Querying Semantic Web Data with SPARQL (and SPARQL 1.1)

Marcelo Arenas

PUC Chile
&
Center for Semantic Web Research
(ciws.cl)

Semantic Web

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

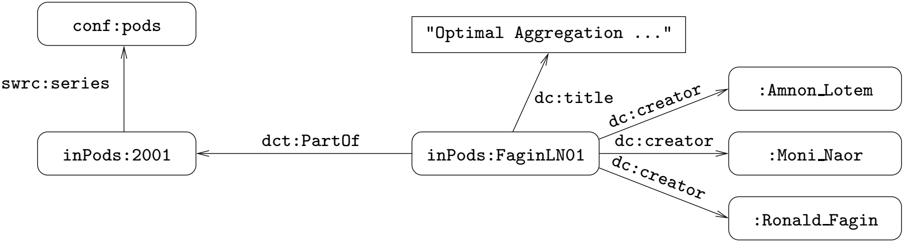
[Tim Berners-Lee et al. 2001.]

Specific goals:

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

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/>
```



Querying RDF: SPARQL

- SPARQL is the W3C recommendation query language for RDF (January 2008).
  - SPARQL is a recursive acronym that stands for *SPARQL Protocol and RDF Query Language*

- SPARQL is a graph-matching query language.

- A SPARQL query consists of three parts:
  - Pattern matching: optional, union, filtering, ...
  - Solution modifiers: projection, distinct, order, limit, offset, ...
  - Output part: construction of new triples, ...

SPARQL in a nutshell

SPARQL in a nutshell

```
SELECT ?Author
```

SPARQL in a nutshell

```
SELECT ?Author
WHERE
{



}
```

SPARQL in a nutshell

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

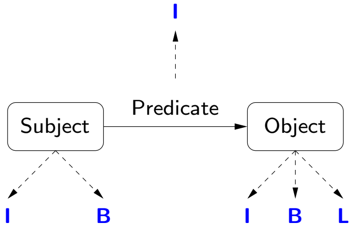
Outline of the talk

- 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

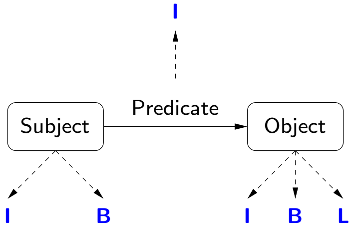
- RDF and SPARQL
- New features in SPARQL 1.1
  - Entailment regimes for RDFS and OWL
  - Navigational capabilities: Property paths
  - An operator to distribute the execution of a query
- Take-home message

RDF formal model



I	:	set of IRIs
B	:	set of blank nodes
L	:	set of literals

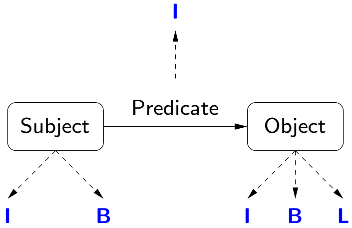
RDF formal model



I	:	set of IRIs
B	:	set of blank nodes
L	:	set of literals

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

RDF formal model



I	:	set of IRIs
B	:	set of blank nodes
L	:	set of literals

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

A finite set of RDF triples is called an RDF graph

RDF formal model

Proviso

- We do not consider blank nodes in RDF graphs
  - $(s, p, o) \in \mathbf{I} \times \mathbf{I} \times (\mathbf{I} \cup \mathbf{L})$ 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

SPARQL: An algebraic syntax

V: set of variables

- Each variable is assumed to start with ?

Triple pattern: $t \in (\mathbf{I} \cup \mathbf{B} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{B} \cup \mathbf{L} \cup \mathbf{V})$

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

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

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

SPARQL: An algebraic syntax (cont'd)

Recursive definition of SPARQL graph patterns:

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

SPARQL query:

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

Mappings: building block for the semantics

Definition

A mapping is a partial function:

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

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

Mappings: building block for the semantics

Definition

A mapping is a partial function:

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

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

Semantics of SPARQL: Basic graph patterns

Additional notation: $\sigma : \mathbf{B} \rightarrow (\mathbf{I} \cup \mathbf{L})$ is an instance mapping.

Semantics of SPARQL: Basic graph patterns

Additional notation: $\sigma : \mathbf{B} \to (\mathbf{I} \cup \mathbf{L})$ is an instance mapping.

Let P be a basic graph pattern

- 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 μ:

- dom(μ) = var(P)
- there exists an instance mapping σ 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 $\{t\}$

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

	?X	?Y
μ_1:	R_1	john
μ_2:	R_2	paul

Semantics of basic graph patterns: Some examples

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

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

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

Semantics of basic graph patterns: Some examples

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

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

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

Semantics of basic graph patterns: Some examples

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

graph	bgp	evaluation
(R_1, name, john) (R_1, email, J@ed.ex) (R_2, name, paul)	(?X, name, ?Y)	see table below
(R_1, name, john) (R_1, email, J@ed.ex) (R_2, name, paul)	(?X, name, _:b)	see table below

	?X	?Y
μ_1:	R_1	john
μ_2:	R_2	paul

	?X
μ_3:	R_1
μ_4:	R_2

Compatible mappings

Definition

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

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

Example

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

Compatible mappings

Definition

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

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

Example

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

Compatible mappings

Definition

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

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

Example

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

Compatible mappings

Definition

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

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

Example

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

Compatible mappings

Definition

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

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

Example

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

Compatible mappings

Definition

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

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

Example

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

- μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let Ω_1 and Ω_2 be sets of mappings.

Definition

Join: extends mappings in Ω_1 with compatible mappings in Ω_2

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

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

- $\Omega_1 \smallsetminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid$ there is no mapping in Ω_2 compatible with $\mu_1\}$

Sets of mappings and operations

Definition

Union: includes mappings in Ω_1 and in Ω_2

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

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

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

Semantics of SPARQL: AND, UNION, OPT and SELECT

Given an RDF graph G

Definition

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

Semantics of SPARQL: AND, UNION, OPT and SELECT

Given an RDF graph G

Definition

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

Semantics of SPARQL: AND, UNION, OPT and SELECT

Given an RDF graph G

Definition

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

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

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

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

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

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

Semantics of SPARQL: An example

Example

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

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

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

Semantics of SPARQL: An example

Example

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

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

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

Semantics of SPARQL: An example

Example

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

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

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

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

Semantics of SPARQL: An example

Example

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

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

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

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

Semantics of SPARQL: An example

Example

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

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

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

$?X$	$?Y$	$?E$

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

Semantics of SPARQL: An example

Example

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

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

?X	?Y
R_1	john
R_2	paul

?X	?Y	?E
R_1	john	J@ed.ex

?X	?E
R_1	J@ed.ex

- from the Join

Semantics of SPARQL: An example

Example

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

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

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

$?X$	$?Y$	$?E$
R_2	paul	

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

- from the Difference

Semantics of SPARQL: An example

Example

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

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

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

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

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

- from the Union

Filter expressions (value constraints)

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

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

We consider in R:

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

Satisfaction of value constraints

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

Satisfaction of value constraints

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

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

Satisfaction of value constraints

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

- R is $?X = c$, $?X \in \text{dom}(\mu)$ 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 μ satisfies a condition R ($\mu \models R$) if:

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

Definition

FILTER : selects mappings that satisfy a condition

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

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 was released in 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

Syntax of RDFS

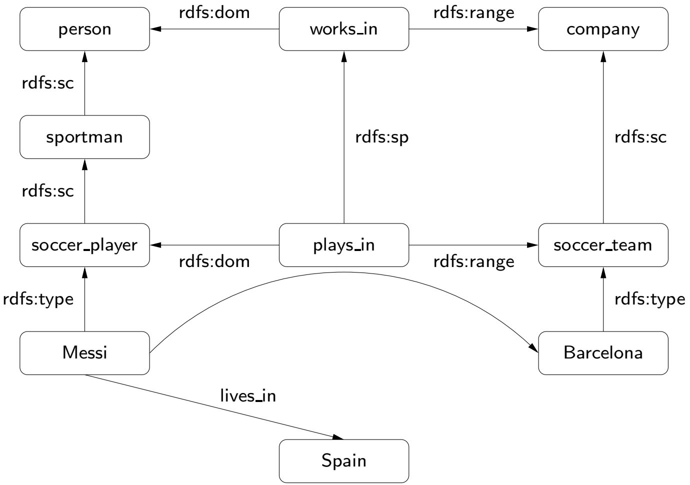
RDFS extends RDF with a schema vocabulary: subPropertyOf (rdfs:sp), subClassOf (rdfs:sc), domain (rdfs:dom), range (rdfs:range), type (rdfs:type).

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: (Messi, rdfs:type, person)



Semantics of RDFS

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

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

The notion of entailment in RDFS can be defined as for first-order logic.

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

An inference system for RDFS

Sub-property :
$$\frac{(\mathcal{A},\ \text{rdfs:sp},\ \mathcal{B})\ (\mathcal{B},\ \text{rdfs:sp},\ \mathcal{C})}{(\mathcal{A},\ \text{rdfs:sp},\ \mathcal{C})}$$

$$\frac{(\mathcal{A},\ \text{rdfs:sp},\ \mathcal{B})\ (\mathcal{X},\ \mathcal{A},\ \mathcal{Y})}{(\mathcal{X},\ \mathcal{B},\ \mathcal{Y})}$$

Subclass :
$$\frac{(\mathcal{A},\ \text{rdfs:sc},\ \mathcal{B})\ (\mathcal{B},\ \text{rdfs:sc},\ \mathcal{C})}{(\mathcal{A},\ \text{rdfs:sc},\ \mathcal{C})}$$

$$\frac{(\mathcal{A},\ \text{rdfs:sc},\ \mathcal{B})\ (\mathcal{X},\ \text{rdfs:type},\ \mathcal{A})}{(\mathcal{X},\ \text{rdfs:type},\ \mathcal{B})}$$

Typing :
$$\frac{(\mathcal{A},\ \text{rdfs:dom},\ \mathcal{B})\ (\mathcal{X},\ \mathcal{A},\ \mathcal{Y})}{(\mathcal{X},\ \text{rdfs:type},\ \mathcal{B})}$$

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

Entailment in RDFS

Theorem (H03,MPG09,GHM11)

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

Thus, a triple t can be deduced from an RDF graph G ($G \models t$) iff t can be deduced from G by applying the inference rules a finite number of times.

An entailment regime for RDFS in SPARQL 1.1

Basic graph patterns are evaluated by considering RDFS entailment.

Definition

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

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

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 μ:

- $\text{dom}(\mu) = \text{var}(P)$
- there exists an instance mapping σ 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: $(?X, \text{rdfs:sc}, \text{person})$

Entailment regimes in SPARQL 1.1: Some observations

- SPARQL 1.1 can be used to query not only data but also schema information
  - For example: $(?X, \text{rdfs:sc}, \text{person})$
- 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

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

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

Example

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

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

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

- 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

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:

$$[\![(P_1 \text{ MINUS } P_2)]\!]_G \quad = \quad [\![P_1]\!]_G \smallsetminus [\![P_2]\!]_G$$

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:

$$[\![(P_1 \text{ MINUS } P_2)]\!]_G \quad = \quad [\![P_1]\!]_G \smallsetminus [\![P_2]\!]_G$$

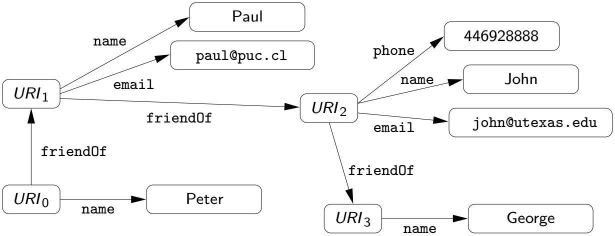
Open issues

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

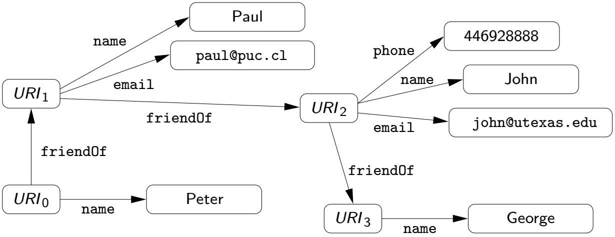
Outline of the talk

- 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

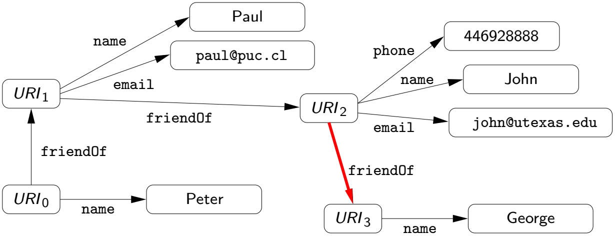


SPARQL provides limited navigational capabilities



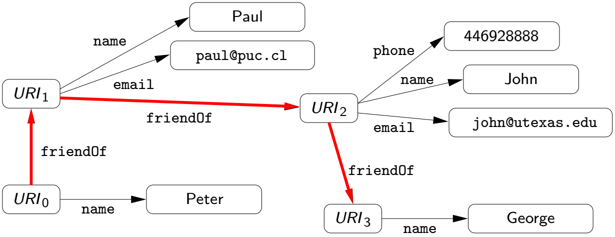
(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))

SPARQL provides limited navigational capabilities



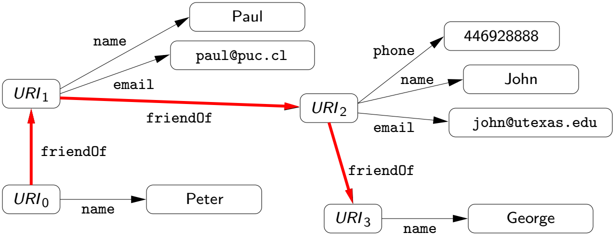
(SELECT $?X$ (($?X$, friendOf, $?Y$) AND ($?Y$, name, George)))

SPARQL provides limited navigational capabilities

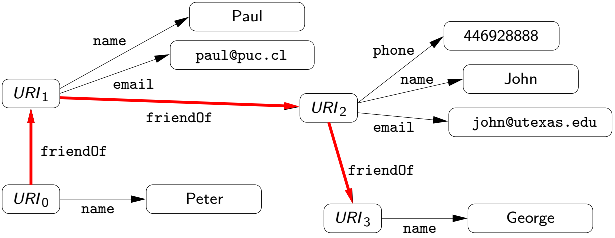


(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))

A possible solution: Property paths



A possible solution: Property paths



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

Navigational capabilities in SPARQL 1.1: Property paths

Syntax of property paths:

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

where $a \in \mathbf{I}$

Navigational capabilities in SPARQL 1.1: Property paths

Syntax of property paths:

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

where $a \in \mathbf{I}$

Other expressions are allowed:

$\hat{}\,exp$	:	inverse path
$!(a_1 \mid \ldots \mid a_n)$	:	an IRI which is not one of a_i $(1 \leq i \leq n)$

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

Evaluating property paths

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

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

Evaluating property paths

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

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

Evaluating property paths

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

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

Evaluating property paths

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

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

Property paths in SPARQL 1.1

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

- *exp* is a property path
- x (resp. y) is either an element from $\mathbf{I}$ 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 **I** or a variable

Example

- $(?X$, (rdfs:sc)*, person): Verifies whether the value stored in $?X$ is a subclass of person
- $(?X$, (rdfs:sp)*, $?Y)$: Verifies whether the value stored in $?X$ is a subproperty of the value stored in $?Y$

Semantics of property paths

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

Semantics of property paths

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

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

Semantics of property paths

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

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

Other cases are defined analogously.

Semantics of property paths

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

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

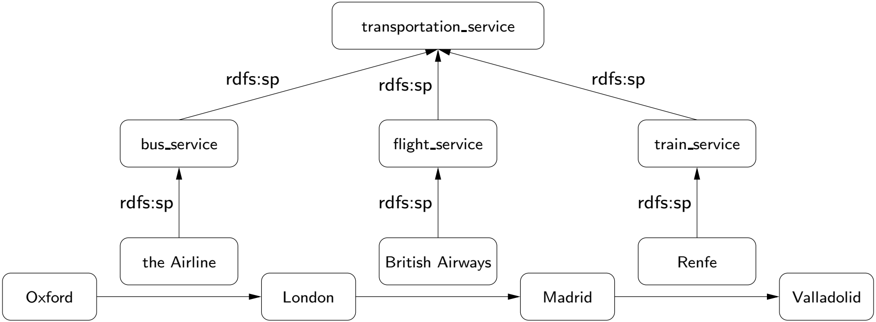
Other cases are defined analogously.

Example

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

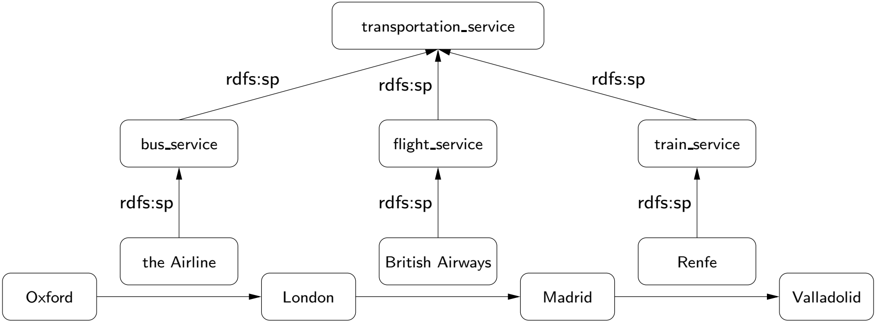
SPARQL 1.1: Entailment regimes and property paths

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



SPARQL 1.1: Entailment regimes and property paths

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

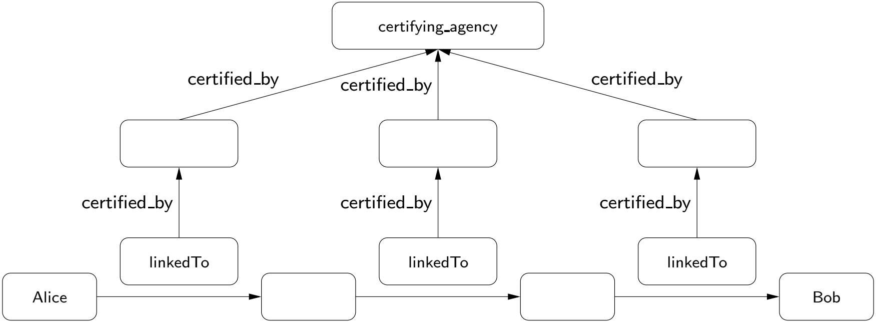


In SPARQL 1.1: $(?X, \text{transportation\_service}^*, ?Y)$

Navigational capabilities in SPARQL 1.1: Some observations

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

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



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

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

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

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

Open issues

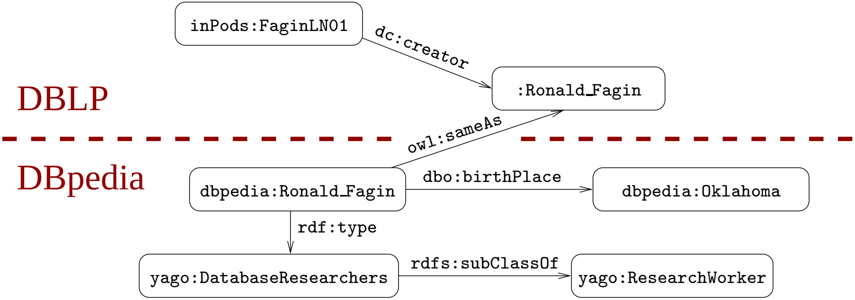
- How can OWL entailment be handled in these proposals?
- What navigational capabilities should be added to SPARQL 1.1?
- There is a need for query languages that can return paths

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

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



Querying interconnected RDF graphs

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

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:PartOf     ?Conf .
  ?Conf       swrc:series    conf:pods .
  SERVICE <http://dbpedia.org/sparql> {
    ?Person      owl:sameAs      ?Author .
    ?Person      dbo:birthPlace dbpedia:Oklahoma . }
}
```

Federation in SPARQL 1.1

New rule to generate graph patterns:

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

Federation in SPARQL 1.1

New rule to generate graph patterns:

- If P is a graph pattern and $c \in (\mathbf{I} \cup \mathbf{V})$, 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 $c \in \mathbf{I}$
- (SERVICE $?X$ P) is allowed in the official specification of SPARQL 1.1, but its semantics is not defined

Semantics of SERVICE

ep$(\cdot)$: Partial function from $\mathbf{I}$ 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 **I** 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 $c \in \mathbf{I} \setminus \text{dom}(\text{ep})$, then $[\![P]\!]_G = \{\mu_\emptyset\}$ (where $\mu_\emptyset$ is the mapping with empty domain)
- if $c \in \mathbf{V}$, then

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

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

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:

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

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

- Can this strategy be generalized?

How can we evaluate SERVICE queries?

We need some notion of boundedness

- A variable $?X$ is bound in a graph pattern P if for every RDF graph G and every $\mu \in [\![P]\!]_G$, it holds that $?X \in \text{dom}(\mu)$ and $\mu(?X)$ 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
  $(?X, \text{service\_address}, ?Y)$ AND (SERVICE $?Y$ $(?N, \text{email}, ?E)$)

The first attempt: Too restrictive

Consider the query:

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

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

The first attempt: Not appropriate for nested SERVICE operators

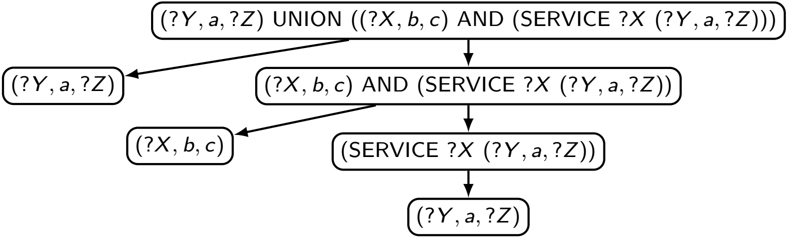
Consider the query:

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

Solving the problems . . .

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

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



A more appropriate notion of boundedness

Definition (BACP13)

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

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

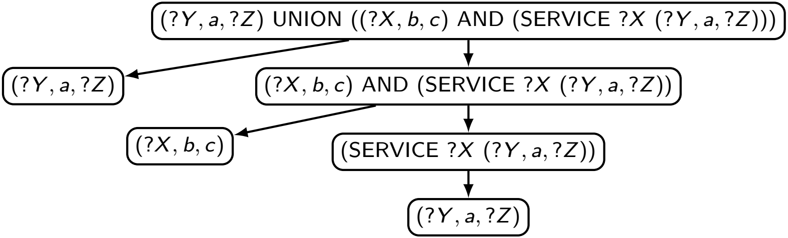
A more appropriate notion of boundedness

Definition (BACP13)

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

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

Examples:



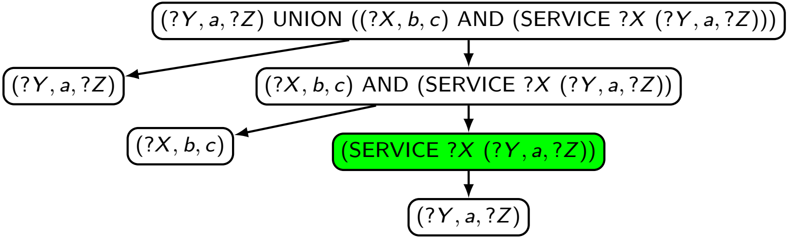
A more appropriate notion of boundedness

Definition (BACP13)

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

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

Examples:



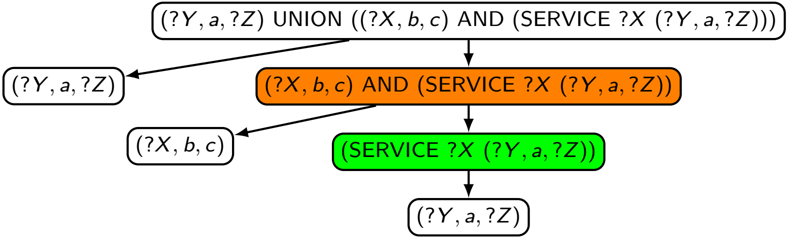
A more appropriate notion of boundedness

Definition (BACP13)

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

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

Examples:



A more appropriate notion of boundedness

Definition (BACP13)

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

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

Examples:

$(?Y, a, ?Z)$

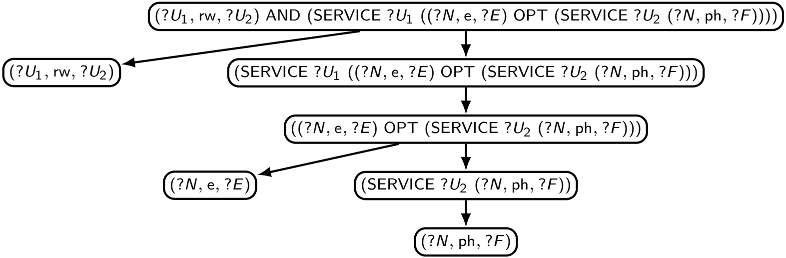
A more appropriate notion of boundedness

Definition (BACP13)

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

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

Examples:



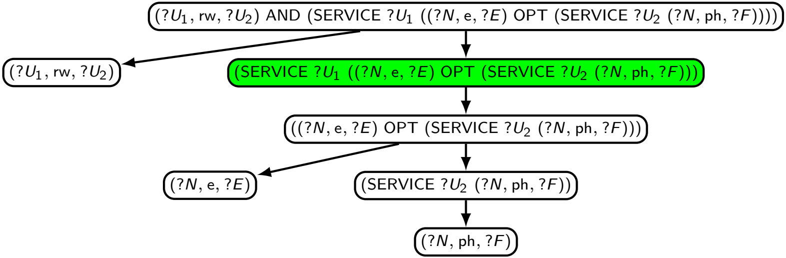
A more appropriate notion of boundedness

Definition (BACP13)

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

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

Examples:



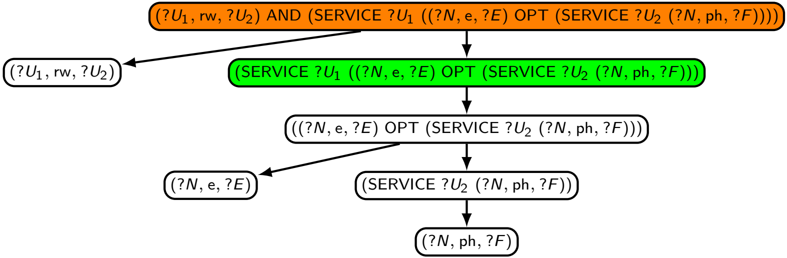
A more appropriate notion of boundedness

Definition (BACP13)

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

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

Examples:

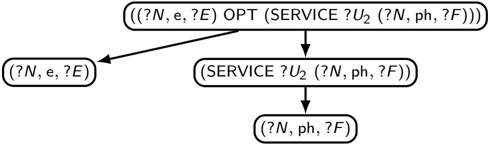


Definition (BACP13)

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

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

Examples:



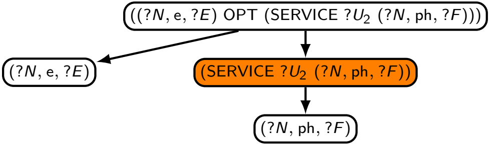
A more appropriate notion of boundedness

Definition (BACP13)

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

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

Examples:



A more appropriate notion of boundedness (cont'd)

But we still have a problem:

Proposition (BACP13)

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

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

An appropriate notion: Service-safeness

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

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

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

Definition (BACP13)

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

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

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

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

Definition (BACP13)

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

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

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

Open issue

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

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!

Bibliography

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

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

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

Bibliography (cont'd)

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

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

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

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