Semantic Web research inspired by W3C standards, or the hell of the practice without theory

Jorge Pérez

Assistant Professor
Department of Computer Science
Universidad de Chile

joint work with M. Arenas, S. Conca (PUC - Chile) and C. Gutierrez (U. Chile)
The Semantic Web vision: not only human but also machine readable Web

*The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.*

Tim Berners-Lee et al. 2001.

Specific goals:
- Build a description language with standard semantics
- Make semantics machine-processable and understandable
The Semantic Web vision: not only human but also machine readable Web

*The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation.*

Tim Berners-Lee et al. 2001.

Specific goals:
- Build a description language with standard semantics
- Make semantics machine-processable and understandable

W3C proposals:
- *Resource Description Framework (RDF)* as data model
- *SPARQL* as (graph pattern matching) query language
An example of an RDF graph: DBLP

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

```
swrc:series

conf:amw

inAMW:2009

dct:partOf

"Schema Design for ..."

dc:title

dc:creator :Anish_Das_Sarma

dc:creator :Jeffrey_D._Ullman

dc:creator :Jennifer_Widom
```

inAMW:SarmaUW09
URLs can be used for any abstract resource

http://dblp.l3s.de/d2r/resource/publications/conf/amw/2009

http://dblp.l3s.de/d2r/resource/authors/Leopoldo_E._Bertossi

http://dblp.l3s.de/d2r/resource/publications/conf/pods/MiklauS02
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-triples: ($URI_2$, :email, rgarrido@utalca.cl)
**RDF** data model for the Semantic Web

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

RDF Graph:

```
URI1
  :name Federico Meza
  :email fmeza@utalca.cl
  :friendOf URI2

URI2
  :phone 35-446928
  :name Ruth Garrido
  :email rgarrido@utalca.cl
```

RDF-triples: \( (URI_2, :\text{email}, \text{rgarrido@utalca.cl}) \)

SPARQL Query:

```
SELECT ?N
WHERE {
    ?X :\text{name} ?N .
}
```
**RDF** data model for the Semantic Web

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

RDF Graph:

```
URI1
  :name Federico Meza
  :email fmeza@utalca.cl
  :friendOf
    URI2
      :name Ruth Garrido
      :phone 35-446928
      :email rgarrido@utalca.cl
```

RDF-triples: \((URI_2, :email, rgarrido@utalca.cl)\)

SPARQL Query:

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

**RDF Graph:**

```
URI1
:email  fmeza@utalca.cl
:name   Federico Meza
:friendOf  URI2
:email  rgarrido@utalca.cl
:name   Ruth Garrido
:phone  35-446928
```

RDF-triples: \((URI_2, \text{:email}, \text{rgarrido@utalca.cl})\)

**SPARQL Query:**

```sql
SELECT ?N ?E
WHERE
{
}
```
SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW
Example: Authors that have published in AMW

SELECT ?Author
Example: Authors that have published in AMW

SELECT ?Author
WHERE
{

}

Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
}
```
Example: Authors that have published in AMW

SELECT ?Author
WHERE
{
}
SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
}
```
SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
}
```

A SPARQL query consists of a:
Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
}
```

A SPARQL query consists of a:

**Head:** Processing of the variables
SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
}
```

A SPARQL query consists of a:

- **Head**: Processing of the variables
- **Body**: Pattern matching expression
Example: Authors that have published in AMW, and their Web pages if this information is available:

```
SELECT ?Author ?WebPage
WHERE
{

  OPTIONAL {
    ?Author foaf:homePage ?WebPage . }
}
```
Example: Authors that have published in AMW, and their Web pages if this information is available:

```sparql
SELECT ?Author ?WebPage
WHERE
{

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

Interesting features of pattern matching on graphs

{ P1 .
P2 }
But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping

```
{ { P1 .
P2 } }

{ P3 .
P4 }
```

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

Interesting features of pattern matching on graphs

▶ Grouping
▶ Optional parts
But things can become more complex...

Interesting features of pattern matching on graphs:

- Grouping
- Optional parts
- Nesting

```plaintext
{ { 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

```plaintext
{ { 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
- ...

```
{ { 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
- ...

What is the *meaning* of a general SPARQL graph pattern?

```sparql
{ { P1 .
   P2
   OPTIONAL { P5 } }

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

SELECT Name, Salary
FROM Employees
WHERE Salary >= 500000

\[ \pi_{Name, Salary} (\sigma_{Salary \geq 500000} (Employees)) \]
The SPARQL W3C specification had no formal semantics

Until 2006:

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

Until 2006:

- 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 until 2006:

- 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
- Our group proposed a formalization of *graph patterns* in Jun-2006
The SPARQL W3C specification had no formal semantics

Until 2006:

- 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
- Our group proposed a formalization of *graph patterns* in
  Jun-2006

A formal approach is beneficial to:

- Clarify corner cases
- Help in the implementation process
- Provide sound database foundations
W3C acknowledged the need for a formal semantics

- Back to Working Draft in Oct-2006
- Final recommendation in Jan-2008

Include a formalization using our proposal as input
Outline

Syntax and Semantics of SPARQL

Complexity: Evaluation and Static Analysis

SPARQL 1.1 path queries
A standard algebraic syntax

- Triple patterns: just triples + variables

?X :name "john"  
(?X, name, john)
A standard algebraic syntax

- Triple patterns: just triples + variables

\[ ?X : \text{name} "john" \]

- Graph patterns: full parenthesized algebra

\[ \{ \text{P1} \ \text{P2} \} \]

\[ \{ \text{P1} \ \text{OPT} \ \{ \text{P2} \} \} \]

\[ \{ \text{P1} \ \text{UNION} \ \{ \text{P2} \} \} \]

\[ \{ \text{P1} \ \text{FILTER} \ (\text{R}) \} \]

original SPARQL syntax  algebraic syntax
A standard algebraic syntax (cont.)

- Explicit precedence/association

Example

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

((( t1 AND t2 ) OPT t3 ) OPT t4 ) AND t5 )
```
A mapping is a partial function from variables to RDF terms.

\[ \mu : \text{Variables} \rightarrow \text{RDF Terms} \]
Mappings: building block for the semantics

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

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

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 \textit{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>$(R_1, \text{name}, \text{john})$</td>
<td>(?X, name, ?Y)</td>
<td>$\mu_1$:</td>
</tr>
<tr>
<td>$(R_1, \text{email}, \text{<a href="mailto:J@ed.ex">J@ed.ex</a>})$</td>
<td></td>
<td>$R_1$ john</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 \mapsto \text{john})</td>
</tr>
<tr>
<td>((R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>}))</td>
<td>((?X, \text{name, } ?Y))</td>
<td>(\mu_2: R_2 \mapsto \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>$\mu_1: R_1 \mapsto \text{john}$</td>
</tr>
<tr>
<td>$(R_1, \text{email, <a href="mailto:J@ed.ex">J@ed.ex</a>})$</td>
<td></td>
<td>$\mu_2: R_2 \mapsto \text{paul}$</td>
</tr>
<tr>
<td>$(R_2, \text{name, paul})$</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></td>
<td>$R_1$</td>
<td><strong>john</strong></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td></td>
<td>$R_1$</td>
<td></td>
<td>$<a href="mailto:J@edu.ex">J@edu.ex</a>$</td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td></td>
<td>$<a href="mailto:P@edu.ex">P@edu.ex</a>$</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></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ :</td>
<td>$R_1$</td>
<td></td>
<td>john</td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td></td>
<td>$<a href="mailto:J@edu.ex">J@edu.ex</a>$</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<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></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></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></td>
</tr>
<tr>
<td></td>
<td>$R_2$</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></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mu_1$ : $R_1$</td>
<td>john</td>
<td><a href="mailto:J@edu.ex">J@edu.ex</a></td>
<td>$R_2$</td>
</tr>
<tr>
<td>$\mu_2$ : $R_1$</td>
<td></td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></td>
</tr>
<tr>
<td>$\mu_3$ :</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_2$ :</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>$\mu_1$ :</td>
<td>$R_1$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\mu_2$ :</td>
<td>$R_1$</td>
<td>john</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>$R_1$</td>
<td>john</td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>$R_2$</td>
</tr>
<tr>
<td>$\mu_1 \cup \mu_3$ :</td>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></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></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>john</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><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td></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>
<tr>
<td>$\mu_1 \cup \mu_3$</td>
<td>$R_1$</td>
<td>john</td>
<td><a href="mailto:P@edu.ex">P@edu.ex</a></td>
<td>$R_2$</td>
</tr>
</tbody>
</table>

$\mu_2$ and $\mu_3$ 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 $M_2$

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

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

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

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

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

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

$\triangleright M_1 \bowtie M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2)$
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 \textit{evaluation} of $P_1$ and $P_2$.

\textbf{Definition}

The evaluation of:

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

- $(P_1 \text{ AND } P_2) \rightarrow M_1 \bowtie M_2$
- $(P_1 \text{ UNION } P_2) \rightarrow M_1 \cup M_2$
- $(P_1 \text{ OPT } P_2) \rightarrow$
Semantics in terms of operations between evaluations

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

<table>
<thead>
<tr>
<th>Operation</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>$M_1 \Join M_2$</td>
</tr>
</tbody>
</table>
Simple example

Example


tuple 1

(R₁, name, john)
(R₁, email, J@ed.ex)

(R₂, name, paul)


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


Simple example

Example

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

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

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>
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>

<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

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

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

<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>
Simple example

Example

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

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

\begin{array}{|c|c|}
\hline
?X & ?Y \\
\hline
R_1 & \text{john} \\
R_2 & \text{paul} \\
\hline
\end{array}

\begin{array}{|c|c|c|}
\hline
?X & ?Y & ?E \\
\hline
R_1 & \text{john} & \text{J@ed.ex} \\
R_2 & \text{paul} & \\
\hline
\end{array}

\begin{array}{|c|c|}
\hline
?X & ?E \\
\hline
R_1 & \text{J@ed.ex} \\
\hline
\end{array}
Simple example

Example

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

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

\begin{tabular}{|c|c|}
  \hline
  ?X & ?Y \\
  \hline
  \text{R}_1 & \text{john} \\
  \text{R}_2 & \text{paul} \\
  \hline
\end{tabular}

\begin{tabular}{|c|c|c|}
  \hline
  ?X & ?Y & ?E \\
  \hline
  \text{R}_1 & \text{john} & \text{J@ed.ex} \\
  \text{R}_2 & \text{paul} & \text{J@ed.ex} \\
  \hline
\end{tabular}

\[\Rightarrow \text{ from the Join}\]
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>?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>

▶ from the Join
▶ from the Difference
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>
The formal semantics of SPARQL is simple
The formal semantics of SPARQL is simple

Key aspects:

- use *partial mappings* for individual solutions
- adapt classical operators to deal with sets of partial mappings

Official SPARQL specification by W3C based on this semantics
A formalization allows us to study theoretical problems

- Complexity of the evaluation problem
- Static analysis and optimization
- Logical expressiveness of the language
A formalization allows us to study theoretical problems

- Complexity of the evaluation problem
- Static analysis and optimization
- Logical expressiveness of the language
The evaluation decision problem

<table>
<thead>
<tr>
<th>Evaluation problem for SPARQL patterns</th>
</tr>
</thead>
</table>
| **Input:** A mapping $\mu$, an RDF graph $G$  
a graph pattern $P$ |
| **Output:** Is the mapping $\mu$ in the evaluation of pattern $P$  
over the RDF graph $G$ |
Brief complexity-theory reminder
Brief complexity-theory reminder
Brief complexity-theory reminder
Complexity of the evaluation problem

Theorem (PAG09)

For patterns using only the AND operator,

the evaluation problem is in $P$
## Complexity of the evaluation problem

<table>
<thead>
<tr>
<th>Theorem (PAG09)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>For patterns using only the AND operator,</td>
<td>the evaluation problem is in $\text{P}$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Theorem (SML10)</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>For patterns using AND and UNION operators,</td>
<td>the evaluation problem is $\text{NP}$-complete.</td>
</tr>
</tbody>
</table>
**Complexity of the evaluation problem**

| Theorem (PAG09) | For patterns using only the AND operator,  
| the evaluation problem is in P |
| Theorem (SML10) | For patterns using AND and UNION operators,  
| the evaluation problem is NP-complete. |
| Theorem (PAG09,SML10) | For general patterns that include OPT operator,  
| the evaluation problem is PSPACE-complete. |
Complexity of the evaluation problem

Theorem (PAG09)
For patterns using only the AND operator, 
the evaluation problem is in \( P \)

Theorem (SML10)
For patterns using AND and UNION operators, 
the evaluation problem is \( NP \)-complete.

Theorem (PAG09,SML10)
For general patterns that include OPT operator, 
the evaluation problem is \( PSPACE \)-complete.

Good news: evaluation in \( P \) if the query is fixed (data complexity)
Well–designed patterns

Can we find a natural fragment with better complexity?

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

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

if a variable occurs
Well–designed patterns

Can we find a natural fragment with better complexity?

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

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

if a variable occurs *inside* \( B \)
Can we find a natural fragment with better complexity?

**Definition**

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

\[
\begin{array}{ccc}
\cdots & (A & \text{OPT} & B) & \cdots
\end{array}
\]

if a variable occurs *inside B* and *anywhere outside the OPT*,

up up up
Well–designed patterns

Can we find a natural fragment with better complexity?

**Definition**

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

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

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

Can we find a natural fragment with better complexity?

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

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

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

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

Can we find a natural fragment with better complexity?

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

\[
\begin{array}{c}
\text{( \ldots \ldots...) ( } \ A \text{ OPT } B \text{ ) \ldots \ldots...) ( \ldots \ldots) \\
\uparrow \quad \uparrow \quad \uparrow \quad \uparrow
\end{array}
\]

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

Example
\[
\begin{array}{c}
\text{[ [ (?Y, name, paul) OPT (?X, email, ?Z) ] AND (?X, name, john) ]}
\end{array}
\]
Can we find a natural fragment with better complexity?

**Definition**

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

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

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

**Example**

\[
[ [ ( ?Y, \ name, \ paul ) \ OPT \ ( ?X, \ email, \ ?Z ) ] \ AND \ ( ?X, \ name, \ john ) ]
\]
Can we find a natural fragment with better complexity?

**Definition**

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

\[ \begin{array}{c}
( \cdots \cdots \cdots \, ( \text{A OPT } B \, ) \, \cdots \cdots \cdots ) \\
\uparrow \quad \uparrow \quad \uparrow \quad \uparrow
\end{array} \]

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

**Example**

\[ \begin{array}{c}
[ \begin{array}{c}
( ?Y, \text{name, paul} ) \, \text{OPT} \, ( ?X, \text{email, } \, ?Z ) \end{array} ] \quad \text{AND} \quad ( ?X, \text{name, john} ) \end{array} \]
A bit more on complexity...
A bit more on complexity...
Evaluation of well-designed patterns is in coNP-complete

Theorem (PAG09)

For AND-OPT well–designed graph patterns

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

Theorem (PAG09)
For AND-OPT well–designed graph patterns

the evaluation problem is coNP-complete

Well-designed patterns also allow to study static analysis:

Theorem (LPPS12)
Equivalence of well-designed SPARQL patterns is in NP
Outline

Syntax and Semantics of SPARQL

Complexity: Evaluation and Static Analysis

SPARQL 1.1 path queries
SPARQL 1.0 provides limited navigational capabilities
SPARQL 1.0 provides limited navigational capabilities

```
SELECT ?X
WHERE
{
    ?Y :name "Maria" .
}
```
SPARQL 1.0 provides limited navigational capabilities

```
SELECT ?X
WHERE
{
    ?Y :name "Maria" .
}
```
SPARQL 1.0 provides limited navigational capabilities

### SELECT ?X
WHERE
{
  ?X (:friendOf)?Y .
  ?Y :name "Maria" .
}
SPARQL 1.0 provides limited navigational capabilities

```
SELECT ?X
WHERE
{
  ?X (:friendOf)* ?Y .
  ?Y :name "Maria" .
}
```
SPARQL 1.0 provides limited navigational capabilities

```
SELECT ?X
WHERE
{
  ?X (:friendOf)* ?Y .
  ?Y :name "Maria" .
}
```

← SPARQL 1.1 property path
SPARQL 1.1 implementations had a poor performance

Data:
- **cliques** (complete graphs) of different size
- from 2 nodes (87 bytes) to 13 nodes (970 bytes)

RDF clique with 4 nodes (127 bytes)
SPARQL 1.1 implementations had a poor performance

\[
\text{SELECT * WHERE \{ :a0 (:p)* :a1 \}}
\]
Poor performance with real Web data of small size

Data:
- Social Network data given by foaf:knows links
- Crawled from Axel Polleres’ foaf document (3 steps)
- Different documents, deleting some nodes
Poor performance with real Web data of small size

```
SELECT * WHERE { axel:me (foaf:knows)* ?x }
```
Poor performance with real Web data of small size

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

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>5.13</td>
<td>75.70</td>
<td>313.37</td>
<td>–</td>
</tr>
<tr>
<td>10.9KB</td>
<td>8.20</td>
<td>325.83</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>11.4KB</td>
<td>65.87</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>13.2KB</td>
<td>292.43</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>14.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>17.2KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>20.5KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>25.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

(time in seconds, timeout = 1hr)
Poor performance with real Web data of small size

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

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>5.13</td>
<td>75.70</td>
<td>313.37</td>
<td>–</td>
</tr>
<tr>
<td>10.9KB</td>
<td>8.20</td>
<td>325.83</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>11.4KB</td>
<td>65.87</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>13.2KB</td>
<td>292.43</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>14.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>17.2KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>20.5KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
<tr>
<td>25.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

(time in seconds, timeout = 1hr)

Is this a problem of these particular implementations?
This is a problem of the specification

Arenas, Conca, P. (WWW 2012)

Any implementation that follows SPARQL 1.1 standard (as of January 2012) is doomed to show the same behavior
This is a problem of the specification

Arenas, Conca, P. (WWW 2012)

Any implementation that follows SPARQL 1.1 standard (as of January 2012) is doomed to show the same behavior

The main sources of complexity is counting
This is a problem of the specification

Arenas, Conca, P. (WWW 2012)

Any implementation that follows SPARQL 1.1 standard (as of January 2012) is doomed to show the same behavior

The main sources of complexity is *counting*

*Impact on W3C standard:*

- Standard semantics of SPARQL 1.1 property paths will be changed to overcome the issues raised in [ACP12]
SPARQL 1.1 property paths match regular expressions but also *count*

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

```
SELECT ?X
WHERE { :a (:p)* ?X }
```
SPARQL 1.1 property paths match regular expressions but also *count*

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

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

?X
: a
: b
: c
: d
: d
SPARQL 1.1 property paths match regular expressions but also *count*

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

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

```
?X
:a
:b
:c
:d
```
SPARQL 1.1 property paths match regular expressions but also *count*

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

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

```
?X
:a
:b
:c
:d
:d
```
SPARQL 1.1 property paths match regular expressions but also *count*

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

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

?X
---
:a
:b
:c
:d
:d
:c
:d
SPARQL 1.1 property paths match regular expressions but also *count*

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

But what if we have cycles?
Evaluation of \textit{path*}

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

SPARQL 1.1 Last Call (Jan 2012)
SPARQL 1.1 document provides a special procedure to handle cycles (and make the count)

Evaluation of \textit{path*}

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

SPARQL 1.1 Last Call (Jan 2012)

- W3C document provides a procedure (\texttt{ArbitraryLengthPath})
- This procedure was formalized in [ACP12]
Counting the number of solutions...

Data: Clique of size $n$

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

every solution is a copy of the empty mapping $\| \| \text{ in ARQ}$
Counting the number of solutions...

Data: Clique of size \( n \)

\[
\{ :a_0 (:p)* :a_1 \}
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
</tr>
<tr>
<td>13</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the empty mapping (\( | \) in ARQ)
Counting the number of solutions...

Data: Clique of size $n$

\[
\{ :a0 (:p)* :a1 \} \quad \{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
</tr>
<tr>
<td>13</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the *empty mapping* (⊥ ⊥ in ARQ)
Counting the number of solutions...

Data: Clique of size $n$

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
<th>$n$</th>
<th># Sol</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>13</td>
<td>–</td>
<td>6</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the *empty mapping* (\|\| in ARQ)
Counting the number of solutions...

Data: Clique of size $n$

\[
\{ :a0 (:p)* :a1 \} \quad \{ :a0 ((:p)*)* :a1 \} \quad \{ :a0 (((:p)*)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th>$# \text{Sol.}$</th>
<th>$n$</th>
<th>$# \text{Sol}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>13</td>
<td>–</td>
<td>6</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the empty mapping (| | in ARQ)
Counting the number of solutions...

Data: Clique of size \( n \)

\[
\{ :a0 (:p)* :a1 \} \quad \{ :a0 ((:p)*)* :a1 \} \quad \{ :a0 (((:p)*)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th># Sol.</th>
<th>( n )</th>
<th># Sol</th>
<th>( n )</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>13,700</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>109,601</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>42</td>
</tr>
<tr>
<td>11</td>
<td>986,410</td>
<td>4</td>
<td>305</td>
<td>4</td>
<td>–</td>
</tr>
<tr>
<td>12</td>
<td>9,864,101</td>
<td>5</td>
<td>418,576</td>
<td>6</td>
<td>–</td>
</tr>
</tbody>
</table>

every solution is a copy of the \textit{empty mapping} (\|\| in ARQ)
More on counting the number of solutions...

Data: foaf links crawled from the Web

```
{ axel:me (foaf:knows)* ?x }
```
More on counting the number of solutions...

Data: foaf links crawled from the Web

\[
\{ \text{axel:me (foaf:knows)}^* \ ?x \ \}
\]

<table>
<thead>
<tr>
<th>File</th>
<th># URIs</th>
<th># Sol.</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>38</td>
<td>29,817</td>
<td>2MB</td>
</tr>
<tr>
<td>10.9KB</td>
<td>43</td>
<td>122,631</td>
<td>8.4MB</td>
</tr>
<tr>
<td>11.4KB</td>
<td>47</td>
<td>1,739,331</td>
<td>120MB</td>
</tr>
<tr>
<td>13.2KB</td>
<td>52</td>
<td>8,511,943</td>
<td>587MB</td>
</tr>
<tr>
<td>14.8KB</td>
<td>54</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>
More on counting the number of solutions...

Data: foaf links crawled from the Web

\[
\{ \text{axel:me (foaf:knows)}* ?x \}
\]

<table>
<thead>
<tr>
<th>File</th>
<th># URIs</th>
<th># Sol.</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>38</td>
<td>29,817</td>
<td>2MB</td>
</tr>
<tr>
<td>10.9KB</td>
<td>43</td>
<td>122,631</td>
<td>8.4MB</td>
</tr>
<tr>
<td>11.4KB</td>
<td>47</td>
<td>1,739,331</td>
<td>120MB</td>
</tr>
<tr>
<td>13.2KB</td>
<td>52</td>
<td>8,511,943</td>
<td>587MB</td>
</tr>
<tr>
<td>14.8KB</td>
<td>54</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

What is really happening here?
More on counting the number of solutions...

Data: foaf links crawled from the Web

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

<table>
<thead>
<tr>
<th>File</th>
<th># URIs</th>
<th># Sol.</th>
<th>Output Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>38</td>
<td>29,817</td>
<td>2MB</td>
</tr>
<tr>
<td>10.9KB</td>
<td>43</td>
<td>122,631</td>
<td>8.4MB</td>
</tr>
<tr>
<td>11.4KB</td>
<td>47</td>
<td>1,739,331</td>
<td>120MB</td>
</tr>
<tr>
<td>13.2KB</td>
<td>52</td>
<td>8,511,943</td>
<td>587MB</td>
</tr>
<tr>
<td>14.8KB</td>
<td>54</td>
<td>–</td>
<td>–</td>
</tr>
</tbody>
</table>

What is really happening here?

Theory can help!
A bit more on complexity classes...

Complexity can be measured by using *counting-complexity classes*

<table>
<thead>
<tr>
<th><strong>NP</strong></th>
<th><strong>#P</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sat</strong>: is a propositional formula satisfiable?</td>
<td><strong>CountSat</strong>: how many assignments satisfy a propositional formula?</td>
</tr>
</tbody>
</table>
A bit more on complexity classes...

Complexity can be measured by using *counting-complexity classes*

<table>
<thead>
<tr>
<th>NP</th>
<th>#P</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sat</strong>: is a propositional formula satisfiable?</td>
<td><strong>CountSat</strong>: how many assignments satisfy a propositional formula?</td>
</tr>
</tbody>
</table>

Formally

A function $f(\cdot)$ on strings is in $\#P$ if there exists a polynomial-time non-deterministic TM $M$ such that

$$f(x) = \text{number of accepting computations of } M \text{ with input } x$$
A bit more on complexity classes...

Complexity can be measured by using *counting-complexity classes*

<table>
<thead>
<tr>
<th>Complexity Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NP</strong></td>
<td><em>Sat</em>: is a propositional formula satisfiable?</td>
</tr>
<tr>
<td><strong>#P</strong></td>
<td><strong>CountSat</strong>: how many assignments satisfy a propositional formula?</td>
</tr>
</tbody>
</table>

**Formally**

A function $f(\cdot)$ on strings is in **#P** if there exists a polynomial-time non-deterministic TM $M$ such that

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

- **CountSat** is **#P**-complete
Counting problem for property paths

**Input:** RDF graph $G$
Property path triple $\{ :a \ path :b \}$

**Output:** Count the number of solutions of $\{ :a \ path :b \}$ over $G$ (according to the semantics proposed by W3C)
The complexity of property paths is *intractable*.

**Theorem (ACP12)**

COUNTW3C is outside \( \#P \)
The complexity of property paths is *intractable*

**Theorem (ACP12)**

\[ \text{CountW3C is outside } \#P \]

\[ \text{CountW3C is hard to solve even if } P = NP \]
A *doubly exponential* lower bound for counting

- Let $\text{path}_s$ be a property path of the form

\[
(\cdots ((:p)\ast)\ast)\cdots )\ast
\]

with $s$ nested stars
A *doubly exponential* lower bound for counting

- Let $\text{paths}_s$ be a property path of the form

  $$(\cdots ((:p)^*)^*)\cdots)^*$$

  with $s$ nested stars

- Let $K_n$ be a clique with $n$ nodes
A \textit{doubly exponential} lower bound for counting

- Let $\textit{path}_s$ be a property path of the form

\[
(\cdots ((:p)*\cdots)*\cdots)*
\]

with $s$ nested stars

- Let $K_n$ be a clique with $n$ nodes

- Let $\textit{CountClique}(s, n)$ be the number of solutions of

$\{ :a0 \ \textit{path}_s :a1 \}$ over $K_n$
A doubly exponential lower bound for counting

- Let \( path_s \) be a property path of the form
  \[
  (\cdots((:p)*)*)\cdots)*
  \]
  with \( s \) nested stars
- Let \( K_n \) be a clique with \( n \) nodes
- Let \( CountClique(s, n) \) be the number of solutions of \( \{ :a0\, path_s\, :a1 \} \) over \( K_n \)

**Lemma (ACP12)**

\[
CountClique(s, n) \geq (n - 2)!^{(n-1)^{s-1}}
\]
A *doubly exponential* lower bound for counting

- Let $\text{path}_s$ be a property path of the form 
  \[(\cdots ((:p)*)*\cdots)*\]
  
  with $s$ nested stars

- Let $K_n$ be a clique with $n$ nodes

- Let $\text{CountClique}(s, n)$ be the number of solutions of
  \[
  \{ :a_0 \text{ path}_s :a_1 \}
  \]
  over $K_n$

**Lemma (ACP12)**

\[
\text{CountClique}(s, n) \geq (n - 2)!^{(n-1)^{s-1}}
\]

In [ACP12]: A recursive formula for calculating $\text{CountClique}(s, n)$
We can explain the experimental results

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

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

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can explain the experimental results

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

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can explain the experimental results

$\text{CountClique}(s, n)$ allows to \textit{fill in the blanks}

\[
\{ :a0 ((:p)*))* :a1 \}
\]

<table>
<thead>
<tr>
<th>$n$</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
</tr>
</tbody>
</table>
We can explain the experimental results

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

\[\{ :a0 ((:p)*)* :a1 \}\]

<table>
<thead>
<tr>
<th>(n)</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
</tr>
</tbody>
</table>
We can explain the experimental results

\[ \text{CountClique}(s, n) \] allows to \textit{fill in the blanks}

\[ \{ :a0 ((:p)*)* :a1 \} \]

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 ✓</td>
</tr>
<tr>
<td>3</td>
<td>6 ✓</td>
</tr>
<tr>
<td>4</td>
<td>305 ✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576 ✓</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
</tr>
</tbody>
</table>
We can explain the experimental results

\( \text{CountClique}(s, n) \) allows to fill in the blanks

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>( n )</th>
<th># Sol.</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>← 28 \times 10^9</td>
</tr>
<tr>
<td>7</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
We can explain the experimental results

\textit{CountClique}(s, n) allows to fill in the blanks

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

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 ✓</td>
</tr>
<tr>
<td>3</td>
<td>6 ✓</td>
</tr>
<tr>
<td>4</td>
<td>305 ✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576 ✓</td>
</tr>
<tr>
<td>6</td>
<td>- ← 28 \times 10^9</td>
</tr>
<tr>
<td>7</td>
<td>- ← 144 \times 10^{15}</td>
</tr>
<tr>
<td>8</td>
<td>-</td>
</tr>
</tbody>
</table>
We can explain the experimental results

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

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>(n)</th>
<th>(#) Sol.</th>
<th>(\checkmark)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
<td>(\leftarrow ) (28 \times 10^9)</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td>(\leftarrow ) (144 \times 10^{15})</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
<td>(\leftarrow ) (79 \times 10^{24})</td>
</tr>
</tbody>
</table>
We can explain the experimental results

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

\[
\{ :a0 ((:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1 ✓</td>
</tr>
<tr>
<td>3</td>
<td>6 ✓</td>
</tr>
<tr>
<td>4</td>
<td>305 ✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576 ✓</td>
</tr>
<tr>
<td>6</td>
<td>– ← 28 × 10^9</td>
</tr>
<tr>
<td>7</td>
<td>– ← 144 × 10^{15}</td>
</tr>
<tr>
<td>8</td>
<td>– ← 79 × 10^{24}</td>
</tr>
</tbody>
</table>

79 Yottabytes for the answer over a file of 379 bytes
We can explain the experimental results

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

\[
\{ :a0 ( (:p)*)* :a1 \}
\]

<table>
<thead>
<tr>
<th>n</th>
<th># Sol.</th>
<th>✓</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>✓</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>✓</td>
</tr>
<tr>
<td>4</td>
<td>305</td>
<td>✓</td>
</tr>
<tr>
<td>5</td>
<td>418,576</td>
<td>✓</td>
</tr>
<tr>
<td>6</td>
<td>–</td>
<td>← 28 \times 10^9</td>
</tr>
<tr>
<td>7</td>
<td>–</td>
<td>← 144 \times 10^{15}</td>
</tr>
<tr>
<td>8</td>
<td>–</td>
<td>← 79 \times 10^{24}</td>
</tr>
</tbody>
</table>

79 Yottabytes for the answer over a file of 379 bytes

1 Yottabyte > the estimated capacity of all digital storage in the world
Data complexity of property path is still intractable

Common assumption in Databases:

» queries are much smaller than data sources
Data complexity of property path is still intractable

Common assumption in Databases:
- queries are much smaller than data sources

*Data complexity*
- measure the complexity considering the query fixed
Data complexity of property path is still intractable

Common assumption in Databases:
- queries are much smaller than data sources

*Data complexity*
- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial
Data complexity of property path is still intractable

Common assumption in Databases:
▶ queries are much smaller than data sources

Data complexity
▶ measure the complexity considering the query fixed
▶ Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

Theorem (ACP12)
Data complexity of COUNTW3C is $\#P$-complete
Data complexity of property path is still intractable

Common assumption in Databases:
- queries are much smaller than data sources

Data complexity
- measure the complexity considering the query fixed
- Data complexity of SQL, XPath, SPARQL 1.0, etc. are all polynomial

Theorem (ACP12)
Data complexity of COUNTW3C is \#P-complete

Corollary
SPARQL 1.1 query evaluation is intractable in Data Complexity
Existential semantics: a possible alternative

Possible solution

Do not count

Just check whether there exists a path satisfying the property path expression
Existential semantics: a possible alternative

Possible solution

Do not count

Just check whether there exists a path satisfying the property path expression

Years of experiences (theory and practice) in:

- Graph Databases
- XML
- SPARQL 1.0 (PSPARQL, Gleen)

+ equivalent regular expressions giving equivalent results
Existential semantics: decision problems

**Input:** RDF graph \( G \)
- Property path triple \( \{ :a path :b \} \)

**ExistsPath**

**Question:** Is there a path from \( :a \) to \( :b \) in \( G \) satisfying the regular expression \( path \)?

**ExistsW3C**

**Question:** Is the number of solutions of \( \{ :a path :b \} \) over \( G \) greater than 0 (according to W3C semantics)?
Evaluating existential paths is tractable

Theorem (well-known result)

\[ \text{EXISTSPath} \text{ can be solved in } O(|G| \times |path|) \]

Can be proved by using automata theory
Evaluating existential paths is tractable

Theorem (well-known result)

\texttt{ExistsPath} can be solved in $O(|G| \times |\text{path}|)$

Can be proved by using automata theory

Theorem (ACP12)

\texttt{ExistsPath} and \texttt{ExistsW3C} are equivalent decision problems
Evaluating existential paths is tractable

**Theorem (well-known result)**

\( \text{ExistsPath} \) can be solved in \( O(|G| \times |\text{path}|) \)

Can be proved by using automata theory

**Theorem (ACP12)**

\( \text{ExistsPath} \) and \( \text{ExistsW3C} \) are *equivalent* decision problems

**Corollary (ACP12)**

\( \text{ExistsW3C} \) can be solved in \( O(|G| \times |\text{path}|) \)
So there are possibilities for optimization
So there are possibilities for optimization

<table>
<thead>
<tr>
<th>Corollary</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property path queries with \texttt{SELECT DISTINCT} can be efficiently evaluated</td>
</tr>
</tbody>
</table>
So there are possibilities for optimization

**Corollary**

Property path queries with `SELECT DISTINCT` can be efficiently evaluated

And we can also use `DISTINCT` over general queries

**Theorem**

`SELECT DISTINCT` SPARQL 1.1 queries are tractable in Data Complexity
SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

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

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
<th>Psparql</th>
<th>Glenn</th>
</tr>
</thead>
</table>

SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

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

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
<th>Psparql</th>
<th>Glen</th>
</tr>
</thead>
</table>

SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

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

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
<th>Psparql</th>
<th>Gleen</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>2.24</td>
<td>47.31</td>
<td>2.37</td>
<td>–</td>
<td>0.29</td>
<td>1.39</td>
</tr>
<tr>
<td>10.9KB</td>
<td>2.60</td>
<td>204.95</td>
<td>6.43</td>
<td>–</td>
<td>0.30</td>
<td>1.32</td>
</tr>
<tr>
<td>11.4KB</td>
<td>6.88</td>
<td>3222.47</td>
<td>80.73</td>
<td>–</td>
<td>0.30</td>
<td>1.34</td>
</tr>
<tr>
<td>13.2KB</td>
<td>24.42</td>
<td>–</td>
<td>394.61</td>
<td>–</td>
<td>0.31</td>
<td>1.38</td>
</tr>
<tr>
<td>14.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.33</td>
<td>1.38</td>
</tr>
<tr>
<td>17.2KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.35</td>
<td>1.42</td>
</tr>
<tr>
<td>20.5KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.44</td>
<td>1.50</td>
</tr>
<tr>
<td>25.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.45</td>
<td>1.52</td>
</tr>
</tbody>
</table>
SPARQL 1.1 implementations do not take advantage of `SELECT DISTINCT`

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

<table>
<thead>
<tr>
<th>Input</th>
<th>ARQ</th>
<th>RDFQ</th>
<th>Kgram</th>
<th>Sesame</th>
<th>Psparql</th>
<th>Glenn</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.2KB</td>
<td>2.24</td>
<td>47.31</td>
<td>2.37</td>
<td>–</td>
<td>0.29</td>
<td>1.39</td>
</tr>
<tr>
<td>10.9KB</td>
<td>2.60</td>
<td>204.95</td>
<td>6.43</td>
<td>–</td>
<td>0.30</td>
<td>1.32</td>
</tr>
<tr>
<td>11.4KB</td>
<td>6.88</td>
<td>3222.47</td>
<td>80.73</td>
<td>–</td>
<td>0.30</td>
<td>1.34</td>
</tr>
<tr>
<td>13.2KB</td>
<td>24.42</td>
<td>–</td>
<td>394.61</td>
<td>–</td>
<td>0.31</td>
<td>1.38</td>
</tr>
<tr>
<td>14.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.33</td>
<td>1.38</td>
</tr>
<tr>
<td>17.2KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.35</td>
<td>1.42</td>
</tr>
<tr>
<td>20.5KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.44</td>
<td>1.50</td>
</tr>
<tr>
<td>25.8KB</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>–</td>
<td>0.45</td>
<td>1.52</td>
</tr>
</tbody>
</table>

Optimization possibilities can remain hidden in a complicated specification
Theory can be of help in the design of new standards

In the case of SPARQL 1.0

- theory helped to formalize of the semantics, clarify corner cases, and propose optimization procedures
Theory can be of help in the design of new standards

In the case of SPARQL 1.0

- theory helped to formalize of the semantics, clarify corner cases, and propose optimization procedures

In the case of SPARQL 1.1 and property paths

- theory helped to understand that counting may lead to unfeasibility of query evaluation
Theory can be of help in the design of new standards

In the case of SPARQL 1.0
- theory helped to formalize of the semantics, clarify corner cases, and propose optimization procedures

In the case of SPARQL 1.1 and property paths
- theory helped to understand that counting may lead to unfeasibility of query evaluation

In both cases, theory helped in designing the language
He who loves practice without theory is like the sailor who boards ship without a rudder and compass and never knows where he may cast.

Leonardo da Vinci
Semantic Web research inspired by W3C standards, or the hell of the practice without theory

Jorge Pérez

Assistant Professor
Department of Computer Science
Universidad de Chile

joint work with M. Arenas, S. Conca (PUC - Chile) and C. Gutierrez (U. Chile)
Outline

Syntax and Semantics of SPARQL

Complexity: Evaluation and Static Analysis

SPARQL 1.1 path queries
References

PAG06-09  Semantics and Complexity of SPARQL, ISWC 2006, TODS 2009
SML10  Foundations of SPARQL Query Optimization, ICDT 2010
ACP12  Counting Beyond a Yottabyte ..., WWW 2012
LPSS12  Static Analysis and Optimization of SemWeb Queries, PODS 2012
LM12  Complexity of Evaluating Path Expressions in SPARQL, PODS 2012