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

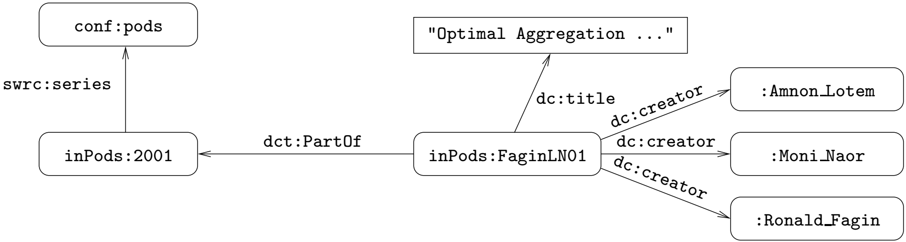
RDF in a nutshell

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.l3s.de/d2r/resource/authors/>
   conf: <http://dblp.l3s.de/d2r/resource/conferences/>
inPods: <http://dblp.l3s.de/d2r/resource/publications/conf/pods/>
   swrc: <http://swrc.ontoware.org/ontology#>
     dc: <http://purl.org/dc/elements/1.1/>
    dct: <http://purl.org/dc/terms/>
```



An example of a URI

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

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

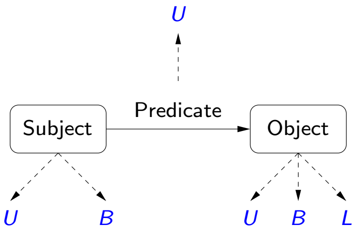
Outline

- 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

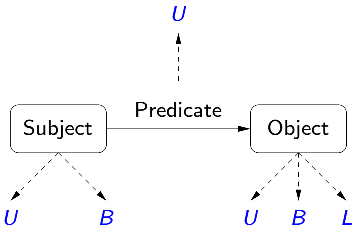
- RDF and SPARQL
- Navigation in SPARQL 1.1: Property paths
  - Syntax and semantics
- Our contributions:
  - Experimental evaluation
  - Study of the complexity of evaluating property paths
- Final remarks

RDF formal model



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

RDF formal model



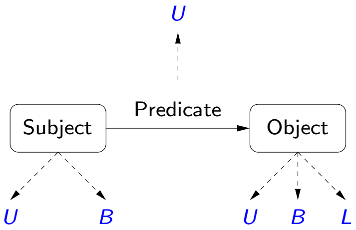
U : set of URIs

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



$\mathbf{U}$: set of URIs

$\mathbf{B}$: set of blank nodes

$\mathbf{L}$: 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`	$(?X, \texttt{name}, ?Y)$
`{  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 }`	$(\text{SELECT } \{?X, ?Y, \ldots\}\ P)$

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 elements from $\mathbf{U}$ and $\mathbf{L}$
- unary predicate bound$(\cdot)$
- boolean combinations $(\wedge, \vee, \neg)$

Mappings: building block for the semantics

Definition

A mapping is a partial function:

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

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:

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

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	

- 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

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

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

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

Sets of mappings and operations

Definition

Union: includes mappings in M_1 and in M_2

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

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

- $M_1 ⟕ M_2 = (M_1 \bowtie M_2) \cup (M_1 \smallsetminus M_2)$

Semantics of SPARQL

Given an RDF graph G.

Definition

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

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{aligned}
[\![t]\!]_G &= \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\} \\
[\![(P_1 \text{ AND } P_2)]\!]_G &= [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \\
[\![(P_1 \text{ UNION } P_2)]\!]_G &= [\![P_1]\!]_G \cup [\![P_2]\!]_G \\
[\![(P_1 \text{ OPT } P_2)]\!]_G &= [\![P_1]\!]_G \,⟕\, [\![P_2]\!]_G \\
[\![(\text{SELECT } W \; P)]\!]_G &= \{\mu_{|W} \mid \mu \in [\![P]\!]_G\} \\
& \quad \mathrm{dom}(\mu_{|W}) = \mathrm{dom}(\mu) \cap W \text{ and} \\
& \quad \mu_{|W}(?X) = \mu(?X) \text{ for every } ?X \in \mathrm{dom}(\mu_{|W})
\end{aligned}
$$

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

Satisfaction of value constraints

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

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

Definition

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

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

$(R_1,$ 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

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

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

$?X$	$?Y$
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	

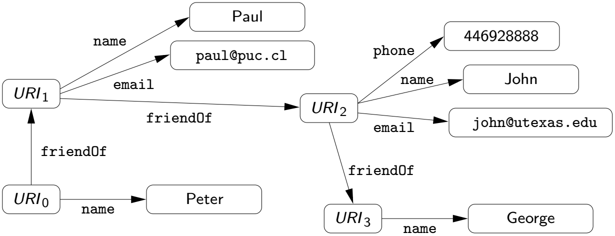
$?X$	$?E$
R_1	J@ed.ex

- from the union

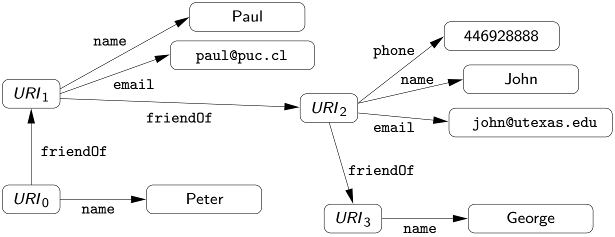
Outline

- 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

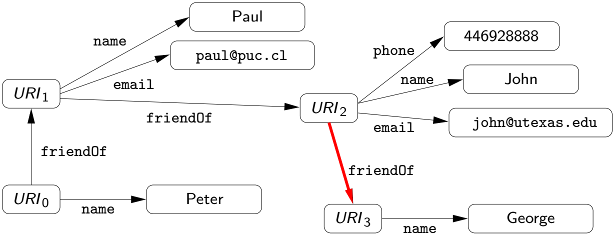


SPARQL 1.0 provides limited navigational capabilities



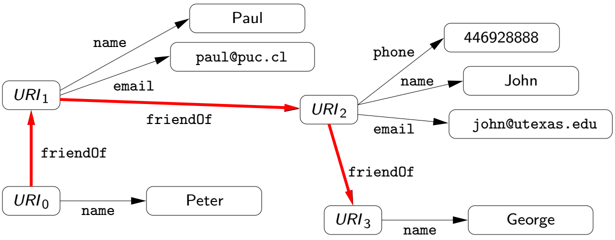
(SELECT $?X$ (($?X$, `friendOf`, $?Y$) AND ($?Y$, `name`, `George`)))

SPARQL 1.0 provides limited navigational capabilities



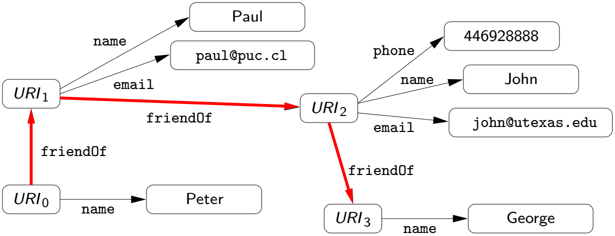
(SELECT $?X$ (($?X$, `friendOf`, $?Y$) AND ($?Y$, `name`, `George`)))

SPARQL 1.0 provides limited navigational capabilities

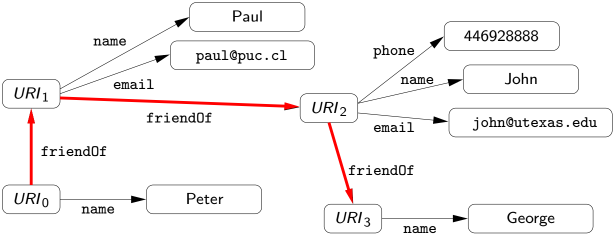


(SELECT $?X$ (($?X$, `friendOf`, $?Y$) AND ($?Y$, `name`, `George`)))

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

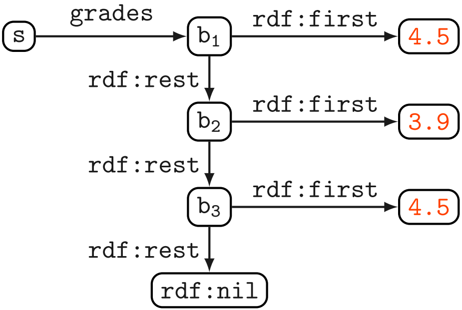
- SPARQL uses a bag semantics: Duplicates are not eliminated

Syntax and semantics of property paths

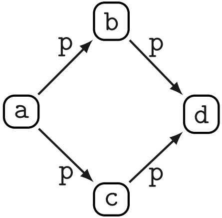
Syntax: Property paths are regular expressions (/, |, *)

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

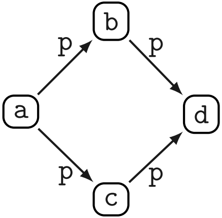


Property paths are designed to count



$(\mathtt{a},\ \mathtt{p}^*,\ ?X)$

Property paths are designed to count



$(\texttt{a}, \texttt{p}^*, ?X)$

$?X$
a
b
c
d
d

Definition of the semantics of property paths

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

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

Definition of the semantics of property paths

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

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

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

$$((?X, path_1, ?Y) \text{ UNION } (?X, path_2, ?Y))$$

Definition of the semantics of property paths

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

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

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

$$((?X, path_1, ?Y) \text{ UNION } (?X, path_2, ?Y))$$

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**

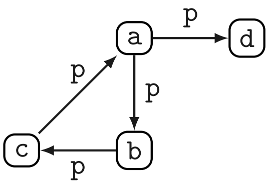
"the algorithm extends the multiset of results by one application of path. If a node has been visited for path, it is not a candidate for another step. A node can be visited multiple times if different paths visit it."

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

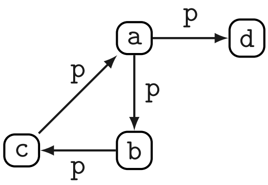
RDF Graph G:



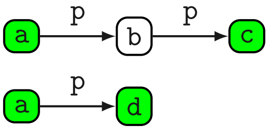
Evaluation of $(?X, \mathtt{p}^*, ?Y)$:

The procedure in a nutshell

RDF Graph G:

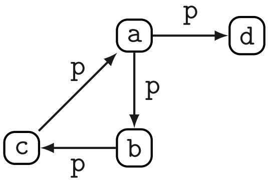


Evaluation of $(?X, \mathtt{p}^*, ?Y)$:

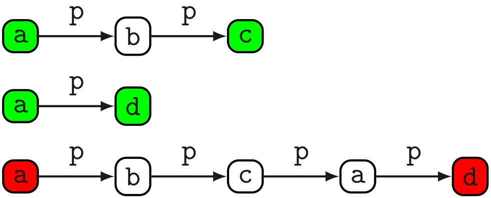


The procedure in a nutshell

RDF Graph G:

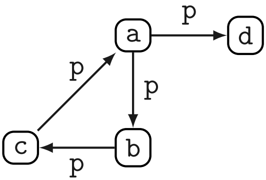


Evaluation of $(?X, \texttt{p}^*, ?Y)$:



The procedure in a nutshell

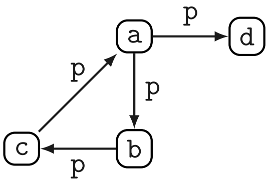
RDF Graph G:



Evaluation of $(?X, (\mathtt{p}^*)^*, ?Y)$:

The procedure in a nutshell

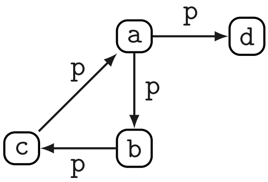
RDF Graph G:



Evaluation of $(?X, (\texttt{p}^*)^*, ?Y)$:

The procedure in a nutshell

RDF Graph G:

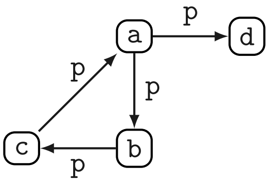


Evaluation of $(?X, (\texttt{p}^*)^*, ?Y)$:

Evaluation of $\texttt{p}^*$:

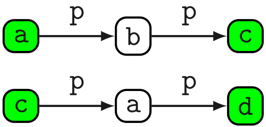
The procedure in a nutshell

RDF Graph G:



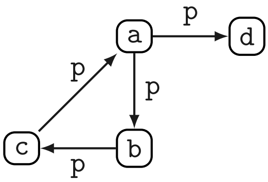
Evaluation of $(?X, (\texttt{p}^*)^*, ?Y)$:

Evaluation of $\texttt{p}^*$:

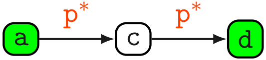


The procedure in a nutshell

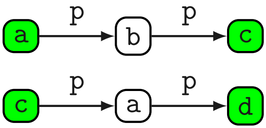
RDF Graph G:



Evaluation of $(?X, (\mathtt{p}^*)^*, ?Y)$:

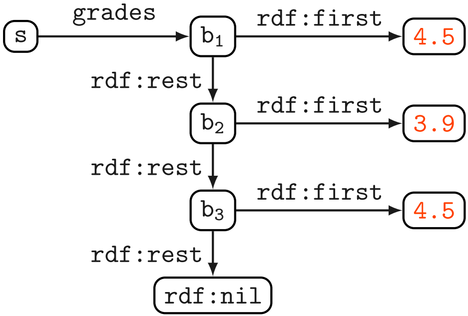


Evaluation of $\mathtt{p}^*$:



Is this a good semantics?

Linked list example:



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

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

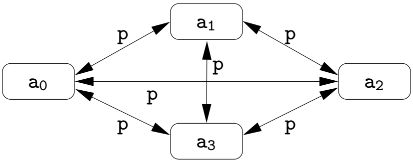
Impact on W3C standard:

- 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

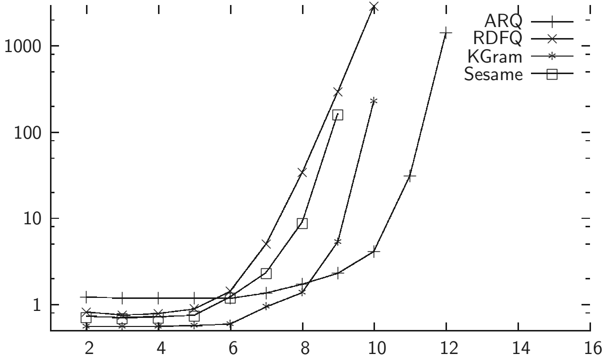
Data:

- 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

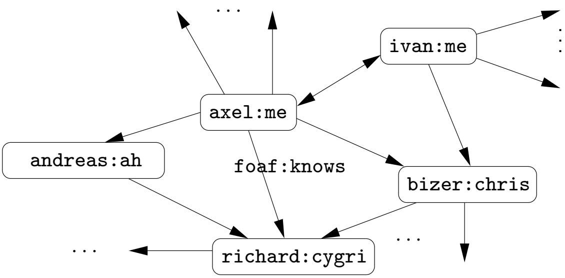


$(\mathtt{a}_0, \mathtt{p}^*, \mathtt{a}_1)$

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

$$(\texttt{axel:me}, \texttt{foaf:knows}^*, ?X)$$

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

$(\mathtt{a}_0, \mathtt{p}^*, \mathtt{a}_1)$

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

Counting the number of solutions

Data: Clique of size n

$(\mathtt{a}_0, \mathtt{p}^*, \mathtt{a}_1)$

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

$(\mathtt{a}_0, \mathtt{p}^*, \mathtt{a}_1)$ $\quad$ $(\mathtt{a}_0, (\mathtt{p}^*)^*, \mathtt{a}_1)$

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

$(\mathtt{a}_0, \mathtt{p}^*, \mathtt{a}_1)$

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

$(\mathtt{a}_0, (\mathtt{p}^*)^*, \mathtt{a}_1)$

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

$(\mathtt{a}_0, \mathtt{p}^*, \mathtt{a}_1)$

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

$(\mathtt{a}_0, (\mathtt{p}^*)^*, \mathtt{a}_1)$

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

$(\mathtt{a}_0, ((\mathtt{p}^*)^*)^*, \mathtt{a}_1)$

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

Counting the number of solutions

Data: Clique of size n

$(\texttt{a}_0, \texttt{p}^*, \texttt{a}_1)$

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

$(\texttt{a}_0, (\texttt{p}^*)^*, \texttt{a}_1)$

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

$(\texttt{a}_0, ((\texttt{p}^*)^*)^*, \texttt{a}_1)$

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

$$(\texttt{axel:me}, \texttt{foaf:knows}^*, ?X)$$

Counting the number of solutions (cont.)

Data: foaf links crawled from the Web

$(\texttt{axel:me}, \texttt{foaf:knows}^*, ?X)$

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

$(\texttt{axel:me}, \texttt{foaf:knows}^{*}, ?X)$

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 ($\texttt{a}$, *path*, $\texttt{b}$)

Output: Count the number of solutions of ($\texttt{a}$, *path*, $\texttt{b}$) 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

$$(\cdots((\mathrm{p}^*)^*)^* \cdots)^*$$

  with s nested stars

- Let K_n be a clique with n nodes
- Let *CountClique*(s, n) be the number of solutions of $(\mathrm{a}_0, \pi_s, \mathrm{a}_1)$ over K_n

A double-exponential lower bound for counting

- Let π_s be a property path of the form

$$(\cdots((\mathrm{p}^*)^*)^* \cdots)^*$$

  with s nested stars

- Let K_n be a clique with n nodes
- Let *CountClique*(s, n) be the number of solutions of $(\mathrm{a}_0, \pi_s, \mathrm{a}_1)$ over K_n

Lemma (ACP12)

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

A double-exponential lower bound for counting

- Let π_s be a property path of the form

$$(\cdots((\mathrm{p}^*)^*)^* \cdots)^*$$

  with s nested stars

- Let K_n be a clique with n nodes
- Let $CountClique(s, n)$ be the number of solutions of $(\mathrm{a}_0, \pi_s, \mathrm{a}_1)$ over K_n

Lemma (ACP12)

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

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

We can now explain our experimental results

CountClique(s, n) allows us to fill in the blanks

$(\mathtt{a}_0,\ (\mathtt{p}^*)^*,\ \mathtt{a}_1)$

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

We can now explain our experimental results

CountClique(s, n) allows us to fill in the blanks

$(\mathtt{a}_0,\ (\mathtt{p}^*)^*,\ \mathtt{a}_1)$

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

We can now explain our experimental results

CountClique(s, n) allows us to fill in the blanks

$(\mathtt{a}_0,\ (\mathtt{p}^*)^*,\ \mathtt{a}_1)$

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

We can now explain our experimental results

CountClique(s, n) allows us to fill in the blanks

$(\mathtt{a}_0,\ (\mathtt{p}^*)^*,\ \mathtt{a}_1)$

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:

$\textsc{CountW3C}(\textit{path})$

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

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

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?

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 $\#\mathrm{P}$ if there exists a polynomial-time non-deterministic TM M such that:

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

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 $\#\mathrm{P}$ if there exists a polynomial-time non-deterministic TM M such that:

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

- COUNTSAT is $\#\mathrm{P}$-complete

Data complexity of property paths is intractable

Theorem (ACP12)

- For every property path π: $\textsc{CountW3C}(\pi)$ is in $\#\mathrm{P}$
- $\textsc{CountW3C}(a^*)$ is $\#\mathrm{P}$-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)

- $\textsc{CountSimplePath}$ is $\#\mathrm{P}$-complete
- In fact, $\textsc{CountSimplePath}(a^*)$ is $\#\mathrm{P}$-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 $(\mathtt{a}, \mathit{path}, \mathtt{b})$

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

A bit more on complexity classes ...

Definition

- A function $f(\cdot)$ is in $\#\mathrm{L}$ 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 bit more on complexity classes ...

Definition

- A function $f(\cdot)$ is in $\#\mathrm{L}$ 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 $\textsc{span}\mathrm{L}$ 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 more on complexity classes ...

Definition

- A function $f(\cdot)$ is in $\#\mathrm{L}$ 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 $\textsc{span}\mathrm{L}$ 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: $\textsc{span}\mathrm{P}$ is defined as $\textsc{span}\mathrm{L}$ but considering a polynomial-time non-deterministic TM with output tape.

- $\#\mathrm{P}$: Given a graph G, return the number of Hamiltonian cycles of G
- $\textsc{span}\mathrm{P}$: 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:

- $\#\mathrm{L} \subseteq \mathrm{FP}$
- $\textsc{span}\mathrm{L} \subseteq \#\mathrm{P}$, and $\textsc{span}\mathrm{L} \subseteq \mathrm{FP}$ iff $\mathrm{P} = \mathrm{NP}$

Complexity results for the usual paths

Known results:

- $\#\mathrm{L} \subseteq \mathrm{FP}$
- $\textsc{spanL} \subseteq \#\mathrm{P}$, and $\textsc{spanL} \subseteq \mathrm{FP}$ iff $\mathrm{P} = \mathrm{NP}$

Theorem (ACP12)

- $\textsc{CountPath}$ is $\textsc{spanL}$-complete
- $\textsc{CountPath}(\pi)$ is in $\#\mathrm{L}$ for every property path π. Moreover, there exists a property path π_0 such that $\textsc{CountPath}(\pi_0)$ is $\#\mathrm{L}$-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? $(\mathtt{a}/\mathtt{b}/\mathtt{c})$ counts, but $(\mathtt{a}/\mathtt{b}/\mathtt{c})^*$ 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 $(\mathtt{a}, \mathit{path}, \mathtt{b})$

ExistsPath

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

ExistsW3C

Question: Is the number of solutions of $(\mathtt{a}, \mathit{path}, \mathtt{b})$ 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 $O(|G| \cdot |\mathit{path}|)$

Theorem (ACP12)

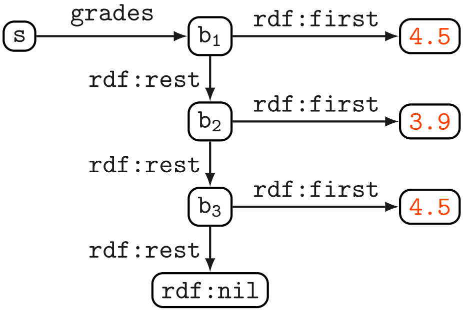
ExistsPath and ExistsW3C are *equivalent*

Corollary

ExistsW3C can be solved in $O(|G| \cdot |\mathit{path}|)$

A pure existential semantics can handle the use cases

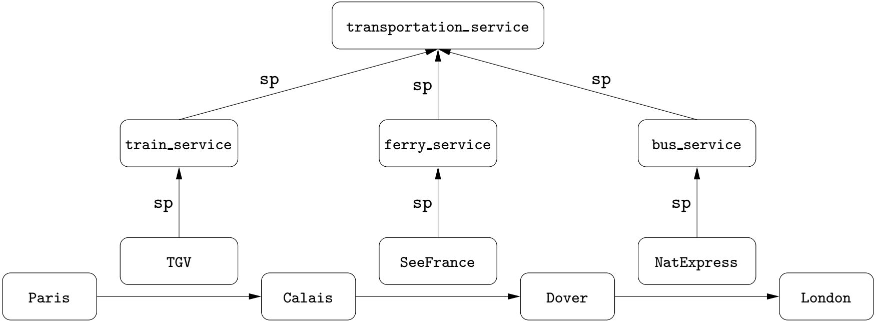
Linked list example:



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

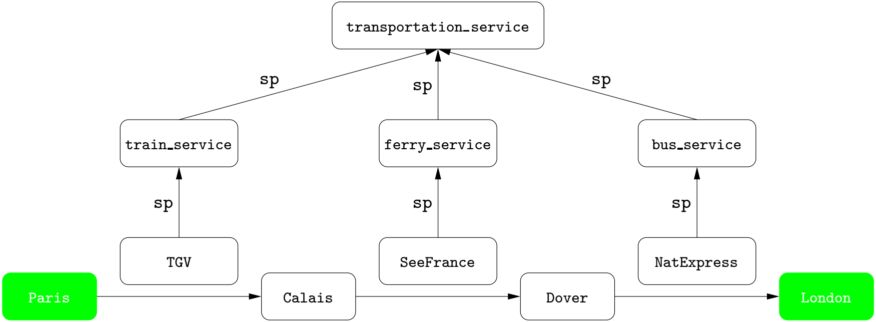
Expressiveness: There is still some work to do

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



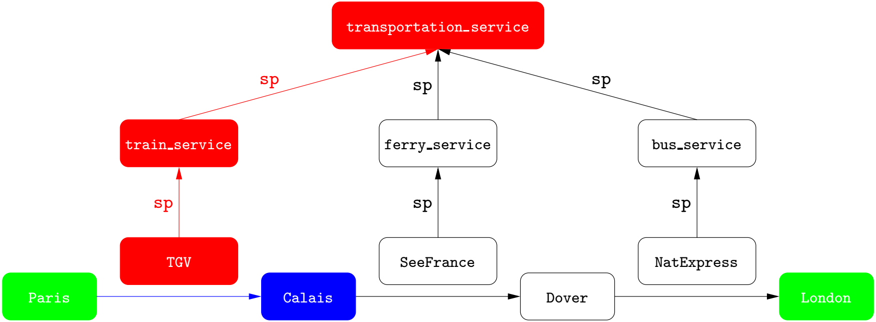
Expressiveness: There is still some work to do

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



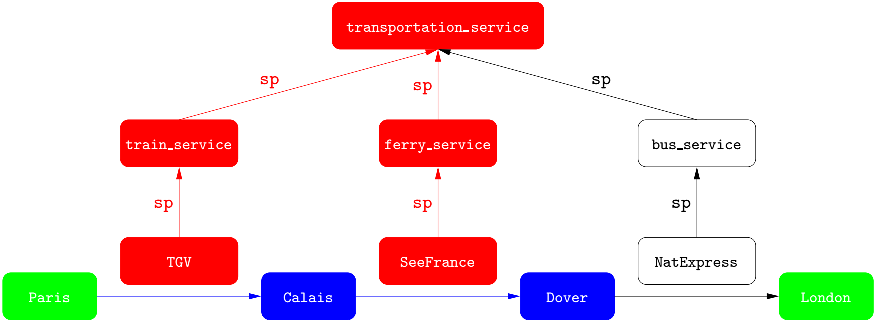
Expressiveness: There is still some work to do

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



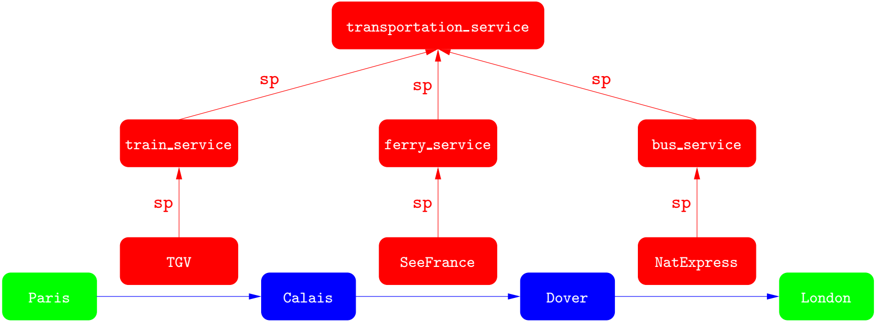
Expressiveness: There is still some work to do

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