# Querying Semantic Web Data with SPARQL

Marcelo Arenas* and Jorge Pérez†

*Pontificia Universidad Católica de Chile
†Universidad de Chile

Santiago, January 2015

# RDF + SPARQL

# MOTIVATION

| 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<br>(inside an organization) | *Open Data*<br>(available on the Web) |

# Demo: Can you answer these questions?

# Demo: Can you answer these questions?

People in Wikipedia that has "University of Chile" as *alma mater*?

# Demo: Can you answer these questions?

People in Wikipedia that has "University of Chile" as *alma mater*?

Who is the oldest person appearing in Wikipedia
that was born in Chile?

# Demo: Can you answer these questions?

People in Wikipedia that has “University of Chile” as *alma mater*?

Who is the oldest person appearing in Wikipedia
that was born in Chile?

What is the rainiest place during February?

# RDF

# 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.]

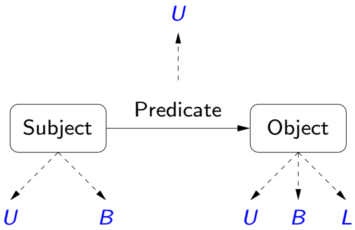
Specific Goals:

- 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

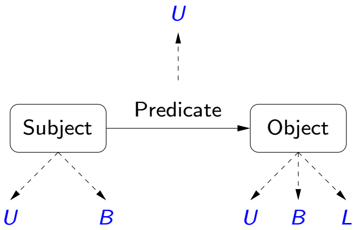
- RDF is the W3C proposal framework for representing information in the Web
- Abstract syntax based on directed labeled graph
- Schema definition language (RDFS): Define new vocabulary (typing, inheritance of classes and properties)
- Extensible URI-based vocabulary
- Formal semantics

# 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



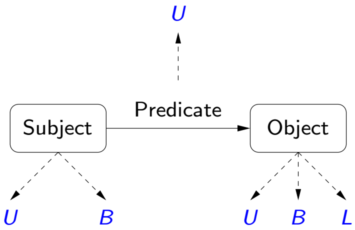

$U$ = set of Uris

$B$ = 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



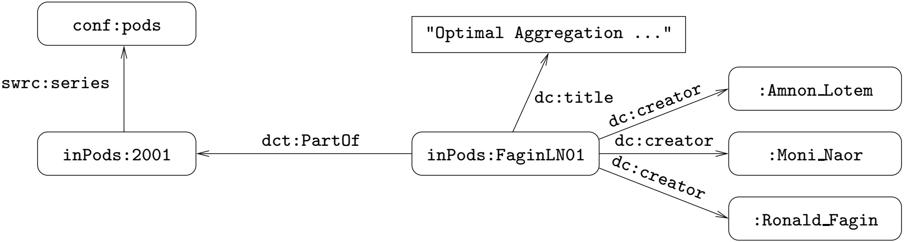

$$
\begin{aligned}
U &= \text{set of Uris} \\
B &= \text{set of Blank nodes} \\
L &= \text{set of Literals}
\end{aligned}
$$

$(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ 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/>
```



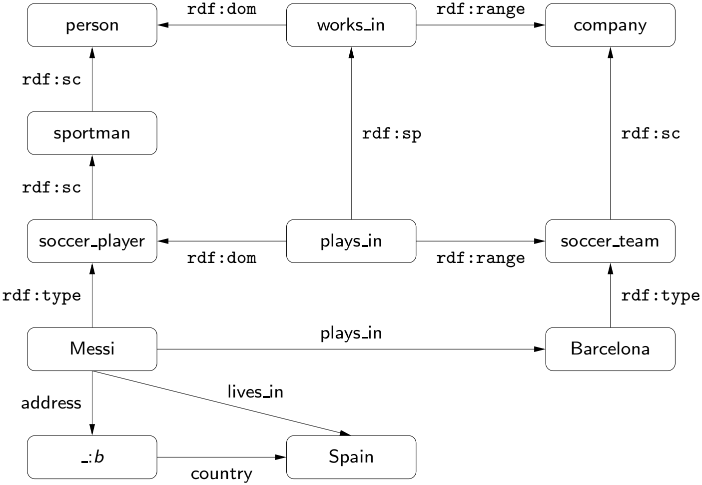
# An example of a URI

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

# URI can be used for any abstract resource

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

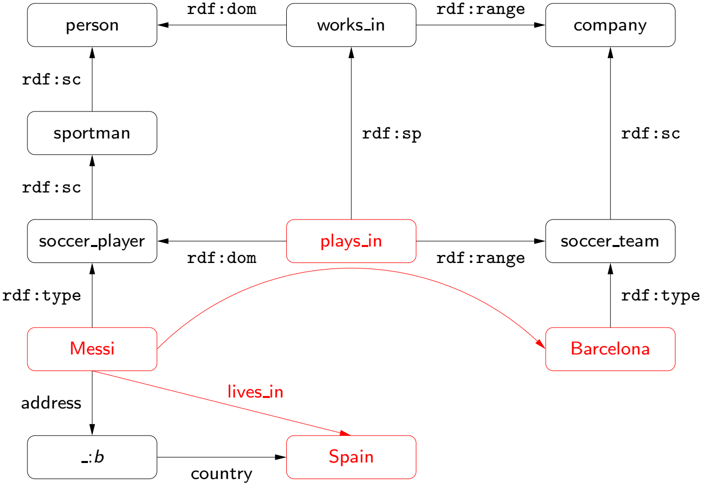
# RDF: Another example



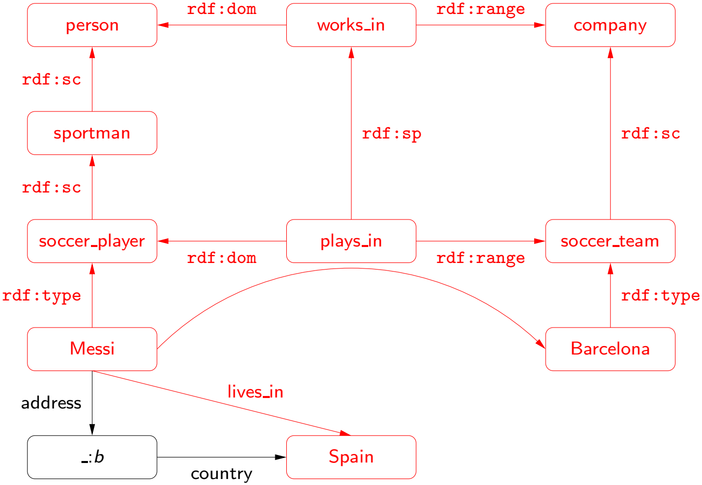
# Some peculiarities of the RDF data model

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

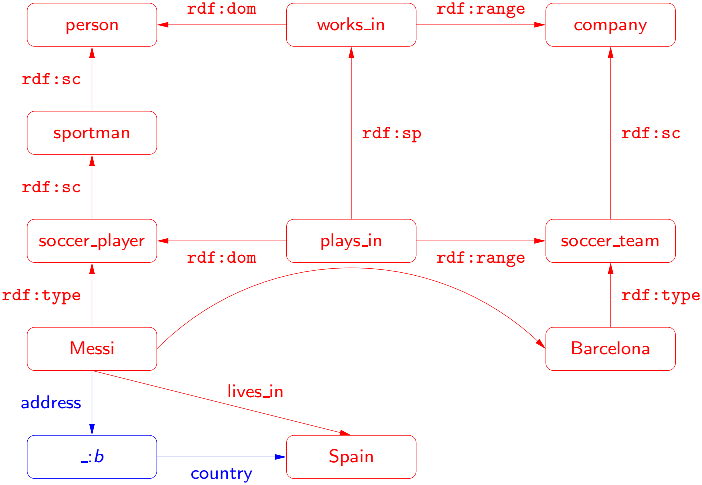
# Previous example: A better representation



# Previous example: A better representation



# Previous example: A better representation

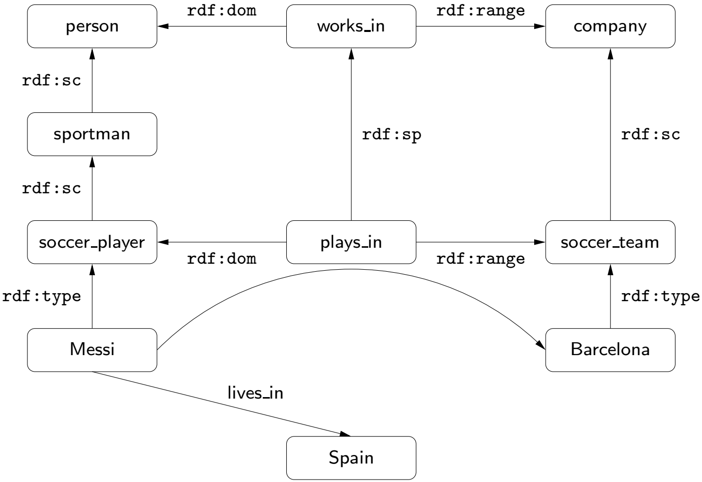


# RDF + RDFS

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

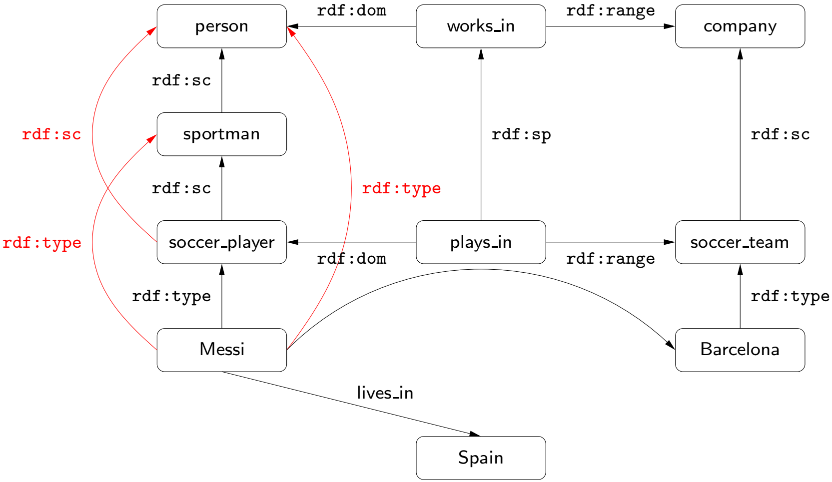
This notion can also be characterized by a set of inference rules.

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 $G$.

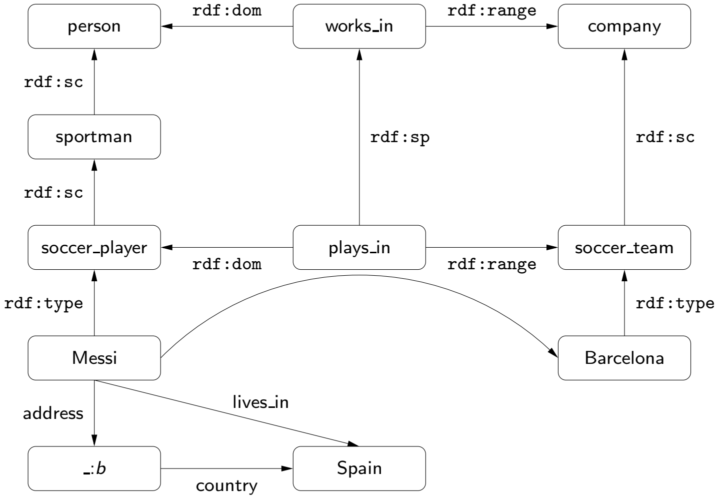
A basic property of the closure:

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

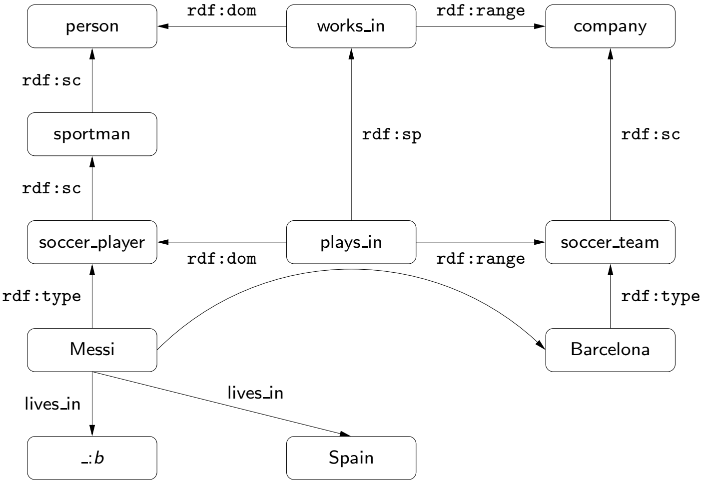
# Example: (Messi, rdf:type, person) over the closure



# 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 ISWC

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

```
SELECT ?Author
```

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

```
SELECT ?Author
WHERE
{



}
```

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

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


}
```

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

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

}
```

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

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

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

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

A SPARQL query consists of a:

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

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

A SPARQL query consists of a:

Head: Processing of the variables

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

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

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 ISWC, 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:iswc .

  OPTIONAL {
      ?Author  foaf:homePage  ?WebPage . }
}
```

# SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC, 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:iswc .

  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"
```

(?$X$, name, john)

```
{  P1 .  P2  }
```

( $P_1$ 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"` | $(?X, \text{name}, \text{john})$ |
| `{  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 ) }` | $(\, P_1 \text{ FILTER } R \,)$ |

# 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 $\mu$ 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 $\mu$ and a triple pattern $t$:

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

# 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 $\mu$ and a triple pattern $t$:

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

**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 $\mu$ and a triple pattern $t$:

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

**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 $\mu$ and a triple pattern $t$:

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

**Example**

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

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

# 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 $\mu$ and a triple pattern $t$:

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

**Example**

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

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

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

# 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 $\mu$ 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 $\mu$ 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 $\mu$ such that:

- $\mathrm{dom}(\mu)$ 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

$G$

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

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

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

## Example

$$G$$

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

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

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

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

## Example

$$G$$

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

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

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

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

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

## Example

$$G$$

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |

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

| | $?X$ | $?E$ |
|---|---|---|
| $\mu$ | $R_1$ | 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

$$G$$

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

- $\mathrm{var}(P)$: set of variables mentioned 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 $\mu$ such that var($P$) $\subseteq$ dom($\mu$):

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

# Semantics of SPARQL: Basic graph patterns

Let $P$ be a basic graph pattern

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

Given a mapping $\mu$ such that var($P$) $\subseteq$ 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 $\mu$:

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

# Semantics of basic graph patterns: An example

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

# Semantics of basic graph patterns: An example

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

# Semantics of basic graph patterns: An example

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

# Semantics of basic graph patterns: An example

| graph | bgp | evaluation |
|---|---|---|
| $(R_1, \text{name}, \text{john})$<br>$(R_1, \text{email}, \text{J@ed.ex})$<br>$(R_2, \text{name}, \text{paul})$ | $\{(?X, \text{name}, ?Y),$<br>$(?X, \text{email}, ?Z)\}$ | $\mu$: |

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

# Semantics of basic graph patterns: An example

| graph | bgp | evaluation |
| --- | --- | --- |
| $(R_1$, name, john) <br> $(R_1$, email, J@ed.ex) <br> $(R_2$, name, paul) | $\{(?X$, name, $?Y)$, <br> $(?X$, email, $?Z)\}$ | $\mu$: see table below |

| | $?X$ | $?Y$ | $?Z$ |
| --- | --- | --- | --- |
| $\mu$: | $R_1$ | john | J@ed.ex |

## Notation

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

# Compatible mappings: mappings that can be merged

**Definition**

Mappings $\mu_1$ and $\mu_2$ 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 $\mu_1$ and $\mu_2$ 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

| | $?X$ | $?Y$ | $?Z$ | $?V$ |
|---|---|---|---|---|
| $\mu_1$ : | $R_1$ | john | | |
| $\mu_2$ : | $R_1$ | | J@edu.ex | |
| $\mu_3$ : | | | P@edu.ex | $R_2$ |
| | | | | |

# Compatible mappings: mappings that can be merged

### Definition

Mappings $\mu_1$ and $\mu_2$ 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

| | $?X$ | $?Y$ | $?Z$ | $?V$ |
|---|---|---|---|---|
| $\mu_1$ : | $R_1$ | john | | |
| $\mu_2$ : | $R_1$ | | J@edu.ex | |
| $\mu_3$ : | | | P@edu.ex | $R_2$ |
| | | | | |

# Compatible mappings: mappings that can be merged

**Definition**

Mappings $\mu_1$ and $\mu_2$ 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**

| | $?X$ | $?Y$ | $?Z$ | $?V$ |
|---|---|---|---|---|
| $\mu_1$ : | $R_1$ | john | | |
| $\mu_2$ : | $R_1$ | | J@edu.ex | |
| $\mu_3$ : | | | 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 $\mu_1$ and $\mu_2$ 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**

| | $?X$ | $?Y$ | $?Z$ | $?V$ |
|---|---|---|---|---|
| $\mu_1$ : | $R_1$ | john | | |
| $\mu_2$ : | $R_1$ | | J@edu.ex | |
| $\mu_3$ : | | | 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 $\mu_1$ and $\mu_2$ 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**

| | $?X$ | $?Y$ | $?Z$ | $?V$ |
|---|---|---|---|---|
| $\mu_1$ : | $R_1$ | john | | |
| $\mu_2$ : | $R_1$ | | J@edu.ex | |
| $\mu_3$ : | | | 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 $\mu_1$ and $\mu_2$ 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**

| | $?X$ | $?Y$ | $?Z$ | $?V$ |
|---|---|---|---|---|
| $\mu_1$ : | $R_1$ | john | | |
| $\mu_2$ : | $R_1$ | | J@edu.ex | |
| $\mu_3$ : | | | 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 | $R_2$ |

- $\mu_2$ and $\mu_3$ are not compatible

# Sets of mappings and operations

Let $\Omega_1$ and $\Omega_2$ 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 $\mu_1, \mu_2$ are compatibles$\}$
- extending mappings in $\Omega_1$ with compatible mappings in $\Omega_2$

will be used to define AND

# Sets of mappings and operations

Let $\Omega_1$ and $\Omega_2$ be sets of mappings:

Definition

Join: $\Omega_1 \bowtie \Omega_2$

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

will be used to define AND

Definition

Union: $\Omega_1 \cup \Omega_2$

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

will be used to define UNION

# Sets of mappings and operations

**Definition**

Difference: $\Omega_1 \smallsetminus \Omega_2$

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

# Sets of mappings and operations

**Definition**

Difference: $\Omega_1 \smallsetminus \Omega_2$

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

**Definition**

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

- extension of mappings in $\Omega_1$ with compatible mappings in $\Omega_2$
- plus the mappings in $\Omega_1$ 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**

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

# Semantics of SPARQL: AND, UNION, OPT and SELECT

Given an RDF graph $G$

**Definition**

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

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

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

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

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

## Example (AND)

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

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

## Example (AND)

$G$ :

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

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

| | $?X$ | $?E$ |
|---|---|---|
| $\mu_4$ | $R_1$ | J@ed.ex |
| $\mu_5$ | $R_3$ | R@ed.ex |

## Example (AND)

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

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

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

| | ?X | ?N |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

$\bowtie$

| | ?X | ?E |
|---|---|---|
| $\mu_4$ | $R_1$ | J@ed.ex |
| $\mu_5$ | $R_3$ | R@ed.ex |

## Example (AND)

$G$ :

| | | |
|---|---|---|
| ($R_1$, name, john) | ($R_2$, name, paul) | ($R_3$, name, ringo) |
| ($R_1$, email, J@ed.ex) | | ($R_3$, email, R@ed.ex) |
| | | ($R_3$, webPage, 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 |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

$\bowtie$

| | ?X | ?E |
|---|---|---|
| $\mu_4$ | $R_1$ | J@ed.ex |
| $\mu_5$ | $R_3$ | 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)

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

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

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

## Example (OPT)

$G$ :

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

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

$(R_3,\ \text{name},\ \text{ringo})$ $(R_3,\ \text{email},\ \text{R@ed.ex})$ $(R_3,\ \text{webPage},\ \text{www.ringo.com})$

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

## Example (OPT)

$G$ :

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

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

| | $?X$ | $?E$ |
|---|---|---|
| $\mu_4$ | $R_1$ | J@ed.ex |
| $\mu_5$ | $R_3$ | R@ed.ex |

## Example (OPT)

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

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

⟕

| | $?X$ | $?E$ |
|---|---|---|
| $\mu_4$ | $R_1$ | J@ed.ex |
| $\mu_5$ | $R_3$ | R@ed.ex |

## Example (OPT)

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

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

⟕

| | $?X$ | $?E$ |
|---|---|---|
| $\mu_4$ | $R_1$ | J@ed.ex |
| $\mu_5$ | $R_3$ | 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 |
| $\mu_2$ | $R_2$ | paul | |

## Example (OPT)

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

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

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

| | ?X | ?N |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

⟕

| | ?X | ?E |
|---|---|---|
| $\mu_4$ | $R_1$ | J@ed.ex |
| $\mu_5$ | $R_3$ | 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 |
| $\mu_2$ | $R_2$ | paul | |

## Example (UNION)

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

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

## 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)

$G$ :

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

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

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

| | $?X$ | $?\mathit{Info}$ |
|---|---|---|
| $\mu_1$ | $R_1$ | J@ed.ex |
| $\mu_2$ | $R_3$ | R@ed.ex |

## Example (UNION)

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

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

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

| | $?X$ | $?Info$ |
|---|---|---|
| $\mu_1$ | $R_1$ | J@ed.ex |
| $\mu_2$ | $R_3$ | R@ed.ex |

| | $?X$ | $?Info$ |
|---|---|---|
| $\mu_3$ | $R_3$ | www.ringo.com |

## Example (UNION)

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

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

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

| | $?X$ | $?Info$ |
|---|---|---|
| $\mu_1$ | $R_1$ | J@ed.ex |
| $\mu_2$ | $R_3$ | R@ed.ex |

$\cup$

| | $?X$ | $?Info$ |
|---|---|---|
| $\mu_3$ | $R_3$ | www.ringo.com |

## Example (UNION)

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

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

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

| | $?X$ | $?Info$ |
|---|---|---|
| $\mu_1$ | $R_1$ | J@ed.ex |
| $\mu_2$ | $R_3$ | R@ed.ex |

$\cup$

| | $?X$ | $?Info$ |
|---|---|---|
| $\mu_3$ | $R_3$ | www.ringo.com |

| | $?X$ | $?Info$ |
|---|---|---|
| $\mu_1$ | $R_1$ | J@ed.ex |
| $\mu_2$ | $R_3$ | R@ed.ex |
| $\mu_3$ | $R_3$ | 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)

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

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

$$\text{SELECT}\{?N, ?E\} \quad \begin{array}{c|c|c|c|} \cline{2-4} & ?X & ?N & ?E \\ \cline{2-4} \mu_1 & R_1 & \text{john} & \text{J@ed.ex} \\ \cline{2-4} \mu_2 & R_3 & \text{ringo} & \text{R@ed.ex} \\ \cline{2-4} \end{array}$$

## Example (SELECT)

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

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

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

| | $?X$ | $?N$ | $?E$ |
|---|---|---|---|
| $\mu_1$ | $R_1$ | john | J@ed.ex |
| $\mu_2$ | $R_3$ | ringo | R@ed.ex |

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

# Filter expressions (value constraints)

Filter expression: $(P\ \text{FILTER}\ R)$

- $P$ 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 $\mu$ satisfies a condition $R$ ($\mu \models R$) if:

# Satisfaction of value constraints

A mapping $\mu$ 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 $\mu$ 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 $?X \in \text{dom}(\mu)$
- usual rules for Boolean connectives

# Satisfaction of value constraints

A mapping $\mu$ 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 $?X \in \text{dom}(\mu)$
- 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)

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

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

## Example (FILTER)

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

## Example (FILTER)

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

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

## Example (FILTER)

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

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_1$ | $R_1$ | john |
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_2$ | $R_2$ | paul |
| $\mu_3$ | $R_3$ | ringo |

## Example (FILTER)

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

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

## 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 |
| $\mu_2$ | $R_2$ | 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 |
| $\mu_2$ | $R_2$ | paul | |

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

## Example (FILTER)

$G$ :

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

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

$(R_3, \text{name}, \text{ringo})$ $(R_3, \text{email}, \text{R@ed.ex})$ $(R_3, \text{webPage}, \text{www.ringo.com})$

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

| | $?X$ | $?N$ | $?E$ |
|---|---|---|---|
| $\mu_1 \cup \mu_4$ | $R_1$ | john | J@ed.ex |
| $\mu_3 \cup \mu_5$ | $R_3$ | ringo | R@ed.ex |
| $\mu_2$ | $R_2$ | paul | |

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

| | $?X$ | $?N$ |
|---|---|---|
| $\mu_2$ | $R_2$ | paul |

# SPARQL 1.1
## (and some research issues)

# 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
- An operator (SERVICE) to distribute the execution of a query

Also in this version: Nesting of SELECT expressions, aggregates and some forms of negation (NOT EXISTS, MINUS)

# To remember: Syntax of RDFS

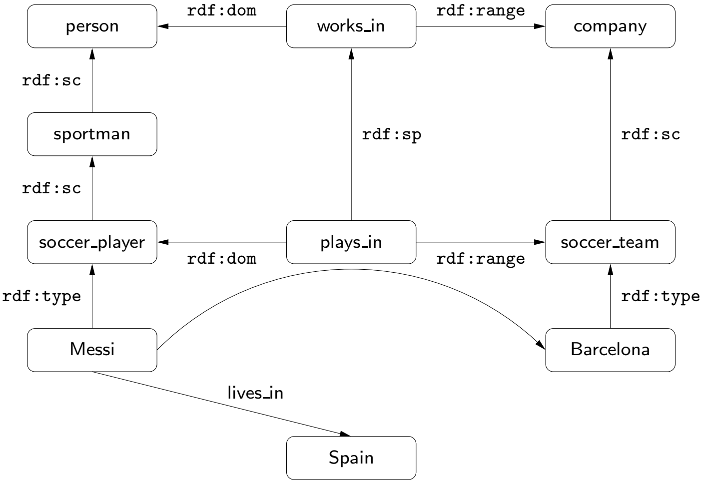
RDFS extends RDF with a schema vocabulary: subPropertyOf (`rdf:sp`), subClassOf (`rdf:sc`), domain (`rdf:dom`), range (`rdf:range`), type (`rdf:type`).

# To remember: 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 do we evaluate a query over RDFS data?

# 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 as for first-order logic.

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

# An inference system for RDFS

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

**Theorem (H03,MPG09,GHM11)**

*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 $G$ ($G \models t$) iff $t$ can be deduced from $G$ by applying the inference rules a finite number of times.

# An entailment regime for RDFS in SPARQL 1.1

Basic graph patterns are evaluated by considering RDFS entailment.

**Definition**

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

- $\text{dom}(\mu) = \text{var}(P)$
- For every $t \in P$: $G \models \mu(t)$

# An entailment regime for RDFS in SPARQL 1.1

Basic graph patterns are evaluated by considering RDFS entailment.

**Definition**

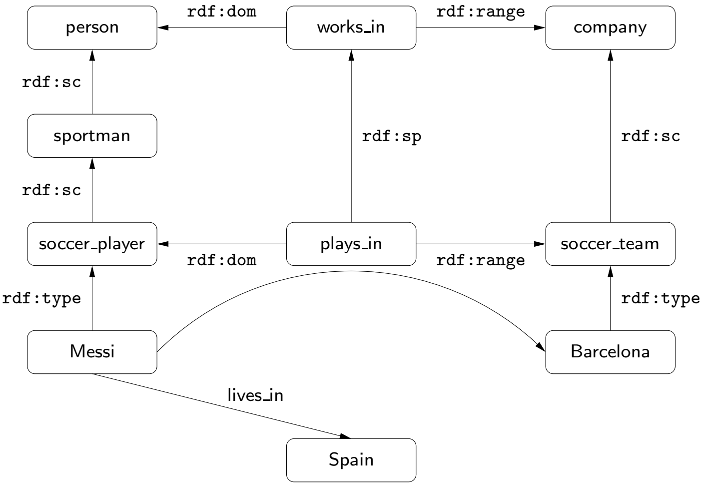
The evaluation of a bgp $P$ over an RDF graph $G$, denoted by $[\![P]\!]_G$, is the set of mappings $\mu$:

- $\text{dom}(\mu) = \text{var}(P)$
- For every $t \in P$: $G \models \mu(t)$

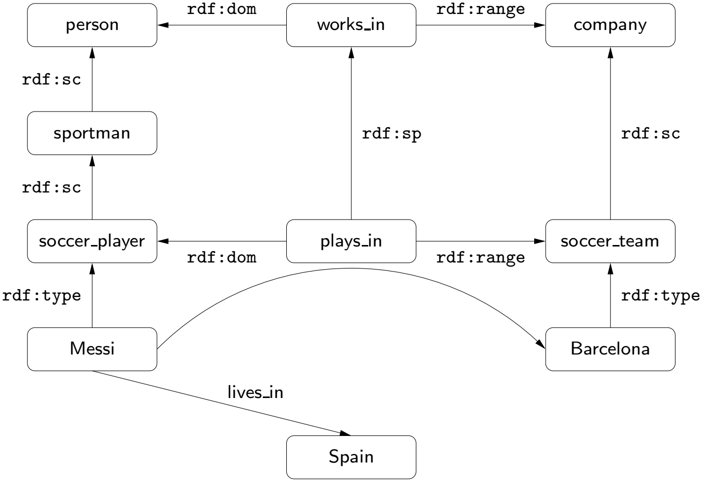
The semantics of AND, UNION, OPT, FILTER and SELECT are defined as before.

- RDFS entailment is only used at the level of bgps

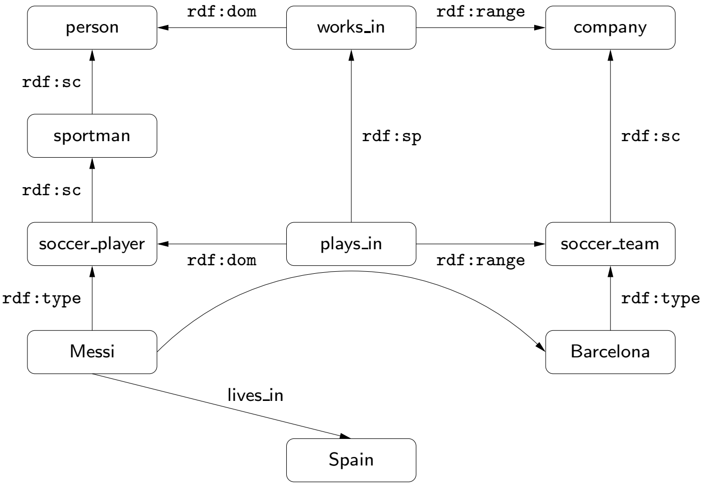
# Example 1: What is the answer to $(?X$, rdf:type, person$)$?



# Example 2: What is the answer to (Messi, `rdf:type`, person)?



# Example 3: What is the answer to $\{(\text{Messi}, \texttt{rdf:type}, ?Y), (?Y, \texttt{rdf:sc}, \text{person})\}$?



# Entailment regimes in SPARQL 1.1: Some observations

SPARQL 1.1 can be used to query not only data but also schema information

- For example: $(?X, \texttt{rdf:sc}, \text{person})$

## Entailment regimes in SPARQL 1.1: Some observations (cont'd)

Web Ontology Language (OWL): A more general ontology language for the Semantic Web

- Users can define their own axioms

  For example: every Chilean has a RUT number

# Entailment regimes in SPARQL 1.1: Some observations (cont'd)

Web Ontology Language (OWL): A more general ontology language for the Semantic Web

- Users can define their own axioms

  For example: every Chilean has a RUT number

Basic graph patterns can also be evaluated by considering OWL entailment.

- $G \models \mu(t)$ has to be defined according to the semantics of OWL

# Entailment regimes in SPARQL 1.1: Some observations (cont'd)

- What are the consequences of considering entailment only at the level bgps?

## Example

Let $G$ be a graph consisting of $(\text{john}, \texttt{rdf:type}, \text{student})$ together with:

$$\left.\begin{array}{l}(\text{student}, \texttt{rdf:sc}, u)\\(u, \text{owl:union}, l)\\(l, \text{rdf:first}, \text{undergrad})\\(l, \text{rdf:rest}, r)\\(r, \text{rdf:first}, \text{grad})\\(r, \text{rdf:rest}, \text{rdf:nil})\end{array}\right\} \text{axiom } \text{student} \sqsubseteq (\text{undergrad} \sqcup \text{grad})$$

# Entailment regimes in SPARQL 1.1: Some observations (cont'd)

- What are the consequences of considering entailment only at the level bgps?

## Example

Let $G$ be a graph consisting of $(\text{john}, \texttt{rdf:type}, \text{student})$ together with:

$$\left.\begin{array}{l} (\text{student}, \texttt{rdf:sc}, u) \\ (u, \text{owl:union}, l) \\ (l, \text{rdf:first}, \text{undergrad}) \\ (l, \text{rdf:rest}, r) \\ (r, \text{rdf:first}, \text{grad}) \\ (r, \text{rdf:rest}, \text{rdf:nil}) \end{array}\right\} \text{axiom } \text{student} \sqsubseteq (\text{undergrad} \sqcup \text{grad})$$

What should be the answer to
$P = ((?X, \texttt{rdf:type}, \text{undergrad}) \text{ UNION } (?X, \texttt{rdf:type}, \text{grad}))$?

# Entailment regimes in SPARQL 1.1: Some observations (cont'd)

- What are the consequences of considering entailment only at the level bgps?

## Example

Let $G$ be a graph consisting of $(\text{john}, \texttt{rdf:type}, \text{student})$ together with:

$$\left.\begin{array}{l}(\text{student}, \texttt{rdf:sc}, u)\\ (u, \text{owl:union}, l)\\ (l, \text{rdf:first}, \text{undergrad})\\ (l, \text{rdf:rest}, r)\\ (r, \text{rdf:first}, \text{grad})\\ (r, \text{rdf:rest}, \text{rdf:nil})\end{array}\right\} \text{axiom student} \sqsubseteq (\text{undergrad} \sqcup \text{grad})$$

What should be the answer to
$P = ((?X, \texttt{rdf:type}, \text{undergrad}) \text{ UNION } (?X, \texttt{rdf:type}, \text{grad}))$?

- Under the current semantics: $[\![P]\!]_G = \emptyset$

# Entailment regimes in SPARQL 1.1: Some observations (cont'd)

It is possible to define a certain-answers semantics for SPARQL 1.1.

- Previous example shows that this semantics does not coincide with the official semantics of SPARQL 1.1

# Entailment regimes in SPARQL 1.1: Some observations (cont'd)

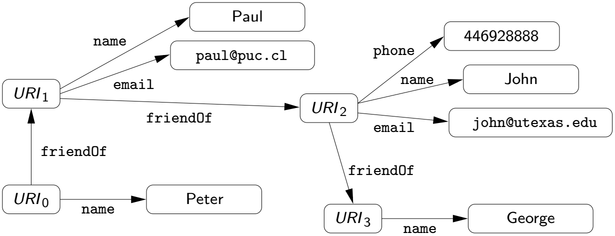
It is possible to define a certain-answers semantics for SPARQL 1.1.

- Previous example shows that this semantics does not coincide with the official semantics of SPARQL 1.1

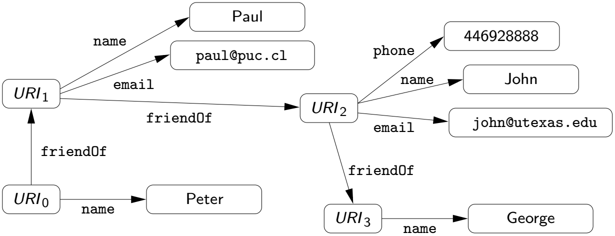
## Open issues

- How natural is the semantics of SPARQL 1.1? Is it a good semantics? Why?
- Under which (natural) restrictions these two semantics coincide?

# SPARQL provides limited navigational capabilities

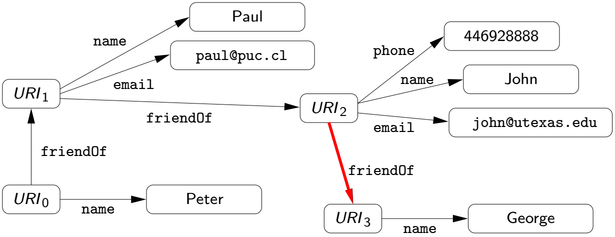


# SPARQL provides limited navigational capabilities



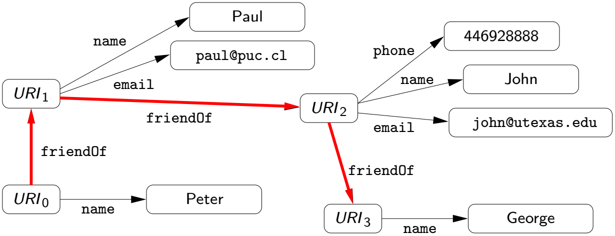

(SELECT ?$X$ ((?$X$, friendOf, ?$Y$) AND (?$Y$, name, George)))

# SPARQL provides limited navigational capabilities



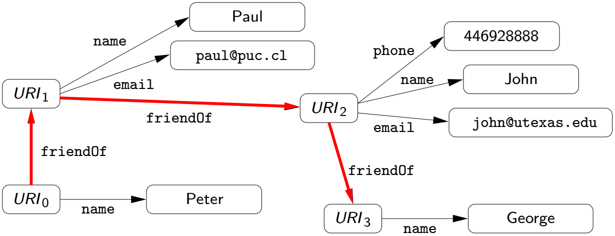

(SELECT ?$X$ ((?$X$, friendOf, ?$Y$) AND (?$Y$, name, George)))

# SPARQL provides limited navigational capabilities

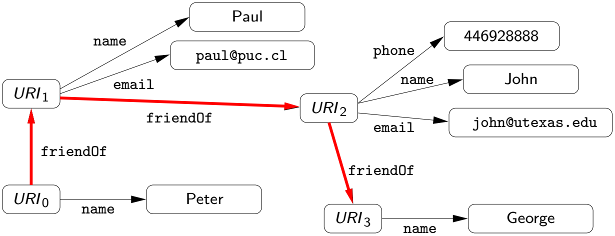



(SELECT $?X$ (($?X$, friendOf, $?Y$) AND ($?Y$, name, George)))

# 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 $a \in U$

# 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 |
| $!(a_1 \mid \ldots \mid a_n)$ | : | 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:

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

# 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:

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

# 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

- $(?X, (\text{friendOf})^*, ?Y)$: Verifies 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**

- $(?X, (\text{friendOf})^*, ?Y)$: Verifies whether there exists a path of friends of arbitrary length from $?X$ to $?Y$
- $(?X, (\texttt{rdf:sc})^*, \text{person})$: Verifies 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

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

# Semantics of property paths

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

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

# Semantics of property paths

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

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

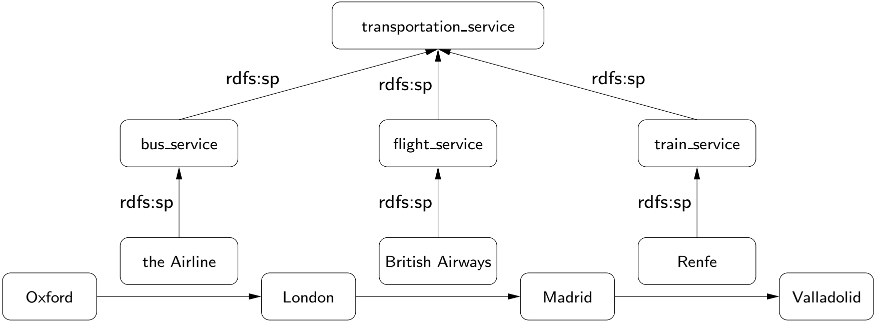
Other cases are defined analogously.

**Example**

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

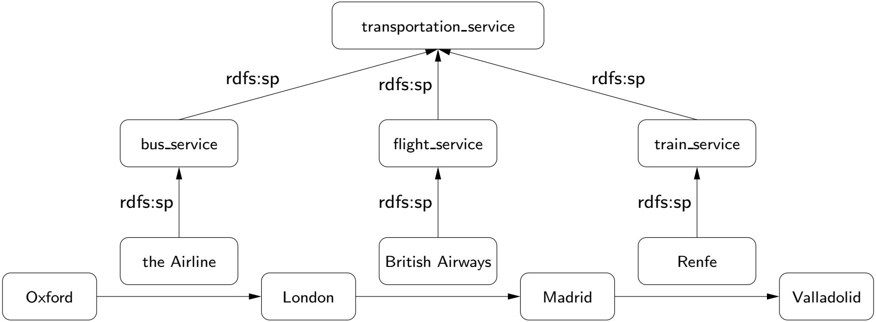
# SPARQL 1.1: Entailment regimes and property paths

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



# SPARQL 1.1: Entailment regimes and property paths

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




In SPARQL 1.1: $(?X, \text{transportation\_service}^*, ?Y)$

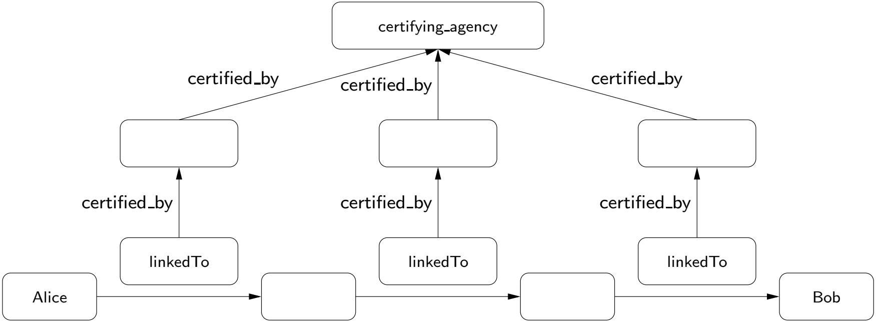
# Navigational capabilities in SPARQL 1.1: Some observations

Previous query can be expressed in SPARQL 1.1 as the intermediate form of navigation involves RDFS vocabulary.

# Navigational capabilities in SPARQL 1.1: Some observations

Previous query can be expressed in SPARQL 1.1 as the intermediate form of navigation involves RDFS vocabulary.

Not expressible: List pairs *a*, *b* of persons that are connected through a path of nodes certified by certifying_agency [RK13]:



# Navigational capabilities in SPARQL 1.1: Some observations (cont'd)

- Some proposals solve the aforementioned issues: nSPARQL [PAG10], nested monadically defined queries [RK13], triple algebra [LRV13]
  - RDFS entailment can be handled in these proposals by using navigational capabilities

# Navigational capabilities in SPARQL 1.1: Some observations (cont'd)

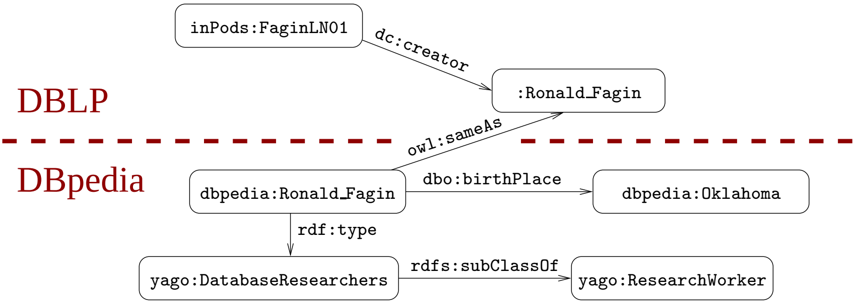
- Some proposals solve the aforementioned issues: nSPARQL [PAG10], nested monadically defined queries [RK13], triple algebra [LRV13]
  - RDFS entailment can be handled in these proposals by using navigational capabilities

## Open issues

- How can OWL entailment be handled in these proposals?
- What navigational capabilities should be added to SPARQL 1.1?
- There is a need for query languages that can return paths

# RFD graphs can be interconnected

```
        : <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/>
```



# 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 .
  SERVICE <http://dbpedia.org/sparql> {
    ?Person     owl:sameAs     ?Author .
    ?Person     dbo:birthPlace dbpedia:Oklahoma . }
}
```

# Federation in SPARQL 1.1

New rule to generate graph patterns:

- If $P$ is a graph pattern and $c \in (U \cup V)$, then (SERVICE $c$ $P$) is a graph pattern.

# Federation in SPARQL 1.1

New rule to generate graph patterns:

- If $P$ is a graph pattern and $c \in (U \cup V)$, then $(\text{SERVICE}\ \ c\ P)$ is a graph pattern.

We will define the semantics of this new operator.

- This corresponds with the official semantics for $(\text{SERVICE}\ \ c\ P)$ with $c \in U$
- $(\text{SERVICE}\ ?X\ \ P)$ is allowed in the official specification of SPARQL 1.1, but its semantics is not defined

# Semantics of SERVICE

ep($\cdot$): Partial function from $U$ to the set of all RDF graphs

- If $c \in \mathrm{dom}(\mathrm{ep})$, then $\mathrm{ep}(c)$ is the RDF graph associated with the endpoint accessible via $c$

# Semantics of SERVICE

ep(·): Partial function from $U$ to the set of all RDF graphs

- If $c \in \text{dom}(\text{ep})$, then $\text{ep}(c)$ is the RDF graph associated with the endpoint accessible via $c$

## Definition [BACP13]

The evaluation of $P = (\text{SERVICE} \;\; c \;\; P_1)$ over an RDF graph $G$ is defined as:

- if $c \in \text{dom}(\text{ep})$, then $[\![P]\!]_G = [\![P_1]\!]_{\text{ep}(c)}$
- if $c \in U \setminus \text{dom}(\text{ep})$, then $[\![P]\!]_G = \{\mu_\emptyset\}$ (where $\mu_\emptyset$ is the mapping with empty domain)
- if $c \in V$, then

$$[\![P]\!]_G \quad = \quad \bigcup_{a \in \text{dom}(\text{ep})} \left( [\![P_1]\!]_{\text{ep}(a)} \bowtie \{c \to a\} \right),$$

# Are variables useful in SERVICE queries?

Consider the query:

$$(?X, \text{service\_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y \ (?N, \text{email}, ?E))$$

# Are variables useful in SERVICE queries?

Consider the query:

$$(?X, \text{service\_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y\ (?N, \text{email}, ?E))$$

There is a simple strategy to compute the answer to this query.

- Can this strategy be generalized?

# How can we evaluate SERVICE queries?

We need some notion of boundedness

- A variable $?X$ is bound in a graph pattern $P$ if for every RDF graph $G$ and every $\mu \in [\![P]\!]_G$, it holds that $?X \in \text{dom}(\mu)$ and $\mu(?X) \in \mathbf{U}$

First attempt: Graph pattern $P$ can be evaluated if for every sub-pattern (SERVICE $?X$ $P_1$) of $P$, it holds that $?X$ is bound in $P$

- $?Y$ is bound in $(?X, \text{service\_address}, ?Y)$ AND (SERVICE $?Y$ $(?N, \text{email}, ?E)$)

# The first attempt: Too restrictive

Consider the query:

$$(?X, \text{service\_description}, ?Z) \text{ UNION}$$
$$\Big((?X, \text{service\_address}, ?Y) \text{ AND } (\text{SERVICE } ?Y\ (?N, \text{email}, ?E))\Big)$$

$?Y$ is not bound in this query, but there is a simple strategy to evaluate it.

# The first attempt: Not appropriate for nested SERVICE operators

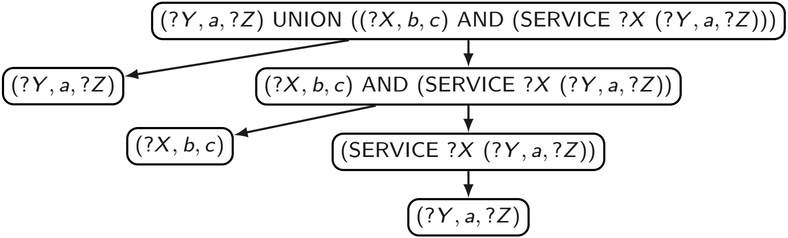
Consider the query:

$$
\begin{aligned}
&(?U_1, \text{related\_with}, ?U_2) \ \ \text{AND} \\
&\qquad\qquad \Big[\text{SERVICE } ?U_1 \ \Big((?N, \text{email}, ?E) \ \ \text{OPT} \\
&\qquad\qquad\qquad\qquad (\text{SERVICE } ?U_2 \ (?N, \text{phone}, ?F))\Big)\Big]
\end{aligned}
$$

# Solving the problems . . .

Notation: $\mathcal{T}(P)$ is the *parse* tree of $P$, in which every node corresponds to a sub-pattern of $P$

Parse tree of $(?Y, a, ?Z)$ UNION $((?X, b, c)$ AND (SERVICE $?X$ $(?Y, a, ?Z)))$:



# A more appropriate notion of boundedness

## Definition [BACP13]

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

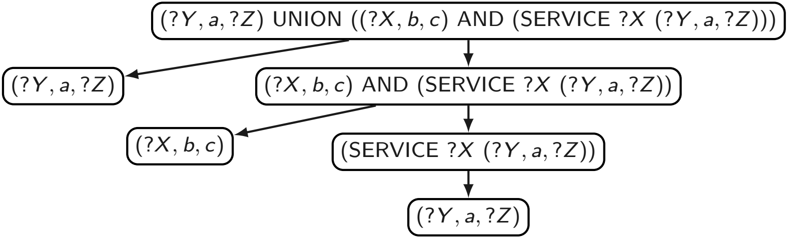
# A more appropriate notion of boundedness

**Definition [BACP13]**

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:



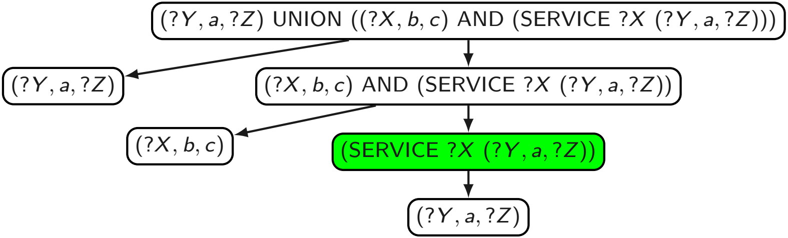
# A more appropriate notion of boundedness

**Definition [BACP13]**

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:



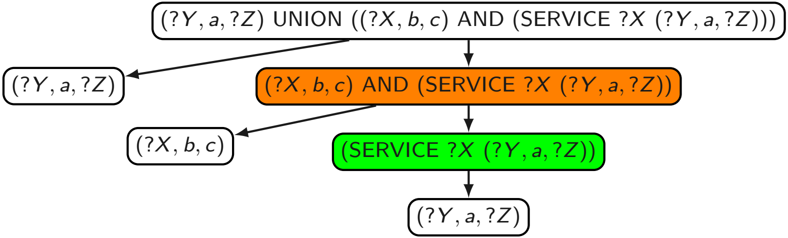
# A more appropriate notion of boundedness

**Definition [BACP13]**

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:



# A more appropriate notion of boundedness

**Definition [BACP13]**

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:

$$\boxed{(?Y, a, ?Z)}$$

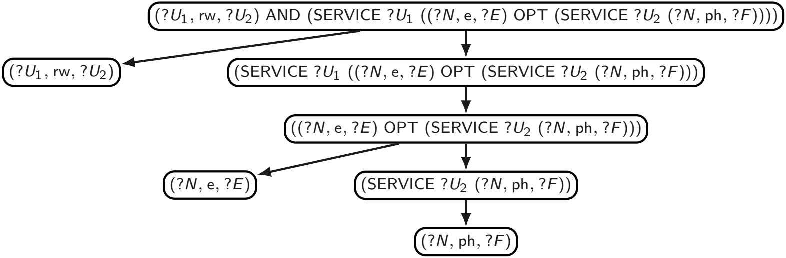
# A more appropriate notion of boundedness

**Definition [BACP13]**

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:



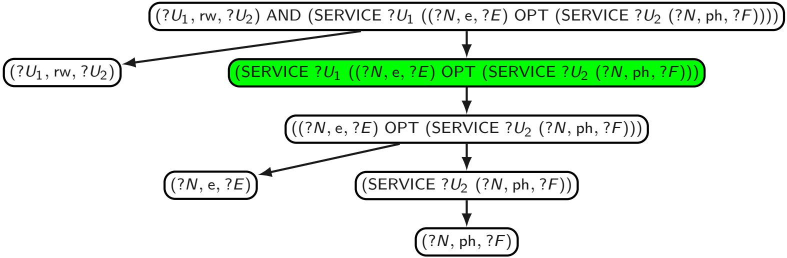
# A more appropriate notion of boundedness

## Definition [BACP13]

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:



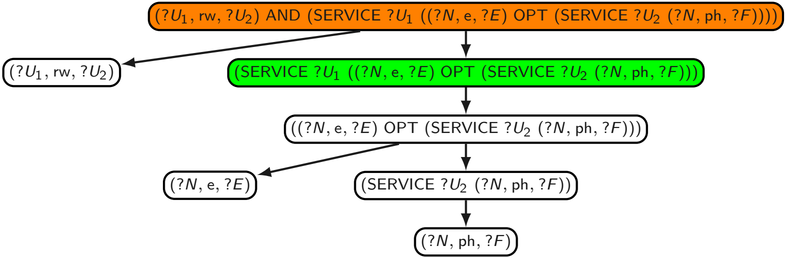
# A more appropriate notion of boundedness

**Definition [BACP13]**

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:



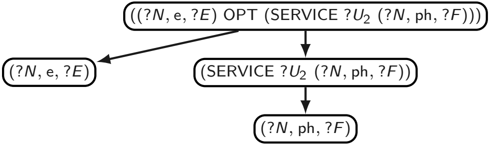
# A more appropriate notion of boundedness

**Definition [BACP13]**

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:



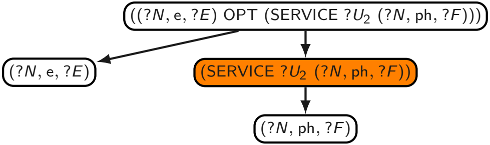
# A more appropriate notion of boundedness

**Definition [BACP13]**

A graph pattern $P$ is service-bound if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:



# A more appropriate notion of boundedness (cont'd)

But we still have a problem:

**Proposition (BACP13)**

*The problem of verifying, given a graph pattern $P$, whether $P$ is service-bound is undecidable.*

We consider a (syntactic) sufficient condition for service-boundedness.

# An appropriate notion: Service-safeness

The set of strongly bound variables in $P$, denoted by $\mathrm{SB}(P)$, is recursively defined as follows:

- if $P$ is a bgp, then $\mathrm{SB}(P) = \mathrm{var}(P)$
- if $P = (P_1 \text{ AND } P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cup \mathrm{SB}(P_2)$
- if $P = (P_1 \text{ UNION } P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1) \cap \mathrm{SB}(P_2)$
- if $P = (P_1 \text{ OPT } P_2)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1)$
- if $P = (P_1 \text{ FILTER } R)$, then $\mathrm{SB}(P) = \mathrm{SB}(P_1)$
- if $P = (\text{SERVICE } c\ P_1)$, then $\mathrm{SB}(P) = \emptyset$

# An appropriate notion: Service-safeness (cont'd)

**Definition [BACP13]**

A graph pattern $P$ is service-safe if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X \in \mathrm{SB}(P_2)$
- $P_1$ is service-safe

If $P$ is service-safe, then there is a strategy to evaluate $P$ without considering all possible SPARQL endpoints.

# An appropriate notion: Service-safeness (cont'd)

**Definition [BACP13]**

A graph pattern $P$ is service-safe if for every node $u$ of $\mathcal{T}(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

- there exists a node $v$ of $\mathcal{T}(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $\mathcal{T}(P)$ and $?X \in \mathrm{SB}(P_2)$
- $P_1$ is service-safe

If $P$ is service-safe, then there is a strategy to evaluate $P$ without considering all possible SPARQL endpoints.

**Open issue**

Is service-safeness the right condition to ensure that a query containing the SERVICE operator can be executed? Why?

# Take-home message

- RDF is the framework proposed by the W3C to represent information in the Web
- SPARQL is the W3C recommendation query language for RDF (January 2008)
- SPARLQ 1.1 is the new version of SPARQL (March 2013)
- SPARQL 1.1 includes some interesting and useful new features
  - Entailment regimes for RDFS and OWL, navigational capabilities and an operator to distribute the execution of a query
  - There are some interesting open issues about these features

# Thank you!

# Bibliography

[BACP13] C. Buil-Aranda, M. Arenas, O. Corcho, A. Polleres: Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. J. Web Sem. 18(1): 1-17 (2013)

[GHM11] C. Gutierrez, C. A. Hurtado, A. O. Mendelzon, J. Pérez: Foundations of Semantic Web databases. J. Comput. Syst. Sci. 77(3): 520-541 (2011)

[H04] P. Hayes: RDF Semantics. W3C Recommendation 10 February 2004

# Bibliography (cont'd)


[LRV13] L. Libkin, J. L. Reutter, D. Vrgoc: Trial for RDF: adapting graph query languages for RDF data. PODS 2013: 201-212

[MPG09] S. Muñoz, J. Pérez, C. Gutierrez: Simple and Efficient Minimal RDFS. J. Web Sem. 7(3): 220-234 (2009)

[PAG10] J. Pérez, M. Arenas, C. Gutierrez: nSPARQL: A navigational language for RDF. J. Web Sem. 8(4): 255-270 (2010)

[RK13] S. Rudolph, M. Krötzsch: Flag & check: data access with monadically defined queries. PODS 2013: 151-162