

Querying Semantic Web Data with SPARQL: State of the Art and Research Perspectives

Marcelo Arenas¹ and Jorge Pérez²

¹ Pontificia Universidad Católica de Chile
² Universidad de Chile

“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:

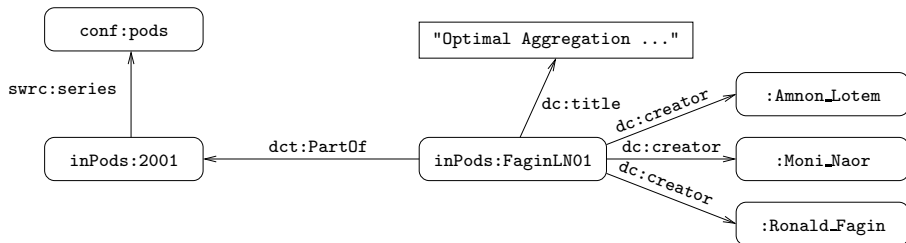
- ▶ URI vocabulary
 - ▶ A URI is an atomic piece of data, and it identifies an abstract resource
- ▶ Syntax based on directed labeled graphs
 - ▶ URIs are used as node labels and edge labels
- ▶ Schema definition language (**RDFS**): Define new vocabulary
 - ▶ Typing, inheritance of classes and properties,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

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`



PODS | D2R Server publishing the

http://dblp.l3s.de/d2r/page/conferences/pods

Apple (136) Amazon Yahoo! News (919)

Resource URI: http://

[Home](#) | [Example Conferences](#)

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>

URI can be used for any abstract resource

http://dblp.l3s.de/d2r/page/authors/Ronald_Fagin

Ronald Fagin | D2R Server publishing the

Resource URI: http://dblp.l3s.de/d2r/page/authors/Ronald_Fagin

[Home](#) | [Example Authors](#)

Property	Value
is dc:creator of	http://dblp.l3s.de/d2r/resource/publications/conf/aaai/FagiHV86
is dc:creator of	http://dblp.l3s.de/d2r/resource/publications/conf/aaai/FaginHVM94
is dc:creator of	http://dblp.l3s.de/d2r/resource/publications/conf/aaai/HalpernF90
is dc:creator of	http://dblp.l3s.de/d2r/resource/publications/conf/apccm/Fagin09
is dc:creator of	http://dblp.l3s.de/d2r/resource/publications/conf/birthday/FaginHHMPV09
is dc:creator of	http://dblp.l3s.de/d2r/resource/publications/conf/caap/Fagin83
is dc:creator of	http://dblp.l3s.de/d2r/resource/publications/conf/coco/FaginSV93
is dc:creator of	http://dblp.l3s.de/d2r/resource/publications/conf/concur/HalpernF88

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 .
}
```

SPARQL in a nutshell

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

SPARQL in a nutshell

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

SPARQL in a nutshell

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

SPARQL in a nutshell

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

SPARQL in a nutshell

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

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

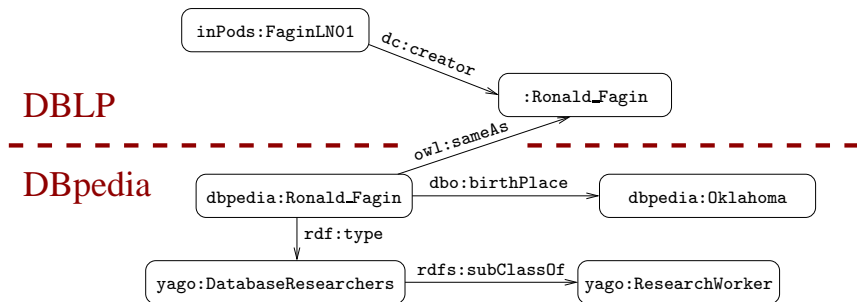
What are the challenges in implementing SPARQL?

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

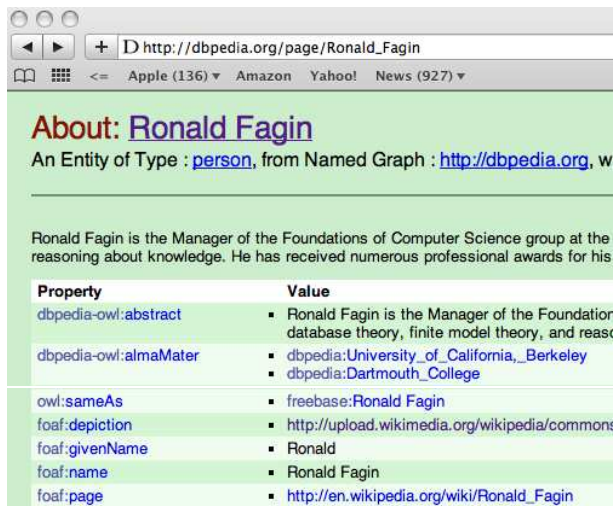
Extracting information from interconnected RDF graphs

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



Dereferenceable URIs are the glue

http://dbpedia.org/resource/Ronald_Fagin



The screenshot shows a web browser window with the address bar containing `http://dbpedia.org/page/Ronald_Fagin`. The page title is "About: [Ronald Fagin](#)". Below the title, it states "An Entity of Type : [person](#), from Named Graph : <http://dbpedia.org>, w".

Ronald Fagin is the Manager of the Foundations of Computer Science group at the reasoning about knowledge. He has received numerous professional awards for his

Property	Value
<code>dbpedia-owl:abstract</code>	<ul style="list-style-type: none">Ronald Fagin is the Manager of the Foundation database theory, finite model theory, and reaso
<code>dbpedia-owl:almaMater</code>	<ul style="list-style-type: none"><code>dbpedia:University_of_California,_Berkeley</code><code>dbpedia:Dartmouth_College</code>
<code>owl:sameAs</code>	<ul style="list-style-type: none"><code>freebase:Ronald Fagin</code>
<code>foaf:depiction</code>	<ul style="list-style-type: none">http://upload.wikimedia.org/wikipedia/commons
<code>foaf:givenName</code>	<ul style="list-style-type: none">Ronald
<code>foaf:name</code>	<ul style="list-style-type: none">Ronald Fagin
<code>foaf:page</code>	<ul style="list-style-type: none">http://en.wikipedia.org/wiki/Ronald_Fagin

Querying interconnected RDF graphs

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

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:PartOf      ?Conf .
  ?Conf       swrc:series      conf: pods .
  ?Person     owl:sameAs     ?Author .
  ?Person     dbo:birthPlace   dbpedia:Oklahoma .
}
```

Retrieving optional information

Retrieve the authors that have published in PODS, and their Web pages if this information is available:

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

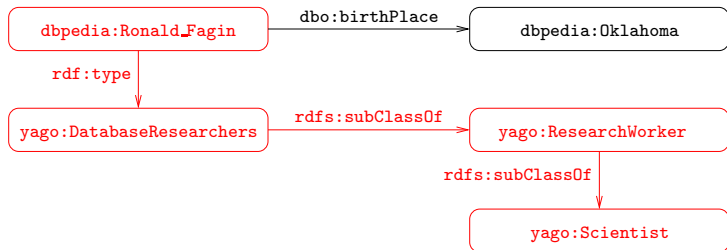
Taking into account vocabularies with predefined semantics

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

```
SELECT ?Author
WHERE
{
  ?Author      rdf:type      yago:Scientist .
  ?Author      dbo:birthPlace dbpedia:Oklahoma .
  ?Paper       dc:creator    ?Author .
  ?Paper       dct:PartOf    ?Conf .
  ?Conf        swrc:series    conf:pods .
}
```


Taking into account vocabularies with predefined semantics

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



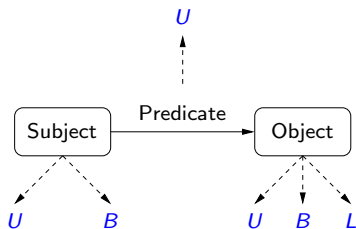
Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
- ▶ Concluding remarks

Outline of the talk

- ▶ **RDF formal model**
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
- ▶ Concluding remarks

RDF formal model

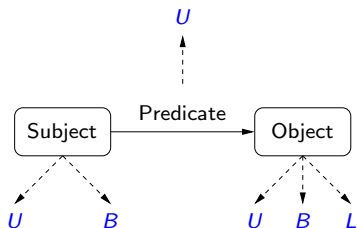


U : set of URIs

B : set of blank nodes

L : set of literals

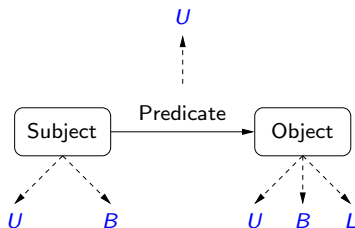
RDF formal model



- 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 set of RDF triples is called an **RDF graph**

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

Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
- ▶ Concluding remarks

SPARQL queries can be complex

Interesting features:

- ▶ Grouping
- ▶ Optional parts
- ▶ Nesting
- ▶ Union of patterns
- ▶ Filtering

```
{ P1  
  P2 }
```

SPARQL queries can be complex

Interesting features:

- ▶ **Grouping**
- ▶ Optional parts
- ▶ Nesting
- ▶ Union of patterns
- ▶ Filtering

```
{ { P1
  P2 }

  { P3
    P4 }

}
```

SPARQL queries can be complex

Interesting features:

- ▶ Grouping
- ▶ **Optional parts**
- ▶ Nesting
- ▶ Union of patterns
- ▶ Filtering

```
{ { P1
  P2
  OPTIONAL { P5 } }

{ P3
  P4
  OPTIONAL { P7 } }

}
```

SPARQL queries can be complex

Interesting features:

- ▶ Grouping
- ▶ Optional parts
- ▶ **Nesting**
- ▶ Union of patterns
- ▶ Filtering

```
{ { P1
  P2
  OPTIONAL { P5 } }

{ P3
  P4
  OPTIONAL { P7
    OPTIONAL { P8 } } }
}
```

SPARQL queries can be complex

Interesting features:

- ▶ Grouping
- ▶ Optional parts
- ▶ Nesting
- ▶ Union of patterns
- ▶ Filtering

```
{ { P1
  P2
  OPTIONAL { P5 } }

{ P3
  P4
  OPTIONAL { P7
    OPTIONAL { P8 } } }
}
UNION
{ P9 }
```

SPARQL queries can be complex

Interesting features:

- ▶ Grouping
- ▶ Optional parts
- ▶ Nesting
- ▶ Union of patterns
- ▶ **Filtering**

```
{ { P1
  P2
  OPTIONAL { P5 } }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 } } }
}
UNION
{ P9
  FILTER ( R ) }
```

SPARQL queries can be complex

Interesting features:

- ▶ Grouping
- ▶ Optional parts
- ▶ Nesting
- ▶ Union of patterns
- ▶ Filtering

```
{ { P1
  P2
  OPTIONAL { P5 } }

  { P3
    P4
    OPTIONAL { P7
      OPTIONAL { P8 } } }
}
UNION
{ P9
  FILTER ( R ) }
```

We focus on the body of SPARQL queries: **Pattern matching expressions**

A standard algebraic syntax

- ▶ Triple patterns: triples including variables from a set \mathbf{V}

`?X :name "john"`

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

- ▶ Graph patterns: full parenthesized algebra

`{ 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) }`

$(P_1 \text{ FILTER } R)$

original SPARQL syntax

algebraic syntax

A standard algebraic syntax

- ▶ **Explicit** precedence/association

Example

```
{ t1
  t2
  OPTIONAL { t3 }
  OPTIONAL { t4 }
  t5
}
```

$(((((t_1 \text{ AND } t_2) \text{ OPT } t_3) \text{ OPT } t_4) \text{ AND } t_5))$

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 graph pattern results in a set of mappings.

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 graph pattern results in a set of mappings.

The semantics of triple patterns

Given an RDF graph G and a triple pattern t .

Definition

The evaluation of t over G is the set of mappings μ such that:

- ▶ μ has as domain the variables in t : $\text{dom}(\mu) = \text{var}(t)$
- ▶ μ makes t to match the graph: $\mu(t) \in G$

The semantics of triple patterns

Given an RDF graph G and a triple pattern t .

Definition

The evaluation of t over G is the set of mappings μ such that:

- ▶ μ has as domain the variables in t : $\text{dom}(\mu) = \text{var}(t)$
- ▶ μ makes t to match the graph: $\mu(t) \in G$

Example

graph	triple	evaluation		
$(R_1, \text{name}, \text{john})$	$(?X, \text{name}, ?Y)$	$\mu_1:$	$?X$	$?Y$
$(R_1, \text{email}, \text{J@ed.ex})$			R_1	john
$(R_2, \text{name}, \text{paul})$			R_2	paul

The semantics of triple patterns

Given an RDF graph G and a triple pattern t .

Definition

The evaluation of t over G is the set of mappings μ such that:

- ▶ μ has as domain the variables in t : $\text{dom}(\mu) = \text{var}(t)$
- ▶ μ makes t to match the graph: $\mu(t) \in G$

Example

graph	triple	evaluation				
$(R_1, \text{name}, \text{john})$						
$(R_1, \text{email}, \text{J@ed.ex})$	$(?X, \text{name}, ?Y)$	μ_1 : <table border="1"><tr><td>?X</td><td>?Y</td></tr><tr><td>R_1</td><td>john</td></tr></table>	?X	?Y	R_1	john
?X	?Y					
R_1	john					
$(R_2, \text{name}, \text{paul})$		μ_2 : <table border="1"><tr><td>?X</td><td>?Y</td></tr><tr><td>R_2</td><td>paul</td></tr></table>	?X	?Y	R_2	paul
?X	?Y					
R_2	paul					

The semantics of triple patterns

Given an RDF graph G and a triple pattern t .

Definition

The evaluation of t over G is the set of mappings μ such that:

- ▶ μ has as domain the variables in t : $\text{dom}(\mu) = \text{var}(t)$
- ▶ μ makes t to match the graph: $\mu(t) \in G$

Example

graph	triple	evaluation				
$(R_1, \text{name}, \text{john})$						
$(R_1, \text{email}, \text{J@ed.ex})$	$(?X, \text{name}, ?Y)$	μ_1 : <table border="1"><tr><td>?X</td><td>?Y</td></tr><tr><td>R_1</td><td>john</td></tr></table>	?X	?Y	R_1	john
?X	?Y					
R_1	john					
$(R_2, \text{name}, \text{paul})$		μ_2 : <table border="1"><tr><td>?X</td><td>?Y</td></tr><tr><td>R_2</td><td>paul</td></tr></table>	?X	?Y	R_2	paul
?X	?Y					
R_2	paul					

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

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

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	

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	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

Compatible mappings

Definition

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

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	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

► μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let Ω_1 and Ω_2 be sets of mappings.

Definition

Join: extends mappings in Ω_1 with compatible mappings in Ω_2

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

Sets of mappings and operations

Let Ω_1 and Ω_2 be sets of mappings.

Definition

Join: extends mappings in Ω_1 with compatible mappings in Ω_2

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

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

- ▶ $\Omega_1 \setminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid \text{there is no mapping in } \Omega_2 \text{ compatible with } \mu_1\}$

Definition

Union: includes mappings in Ω_1 and in Ω_2

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

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

$$\blacktriangleright \Omega_1 \bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$$

Semantics of SPARQL

Given an RDF graph G .

Definition

$$\llbracket t \rrbracket_G =$$

$$\llbracket P_1 \text{ AND } P_2 \rrbracket_G =$$

$$\llbracket P_1 \text{ UNION } P_2 \rrbracket_G =$$

$$\llbracket P_1 \text{ OPT } P_2 \rrbracket_G =$$

Given an RDF graph G .

Definition

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

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

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

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

Example

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

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

Example

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

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

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

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
R_1	john	J@ed.ex
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
R_1	john	J@ed.ex
R_2	paul	

?X	?E
R_1	J@ed.ex

► from the **Join**

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 **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**

Filter expressions (value constraints)

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

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

We consider in R :

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

We impose a safety condition: $\text{var}(R) \subseteq \text{var}(P)$

Satisfaction of value constraints

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

Satisfaction of value constraints

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

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

Satisfaction of value constraints

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

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

Definition

FILTER : selects mappings that satisfy a condition

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

Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
- ▶ Concluding remarks

The evaluation problem

Input:

A mapping μ , a graph pattern P , and an RDF graph G

Question:

Does μ belong to the evaluation of P over G ?

Does $\mu \in \llbracket P \rrbracket_G$?

The evaluation problem

Input:

A mapping μ , a graph pattern P , and an RDF graph G

Question:

Does μ belong to the evaluation of P over G ?

Does $\mu \in \llbracket P \rrbracket_G$?

We study the *combined complexity* of the evaluation problem.

- ▶ μ , P and G are part of the input

Evaluation of simple patterns is polynomial

Theorem (Pérez, A. and Gutierrez 2006)

For patterns using only AND and FILTER operators (AND-FILTER fragment), the evaluation problem is polynomial:

$O(\text{size of the pattern} \times \text{size of the graph})$.

Evaluation of simple patterns is polynomial

Theorem (Pérez, A. and Gutierrez 2006)

For patterns using only AND and FILTER operators (AND-FILTER fragment), the evaluation problem is polynomial:

$$O(\text{size of the pattern} \times \text{size of the graph}).$$

Proof sketch

- ▶ Check that the mapping makes every triple to match
- ▶ Then check that the mapping satisfies the FILTERs

Evaluation including UNION is NP-complete

Theorem (Pérez, A. and Gutierrez 2006)

The evaluation problem is NP-complete for the AND-FILTER-UNION fragment of SPARQL.

Evaluation including UNION is NP-complete

Theorem (Pérez, A. and Gutierrez 2006)

The evaluation problem is NP-complete for the AND-FILTER-UNION fragment of SPARQL.

Proof sketch of hardness

- ▶ Reduction from 3SAT
- ▶ \neg bound is used to verify that a satisfying truth assignment is well defined

In general: Evaluation problem is PSPACE-complete

Theorem (Pérez, A. and Gutierrez 2006, 2009)

The evaluation problem for SPARQL is PSPACE-complete.

In general: Evaluation problem is PSPACE-complete

Theorem (Pérez, A. and Gutierrez 2006, 2009)

The evaluation problem for SPARQL is PSPACE-complete.

- ▶ *In fact, the evaluation problem remains PSPACE-hard for the AND-FILTER-OPT fragment of SPARQL*

In general: Evaluation problem is PSPACE-complete

Theorem (Pérez, A. and Gutierrez 2006, 2009)

The evaluation problem for SPARQL is PSPACE-complete.

- ▶ *In fact, the evaluation problem remains PSPACE-hard for the AND-FILTER-OPT fragment of SPARQL*

Theorem (Schmidt, Meier and Lausen 2010)

The evaluation problem remains PSPACE-complete for the OPT fragment of SPARQL.

What is the source of the high complexity?

The use of the OPT operator makes the evaluation problem harder.

- ▶ How can we deal with this operator? How can we reduce the complexity?

What is the source of the high complexity?

The use of the OPT operator makes the evaluation problem harder.

- ▶ How can we deal with this operator? How can we reduce the complexity?
- ▶ Later we will come back to this point

Research opportunities: SPARQL features under development

A new version of SPARQL is under development: **SPARQL 1.1**

It includes new features like:

Research opportunities: SPARQL features under development

A new version of SPARQL is under development: **SPARQL 1.1**

It includes new features like:

- ▶ Aggregates
 - ▶ Formal definition of the semantics

Research opportunities: SPARQL features under development

A new version of SPARQL is under development: **SPARQL 1.1**

It includes new features like:

- ▶ Aggregates
 - ▶ Formal definition of the semantics
- ▶ An operator SERVICE to distribute the execution of a query
 - ▶ Safety issues: (SERVICE ?X P)

Research opportunities: SPARQL features under development

A new version of SPARQL is under development: **SPARQL 1.1**

It includes new features like:

- ▶ Aggregates
 - ▶ Formal definition of the semantics
- ▶ An operator SERVICE to distribute the execution of a query
 - ▶ Safety issues: (SERVICE ?X P)
- ▶ Property paths based on regular expressions
 - ▶ Current semantics counts paths

SPARQL features under development

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

This query can be executed in the DBLP SPARQL endpoint:

- ▶ Answer: 969

Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
- ▶ Concluding remarks

Is SPARQL the right query language for RDF?

Semantics of RDF is open world.

Given an RDF graph G : We know that the triples in G hold

- ▶ But we have no information about the triples that are not included in G

If H is an RDF graph such that $G \subseteq H$, then H is a possible *interpretation* of G .

Is the semantics of SPARQL appropriate for RDF

How can a query be answered over a graph with infinitely many interpretations?

- ▶ Certain answer semantics is appropriate for this scenario

Certain answers of a graph pattern P over an RDF graph G :

$$\text{CERTAINANSWERS}(P, G) = \bigcap_{H: G \subseteq H} \llbracket P \rrbracket_H$$

Is the semantics of SPARQL appropriate for RDF

How can a query be answered over a graph with infinitely many interpretations?

- ▶ Certain answer semantics is appropriate for this scenario

Certain answers of a graph pattern P over an RDF graph G :

$$\text{CERTAINANSWERS}(P, G) = \bigcap_{H: G \subseteq H} \llbracket P \rrbracket_H$$

We have two alternative semantics for SPARQL queries.

- ▶ Is it true that $\llbracket P \rrbracket_G = \text{CERTAINANSWERS}(P, G)$?

Monotone queries are appropriate for RDF

A graph pattern P is monotone if:

For every pair G_1, G_2 of RDF graphs: $G_1 \subseteq G_2 \Rightarrow \llbracket P \rrbracket_{G_1} \subseteq \llbracket P \rrbracket_{G_2}$

Proposition

Every query in the AND-FILTER-UNION fragment of SPARQL is monotone.

This fragment is positive.

Monotone queries are appropriate for RDF

Corollary

Given a query P in the AND-FILTER-UNION fragment of SPARQL and an RDF graph G :

$$\llbracket P \rrbracket_G = \text{CERTAINANSWERS}(P, G)$$

What about the OPT operator?

Is the OPT operator positive?

- ▶ If this is the case, then SPARQL is appropriate for the open-world semantics of RDF

What about the OPT operator?

Is the OPT operator positive?

- ▶ If this is the case, then SPARQL is appropriate for the open-world semantics of RDF

Graph patterns do not form a positive language!

- ▶ We will see why ...

Are graph patterns including the OPT operator monotone?

Notion of monotonicity is not appropriate for the OPT operator.

- ▶ This operator can add information to a mapping

Given mappings μ_1, μ_2 : μ_1 is subsumed by μ_2 ($\mu_1 \preceq \mu_2$) if

1. $\text{dom}(\mu_1) \subseteq \text{dom}(\mu_2)$
2. $\mu_1(?X) = \mu_2(?X)$ for every $?X \in \text{dom}(\mu_1)$

Given sets Ω_1, Ω_2 of mappings: Ω_1 is subsumed by Ω_2 ($\Omega_1 \sqsubseteq \Omega_2$) if for every $\mu_1 \in \Omega_1$, there exists $\mu_2 \in \Omega_2$ such that $\mu_1 \preceq \mu_2$.

Are graph patterns including the OPT operator monotone?

A graph pattern P is weakly monotone if:

For every pair G_1, G_2 of RDF graphs: $G_1 \subseteq G_2 \Rightarrow \llbracket P \rrbracket_{G_1} \sqsubseteq \llbracket P \rrbracket_{G_2}$

Are graph patterns including the OPT operator monotone?

A graph pattern P is weakly monotone if:

For every pair G_1, G_2 of RDF graphs: $G_1 \subseteq G_2 \Rightarrow \llbracket P \rrbracket_{G_1} \sqsubseteq \llbracket P \rrbracket_{G_2}$

Weakly monotone graph patterns are appropriate for the open-world semantics of RDF:

Observation

If graph pattern P is weakly monotone, then for every RDF graph G : $\llbracket P \rrbracket_G$ is a greatest lower bound of $\{\llbracket P \rrbracket_H \mid G \subseteq H\}$ w.r.t. \sqsubseteq

Are graph patterns including the OPT operator **weakly** monotone?

If the answer to this question is positive, then SPARQL is appropriate for the open-world semantics of RDF.

Are graph patterns including the OPT operator **weakly** monotone?

If the answer to this question is positive, then SPARQL is appropriate for the open-world semantics of RDF.

But the answer is negative.

Are graph patterns including the OPT operator **weakly** monotone?

If the answer to this question is positive, then SPARQL is appropriate for the open-world semantics of RDF.

But the answer is negative.

- ▶ In fact, we can express a minus operator

A MINUS operator in SPARQL

Let MINUS be defined as:

$$\llbracket P_1 \text{ MINUS } P_2 \rrbracket_G = \llbracket P_1 \rrbracket_G \setminus \llbracket P_2 \rrbracket_G$$

A MINUS operator in SPARQL

Let MINUS be defined as:

$$\llbracket P_1 \text{ MINUS } P_2 \rrbracket_G = \llbracket P_1 \rrbracket_G \setminus \llbracket P_2 \rrbracket_G$$

Proposition

$(P_1 \text{ MINUS } P_2)$ is equivalent to:

$$\left(P_1 \text{ OPT } (P_2 \text{ AND } (?X_1, ?X_2, ?X_3)) \right) \text{ FILTER } \neg \text{bound}(?X_1),$$

where $?X_1, ?X_2, ?X_3$ are mentioned neither in P_1 nor in P_2 .

What went wrong?

The queries used to express the MINUS operator are not natural.

- ▶ Interestingly, there is a common syntactic pattern between these queries and the queries used in the proofs of PSPACE-hardness of the evaluation problem

What went wrong?

The queries used to express the MINUS operator are not natural.

- ▶ Interestingly, there is a common syntactic pattern between these queries and the queries used in the proofs of PSPACE-hardness of the evaluation problem

This allows us to identify a fragment of SPARQL that:

What went wrong?

The queries used to express the MINUS operator are not natural.

- ▶ Interestingly, there is a common syntactic pattern between these queries and the queries used in the proofs of PSPACE-hardness of the evaluation problem

This allows us to identify a fragment of SPARQL that:

- ▶ is appropriate for the open-world semantics of RDF

What went wrong?

The queries used to express the MINUS operator are not natural.

- ▶ Interestingly, there is a common syntactic pattern between these queries and the queries used in the proofs of PSPACE-hardness of the evaluation problem

This allows us to identify a fragment of SPARQL that:

- ▶ is appropriate for the open-world semantics of RDF
- ▶ can be evaluated more efficiently

Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
- ▶ Concluding remarks


Identifying a good fragment of SPARQL

Graph patterns in the proofs of PSPACE-hardness in [Pérez, A. and Gutierrez 2006, 2009] and [Schmidt, Meier and Lausen 2010] are not natural:

$$(a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi))$$

Identifying a good fragment of SPARQL


Graph patterns in the proofs of PSPACE-hardness in [Pérez, A. and Gutierrez 2006, 2009] and [Schmidt, Meier and Lausen 2010] are not natural:

$(a, \text{true}, ?B_0)$ OPT $(P_1$ OPT $(Q_1$ AND $P_\psi))$


Identifying a good fragment of SPARQL

Graph patterns in the proofs of PSPACE-hardness in [Pérez, A. and Gutierrez 2006, 2009] and [Schmidt, Meier and Lausen 2010] are not natural:

$(a, \text{true}, ?B_0)$ OPT $(P_1$ OPT $(Q_1$ AND $P_\psi))$



Identifying a good fragment of SPARQL

Graph patterns in the proofs of PSPACE-hardness in [Pérez, A. and Gutierrez 2006, 2009] and [Schmidt, Meier and Lausen 2010] are not natural:

$(a, \text{true}, ?B_0)$ OPT (P_1) OPT $(Q_1$ AND $P_\psi)$)

\uparrow \uparrow \uparrow

$?B_0$ \times $?B_0$

Identifying a good fragment of SPARQL

Graph patterns in the proofs of PSPACE-hardness in [Pérez, A. and Gutierrez 2006, 2009] and [Schmidt, Meier and Lausen 2010] are not natural:

$(a, \text{true}, ?B_0)$ OPT $(P_1$ OPT $(Q_1$ AND $P_\psi))$

\uparrow \uparrow \uparrow


$?B_0$ \times $?B_0$

Is $?B_0$ giving optional information for P_1 ?

Identifying a good fragment of SPARQL

Graph patterns in the proofs of PSPACE-hardness in [Pérez, A. and Gutierrez 2006, 2009] and [Schmidt, Meier and Lausen 2010] are not natural:

$(a, \text{true}, ?B_0)$ OPT (P_1) OPT $(Q_1 \text{ AND } P_\psi)$



Is $?B_0$ giving optional information for P_1 ?

- ▶ No, $?B_0$ is giving optional information for $(a, \text{true}, ?B_0)$?

Well-designed graph patterns

Definition

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

$$(\dots\dots\dots (P \text{ OPT } Q) \dots\dots\dots)$$

if a variable occurs **inside Q** and **anywhere outside the OPT operator**, then the variable **must also occur inside P** .

Well-designed graph patterns

Definition

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

$$\left(\dots \left(P \text{ OPT } Q \right) \dots \right)$$

↑

if a variable occurs **inside Q** and **anywhere outside the OPT operator**, then the variable **must also occur inside P** .

Well-designed graph patterns

Definition

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

$$\left(\dots \left(P \text{ OPT } Q \right) \dots \right)$$


if a variable occurs **inside Q** and **anywhere outside the OPT operator**, then the variable **must also occur inside P** .

Well-designed graph patterns

Definition

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

$$\left(\dots \left(P \text{ OPT } Q \right) \dots \right)$$


if a variable occurs **inside Q** and **anywhere outside the OPT operator**, then the variable **must also occur inside P** .

Well-designed graph patterns

Definition

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

$$\left(\dots \left(P \text{ OPT } Q \right) \dots \right)$$

↑ ↑ ↑ ↑

if a variable occurs **inside Q** and **anywhere outside the OPT operator**, then the variable **must also occur inside P** .

Example

$$\left((?Y, \text{name}, \text{paul}) \text{ OPT } (?X, \text{email}, ?Z) \right) \text{ AND } (?X, \text{name}, \text{john})$$

Well-designed graph patterns

Definition

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

$$\left(\dots \left(P \text{ OPT } Q \right) \dots \right)$$

↑
↑
↑
↑

if a variable occurs **inside Q** and **anywhere outside the OPT operator**, then the variable **must also occur inside P** .

Example

$$\left((?Y, \text{name}, \text{paul}) \text{OPT} (?X, \text{email}, ?Z) \right) \text{AND} (?X, \text{name}, \text{john})$$

↑

Well-designed graph patterns

Definition

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

$$\left(\dots \left(P \text{ OPT } Q \right) \dots \right)$$

↑ ↑ ↑ ↑

if a variable occurs **inside Q** and **anywhere outside the OPT operator**, then the variable **must also occur inside P** .

Example

$$\left((?Y, name, paul) \text{ OPT } (?X, email, ?Z) \right) \text{ AND } (?X, name, john)$$

 ↑ ↑

Well-designed graph patterns

Definition

A query in the AND-FILTER-OPT fragment of SPARQL is well-designed if for every OPT in the pattern:

$$\left(\dots \left(P \text{ OPT } Q \right) \dots \right)$$

↑ ↑ ↑ ↑

if a variable occurs **inside Q** and **anywhere outside the OPT operator**, then the variable **must also occur inside P** .

Example

$$\left(\left(?Y, \text{ name, paul} \right) \text{ OPT } \left(?X, \text{ email, ?Z} \right) \right) \text{ AND } \left(?X, \text{ name, john} \right)$$

× ↑ ↑

How common are well-designed patterns?

What are real SPARQL queries like? [Picalausa and Vansummeren, 2011]

- ▶ DBpedia query log: 623,000 queries without the UNION operator

How common are well-designed patterns?

What are real SPARQL queries like? [Picalausa and Vansummeren, 2011]

- ▶ DBpedia query log: 623,000 queries without the UNION operator
- ▶ 52% of these queries are well designed

How common are well-designed patterns?

What are real SPARQL queries like? [Picalausa and Vansummeren, 2011]

- ▶ DBpedia query log: 623,000 queries without the UNION operator
- ▶ 52% of these queries are well designed

Examples of queries from DBpedia that are not well designed are unnatural

How common are well-designed patterns?

What are real SPARQL queries like? [Picalausa and Vansummeren, 2011]

- ▶ DBpedia query log: 623,000 queries without the UNION operator
- ▶ 52% of these queries are well designed

Examples of queries from DBpedia that are not well designed are unnatural

- ▶ In general, they are equivalent to or can be reformulated as well-designed graph patterns

Theorem (Pérez, A. and Gutierrez 2009)

*The evaluation problem is **coNP-complete** for well-designed graph patterns.*

Theorem (Pérez, A. and Gutierrez 2009)

*The evaluation problem is **coNP-complete** for well-designed graph patterns.*

Can we use this in practice?

- ▶ Well-designed graph patterns are suitable for optimization

Classical optimization

- ▶ Classical optimization assumes **null-rejection**.

Classical optimization

- ▶ Classical optimization assumes **null-rejection**.
 - ▶ Null-rejection: the join condition must fail in the presence of nulls

Classical optimization

- ▶ Classical optimization assumes **null-rejection**.
 - ▶ Null-rejection: the join condition must fail in the presence of nulls

- ▶ SPARQL operations are **not null-rejecting**.
 - ▶ By definition of compatible mappings

Classical optimization

- ▶ Classical optimization assumes **null-rejection**.
 - ▶ Null-rejection: the join condition must fail in the presence of nulls
- ▶ SPARQL operations are **not null-rejecting**.
 - ▶ By definition of compatible mappings
- ▶ Can we use classical optimization in the context of SPARQL?

Classical optimization

- ▶ Classical optimization assumes **null-rejection**.
 - ▶ Null-rejection: the join condition must fail in the presence of nulls
- ▶ SPARQL operations are **not null-rejecting**.
 - ▶ By definition of compatible mappings
- ▶ Can we use classical optimization in the context of SPARQL?
 - ▶ **Well-designed graph patterns are suitable for reordering, and then for classical optimization**

Well-designed graph patterns and optimization

Consider the following rules:

$$((P_1 \text{ OPT } P_2) \text{ FILTER } R) \longrightarrow ((P_1 \text{ FILTER } R) \text{ OPT } P_2) \quad (1)$$

$$(P_1 \text{ AND } (P_2 \text{ OPT } P_3)) \longrightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3) \quad (2)$$

$$((P_1 \text{ OPT } P_2) \text{ AND } P_3) \longrightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2) \quad (3)$$

Proposition (Pérez, A. and Gutierrez 2006)

If P is well-designed and Q is obtained from P by applying either (1) or (2) or (3), then Q is a well-designed and equivalent to P .

Well-designed graph patterns are appropriate for the open-world semantics of RDF

Theorem

Every well-designed graph pattern is weakly monotone.

Well-designed graph patterns are appropriate for the open-world semantics of RDF

Theorem

Every well-designed graph pattern is weakly monotone.

That is, if P is well designed and G_1, G_2 are RDF graphs such that $G_1 \subseteq G_2$, then $\llbracket P \rrbracket_{G_1} \sqsubseteq \llbracket P \rrbracket_{G_2}$

Research opportunities: Well-designed graph patterns form a good fragment of SPARQL

Open questions:

Research opportunities: Well-designed graph patterns form a good fragment of SPARQL

Open questions:

- ▶ How the notion of being well-designed can be extended to consider the UNION operator?

Research opportunities: Well-designed graph patterns form a good fragment of SPARQL

Open questions:

- ▶ How the notion of being well-designed can be extended to consider the UNION operator?
- ▶ How far are well-designed graph patterns from weakly monotone SPARQL queries?

Research opportunities: Well-designed graph patterns form a good fragment of SPARQL

Open questions:

- ▶ How the notion of being well-designed can be extended to consider the UNION operator?
- ▶ How far are well-designed graph patterns from weakly monotone SPARQL queries?
- ▶ Are the optimization techniques useful in practice?
 - ▶ They have been used with good results [Buil-Aranda, A. and Corcho 2011]

Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
- ▶ Concluding remarks

Syntax of RDFS

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

Syntax of RDFS

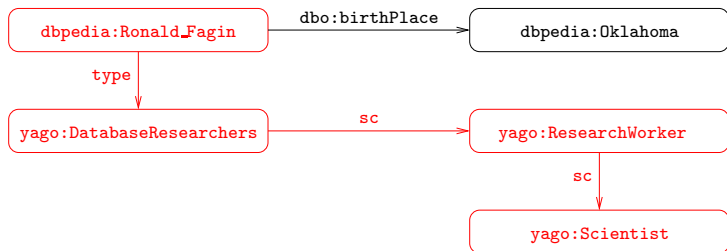
RDFS extends RDF with a schema vocabulary: subPropertyOf (**sp**), subClassOf (**sc**), domain (**dom**), range (**range**), type (**type**).

How can one query RDFS data?

- ▶ Evaluating queries which involve this vocabulary is challenging
- ▶ There is not yet consensus in the Semantic Web community on how to define a query language for RDFS

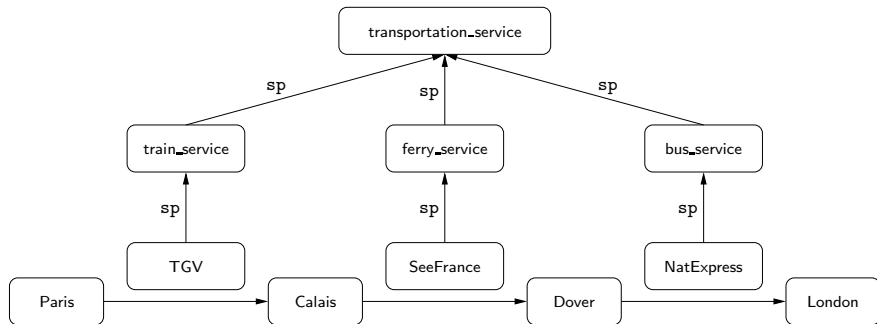
A simple SPARQL query:

(dbpedia:Ronald_Fagin, type, yago:Scientist)



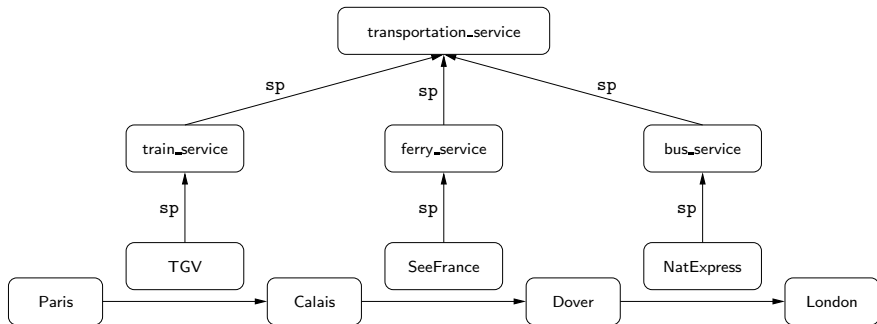
A more complex query

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



A more complex query

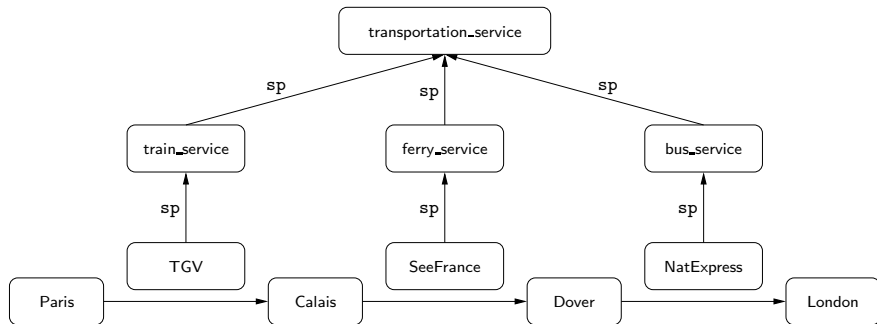
List the pairs a, b of cities such that there is a way to travel from a to b .



nSPARQL:

A more complex query

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

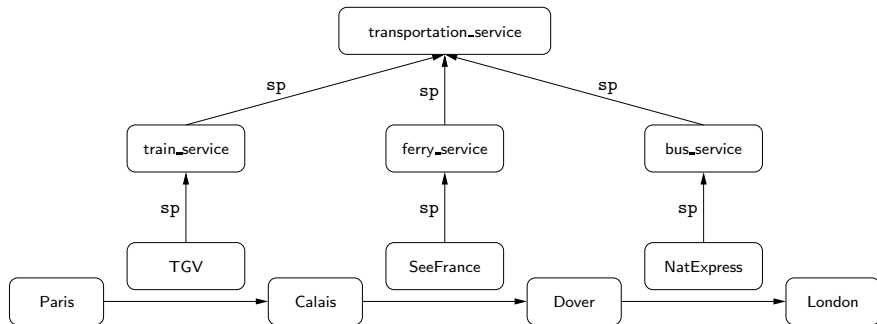


nSPARQL:

($?X$, $?Y$)

A more complex query

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

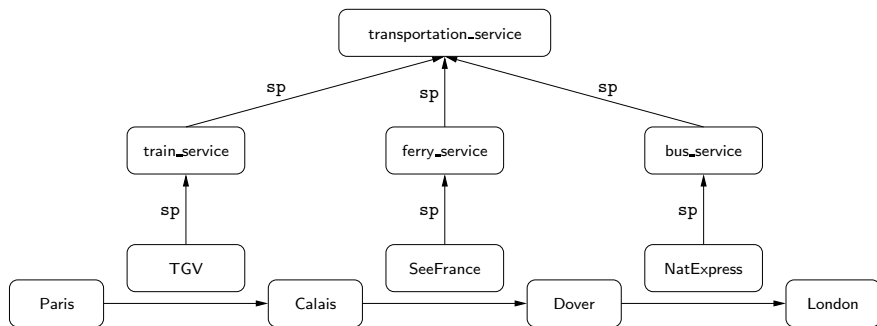


nSPARQL:

$(?X, (\text{next}:: \text{ })^+, ?Y)$

A more complex query

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

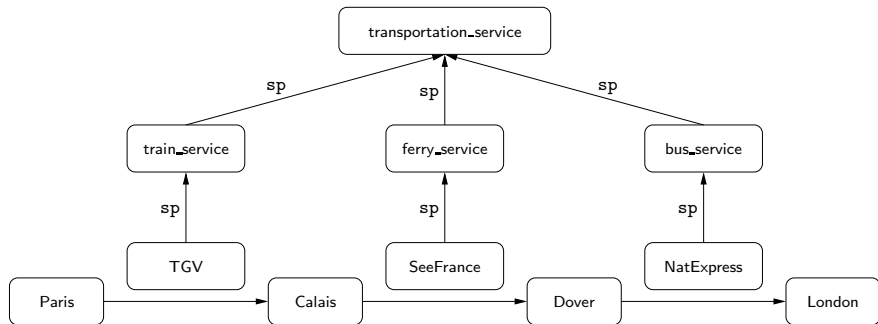


nSPARQL:

$(?X, (\text{next}::[\quad])^+, ?Y)$

A more complex query

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

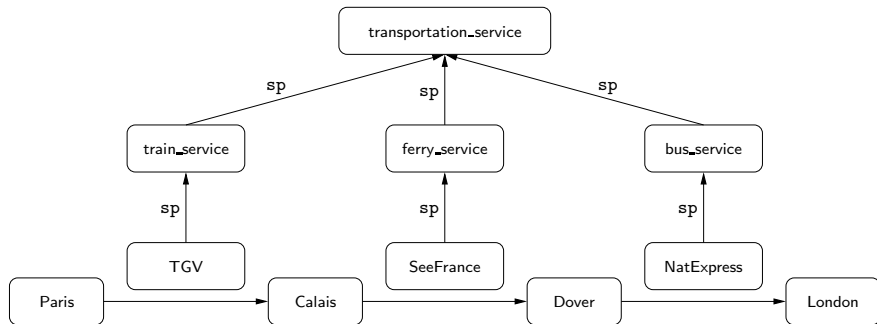


nSPARQL:

$(?X, (\text{next}::[(\text{next}::\text{sp})^* /])^+, ?Y)$

A more complex query

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

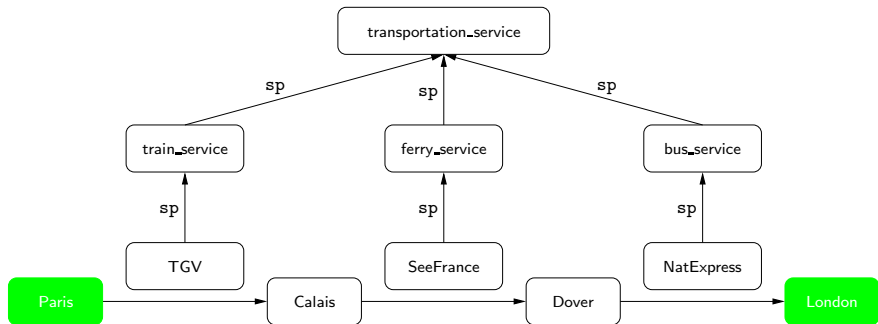


nSPARQL:

```
(?X, (next::[(next::sp)*/(self::transportation_service)]+, ?Y)
```

A more complex query

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

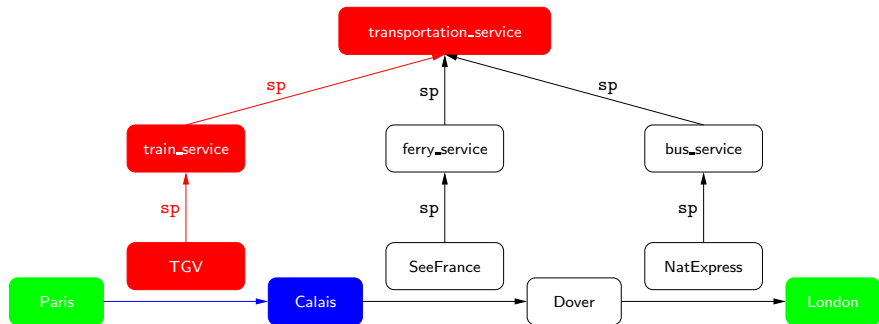


nSPARQL:

```
(?X, (next::[(next::sp)*/(self::transportation_service)]+, ?Y)
```

A more complex query

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

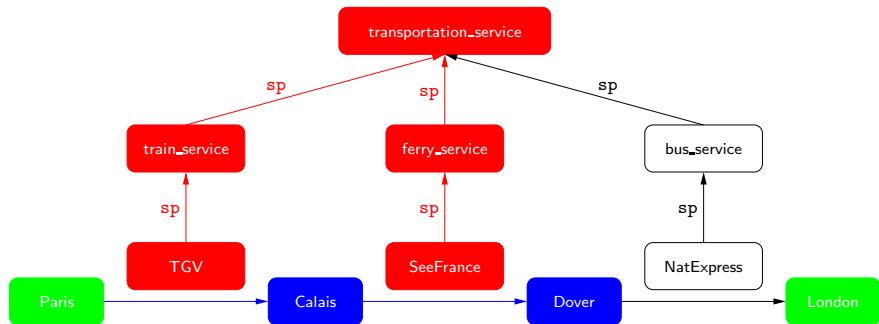


nSPARQL:

```
(?X, (next::[(next::sp)*/(self::transportation_service)]+, ?Y)
```

A more complex query

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

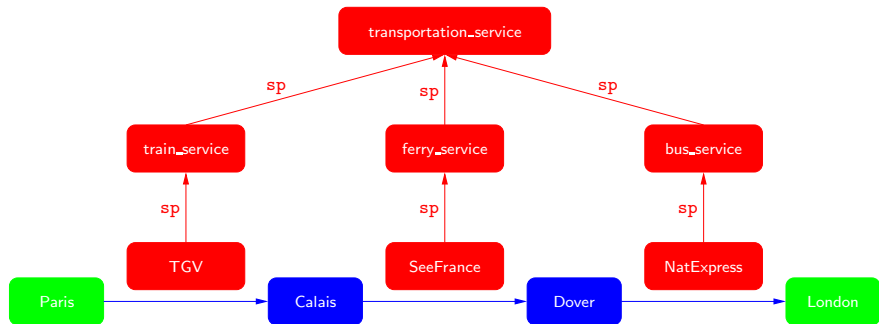


nSPARQL:

$(?X, (\text{next}::[(\text{next}::\text{sp})^*/(\text{self}::\text{transportation_service})])^+, ?Y)$

A more complex query

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



nSPARQL:

```
(?X, (next::[(next::sp)*/(self::transportation_service)]+, ?Y)
```

Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
- ▶ **Concluding remarks**

Concluding remarks

- ▶ We have witnessed a constant growth in the amount of RDF data available on the Web.
 - ▶ Two fundamental components: RDF and SPARQL
- ▶ Some of the distinctive features of RDF have made the study and implementation of SPARQL challenging.
- ▶ RDF and SPARQL have attracted interest from the database community.
 - ▶ There are many interesting research opportunities in the area

Concluding remarks

- ▶ We have witnessed a constant growth in the amount of RDF data available on the Web.
 - ▶ Two fundamental components: RDF and SPARQL
- ▶ Some of the distinctive features of RDF have made the study and implementation of SPARQL challenging.
- ▶ RDF and SPARQL have attracted interest from the database community.
 - ▶ There are many interesting research opportunities in the area
- ▶ The database community has much more to say about these technologies, and, in particular, about the fundamental database problems that need to be solved in order to provide solid foundations for their development.

Thank you!

Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
 - ▶ **Semantics of RDFS**
 - ▶ Extending SPARQL with navigational capabilities
- ▶ Concluding remarks

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

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

The notion of entailment in RDFS can be defined in terms of classical notions such as model, interpretation, etc.

- ▶ As for the case of first-order logic

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

An inference system for RDFS

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

$\frac{(A,sp,B) (X,A,Y)}{(X,B,Y)}$

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

$\frac{(A,sc,B) (X,type,A)}{(X,type,B)}$

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

$\frac{(A,range,B) (X,A,Y)}{(Y,type,B)}$

Theorem (Hayes 2003, Gutierrez, Hurtado and Mendelzon 2004, Muñoz, Pérez and Gutierrez 2007)

The previous system of inference rules characterize the notion of entailment in RDFS.

Thus, a triple t can be deduced from an RDF graph G ($G \models t$) if there exists an RDF G' such that:

- ▶ $t \in G'$
- ▶ G' can be obtained from G by successively applying the rules in the previous system

Definition

The closure of an RDFS graph G , denoted by $\text{cl}(G)$, is the graph obtained by adding to G all the triples that are implied by G .

A basic property of the closure:

- ▶ $G \models t$ iff $t \in \text{cl}(G)$

Outline of the talk

- ▶ RDF formal model
- ▶ A formal study of SPARQL
 - ▶ Syntax and semantics
 - ▶ Complexity of the evaluation problem
- ▶ Is SPARQL the right query language for RDF?
- ▶ Well-designed graphs patterns
- ▶ Including RDFS vocabulary: Motivation
 - ▶ Semantics of RDFS
 - ▶ Extending SPARQL with navigational capabilities
- ▶ Concluding remarks

Basic step for answering queries over RDFS:

- ▶ Checking whether a triple t is in $cl(G)$

Basic step for answering queries over RDFS:

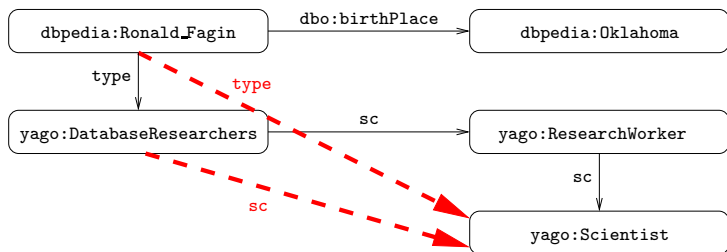
- ▶ Checking whether a triple t is in $\text{cl}(G)$

Definition

The *RDFS-evaluation* of a graph pattern P over an RDFS graph G is defined as the evaluation of P over $\text{cl}(G)$:

$$\llbracket P \rrbracket_G^{\text{rdfs}} = \llbracket P \rrbracket_{\text{cl}(G)}$$

Example: (dbpedia:Ronald_Fagin, type, yago:Scientist) over the closure



Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G :

- ▶ Compute $cl(G)$, and then evaluate P over $cl(G)$ as for RDF

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G :

- ▶ Compute $cl(G)$, and then evaluate P over $cl(G)$ as for RDF

This approach has some drawbacks:

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G :

- ▶ Compute $cl(G)$, and then evaluate P over $cl(G)$ as for RDF

This approach has some drawbacks:

- ▶ The size of the closure of G can be quadratic in the size of G

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G :

- ▶ Compute $cl(G)$, and then evaluate P over $cl(G)$ as for RDF

This approach has some drawbacks:

- ▶ The size of the closure of G can be quadratic in the size of G
- ▶ Once the closure has been computed, all the queries are evaluated over a graph which can be much larger than the original graph

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G :

- ▶ Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF

This approach has some drawbacks:

- ▶ The size of the closure of G can be quadratic in the size of G
- ▶ Once the closure has been computed, all the queries are evaluated over a graph which can be much larger than the original graph
- ▶ The approach is not goal-oriented

When evaluating (a, sc, b) , a goal-oriented approach should not compute $\text{cl}(G)$:

- ▶ It should just verify whether there exists a path from a to b in G where every edge has label sc

Extending SPARQL with navigational capabilities

The example (a , sc , b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Extending SPARQL with navigational capabilities

The example (a , sc , b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

Extending SPARQL with navigational capabilities

The example (a, sc, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- ▶ A query P over an RDFS graph G is answered by navigating G ($cl(G)$ is not computed)

Extending SPARQL with navigational capabilities

The example (a, sc, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- ▶ A query P over an RDFS graph G is answered by navigating G ($cl(G)$ is not computed)

This approach has some advantages:

Extending SPARQL with navigational capabilities

The example (a, sc, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- ▶ A query P over an RDFS graph G is answered by navigating G ($cl(G)$ is not computed)

This approach has some advantages:

- ▶ It is goal-oriented

Extending SPARQL with navigational capabilities

The example (a, sc, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- ▶ A query P over an RDFS graph G is answered by navigating G ($cl(G)$ is not computed)

This approach has some advantages:

- ▶ It is goal-oriented
- ▶ It has been used to design query languages for XML (e.g., XPath and XQuery). The results for these languages can be used here

Extending SPARQL with navigational capabilities

The example (a, sc, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

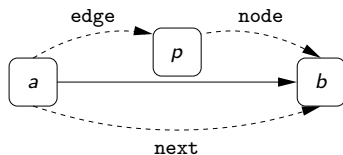
- ▶ A query P over an RDFS graph G is answered by navigating G ($cl(G)$ is not computed)

This approach has some advantages:

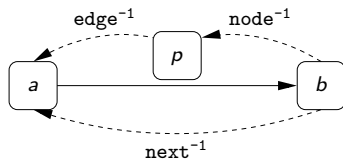
- ▶ It is goal-oriented
- ▶ It has been used to design query languages for XML (e.g., XPath and XQuery). The results for these languages can be used here
- ▶ Navigational operators allow to express natural queries that are **not expressible in SPARQL over RDFS**

Navigational axes

Forward axes for an RDF triple (a, p, b) :



Backward axes for an RDF triple (a, p, b) :



The basic component: Nested regular expressions

Syntax of nested regular expressions:

$$\begin{aligned} \text{exp} &:= \text{axis} \mid \text{axis}::a \mid \\ &\quad \text{axis}::[\text{exp}] \mid \text{exp}/\text{exp} \mid \text{exp}|\text{exp} \mid \text{exp}^* \end{aligned}$$

where $a \in U$ and $\text{axis} \in \{\text{self}, \text{next}, \text{next}^{-1}, \text{edge}, \text{edge}^{-1}, \text{node}, \text{node}^{-1}\}$.

Semantics of nested regular expressions

Given an RDFS graph G :

Semantics of nested regular expressions

Given an RDFS graph G :

$$\begin{aligned} \llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \text{ is in } G\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \exists z \in U (x, z, y) \in G\} \\ \llbracket \text{edge} \rrbracket_G &= \{(x, y) \mid \exists z \in U (x, y, z) \in G\} \end{aligned}$$

Semantics of nested regular expressions

Given an RDFS graph G :

$$\begin{aligned} \llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \text{ is in } G\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \exists z \in U (x, z, y) \in G\} \\ \llbracket \text{edge} \rrbracket_G &= \{(x, y) \mid \exists z \in U (x, y, z) \in G\} \end{aligned}$$

$$\begin{aligned} \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\} \\ \llbracket \text{next}::a \rrbracket_G &= \{(x, y) \mid (x, a, y) \in G\} \\ \llbracket \text{edge}::a \rrbracket_G &= \{(x, y) \mid (x, y, a) \in G\} \end{aligned}$$

Semantics of nested regular expressions

Given an RDFS graph G :

$$\begin{aligned} \llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \text{ is in } G\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \exists z \in U (x, z, y) \in G\} \\ \llbracket \text{edge} \rrbracket_G &= \{(x, y) \mid \exists z \in U (x, y, z) \in G\} \end{aligned}$$

$$\begin{aligned} \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\} \\ \llbracket \text{next}::a \rrbracket_G &= \{(x, y) \mid (x, a, y) \in G\} \\ \llbracket \text{edge}::a \rrbracket_G &= \{(x, y) \mid (x, y, a) \in G\} \end{aligned}$$

$$\begin{aligned} \llbracket \text{exp}_1/\text{exp}_2 \rrbracket_G &= \{(x, y) \mid \exists z (x, z) \in \llbracket \text{exp}_1 \rrbracket_G \text{ and} \\ &\quad (z, y) \in \llbracket \text{exp}_2 \rrbracket_G\} \\ \llbracket \text{exp}_1\text{exp}_2 \rrbracket_G &= \llbracket \text{exp}_1 \rrbracket_G \cup \llbracket \text{exp}_2 \rrbracket_G \\ \llbracket \text{exp}^* \rrbracket_G &= \llbracket \text{self} \rrbracket_G \cup \llbracket \text{exp} \rrbracket_G \cup \llbracket \text{exp/exp} \rrbracket_G \cup \\ &\quad \llbracket \text{exp/exp/exp} \rrbracket_G \cup \dots \end{aligned}$$

Semantics of nested regular expressions (cont'd)

Given an RDFS graph G :

Semantics of nested regular expressions (cont'd)

Given an RDFS graph G :

$$\llbracket \text{next}::[\text{exp}] \rrbracket_G = \{(x, y) \mid \exists z, w \in U \ (x, z, y) \in G \text{ and} \\ (z, w) \in \llbracket \text{exp} \rrbracket_G\}$$

Semantics of nested regular expressions (cont'd)

Given an RDFS graph G :

$$\llbracket \text{next}::[\text{exp}] \rrbracket_G = \{(x, y) \mid \exists z, w \in U \ (x, z, y) \in G \text{ and} \\ (z, w) \in \llbracket \text{exp} \rrbracket_G\}$$

$$\llbracket \text{edge}::[\text{exp}] \rrbracket_G = \{(x, y) \mid \exists z, w \in U \ (x, y, z) \in G \text{ and} \\ (z, w) \in \llbracket \text{exp} \rrbracket_G\}$$

The query language nSPARQL

Syntax of nSPARQL:

- ▶ Basic component: A triple of the form (x, exp, y)
 - ▶ exp is a nested regular expression
 - ▶ x (resp. y) is either an element from U or a variable
- ▶ Operators: AND, FILTER, UNION and OPT

The query language nSPARQL

Syntax of nSPARQL:

- ▶ Basic component: A triple of the form (x, exp, y)
 - ▶ exp is a nested regular expression
 - ▶ x (resp. y) is either an element from U or a variable
- ▶ Operators: AND, FILTER, UNION and OPT

Triple $(?X, ?Y, ?Z)$ is not allowed.

The query language nSPARQL

Syntax of nSPARQL:

- ▶ Basic component: A triple of the form (x, exp, y)
 - ▶ exp is a nested regular expression
 - ▶ x (resp. y) is either an element from U or a variable
- ▶ Operators: AND, FILTER, UNION and OPT

Triple $(?X, ?Y, ?Z)$ is not allowed.

- ▶ It computes the closure!

nSPARQL: What can we express?

Example

- ▶ $(?X, \text{next}::a, ?Y)$: Equivalent to $(?X, a, ?Y)$
- ▶ $(?X, \text{edge}::a, ?Y)$: Equivalent to $(?X, ?Y, a)$
- ▶ $(?X, \text{node}::a, ?Y)$: Equivalent to $(a, ?X, ?Y)$

nSPARQL: What can we express?

Example

- ▶ $(?X, \text{next}::a, ?Y)$: Equivalent to $(?X, a, ?Y)$
- ▶ $(?X, \text{edge}::a, ?Y)$: Equivalent to $(?X, ?Y, a)$
- ▶ $(?X, \text{node}::a, ?Y)$: Equivalent to $(a, ?X, ?Y)$
- ▶ $(?X, (\text{next}::(\text{sc}))^+, ?Y)$: Verifies whether $?X$ is a subclass of $?Y$

Semantics of nSPARQL

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

Semantics of nSPARQL

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

- ▶ The domain of μ is $\{?X, ?Y\}$

Semantics of nSPARQL

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

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

Semantics of nSPARQL

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

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

Example

What does $(?X, (\text{next::KLM} \mid \text{next::AirFrance})^+, ?Y)$ represent?

Is nSPARQL a good language for RDFS?

How do we test whether a language is appropriate for RDFS?

- ▶ Can we capture SPARQL over RDFS?

Is nSPARQL a good language for RDFS?

How do we test whether a language is appropriate for RDFS?

- ▶ Can we capture SPARQL over RDFS?

For every RDFS graph G and SPARQL pattern P , we would like to find an nSPARQL pattern Q such that:

$$\llbracket P \rrbracket_G^{\text{rdfs}} = \llbracket Q \rrbracket_G$$

Is nSPARQL a good language for RDFS?

How do we test whether a language is appropriate for RDFS?

- ▶ Can we capture SPARQL over RDFS?

For every RDFS graph G and SPARQL pattern P , we would like to find an nSPARQL pattern Q such that:

$$\llbracket P \rrbracket_G^{\text{rdfs}} = \llbracket Q \rrbracket_G$$

But we trivially fail because of triple $(?X, ?Y, ?Z)$.

Is nSPARQL a good language for RDFS?

How do we test whether a language is appropriate for RDFS?

- ▶ Can we capture SPARQL over RDFS?

For every RDFS graph G and SPARQL pattern P , we would like to find an nSPARQL pattern Q such that:

$$\llbracket P \rrbracket_G^{\text{rdfs}} = \llbracket Q \rrbracket_G$$

But we trivially fail because of triple $(?X, ?Y, ?Z)$.

- ▶ We need to use a fragment of SPARQL

A good fragment of SPARQL for our study

\mathcal{T} : Set of triples (x, y, z) where $x \in U$ or $y \in U$ or $z \in U$.

A good fragment of SPARQL for our study

- \mathcal{T} : Set of triples (x, y, z) where $x \in U$ or $y \in U$ or $z \in U$.
- ▶ $(?X, a, b)$, $(?X, a, ?Y)$ and $(?X, ?Y, a)$

A good fragment of SPARQL for our study

\mathcal{T} : Set of triples (x, y, z) where $x \in U$ or $y \in U$ or $z \in U$.

- ▶ $(?X, a, b)$, $(?X, a, ?Y)$ and $(?X, ?Y, a)$

\mathcal{T} -SPARQL: Fragment of SPARQL where triple patterns are taken from \mathcal{T} .

Theorem (Pérez, A. and Gutierrez 2008)

For every \mathcal{T} -SPARQL pattern P , there exists an nSPARQL pattern Q such that $\llbracket P \rrbracket_G^{\text{rdfs}} = \llbracket Q \rrbracket_G$ for every RDF graph G .

nSPARQL captures \mathcal{T} -SPARQL over RDFS

Theorem (Pérez, A. and Gutierrez 2008)

For every \mathcal{T} -SPARQL pattern P , there exists an nSPARQL pattern Q such that $\llbracket P \rrbracket_G^{\text{rdfs}} = \llbracket Q \rrbracket_G$ for every RDF graph G .

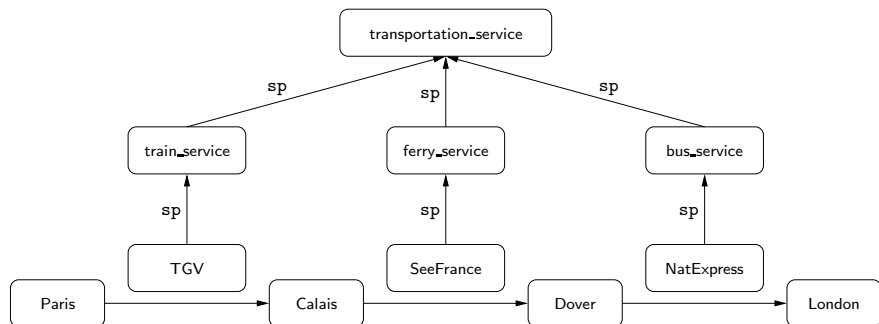
Proof sketch

Replace $(?X, a, ?Y)$ by $(?X, \text{trans}(a), ?Y)$, where:

$$\begin{aligned} \text{trans}(\text{dom}) &= \text{next::dom} \\ \text{trans}(\text{range}) &= \text{next::range} \\ \text{trans}(\text{sc}) &= (\text{next::sc})^+ \\ \text{trans}(\text{sp}) &= (\text{next::sp})^+ \end{aligned}$$

$$\begin{aligned} \text{trans}(\text{type}) = & \\ & \text{next::type}/(\text{next::sc})^* \mid \\ & \text{edge}/(\text{next::sp})^*/\text{next::dom}/(\text{next::sc})^* \mid \\ & \text{node}^{-1}/(\text{next::sp})^*/\text{next::range}/(\text{next::sc})^* \end{aligned}$$
$$\begin{aligned} \text{trans}(p) = & \text{next::}[(\text{next::sp})^*/\text{self::}p] \\ & \text{for } p \notin \{\text{sc}, \text{sp}, \text{range}, \text{dom}, \text{type}\} \end{aligned}$$

The extra expressive power of nSPARQL



$(?X, (next::[(next::sp)^*/(self::transportation_service)]^+, ?Y)$ cannot be expressed in SPARQL over RDFS

Thank you!