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

Marcelo Arenas\textsuperscript{1} and Jorge Pérez\textsuperscript{2}

\textsuperscript{1} Pontificia Universidad Católica de Chile
\textsuperscript{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.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/>
```

![RDF graph diagram]

- **conf:pods**: "Optimal Aggregation ..."
- **inPods:2001**
- **inPods:FaginLN01**
  - dc:title
  - dc:creator: :Amnon_Lotem
  - dc:creator: :Moni_Naor
  - dc:creator: :Ronald_Fagin
An example of a URI

http://dblp.l3s.de/d2r/resource/conferences/pods
URI can be used for any abstract resource

http://dblp.l3s.de/d2r/page/authors/Ronald_Fagin
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
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
{
}
```
SELECT ?Author
WHERE
{
}
SELECT ?Author
WHERE
{
}
SELECT ?Author
WHERE
{
}
SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
}
```
SPARQL in a nutshell

```
SELECT ?Author
WHERE
{
}
```
SPARQL in a nutshell

A SPARQL query consists of a:

```sparql
SELECT ?Author
WHERE
{
}
```
A SPARQL query consists of a:

**Body:** Pattern matching expression

```sparql
SELECT ?Author
WHERE {
}
```
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
Dereferenceable URIs are the glue

http://dbpedia.org/resource/Ronald_Fagin
Retrieve the authors that have published in PODS and were born in Oklahoma:

```
SELECT ?Author
WHERE
{
  ?Person owl:sameAs ?Author .
}
```
Retrieve the authors that have published in PODS, and their Web pages if this information is available:

```
SELECT ?Author ?WebPage
WHERE
{
  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 .
}
```
Taking into account vocabularies with predefined semantics

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

![Diagram showing the query with dbpedia:Ronald_Fagin as subject, dbo:birthPlace as predicate, and dbpedia:Oklahoma as object. Additionally, yago:DatabaseResearchers rdf:type rdfs:subClassOf dbo:birthPlace, yago:ResearchWorker rdf:type rdfs:subClassOf yago:Scientist.](image-url)
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

\[(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)\] is called an RDF triple

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

A set of RDF triples is called an RDF graph

\[(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)\] is called an RDF triple

\[U : \text{set of URIs}\]
\[B : \text{set of blank nodes}\]
\[L : \text{set of literals}\]
Proviso

In this talk, we do not consider blank nodes

\[ (s, p, o) \in U \times U \times (U \cup L) \] 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
SPARQL queries can be complex

Interesting features:

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

```
{ { P1
  P2 }

{ P3
  P4 }

}
```
Interesting features:

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

{ { P1
  P2
  OPTIONAL { P5 } }

{ P3
  P4
  OPTIONAL { P7 } }
}

M. Arenas and J. Pérez – Querying Semantic Web Data with SPARQL
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

```sparql
{ { 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

```sparql
{ { 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 $V$

```plaintext
?X :name "john"
```

- **Graph patterns:** full parenthesized algebra

```plaintext
{ P1 P2 }
{ P1 OPTIONAL { P2 } }
{ P1 } UNION { P2 }
{ P1 FILTER ( R ) }
```

- **Original SPARQL syntax vs. algebraic syntax**
A standard algebraic syntax

- **Explicit** precedence/association

Example

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

(((( t1 AND t2 ) OPT t3 ) OPT t4 ) AND t5 )
```
Definition

A mapping is a partial function:

\[ \mu : V \rightarrow (U \cup L) \]

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:

\[ \mu : V \rightarrow (U \cup L) \]

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 $\mu$ such that:

- $\mu$ has as domain the variables in $t$: $\text{dom}(\mu) = \text{var}(t)$
- $\mu$ 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 $\mu$ such that:

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

**Example**

<table>
<thead>
<tr>
<th>graph</th>
<th>triple</th>
<th>evaluation</th>
</tr>
</thead>
</table>
| $(R_1, \text{name, john})$ | $(?X, \text{name, ?Y})$ | $\mu_1$: $\begin{array}{|c|c|}
\hline
?X & R_1 \\
\hline
\end{array}$ $\begin{array}{|c|}
\hline
\text{john} \\
\hline
\end{array}$ |
| $(R_1, \text{email, J@ed.ex})$ | $(?X, \text{name, ?Y})$ | $\mu_2$: $\begin{array}{|c|c|}
\hline
R_2 & ?Y \\
\hline
\end{array}$ $\begin{array}{|c|}
\hline
\text{paul} \\
\hline
\end{array}$ |
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 $\mu$ such that:

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

**Example**

<table>
<thead>
<tr>
<th>Graph</th>
<th>Triple</th>
<th>Evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R_1, \text{name, john})$</td>
<td>$(?X, \text{name, ?Y})$</td>
<td>$\mu_1$: $\begin{array}{</td>
</tr>
<tr>
<td>$(R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>})$</td>
<td>$\mu_2$: $\begin{array}{</td>
<td>c</td>
</tr>
</tbody>
</table>
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 $\mu$ such that:

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

**Example**

<table>
<thead>
<tr>
<th>graph</th>
<th>triple</th>
<th>evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R_1, \text{name}, \text{john})$</td>
<td>$(?X, \text{name}, ?Y)$</td>
<td>$\mu_1$: $R_1$ john</td>
</tr>
<tr>
<td>$(R_1, \text{email}, <a href="mailto:J@ed.ex">J@ed.ex</a>)$</td>
<td></td>
<td>$\mu_2$: $R_2$ paul</td>
</tr>
<tr>
<td>$(R_2, \text{name}, \text{paul})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings

**Definition**

Mappings $\mu_1$ and $\mu_2$ 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**

<table>
<thead>
<tr>
<th>$\mu_1$ :</th>
<th>$\mu_2$ :</th>
<th>$\mu_3$ :</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>john</td>
<td>$\text{<a href="mailto:J@edu.ex">J@edu.ex</a>}$</td>
</tr>
<tr>
<td>$\text{<a href="mailto:P@edu.ex">P@edu.ex</a>}$</td>
<td>$R_2$</td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings

**Definition**

Mappings $\mu_1$ and $\mu_2$ 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**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ :</td>
<td>$R_1$</td>
<td>john</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td>$R_1$</td>
<td></td>
<td>$<a href="mailto:J@edu.ex">J@edu.ex</a>$</td>
<td></td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td>$<a href="mailto:P@edu.ex">P@edu.ex</a>$</td>
<td>$R_2$</td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings

Definition

Mappings $\mu_1$ and $\mu_2$ 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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$</td>
<td>$R_1$</td>
<td>john</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_3$</td>
<td>$R_1$</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_2$</td>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td>$R_2$</td>
</tr>
</tbody>
</table>
Compatible mappings

**Definition**

Mappings $\mu_1$ and $\mu_2$ 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**

$$
\begin{array}{c|c|c|c}
\hline
\mu_1: & R_1 & \text{john} & & \\
\mu_2: & R_1 & & J@edu.ex & \\
\mu_3: & & & P@edu.ex & \\
\mu_1 \cup \mu_2: & R_1 & \text{john} & J@edu.ex & R_2 \\
\end{array}
$$
Compatible mappings

**Definition**

Mappings $\mu_1$ and $\mu_2$ 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**

<table>
<thead>
<tr>
<th>$\mu_1$</th>
<th>$\mu_2$</th>
<th>$\mu_3$</th>
<th>$\mu_1 \cup \mu_2$</th>
<th>$\mu_1 \cup \mu_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>$R_1$</td>
<td>$R_1$</td>
<td>$R_1$</td>
<td>$R_1$</td>
</tr>
<tr>
<td>john</td>
<td></td>
<td></td>
<td>john</td>
<td>john</td>
</tr>
<tr>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>R_2</td>
<td>R_2</td>
<td>R_2</td>
</tr>
</tbody>
</table>
Compatible mappings

**Definition**

Mappings $\mu_1$ and $\mu_2$ 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**

$\mu_1 : R_1 \mapsto \text{john}$

$\mu_2 : R_1 \mapsto \text{J@edu.ex}$

$\mu_3 : R_1 \mapsto \text{P@edu.ex}$

$\mu_1 \cup \mu_2 : R_1 \mapsto \text{john}$

$\mu_1 \cup \mu_3 : R_1 \mapsto \text{john}$

$\Rightarrow \mu_2$ and $\mu_3$ are not compatible
Sets of mappings and operations

Let \( \Omega_1 \) and \( \Omega_2 \) be sets of mappings.

**Definition**

**Join:** extends mappings in \( \Omega_1 \) with compatible mappings in \( \Omega_2 \)

\[
\Omega_1 \Join \Omega_2 = \{ \mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1, \mu_2 \in \Omega_2 \text{ and } \mu_1, \mu_2 \text{ are compatible} \}
\]
Sets of mappings and operations

Let $\Omega_1$ and $\Omega_2$ be sets of mappings.

**Definition**

**Join:** extends mappings in $\Omega_1$ with compatible mappings in $\Omega_2$

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

**Difference:** selects mappings in $\Omega_1$ that cannot be extended with mappings in $\Omega_2$

- $\Omega_1 \setminus \Omega_2 = \{ \mu_1 \in \Omega_1 \mid \text{there is no mapping in } \Omega_2 \text{ compatible with } \mu_1 \}$
Sets of mappings and operations

Definition

**Union**: includes mappings in $\Omega_1$ and in $\Omega_2$

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

**Left Outer Join**: extends mappings in $\Omega_1$ with compatible mappings in $\Omega_2$ if possible

$\Omega_1 \bowtie \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$
Given an RDF graph $G$.

**Definition**

$[t]_G = \emptyset$

$[P_1 \text{ AND } P_2]_G = [P_1]_G \cap [P_2]_G$

$[P_1 \text{ UNION } P_2]_G = [P_1]_G \cup [P_2]_G$

$[P_1 \text{ OPT } P_2]_G = [P_1]_G \cup [P_2]_G$
Semantics of SPARQL

Given an RDF graph $G$.

**Definition**

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$[t]_G$</td>
<td>${ \mu \mid \text{dom}(\mu) = \text{var}(t) \text{ and } \mu(t) \in G }$</td>
</tr>
<tr>
<td>$[P_1 \text{ AND } P_2]_G$</td>
<td>$[P_1]_G \bowtie [P_2]_G$</td>
</tr>
<tr>
<td>$[P_1 \text{ UNION } P_2]_G$</td>
<td>$[P_1]_G \cup [P_2]_G$</td>
</tr>
<tr>
<td>$[P_1 \text{ OPT } P_2]_G$</td>
<td>$[P_1]_G \bowtie [P_2]_G$</td>
</tr>
</tbody>
</table>
Example

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

\[((?X, \text{name, ?Y}) \text{OPT} (?X, \text{email, ?E})\)]
Semantics of SPARQL: An example

Example

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

\(( (\text{?X, name, ?Y}) \text{ OPT } (\text{?X, email, ?E}) )\)
Semantics of SPARQL: An example

Example

(R₁, name, john)
(R₁, email, J@ed.ex)
(R₂, name, paul)

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>john</td>
</tr>
<tr>
<td>R₂</td>
<td>paul</td>
</tr>
</tbody>
</table>
Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>
Semantics of SPARQL: An example

Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>john</td>
</tr>
<tr>
<td>$R_2$</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>
Semantics of SPARQL: An example

Example

(R₁, name, john)
(R₁, email, J@ed.ex)
(R₂, name, paul)

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>john</td>
</tr>
<tr>
<td>R₂</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>
Example

(R₁, name, john)
(R₁, email, J@ed.ex)
(R₂, name, paul)

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>john</td>
</tr>
<tr>
<td>R₂</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>john</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
<tr>
<td>R₂</td>
<td>paul</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>
Semantics of SPARQL: An example

Example

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

from the Join
Semantics of SPARQL: An example

Example

(R₁, name, john)
(R₁, email, J@ed.ex)
(R₂, name, paul)

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>john</td>
</tr>
<tr>
<td>R₂</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>john</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
<tr>
<td>R₂</td>
<td>paul</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>

▶ from the Difference
Semantics of SPARQL: An example

Example

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

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

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?Y</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>john</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>?X</th>
<th>?E</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
</tr>
</tbody>
</table>

▶ 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 (\( \land \), \( \lor \), \( \neg \))

We impose a safety condition: \( \text{var}(R) \subseteq \text{var}(P) \)
A mapping $\mu$ satisfies a condition $R (\mu \models R)$ if:
Satisfaction of value constraints

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

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

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

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

\[ [P \ \text{FILTER} \ R]_G = \{ \mu \in [P]_G \mid \mu \models R \} \]
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 \( \mu \), a graph pattern \( P \), and an RDF graph \( G \)

**Question:**
Does \( \mu \) belong to the evaluation of \( P \) over \( G \)?

\[ \mu \in [P]_G \]
The evaluation problem

Input:
A mapping $\mu$, a graph pattern $P$, and an RDF graph $G$

Question:
Does $\mu$ belong to the evaluation of $P$ over $G$?

Does $\mu \in [P]_G$?

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

- $\mu$, $P$ 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
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)
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 {
}
```

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 $G$:

$$\text{CERTAIN ANSWERS}(P, G) = \bigcap_{H : G \subseteq H} [P]_H$$
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 $\llbracket P \rrbracket_G = \text{CertainAnswers}(P, G)$?
A graph pattern $P$ is monotone if:

For every pair $G_1, G_2$ of RDF graphs: $G_1 \subseteq G_2 \Rightarrow [P]_{G_1} \subseteq [P]_{G_2}$

**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 $G$:*

$$[P]_G = \text{CERTAIN ANSWERS}(P, G)$$
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 $\mu_1, \mu_2$: $\mu_1$ is subsumed by $\mu_2$ ($\mu_1 \preceq \mu_2$) 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 $\Omega_1, \Omega_2$ of mappings: $\Omega_1$ is subsumed by $\Omega_2$ ($\Omega_1 \sqsubseteq \Omega_2$) 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: $G_1 \subseteq G_2 \Rightarrow [P]_{G_1} \sqsubseteq [P]_{G_2}$
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:

\[
[P_1 \ MINUS \ P_2]_G = [P_1]_G \setminus [P_2]_G
\]
A MINUS operator in SPARQL

Let MINUS be defined as:

\[[P_1 \text{ MINUS } P_2]_G = [P_1]_G \setminus [P_2]_G\]

**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:

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

↑

?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:

\[(a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi))\]
Graph patterns in the proofs of PSPACE-hardness in [Pérez, A. and Gutierrez 2006, 2009] and [Schmidt, Meier and Lausen 2010] are not natural:

\[(a, \text{true}, \exists B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi))\]
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:

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

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:

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

\[?B_0 \quad \times \quad ?B_0\]

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:

\[(\cdots (P \text{ OPT } Q) \cdots)\]

if a variable occurs inside \(Q\) and anywhere outside the OPT operator, then the variable must also occur inside \(P\).
Definition

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

\[
( \cdots \cdots \cdots ( P \text{ OPT } Q ) \cdots \cdots \cdots )
\]

↑

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:

\[
( \cdots \quad ( \text{OPT} \quad Q ) \quad \cdots )
\]

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:

\[
( \cdots \cdots \cdots ( P \text{ \ OPT \ Q } ) \cdots \cdots \cdots )
\]

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:

\[
(... \ldots \ldots \ldots \ldots \ldots (P \text{ OPT } Q) \ldots \ldots \ldots) \\
\uparrow \quad \uparrow \quad \uparrow \quad \uparrow
\]

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

**Example**

\[
\left( (\textbf{?Y, name, paul}) \text{ OPT } (\textbf{?X, email, ?Z}) \right) \text{ AND } (\textbf{?X, name, john})
\]
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:

\[
\left( \ldots \ldots \ldots \left( P \text{ OPT } Q \right) \ldots \ldots \ldots \right)
\]

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

**Example**

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

\[
\left( \begin{array}{c}
\vdots \\
P & \text{OPT} & Q \\
\vdots
\end{array} \right)
\]

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:

\[(\ldots \quad (P \text{ OPT } Q) \quad \ldots \ldots )\]

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

Example
\[
\left( (\text{?Y, name, paul}) \text{ OPT } (\text{?X, email, ?Z}) \right) \text{ AND } (\text{?X, name, john})
\]
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 Vansummereren, 2011]

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

Theorem (Pérez, A. and Gutierrez 2009)

The evaluation problem is coNP-complete for well-designed graph patterns.
Reducing the complexity

**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 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 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 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
Consider the following rules:

\[
(P_1 \text{ OPT } P_2) \text{ FILTER } R \rightarrow (P_1 \text{ FILTER } R) \text{ OPT } P_2 \quad (1)
\]

\[
(P_1 \text{ AND } (P_2 \text{ OPT } P_3)) \rightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3) \quad (2)
\]

\[
((P_1 \text{ OPT } P_2) \text{ AND } P_3) \rightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2) \quad (3)
\]

**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 $\llbracket P \rrbracket_{G_1} \subseteq \llbracket P \rrbracket_{G_2}$
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?
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).
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:

$$(?X, \text{transportation\_service}, \text{ferry\_service}, \text{train\_service}, \text{TGV}, \text{SeeFrance}, \text{NatExpress}, \text{bus\_service}, \text{Dover}, \text{Calais}, \text{Paris}, \text{London}, \text{?Y})$$
A more complex query

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

nSPARQL:

$$(?X, \text{next}::) \, ^+ \, (?Y)$$
A more complex query

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

nSPARQL:

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

$(?X, \text{next}::[(\text{next}::\text{sp})^*/(\text{self}::\text{transportation\_service})])^+, ?Y)$
A more complex query

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

nSPARQL:

$$(?X, (\text{next}::[\text{next}::\text{sp}]^* / (\text{self}::\text{transportation\_service})))^+, ?Y)$$
A more complex query

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

nSPARQL:

$$(\text{?}X, (\text{next::}[(\text{next::sp})^*/(\text{self::transportation\_service})])^+, \text{?}Y)$$
A more complex query

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

nSPARQL:

$$(\exists X, (\text{next} :: [(\text{next} :: \text{sp})^* / (\text{self} :: \text{transportation} \_ \text{service})]^+) \quad , \quad ?Y)$$
A more complex query

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

nSPARQL:

\[
(?X, (\text{next}::[(\text{next}::\text{sp})^+/(\text{self}::\text{transportation\_service})])^+, ?Y)
\]
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: 

\[ \frac{(A, dom, B) \ (X, A, Y)}{(X, type, B)} \]

\[ \frac{(A, range, B) \ (X, A, Y)}{(Y, type, B)} \]
Entailment in RDFS

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
Entailment in RDFS: Closure of a graph

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

$$\llbracket P \rrbracket^{\text{rdfs}}_G = \llbracket P \rrbracket_{\text{cl}(G)}$$
Example: (dbpedia:Ronald_Fagin, type, yago:Scientist) over the closure
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
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:
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 $G$. 
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
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 $(a, \text{sc}, b)$, 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 $\text{sc}$
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.
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 \(G\) (\(\text{cl}(G)\) 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 \(G\) \((\text{cl}(G)\) 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 \((a, p, b)\):

\[
\begin{align*}
a & \xrightarrow{\text{edge}} p \\
p & \xrightarrow{\text{node}} b
\end{align*}
\]

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

\[
\begin{align*}
a & \xleftarrow{\text{edge}^{-1}} p \\
p & \xleftarrow{\text{node}^{-1}} b
\end{align*}
\]
The basic component: Nested regular expressions

Syntax of nested regular expressions:

\[ exp := \text{axis} \mid \text{axis::a} \mid \text{axis::[exp]} \mid \text{exp/exp} \mid \text{exp|exp} \mid \text{exp}^* \]

where \( a \in U \) and \( \text{axis} \in \{\text{self, next, next}^{-1}, \text{edge, edge}^{-1}, \text{node, node}^{-1}\} \).
Semantics of nested regular expressions

Given an RDFS graph $G$: 
Semantics of nested regular expressions

Given an RDFS graph $G$:

\[
\begin{align*}
\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \text{ is in } G\} \\
\llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \exists z \in U \ (x, z, y) \in G\} \\
\llbracket \text{edge} \rrbracket_G &= \{(x, y) \mid \exists z \in U \ (x, y, z) \in G\}
\end{align*}
\]
Semantics of nested regular expressions

Given an RDFS graph $G$:

\[
\begin{align*}
\text{[self]}_G & = \{(x,x) \mid x \text{ is in } G\} \\
\text{[next]}_G & = \{(x,y) \mid \exists z \in U \ (x,z,y) \in G\} \\
\text{[edge]}_G & = \{(x,y) \mid \exists z \in U \ (x,y,z) \in G\} \\
\text{[self::a]}_G & = \{(a,a)\} \\
\text{[next::a]}_G & = \{(x,y) \mid (x,a,y) \in G\} \\
\text{[edge::a]}_G & = \{(x,y) \mid (x,y,a) \in G\}
\end{align*}
\]
Semantics of nested regular expressions

Given an RDFS graph $G$:

$$[\text{self}]_G = \{(x, x) \mid x \text{ is in } G\}$$

$$[\text{next}]_G = \{(x, y) \mid \exists z \in U \ (x, z, y) \in G\}$$

$$[\text{edge}]_G = \{(x, y) \mid \exists z \in U \ (x, y, z) \in G\}$$

$$[\text{self}::a]_G = \{(a, a)\}$$

$$[\text{next}::a]_G = \{(x, y) \mid (x, a, y) \in G\}$$

$$[\text{edge}::a]_G = \{(x, y) \mid (x, y, a) \in G\}$$

$$[\text{exp}_1/\text{exp}_2]_G = \{(x, y) \mid \exists z \ (x, z) \in [\text{exp}_1]_G \text{ and } (z, y) \in [\text{exp}_2]_G\}$$

$$[\text{exp}_1|\text{exp}_2]_G = [\text{exp}_1]_G \cup [\text{exp}_2]_G$$

$$[\text{exp}^*]_G = [\text{self}]_G \cup [\text{exp}]_G \cup [\text{exp}/\text{exp}]_G \cup [\text{exp}/\text{exp}/\text{exp}]_G \cup \cdots$$
Given an RDFS graph $G$: 
Given an RDFS graph $G$:

$$
\text{next}::[\text{exp}]_G = \{(x, y) | \exists z, w \in U \ (x, z, y) \in G \text{ and } (z, w) \in [\text{exp}]_G\}
$$
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 \((x, \text{exp}, y)\)
  - \(\text{exp}\) 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 \((x, \text{exp}, y)\)
  - \(\text{exp}\) 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!
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 (\(a, \?X, \?Y\))
- (\(\?X, (\text{next::(sc)})^+, \?Y\)): Verifies whether \(\?X\) is a subclass of \(\?Y\)
Evaluation of $t = (?X, \text{exp}, ?Y)$ over an RDF graph $G$ is the set of mappings $\mu$ such that:
Evaluation of \( t = (?X, \text{exp}, ?Y) \) over an RDF graph \( G \) is the set of mappings \( \mu \) such that:

- The domain of \( \mu \) is \( \{?X, ?Y\} \)
Semantics of nSPARQL

Evaluation of \( t = (?X, \text{exp}, ?Y) \) over an RDF graph \( G \) is the set of mappings \( \mu \) such that:

- The domain of \( \mu \) is \{?X, ?Y\}
- \((\mu(?X), \mu(?Y)) \in \llbracket \text{exp} \rrbracket_G\)
Evaluation of $t = (\?X, \text{exp}, \?Y)$ over an RDF graph $G$ is the set of mappings $\mu$ such that:

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

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:

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

$$[P]_{G}^{\text{rdfs}} = [Q]_{G}$$

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:

$$[P]_{G}^{\text{rdfs}} = [Q]_{G}$$

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

\[ T: \] 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 \(z \in U\).

\[
\begin{align*}
&\triangleright (\exists X, a, b), (\exists X, a, \exists Y) \text{ and } (\exists X, \exists Y, a)
\end{align*}
\]

\( \mathcal{T}\text{-SPARQL} \): Fragment of SPARQL where triple patterns are taken from \( \mathcal{T} \).
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 \( \llbracket P \rrbracket^\text{rdfs}_G = \llbracket Q \rrbracket_G \) 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 $[P]_G^{\text{rdfs}} = [Q]_G$ for every RDF graph $G$.

**Proof sketch**

Replace $(?X, a, ?Y)$ by $(?X, trans(a), ?Y)$, where:

- $trans(\text{dom}) = \text{next::dom}$
- $trans(\text{range}) = \text{next::range}$
- $trans(\text{sc}) = (\text{next::sc})^+$
- $trans(\text{sp}) = (\text{next::sp})^+$
trans(type) =
\[
\text{next::type/}(\text{next::sc})^* \mid \\
\text{edge/}(\text{next::sp})^*/\text{next::dom/}(\text{next::sc})^* \mid \\
\text{node}^{-1}/(\text{next::sp})^*/\text{next::range/}(\text{next::sc})^*
\]

trans(p) = next::[(\text{next::sp})^*/\text{self::p}]
\quad \text{for } p \notin \{\text{sc, sp, range, dom, type}\}
The extra expressive power of nSPARQL

(\(?X, (\text{next}::[(\text{next}::\text{sp})^*/(\text{self}::\text{transportation}\_\text{service})])]^{+}, \text{?Y}) \) cannot be expressed in SPARQL over RDFS
Thank you!