

Paths in semantic search: A back and forth story

Marcelo Arenas

PUC Chile

The story

Navigational capabilities are important for graph data models.

RDF is a new data model.

- ▶ It can be considered as a graph data model, but it has some non-trivial new features.

Interaction between databases and semantic web.

The story

Navigational capabilities are important for graph data models.

RDF is a new data model.

- ▶ It can be considered as a graph data model, but it has some non-trivial new features.

Interaction between databases and semantic web.

- ◀ Need for navigational capabilities in SPARQL

The story

Navigational capabilities are important for graph data models.

RDF is a new data model.

- ▶ It can be considered as a graph data model, but it has some non-trivial new features.

Interaction between databases and semantic web.

- ← Need for navigational capabilities in SPARQL
- Extensive use of regular expressions to specify paths in graph databases and XML

The story

Navigational capabilities are important for graph data models.

RDF is a new data model.

- ▶ It can be considered as a graph data model, but it has some non-trivial new features.

Interaction between databases and semantic web.

- ← Need for navigational capabilities in SPARQL
- Extensive use of regular expressions to specify paths in graph databases and XML
- ← Regular expressions are included in SPARQL 1.1, but with a multiset (bag) semantics

The story (cont.)

- Techniques from graph databases, automata theory and computational complexity

The story (cont.)

- Techniques from graph databases, automata theory and computational complexity
- ← New proposal of an *intermediate* semantics for regular expressions in SPARQL 1.1

The story (cont.)

- Techniques from graph databases, automata theory and computational complexity
- ← New proposal of an *intermediate* semantics for regular expressions in SPARQL 1.1
- Pure existential (set) semantics could be a better alternative

The story (cont.)

- Techniques from graph databases, automata theory and computational complexity
- ← New proposal of an *intermediate* semantics for regular expressions in SPARQL 1.1
- Pure existential (set) semantics could be a better alternative
- ← RDFS and OWL vocabularies have to be considered when discovering paths

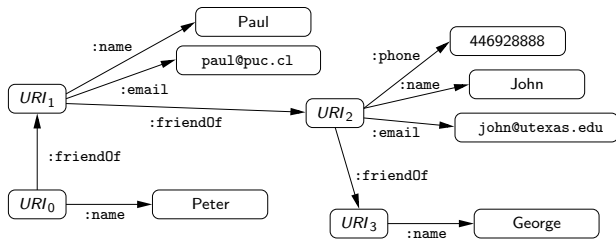
The story (cont.)

- Techniques from graph databases, automata theory and computational complexity
- ← New proposal of an *intermediate* semantics for regular expressions in SPARQL 1.1
- Pure existential (set) semantics could be a better alternative
- ← RDFS and OWL vocabularies have to be considered when discovering paths
- ← Paths can be part of the output

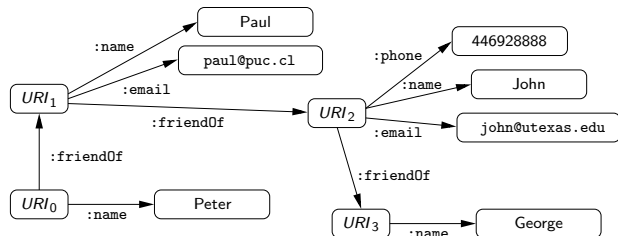
The story (cont.)

- Techniques from graph databases, automata theory and computational complexity
- ← New proposal of an *intermediate* semantics for regular expressions in SPARQL 1.1
- Pure existential (set) semantics could be a better alternative
- ← RDFS and OWL vocabularies have to be considered when discovering paths
- ← Paths can be part of the output
- ...
- ← ...

SPARQL 1.0 provides limited navigational capabilities

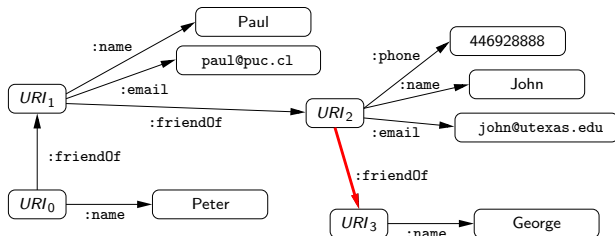


SPARQL 1.0 provides limited navigational capabilities



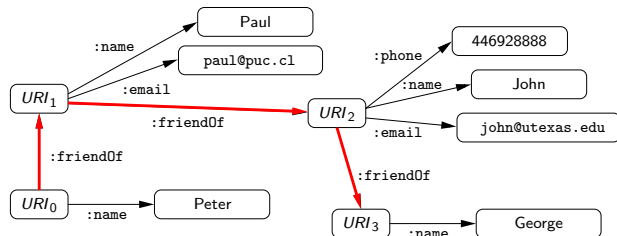
```
SELECT ?x
WHERE
{
    ?x :friendOf ?y .
    ?y :name "George" .
}
```

SPARQL 1.0 provides limited navigational capabilities



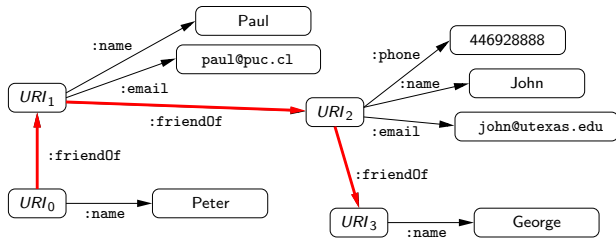
```
SELECT ?x
WHERE
{
    ?x :friendOf ?y .
    ?y :name "George" .
}
```

SPARQL 1.0 provides limited navigational capabilities

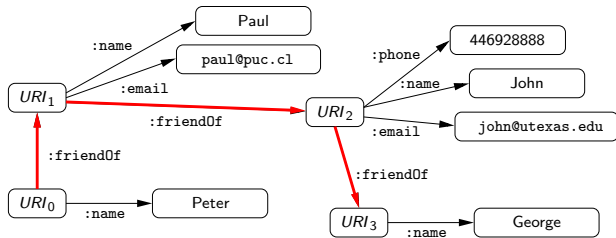


```
SELECT ?x
WHERE
{
    ?x :friendOf ?y .
    ?y :name      "George" .
}
```

A possible solution: Regular expressions in graph databases

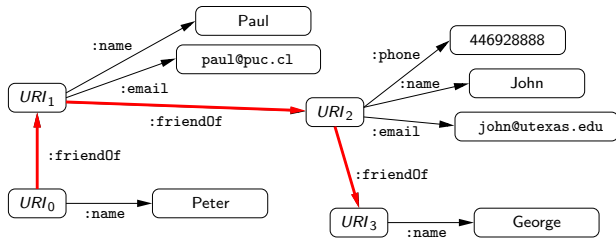


A possible solution: Regular expressions in graph databases



```
SELECT ?x
WHERE
{
  ?x (:friendOf)* ?y .
  ?y :name      "George" .
}
```

A possible solution: Regular expressions in graph databases



```
SELECT ?x
WHERE
{
  ?x (:friendOf)* ?y .      ← SPARQL 1.1 property path
  ?y :name "George" .
}
```

Problems to study

- ▶ Syntax
- ▶ Semantics
- ▶ Efficient algorithms for evaluating property paths
 - ▶ Complexity of the evaluation problem

Problems to study

- ▶ Syntax
- ▶ Semantics
- ▶ Efficient algorithms for evaluating property paths
 - ▶ Complexity of the evaluation problem

All this has to be done considering the use cases.

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.

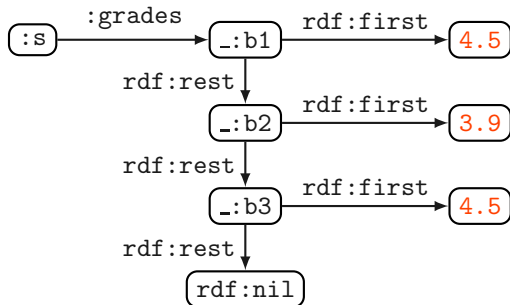
- ▶ Retrieving all the elements of a linked list

Syntax and semantics of property paths

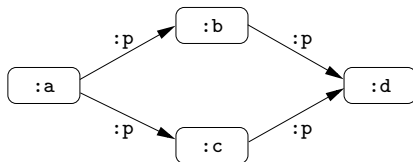
Syntax: Property paths are regular expressions (`/`, `|`, `*`)

Semantics: Repeated values are needed in some use cases.

- ▶ Retrieving all the elements of a linked list

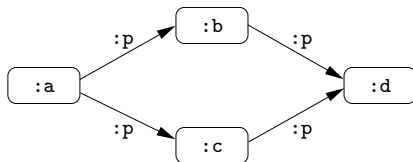


Property paths are designed to count



```
SELECT ?x  
WHERE { :a (:p)* ?x }
```

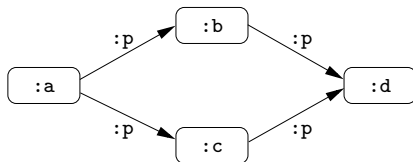

Property paths are designed to count



```
SELECT ?x  
WHERE { :a (:p)* ?x }
```

<u>?x</u>
:a
:b
:c
:d
:d

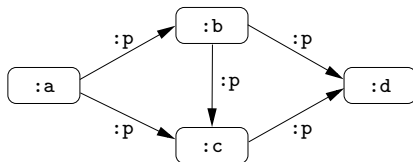
Property paths are designed to count



```
SELECT ?x  
WHERE { :a (:p)* ?x }
```

```
?x  
-----  
:a  
:b  
:c  
:d  
:d
```

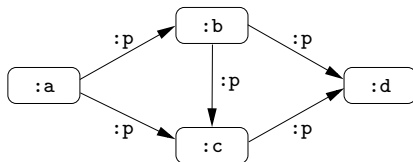
Property paths are designed to count



```
SELECT ?x  
WHERE { :a (:p)* ?x }
```

<u>?x</u>
:a
:b
:c
:d
:d

Property paths are designed to count



```
SELECT ?x  
WHERE { :a (:p)* ?x }
```

?x
:a
:b
:c
:d
:d
:c
:d

Definition of the semantics of property paths

```
SELECT ?x  
WHERE { :a (:b/:c) ?x }
```

Definition of the semantics of property paths

```
SELECT ?x  
WHERE { :a (:b/:c) ?x }
```

is replaced by:

```
SELECT ?x  
WHERE { :a :b ?y .  
        ?y :c ?x . }
```

Definition of the semantics of property paths

```
SELECT ?x  
WHERE { :a (:b/:c) ?x }
```

is replaced by:

```
SELECT ?x  
WHERE { :a :b ?y .  
        ?y :c ?x . }
```

Same idea is applied to define |

Definition of the semantics of property paths

```
SELECT ?x  
WHERE { :a (:b/:c) ?x }
```

is replaced by:

```
SELECT ?x  
WHERE { :a :b ?y .  
        ?y :c ?x . }
```

Same idea is applied to define |

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.”*

SPARQL 1.1 Last Call (Jan 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.”*

SPARQL 1.1 Last Call (Jan 2012)

- ▶ SPARQL 1.1 document provides a special procedure to handle cycles and make the count

Is this a good semantics?

Linked list example:

```
SELECT ?x  
WHERE { :s :grades/(rdf:rest)*/rdf:first ?x }
```

Is this a good semantics?

Linked list example:

```
SELECT ?x  
WHERE { :s :grades/(rdf:rest)*/rdf:first ?x }
```

Couldn't these use cases be handled with a simpler semantics?

- ▶ Isn't a problem to use an arbitrary procedure to count paths?
What are we counting?

Is this a good semantics? (cont.)

Regular expressions with an *existential* semantics have been widely studied and used in databases.

- ▶ Why don't we take advantage of this experience?

Is this a good semantics? (cont.)

Regular expressions with an *existential* semantics have been widely studied and used in databases.

- ▶ Why don't we take advantage of this experience?

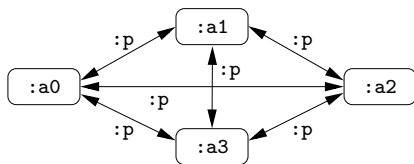
A new problem need to be solved: **Counting the number of paths in a graph that conform to a regular expression**

- ▶ How difficult is this problem?

Some experimental results with synthetic data

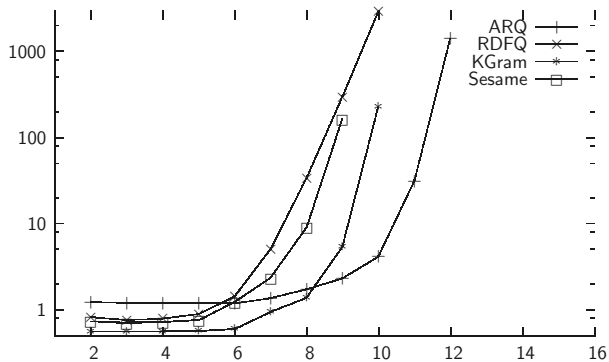
Data:

- ▶ 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

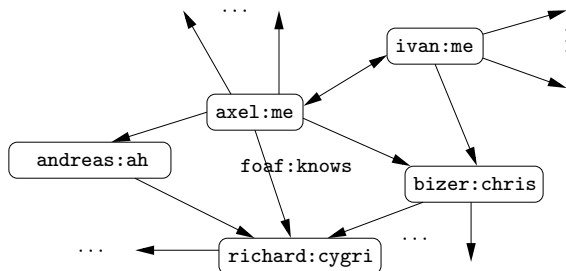


```
SELECT * WHERE { :a0 (:p)* :a1 }
```


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

```
SELECT * WHERE { axel:me (foaf:knows)* ?x }
```

Some experimental results with real data

```
SELECT * WHERE { 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

$\{ :a0 (:p)* :a1 \}$

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

Counting the number of solutions

Data: Clique of size n

```
{ :a0 (:p)* :a1 }
```

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

$\{ :a0 (:p)* :a1 \}$ $\{ :a0 ((:p)*)* :a1 \}$

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

$\{ :a0 (:p)* :a1 \}$ $\{ :a0 ((:p)*)* :a1 \}$

n	# Sol.	n	# Sol.
9	13,700	2	1
10	109,601	3	6
11	986,410	4	305
12	9,864,101	5	418,576
13	–	6	–

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

Counting the number of solutions

Data: Clique of size n

$\{ :a0 (:p)* :a1 \}$ $\{ :a0 ((:p)*)* :a1 \}$ $\{ :a0 (((:p)*)*)* :a1 \}$

n	# Sol.	n	# Sol.
9	13,700	2	1
10	109,601	3	6
11	986,410	4	305
12	9,864,101	5	418,576
13	–	6	–

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

Counting the number of solutions

Data: Clique of size n

$\{ :a0 (:p)* :a1 \}$

$\{ :a0 ((:p)*)* :a1 \}$

$\{ :a0 (((:p)*)*)* :a1 \}$

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	–

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?

Theory can help

It is possible to construct a formula for calculating the number of solutions in the clique experiment.

- ▶ A double exponential lower bound is obtained

Theory can help

It is possible to construct a formula for calculating the number of solutions in the clique experiment.

- ▶ A double exponential lower bound is obtained

$\{ :a0 ((:p)*)* :a1 \}$

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

Theory can help

It is possible to construct a formula for calculating the number of solutions in the clique experiment.

- ▶ A double exponential lower bound is obtained

$\{ :a0 ((:p)*)* :a1 \}$

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

Theory can help

It is possible to construct a formula for calculating the number of solutions in the clique experiment.

- ▶ A double exponential lower bound is obtained

`{ :a0 ((:p)*)* :a1 }`

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

Current situation

Normative semantics of SPARQL 1.1 property paths will be changed to overcome these issues.

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

Current situation

Normative semantics of SPARQL 1.1 property paths will be changed to overcome these issues.

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

Are we done?

Current situation

Normative semantics of SPARQL 1.1 property paths will be changed to overcome these issues.

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

Are we done? **Some questions have to be answered.**

Current situation

Normative semantics of SPARQL 1.1 property paths will be changed to overcome these issues.

- ▶ 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**

Current situation

Normative semantics of SPARQL 1.1 property paths will be changed to overcome these issues.

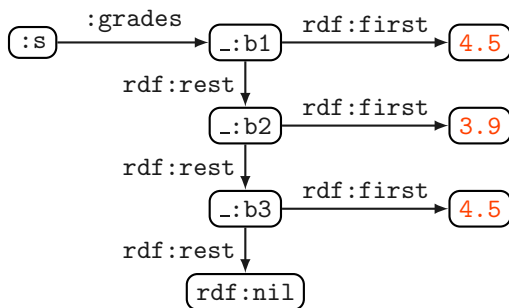
- ▶ Existential semantics (no counting) when evaluating *
- ▶ / and | are defined as before

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?**

A pure existential semantics can handle the use cases

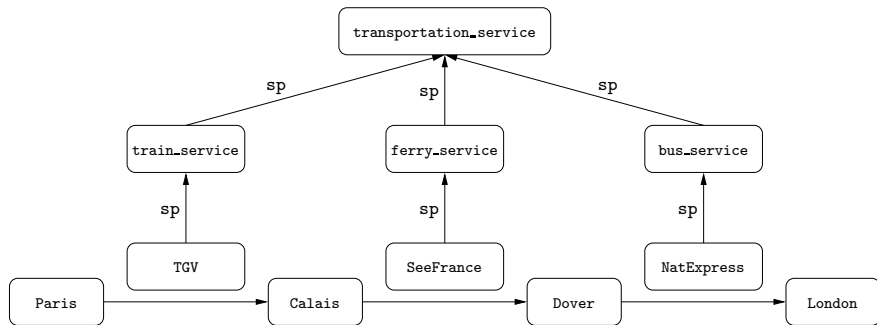
Linked list example:



```
SELECT ?x
WHERE { :s :grades ?y .
        ?y (rdf:rest)* ?z .
        ?z rdf:first ?x . }
```

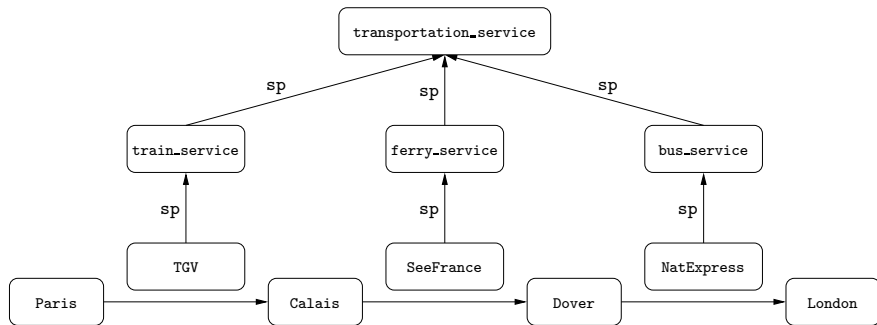
Expressiveness: There is still some work to do

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

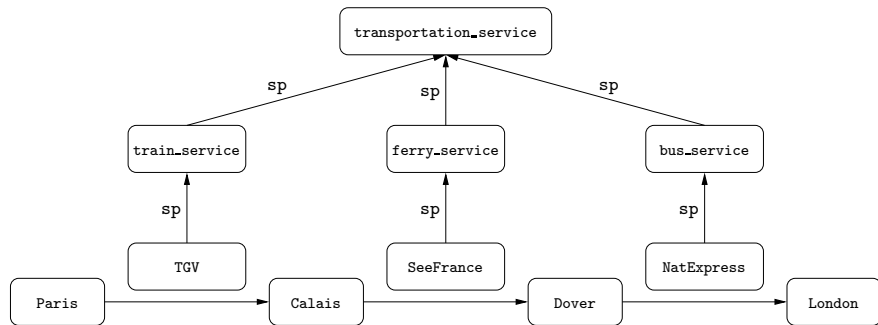
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL:

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 .

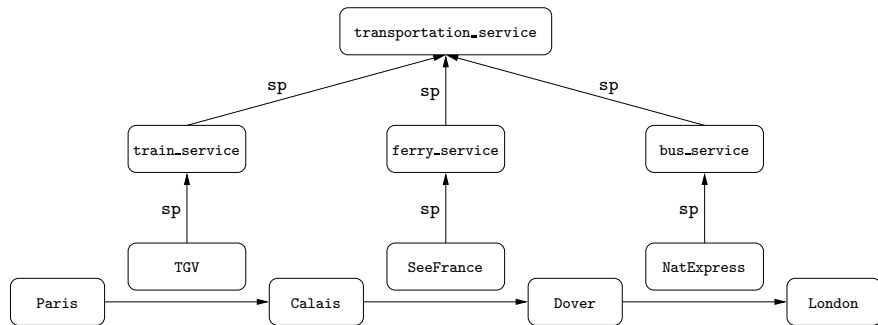


nSPARQL: $?x$

$?y$

Expressiveness: There is still some work to do

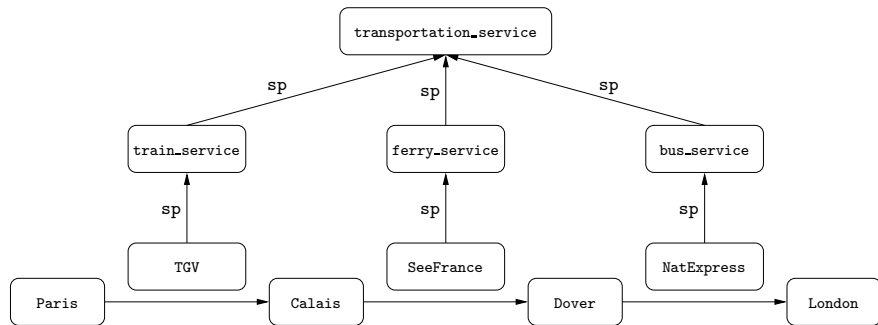
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL: $?x$ (next: $?y$)⁺

Expressiveness: There is still some work to do

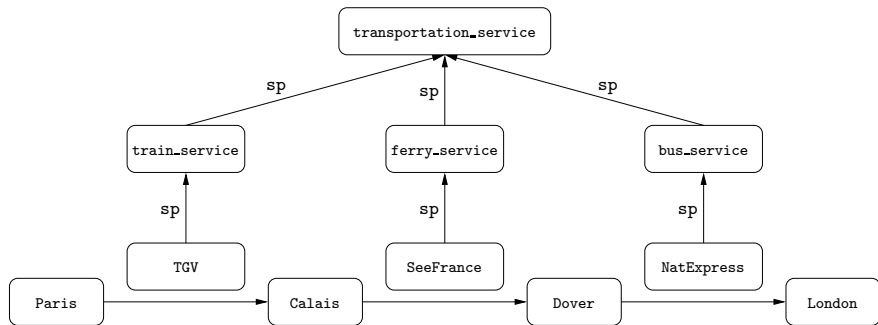
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL: `?x (next:[` `])+ ?y`

Expressiveness: There is still some work to do

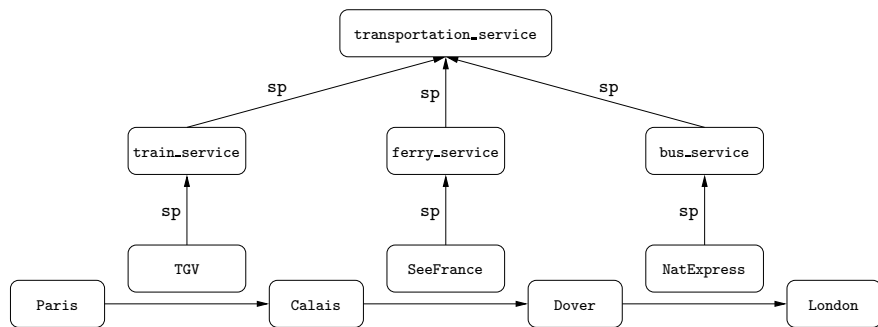
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL: `?x (next:[(next:sp)* /])+ ?y`

Expressiveness: There is still some work to do

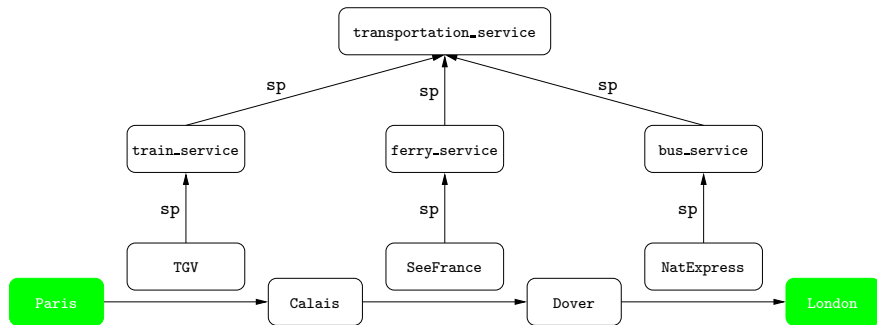
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL: `?x (next:[(next:sp)*]/transportation_service)]+ ?y`

Expressiveness: There is still some work to do

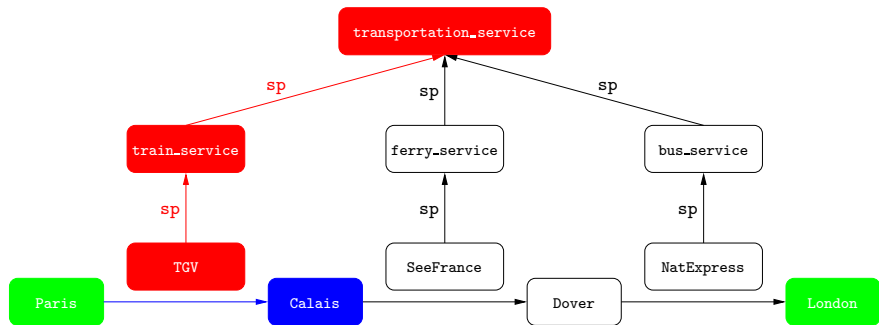
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL: `?x (next:[(next:sp)*]/transportation_service)+ ?y`

Expressiveness: There is still some work to do

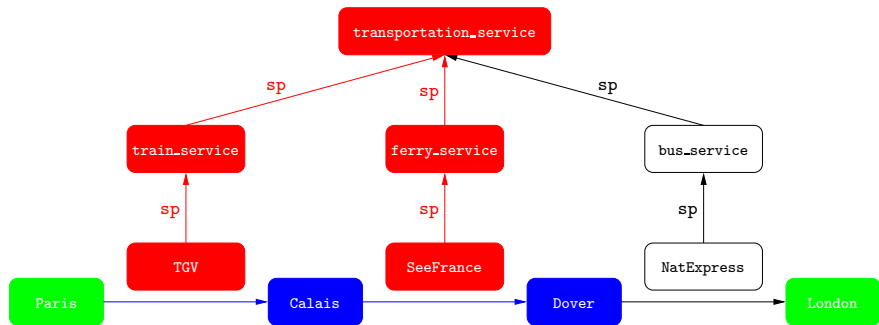
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL: `?x (next:[(next:sp)*]/transportation_service)]+ ?y`

Expressiveness: There is still some work to do

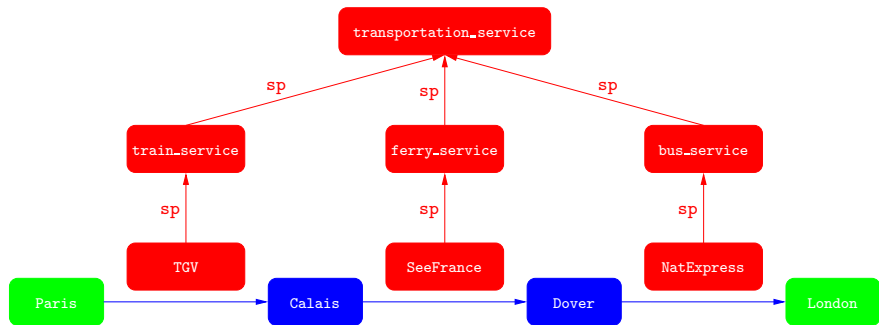
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL: `?x (next:[(next:sp)*]/transportation_service)+ ?y`

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 .



nSPARQL: `?x (next:[(next:sp)*/transportation_service])+ ?y`

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