/resource/publications/conf/birthday/FaginHHMPV09>
is dc:creator of	<http://dblp.l3s.de/d2r/resource/publications/conf/caap/Fagin83>
is dc:creator of	<http://dblp.l3s.de/d2r/resource/publications/conf/coco/FaginSV93>
is dc:creator of	<http://dblp.l3s.de/d2r/resource/publications/conf/concur/HalpernF88>

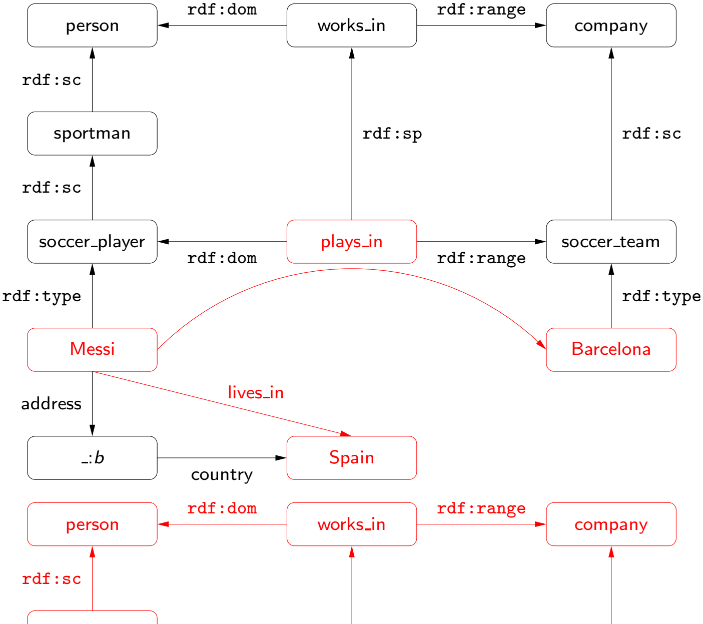
RDF: Another example



Some peculiarities of the RDF data model

- *Existential variables* as datavalues (null values)
- Built-in vocabulary with fixed semantics (RDFS)
- Graph model where nodes may also be edge labels

Previous example: A better representation

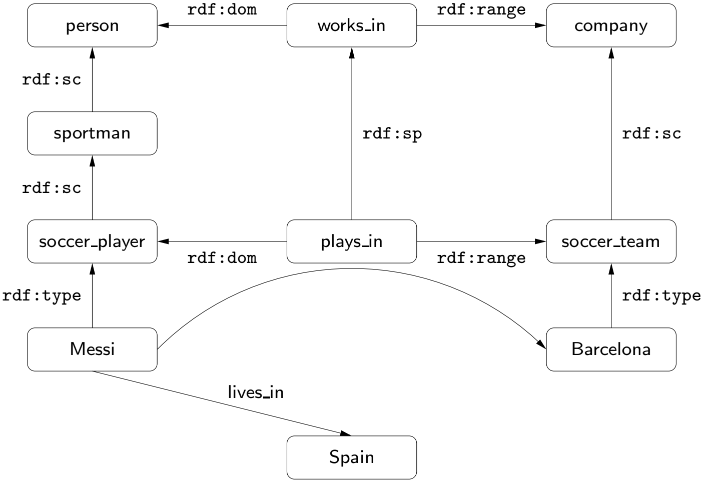


RDF + RDFS

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

plus *semantics* for this vocabulary

RDFS: Messi is a Person



Semantics of RDFS

Checking whether a triple t is in a graph G is the basic step when reasoning about RDF(S).

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

Semantics of RDFS

Checking whether a triple t is in a graph G is the basic step when reasoning about RDF(S).

- 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

Semantics of RDFS

Checking whether a triple t is in a graph G is the basic step when reasoning about RDF(S).

- 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.

Semantics of RDFS

Checking whether a triple t is in a graph G is the basic step when reasoning about RDF(S).

- 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

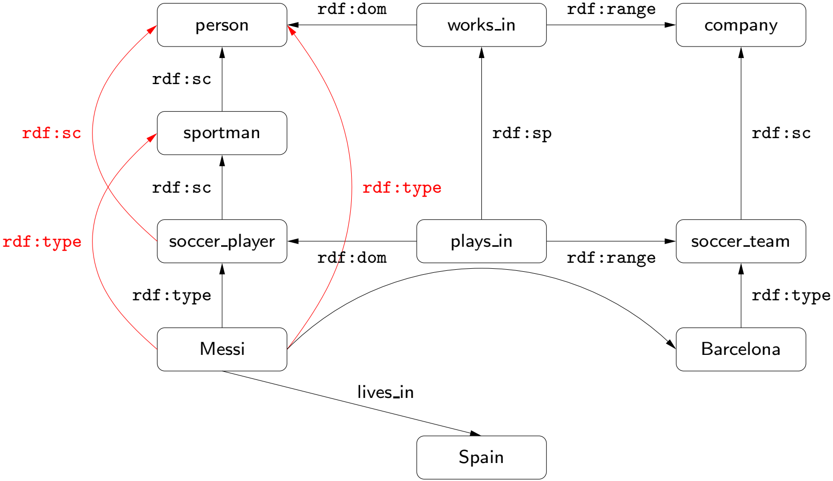
This notion can also be characterized by a set of inference rules.

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

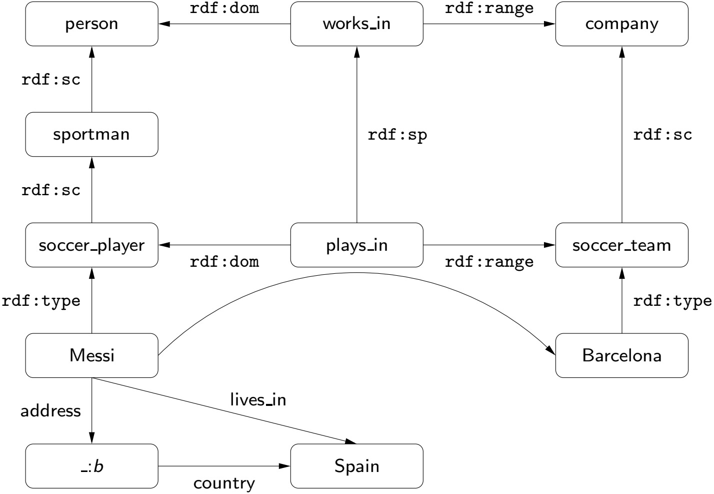
A basic property of the closure:

- G implies t iff $t \in \mathrm{cl}(G)$

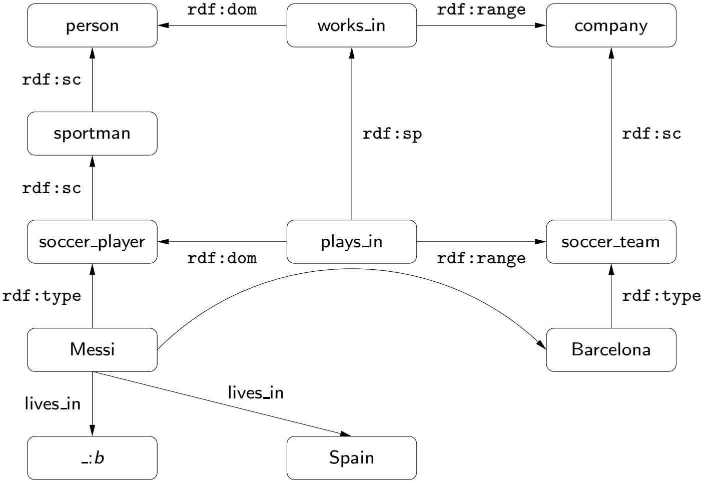
Example: (Messi, rdf:type, person) over the closure



Does the blank node add some information?



What about now?



SPARQL

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: A Simple RDF Query Language

Example: Authors that have published in PODS

SPARQL: A Simple RDF Query Language

Example: Authors that have published in PODS

```
SELECT ?Author
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in PODS

```
SELECT ?Author
WHERE
{



}
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in PODS

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .


}
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in PODS

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:partOf     ?Conf .

}
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in PODS

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

SPARQL: A Simple RDF Query Language

Example: Authors that have published in PODS

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

A SPARQL query consists of a:

SPARQL: A Simple RDF Query Language

Example: Authors that have published in PODS

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

A SPARQL query consists of a:

Head: Processing of the variables

SPARQL: A Simple RDF Query Language

Example: Authors that have published in PODS

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

A SPARQL query consists of a:

Head: Processing of the variables

Body: Pattern matching expression

SPARQL: A Simple RDF Query Language

Example: 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 . }
}
```

SPARQL: A Simple RDF Query Language

Example: 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 . }
}
```

But things can become more complex...

Interesting features of pattern matching on graphs

```
SELECT ?X1 ?X2 ...
  { P1 .
    P2 }
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping

```
SELECT ?X1 ?X2 ...
 {{ P1 .
    P2 }


  { P3 .
    P4 }


 }
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts

```
SELECT ?X1 ?X2 ...
 {{ P1 .
    P2
    OPTIONAL { P5 }  }

  { P3 .
    P4
    OPTIONAL { P7 }  }

 }
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting

```
SELECT ?X1 ?X2 ...
 {{ P1 .
    P2
    OPTIONAL { P5 }  }

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

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns

```
SELECT ?X1 ?X2 ...
{{{ P1 .
    P2
    OPTIONAL { P5 }  }

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

But things can become more complex...

Interesting features of pattern matching on graphs

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

```
SELECT ?X1 ?X2 ...
{{{ P1 .
    P2
    OPTIONAL { P5 }  }

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

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...
- + several new features in the new version (March 2013): navigation, entailment regimes, federation, . . .

```
SELECT ?X1 ?X2 ...
{{{ P1 .
    P2
    OPTIONAL { P5 }  }

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

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns
- Filtering
- ...
- + several new features in the new version (March 2013): navigation, entailment regimes, federation, . . .

```
SELECT ?X1 ?X2 ...
{{{ P1 .
    P2
    OPTIONAL { P5 }  }

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

What is the (formal) *meaning* of a general SPARQL query?

SPARQL: An algebraic syntax

V: set of variables

Each variable is assumed to start with ?

SPARQL: An algebraic syntax

V: set of variables

Each variable is assumed to start with ?

Triple pattern: $t \in (U \cup V) \times (U \cup V) \times (U \cup L \cup V)$

Examples: $(?X, \text{name}, \text{john})$, $(?X, \text{name}, ?Y)$

SPARQL: An algebraic syntax

V: set of variables

Each variable is assumed to start with ?

Triple pattern: $t \in (U \cup V) \times (U \cup V) \times (U \cup L \cup V)$

Examples: $(?X, \text{name}, \text{john})$, $(?X, \text{name}, ?Y)$

Basic graph pattern (bgp): Finite set of triple patterns

Examples: $\{(?X, \text{knows}, ?Y), (?Y, \text{name}, \text{john})\}$

SPARQL: An algebraic syntax (cont'd)

Recursive definition of SPARQL graph patterns:

- Every basic graph pattern is a graph pattern
- If P_1, P_2 are graph patterns, then (P_1 AND P_2), (P_1 OPT P_2), (P_1 UNION P_2) are graph pattern
- If P is a graph pattern and R is a *built-in condition*, then (P FILTER R) is a graph pattern

SPARQL query:

- If P is a graph pattern and W is a finite set of variables, then (SELECT W P) is a SPARQL query

Standard versus algebraic notation

```
?X :name "john"
```

$(?X, \text{name}, \text{john})$

Standard versus algebraic notation

```
?X :name "john"
```

$(?X, \text{name}, \text{john})$

```
{  P1 .  P2  }
```

$(\, P_1 \text{ AND } P_2 \,)$

Standard versus algebraic notation

`?X :name "john"`	$(?X, \text{name}, \text{john})$
`{  P1 .  P2  }`	$(\, P_1 \text{ AND } P_2 \,)$
`{ P1 OPTIONAL { P2 }}`	$(\, P_1 \text{ OPT } P_2 \,)$

Standard versus algebraic notation

`?X :name "john"`	$(?X, \text{name}, \text{john})$
`{  P1 .  P2  }`	$(\,P_1 \text{ AND } P_2\,)$
`{ P1 OPTIONAL { P2 }}`	$(\,P_1 \text{ OPT } P_2\,)$
`{  P1 } UNION { P2 }`	$(\,P_1 \text{ UNION } P_2\,)$

Standard versus algebraic notation

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

Standard versus algebraic notation

`?X :name "john"`	$(?X, \text{name}, \text{john})$
`{  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 ) }`	$(\, P_1 \text{ FILTER } R \,)$
`SELECT  W  WHERE { P }`	$(\, \text{SELECT } \; W \; P \,)$

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \quad : \quad V \longrightarrow (U \cup L \cup B)$$

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \quad : \quad V \longrightarrow (U \cup L \cup B)$$

Given a mapping μ and a triple pattern t:

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \quad : \quad V \longrightarrow (U \cup L \cup B)$$

Given a mapping μ and a triple pattern t:

- $\mu(t)$: triple obtained from t replacing variables according to μ

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \quad : \quad V \longrightarrow (U \cup L \cup B)$$

Given a mapping μ and a triple pattern t:

- $\mu(t)$: triple obtained from t replacing variables according to μ

Example

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \quad : \quad V \longrightarrow (U \cup L \cup B)$$

Given a mapping μ and a triple pattern t:

- $\mu(t)$: triple obtained from t replacing variables according to μ

Example

$$\mu = \{?X \rightarrow R_1, ?Y \rightarrow R_2, ?Z \rightarrow \text{john}\}$$

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \quad : \quad V \longrightarrow (U \cup L \cup B)$$

Given a mapping μ and a triple pattern t:

- $\mu(t)$: triple obtained from t replacing variables according to μ

Example

$$\mu = \{?X \rightarrow R_1, ?Y \rightarrow R_2, ?Z \rightarrow \text{john}\}$$

$$t = (?X, \text{ name}, \ ?Z)$$

Mappings: building block for the semantics

Definition

A mapping is a partial function:

$$\mu \quad : \quad V \longrightarrow (U \cup L \cup B)$$

Given a mapping μ and a triple pattern t:

- $\mu(t)$: triple obtained from t replacing variables according to μ

Example

$$\mu = \{?X \rightarrow R_1, ?Y \rightarrow R_2, ?Z \rightarrow \text{john}\}$$

$$t = (?X, \text{ name}, \ ?Z)$$

$$\mu(t) = (R_1, \text{ name}, \text{ john})$$

The semantics of triple patterns

Definition

The evaluation of triple pattern t over a graph G, denoted by $[\![t]\!]_G$, is the set of all mappings μ such that:

The semantics of triple patterns

Definition

The evaluation of triple pattern t over a graph G, denoted by $[\![t]\!]_G$, is the set of all mappings μ such that:

- $\text{dom}(\mu)$ is exactly the set of variables occurring in t

The semantics of triple patterns

Definition

The evaluation of triple pattern t over a graph G, denoted by $[\![t]\!]_G$, is the set of all mappings μ such that:

- $\text{dom}(\mu)$ is exactly the set of variables occurring in t
- $\mu(t) \in G$

Example

$$G$$

$$(R_1, \text{name}, \text{john})$$

$$(R_1, \text{email}, \text{J@ed.ex})$$

$$(R_2, \text{name}, \text{paul})$$

$$[\![(?X, \text{name}, ?N)]\!]_G$$

Example

G

$(R_1, \text{name}, \text{john})$

$(R_1, \text{email}, \text{J@ed.ex})$

$(R_2, \text{name}, \text{paul})$

$$[\![(?X, \text{name}, ?N)]\!]_G$$

$$\left\{ \begin{array}{l} \mu_1 = \{?X \rightarrow R_1, ?N \rightarrow \text{john}\} \\ \mu_2 = \{?X \rightarrow R_2, ?N \rightarrow \text{paul}\} \end{array} \right\}$$

Example

$$G$$

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

$$[\![(?X, \text{name}, ?N)]\!]_G$$

$$\left\{ \begin{array}{l} \mu_1 = \{?X \rightarrow R_1, ?N \rightarrow \text{john}\} \\ \mu_2 = \{?X \rightarrow R_2, ?N \rightarrow \text{paul}\} \end{array} \right\}$$

$$[\![(?X, \text{email}, ?E)]\!]_G$$

Example

$$G$$

$$(R_1, \text{name}, \text{john})$$

$$(R_1, \text{email}, \text{J@ed.ex})$$

$$(R_2, \text{name}, \text{paul})$$

$$[\![(?X, \text{name}, ?N)]\!]_G$$

$$\left\{ \begin{array}{l} \mu_1 = \{?X \rightarrow R_1, ?N \rightarrow \text{john}\} \\ \mu_2 = \{?X \rightarrow R_2, ?N \rightarrow \text{paul}\} \end{array} \right\}$$

$$[\![(?X, \text{email}, ?E)]\!]_G$$

$$\{ \ \mu = \{?X \rightarrow R_1, ?E \rightarrow \text{J@ed.ex}\} \ \}$$

Example

G

$(R_1, \text{name}, \text{john})$

$(R_1, \text{email}, \text{J@ed.ex})$

$(R_2, \text{name}, \text{paul})$

$[\![(?X, \text{name}, ?N)]\!]_G$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul

$[\![(?X, \text{email}, ?E)]\!]_G$

	$?X$	$?E$
μ	R_1	J@ed.ex

Example

$$G$$

$$(R_1,\ \text{name},\ \text{john})$$

$$(R_1,\ \text{email},\ \text{J@ed.ex})$$

$$(R_2,\ \text{name},\ \text{paul})$$

$[\![(R_1, \text{webPage}, ?W)]\!]_G$

$[\![(R_3, \text{name}, \text{ringo})]\!]_G$

$[\![(R_2, \text{name}, \text{paul})]\!]_G$

Example

$$G$$

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

$[\![(R_1, \text{webPage}, ?W)]\!]_G$

$\{\quad\}$

$[\![(R_3, \text{name}, \text{ringo})]\!]_G$

$[\![(R_2, \text{name}, \text{paul})]\!]_G$

Example

$$G$$

$$(R_1,\ \text{name},\ \text{john})$$

$$(R_1,\ \text{email},\ \text{J@ed.ex})$$

$$(R_2,\ \text{name},\ \text{paul})$$

$$[\![(R_1, \text{webPage}, ?W)]\!]_G$$

$$\{\quad\}$$

$$[\![(R_2, \text{name}, \text{paul})]\!]_G$$

$$[\![(R_3, \text{name}, \text{ringo})]\!]_G$$

$$\{\quad\}$$

Example

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

$$[\![(R_1, \text{webPage}, ?W)]\!]_G$$

$$\{\quad\}$$

$$[\![(R_2, \text{name}, \text{paul})]\!]_G$$

$$\{\ \mu_\emptyset = \{\,\}\ \}$$

$$[\![(R_3, \text{name}, \text{ringo})]\!]_G$$

$$\{\quad\}$$

Semantics of SPARQL: Basic graph patterns

Let P be a basic graph pattern

- var(P): set of variables mentioned in P

Semantics of SPARQL: Basic graph patterns

Let P be a basic graph pattern

- $\text{var}(P)$: set of variables mentioned in P

Given a mapping μ such that $\text{var}(P) \subseteq \text{dom}(\mu)$:

$$\mu(P) \;=\; \{\mu(t) \mid t \in P\}$$

Semantics of SPARQL: Basic graph patterns

Let P be a basic graph pattern

- var(P): set of variables mentioned in P

Given a mapping μ such that $\text{var}(P) \subseteq \text{dom}(\mu)$:

$$\mu(P) \quad = \quad \{\mu(t) \mid t \in P\}$$

Definition

The evaluation of P over an RDF graph G, denoted by $[\![P]\!]_G$, is the set of mappings μ:

- $\text{dom}(\mu) = \text{var}(P)$
- $\mu(P) \subseteq G$

Semantics of basic graph patterns: An example

graph	bgp	evaluation
$(R_1, \text{name}, \text{john})$ $(R_1, \text{email}, \text{J@ed.ex})$ $(R_2, \text{name}, \text{paul})$	$\{(?X, \text{name}, ?Y),$ $(?X, \text{email}, ?Z)\}$	

Semantics of basic graph patterns: An example

graph	bgp	evaluation
$(R_1$, name, john$)$ $(R_1$, email, J@ed.ex$)$ $(R_2$, name, paul$)$	$\{(?X$, name, $?Y)$, $(?X$, email, $?Z)\}$	

Semantics of basic graph patterns: An example

graph	bgp	evaluation
$(R_1$, name, john$)$ $(R_1$, email, J@ed.ex$)$ $(R_2$, name, paul$)$	$\{(?X$, name, $?Y)$, $(?X$, email, $?Z)\}$	

Semantics of basic graph patterns: An example

graph

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

bgp

$\{(?X$, name, $?Y)$,
$(?X$, email, $?Z)\}$

evaluation

	$?X$	$?Y$	$?Z$
μ:	R_1	john	J@ed.ex

Semantics of basic graph patterns: An example

graph

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

bgp

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

evaluation

μ:

$?X$	$?Y$	$?Z$
R_1	john	J@ed.ex

Notation

t is used to represent $\{t\}$

Compatible mappings: mappings that can be merged

Definition

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

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

Compatible mappings: mappings that can be merged

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: mappings that can be merged

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: mappings that can be merged

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: mappings that can be merged

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: mappings that can be merged

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: mappings that can be merged

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: $\Omega_1 \bowtie \Omega_2$

- $\{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2,$ and μ_1, μ_2 are compatibles$\}$
- extending mappings in Ω_1 with compatible mappings in Ω_2

will be used to define AND

Sets of mappings and operations

Let Ω_1 and Ω_2 be sets of mappings:

Definition

Join: $\Omega_1 \bowtie \Omega_2$

- $\{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2,$ and μ_1, μ_2 are compatibles$\}$
- extending mappings in Ω_1 with compatible mappings in Ω_2

will be used to define AND

Definition

Union: $\Omega_1 \cup \Omega_2$

- $\{\mu \mid \mu \in \Omega_1$ or $\mu \in \Omega_2\}$
- mappings in Ω_1 plus mappings in Ω_2 (the usual union of sets)

will be used to define UNION

Sets of mappings and operations

Definition

Difference: $\Omega_1 \setminus \Omega_2$

- $\{\mu \in \Omega_1 \mid$ for all $\mu' \in \Omega_2$, μ and μ' are not compatibles$\}$
- mappings in Ω_1 that cannot be extended with mappings in Ω_2

Sets of mappings and operations

Definition

Difference: $\Omega_1 \smallsetminus \Omega_2$

- $\{\mu \in \Omega_1 \mid$ for all $\mu' \in \Omega_2$, μ and μ' are not compatibles$\}$
- mappings in Ω_1 that cannot be extended with mappings in Ω_2

Definition

Left outer join: $\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \smallsetminus \Omega_2)$

- extension of mappings in Ω_1 with compatible mappings in Ω_2
- plus the mappings in Ω_1 that cannot be extended.

will be used to define OPT

Semantics of SPARQL: AND, UNION, OPT and SELECT

Given an RDF graph G

Definition

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

Semantics of SPARQL: AND, UNION, OPT and SELECT

Given an RDF graph G

Definition

$$
\begin{array}{lcl}
[\![(P_1 \text{ AND } P_2)]\!]_G & = & [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \\
[\![(P_1 \text{ UNION } P_2)]\!]_G & = & [\![P_1]\!]_G \cup [\![P_2]\!]_G \\
[\![(P_1 \text{ OPT } P_2)]\!]_G & = & [\![P_1]\!]_G \text{⟕} [\![P_2]\!]_G \\
[\![(\text{SELECT } W \; P)]\!]_G & = & \{\mu_{|W} \mid \mu \in [\![P]\!]_G\}
\end{array}
$$

Semantics of SPARQL: AND, UNION, OPT and SELECT

Given an RDF graph G

Definition

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

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

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

Example (AND)

$$G: \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{AND}\ (?X,\ \text{email},\ ?E))]\!]_G$$

Example (AND)

$$G : \begin{array}{lll} (R_1, \text{name}, \text{john}) & (R_2, \text{name}, \text{paul}) & (R_3, \text{name}, \text{ringo}) \\ (R_1, \text{email}, \text{J@ed.ex}) & & (R_3, \text{email}, \text{R@ed.ex}) \\ & & (R_3, \text{webPage}, \text{www.ringo.com}) \end{array}$$

$$[\![((?X, \text{name}, ?N) \text{ AND } (?X, \text{email}, ?E))]\!]_G$$

$$[\![(?X, \text{name}, ?N)]\!]_G \bowtie [\![(?X, \text{email}, ?E)]\!]_G$$

Example (AND)

$$G:\quad \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{AND}\ (?X,\ \text{email},\ ?E))]\!]_G$$

$$[\![(?X,\ \text{name},\ ?N)]\!]_G \bowtie [\![(?X,\ \text{email},\ ?E)]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

Example (AND)

$$G : \begin{array}{lll} (R_1, \text{ name}, \text{ john}) & (R_2, \text{ name}, \text{ paul}) & (R_3, \text{ name}, \text{ ringo}) \\ (R_1, \text{ email}, \text{ J@ed.ex}) & & (R_3, \text{ email}, \text{ R@ed.ex}) \\ & & (R_3, \text{ webPage}, \text{ www.ringo.com}) \end{array}$$

$$[\![((?X, \text{ name}, \ ?N) \text{ AND } (?X, \text{ email}, \ ?E))]\!]_G$$

$$[\![(?X, \text{ name}, \ ?N)]\!]_G \bowtie [\![(?X, \text{ email}, \ ?E)]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

	$?X$	$?E$
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

Example (AND)

$$G : \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$\llbracket((?X,\ \text{name},\ ?N)\ \text{AND}\ (?X,\ \text{email},\ ?E))\rrbracket_G$$

$$\llbracket(?X,\ \text{name},\ ?N)\rrbracket_G \bowtie \llbracket(?X,\ \text{email},\ ?E)\rrbracket_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

$\bowtie$

	$?X$	$?E$
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

Example (AND)

G :	$(R_1,\ \text{name},\ \text{john})$	$(R_2,\ \text{name},\ \text{paul})$	$(R_3,\ \text{name},\ \text{ringo})$
	$(R_1,\ \text{email},\ \text{J@ed.ex})$		$(R_3,\ \text{email},\ \text{R@ed.ex})$
			$(R_3,\ \text{webPage},\ \text{www.ringo.com})$

$$[\![((?X,\ \text{name},\ ?N)\ \text{AND}\ (?X,\ \text{email},\ ?E))]\!]_G$$

$$[\![(?X,\ \text{name},\ ?N)]\!]_G \bowtie [\![(?X,\ \text{email},\ ?E)]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

$\bowtie$

	$?X$	$?E$
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

	$?X$	$?N$	$?E$
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex

Example (OPT)

$$G: \begin{array}{lll} (R_1, \text{name}, \text{john}) & (R_2, \text{name}, \text{paul}) & (R_3, \text{name}, \text{ringo}) \\ (R_1, \text{email}, \text{J@ed.ex}) & & (R_3, \text{email}, \text{R@ed.ex}) \\ & & (R_3, \text{webPage}, \text{www.ringo.com}) \end{array}$$

$$[\![((?X, \text{name}, ?N)\ \text{OPT}\ (?X, \text{email}, ?E))]\!]_G$$

Example (OPT)

$$G : \begin{array}{lll} (R_1, \text{name}, \text{john}) & (R_2, \text{name}, \text{paul}) & (R_3, \text{name}, \text{ringo}) \\ (R_1, \text{email}, \text{J@ed.ex}) & & (R_3, \text{email}, \text{R@ed.ex}) \\ & & (R_3, \text{webPage}, \text{www.ringo.com}) \end{array}$$

$$[\![((?X, \text{name}, ?N) \text{ OPT } (?X, \text{email}, ?E))]\!]_G$$

$$[\![(?X, \text{name}, ?N)]\!]_G \mathbin{⟕} [\![(?X, \text{email}, ?E)]\!]_G$$

Example (OPT)

$$G:\quad \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{OPT}\ (?X,\ \text{email},\ ?E))]\!]_G$$

$$[\![(?X,\ \text{name},\ ?N)]\!]_G \mathbin{⟕} [\![(?X,\ \text{email},\ ?E)]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

Example (OPT)

	$(R_1$, name, john$)$	$(R_2$, name, paul$)$	$(R_3$, name, ringo$)$
G :	$(R_1$, email, J@ed.ex$)$		$(R_3$, email, R@ed.ex$)$
			$(R_3$, webPage, www.ringo.com$)$

$$[\![((?X, \text{name}, ?N) \text{ OPT } (?X, \text{email}, ?E))]\!]_G$$

$$[\![(?X, \text{name}, ?N)]\!]_G ⟕ [\![(?X, \text{email}, ?E)]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

	$?X$	$?E$
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

Example (OPT)

$$G : \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{OPT}\ (?X,\ \text{email},\ ?E))]\!]_G$$

$$[\![(?X,\ \text{name},\ ?N)]\!]_G \mathbin{⟕} [\![(?X,\ \text{email},\ ?E)]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

$$⟕$$

	$?X$	$?E$
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

Example (OPT)

$$G: \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{OPT}\ (?X,\ \text{email},\ ?E))]\!]_G$$

$$[\![(?X,\ \text{name},\ ?N)]\!]_G ⟕ [\![(?X,\ \text{email},\ ?E)]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

⟕

	$?X$	$?E$
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

	$?X$	$?N$	$?E$
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex
μ_2	R_2	paul	

Example (OPT)

$$G: \quad \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{OPT}\ (?X,\ \text{email},\ ?E))]\!]_G$$

$$[\![(?X,\ \text{name},\ ?N)]\!]_G ⟕ [\![(?X,\ \text{email},\ ?E)]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

⟕

	$?X$	$?E$
μ_4	R_1	J@ed.ex
μ_5	R_3	R@ed.ex

	$?X$	$?N$	$?E$
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex
μ_2	R_2	paul	

Example (UNION)

$$G: \quad \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{email},\ ?\mathit{Info})\ \text{UNION}\ (?X,\ \text{webPage},\ ?\mathit{Info}))]\!]_G$$

Example (UNION)

$$G: \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{email},\ ?\mathit{Info})\ \text{UNION}\ (?X,\ \text{webPage},\ ?\mathit{Info}))]\!]_G$$

$$[\![(?X,\ \text{email},\ ?\mathit{Info})]\!]_G \cup [\![(?X,\ \text{webPage},\ ?\mathit{Info})]\!]_G$$

Example (UNION)

$$G: \begin{array}{lll} (R_1, \text{name}, \text{john}) & (R_2, \text{name}, \text{paul}) & (R_3, \text{name}, \text{ringo}) \\ (R_1, \text{email}, \text{J@ed.ex}) & & (R_3, \text{email}, \text{R@ed.ex}) \\ & & (R_3, \text{webPage}, \text{www.ringo.com}) \end{array}$$

$$[\![((?X, \text{email}, ?\mathit{Info}) \text{ UNION } (?X, \text{webPage}, ?\mathit{Info}))]\!]_G$$

$$[\![(?X, \text{email}, ?\mathit{Info})]\!]_G \cup [\![(?X, \text{webPage}, ?\mathit{Info})]\!]_G$$

	$?X$	$?\mathit{Info}$
μ_1	R_1	J@ed.ex
μ_2	R_3	R@ed.ex

Example (UNION)

$$G: \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$\llbracket((?X,\ \text{email},\ ?\mathit{Info})\ \text{UNION}\ (?X,\ \text{webPage},\ ?\mathit{Info}))\rrbracket_G$$

$$\llbracket(?X,\ \text{email},\ ?\mathit{Info})\rrbracket_G \cup \llbracket(?X,\ \text{webPage},\ ?\mathit{Info})\rrbracket_G$$

	$?X$	$?\mathit{Info}$
μ_1	R_1	J@ed.ex
μ_2	R_3	R@ed.ex

	$?X$	$?\mathit{Info}$
μ_3	R_3	www.ringo.com

Example (UNION)

$$G: \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{email},\ ?\mathit{Info})\ \text{UNION}\ (?X,\ \text{webPage},\ ?\mathit{Info}))]\!]_G$$

$$[\![(?X,\ \text{email},\ ?\mathit{Info})]\!]_G \cup [\![(?X,\ \text{webPage},\ ?\mathit{Info})]\!]_G$$

	$?X$	$?\mathit{Info}$
μ_1	R_1	J@ed.ex
μ_2	R_3	R@ed.ex

$\cup$

	$?X$	$?\mathit{Info}$
μ_3	R_3	www.ringo.com

Example (UNION)

	$(R_1, \text{name}, \text{john})$	$(R_2, \text{name}, \text{paul})$	$(R_3, \text{name}, \text{ringo})$
G :	$(R_1, \text{email}, \text{J@ed.ex})$		$(R_3, \text{email}, \text{R@ed.ex})$
			$(R_3, \text{webPage}, \text{www.ringo.com})$

$$[\![((?X, \text{email}, ?\mathit{Info}) \text{ UNION } (?X, \text{webPage}, ?\mathit{Info}))]\!]_G$$

$$[\![(?X, \text{email}, ?\mathit{Info})]\!]_G \cup [\![(?X, \text{webPage}, ?\mathit{Info})]\!]_G$$

	$?X$	$?\mathit{Info}$
μ_1	R_1	J@ed.ex
μ_2	R_3	R@ed.ex

$\cup$

	$?X$	$?\mathit{Info}$
μ_3	R_3	www.ringo.com

	$?X$	$?\mathit{Info}$
μ_1	R_1	J@ed.ex
μ_2	R_3	R@ed.ex
μ_3	R_3	www.ringo.com

Example (SELECT)

$$G: \quad \begin{array}{lll} (R_1, \text{name}, \text{john}) & (R_2, \text{name}, \text{paul}) & (R_3, \text{name}, \text{ringo}) \\ (R_1, \text{email}, \text{J@ed.ex}) & & (R_3, \text{email}, \text{R@ed.ex}) \\ & & (R_3, \text{webPage}, \text{www.ringo.com}) \end{array}$$

$$[\![(\text{SELECT}\ \{?N, ?E\}\ ((?X, \text{name}, ?N)\ \text{AND}\ (?X, \text{email}, ?E)))]\!]_G$$

Example (SELECT)

$$G: \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![(\text{SELECT } \{?N, ?E\}\ ((?X,\ \text{name},\ ?N)\ \text{AND}\ (?X,\ \text{email},\ ?E)))]\!]_G$$

SELECT$\{?N, ?E\}$

	$?X$	$?N$	$?E$
μ_1	R_1	john	J@ed.ex
μ_2	R_3	ringo	R@ed.ex

Example (SELECT)

$$G : \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![(\text{SELECT } \{?N, ?E\}\ ((?X,\ \text{name},\ ?N)\ \text{AND}\ (?X,\ \text{email},\ ?E)))]\!]_G$$

SELECT$\{?N, ?E\}$

	$?X$	$?N$	$?E$
μ_1	R_1	john	J@ed.ex
μ_2	R_3	ringo	R@ed.ex

	$?N$	$?E$
$\mu_1\vert_{\{?N,?E\}}$	john	J@ed.ex
$\mu_2\vert_{\{?N,?E\}}$	ringo	R@ed.ex

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 ($\wedge$, $\vee$, $\neg$)

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 $\text{bound}(?X)$ and $?X \in \text{dom}(\mu)$

Satisfaction of value constraints

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

- R is $?X = c$, $?X \in \mathrm{dom}(\mu)$ and $\mu(?X) = c$
- R is $?X = ?Y$, $?X, ?Y \in \mathrm{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$
- R is $\mathrm{bound}(?X)$ and $?X \in \mathrm{dom}(\mu)$
- usual rules for Boolean connectives

Satisfaction of value constraints

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

- R is $?X = c$, $?X \in \mathrm{dom}(\mu)$ and $\mu(?X) = c$
- R is $?X = ?Y$, $?X, ?Y \in \mathrm{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$
- R is $\mathrm{bound}(?X)$ and $?X \in \mathrm{dom}(\mu)$
- usual rules for Boolean connectives

Definition

FILTER : selects mappings that satisfy a condition

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

Example (FILTER)

$$G: \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{FILTER}\ (?N = \text{ringo} \vee ?N = \text{paul}))]\!]_G$$

Example (FILTER)

$$G : \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{FILTER}\ (?N = \text{ringo} \lor ?N = \text{paul}))]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

Example (FILTER)

$$G: \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{FILTER}\ (?N = \text{ringo} \vee ?N = \text{paul}))]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

$?N = \text{ringo} \vee ?N = \text{paul}$

Example (FILTER)

$$G : \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![((?X,\ \text{name},\ ?N)\ \text{FILTER}\ (?N = \text{ringo} \vee ?N = \text{paul}))]\!]_G$$

	$?X$	$?N$
μ_1	R_1	john
μ_2	R_2	paul
μ_3	R_3	ringo

$$?N = \text{ringo} \vee ?N = \text{paul}$$

	$?X$	$?N$
μ_2	R_2	paul
μ_3	R_3	ringo

Example (FILTER)

$$G: \quad \begin{array}{lll} (R_1, \text{name}, \text{john}) & (R_2, \text{name}, \text{paul}) & (R_3, \text{name}, \text{ringo}) \\ (R_1, \text{email}, \text{J@ed.ex}) & & (R_3, \text{email}, \text{R@ed.ex}) \\ & & (R_3, \text{webPage}, \text{www.ringo.com}) \end{array}$$

$$[\![(((?X, \text{name}, ?N) \text{ OPT } (?X, \text{email}, ?E)) \text{ FILTER } \neg\,\text{bound}(?E))]\!]_G$$

Example (FILTER)

$$G: \begin{array}{lll} (R_1, \text{name}, \text{john}) & (R_2, \text{name}, \text{paul}) & (R_3, \text{name}, \text{ringo}) \\ (R_1, \text{email}, \text{J@ed.ex}) & & (R_3, \text{email}, \text{R@ed.ex}) \\ & & (R_3, \text{webPage}, \text{www.ringo.com}) \end{array}$$

$$[\![(((?X, \text{name}, ?N) \text{ OPT } (?X, \text{email}, ?E)) \text{ FILTER } \neg\,\text{bound}(?E))]\!]_G$$

	$?X$	$?N$	$?E$
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex
μ_2	R_2	paul	

Example (FILTER)

$$G: \begin{array}{lll} (R_1, \text{name}, \text{john}) & (R_2, \text{name}, \text{paul}) & (R_3, \text{name}, \text{ringo}) \\ (R_1, \text{email}, \text{J@ed.ex}) & & (R_3, \text{email}, \text{R@ed.ex}) \\ & & (R_3, \text{webPage}, \text{www.ringo.com}) \end{array}$$

$$[\![(((?X, \text{name}, ?N) \text{ OPT } (?X, \text{email}, ?E)) \text{ FILTER } \neg \text{bound}(?E))]\!]_G$$

	$?X$	$?N$	$?E$
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex
μ_2	R_2	paul	

$\neg\,\text{bound}(?E)$

Example (FILTER)

$$G: \quad \begin{array}{lll} (R_1,\ \text{name},\ \text{john}) & (R_2,\ \text{name},\ \text{paul}) & (R_3,\ \text{name},\ \text{ringo}) \\ (R_1,\ \text{email},\ \text{J@ed.ex}) & & (R_3,\ \text{email},\ \text{R@ed.ex}) \\ & & (R_3,\ \text{webPage},\ \text{www.ringo.com}) \end{array}$$

$$[\![(((?X,\ \text{name},\ ?N)\ \text{OPT}\ (?X,\ \text{email},\ ?E))\ \text{FILTER}\ \neg\,\text{bound}(?E))]\!]_G$$

	$?X$	$?N$	$?E$
$\mu_1 \cup \mu_4$	R_1	john	J@ed.ex
$\mu_3 \cup \mu_5$	R_3	ringo	R@ed.ex
μ_2	R_2	paul	

$\neg\,\text{bound}(?E)$

	$?X$	$?N$
μ_2	R_2	paul

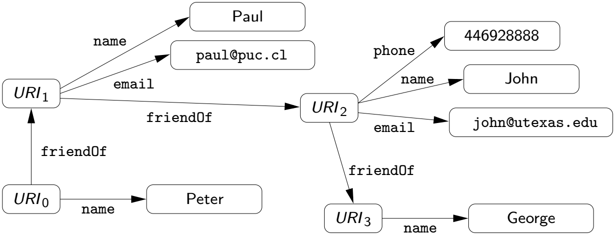
SPARQL 1.1

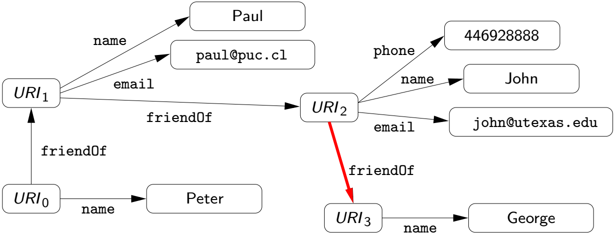
A new version of SPARQL was recently released (March 2013): SPARQL 1.1

Some new features in SPARQL 1.1:

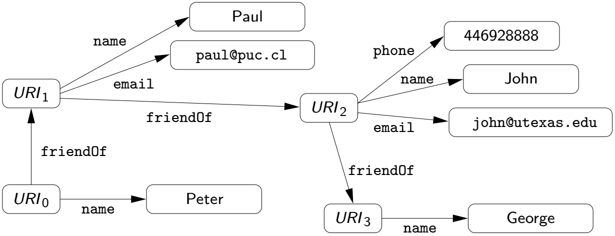
- Entailment regimes for RDFS and OWL
- Navigational capabilities: Property paths

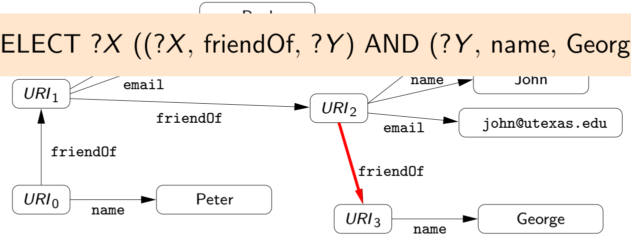
SPARQL provides limited navigational capabilities





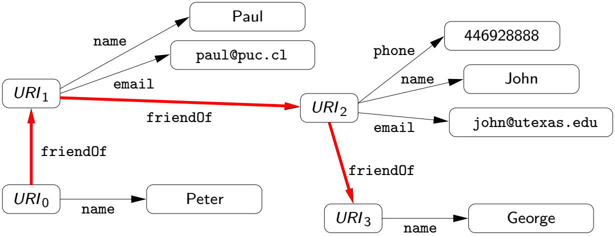
SPARQL provides limited navigational capabilities



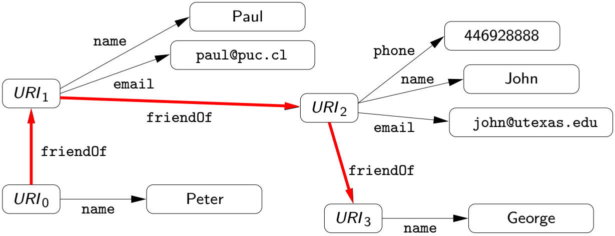


(SELECT $?X$ (($?X$, friendOf, $?Y$) AND ($?Y$, name, George)))

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 $a \in U$

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 $a \in U$

Other expressions are allowed:

$\hat{}exp$	:	inverse path
$!(a_1|\ldots|a_n)$	:	a URI which is not one of a_i $(1 \leq i \leq n)$

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:

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

Evaluating property paths

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

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

Evaluating property paths

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

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

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, (friendOf)*, ?Y): Checks whether there exists a path of friends of arbitrary length from ?X to ?Y

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

- $(?X, (\text{friendOf})^*, ?Y)$: Checks whether there exists a path of friends of arbitrary length from $?X$ to $?Y$
- $(?X, (\texttt{rdf:sc})^*, \text{person})$: Checks whether the value stored in $?X$ is a subclass of person

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

- $(?X, (\text{friendOf})^*, ?Y)$: Checks whether there exists a path of friends of arbitrary length from $?X$ to $?Y$
- $(?X, (\texttt{rdf:sc})^*, \text{person})$: Checks whether the value stored in $?X$ is a subclass of person
- $(?X, (\texttt{rdf:sp})^*, ?Y)$: Checks whether the value stored in $?X$ is a subproperty of the value stored in $?Y$

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:

- $\text{dom}(\mu) = \{?X, ?Y\}$

Semantics of property paths

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

- $\mathrm{dom}(\mu) = \{?X, ?Y\}$
- $(\mu(?X), \mu(?Y)) \in [\![exp]\!]_G$

Semantics of property paths

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

- $\text{dom}(\mu) = \{?X, ?Y\}$
- $(\mu(?X), \mu(?Y)) \in [\![exp]\!]_G$

Other cases are defined analogously.

Semantics of property paths

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

- $\text{dom}(\mu) = \{?X, ?Y\}$
- $(\mu(?X), \mu(?Y)) \in [\![exp]\!]_G$

Other cases are defined analogously.

Example

- $((?X$, KLM/(KLM)*, $?Y)$ FILTER $\neg(?X = ?Y))$: It is possible to go from $?X$ to $?Y$ by using the airline KLM, where $?X$, $?Y$ are different cities

Comments on papers

- Jorge Perez, Marcelo Arenas, Claudio Gutierrez: Semantics and complexity of SPARQL. ACM Trans. Database Syst. 34(3) (2009)
- M. Arenas, J. Perez: Querying semantic web data with SPARQL. PODS 2011: 305-316
  In these two papers, your essays ought to concentrate on complexity, as semantics was already covered.
- Marcelo Arenas, Georg Gottlob, Andreas Pieris: Expressive languages for querying the semantic web. PODS 2014: 14-26
  Extend SPARQL with more expressive ontologies and recursion, and translation into datalog.
- Leonid Libkin, Juan L. Reutter, Domagoj Vrgoc: Trial for RDF: adapting graph query languages for RDF data. PODS 2013: 201-212
  Are graph data and RDF the same? Not really. This shows how to bridge them.
- Jorge Perez, Marcelo Arenas, Claudio Gutierrez: nSPARQL: A navigational language for RDF. J. Web Sem. 8(4): 255-270 (2010)
  Extending navigational capabilities, using some XPath ideas.
- Marcelo Arenas, Sebastian Conca, Jorge Perez: Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. WWW 2012: 629-638
- Katja Losemann, Wim Martens: The complexity of regular expressions and property paths in SPARQL. ACM Trans. Database Syst. 38(4): 24 (2013)
  Two papers showing that bad things happen if one queries RDF accoring to SPARQL 1.1 standard, and different solutions for fixing the problem.