Semantics is an indispensable aspect of a query language.
Semantics is an indispensable aspect of a query language

SELECT Name, Salary
FROM Employees
WHERE Salary >= 500000
Semantics is an indispensable aspect of a query language.

```sql
SELECT Name, Salary
FROM Employees
WHERE Salary >= 500000
```

\[
\pi_{\text{Name}, \text{Salary}}(\sigma_{\text{Salary} \geq 500000}(\text{Employees}))
\]
RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)
**RDF** data model for the Semantic Web

**SPARQL** query language for RDF (W3C initiatives)

RDF Graph:
RDF data model for the Semantic Web

SPARQL query language for RDF (W3C initiatives)

RDF Graph:

SPARQL Query:

```sparql
SELECT ?N
WHERE
{
}
```
RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

RDF Graph:

SPARQL Query:

```sparql
SELECT ?N ?E
WHERE
{
}
```
RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

RDF Graph:

SPARQL Query:

```
SELECT ?N ?E
WHERE
{
}
```
But things can become more complex...

Interesting features of pattern matching on graphs

```
SELECT ?X1 ?X2 ... 
WHERE 
  { P1 . 
    P2 } 
```
But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping

```sql
SELECT ?X1 ?X2 ...
WHERE
{ { P1 .
    P2 } 

  { P3 .
    P4 } }
```
But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts

```sql
SELECT ?X1 ?X2 ...
WHERE
{ { P1 .
    P2
    OPTIONAL { P5 } }

{ P3 .
    P4
    OPTIONAL { P7 } }

}
```
But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting

```
SELECT ?X1 ?X2 ...
WHERE
{ { P1 .
    P2
    OPTIONAL { P5 } }

{ P3 .
    P4
    OPTIONAL { P7
        OPTIONAL { P8 } } }
}
```
But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns

```
SELECT ?X1 ?X2 ...
WHERE
{ { P1 .
    P2
    OPTIONAL { P5 } }

{ P3 .
    P4
    OPTIONAL { P7
              OPTIONAL { P8 } } }
}
UNION
{ P9 }
```
But things can become more complex...

Interesting features of pattern matching on graphs

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

```
SELECT ?X1 ?X2 ...
WHERE
{ { P1 .
   P2
   OPTIONAL { P5 } }

{ P3 .
   P4
   OPTIONAL { P7
   OPTIONAL { P8 } } }
}
UNION
{ P9
  FILTER ( R ) }
```
But things can become more complex...

Interesting features of pattern matching on graphs

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

```
SELECT ?X1 ?X2 ...
WHERE
{ { P1 .
    P2
    OPTIONAL { P5 } }

{ P3 .
    P4
    OPTIONAL { P7
        OPTIONAL { P8 } } }
}
UNION
{ P9
    FILTER ( R ) }
```
But things can become more complex...

Interesting features of pattern matching on graphs

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

```sparql
SELECT ?X1 ?X2 ...
WHERE
{ { P1 .
   P2
   OPTIONAL { P5 } }

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

What is the *meaning* of a general SPARQL query?
The SPARQL W3C specification had no formal semantics!

- Specification primarily based on use cases and examples
- Semantics given in *natural language*
The SPARQL W3C specification had no formal semantics!

- Specification primarily based on use cases and examples
- Semantics given in *natural language*
- 6 *Working Drafts* from Feb-2004 to Feb-2006
- *Candidate Recommendation* in Apr-2006 but still no formal semantics!
The SPARQL W3C specification had no formal semantics!

- Specification primarily based on use cases and examples
- Semantics given in *natural language*
- 6 *Working Drafts* from Feb-2004 to Feb-2006
- *Candidate Recommendation* in Apr-2006
  but still no formal semantics!

A formal approach is beneficial to:
- Clarify corner cases
- Help in the implementation process
- Provide sound database foundations
Semantics and Complexity of SPARQL

Jorge Pérez

PhD Student
Computer Science Department, PUC – Chile

Work presented at the International Semantic Web Conference 2006
W3C SPARQL specification is currently based on our work

W3C document:

- Oct-2006 back to working draft
- Finally, a recommendation in Jan-2008 incorporating our formalization
W3C SPARQL specification is currently based on our work.

W3C document:
- Oct-2006 back to working draft
- Finally, a recommendation in Jan-2008 incorporating our formalization

In our work:
- A formal compositional semantics (for graph patterns)
- Complexity bounds
- Normalization and optimization procedures
Outline

Motivation
  Our contributions

Syntax and semantics of SPARQL graph patterns
  Syntax
  Semantics

Complexity results

Well–designed graph patterns
  Complexity
  Normalization and optimization
Outline

Motivation
  Our contributions

**Syntax and semantics of SPARQL graph patterns**
  Syntax
  Semantics

Complexity results

Well–designed graph patterns
  Complexity
  Normalization and optimization
A standard algebraic syntax

- **Triple patterns**: just triples + variables ($V$)

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

  ```
  (?X, name, john)
  ```

- **Graph patterns**: full parenthesized algebra

  ```
  \{ P1 P2 \}
  ```

  ```
  (P1 AND P2)
  ```

  ```
  \{ P1 OPTIONAL \{ P2 \} \}
  ```

  ```
  (P1 OPT P2)
  ```

  ```
  \{ P1 \} UNION \{ P2 \}
  ```

  ```
  (P1 UNION P2)
  ```

  ```
  \{ P1 FILTER ( R ) \}
  ```

  ```
  (P1 FILTER R)
  ```

original SPARQL syntax    algebraic syntax
Explicit precedence/association

Example

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

(((t1 AND t2) OPT t3) OPT t4) AND t5)
```
Mappings: building block for the semantics

Definition

A mapping is a *partial function* from variables to RDF terms.

\[ \mu : \text{Variables} \rightarrow \text{RDF Terms} \]
Definition
A mapping is a *partial function* from variables to RDF terms.

\[ \mu : \text{Variables} \rightarrow \text{RDF Terms} \]

The *evaluation* of a 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 $\mu$ that:
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 $\mu$ that:

- make $t$ 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 $\mu$ that:

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

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

**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>(?$X$, name, $?Y)$</td>
<td>$\mu_1$: $R_1 \quad \text{john}$</td>
</tr>
<tr>
<td>$(R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>})$</td>
<td></td>
<td>$\mu_2$: $R_2 \quad \text{paul}$</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 $\mu$ that:

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

**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>$\mu_1$: $R_1$, 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>$(R_2, \text{name, paul})$</td>
<td></td>
<td></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 $\mu$ that:

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

**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>$(R_1, \text{email}, \text{<a href="mailto:J@ed.ex">J@ed.ex</a>})$</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: mappings that can be merged.

Definition
Mappings are *compatibles* if they agree in their common variables.

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><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>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings: mappings that can be merged.

**Definition**

Mappings are *compatibles* if they agree in their common variables.

**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>$\text{john}$</td>
<td>$\text{John}\text{@edu.ex}$</td>
</tr>
<tr>
<td>$\text{John}\text{@edu.ex}$</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>$R_2$</td>
</tr>
</tbody>
</table>
Compatible mappings: mappings that can be merged.

**Definition**

Mappings are *compatibles* if they agree in their common variables.

**Example**

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>V</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>R1</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td></td>
</tr>
</tbody>
</table>
Compatible mappings: mappings that can be merged.

**Definition**

Mappings are *compatibles* if they agree in their common variables.

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

**Definition**
Mappings are *compatibles* if they agree in their common variables.

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>(\mu_1) ((R_1))</td>
<td>?X</td>
<td>john</td>
<td>?Z</td>
<td>?V</td>
</tr>
<tr>
<td>(\mu_2) ((R_1))</td>
<td>?X</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
</tr>
<tr>
<td>(\mu_3) ((R_2))</td>
<td>R_1</td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>R_2</td>
<td></td>
</tr>
</tbody>
</table>

\[\mu_1 \cup \mu_2 : \]

\[\mu_1 \cup \mu_3 : \]

\[R_1 \cup \mu_2 : \]

\[R_1 \cup \mu_3 : \]

\[R_2 \]
Compatible mappings: mappings that can be merged.

**Definition**

Mappings are *compatibles* if they agree in their common variables.

**Example**

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>µ₁</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>µ₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>µ₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>µ₁ ∪ µ₂</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>µ₁ ∪ µ₃</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- µ₁ : \(R₁\) \(\mu₁\) john
- µ₂ : \(R₁\) \(\mu₂\) J@edu.ex
- µ₃ : \(P@edu.ex\) \(\mu₃\)
- \(\mu₁ \cup \mu₂\) : \(R₁\) john J@edu.ex
- \(\mu₁ \cup \mu₃\) : \(R₁\) john P@edu.ex

\(\mu₂\) and \(\mu₃\) are not compatible
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:

Definition

Join: extends mappings in $M_1$ with compatible mappings in $M_2$
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:

**Definition**

**Join**: extends mappings in $M_1$ with compatible mappings in $M_2$

$M_1 \Join M_2 = \{ \mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2, \text{ and } \mu_1, \mu_2 \text{ are compatible} \}$
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:

**Definition**

**Join**: extends mappings in $M_1$ with compatible mappings in $M_2$

\[ M_1 \Join M_2 = \{ \mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2, \text{ and } \mu_1, \mu_2 \text{ are compatible} \} \]

**Difference**: selects mappings in $M_1$ that cannot be extended with mappings in $M_2$
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:

**Definition**

**Join**: extends mappings in $M_1$ with compatible mappings in $M_2$

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

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

- $M_1 \setminus M_2 = \{ \mu_1 \in M_1 \mid \text{there is no mapping } \mu_2 \in M_2 \text{ compatible with } \mu_1 \}$
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:

**Definition**
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:

**Definition**

**Union**: includes mappings in $M_1$ plus mappings in $M_2$ (set union)

$$M_1 \cup M_2 = \{ \mu \mid \mu \in M_1 \text{ or } \mu \in M_2 \}$$
Sets of mappings and operations

Let $M_1$ and $M_2$ be sets of mappings:

**Definition**

**Union**: includes mappings in $M_1$ plus mappings in $M_2$ (set union)

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

**Left Outer Join**: considers mappings in $M_1$ extending them with compatible mappings in $M_2$ *whenever it is possible*

$M_1 \bowtie M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2)$
Semantics in terms of operations between evaluations

Let $M_1$ and $M_2$ be the evaluation of $P_1$ and $P_2$.

Definition
The evaluation of:

$$(P_1 \text{ AND } P_2) \rightarrow$$
$$(P_1 \text{ UNION } P_2) \rightarrow$$
$$(P_1 \text{ OPT } P_2) \rightarrow$$
Semantics in terms of operations between evaluations

Let \( M_1 \) and \( M_2 \) be the evaluation of \( P_1 \) and \( P_2 \).

**Definition**

The evaluation of:

\[
(P_1 \text{ AND } P_2) \rightarrow M_1 \bowtie M_2 \\
(P_1 \text{ UNION } P_2) \rightarrow \\
(P_1 \text{ OPT } P_2) \rightarrow
\]
Let $M_1$ and $M_2$ be the *evaluation* of $P_1$ and $P_2$.

**Definition**

The evaluation of:

<table>
<thead>
<tr>
<th>Expression</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(P_1 \text{ AND } P_2)$</td>
<td>$M_1 \Join M_2$</td>
</tr>
<tr>
<td>$(P_1 \text{ UNION } P_2)$</td>
<td>$M_1 \cup M_2$</td>
</tr>
<tr>
<td>$(P_1 \text{ OPT } P_2)$</td>
<td>$\rightarrow$</td>
</tr>
</tbody>
</table>
Semantics in terms of operations between evaluations

Let $M_1$ and $M_2$ be the *evaluation* of $P_1$ and $P_2$.

**Definition**

The evaluation of:

\[
\begin{align*}
(P_1 \text{ AND } P_2) & \rightarrow M_1 \Join M_2 \\
(P_1 \text{ UNION } P_2) & \rightarrow M_1 \cup M_2 \\
(P_1 \text{ OPT } P_2) & \rightarrow M_1 \Join M_2
\end{align*}
\]
Example

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

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

Example

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

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

Example

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

((?X, name, ?Y) OPT (?X, email, ?E))
Simple 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>
Simple example

Example

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

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

<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>
Simple example

Example

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

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

<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>
</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>

▸ from the Join
Simple example

Example

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

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

<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 Join
- from the Difference
Simple 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>\text{<a href="mailto:J@ed.ex">J@ed.ex</a>}</td>
</tr>
<tr>
<td>(R_2)</td>
<td>paul</td>
<td>\text{<a href="mailto:J@ed.ex">J@ed.ex</a>}</td>
</tr>
</tbody>
</table>

- from the Join
- from the Difference
- from the Union
Boolean filter expressions (value constraints)

In filter expressions we consider:

- equality (\(=\)) among variables and RDF terms
- unary predicate bound
- boolean combinations (\(\land, \lor, \neg\))
Satisfaction of value constraints

A mapping satisfies

- \(?X = c\) if it gives the value \(c\) to variable \(?X\)
- \(?X = ?Y\) if it gives the same value to \(?X\) and \(?Y\)
- \(\text{bound}(?X)\) if it is defined for \(?X\)
Satisfaction of value constraints

A mapping satisfies

1. $\forall X = c$ if it gives the value $c$ to variable $X$
2. $\forall X = Y$ if it gives the same value to $X$ and $Y$
3. $\text{bound}(X)$ if it is defined for $X$

Definition

The evaluation of $(P \ \text{FILTER} \ R)$:

- mappings in the evaluation of $P$ that satisfy $R$. 
It was not that difficult to define a formal semantics for SPARQL
It was not that difficult to define a formal semantics for SPARQL.

Key aspects:
- use *partial mappings* for individual solutions
- adapt classical operators to deal with partial mappings
Outline

Motivation
  Our contributions

Syntax and semantics of SPARQL graph patterns
  Syntax
  Semantics

Complexity results

Well–designed graph patterns
  Complexity
  Normalization and optimization
The evaluation decision problem

**INPUT:**
A mapping, a graph pattern, and an RDF graph.

**OUTPUT:**
Is the mapping in the evaluation of the pattern over the graph?
Evaluation of simple patterns is polynomial.

**Theorem**

For patterns using only **AND** and **FILTER** operators,

the evaluation problem is polynomial:

\[ O(\text{size of the pattern} \times \text{size of the graph}) \].
Evaluation of simple patterns is polynomial.

**Theorem**

*For patterns using only AND and FILTER operators, the evaluation problem is polynomial:*

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

**Proof idea**

- Check that the mapping makes every triple to match.
- Then check that the mapping satisfies the FILTERs.
Evaluation including UNION is NP-complete.

**Theorem**

*For patterns using AND, FILTER and UNION operators,*

*the evaluation problem is NP-complete.*
Evaluation including UNION is NP-complete.

**Theorem**

For patterns using AND, FILTER and UNION operators, the evaluation problem is NP-complete.

**Proof idea**

- Reduction from propositional SAT
- The pattern codifies a propositional formula.
Evaluation including UNION is NP-complete.

**Theorem**

For patterns using **AND**, **FILTER** and **UNION** operators, the evaluation problem is NP-complete.

**Proof idea**

- Reduction from propositional SAT
- The pattern codifies a propositional formula.
- Using \( \neg \) bound to codify negation.
A simple normal from

Theorem (UNION Normal Form)

*Every graph pattern is equivalent to one of the form*

\[ P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_n \]

*with \( P_i \) UNION–free.*
A simple normal from

Theorem (UNION Normal Form)

Every graph pattern is equivalent to one of the form

\[ P_1 \cup P_2 \cup \cdots \cup P_n \]

with \( P_i \) UNION-free.

Theorem

The evaluation problem for AND-FILTER-UNION patterns in UNION normal form, is polynomial.
Evaluation in general is PSPACE-complete.

**Theorem**

*For general patterns that include OPT operator,*

the evaluation problem is *PSPACE-complete.*
Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT operator,

the evaluation problem is PSPACE-complete.

- still PSPACE-complete for AND-FILTER-OPT patterns
Evaluation in general is PSPACE-complete.

**Theorem**

For general patterns that include OPT operator, the evaluation problem is PSPACE-complete.

- still PSPACE-complete for AND-FILTER-OPT patterns

**Proof idea**

- Reduction from propositional quantified SAT
- The pattern codifies a quantified formula

\[
\forall x_1 \exists x_2 \forall x_3 \cdots \varphi.
\]
Evaluation in general is PSPACE-complete.

**Theorem**

*For general patterns that include OPT operator,*

the evaluation problem is PSPACE-complete.

- still PSPACE-complete for AND-FILTER-OPT patterns

**Proof idea**

- Reduction from propositional quantified SAT
- The pattern codifies a quantified formula

\[ \forall x_1 \exists x_2 \forall x_3 \cdots \varphi. \]

- Using nested OPTs to codify quantifier alternations.
Outline

Motivation
  Our contributions

Syntax and semantics of SPARQL graph patterns
  Syntax
  Semantics

Complexity results

Well–designed graph patterns
  Complexity
  Normalization and optimization
Well–designed patterns

Definition
An AND-FILTER-OPT pattern is \textit{well–designed} iff for every OPT in the pattern

\[(\ldots \ldots \ldots \, (\ A \ \text{OPT} \ B \ ) \ \ldots \ldots \ldots)\]

if a variable occurs
Well–designed patterns

Definition
An AND-FILTER-OPT pattern is well–designed iff for every OPT in the pattern

\[(\text{\ldots \ldots . \ (A \ OPT \ B) \ \text{\ldots \ldots . \)}}\]

↑

if a variable occurs inside $B$
Well–designed patterns

Definition
An AND-FILTER-OPT pattern is well–designed iff for every OPT in the pattern

\[ ( \ldots \ldots \ldots \ ( A \ \text{OPT} \ B \ \ldots \ldots \ldots ) \]

↑     ↑     ↑

if a variable occurs inside B and anywhere outside the OPT,
Well–designed patterns

Definition
An AND-FILTER-OPT pattern is well–designed iff for every OPT in the pattern

\[
( \ldots \quad ( \quad A \quad \text{OPT} \quad B \quad ) \quad \ldots \ldots \ldots )
\]

↑ ↑ ↑ ↑

if a variable occurs inside \( B \) and anywhere outside the OPT, then the variable must also occur inside \( A \).
Well–designed patterns

Definition
An AND-FILTER-OPT pattern is well–designed iff for every OPT in the pattern

\[
( \cdots \cdots \ ( A \ OPT \ B ) \ \cdots \cdots )
\]

\[\uparrow \ \uparrow \ \uparrow \ \uparrow \]

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example
\[
[ [ (?Y, name, paul) OPT (?X, email, ?Z) ] \ \text{AND} \ (?X, name, john) ]
\]
Well–designed patterns

**Definition**
An AND-FILTER-OPT pattern is *well–designed* iff for every OPT in the pattern

\[
\text{\begin{array}{c}
\text{( ………………… ( A OPT B ) ………………… )}
\end{array}}
\]

\[
\begin{array}{c}
\uparrow \\
\uparrow \\
\uparrow \\
\uparrow
\end{array}
\]

if a variable occurs *inside B and anywhere outside the OPT*, then the variable *must also occur inside A*.

**Example**

\[
\text{\begin{array}{c}
\text{[ [ (?Y, name, paul) OPT (?X, email, ?Z) ] AND (?X, name, john) ]}
\end{array}}
\]
Well–designed patterns

Definition
An AND-FILTER-OPT pattern is well–designed iff for every OPT in the pattern
\[
( \cdots \cdots \cdots ( A \ \text{OPT} \ B ) \cdots \cdots )
\]
if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

Example
\[
[ [ (Y, \text{name}, \text{paul}) \ \text{OPT} \ (X, \text{email}, Z) ] \ \text{AND} \ (X, \text{name}, \text{john}) ]
\]
Well–designed patterns

Definition

An AND-FILTER-OPT pattern is well–designed iff for every OPT in the pattern

\[
( \cdots \cdots ( A \text{ OPT } B ) \cdots )
\]

if a variable occurs inside \(B\) and anywhere outside the OPT, then the variable must also occur inside \(A\).

Example

\[
[ [ (?Y, \text{name, paul}) \text{ OPT } (?X, \text{email, ?Z}) ] \text{ AND } (?X, \text{name, john}) ]
\]
Well–designed patterns

Definition
An AND-FILTER-OPT pattern is well–designed iff for every OPT in the pattern

\[
\left( \ldots \ldots \ldots \ldots \left( A \text{ OPT } B \right) \ldots \ldots \ldots \right) \uparrow \uparrow \uparrow \uparrow
\]

if a variable occurs inside B and anywhere outside the OPT, then the variable must also occur inside A.

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

- Well-designed patterns initially proposed to show equivalence with a *procedural semantics* (by W3C)
Evaluation of well-designed patterns is in coNP-complete

**Theorem**

*For AND-FILTER-OPT well–designed graph patterns*

*the evaluation problem is coNP-complete*
Evaluation of well-designed patterns is in coNP-complete

**Theorem**

For AND-FILTER-OPT well–designed graph patterns

the evaluation problem is coNP-complete

**Corollary**

For patterns of the form $P_1 \text{ UNION } P_2 \text{ UNION } \cdots \text{ UNION } P_k$ where every $P_i$ is a UNION-free well–designed pattern,

the evaluation problem is coNP-complete
Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.

- SPARQL operations are never null–rejecting
  - by definition of compatible mappings.
Classical optimization is not directly applicable.

- Classical optimization assumes **null–rejection**.
  - null–rejection: the join/outer–join condition must fail in the presence of null.

- SPARQL operations are **never null–rejecting**
  - by definition of compatible mappings.

- Can we use classical optimization in the context of SPARQL?
Classical optimization is not directly applicable.

- Classical optimization assumes null-rejection.
  - null-rejection: the join/outer-join condition must fail in the presence of null.

- SPARQL operations are never null-rejecting
  - by definition of compatible mappings.

- Can we use classical optimization in the context of SPARQL?
  - Well-designed patterns are suitable for reordering, and then for classical optimization.
Consider the following rules:

\[
\begin{align*}
((P_1 \text{ OPT } P_2) \text{ FILTER } R) & \rightarrow ((P_1 \text{ FILTER } R) \text{ OPT } P_2) & (1) \\
(P_1 \text{ AND } (P_2 \text{ OPT } P_3)) & \rightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3) & (2) \\
((P_1 \text{ OPT } P_2) \text{ AND } P_3) & \rightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2) & (3)
\end{align*}
\]
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**

If \( P \) is a well-designed pattern and \( Q \) is obtained from \( P \) by applying either (1) or (2) or (3), then \( Q \) is a well-designed pattern equivalent to \( P \).
Well–designed graph patterns and optimization

**Definition**

A graph pattern $P$ is in **OPT normal form** if there exist AND-FILTER patterns $Q_1, \ldots, Q_k$ such that:

$P$ is constructed from $Q_1, \ldots, Q_k$ by using only the OPT operator.
A graph pattern $P$ is in OPT normal form if there exist AND-FILTER patterns $Q_1, \ldots, Q_k$ such that:

$P$ is constructed from $Q_1, \ldots, Q_k$ by using only the OPT operator.

\textbf{Theorem}

\textit{Every well-designed pattern is equivalent to a pattern in OPT normal form.}
Summary

- A formal compositional semantics for SPARQL
- Complexity bounds
- Normalization and initial optimizations procedures
Summary

- A formal compositional semantics for SPARQL
- Complexity bounds
- Normalization and initial optimizations procedures

Impact:
- Official semantics for SPARQL by W3C based on our work
- Base of the theoretical studies around SPARQL
- Journal version published in *ACM TODS 2009*
Semantics and Complexity of SPARQL

Jorge Pérez

PhD Student
Computer Science Department, PUC – Chile

Work presented at the International Semantic Web Conference 2006