

Navigation in SPARQL 1.1

Marcelo Arenas

PUC Chile & U. of Oxford

Oxford, November 2012

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**

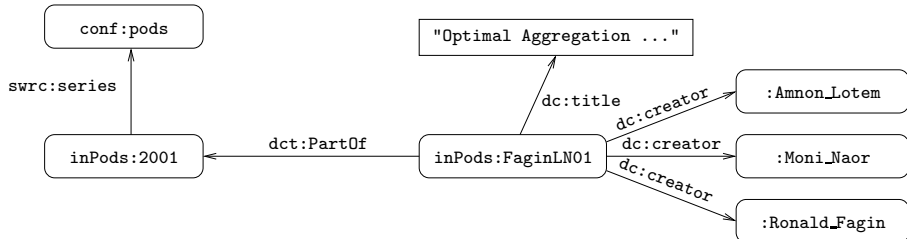
RDF in a nutshell

RDF is the framework proposed by the W3C to represent information in the Web:

- ▶ URI vocabulary
 - ▶ A URI is an atomic piece of data, and it identifies an abstract resource
- ▶ Syntax based on directed labeled graphs
 - ▶ URIs are used as node labels and edge labels
- ▶ Schema definition language (**RDFS**): Define new vocabulary
 - ▶ Typing, inheritance of classes and properties, ...

An example of an RDF graph: DBLP

```
    : <http://dblp.13s.de/d2r/resource/authors/>  
  conf: <http://dblp.13s.de/d2r/resource/conferences/>  
inPods: <http://dblp.13s.de/d2r/resource/publications/conf/pods/>  
  swrc: <http://swrc.ontoware.org/ontology#>  
    dc: <http://purl.org/dc/elements/1.1/>  
    dct: <http://purl.org/dc/terms/>
```



An example of a URI

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



The screenshot shows a web browser window with the address bar containing the URI `http://dblp.l3s.de/d2r/page/conferences/pods`. The page title is "PODS | D2R Server publishing the". Below the address bar, there are navigation buttons and a search bar. The main content area is green and displays the text "Resource URI: http://". Below this, there are two blue links: "Home" and "Example Conferences". At the bottom, there is a table with two columns: "Property" and "Value".

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

Querying RDF

Why is this an interesting problem? Why is it challenging?

- ▶ RDF graphs can be interconnected
 - ▶ URIs should be dereferenceable
- ▶ Semantics of RDF is open world
 - ▶ RDF graphs are inherently incomplete
 - ▶ The possibility of adding optional information if present is an important feature
- ▶ Vocabulary with predefined semantics
- ▶ Navigational capabilities are needed

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

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

A SPARQL query consists of a:

SPARQL in a nutshell

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

A SPARQL query consists of a:

Body: Pattern matching expression

SPARQL in a nutshell

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

A SPARQL query consists of a:

Body: Pattern matching expression

Head: Processing of the variables

What are the challenges in implementing SPARQL?

SPARQL has to take into account the distinctive features of RDF:

- ▶ Should be able to extract information from interconnected RDF graphs
- ▶ Should be consistent with the open-world semantics of RDF
 - ▶ Should offer the possibility of adding optional information if present
- ▶ Should be able to properly interpret RDF graphs with a vocabulary with predefined semantics
- ▶ Should offer some functionalities for navigating in an RDF graph

What are the challenges in implementing SPARQL?

SPARQL has to take into account the distinctive features of RDF:

- ▶ Should be able to extract information from interconnected RDF graphs
- ▶ Should be consistent with the open-world semantics of RDF
 - ▶ Should offer the possibility of adding optional information if present
- ▶ Should be able to properly interpret RDF graphs with a vocabulary with predefined semantics
- ▶ Should offer some functionalities for navigating in an RDF graph

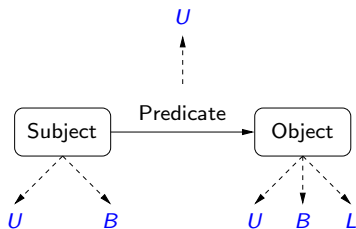
Outline

- ▶ RDF and SPARQL
- ▶ Navigation in SPARQL 1.1: Property paths
 - ▶ Syntax and semantics
- ▶ Our contributions:
 - ▶ Experimental evaluation
 - ▶ Study of the complexity of evaluating property paths
- ▶ Final remarks

Outline

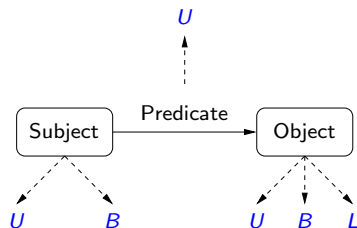
- ▶ RDF and SPARQL
- ▶ Navigation in SPARQL 1.1: Property paths
 - ▶ Syntax and semantics
- ▶ Our contributions:
 - ▶ Experimental evaluation
 - ▶ Study of the complexity of evaluating property paths
- ▶ Final remarks

RDF formal model



- U** : set of URIs
- B** : set of blank nodes
- L** : set of literals

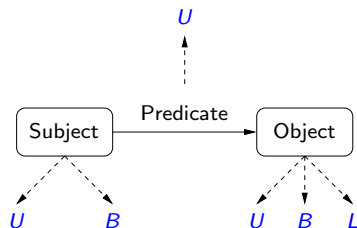
RDF formal model



- U** : set of URIs
- B** : set of blank nodes
- L** : set of literals

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

RDF formal model



- U** : set of URIs
- B** : set of blank nodes
- L** : set of literals

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

A finite set of RDF triples is called an **RDF graph**

RDF formal model

Proviso

In this talk, we do not consider blank nodes

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

SPARQL: An algebraic syntax

▶ Graph pattern:

`?X name ?Y`

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

`{ P1 . P2 }`

$(P_1 \text{ AND } P_2)$

`{ P1 OPTIONAL { P2 } }`

$(P_1 \text{ OPT } P_2)$

`{ P1 } UNION { P2 }`

$(P_1 \text{ UNION } P_2)$

`{ P1 FILTER (R) }`

$(P_1 \text{ FILTER } R)$

▶ SPARQL query:

`SELECT ?X ?Y ... { P }`

$(\text{SELECT } \{?X, ?Y, \dots\} P)$

Filter expressions (value constraints)

Filter expression: (P FILTER R)

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

We consider in R :

- ▶ equality = among variables and elements from **U** and **L**
- ▶ unary predicate $\text{bound}(\cdot)$
- ▶ boolean combinations (\wedge , \vee , \neg)

Mappings: building block for the semantics

Definition

A mapping is a partial function:

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

The evaluation of a SPARQL query results in a set of mappings

Compatible mappings

Definition

Mappings μ_1 and μ_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

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

Compatible mappings

Definition

Mappings μ_1 and μ_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

	?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:

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

► μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings.

Definition

Join: extends mappings in M_1 with compatible mappings in M_2

- ▶ $M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2 \text{ and } \mu_1, \mu_2 \text{ are compatible}\}$

Difference: selects mappings in M_1 that cannot be extended with mappings in M_2

- ▶ $M_1 \setminus M_2 = \{\mu_1 \in M_1 \mid \text{there is no mapping in } M_2 \text{ compatible with } \mu_1\}$

Sets of mappings and operations

Definition

Union: includes mappings in M_1 and in M_2

$$\blacktriangleright M_1 \cup M_2 = \{\mu \mid \mu \in M_1 \text{ or } \mu \in M_2\}$$

Left Outer Join: extends mappings in M_1 with compatible mappings in M_2 **if possible**

$$\blacktriangleright M_1 \bowtie M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2)$$

Semantics of SPARQL

Given an RDF graph G .

Definition

$$\llbracket t \rrbracket_G = \{ \mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in G \}$$

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

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

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

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

Semantics of SPARQL

Given an RDF graph G .

Definition

$$\llbracket t \rrbracket_G = \{ \mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in G \}$$

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

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

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

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

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

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

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

Definition

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

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

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

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

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

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

?X	?Y
R_1	john
R_2	paul

Semantics of SPARQL: An example

Example

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

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

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

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

?X	?Y
R_1	john
R_2	paul

Semantics of SPARQL: An example

Example

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

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

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

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

?X	?Y
R_1	john
R_2	paul

?X	?E
R_1	J@ed.ex

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

?X	?E
R_1	J@ed.ex

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

?X	?Y	?E

?X	?E
R_1	J@ed.ex

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

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

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

▶ from the **join**

Semantics of SPARQL: An example

Example

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

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

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

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

?X	?Y
R_1	john
R_2	paul

?X	?Y	?E
R_2	paul	

?X	?E
R_1	J@ed.ex

- ▶ from the **difference**

Semantics of SPARQL: An example

Example

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

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

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

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

?X	?Y
R_1	john
R_2	paul

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

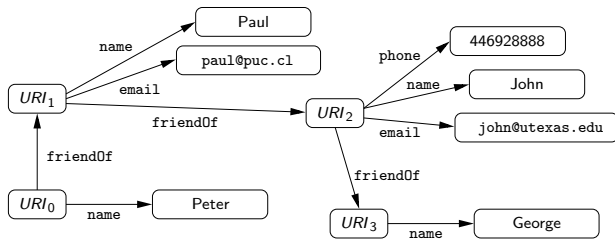
?X	?E
R_1	J@ed.ex

▶ from the **union**

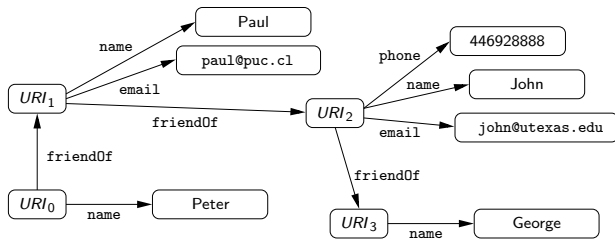
Outline

- ▶ RDF and SPARQL
- ▶ Navigation in SPARQL 1.1: Property paths
 - ▶ Syntax and semantics
- ▶ Our contributions:
 - ▶ Experimental evaluation
 - ▶ Study of the complexity of evaluating property paths
- ▶ Final remarks

SPARQL 1.0 provides limited navigational capabilities

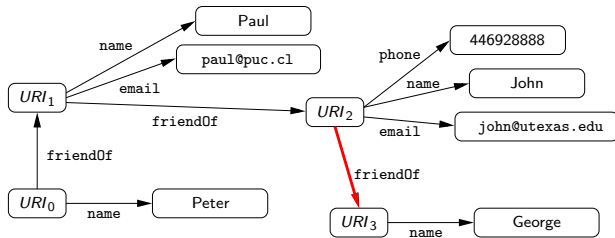


SPARQL 1.0 provides limited navigational capabilities



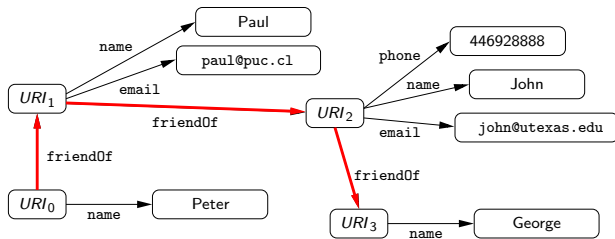
```
(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))
```

SPARQL 1.0 provides limited navigational capabilities



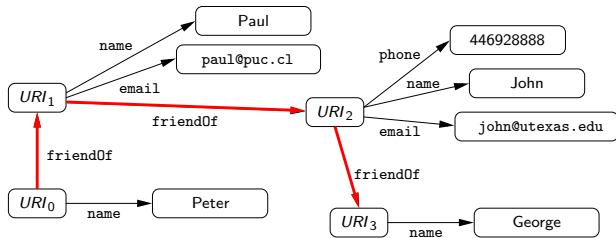
```
(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))
```

SPARQL 1.0 provides limited navigational capabilities

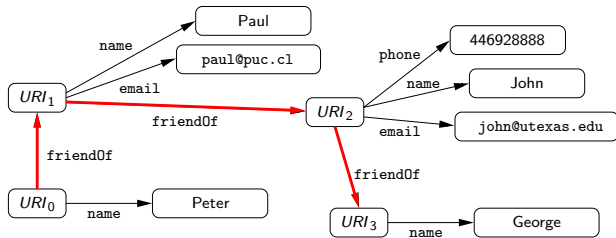


```
(SELECT ?X ((?X, friendOf, ?Y) AND (?Y, name, George)))
```

A possible solution: Regular expressions in graph databases



A possible solution: Regular expressions in graph databases



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

Syntax and semantics of property paths

Syntax: Property paths are regular expressions (`/`, `|`, `*`)

Semantics: Repeated values are needed in some use cases

Syntax and semantics of property paths

Syntax: Property paths are regular expressions (`/`, `|`, `*`)

Semantics: Repeated values are needed in some use cases

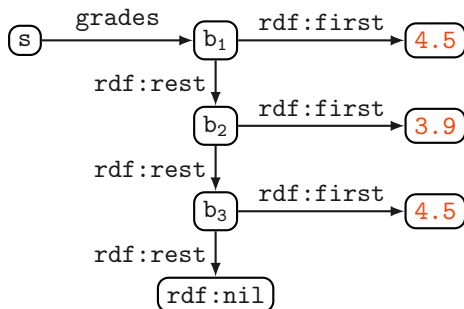
- ▶ SPARQL uses a bag semantics: **Duplicates are not eliminated**

Syntax and semantics of property paths

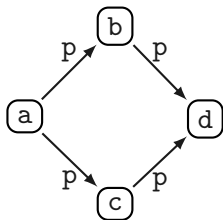
Syntax: Property paths are regular expressions (`/`, `|`, `*`)

Semantics: Repeated values are needed in some use cases

- ▶ SPARQL uses a bag semantics: **Duplicates are not eliminated**
- ▶ Use case for property paths: Retrieving the elements of a linked list

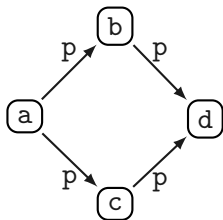


Property paths are designed to count



$(a, p^*, ?X)$

Property paths are designed to count



$(a, p^*, ?X)$

$\frac{?X}{a}$
b
c
d
d

Definition of the semantics of property paths

$(?X, (path_1/path_2), ?Y)$ is replaced by:

$(SELECT \{?X, ?Y\} ((?X, path_1, ?Z) AND (?Z, path_2, ?Y)))$

Definition of the semantics of property paths

$(?X, (path_1/path_2), ?Y)$ is replaced by:

$(SELECT \{?X, ?Y\} ((?X, path_1, ?Z) AND (?Z, path_2, ?Y)))$

$(?X, (path_1|path_2), ?Y)$ is replaced by:

$((?X, path_1, ?Y) UNION (?X, path_2, ?Y))$

Definition of the semantics of property paths

$(?X, (path_1/path_2), ?Y)$ is replaced by:

$(SELECT \{?X, ?Y\} ((?X, path_1, ?Z) AND (?Z, path_2, ?Y)))$

$(?X, (path_1|path_2), ?Y)$ is replaced by:

$((?X, path_1, ?Y) UNION (?X, path_2, ?Y))$

But how do we evaluate $*$?

- ▶ How do we deal with cycles?

Definition of the semantics of *

Evaluation of *path**

*“the algorithm extends the multiset of results by one application of **path**. If a node has been visited for **path**, it is not a candidate for another step. A node can be visited multiple times if different paths visit it.”*

W3C Working Draft (January 5, 2012)

Definition of the semantics of *

Evaluation of *path**

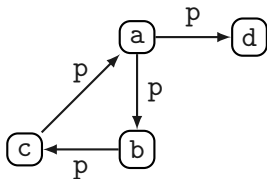
*“the algorithm extends the multiset of results by one application of *path*. If a node has been visited for *path*, it is not a candidate for another step. A node can be visited multiple times if different paths visit it.”*

W3C Working Draft (January 5, 2012)

- ▶ SPARQL 1.1 specification provides a special (recursive) procedure to handle cycles and make the count

The procedure in a nutshell

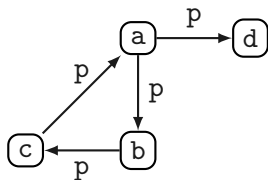
RDF Graph G :



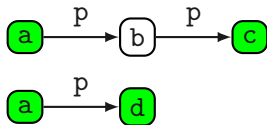
Evaluation of $(?X, p^*, ?Y)$:

The procedure in a nutshell

RDF Graph G :

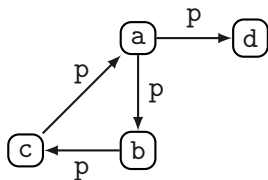


Evaluation of $(?X, p^*, ?Y)$:

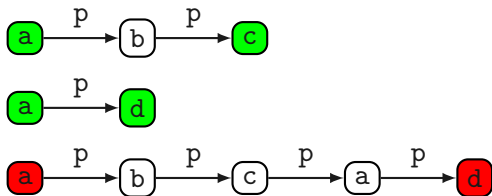


The procedure in a nutshell

RDF Graph G :

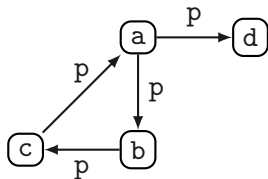


Evaluation of $(?X, p^*, ?Y)$:



The procedure in a nutshell

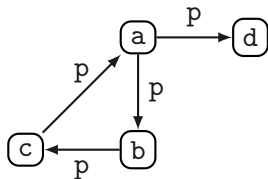
RDF Graph G :



Evaluation of $(?X, (p^*)^*, ?Y)$:

The procedure in a nutshell

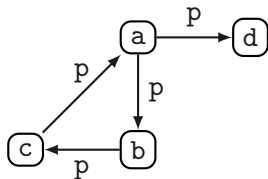
RDF Graph G :



Evaluation of $(?X, (p^*)^*, ?Y)$:

The procedure in a nutshell

RDF Graph G :

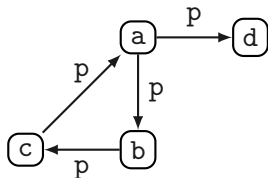


Evaluation of $(?X, (p^*)^*, ?Y)$:

Evaluation of p^* :

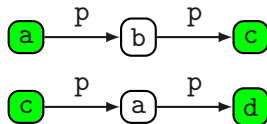
The procedure in a nutshell

RDF Graph G :



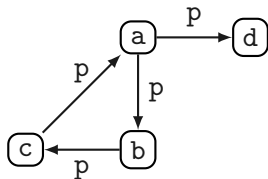
Evaluation of $(?X, (p^*)^*, ?Y)$:

Evaluation of p^* :

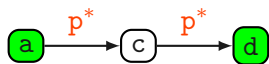


The procedure in a nutshell

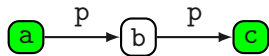
RDF Graph G :



Evaluation of $(?X, (p^*)^*, ?Y)$:

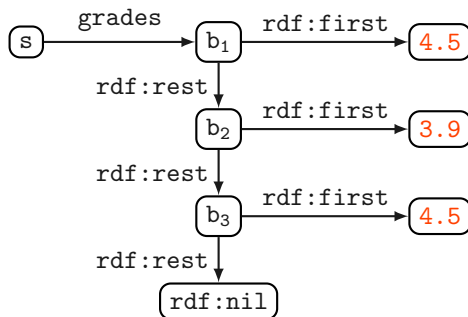


Evaluation of p^* :



Is this a good semantics?

Linked list example:



$(s, \text{grades}/\text{rdf:rest}^*/\text{rdf:first}, ?X)$

Outline

- ▶ RDF and SPARQL
- ▶ Navigation in SPARQL 1.1: Property paths
 - ▶ Syntax and semantics
- ▶ **Our contributions:**
 - ▶ **Experimental evaluation**
 - ▶ Study of the complexity of evaluating property paths
- ▶ Final remarks

Our contributions in [ACP12]

- ▶ Experimental evaluation of the main implementations of SPARQL 1.1 including property paths
- ▶ Complete study of the complexity of path evaluation
- ▶ Identification of the main sources of complexity (counting!)
- ▶ Proposal of a semantics that can be efficiently evaluated

Our contributions in [ACP12]

- ▶ Experimental evaluation of the main implementations of SPARQL 1.1 including property paths
- ▶ Complete study of the complexity of path evaluation
- ▶ Identification of the main sources of complexity (counting!)
- ▶ Proposal of a semantics that can be efficiently evaluated

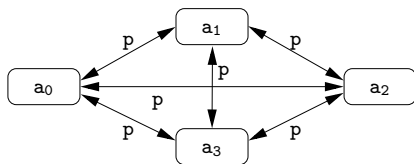
Impact on W3C standard:

- ▶ Normative semantics of SPARQL 1.1 property paths was changed to overcome the issues raised in [KM12] and in our study.

Some experimental results with synthetic data

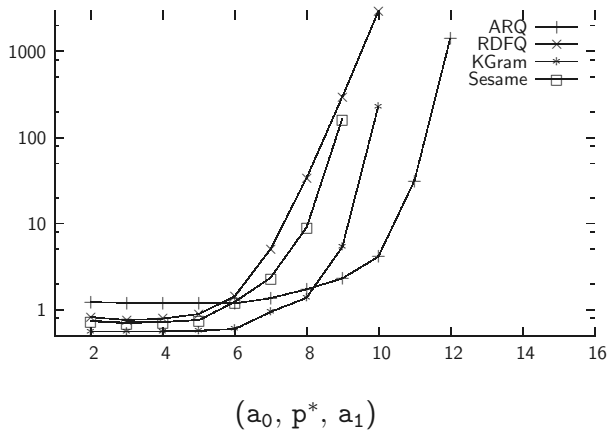
Data:

- ▶ cliques (complete graphs) of different size
- ▶ from 2 nodes (87 bytes) to 13 nodes (970 bytes)



RDF clique with 4 nodes (127 bytes)

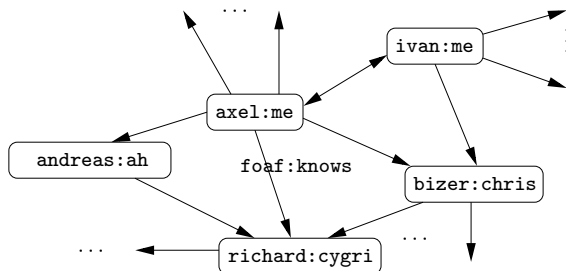
Some experimental results with synthetic data



Some experimental results with real data

Data:

- ▶ Social Network data given by `foaf:knows` links
- ▶ Crawled from Axel Polleres' foaf document (3 steps)
- ▶ Different documents, deleting some nodes



Some experimental results with real data

(axel:me, foaf:knows*, ?X)

Some experimental results with real data

(axel:me, foaf:knows*, ?X)

Input	ARQ	RDFQ	Kgram	Sesame
9.2KB	5.13	75.70	313.37	-
10.9KB	8.20	325.83	-	-
11.4KB	65.87	-	-	-
13.2KB	292.43	-	-	-
14.8KB	-	-	-	-
17.2KB	-	-	-	-
20.5KB	-	-	-	-
25.8KB	-	-	-	-

(time in seconds, timeout = 1hr)

Counting the number of solutions

Data: Clique of size n

$$(a_0, p^*, a_1)$$

every solution is a copy of the empty mapping ($| \cdot |$ in ARQ)

Counting the number of solutions

Data: Clique of size n

(a_0, p^*, a_1)

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

every solution is a copy of the empty mapping ($| \ |$ in ARQ)

Counting the number of solutions

Data: Clique of size n

(a_0, p^*, a_1)

$(a_0, (p^*)^*, a_1)$

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

every solution is a copy of the empty mapping ($| \ |$ in ARQ)

Counting the number of solutions

Data: Clique of size n

(a_0, p^*, a_1)

$(a_0, (p^*)^*, a_1)$

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

every solution is a copy of the empty mapping ($|$ $|$ in ARQ)

Counting the number of solutions

Data: Clique of size n

(a_0, p^*, a_1)

$(a_0, (p^*)^*, a_1)$

$(a_0, ((p^*)^*)^*, a_1)$

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

every solution is a copy of the empty mapping (| | in ARQ)

Counting the number of solutions

Data: Clique of size n

(a_0, p^*, a_1)

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

$(a_0, (p^*)^*, a_1)$

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

$(a_0, ((p^*)^*)^*, a_1)$

n	# Sol.
2	1
3	42
4	–

every solution is a copy of the empty mapping (| | in ARQ)

Counting the number of solutions (cont.)

Data: foaf links crawled from the Web

`(axel:me, foaf:knows*, ?X)`

Counting the number of solutions (cont.)

Data: foaf links crawled from the Web

(axel:me, foaf:knows*, ?X)

File	# URIs	# Sol.	Output Size
9.2KB	38	29,817	2MB
10.9KB	43	122,631	8.4MB
11.4KB	47	1,739,331	120MB
13.2KB	52	8,511,943	587MB
14.8KB	54	-	-

Counting the number of solutions (cont.)

Data: foaf links crawled from the Web

(axel:me, foaf:knows*, ?X)

File	# URIs	# Sol.	Output Size
9.2KB	38	29,817	2MB
10.9KB	43	122,631	8.4MB
11.4KB	47	1,739,331	120MB
13.2KB	52	8,511,943	587MB
14.8KB	54	-	-

What is going on?

Outline

- ▶ RDF and SPARQL
- ▶ Navigation in SPARQL 1.1: Property paths
 - ▶ Syntax and semantics
- ▶ Our contributions:
 - ▶ Experimental evaluation
 - ▶ Study of the complexity of evaluating property paths
- ▶ Final remarks

Counting problem for property paths

COUNTW3C

Input: RDF graph G
Property path triple (a, \textit{path}, b)

Output: Count the number of solutions of (a, \textit{path}, b) over G
(according to the semantics proposed by the W3C)

A double-exponential lower bound for counting

- ▶ Let π_s be a property path of the form

$$(\dots((p^*)^*)^* \dots)^*$$

with s nested stars

- ▶ Let K_n be a clique with n nodes
- ▶ Let $\text{CountClique}(s, n)$ be the number of solutions of (a_0, π_s, a_1) over K_n

A double-exponential lower bound for counting

- ▶ Let π_s be a property path of the form

$$(\dots((p^*)^*)^*\dots)^*$$

with s nested stars

- ▶ Let K_n be a clique with n nodes
- ▶ Let $\text{CountClique}(s, n)$ be the number of solutions of (a_0, π_s, a_1) over K_n

Lemma (ACP12)

$$\text{CountClique}(s, n) \geq (n-2)!^{(n-1)^{s-1}}$$

A double-exponential lower bound for counting

- ▶ Let π_s be a property path of the form

$$(\dots((p^*)^*)^*\dots)^*$$

with s nested stars

- ▶ Let K_n be a clique with n nodes
- ▶ Let $CountClique(s, n)$ be the number of solutions of (a_0, π_s, a_1) over K_n

Lemma (ACP12)

$$CountClique(s, n) \geq (n-2)!^{(n-1)^{s-1}}$$

In fact, there is a recursive formula for calculating $CountClique(s, n)$

We can now explain our experimental results

$\text{CountClique}(s, n)$ allows us to fill in the blanks

$$(a_0, (p^*)^*, a_1)$$

n	# Sol.
2	1
3	6
4	305
5	418,576
6	—
7	—
8	—

We can now explain our experimental results

$\text{CountClique}(s, n)$ allows us to fill in the blanks

$$(a_0, (p^*)^*, a_1)$$

n	# Sol.	
2	1	✓
3	6	✓
4	305	✓
5	418,576	✓
6	—	
7	—	
8	—	

We can now explain our experimental results

$\text{CountClique}(s, n)$ allows us to fill in the blanks

$$(a_0, (p^*)^*, a_1)$$

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	—	←	28×10^9
7	—	←	144×10^{15}
8	—	←	79×10^{24}

We can now explain our experimental results

$\text{CountClique}(s, n)$ allows us to fill in the blanks

$$(a_0, (p^*)^*, a_1)$$

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	—	←	28×10^9
7	—	←	144×10^{15}
8	—	←	79×10^{24}

79 Yottabytes for the answer over a file of 379 bytes

- ▶ 1 Yottabyte > the estimated capacity of all digital storage in the world

What about data complexity?

Common assumption in Databases: Queries are much smaller than data sources

Data complexity

- ▶ Measure the complexity considering the query fixed
- ▶ Data complexity of SPARQL is polynomial

What about data complexity?

Common assumption in Databases: Queries are much smaller than data sources

Data complexity

- ▶ Measure the complexity considering the query fixed
- ▶ Data complexity of SPARQL is polynomial

In our setting:

COUNTW3C(*path*)

Input: RDF graph G and $a, b \in \mathbf{U}$

Output: Count the number of solutions of (a, \textit{path}, b) over G

A bit on complexity classes . . .

We measure the complexity by using *counting-complexity classes*

NP

SAT: is a propositional
formula satisfiable?

#P

COUNTSAT: how many assignments
satisfy a propositional formula?

A bit on complexity classes ...

We measure the complexity by using *counting-complexity classes*

NP

#P

SAT: is a propositional formula satisfiable?

COUNTSAT: how many assignments satisfy a propositional formula?

Definition

A function $f(\cdot)$ is in #P if there exists a polynomial-time non-deterministic TM M such that:

$f(x)$ = number of accepting computations of M with input x

A bit on complexity classes ...

We measure the complexity by using *counting-complexity classes*

NP

#P

SAT: is a propositional formula satisfiable?

COUNTSAT: how many assignments satisfy a propositional formula?

Definition

A function $f(\cdot)$ is in #P if there exists a polynomial-time non-deterministic TM M such that:

$f(x)$ = number of accepting computations of M with input x

- ▶ COUNTSAT is #P-complete

Data complexity of property paths is intractable

Theorem (ACP12)

- ▶ For every property path π : $\text{COUNTW3C}(\pi)$ is in $\#P$
- ▶ $\text{COUNTW3C}(a^*)$ is $\#P$ -hard, where $a \in \mathbf{U}$

An alternative semantics: Simple paths

A simple path is a path without repeated vertices

- ▶ Cycles are not allowed

An alternative to the W3C semantics: Count only the **simple paths** satisfying a property path expression

An alternative semantics: Simple paths

A simple path is a path without repeated vertices

- ▶ Cycles are not allowed

An alternative to the W3C semantics: Count only the **simple paths** satisfying a property path expression

Theorem (LM12,ACP12)

- ▶ COUNTSIMPLEPATH is $\#P$ -complete
- ▶ In fact, $\text{COUNTSIMPLEPATH}(a^*)$ is $\#P$ -hard, where $a \in \mathbf{U}$

A more fundamental result

In an acyclic RDF graph G , the previous two notions of paths coincide with the usual notion of path.

- ▶ A reasonable notion of path should satisfy this condition

A fundamental problem to study:

COUNTPATH

Input: Acyclic RDF graph G
Property path triple $(a, path, b)$

Output: Count the number of (usual) paths from a to b in G that conform to $path$

A bit more on complexity classes ...

Definition

- ▶ A function $f(\cdot)$ is in $\#L$ if there exists a logarithmic-space non-deterministic TM M such that:

$f(x) =$ number of accepting computations of M with input x

A bit more on complexity classes ...

Definition

- ▶ A function $f(\cdot)$ is in $\#L$ if there exists a logarithmic-space non-deterministic TM M such that:

$f(x) =$ number of accepting computations of M with input x

- ▶ A function $f(\cdot)$ is in $SPANL$ if there exists a logarithmic-space non-deterministic TM M with **output tape** such that:

$f(x) =$ number of **distinct valid outputs** of M with input x

A bit more on complexity classes ...

Definition

- ▶ A function $f(\cdot)$ is in $\#L$ if there exists a logarithmic-space non-deterministic TM M such that:

$$f(x) = \text{number of accepting computations of } M \text{ with input } x$$

- ▶ A function $f(\cdot)$ is in $SPANL$ if there exists a logarithmic-space non-deterministic TM M with **output tape** such that:

$$f(x) = \text{number of distinct valid outputs of } M \text{ with input } x$$

A bit of intuition: $SPANP$ is defined as $SPANL$ but considering a polynomial-time non-deterministic TM with output tape.

- ▶ $\#P$: Given a graph G , return the number of Hamiltonian cycles of G
- ▶ $SPANP$: Given a graph G and an integer k , return the number of Hamiltonian subgraphs of G of size k

Complexity results for the usual paths

Known results:

- ▶ $\#L \subseteq FP$
- ▶ $SPANL \subseteq \#P$, and $SPANL \subseteq FP$ iff $P = NP$

Complexity results for the usual paths

Known results:

- ▶ $\#L \subseteq FP$
- ▶ $SPANL \subseteq \#P$, and $SPANL \subseteq FP$ iff $P = NP$

Theorem (ACP12)

- ▶ $COUNTPATH$ is $SPANL$ -complete
- ▶ $COUNTPATH(\pi)$ is in $\#L$ for every property path π . Moreover, there exists a property path π_0 such that $COUNTPATH(\pi_0)$ is $\#L$ -hard

Outline

- ▶ RDF and SPARQL
- ▶ Navigation in SPARQL 1.1: Property paths
 - ▶ Syntax and semantics
- ▶ Our contributions:
 - ▶ Experimental evaluation
 - ▶ Study of the complexity of evaluating property paths
- ▶ Final remarks

Final remarks

Semantics of SPARQL 1.1 property paths was changed (W3C Working Draft, July 24, 2012) to overcome the issues raised in [LM12,APC12]

- ▶ Existential semantics (no counting) when evaluating $*$
- ▶ $/$ and $|$ are defined as before

Final remarks

Semantics of SPARQL 1.1 property paths was changed (W3C Working Draft, July 24, 2012) to overcome the issues raised in [LM12,APC12]

- ▶ Existential semantics (no counting) when evaluating $*$
- ▶ $/$ and $|$ are defined as before

Are we done?

Final remarks

Semantics of SPARQL 1.1 property paths was changed (W3C Working Draft, July 24, 2012) to overcome the issues raised in [LM12,APC12]

- ▶ Existential semantics (no counting) when evaluating *
- ▶ / and | are defined as before

Are we done?

- ▶ Some questions have to be answered:
 - ▶ Is this a reasonable semantics? (a/b/c) counts, but (a/b/c)* does not
 - ▶ Is the language expressive enough?
- ▶ Some functionalities have to be included:
 - ▶ Queries should be able to return paths

Thank you!

Bibliography

- [ACP12] M. Arenas, S. Conca, J. Pérez: Counting beyond a Yottabyte, or how SPARQL 1.1 property paths will prevent adoption of the standard. WWW 2012: 629–638
- [LM12] K. Losemann, W. Martens: The complexity of evaluating path expressions in SPARQL. PODS 2012: 101–112

Backup slides

An existential semantics to the rescue!

Possible solution

Do not count

Just check whether *there exists* a path satisfying the property path expression

Years of experiences (theory and practice) in:

- ▶ Graph Databases
- ▶ XML
- ▶ SPARQL 1.0 (Psparql, Gleen)

Existential semantics: decision problems

Input: RDF graph G and property path triple (a, \textit{path}, b)

EXISTSPATH

Question: Is there a path from a to b in G satisfying \textit{path} ?

EXISTSW3C

Question: Is the number of solutions of (a, \textit{path}, b) over G greater than 0 (according to the W3C semantics)?

Evaluating existential paths is tractable

Theorem (well-known result)

EXISTSPATH can be solved in $O(|G| \cdot |path|)$

Evaluating existential paths is tractable

Theorem (well-known result)

EXISTSPATH can be solved in $O(|G| \cdot |path|)$

Theorem (ACP12)

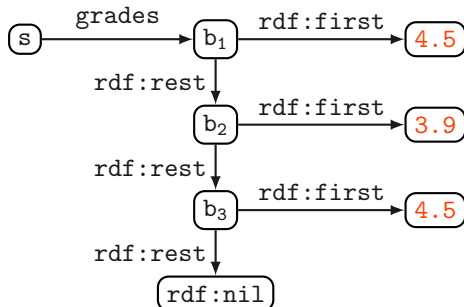
EXISTSPATH and EXISTSW3C are *equivalent*

Corollary

EXISTSW3C can be solved in $O(|G| \cdot |path|)$

A pure existential semantics can handle the use cases

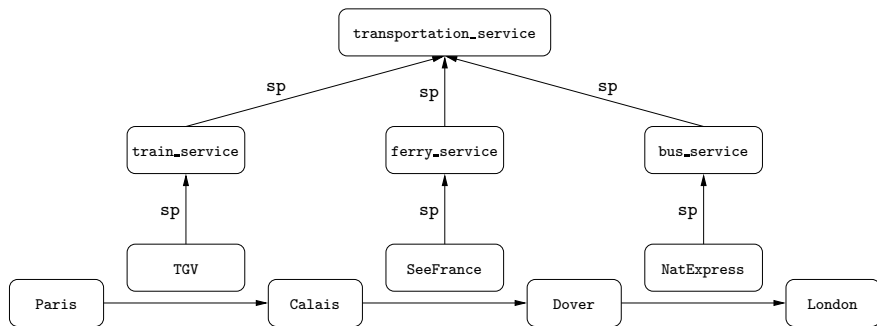
Linked list example:



```
(SELECT ?X ((s, grades, ?Y) AND  
            (?Y, rdf:rest*, ?Z) AND (?Z, rdf:first, ?X)))
```

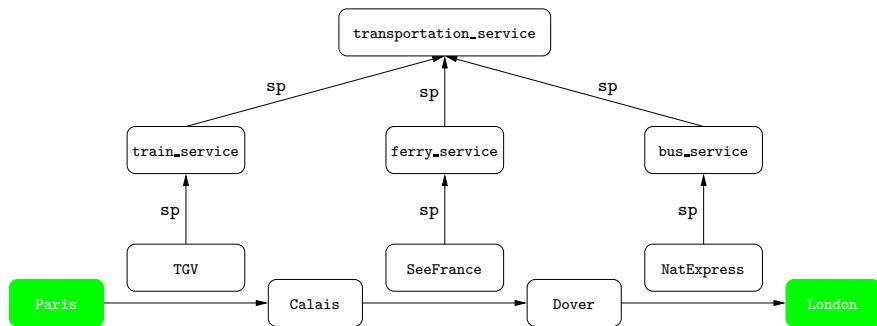
Expressiveness: There is still some work to do

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



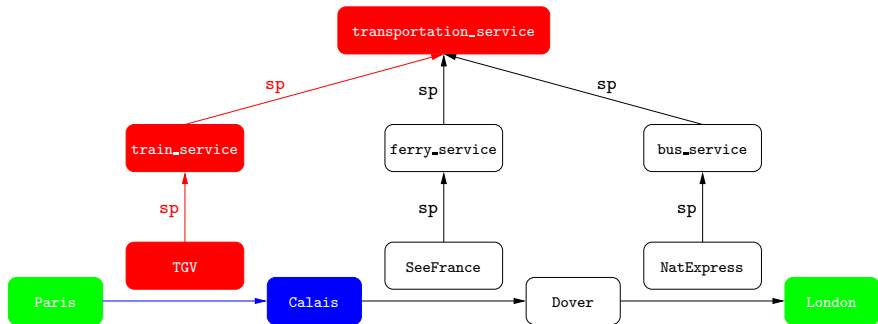
Expressiveness: There is still some work to do

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



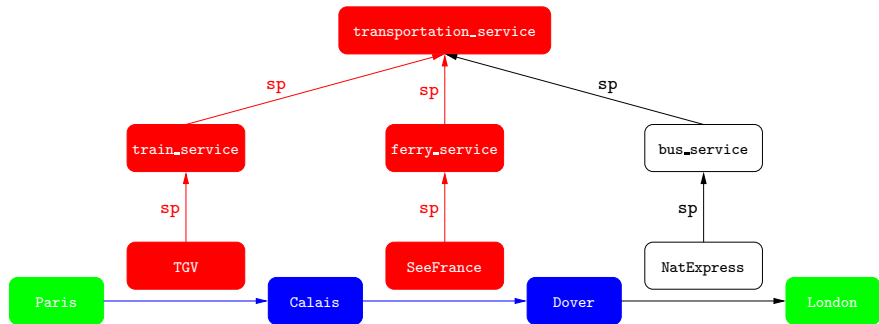
Expressiveness: There is still some work to do

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



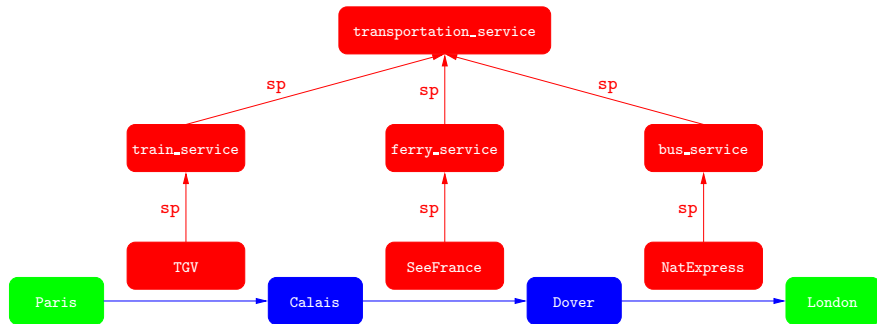
Expressiveness: There is still some work to do

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



Expressiveness: There is still some work to do

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



Expressiveness: There is still some work to do (cont.)

In the previous example, it would be great to be able to list some paths from *a* to *b*.

- ▶ This feature is needed in many use cases

This feature is present in some graph/RDF query languages, but it has not been standardized.

- ▶ Paths can be returned as strings in Cypher (Neo4j)
- ▶ Virtuoso provides some options in the transitivity extension that allow to store paths in the output table