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
<table>
<thead>
<tr>
<th>Relational</th>
<th>Semantic Web</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tables</td>
<td>SQL</td>
</tr>
<tr>
<td>Relational</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Tables</td>
<td>RDF Graphs</td>
</tr>
<tr>
<td>SQL</td>
<td></td>
</tr>
<tr>
<td>Relational</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>------------</td>
<td>--------------</td>
</tr>
<tr>
<td>Tables</td>
<td>RDF Graphs</td>
</tr>
<tr>
<td>SQL</td>
<td>SPARQL</td>
</tr>
<tr>
<td>Relational</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>Tables</td>
<td>RDF Graphs</td>
</tr>
<tr>
<td>SQL</td>
<td>SPARQL</td>
</tr>
<tr>
<td>Closed Data</td>
<td>(inside an organization)</td>
</tr>
<tr>
<td>Relational</td>
<td>Semantic Web</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------------</td>
</tr>
<tr>
<td>Tables</td>
<td><em>RDF Graphs</em></td>
</tr>
<tr>
<td>SQL</td>
<td><em>SPARQL</em></td>
</tr>
<tr>
<td>Closed Data</td>
<td><em>Open Data</em></td>
</tr>
<tr>
<td>(inside an organization)</td>
<td>(available on the Web)</td>
</tr>
</tbody>
</table>
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

\[ U = \text{set of Urils} \]
\[ B = \text{set of Blank nodes} \]
\[ L = \text{set of Literals} \]
RDF formal model

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

\[U = \text{set of UrIs}\]

\[B = \text{set of Blank nodes}\]

\[L = \text{set of Literals}\]
RDF formal model

\[(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)\] 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/>

```
conf:pods

swrc:series

inPods:2001

dct:PartOf

"Optimal Aggregation ..."

dc:title

dc:creator :Amnon_Lotem

dc:creator :Moni_Naor

dc:creator :Ronald_Fagin
```
An example of a URI

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

<table>
<thead>
<tr>
<th>Property</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdfs:label</td>
<td>PODS (xsd:string)</td>
</tr>
<tr>
<td>rdfs:seeAlso</td>
<td><a href="http://dblp.l3s.de/Venues/PODS">http://dblp.l3s.de/Venues/PODS</a></td>
</tr>
<tr>
<td>is swrc:series of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/pods/00">http://dblp.l3s.de/d2r/resource/publications/conf/pods/00</a></td>
</tr>
<tr>
<td>is swrc:series of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/pods/2001">http://dblp.l3s.de/d2r/resource/publications/conf/pods/2001</a></td>
</tr>
<tr>
<td>is swrc:series of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/pods/2003">http://dblp.l3s.de/d2r/resource/publications/conf/pods/2003</a></td>
</tr>
<tr>
<td>is swrc:series of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/pods/2004">http://dblp.l3s.de/d2r/resource/publications/conf/pods/2004</a></td>
</tr>
<tr>
<td>is swrc:series of</td>
<td><a href="http://dblp.l3s.de/d2r/resource/publications/conf/pods/2005">http://dblp.l3s.de/d2r/resource/publications/conf/pods/2005</a></td>
</tr>
</tbody>
</table>
URI can be used for any abstract resource

http://dblp.l3s.de/d2r/page/authors/Ronald_Fagin
RDF: Another example

- person
  - works_in
    - company
  - sportman
    - soccer_player
      - plays_in
        - soccer_team
          - Barcelona
            - Messi
              - plays_in
                - Barcelona
              - address
              - lives_in
                - Spain
                - country

- address
- lives_in
- country
Some peculiarities of the RDF data model

- *Existential variables* as datavals (null values)
- Built-in vocabulary with fixed semantics (RDFS)
- Graph model where nodes may also be edge labels
Previous example: A better representation

- person
  - sportman
  - soccer_player
    - Messi
      - address
      -_:b
  - rdf:sp
  - plays_in
    - soccer_team
      - Barcelona
      - rdf:type
  - rdf:dom
  - works_in
    - company
      - rdf:sp
      - rdf:sc
    - rdf:range
      - Barcelona
      - rdf:typerdf:type
      - Spain
Previous example: A better representation
Previous example: A better representation

```
person rdf:dom works_in rdf:range company

sportman rdf:sp

soccer_player rdf:dom plays_in rdf:range soccer_team

Messi rdf:type

Spain rdf:type Spain

_:b rdf:domain

Messi rdf:range Barcelona

Messi rdf:range Spain

address

lives_in

country
```
RDF + RDFS

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

plus \textit{semantics} for this vocabulary
RDFS: Messi is a Person

- Messi is a Person
- Messi works in a company
- Messi plays in a soccer team
- Messi lives in Spain

-diagram-
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 $G$ ($\text{cl}(G)$) is the graph obtained by adding to $G$ all the triples that are implied by $G$.

A basic property of the closure:

- $G$ implies $t$ iff $t \in \text{cl}(G)$
Example: \((\text{Messi}, \text{rdf:type}, \text{person})\) over the closure
Does the blank node add some information?
What about now?

- Messi
  - lives_in: Spain
  - works_in: Barcelona
  - soccer_player
    - rdf:sc
      - rdf:sp: soccer_team
        - rdf:range: rdfs:domain person
          - rdf:range: rdfs:range company
            - rdf:sc
              - rdf:sp: works_in
                - rdf:range: rdfs:domain person
                  - rdf:range: rdfs:range company
                    - rdf:sc
                      - rdf:sp: plays_in
                        - rdf:range: rdfs:domain soccer_player
                          - rdf:range: rdfs:range soccer_team
                            - rdf:sc
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, .....
Example: Authors that have published in ISWC
SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

SELECT ?Author
Example: Authors that have published in ISWC

```
SELECT ?Author
WHERE
{
}
```
Example: Authors that have published in ISWC

```
SELECT ?Author
WHERE
{
}
```
Example: Authors that have published in ISWC

SELECT ?Author
WHERE
{
}
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
{
}
```

A SPARQL query consists of a:
Example: Authors that have published in ISWC

```
SELECT ?Author
WHERE
{
}
```

A SPARQL query consists of a:

**Head:** Processing of the variables
SPARQL: A Simple RDF Query Language

Example: Authors that have published in ISWC

```sparql
SELECT ?Author
WHERE
{
}
```

A SPARQL query consists of a:

- **Head**: Processing of the variables
- **Body**: Pattern matching expression
Example: Authors that have published in ISWC, and their Web pages if this information is available:

```
SELECT ?Author ?WebPage
WHERE
{

    OPTIONAL {
    }
}
```
Example: Authors that have published in ISWC, and their Web pages if this information is available:

```sparql
SELECT ?Author ?WebPage
WHERE
{

  OPTIONAL {
  }
}
```
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

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

```sql
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
- ...
$\mathcal{V}$: set of variables

Each variable is assumed to start with ?
$V$: 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: ($?X$, name, john), ($?X$, name, $?Y$)
**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, name, john), (?X, name, ?Y)

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

Examples: {(?X, knows, ?Y), (?Y, name, john)}
Recursive definition of SPARQL graph patterns:

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

SPARQL query:

- If $P$ is a graph pattern and $W$ is a finite set of variables, then $(\text{SELECT } W P)$ is a SPARQL query
Standard versus algebraic notation

?X : name "john"

(?X, name, john)
Standard versus algebraic notation

?X : name "john"

{ P1 . P2 }

(?X, name, john)

(P1 AND P2)
Standard versus algebraic notation

?X : name "john"

{ P1 . P2 }

{ P1 OPTIONAL { P2 } }

(?X, name, john)

( P1 AND P2 )

( P1 OPT P2 )
Standard versus algebraic notation

?X : name "john"

\{ P1 . P2 \}

\{ P1 OPTIONAL \{ P2 \}\}

\{ P1 \} UNION \{ P2 \}

(?X, name, john)

(P₁ AND P₂)

(P₁ OPT P₂)

(P₁ UNION P₂)
Standard versus algebraic notation

\(?X \colon \text{name }{ "john"} \)

\(\{ P_1, P_2 \} \)

\(\{ P_1 \text{ OPTIONAL } \{ P_2 \} \} \)

\(\{ P_1 \text{ UNION } \{ P_2 \} \} \)

\(\{ P_1 \text{ FILTER } (R) \} \)

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

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

\((P_1 \text{ OPT } P_2)\)

\((P_1 \text{ UNION } P_2)\)

\((P_1 \text{ FILTER } R)\)
<table>
<thead>
<tr>
<th>Standard notation</th>
<th>Algebraic notation</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>?X :name &quot;john&quot;</code></td>
<td><code>(?X, name, john)</code></td>
</tr>
<tr>
<td><code>{ P1 . P2 }</code></td>
<td><code>(P₁ AND P₂)</code></td>
</tr>
<tr>
<td><code>{ P₁ OPTIONAL { P₂ } }</code></td>
<td><code>(P₁ OPT P₂)</code></td>
</tr>
<tr>
<td><code>{ P₁ } UNION { P₂ }</code></td>
<td><code>(P₁ UNION P₂)</code></td>
</tr>
<tr>
<td><code>{ P₁ FILTER ( R ) }</code></td>
<td><code>(P₁ FILTER R)</code></td>
</tr>
<tr>
<td><code>SELECT W WHERE { P }</code></td>
<td><code>(SELECT W P)</code></td>
</tr>
</tbody>
</table>
Mappings: building block for the semantics

**Definition**

A mapping is a partial function:

\[ \mu : V \rightarrow (U \cup L \cup B) \]
Mappings: building block for the semantics

**Definition**
A mapping is a partial function:

\[ \mu : V \rightarrow (U \cup L \cup B) \]

Given a mapping \( \mu \) and a triple pattern \( t \):
Mappings: building block for the semantics

Definition
A mapping is a partial function:

$$\mu : V \rightarrow (U \cup L \cup B)$$

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:

\[ \mu : V \rightarrow (U \cup L \cup B) \]

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:

\[ \mu : V \rightarrow (U \cup L \cup B) \]

Given a mapping \( \mu \) and a triple pattern \( t \):

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

Example

\[ \mu = \{ ?X \rightarrow R_1, ?Y \rightarrow R_2, ?Z \rightarrow john \} \]
Mappings: building block for the semantics

Definition
A mapping is a partial function:

\[ \mu : V \rightarrow (U \cup L \cup B) \]

Given a mapping \( \mu \) and a triple pattern \( t \):

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

Example

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

\[ t = (?X, \text{name}, ?Z) \]
Definition

A mapping is a partial function:

\[ \mu : V \rightarrow (U \cup L \cup B) \]

Given a mapping \( \mu \) and a triple pattern \( t \):

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

Example

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

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

\[ \mu(t) = (R_1, \text{name}, \text{john}) \]
The semantics of triple patterns

Definition

The evaluation of triple pattern $t$ over a graph $G$, denoted by $\llbracket t \rrbracket_G$, 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 $\llbracket t \rrbracket_G$, 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:

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

\[ G \]

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

\[ \llbracket (?X, \text{name}, ?N) \rrbracket_G \]
Example

$G$

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

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

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

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

$$\mu_1 = \{?X \rightarrow R_1, ?N \rightarrow \text{john}\}$$

$$\mu_2 = \{?X \rightarrow R_2, ?N \rightarrow \text{paul}\}$$
Example

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

\[ \llbracket (?X, \text{name, ?N}) \rrbracket_G \]
\[ \left\{ \begin{array}{l}
\mu_1 = \{ ?X \rightarrow R_1, ?N \rightarrow \text{john}\} \\
\mu_2 = \{ ?X \rightarrow R_2, ?N \rightarrow \text{paul}\}
\end{array} \right\} \]

\[ \llbracket (?X, \text{email, ?E}) \rrbracket_G \]
Example

\[
G
\]

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

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

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

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

\[
\begin{align*}
\mu_1 &= \{ ?X \to R_1, ?N \to \text{john} \} \\
\mu_2 &= \{ ?X \to R_2, ?N \to \text{paul} \}
\end{align*}
\]

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

\[
\begin{align*}
\mu &= \{ ?X \to R_1, ?E \to \text{J@ed.ex} \}
\end{align*}
\]
Example

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

\[
\begin{array}{cc}
?X & ?N \\
\mu_1 & R_1 \\
\mu_2 & R_2 \\
\end{array}
\begin{array}{c}
\text{john} \\
\text{paul} \\
\end{array}
\]

\[
\begin{array}{cc}
?X & ?E \\
\mu & R_1 \\
\end{array}
\begin{array}{c}
\text{J@ed.ex} \\
\end{array}
\]
Example

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

\[\llbracket (R_1, \text{webPage}, ?W) \rrbracket_G\]
\[\llbracket (R_3, \text{name}, \text{ringo}) \rrbracket_G\]
\[\llbracket (R_2, \text{name}, \text{paul}) \rrbracket_G\]
Example

\[ G \]

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

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

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

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

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

\[ [(R_1, \text{webPage, ?W})]_G \]
\[ \{ \} \]
\[ [(R_2, \text{name, paul})]_G \]

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

\[ G \]

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

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

\[ [(R_2, \text{name, paul})]_G \]
\[
\{ \mu_\emptyset = \{ \} \} \]

\[ [(R_3, \text{name, ringo})]_G \]
\[
\{ \} \]
Semantics of SPARQL: Basic graph patterns

Let $P$ be a basic graph pattern

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

$$\mu(P) = \{ \mu(t) \mid t \in P \}$$
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 $\mu$ such that $\text{var}(P) \subseteq \text{dom}(\mu)$:

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

Definition

The evaluation of $P$ over an RDF graph $G$, denoted by $\llbracket P \rrbracket_G$, is the set of mappings $\mu$:
- $\text{dom}(\mu) = \text{var}(P)$
- $\mu(P) \subseteq G$
Semantics of basic graph patterns: An example

<table>
<thead>
<tr>
<th>graph</th>
<th>bgp</th>
<th>evaluation</th>
</tr>
</thead>
</table>
| \((R_1, \text{name, john})\) | \{(?X, \text{name, ?Y}),
         (?X, \text{email, ?Z})\} | |
<p>| ((R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>})) | | |
| ((R_2, \text{name, paul})) | | |</p>
<table>
<thead>
<tr>
<th>graph</th>
<th>bgp</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R_1, \text{name, john})$</td>
<td>$(?X, \text{name, ?Y})$</td>
<td></td>
</tr>
<tr>
<td>$(R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>})$</td>
<td>$(?X, \text{email, ?Z})$</td>
<td></td>
</tr>
<tr>
<td>$(R_2, \text{name, paul})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Semantics of basic graph patterns: An example
Semantics of basic graph patterns: An example

<table>
<thead>
<tr>
<th>graph</th>
<th>bgp</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>((R_1, \text{name, john}))</td>
<td>{(?\text{X}, \text{name, ?Y}), (\text{?X, email, ?Z})}</td>
<td></td>
</tr>
<tr>
<td>((R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>}))</td>
<td></td>
<td></td>
</tr>
<tr>
<td>((R_2, \text{name, paul}))</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Semantics of basic graph patterns: An example

graph
(R₁, name, john)
(R₁, email, J@ed.ex)
(R₂, name, paul)

bgp
{(X, name, Y),
 (X, email, Z)}

μ:
R₁ john J@ed.ex

Σ:
X Y Z
Semantics of basic graph patterns: An example

<table>
<thead>
<tr>
<th>graph</th>
<th>bgp</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>((R_1, \text{name, john}))</td>
<td>{(?X, \text{name, ?Y}),  \nonumber} (R_1\text{john <a href="mailto:J@ed.ex">J@ed.ex</a>})</td>
<td>(\mu:)</td>
</tr>
<tr>
<td>((R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>}))</td>
<td>\nonumber} (R_1\text{john <a href="mailto:J@ed.ex">J@ed.ex</a>})</td>
<td></td>
</tr>
<tr>
<td>((R_2, \text{name, paul}))</td>
<td>\nonumber} (R_1\text{john <a href="mailto:J@ed.ex">J@ed.ex</a>})</td>
<td></td>
</tr>
</tbody>
</table>

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:

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

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

Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$</td>
<td>$R_1$</td>
<td>john</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$</td>
<td>$R_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_3$</td>
<td></td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>R_2</td>
</tr>
</tbody>
</table>
Compatible mappings: mappings that can be merged

**Definition**

Mappings $\mu_1$ and $\mu_2$ are compatible if they agree in their common variables:

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

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>$\text{john}$</td>
<td>$\text{<a href="mailto:J@edu.ex">J@edu.ex</a>}$</td>
<td>$R_2$</td>
</tr>
<tr>
<td>$R_1$</td>
<td>$\text{<a href="mailto:P@edu.ex">P@edu.ex</a>}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$\mu_1: R_1$

$\mu_2: R_1$

$\mu_3: R_2$
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**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$</td>
<td>$R_1$</td>
<td>john</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_3$</td>
<td></td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>R_2</td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_2$</td>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings: mappings that can be merged

**Definition**

Mappings $\mu_1$ and $\mu_2$ are compatible if they agree in their common variables:

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

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$:</td>
<td>$\mu_2$:</td>
<td>$\mu_3$:</td>
<td>$\mu_1 \cup \mu_2$:</td>
</tr>
<tr>
<td>$R_1$:</td>
<td>$R_1$:</td>
<td>$R_1$:</td>
<td>$R_1$:</td>
</tr>
<tr>
<td>john</td>
<td>john</td>
<td>john</td>
<td>john</td>
</tr>
<tr>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
</tr>
<tr>
<td>R2</td>
<td>R2</td>
<td>R2</td>
<td>R2</td>
</tr>
</tbody>
</table>
Compatible mappings: mappings that can be merged

Definition

Mappings $\mu_1$ and $\mu_2$ are compatible if they agree in their common variables:

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

Example

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ :</td>
<td>$R_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td></td>
<td>john</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_2$ :</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>$R_2$</td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_3$ :</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td>$R_2$</td>
</tr>
</tbody>
</table>
Compatible mappings: mappings that can be merged

**Definition**

Mappings $\mu_1$ and $\mu_2$ are compatible if they agree in their common variables:

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

**Example**

<table>
<thead>
<tr>
<th>$\ ?X$</th>
<th>$\ ?Y$</th>
<th>$\ ?Z$</th>
<th>$\ ?V$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td>$R_2$</td>
</tr>
<tr>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></td>
</tr>
</tbody>
</table>

$\mu_1 : R_1 \rightarrow \text{john}$

$\mu_2 : R_1 \rightarrow \text{john}$

$\mu_3 : R_1 \rightarrow \text{john}$

$\mu_1 \cup \mu_2 : R_1 \rightarrow \text{john}$

$\mu_1 \cup \mu_3 : R_1 \rightarrow \text{john}$

$\triangleright \mu_2$ and $\mu_3$ are not compatible
Sets of mappings and operations

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

**Definition**

**Join**: $\Omega_1 \Join \Omega_2$

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

will be used to define AND
Sets of mappings and operations

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

**Definition**

Join: $\Omega_1 \Join \Omega_2$

- $\{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2, \text{ and } \mu_1, \mu_2 \text{ are compatibles}\}$
- 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: $\Omega_1 \setminus \Omega_2$

- $\{\mu \in \Omega_1 \mid \text{for all } \mu' \in \Omega_2, \mu \text{ and } \mu' \text{ are not compatibles}\}$

- mappings in $\Omega_1$ that cannot be extended with mappings in $\Omega_2$
Sets of mappings and operations

Definition

Difference: $\Omega_1 \setminus \Omega_2$

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

Definition

Left outer join: $\Omega_1 \Join \Omega_2 = (\Omega_1 \Join \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$

- 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 $\text{OPT}$
Given an RDF graph $G$

**Definition**

\[
\llbracket (P_1 \text{ AND } P_2) \rrbracket_G = \\
\llbracket (P_1 \text{ UNION } P_2) \rrbracket_G = \\
\llbracket (P_1 \text{ OPT } P_2) \rrbracket_G = \\
\llbracket (\text{SELECT } W P) \rrbracket_G =
\]
Given an RDF graph $G$

**Definition**

- $\llbracket (P_1 \text{ AND } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \times \llbracket P_2 \rrbracket_G$
- $\llbracket (P_1 \text{ UNION } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G$
- $\llbracket (P_1 \text{ OPT } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \vartriangledown \llbracket P_2 \rrbracket_G$
- $\llbracket (\text{SELECT } W \ P) \rrbracket_G = \{ \mu|_W | \mu \in \llbracket P \rrbracket_G \}$
Given an RDF graph $G$

**Definition**

\[
\begin{align*}
\llbracket(P_1 \text{ AND } P_2)\rrbracket_G &= \llbracket P_1 \rrbracket_G \times \llbracket P_2 \rrbracket_G \\
\llbracket(P_1 \text{ UNION } P_2)\rrbracket_G &= \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G \\
\llbracket(P_1 \text{ OPT } P_2)\rrbracket_G &= \llbracket P_1 \rrbracket_G \times \llbracket P_2 \rrbracket_G \\
\llbracket(\text{SELECT } W P)\rrbracket_G &= \{ \mu\upharpoonright_W \mid \mu \in \llbracket P \rrbracket_G \} \\
\text{dom}(\mu\upharpoonright_W) &= \text{dom}(\mu) \cap W \text{ and } \\
\mu\upharpoonright_W(?X) &= \mu(?X) \text{ for every } ?X \in \text{dom}(\mu\upharpoonright_W)
\end{align*}
\]
Example (AND)

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

\[
[(\langle ?X, \text{name, ?N} \rangle \land \langle ?X, \text{email, ?E} \rangle)]_G
\]
Example (AND)

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

\[
\llbracket (\forall X \ (X, \text{name, } ?N)) \land (\forall X \ (X, \text{email, } ?E)) \rrbracket_G
\]

\[
\llbracket (\forall X \ (X, \text{name, } ?N)) \rrbracket_G \land \llbracket (\forall X \ (X, \text{email, } ?E)) \rrbracket_G
\]
Example (AND)

\[ G : (R_1, \text{name}, \text{john}) \quad (R_2, \text{name}, \text{paul}) \quad (R_3, \text{name}, \text{ringo}) \]

\[ (R_1, \text{email}, J@ed.ex) \quad (R_3, \text{email}, R@ed.ex) \quad (R_3, \text{webPage}, \text{www.ringo.com}) \]

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

\[ \llbracket (?X, \text{name}, ?N) \rrbracket_G \Join \llbracket (?X, \text{email}, ?E) \rrbracket_G \]

<table>
<thead>
<tr>
<th>( \mu_1 )</th>
<th>( ?X )</th>
<th>( ?N )</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( R_1 )</td>
<td>\text{john}</td>
</tr>
<tr>
<td>( \mu_2 )</td>
<td>( R_2 )</td>
<td>\text{paul}</td>
</tr>
<tr>
<td>( \mu_3 )</td>
<td>( R_3 )</td>
<td>\text{ringo}</td>
</tr>
</tbody>
</table>
Example (AND)

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

\[
\llbracket ((?X, \text{name, ?N}) \land (?X, \text{email, ?E})) \rrbracket_G \\
\llbracket (?X, \text{name, ?N}) \rrbracket_G \land \llbracket (?X, \text{email, ?E}) \rrbracket_G
\]

<table>
<thead>
<tr>
<th>$\mu_1$</th>
<th>?X</th>
<th>?N</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td></td>
<td>john</td>
</tr>
<tr>
<td>$\mu_2$</td>
<td>$R_2$</td>
<td>paul</td>
</tr>
<tr>
<td>$\mu_3$</td>
<td>$R_3$</td>
<td>ringo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\mu_4$</th>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
<tr>
<td>$\mu_5$</td>
<td>$R_3$</td>
<td><a href="mailto:R@ed.ex">R@ed.ex</a></td>
</tr>
</tbody>
</table>
Example (AND)

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

\[ \llbracket (?X, \text{name}, ?N) \rrbracket_G \Join \llbracket (?X, \text{email}, ?E) \rrbracket_G \]

\[ \begin{array}{|c|c|} \hline ?X & ?N \\ \hline R_1 & \text{john} \\ \hline R_2 & \text{paul} \\ \hline R_3 & \text{ringo} \\ \hline \end{array} \Join \begin{array}{|c|c|} \hline ?X & ?E \\ \hline R_1 & \text{J@ed.ex} \\ \hline R_3 & \text{R@ed.ex} \\ \hline \end{array} \]
Example (AND)

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

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

| \(\mu_1\) | \(\mu_2\) | \(\mu_3\) |
| \(R_1\) | \text{john} | R@ed.ex |
| \(R_2\) | \text{paul} | |
| \(R_3\) | \text{ringo} | |

\(\mu_4\) | \(\mu_5\)
| ?X | ?E |
| \(R_1\) | J@ed.ex |
| \(R_3\) | R@ed.ex |

\(\mu_1 \cup \mu_4\)
| \(R_1\) | \text{john} | J@ed.ex |

\(\mu_3 \cup \mu_5\)
| \(R_3\) | \text{ringo} | R@ed.ex |
Example (OPT)

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

\[ \llbracket (\{ (\text{?X}, \text{name}, \text{?N}) \} \cup \{ (\text{?X}, \text{email}, \text{?E}) \}) \rrbracket_G \]
Example (OPT)

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

\[
\llbracket((?X, \text{name}, ?N) \text{ OPT } (?X, \text{email}, ?E))\rrbracket_G
\]

\[
\llbracket((?X, \text{name, ?N})_G \boxtimes \llbracket((?X, \text{email, ?E})_G
\]

40
Example (OPT)

\[ G : (R_1, \text{name, john}) \quad (R_2, \text{name, paul}) \quad (R_3, \text{name, ringo}) \]

\[ (R_1, \text{email, J@ed.ex}) \quad (R_3, \text{email, R@ed.ex}) \quad (R_3, \text{webPage, www.ringo.com}) \]

\[
\llbracket\left(\left(\forall X, \text{name, } ?N\right) \text{ OPT } \left(\forall X, \text{email, } ?E\right)\right)\rrbracket_G
\]

\[
\llbracket\left(\forall X, \text{name, } ?N\right)\rrbracket_G \mathbin{\otimes} \llbracket\left(\forall X, \text{email, } ?E\right)\rrbracket_G
\]

<table>
<thead>
<tr>
<th>(\mu_1)</th>
<th>(R_1)</th>
<th>john</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu_2)</td>
<td>(R_2)</td>
<td>paul</td>
</tr>
<tr>
<td>(\mu_3)</td>
<td>(R_3)</td>
<td>ringo</td>
</tr>
</tbody>
</table>
Example (OPT)

\( G \) : \( (R_1, \text{name}, \text{john}) \) \( (R_2, \text{name}, \text{paul}) \) \( (R_3, \text{name}, \text{ringo}) \)

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

\[ \llbracket ((?X, \text{name}, ?N) \text{ OPT } (?X, \text{email}, ?E)) \rrbracket_G \]

\[ \llbracket (?X, \text{name}, ?N) \rrbracket_G \boxtimes \llbracket (?X, \text{email}, ?E) \rrbracket_G \]

\( \mu_1 \)
\( \mu_2 \)
\( \mu_3 \)

\( ?X \) \( ?N \)
\( R_1 \) \( \text{john} \)
\( R_2 \) \( \text{paul} \)
\( R_3 \) \( \text{ringo} \)

\( \mu_4 \)
\( \mu_5 \)

\( ?X \) \( ?E \)
\( R_1 \) \( \text{J@ed.ex} \)
\( R_3 \) \( \text{R@ed.ex} \)
Example (OPT)

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

\[
\llbracket((?X, \text{name, ?N}) \text{ OPT } (?X, \text{email, ?E}))\rrbracket_G
\]

\[
\llbracket((?X, \text{name, ?N}))_G \boxtimes \llbracket((?X, \text{email, ?E}))_G
\]

\[
\begin{array}{c|c}
\mu_1 & ?X & ?N \\
\hline
R_1 & \text{john} \\
\mu_2 & R_2 & \text{paul} \\
\mu_3 & R_3 & \text{ringo}
\end{array}
\boxtimes
\begin{array}{c|c}
\mu_4 & ?X & ?E \\
\hline
R_1 & \text{J@ed.ex} \\
\mu_5 & R_3 & \text{R@ed.ex}
\end{array}
\]
Example (OPT)

\[ G : \]

\[ (R_1, \text{name}, \text{john}) \quad (R_2, \text{name}, \text{paul}) \quad (R_3, \text{name}, \text{ringo}) \]

\[ (R_1, \text{email}, \text{J@ed.ex}) \quad (R_3, \text{email}, \text{R@ed.ex}) \quad (R_3, \text{webPage}, \text{www.ringo.com}) \]

\[ \llbracket \((?X, \text{name}, ?N) \text{ OPT } (?X, \text{email}, ?E))\rrbracket_G \]

\[ \llbracket (?X, \text{name}, ?N) \rrbracket_G \boxtimes \llbracket (?X, \text{email}, ?E) \rrbracket_G \]

\[ \begin{array}{c|c|c}
\mu_1 & ?X & ?N \\
R_1 & \text{john} \\
\mu_2 & R_2 & \text{paul} \\
\mu_3 & R_3 & \text{ringo} \\
\end{array} \;
\begin{array}{c|c|c|c}
\mu_4 & ?X & ?E \\
R_1 & \text{J@ed.ex} \\
\mu_5 & R_3 & \text{R@ed.ex} \\
\end{array} \]

\[ \begin{array}{c|c|c|c}
R_1 & \text{john} & \text{J@ed.ex} \\
\mu_1 \cup \mu_4 & R_3 & \text{ringo} & \text{R@ed.ex} \\
\mu_3 \cup \mu_5 & R_2 & \text{paul} \\
\end{array} \]
Example (OPT)

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

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

\[
\begin{array}{|c|c|}
\hline
\mu_1 \quad ?X & ?N \\
\hline
R_1 & \text{john} \\
\hline
R_2 & \text{paul} \\
\hline
R_3 & \text{ringo} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\mu_2 \quad ?X & ?E \\
\hline
R_1 & \text{J@ed.ex} \\
\hline
R_3 & \text{R@ed.ex} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|}
\hline
\mu_1 \cup \mu_4 \quad ?X & ?N & ?E \\
\hline
R_1 & \text{john} & \text{J@ed.ex} \\
\hline
R_3 & \text{ringo} & \text{R@ed.ex} \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|}
\hline
\mu_3 \cup \mu_5 \quad ?X & ?E \\
\hline
R_2 & \text{paul} \\
\hline
\end{array}
\]
Example (UNION)

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

\[ \llbracket (\forall X, \text{email, ?Info}) \cup (\forall X, \text{webPage, ?Info}) \rrbracket_G \]
Example (UNION)

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

\[
\llbracket (\exists X. \text{email} = ?X) \cup (\exists X. \text{webPage} = ?X) \rrbracket_G
\]
Example (UNION)

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

\[
\llbracket ((?X, \text{email, ?Info}) \text{ UNION } (?X, \text{webPage, ?Info})) \rrbracket_G
\]
\[
\llbracket (?X, \text{email, ?Info}) \rrbracket_G \cup \llbracket (?X, \text{webPage, ?Info}) \rrbracket_G
\]

<table>
<thead>
<tr>
<th>(\mu_1)</th>
<th>(?X)</th>
<th>:?Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>(\mu_2)</th>
<th>(?X)</th>
<th>:?Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_3)</td>
<td><a href="mailto:R@ed.ex">R@ed.ex</a></td>
<td></td>
</tr>
</tbody>
</table>
Example (UNION)

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

\[ \llbracket (\text{(?X, email, ?Info)} \cup (\text{(?X, webPage, ?Info)})) \rrbracket_G \]

\[ \llbracket (\text{(?X, email, ?Info)}) \rrbracket_G \cup \llbracket (\text{(?X, webPage, ?Info)}) \rrbracket_G \]

\begin{array}{|c|c|}
\hline
\text{\(\mu_1\)} & \text{\(\mu_2\)} \\
\hline
\text{\(?X\)} & \text{\(?X\)} \\
\hline
\text{\(R_1\)} & \text{\(R_1\)} & \text{\(R_3\)} \\
\hline
\text{\(\text{J@ed.ex}\)} & \text{\(\text{R@ed.ex}\)} & \text{\(\text{www.ringo.com}\)} \\
\hline
\end{array}
Example (UNION)

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

\[
\llbracket \left( (?X, \text{email, ?Info}) \cup (?X, \text{webPage, ?Info}) \right) \rrbracket_G
\]

\[
\llbracket (?X, \text{email, ?Info}) \rrbracket_G \cup \llbracket (?X, \text{webPage, ?Info}) \rrbracket_G
\]

\[
\begin{array}{|c|c|} \hline
\mu_1 & \text{?Info} \\
\hline
R_1 & \text{J@ed.ex} \\
R_3 & \text{R@ed.ex} \\
\hline
\end{array}
\quad \cup \quad
\begin{array}{|c|c|} \hline
\mu_3 & \text{?Info} \\
R_3 & \text{www.ringo.com} \\
\hline
\end{array}
\]
Example (UNION)

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

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

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

\[
\begin{array}{c|c}
\mu_1 & ?X \quad ?\text{Info} \\
\hline
R_1 & \text{J@ed.ex} \\
R_3 & \text{R@ed.ex}
\end{array} \cup \begin{array}{c|c}
\mu_3 & ?X \quad ?\text{Info} \\
\hline
R_3 & \text{www.ringo.com}
\end{array}
\]

\[
\begin{array}{c|c}
\mu_1 & ?X \quad ?\text{Info} \\
\hline
R_1 & \text{J@ed.ex} \\
R_3 & \text{R@ed.ex}
\end{array} \cup \begin{array}{c|c}
\mu_3 & ?X \quad ?\text{Info} \\
\hline
R_3 & \text{www.ringo.com}
\end{array}
\]
Example (SELECT)

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

\[
\llbracket (\text{SELECT} \{?N, ?E\} ((?X, \text{name, } ?N) \text{ AND } (?X, \text{email, } ?E))) \rrbracket_G
\]
Example (SELECT)

\( G : (R_1, \text{name, john}) \quad (R_2, \text{name, paul}) \quad (R_3, \text{name, ringo}) \)

\( G : (R_1, \text{email, J@ed.ex}) \quad (R_3, \text{email, R@ed.ex}) \quad (R_3, \text{webPage, www.ringo.com}) \)

\[\left[ \left( \text{SELECT } \{ ?N, ?E \} \left( (\mu_1 R_1 \text{John J@ed.ex}) \land (\mu_2 R_3 \text{Ringo R@ed.ex}) \right) \right) \right]_G\]
Example (SELECT)

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

\[
\llbracket \left( \text{SELECT } \{ ?N, ?E \} (\left( ?X, \text{name, ?N} \right) \text{ AND } ( ?X, \text{email, ?E} )) \right) \rrbracket_G
\]

\[
\text{SELECT}\{ ?N, ?E \} \\
\mu_1 \\
\mu_2
\]

\[
\begin{array}{l|c|c|c|c|c|c}
\hline
R_1 & \text{john} & J@ed.ex \\
R_3 & \text{ringo} & R@ed.ex \\
\end{array}
\]

\[
\mu_1 | \{ ?N, ?E \}
\]

\[
\begin{array}{l|l}
?N & ?E \\
\hline
\text{john} & J@ed.ex \\
\text{ringo} & R@ed.ex \\
\end{array}
\]

\[
\mu_2 | \{ ?N, ?E \}
\]
Filter expressions (value constraints)

Filter expression: \((P \ \textsc{filter} \ R)\)

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

**Definition**

**FILTER** : selects mappings that satisfy a condition

$$\llbracket (P \text{ FILTER } R) \rrbracket_G = \{ \mu \in \llbracket P \rrbracket_G \mid \mu \models R \}$$
Example (FILTER)

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

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

\[ \llbracket ((\mu X, \text{name}, \mu N) \text{ FILTER } (\mu N = \text{ringo} \lor \mu N = \text{paul})) \rrbracket_G \]

<table>
<thead>
<tr>
<th>( \mu_1 )</th>
<th>( \mu_2 )</th>
<th>( \mu_3 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>\text{john}</td>
<td></td>
</tr>
<tr>
<td>( R_2 )</td>
<td>\text{paul}</td>
<td></td>
</tr>
<tr>
<td>( R_3 )</td>
<td>\text{ringo}</td>
<td></td>
</tr>
</tbody>
</table>
Example (FILTER)

\(G: (R_1, \text{name}, \text{john}) \quad (R_2, \text{name}, \text{paul}) \quad (R_3, \text{name}, \text{ringo})\)

\(G: (R_1, \text{email}, \text{J@ed.ex}) \quad (R_3, \text{email}, \text{R@ed.ex}) \quad (R_3, \text{webPage}, \text{www.ringo.com})\)

\[\llbracket((?X, \text{name}, ?N) \text{ FILTER } (?N = \text{ringo} \lor ?N = \text{paul}))\rrbracket_G\]

\[
\begin{array}{|c|c|}
\hline
?X & ?N \\
\hline
R_1 & \text{john} \\
R_2 & \text{paul} \\
R_3 & \text{ringo} \\
\hline
\end{array}
\]

\(?N = \text{ringo} \lor ?N = \text{paul}\)
Example (FILTER)

\( G : \) (\( R_1, \) email, \( J@ed.ex \))

\( (R_1, \) name, \( \text{john} \) ) \hspace{1cm} (R_2, \) name, \( \text{paul} \) ) \hspace{1cm} (R_3, \) name, \( \text{ringo} \) \\
\( (R_1, \) email, \( J@ed.ex \) ) \hspace{1cm} (R_3, \) email, \( R@ed.ex \) ) \hspace{1cm} (R_3, \) webPage, \( \text{www.ringo.com} \) \\

\( \llbracket ((?X, \text{name}, ?N) \text{ FILTER } (?N = \text{ringo} \lor ?N = \text{paul})) \rrbracket_G \)

\begin{array}{|c|c|}
\hline
\mu_1 & \mu_2 & \mu_3 \\
\hline
R_1 & \text{john} & \text{ringo} \\
R_2 & \text{paul} & \text{} \\
R_3 & \text{} & \text{} \\
\hline
\end{array}

\( ?N = \text{ringo} \lor ?N = \text{paul} \)

\begin{array}{|c|c|}
\hline
\mu_2 & \mu_3 \\
\hline
R_2 & \text{paul} \\
R_3 & \text{ringo} \\
\hline
\end{array}
Example (FILTER)

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

\[
\llbracket(((?X, \text{name, ?N}) \text{ OPT (?X, email, ?E)}) \text{ FILTER } \neg \text{bound(?E)})\rrbracket_G
\]
Example (FILTER)

\[ G = (R_1, \text{name, john}) \quad (R_2, \text{name, paul}) \quad (R_3, \text{name, ringo}) \]

\[ G : (R_1, \text{email, J@ed.ex}) \quad (R_3, \text{email, R@ed.ex}) \quad (R_3, \text{webPage, www.ringo.com}) \]

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu_1 \cup \mu_4)</td>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(\mu_3 \cup \mu_5)</td>
<td>(R_3)</td>
<td>ringo</td>
</tr>
<tr>
<td>(\mu_2)</td>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>
Example (FILTER)

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

\[ \llbracket (((?X, \text{name}, {?N}) \ \text{OPT} \ (?X, \text{email}, {?E})) \ \text{FILTER} \ \neg \text{bound(??E)}) \rrbracket_G \]

\[
\begin{array}{|c|c|c|}
\hline
\hline
R_1 & \text{john} & \text{J@ed.ex} \\
\hline
R_3 & \text{ringo} & \text{R@ed.ex} \\
\hline
R_2 & \text{paul} & \text{} \\
\hline
\end{array}
\]

\( \mu_1 \cup \mu_4 \)
\( \mu_3 \cup \mu_5 \)
\( \mu_2 \)

\( \neg \text{bound(??E)} \)
Example (FILTER)

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

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

\[
\begin{array}{|c|c|c|}
\hline
\hline
R_1 & \text{john} & \text{J@ed.ex} \\
R_3 & \text{ringo} & \text{R@ed.ex} \\
R_2 & \text{paul} & \neg \text{bound(?E)} \\
\hline
\end{array}
\]
SPARQL 1.1
(and some research issues)
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 (\texttt{rdf:sp}), subClassOf (\texttt{rdf:sc}), domain (\texttt{rdf:dom}), range (\texttt{rdf:range}), type (\texttt{rdf:type}).
To remember: Syntax of RDFS

RDFS extends RDF with a schema vocabulary: subPropertyOf (\texttt{rdf:sp}), subClassOf (\texttt{rdf:sc}), domain (\texttt{rdf:dom}), range (\texttt{rdf:range}), type (\texttt{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 : 
\[
\begin{align*}
(A, \text{rdf:sp}, B) & \quad (B, \text{rdf:sp}, C) \\
(A, \text{rdf:sp}, C) & \\
(A, \text{rdf:sp}, B) & \quad (X, A, Y) \\
(X, B, Y) & \\
\end{align*}
\]

Subclass : 
\[
\begin{align*}
(A, \text{rdf:sc}, B) & \quad (B, \text{rdf:sc}, C) \\
(A, \text{rdf:sc}, C) & \\
(A, \text{rdf:sc}, B) & \quad (X, \text{rdf:type}, A) \\
(X, \text{rdf:type}, B) & \\
\end{align*}
\]

Typing : 
\[
\begin{align*}
(A, \text{rdf:dom}, B) & \quad (X, A, Y) \\
(X, \text{rdf:type}, B) & \\
(A, \text{rdf:range}, B) & \quad (X, A, Y) \\
(Y, \text{rdf:type}, B) & \\
\end{align*}
\]
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: What is the answer to (?X, rdf:type, person)?
```
Example 2: What is the answer to (Messi, \textit{rdf:type}, person)?

![RDF Diagram]

- \textit{Messi} is a \texttt{soccer\_player} and \texttt{sportman}.
- \texttt{Messi} \textit{lives in} Spain.
- \texttt{Messi} \textit{plays in} the \texttt{soccer\_team} Barcelona.
- \texttt{Messi} \textit{works in} a \texttt{company}.
- The diagram shows the relationships between these entities using \texttt{rdfs:domain} (\texttt{rdf:dom}) and \texttt{rdfs:range} (\texttt{rdf:range}).
Example 3: What is the answer to
\{(\text{Messi}, \text{rdf:type}, \text{?Y}), (\text{?Y, rdf:sc, 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, rdf:sc, person)
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
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
What are the consequences of considering entailment only at the level bgps?

Example

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

$$(\text{student}, \text{rdf:sc}, u)$$
$$(u, \text{owl:union}, l)$$
$$(l, \text{rdf:first}, \text{undergrad})$$
$$(l, \text{rdf:rest}, r)$$
$$(r, \text{rdf:first}, \text{grad})$$
$$(r, \text{rdf:rest}, \text{rdf:nil})$$

\[
\text{axiom \ student} \sqsubseteq (\text{undergrad} \sqcup \text{grad})
\]
What are the consequences of considering entailment only at the level bgps?

Example

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

- $(\text{student, rdf:sc, u})$
- $(u, \text{owl:union, l})$
- $(l, \text{rdf:first, undergrad})$
- $(l, \text{rdf:rest, r})$
- $(r, \text{rdf:first, grad})$
- $(r, \text{rdf:rest, rdf:nil})$

Axiom: $\text{student} \sqsubseteq (\text{undergrad} \sqcup \text{grad})$

What should be the answer to

$P = ((?X, \text{rdf:type, undergrad}) \text{ UNION } (?X, \text{rdf:type, grad}))$?
What are the consequences of considering entailment only at the level bgps?

Example

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

- $(\text{student, rdf:sc, u})$
- $(u, \text{owl:union, l})$
- $(l, \text{rdf:first, undergrad})$
- $(l, \text{rdf:rest, r})$
- $(r, \text{rdf:first, grad})$
- $(r, \text{rdf:rest, rdf:nil})$

\[
\begin{align*}
\text{axiom student} \sqsubseteq (\text{undergrad} \sqcup \text{grad})
\end{align*}
\]

What should be the answer to

\[P = ((?X, \text{rdf:type, undergrad}) \text{ UNION } (?X, \text{rdf:type, grad}))\]?

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

\[
\text{(SELECT } ?X ( (?X, \text{friendOf}, ?Y) \text{ AND } (?Y, \text{name}, George))))
\]
SPARQL provides limited navigational capabilities

(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))
SPARQL provides limited navigational capabilities

\[
\text{(SELECT } ?X \text{ WHERE } (?X, \text{friendOf}, ?Y) \text{ AND } (?Y, \text{name}, \text{George}))\text{)}
\]
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:

\[ \text{exp} \ := \ a \ | \ \text{exp/exp} \ | \ \text{exp|exp} \ | \ \text{exp}^* \]

where \( a \in U \)
Navigational capabilities in SPARQL 1.1: Property paths

Syntax of property paths:

\[ \text{exp} := a \mid \text{exp}/\text{exp} \mid \text{exp}\mid\text{exp} \mid \text{exp}^* \]

where \( a \in U \)

Other expressions are allowed:

\[ \wedge\text{exp} \quad : \quad \text{inverse path} \]
\[ !\left(a_1|\ldots|a_n\right) \quad : \quad \text{a URI which is not one of } a_i \quad (1 \leq i \leq n) \]
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:

$$[[a]]_G = \{(x, y) \mid (x, a, y) \in G\}$$
The evaluation of a property path over an RDF graph $G$ is defined as follows:

\[
\llbracket a \rrbracket_G = \{(x, y) \mid (x, a, y) \in G\}
\]
\[
\llbracket \text{exp}_1 / \text{exp}_2 \rrbracket_G = \{(x, y) \mid \exists z (x, z) \in \llbracket \text{exp}_1 \rrbracket_G \text{ and } (z, y) \in \llbracket \text{exp}_2 \rrbracket_G\}
\]
Evaluating property paths

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

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

\[
[exp_1/exp_2]_G = \{(x, y) \mid \exists z (x, z) \in [exp_1]_G \text{ and } (z, y) \in [exp_2]_G\}
\]

\[
[exp_1|exp_2]_G = [exp_1]_G \cup [exp_2]_G
\]
Evaluating property paths

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

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

$$[\text{exp}_1/\text{exp}_2]_G = \{(x, y) \mid \exists z \ (x, z) \in [\text{exp}_1]_G \text{ and } (z, y) \in [\text{exp}_2]_G\}$$

$$[\text{exp}_1|\text{exp}_2]_G = [\text{exp}_1]_G \cup [\text{exp}_2]_G$$

$$[\text{exp}^*]_G = \{(a, a) \mid a \text{ is a URI in } G\} \cup [\text{exp}]_G \cup [\text{exp}/\text{exp}]_G \cup [\text{exp}/\text{exp}/\text{exp}]_G \cup \cdots$$
Property paths in SPARQL 1.1

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

- \text{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

- \((?X, (friendOf)^*, ?Y)\): 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 \(?Y\)
- \((?X, (\text{rdf:sc})^*, \text{person})\): Verifies whether the value stored in \(?X\) is a subclass of \text{person}
Property paths in SPARQL 1.1

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

- \(\text{exp}\) 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 \(?Y\)

- \((?X, (\text{rdf:sc})^*, \text{person})\): Verifies whether the value stored in \(?X\) is a subclass of person

- \((?X, (\text{rdf:sp})^*, ?Y)\): Verifies whether the value stored in \(?X\) is a subproperty of the value stored in \(?Y\)
Semantics of property paths

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

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

$\triangleright \text{dom}(\mu) = \{?X, ?Y\}$
Semantics of property paths

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

- \( \text{dom}(\mu) = \{\mathit{?X}, \mathit{?Y}\} \)
- \( (\mu(\mathit{?X}), \mu(\mathit{?Y})) \in \llbracket \mathit{exp} \rrbracket_G \)
Evaluation of $t = (?X, \text{exp}, ?Y)$ over an RDF graph $G$ is the set of mappings $\mu$ such that:

- $\text{dom}(\mu) = \{?X, ?Y\}$
- $(\mu(?X), \mu(?Y)) \in \llbracket \text{exp} \rrbracket_G$

Other cases are defined analogously.
Semantics of property paths

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

- \( \text{dom}(\mu) = \{ ?X, ?Y \} \)
- \( (\mu(?X), \mu(?Y)) \in \llbracket \text{exp} \rrbracket_G \)

Other cases are defined analogously.

Example

- \( ((?X, \text{KLM}/(\text{KLM})^*, ?Y) \text{ FILTER } \neg (?X = ?Y)) \): It is possible to go from \( ?X \) to \( ?Y \) by using the airline KLM, where \( ?X, ?Y \) are different cities
List the pairs $a, b$ of cities such that there is a way to travel from $a$ to $b$. 
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]:
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.
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/>

inPods:FaginLN01

:Ronald_Fagin

owl:sameAs

dbo:birthPlace
dbo:Ronald_Fagin

dbpedia:Ronald_Fagin

rdf:type

yago:DatabaseResearchers

rdfs:subClassOf

yago:ResearchWorker

dbpedia:Oklahoma

```

DBLP

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

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

\[ \text{ep(·)}: \text{Partial function from } U \text{ to the set of all RDF graphs} \]

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

  $$[P]_G = \bigcup_{a \in \text{dom}(\text{ep})} \left( [P_1]_{\text{ep}(a)} \square \{c \rightarrow a\} \right),$$
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 \(\mu \in \llbracket P \rrbracket_G\), it holds that \(?X \in \text{dom}(\mu)\) and \(\mu(\?X) \in U\)

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
  (\(?X\), service_address, \(?Y\)) AND (SERVICE \(?Y\) (\(?N\), email, \(?E\)))
The first attempt: Too restrictive

Consider the query:

\[
(?X, \text{service\_description}, ?Z) \text{ UNION } \\
\left( (?X, \text{service\_address}, ?Y) \text{ AND (SERVICE } ?Y \text{ (?N, email, ?E))} \right)
\]

\(?Y\) 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:

\((?U_1, \text{related\_with}, ?U_2) \ \text{AND} \ \left[ \text{SERVICE } ?U_1 \left( (?N, \text{email}, ?E) \ \text{OPT} \ \left( \text{SERVICE } ?U_2 \ (?N, \text{phone}, ?F) \right) \right) \right] \)
Solving the problems . . .

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

Parse tree of $(？Y, a, ?Z) \cup ((？X, b, c) \text{ AND } \text{SERVICE } ?X (？Y, a, ?Z))$:
A more appropriate notion of boundedness

Definition [BACP13]

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

- there exists a node $v$ of $T(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $T(P)$ 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:

```
(?Y, a, ?Z) UNION ((?X, b, c) AND (SERVICE ?X (?Y, a, ?Z)))
```

```
(?Y, a, ?Z) UNION ((?X, b, c) AND (SERVICE ?X (?Y, a, ?Z)))
```

```
(?X, b, c) AND (SERVICE ?X (?Y, a, ?Z))
```

```
(?X, b, c)
```

```
(SERVICE ?X (?Y, a, ?Z))
```

```
(?Y, a, ?Z)
```

```
(?Y, a, ?Z)
```

```
(?X, b, c)
```

```
(SERVICE ?X (?Y, a, ?Z))
```

```
(?Y, a, ?Z)
```

```
(?Y, a, ?Z)
```
A more appropriate notion of boundedness

Definition [BACP13]

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

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

Examples:

$$
(?Y, a, ?Z) \text{ UNION } ((?X, b, c) \text{ AND } \text{SERVICE } ?X (?Y, a, ?Z)))
$$

$$
(?Y, a, ?Z) \text{ UNION } ((?X, b, c) \text{ AND } \text{SERVICE } ?X (?Y, a, ?Z)))
$$

$$
(?X, b, c) \text{ AND } \text{SERVICE } ?X (?Y, a, ?Z))
$$

$$
\text{SERVICE } ?X (?Y, a, ?Z))
$$

$$
(?Y, a, ?Z)
$$
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:

$$(?Y, a, ?Z) \text{ UNION } ((?X, b, c) \text{ AND (SERVICE } ?X (?Y, a, ?Z)))$$

$$(?Y, a, ?Z)$$

$$(?X, b, c)$$

$$(?X, b, c) \text{ AND (SERVICE } ?X (?Y, a, ?Z))$$

$$(?X, b, c)$$

$$(\text{SERVICE } ?X (?Y, a, ?Z))$$

$$(?Y, a, ?Z)$$

$$(?Y, a, ?Z)$$
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 $(\text{SERVICE } ?X P_1)$, 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:

- $(?Y, a, ?Z)$
- $(?X, b, c)$ 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 $(\text{SERVICE } ?X P_1)$, 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:**

```
(?U_1, rw, ?U_2) AND (SERVICE ?U_1 ((?N, e, ?E) OPT (SERVICE ?U_2 (?N, ph, ?F))))
```

```
(?U_1, rw, ?U_2) -> (SERVICE ?U_1 ((?N, e, ?E) OPT (SERVICE ?U_2 (?N, ph, ?F))))
```

```
```

```
(?U_1, rw, ?U_2) -> (SERVICE ?U_1 ((?N, e, ?E) OPT (SERVICE ?U_2 (?N, ph, ?F))))
```

```
```

```
(SERVICE ?U_2 (?N, ph, ?F)) -> (?N, ph, ?F)
```

```
(?U_1, rw, ?U_2) -> (SERVICE ?U_1 ((?N, e, ?E) OPT (SERVICE ?U_2 (?N, ph, ?F))))
```

```
```

```
(SERVICE ?U_2 (?N, ph, ?F)) -> (?N, ph, ?F)
```
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 $(\text{SERVICE } ?X P_1)$, 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:

- $(?U_1, \text{rw}, ?U_2) \land (\text{SERVICE } ?U_1 ((?N, e, ?E) \text{ OPT } (\text{SERVICE } ?U_2 (?N, \text{ph}, ?F))))$
- $(?U_1, \text{rw}, ?U_2) 
  \rightarrow (\text{SERVICE } ?U_1 ((?N, e, ?E) \text{ OPT } (\text{SERVICE } ?U_2 (?N, \text{ph}, ?F))))$
- $((?N, e, ?E) \text{ OPT } (\text{SERVICE } ?U_2 (?N, \text{ph}, ?F))))$
- $(?N, e, ?E) 
  \rightarrow (\text{SERVICE } ?U_2 (?N, \text{ph}, ?F))$
- $(?N, \text{ph}, ?F)$
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 $\text{?}X$ $P_1$), 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 $\text{?}X$ is bound in $P_2$
- $P_1$ is service-bound

Examples:

```
(?U_1, rw, ?U_2) AND (SERVICE ?U_1 ((?N, e, ?E) OPT (SERVICE ?U_2 (?N, ph, ?F))))
```

```
(SERVICE ?U_1 ((?N, e, ?E) OPT (SERVICE ?U_2 (?N, ph, ?F))))
```

```
((?N, e, ?E) OPT (SERVICE ?U_2 (?N, ph, ?F)))
```

```
(?N, e, ?E)
```

```
(SERVICE ?U_2 (?N, ph, ?F))
```

```
(?N, ph, ?F)
```
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 \( T(P) \) with label (SERVICE ?X \( P_1 \)), it holds that:

- there exists a node \( v \) of \( T(P) \) with label \( P_2 \) such that \( v \) is an ancestor of \( u \) in \( T(P) \) 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 $SB(P)$, is recursively defined as follows:

- if $P$ is a bgp, then $SB(P) = var(P)$
- if $P = (P_1 \ AND \ P_2)$, then $SB(P) = SB(P_1) \cup SB(P_2)$
- if $P = (P_1 \ UNION \ P_2)$, then $SB(P) = SB(P_1) \cap SB(P_2)$
- if $P = (P_1 \ OPT \ P_2)$, then $SB(P) = SB(P_1)$
- if $P = (P_1 \ FILTER \ R)$, then $SB(P) = SB(P_1)$
- if $P = (SERVICE \ c \ P_1)$, then $SB(P) = \emptyset$
An appropriate notion: Service-safeness (cont’d)

Definition [BACP13]

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

- there exists a node $v$ of $T(P)$ with label $P_2$ such that $v$ is an ancestor of $u$ in $T(P)$ and $?X \in SB(P_2)$
- $P_1$ 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 $T(P)$ with label (SERVICE $?X$ $P_1$), it holds that:

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

SPARQL 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 (cont’d)


