Querying Semantic Web Data with SPARQL (and SPARQL 1.1)

Marcelo Arenas

PUC Chile & University of Oxford
“The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.”

[Tim Berners-Lee et al. 2001.]

Specific goals:

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

inPods:FaginLN01

dc:creator

:Amnon_Lotem

:Moni_Naor

:Ronald_Fagin
SPARQL is the W3C recommendation query language for RDF (January 2008).

- SPARQL is a recursive acronym that stands for SPARQL Protocol and RDF Query Language

SPARQL is a graph-matching query language.

- A SPARQL query consists of three parts:
  - Pattern matching: optional, union, filtering, . . .
  - Solution modifiers: projection, distinct, order, limit, offset, . . .
  - Output part: construction of new triples, . . .
SPARQL in a nutshell
SELECT ?Author
SELECT ?Author
WHERE
{

}


SELECT ?Author
WHERE
{
}
Outline of the talk

- RDF and SPARQL
- New features in SPARQL 1.1
  - Entailment regimes for RDFS and OWL
  - Navigational capabilities: Property paths
  - An operator to distribute the execution of a query
- Take-home message
Outline of the talk

- RDF and SPARQL

- New features in SPARQL 1.1
  - Entailment regimes for RDFS and OWL
  - Navigational capabilities: Property paths
  - An operator to distribute the execution of a query

- Take-home message
RDF formal model

\[ \text{I} : \text{set of IRIs} \]
\[ \text{B} : \text{set of blank nodes} \]
\[ \text{L} : \text{set of literals} \]
RDF formal model

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

- \(I\): set of IRIs
- \(B\): set of blank nodes
- \(L\): set of literals
RDF formal model

A finite set of RDF triples is called an RDF graph

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

Proviso

- We do not consider blank nodes in RDF graphs
  - $(s, p, o) \in I \times I \times (I \cup L)$ is called an RDF triple

- We consider blank nodes in queries
  - Each blank node is assumed to start with `_:`, for example `_:b` and `_:b_1`
**V**: set of variables

- Each variable is assumed to start with `?`

**Triple pattern**: \( t \in (I \cup B \cup V) \times (I \cup V) \times (I \cup B \cup L \cup V) \)

- Examples: (?X, name, john), (?X, name, ?Y), (?X, name, _:b)

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

- Examples: \{(?X, knows, ?Y), (?Y, name, john)\}, \{(?X, knows, _:b), (_:b, 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 patterns
- 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
Mappings: building block for the semantics

Definition
A mapping is a partial function:

\[ \mu : V \rightarrow (I \cup L) \]

The evaluation of a graph pattern results in a set of mappings.
Mappings: building block for the semantics

**Definition**

A mapping is a partial function:

\[ \mu : V \rightarrow (I \cup L) \]

The evaluation of a graph pattern results in a set of mappings.
Additional notation: $\sigma : B \rightarrow (I \cup L)$ is an instance mapping.
Additional notation: $\sigma : \mathcal{B} \rightarrow (\mathcal{I} \cup \mathcal{L})$ is an instance mapping.

Let $P$ be a basic graph pattern

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

**Definition**

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

- $\text{dom}(\mu) = \text{var}(P)$
- there exists an instance mapping $\sigma$ such that $\mu(\sigma(P)) \subseteq G$
Semantics of basic graph patterns: Some examples

Notation: $t$ is used to represent $\{t\}$
Semantics of basic graph patterns: Some examples

Notation: \( t \) is used to represent \( \{ t \} \)

<table>
<thead>
<tr>
<th>graph</th>
<th>bgp</th>
<th>evaluation</th>
</tr>
</thead>
</table>
| \((R_1, \text{name, john})\) | \((?X, \text{name, } ?Y)\) | \(\mu_1: \)
| \((R_1, \text{email, J@ed.ex})\) | | \(R_1\) \hspace{1cm} john \hspace{1cm} \mu_2: \)
| \((R_2, \text{name, paul})\) | | \(R_2\) \hspace{1cm} paul |
Semantics of basic graph patterns: Some examples

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

<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>$\mu_1: R_1 \quad \text{john}$</td>
</tr>
<tr>
<td>$(R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>})$</td>
<td></td>
<td>$\mu_2: R_2 \quad \text{paul}$</td>
</tr>
<tr>
<td>$(R_2, \text{name, paul})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Semantics of basic graph patterns: Some examples

Notation: \( t \) is used to represent \( \{t\} \)

<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>(\mu_1:)</td>
</tr>
<tr>
<td>((R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>}))</td>
<td></td>
<td>(R_1)</td>
</tr>
<tr>
<td>((R_2, \text{name, paul}))</td>
<td></td>
<td>(\mu_2:)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(R_2)</td>
</tr>
</tbody>
</table>
Semantics of basic graph patterns: Some examples

Notation: \( t \) is used to represent \( \{ t \} \)

<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, name, ?Y)</td>
<td>(\mu_1: R_1) (\text{john})</td>
</tr>
<tr>
<td>((R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>}))</td>
<td>(\mu_2: R_2 ) (\text{paul})</td>
<td></td>
</tr>
<tr>
<td>((R_2, \text{name, paul}))</td>
<td>(\mu_3: R_1) (\mu_4: R_2)</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>graph</th>
<th>bgp</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>((R_1, \text{name, john}))</td>
<td>(\mu_3: R_1) (\mu_4: R_2)</td>
<td></td>
</tr>
<tr>
<td>((R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>}))</td>
<td>(\mu_3: R_1) (\mu_4: R_2)</td>
<td></td>
</tr>
<tr>
<td>((R_2, \text{name, paul}))</td>
<td>(\mu_3: R_1) (\mu_4: R_2)</td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings

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

**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>$\mu_1$ :</th>
<th>$\mu_2$ :</th>
<th>$\mu_3$ :</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>john</td>
<td>$\text{<a href="mailto:P@edu.ex">P@edu.ex</a>}$</td>
</tr>
<tr>
<td>$R_1$</td>
<td>$\text{<a href="mailto:J@edu.ex">J@edu.ex</a>}$</td>
<td></td>
</tr>
<tr>
<td>$\text{?X}$</td>
<td>$\text{?Y}$</td>
<td>$\text{?Z}$</td>
</tr>
<tr>
<td>$\text{R_2}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings

**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>$\mu_1$ :</td>
<td>$R_1$</td>
<td>john</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>
</tr>
<tr>
<td>$\mu_3 : $</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>$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>
</tr>
</tbody>
</table>
**Compatible mappings**

**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><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td>R2</td>
</tr>
<tr>
<td>$\mu_3$</td>
<td></td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></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

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

**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><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$</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>$\mu_3$</td>
<td></td>
<td></td>
<td></td>
<td></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>
<tr>
<td>$\mu_1 \cup \mu_3$</td>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>$R_2$</td>
</tr>
</tbody>
</table>

$\triangleright \mu_2$ and $\mu_3$ are not compatible
Let $\Omega_1$ and $\Omega_2$ be sets of mappings.

**Definition**

**Join:** extends mappings in $\Omega_1$ with compatible mappings in $\Omega_2$

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

**Difference:** selects mappings in $\Omega_1$ that cannot be extended with mappings in $\Omega_2$

- $\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \text{there is no mapping in } \Omega_2 \text{ compatible with } \mu_1\}$
Sets of mappings and operations

Definition

**Union**: includes mappings in $\Omega_1$ and in $\Omega_2$

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

**Left Outer Join**: extends mappings in $\Omega_1$ with compatible mappings in $\Omega_2$ if possible

$\Omega_1 \bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$
Given an RDF graph $G$

<table>
<thead>
<tr>
<th>Definition</th>
<th>Equation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[(P_1 \text{ AND } P_2)]_G$</td>
<td>=</td>
</tr>
<tr>
<td>$[(P_1 \text{ UNION } P_2)]_G$</td>
<td>=</td>
</tr>
<tr>
<td>$[(P_1 \text{ OPT } P_2)]_G$</td>
<td>=</td>
</tr>
<tr>
<td>$[(\text{SELECT } W \ P)]_G$</td>
<td>=</td>
</tr>
</tbody>
</table>
Given an RDF graph $G$

| Definition |
|------------------|------------------|
| $[(P_1 \text{ AND } P_2)]_G$ | $=[P_1]_G \times [P_2]_G$ |
| $[(P_1 \text{ UNION } P_2)]_G$ | $=[P_1]_G \cup [P_2]_G$ |
| $[(P_1 \text{ OPT } P_2)]_G$ | $=[P_1]_G \triangleright [P_2]_G$ |
| $[(\text{SELECT } W \ P)]_G$ | $=\{\mu|_W | \mu \in [P]_G\}$ |
Given an RDF graph $G$

**Definition**

$\llbracket (P_1 \text{ AND } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \land \llbracket P_2 \rrbracket_G$

$\llbracket (P_1 \text{ UNION } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G$

$\llbracket (P_1 \text{ OPT } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \triangledown \llbracket P_2 \rrbracket_G$

$\llbracket (\text{SELECT } W \text{ P}) \rrbracket_G = \{\mu\upharpoonright_W \mid \mu \in \llbracket P \rrbracket_G\}$

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

$\mu\upharpoonright_W(?X) = \mu(?X)$ for every $?X \in \text{dom}(\mu\upharpoonright_W)$
Example

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

\[((?X, \text{name}, ?Y) \text{OPT} (?X, \text{email}, ?E))\]
Semantics of SPARQL: An example

Example

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

\(( (?X, \text{name}, ?Y) \text{ OPT } (?X, \text{email}, ?E) )\)
Semantics of SPARQL: An example

Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>
Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>
Semantics of SPARQL: An example

Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>john</td>
</tr>
<tr>
<td>R₂</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>
Semantics of SPARQL: An example

Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td>john</td>
</tr>
<tr>
<td>R_2</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_1</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>
Semantics of SPARQL: An example

Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>
Semantics of SPARQL: An example

Example

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

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

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

\[
\begin{array}{c|c|c}
?X & ?Y & ?E \\
\hline
R_1 & \text{john} & \text{J@ed.ex} \\
\end{array}
\]

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

▶ from the Join
Example

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

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

\begin{array}{|c|c|}
\hline
?X & ?Y \\
\hline
R_1 & john \\
R_2 & paul \\
\hline
\end{array}

\begin{array}{|c|c|}
\hline
?X & ?Y & ?E \\
\hline
 & & \\
R_2 & paul & \\
\hline
\end{array}

\begin{array}{|c|c|}
\hline
?X & ?E \\
\hline
R_1 & J@ed.ex \\
\hline
\end{array}

\textbf{from the Difference}
Semantics of SPARQL: An example

Example

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

\[ \left( (\text{?X, name, ?Y}) \text{ OPT } (\text{?X, email, ?E}) \right) \]

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>\text{john}</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>\text{paul}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>( R_1 )</td>
<td>\text{john}</td>
<td>\text{<a href="mailto:J@ed.ex">J@ed.ex</a>}</td>
</tr>
<tr>
<td>( R_2 )</td>
<td>\text{paul}</td>
<td></td>
</tr>
</tbody>
</table>

\[ \text{from the Union} \]
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 \((\land, \lor, \neg)\)
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 $\mu(\,?X\,) = c$
- $R$ 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

**Definition**

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

$$\llbracket (P \text{ FILTER } R) \rrbracket_G = \{ \mu \in \llbracket P \rrbracket_G \mid \mu \models R \}$$
Outline of the talk

- RDF and SPARQL

- New features in SPARQL 1.1
  - Entailment regimes for RDFS and OWL
  - Navigational capabilities: Property paths
  - An operator to distribute the execution of a query

- Take-home message
SPARQL 1.1

A new version of SPARQL has just been 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)
Outline of the talk

- RDF and SPARQL

- New features in SPARQL 1.1
  - Entailment regimes for RDFS and OWL
  - Navigational capabilities: Property paths
  - An operator to distribute the execution of a query

- Take-home message
RDFS extends RDF with a schema vocabulary: subPropertyOf (rdfs:sp), subClassOf (rdfs:sc), domain (rdfs:dom), range (rdfs:range), type (rdfs:type).
Syntax of RDFS

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

How do we evaluate a query over RDFS data?
A simple SPARQL query: $(\text{Messi}, \text{rdfs:type}, \text{person})$
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:

- $(A, rdfs:sp, B)$  $(B, rdfs:sp, C)$
- $(A, rdfs:sp, C)$
- $(A, rdfs:sp, B)$  $(\lambda, A, \gamma)$
- $(\lambda, B, \gamma)$

Subclass:

- $(A, rdfs:sc, B)$  $(B, rdfs:sc, C)$
- $(A, rdfs:sc, C)$
- $(A, rdfs:sc, B)$  $(\lambda, rdfs:type, A)$
- $(\lambda, rdfs:type, B)$

Typing:

- $(A, rdfs:dom, B)$  $(\lambda, A, \gamma)$
- $(\lambda, rdfs:type, B)$
- $(A, rdfs:range, B)$  $(\lambda, A, \gamma)$
- $(\gamma, rdfs:type, B)$
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.
Basic graph patterns are evaluated by considering RDFS entailment.

**Definition**

The evaluation of a bgp $P$ over an RDF graph $G$, denoted by $\llbracket P \rrbracket_G$, is the set of mappings $\mu$:

- $\text{dom}(\mu) = \text{var}(P)$
- there exists an instance mapping $\sigma$ such that for every $t \in P$: $G \models \mu(\sigma(t))$
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 $\mu$:

- $\text{dom}(\mu) = \text{var}(P)$
- there exists an instance mapping $\sigma$ such that for every $t \in P$: $G \models \mu(\sigma(t))$

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

- RDFS entailment is only used at the level of bgps
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, rdfs: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, rdfs:sc, person)

- Basic graph patterns can also be evaluated by considering OWL entailment.
  - $G \models \mu(\sigma(t))$ 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 $(\text{john}, \text{rdfs:type}, \text{student})$ together with:

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

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

What should be the answer to

$P = ((?X, \text{rdfs:type}, \text{undergrad}) \cup (?X, \text{rdfs:type}, \text{grad}))$?

- Under the current semantics: $\llbracket P \rrbracket_G = \emptyset$
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

But what happens if we focus on the case of RDFS?

- The semantics do not coincide as the following operator can be expressed in the language:

\[
\llbracket (P_1 \text{ MINUS } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \setminus \llbracket P_2 \rrbracket_G
\]
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.

But what happens if we focus on the case of RDFS?

The semantics do not coincide as the following operator can be expressed in the language:

\[
\llbracket (P_1 \text{ MINUS } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \setminus \llbracket P_2 \rrbracket_G
\]

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?
Outline of the talk

- RDF and SPARQL

- New features in SPARQL 1.1
  - Entailment regimes for RDFS and OWL
  - Navigational capabilities: Property paths
  - An operator to distribute the execution of a query

- Take-home message
SPARQL provides limited navigational capabilities.

URI

Paul

name

paul@puc.cl

e-mail

friendOf

John

446928888

phone

ame

friendOf

George

friendOf

Peter

name

friendOf

URI1

URI0

URI2

URI3

URI
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

URI1
  name → Paul
  email → paul@puc.cl

friendOf

URI2
  phone → 446928888
  name → John
  email → john@utexas.edu

friendOf

URI0
  name → Peter

friendOf

URI3
  name → George
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:

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

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

Syntax of property paths:

\[ \text{exp} \triangleq \ a \mid \text{exp}/\text{exp} \mid \text{exp}\lvert\text{exp} \mid \text{exp}^* \]

where \( a \in I \)

Other expressions are allowed:

\( \check{\text{exp}} \) : inverse path
\( !(a_1\mid\ldots\mid a_n) \) : an IRI 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:

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

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

$\llbracket a \rrbracket_G = \{ (x, y) \mid (x, a, y) \in G \}$

$\llbracket \text{exp}_1 / \text{exp}_2 \rrbracket_G = \{ (x, y) \mid \exists z \ (x, z) \in \llbracket \text{exp}_1 \rrbracket_G$ 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:

- $\llbracket a \rrbracket_G = \{(x, y) \mid (x, a, y) \in G\}$
- $\llbracket \text{exp}_1/\text{exp}_2 \rrbracket_G = \{(x, y) \mid \exists z (x, z) \in \llbracket \text{exp}_1 \rrbracket_G \text{ and } (z, y) \in \llbracket \text{exp}_2 \rrbracket_G\}$
- $\llbracket \text{exp}_1|\text{exp}_2 \rrbracket_G = \llbracket \text{exp}_1 \rrbracket_G \cup \llbracket \text{exp}_2 \rrbracket_G$
- $\llbracket \text{exp}^* \rrbracket_G = \{(a, a) \mid a \text{ is an IRI in } G\} \cup \llbracket \text{exp} \rrbracket_G \cup \llbracket \text{exp}/\text{exp} \rrbracket_G \cup \llbracket \text{exp}/\text{exp}/\text{exp} \rrbracket_G \cup \cdots$
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 \(\mathbb{I}\) or a variable
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 \(I\) or a variable

Example

- \((?X, \text{rdfs:sc})^*, \text{person})\): Verifies whether the value stored in \(?X\) is a subclass of \text{person}
- \((?X, \text{rdfs:sp})^*, ?Y\): Verifies whether the value stored in \(?X\) is a subproperty of the value stored in \(?Y\)
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 $t = (\textit{?X}, \textit{exp}, \textit{?Y})$ over an RDF graph $G$ is the set of mappings $\mu$ such that:

- The domain of $\mu$ is $\{\textit{?X}, \textit{?Y}\}$, and
- $(\mu(\textit{?X}), \mu(\textit{?Y})) \in [\textit{exp}]_G$
Evaluation of $t = (?X, exp, ?Y)$ over an RDF graph $G$ is the set of mappings $\mu$ such that:

- The domain of $\mu$ is $\{?X, ?Y\}$, and
- $(\mu(?X), \mu(?Y)) \in \llbracket exp \rrbracket_G$

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:

- The domain of $\mu$ is $\{\?X, \?Y\}$, and
- $(\mu(\?X), \mu(\?Y)) \in \llbracket \text{exp} \rrbracket_G$

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: $(?X, \text{transportation\_service}^*, ?Y)$
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.
Outline of the talk

- RDF and SPARQL

- New features in SPARQL 1.1
  - Entailment regimes for RDFS and OWL
  - Navigational capabilities: Property paths
  - An operator to distribute the execution of a query

- Take-home message
RFD graphs can be interconnected.
Retrieve the authors that have published in PODS and were born in Oklahoma:

```sparql
SELECT ?Author
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?Person owl:sameAs ?Author .
  }
}
```
New rule to generate graph patterns:

- If $P$ is a graph pattern and $c \in (I \cup V)$, then (SERVICE $c$ $P$) is a graph pattern.
New rule to generate graph patterns:

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

We will define the semantics of this new operator.

- This corresponds with the official semantics for (SERVICE $c P$) with $c \in I$

- (SERVICE $?X P$) is allowed in the official specification of SPARQL 1.1, but its semantics is not defined.
Semantics of SERVICE

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

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

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

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

\[
[P]_G = \bigcup_{a \in \text{dom}(ep)} \left( [P_1]_{ep(a)} \boxtimes \{\mu_{c \rightarrow a}\} \right),
\]

where \(\mu_{c \rightarrow a}\) is a mapping such that \(\text{dom}(\mu_{c \rightarrow a}) = \{c\}\) and \(\mu_{c \rightarrow a}(c) = a\)
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:

\[(?X, \text{service}\_\text{address}, ?Y) \text{ AND (SERVICE } ?Y (N, \text{email}, ?E))\]

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 \( \mu(?X) \) is mentioned in \( G \)

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 
  \((?X, \text{service}_\text{-address}, ?Y) \) AND (SERVICE \( ?Y \ (\text{/\text{N}}, \text{email}, ?E) \))
The first attempt: Too restrictive

Consider the query:

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

?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:

\[(\text{?}U_1, \text{related_with, ?}U_2) \quad \text{AND} \quad \left[ \text{SERVICE ?}U_1 \left( (\text{?N, email, ?}E) \quad \text{OPT} \quad (\text{SERVICE ?}U_2 (\text{?N, phone, ?}F)) \right) \right] \]
Notation: $\mathcal{T}(P)$ is the parse tree of $P$, in which every node corresponds to a sub-pattern of $P$

Parse tree of $(?Y, a, ?Z) \text{ UNION } ((?X, b, c) \text{ AND } (\text{SERVICE } ?X (?Y, a, ?Z)))$:
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)
```

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

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

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

```
(?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 $\mathcal{T}(P)$ 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:**

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

```
(\textit{?Y}, a, ?Z)
```

```
(\textit{?X}, b, c)
```

```
(\textit{?Y}, a, ?Z)
```

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

```
(\textit{?Y}, a, ?Z)
```

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

```
(\textit{?Y}, a, ?Z)
```

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

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

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

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

```
(SERVICE ?X (\textit{?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:**

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

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

\[
(?Y, a, ?Z)
\]

\[
(?X, b, c)
\]

\[
(?Y, a, ?Z)
\]
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

Examples:

$(?Y, a, ?Z)$

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

**Examples:**

\[(?U_1, rw, ?U_2) \text{ AND (SERVICE } ?U_1 ((?N, e, ?E) \text{ OPT (SERVICE } ?U_2 (?N, ph, ?F)))}\]

\[(?U_1, rw, ?U_2)\]

\[(?N, e, ?E) \text{ OPT (SERVICE } ?U_2 (?N, ph, ?F))\]

\[(?N, e, ?E)\]

\[(?N, ph, ?F)\]

\[(?N, ph, ?F)\]
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:**

\[
(\text{?U}_1, \text{rw}, \text{?U}_2) \quad \land \quad (\text{SERVICE} \ ?U_1 ((?N, \text{e}, ?E) \ \text{OPT} \ (\text{SERVICE} \ ?U_2 (?N, \text{ph}, ?F))))
\]

\[
(\text{?U}_1, \text{rw}, \text{?U}_2) \quad \land \quad (\text{SERVICE} \ ?U_1 ((?N, \text{e}, ?E) \ \text{OPT} \ (\text{SERVICE} \ ?U_2 (?N, \text{ph}, ?F))))
\]

\[
(\text{ SERVICE} \ ?U_1 ((?N, \text{e}, ?E) \ \text{OPT} \ (\text{SERVICE} \ ?U_2 (?N, \text{ph}, ?F))))
\]

\[
(\text{?N, e, ?E}) \quad \land \quad (\text{SERVICE} \ ?U_2 (?N, \text{ph}, ?F))
\]

\[
(\text{?N, ph, ?F})
\]
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:

- $(?U_1, rw, ?U_2)$ AND (SERVICE $?U_1 ((?N, e, ?E) OPT (SERVICE $?U_2 (?N, ph, ?F))))$
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:

\[
((?N, e, ?E) \text{ OPT (SERVICE } ?U_2 (=?N, \text{ph, } ?F)))
\]

\[
(?N, e, ?E)
\]

\[
\text{(SERVICE } ?U_2 (=?N, \text{ph, } ?F))
\]

\[
(=?N, \text{ph, } ?F)
\]
A more appropriate notion of boundedness

Definition (BACP13)
A graph pattern $P$ is service-bound if for every node $u$ of $T(P)$ 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 $SB(P) = \text{var}(P)$
- if $P = (P_1 \text{ AND } P_2)$, then $SB(P) = SB(P_1) \cup SB(P_2)$
- if $P = (P_1 \text{ UNION } P_2)$, then $SB(P) = SB(P_1) \cap SB(P_2)$
- if $P = (P_1 \text{ OPT } P_2)$, then $SB(P) = SB(P_1)$
- if $P = (P_1 \text{ FILTER } R)$, then $SB(P) = SB(P_1)$
- if $P = (\text{SERVICE } c P_1)$, then $SB(P) = \emptyset$
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 ∈ 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?
Outline of the talk

- RDF and SPARQL

- New features in SPARQL 1.1
  - Entailment regimes for RDFS and OWL
  - Navigational capabilities: Property paths
  - An operator to distribute the execution of a query

- Take-home message
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!


