European Semantic Web Conference 2007
Tutorial

SPARQL – Where are we?
Current state, theory and practice

Marcelo Arenas (Pontificia Universidad Católica de Chile)
Claudio Gutiérrez (Universidad de Chile)
    Bijan Parsia (University of Manchester)
    Jorge Pérez (Pontificia Universidad Católica de Chile)
    Axel Polleres (DERI, National University of Ireland, Galway)
    Andy Seaborne (Hewlett-Packard Laboratories)

Supported by FONDECYT grants 1070732, 1070348, CWR project P04-067-F,
EU FP6 project inContext IST-034718, MEC/URJC project SWOS URJC-CM-2006-CET-0300.
SPARQL – Where are we?  
Current state, theory and practice

Marcelo Arenas\textsuperscript{1}, Claudio Gutierrez\textsuperscript{2}, Bijan Parsia\textsuperscript{3}, Jorge P´erez\textsuperscript{1}, Axel Polleres\textsuperscript{4}, and Andy Seaborne\textsuperscript{5}

\textsuperscript{1} Pontificia Universidad Católica de Chile  
\textsuperscript{2} Universidad de Chile  
\textsuperscript{3} University of Manchester  
\textsuperscript{4} DERI, National University of Ireland, Galway  
\textsuperscript{5} Hewlett-Packard Laboratories

Abstract. After the data and ontology layers of the Semantic Web stack have achieved considerable stability, the query layer, realized by SPARQL, is the next item on W3C’s agenda. Short before its completion, we will take the opportunity to reflect on the current state of the language, its applications, recent results on theoretical foundations, and also future challenges. This tutorial will teach SPARQL along two complementary streams: On the one hand, we will provide a practical introduction for newcomers, giving examples from various application domains, providing formal underpinnings and guiding attendees through the jungle of existing implementations, including those which reach beyond the current specification to query more expressive semantic web languages. On the other hand, we will go further into the theoretical foundations of SPARQL, presenting recent results of SPARQL’s complexity, formalization in terms of database theory, as well as its exact semantic relation to the other building blocks in the SW stack, namely, RDF Schema, OWL and the rules layer.

1 Motivation and Objectives

After the data and ontology layers of the Semantic Web stack have achieved considerable stability through standard recommendations such as RDF and OWL, the query layer is the next item to be completed on W3C’s agenda. This layer is realized by the SPARQL Protocol and RDF Query Language (SPARQL) currently under development by W3C’s Data Access working group (DAWG). Although the SPARQL specification is not yet 100% stable, people are taking up this specification at tremendous pace, driven by the strong need for a long awaited standard in querying the Semantic Web and being able of making use of the advantages of RDF together with common metadata-vocabularies at large scale.

This is just the right moment to reflect on the current state of the language and its applications. The contributions of this tutorial will be along two complementary main streams: On the one hand we will provide a practical introduction to SPARQL for newcomers, giving examples from various application domains, providing formal underpinnings and guiding attendees through the jungle of existing implementations, including those which reach beyond the current specification to query more expressive semantic web languages. Thus, participants will get a clear sense of the language as it is specified and as it exists in implementations. On the other hand, we will go further going into the theoretical foundations of SPARQL, presenting recent results of SPARQL’s complexity, and its exact semantics relation to the other building blocks in the SW stack, namely, RDF Schema, OWL and the upcoming rules layer. Finally, we will bring these two streams together, identifying the current limitations and challenges around SPARQL, pointing to possible extensions and emerging application fields.

After the tutorial, attendees new to SPARQL should be able to formulate queries, understand the differences and overlaps of SPARQL with traditional Database query languages and have sufficient insight to understand issues in existing SPARQL engines that might affect their applications. The theoretical background given in the afternoon session will provide deeper understanding of SPARQL’s underlying semantics and complexity. Moreover, we will provide a detailed picture of SPARQL’s position in the space of related Semantic Web standards. Finally, we will give an outlook to emerging research challenges and possible future directions.
Unit 1: SPARQL Basics
SPARQL

1. Query Language
2. Protocol
   - HTTP binding
   - SOAP binding
3. XML Results Format
   - Easy to transform (XSLT, XQuery)

- Status: Later stages of standardisation
  - Design finished, getting implementation feedback
SPARQL

- Basic Graph Pattern Matching
  - Building block for data access and extensibility
- Algebra: combining graph patterns
  - Building block for data access and extensibility
  - Filters for restricting values
- Solution Modifiers
  - ORDER BY, LIMIT/OFFSET, DISTINCT, REDUCED
- Result forms
  - SELECT, CONSTRUCT, DESCRIBE, ASK

It's Turtles all the way down

- Turtle: An RDF serialization
  - The RDF part of N3
  - Commonly used in examples (and tutorials and papers)
  - SPARQL uses Turtle+variables as triple pattern syntax

```sparql
@prefix person: <http://example/person/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .

person:A foaf:name "Alice" .
person:A foaf:mbox <mailto:alice@example.net> .
person:B foaf:name "Bob" .
```
SPARQL : Triple Pattern

@prefix person: <http://example/person/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

person:A foaf:name "Alice" .
person:A foaf:mbox <mailto:alice@example.net> .
person:B foaf:name "Bob" .

PREFIX person: <http://example/person/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name
WHERE
{ ?x foaf:name ?name }
SPARQL : FILTER

```sparql
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix stock: <http://example.org/stock#> .
@prefix inv: <http://example.org/inventory#> .
stock:book3 inv:price 5 ; inv:quantity 0 .
stock:book4 inv:price 20 ; inv:quantity 0 .

PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX stock: <http://example.org/stock#>
PREFIX inv: <http://example.org/inventory#>

SELECT ?book ?title
WHERE {
  FILTER ( ?price < 15 )
  FILTER ( ?num > 0 )
}
```

---

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>stock:book1</td>
<td>&quot;SPARQL Query Language Tutorial&quot;</td>
</tr>
</tbody>
</table>

SPARQL : OPTIONAL

```sparql
@prefix person: <http://example/person/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

person :a foaf:name "Alice" .
person :a foaf:nick "A-online" .

PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name ?nick
WHERE {
  ?x foaf:name ?name .
  OPTIONAL {?x foaf:nick ?nick}
}
```

---

<table>
<thead>
<tr>
<th>name</th>
<th>nick</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Alice&quot;</td>
<td>&quot;A-online&quot;</td>
</tr>
<tr>
<td>&quot;Bob&quot;</td>
<td></td>
</tr>
</tbody>
</table>
SPARQL: UNION

```sparql
@prefix book: <http://example/book/> .
@prefix dc10: <http://purl.org/dc/elements/1.0/> .
@prefix dc11: <http://purl.org/dc/elements/1.1/> .


PREFIX dc10: <http://purl.org/dc/elements/1.0/>
PREFIX dc11: <http://purl.org/dc/elements/1.1/>

SELECT DISTINCT ?title
{
}
```

```
<table>
<thead>
<tr>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;SPARQL Query Language Tutorial&quot;</td>
</tr>
<tr>
<td>&quot;SPARQL&quot;</td>
</tr>
<tr>
<td>&quot;SPARQL Query Language (2nd ed)&quot;</td>
</tr>
</tbody>
</table>
```

Solution Modifiers

- After matching, the set of solutions is turned into a sequence then:
  - ORDER BY
  - Project
  - DISTINCT, REDUCED
  - OFFSET
  - LIMIT
Result Sets

```
<sparql xmlns="http://www.w3.org/2005/sparql-results#">
    <head>
        <variable name="name"/>
        <variable name="mbox"/>
    </head>
    <results ordered="false" distinct="false">
        <result>
            <binding name="name"><literal>Johnny Lee Outlaw</literal></binding>
            <binding name="mbox"><uri>mailto:jlow@example.com</uri></binding>
        </result>
        <result>
            <binding name="mbox"><uri>mailto:peter@example.org</uri></binding>
        </result>
    </results>
</sparql>
```

---

<table>
<thead>
<tr>
<th>name</th>
<th>mbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Johnny Lee Outlaw&quot;</td>
<td><a href="mailto:jlow@example.com">mailto:jlow@example.com</a></td>
</tr>
</tbody>
</table>

Inference

- An RDF graph may be backed by inference
  - OWL, RDFS, application, rules

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
SELECT ?type
WHERE
{
    ?x rdf:type ?type .
}
```

```
<table>
<thead>
<tr>
<th>type</th>
</tr>
</thead>
<tbody>
<tr>
<td>:C</td>
</tr>
<tr>
<td>:D</td>
</tr>
</tbody>
</table>
```
CONSTRUCT

@prefix person: <http://example/person/> .
@prefix foaf:   <http://xmlns.com/foaf/0.1/> .
  person:a foaf:name "Alice" .
  person:a foaf:mbox <mailto:alice@example.net> .
  person:b foaf:name "Bob" .

PREFIX person: <http://example/person/> .
PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT { ?person vcard:FN ?name } WHERE { ?person foaf:name ?name . }

PREFIX person: <http://example/person/> .
PREFIX vcard: <http://www.w3.org/2001/vcard-rdf/3.0#>
person:a vcard:FN "Alice" .
person:b vcard:FN "Bob" .

SPARQL : RDF Dataset

- RDF Dataset – collection of graphs
  - One, unnamed default graph ;
  - Zero or more named graphs
- Access with the GRAPH keyword

SELECT . . .
FROM <contact.ttl>
FROM NAMED <aliceFoaf.ttl>
FROM NAMED <bobFoaf.ttl>
WHERE { . . . }

PREFIX foaf: <http://xmlns.com/foaf/0.1/> .
SELECT ?graph ?name
WHERE {
  ?alice foaf:name "Alice" .
  GRAPH ?graph
  ?person foaf:name ?name .
  }
}

13
Unit 2: SPARQL Semantics
SPARQL: A simple RDF query language

```
SELECT ?Name ?Email
WHERE
{
  ?X :name ?Name
  ?X :email ?Email
}
```

- The *semantics* of simple SPARQL queries is easy to understand, at least intuitively.

  "Give me the name and email of the resources in the datasource"
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 ) \}

A formal semantics for SPARQL is needed.

A formal approach would be beneficial

- Clarifying corner cases
- Helping in the implementation process
- Providing sound foundations

We will see:

- A formal compositional semantics based on  
[PG06: Semantics and Complexity of SPARQL]
- This formalization is the starting point of the official semantics of the SPARQL language by the W3C.
First of all, a simplified algebraic syntax

- Triple patterns: RDF triple + variables (no bnodes for now)
  
  \[ (?X, \text{name}, ?Name) \]

- The base case for the algebra is a set of triple patterns

  \[ \{ t_1, t_2, \ldots, t_k \} \]

  This is called **basic graph pattern** (BGP).

**Example**

\[ \{ (?X, \text{name}, ?Name), (?X, \text{email}, ?Email) \} \]
First of all, a simplified algebraic syntax (cont.)

- We consider initially three basic operators:
  AND, UNION, OPT.
- We will use them to construct graph pattern expressions from basic graph patterns.
- A SPARQL graph pattern:

$$(((\{t_1, t_2\} \text{ AND } t_3) \text{ OPT } \{t_4, t_5\}) \text{ AND } (t_6 \text{ UNION } \{t_7, t_8\}))$$

it is a full parenthesized expression
- Full parenthesized expressions give us explicit precedence/association.

Mappings: building block for the semantics

**Definition**

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

Given a mapping $\mu$ and a basic graph pattern $P$:
- $\text{dom}(\mu)$: the domain of $\mu$.
- $\mu(P)$: the set obtained from $P$ replacing the variables according to $\mu$

**Example**

$$\mu = \{?X \rightarrow R_1, ?Y \rightarrow R_2, ?Name \rightarrow \text{john}, ?Email \rightarrow \text{J@ed.ex}\}$$

$$P = \{(?X, \text{name}, ?Name), (?X, \text{email}, ?Email)\}$$

$$\mu(P) = \{(R_1, \text{name}, \text{john}), (R_1, \text{email, J@ed.ex})\}$$
**The semantics of basic graph pattern**

**Definition**

The evaluation of the BGP $P$ over a graph $G$, denoted by $\llbracket P \rrbracket_G$, is the set of all mappings $\mu$ such that:

- $\text{dom}(\mu)$ is exactly the set of variables occurring in $P$
- $\mu(P) \subseteq G$

**Example**

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

$$\llbracket \{(?X, \text{name}, ?Y)\} \rrbracket_G$$

$\mu_1 = \{ ?X \rightarrow R_1, ?Y \rightarrow \text{john} \}$
$\mu_2 = \{ ?X \rightarrow R_2, ?Y \rightarrow \text{paul} \}$

$\llbracket \{(?X, \text{name}, ?Y), (??X, \text{email}, ?Z)\} \rrbracket_G$

$\mu = \{ ?X \rightarrow R_1, ?Y \rightarrow \text{john}, ?Z \rightarrow \text{J@ed.ex} \}$
Example

\[
\begin{align*}
\mu_1 & = \{ X \mapsto \text{john} \} \\
\mu_2 & = \{ X \mapsto J@edu.ex \} \\
\mu_3 & = \{ X \mapsto P@edu.ex \}
\end{align*}
\]

Compatible mappings: mappings that can be merged.

**Definition**

The mappings \( \mu_1, \mu_2 \) are **compatibles** iff they agree in their **shared variables**:

- \( \mu_1(\text{?}X) = \mu_2(\text{?}X) \) for every \( \text{?}X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2) \).
- \( \mu_1 \cup \mu_2 \) is also a mapping.

**Example**

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

\( \mu_\emptyset = \{ \} \) is compatible with every mapping.
Sets of mappings and operations

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

**Definition**

**Join:** $M_1 \bowtie M_2$
- $\{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2, \text{ and } \mu_1, \mu_2 \text{ are compatibles}\}$
- extending mappings in $M_1$ with compatible mappings in $M_2$

will be used to define AND

**Definition**

**Union:** $M_1 \cup M_2$
- $\{\mu \mid \mu \in M_1 \text{ or } \mu \in M_2\}$
- mappings in $M_1$ plus mappings in $M_2$ (the usual set union)

will be used to define UNION

**Definition**

**Difference:** $M_1 \setminus M_2$
- $\{\mu \in M_1 \mid \text{ for all } \mu' \in M_2, \mu \text{ and } \mu' \text{ are not compatibles}\}$
- mappings in $M_1$ that cannot be extended with mappings in $M_2$

**Definition**

**Left outer join:** $M_1 \bowtie M_2 = (M_1 \bowtie M_2) \cup (M_1 \setminus M_2)$
- extension of mappings in $M_1$ with compatible mappings in $M_2$
- plus the mappings in $M_1$ that cannot be extended.

will be used to define OPT
Semantics of general graph patterns

Definition

Given a graph $G$ the evaluation of a pattern is recursively defined

- $\llbracket(P_1 \text{ AND } P_2)\rrbracket_G = \llbracket P_1 \rrbracket_G \times \llbracket P_2 \rrbracket_G$
- $\llbracket(P_1 \text{ UNION } P_2)\rrbracket_G = \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G$
- $\llbracket(P_1 \text{ OPT } P_2)\rrbracket_G = \llbracket P_1 \rrbracket_G \searrow \llbracket P_2 \rrbracket_G$

the base case is the evaluation of a BGP.

Example (AND)

$G:\ (R_1, \text{name, john}) \quad (R_2, \text{name, paul}) \quad (R_3, \text{name, ringo})$

$\quad (R_1, \text{email, J@ed.ex}) \quad (R_3, \text{email, R@ed.ex})$

$\quad (R_3, \text{webPage, www.ringo.com})$

\[
\llbracket \{(?X, \text{name, ?N})\} \text{ AND } \{(?X, \text{email, ?E})\}\rrbracket_G
\]

\[
\llbracket \{(?X, \text{name, ?N})\} \rrbracket_G \times \llbracket \{(?Y, \text{email, ?E})\} \rrbracket_G
\]

\[
\begin{array}{c|c|c}
\hline
R_1 & \text{john} & R_1 \\
R_2 & \text{paul} & R_1 \\
R_3 & \text{ringo} & R_3 \\
\end{array}
\]

\[
\begin{array}{c|c|c}
\mu_1 & \mu_4 & \mu_5 \\
\hline
R_1 & \text{J@ed.ex} & R_1 \\
R_3 & \text{R@ed.ex} & R_3 \\
\end{array}
\]

\[
\begin{array}{c|c|c|c}
\hline
\mu_1 \cup \mu_4 & \mu_1 \cup \mu_4 & R_1 \\
\mu_3 \cup \mu_5 & \mu_3 \cup \mu_5 & R_3 \\
\end{array}
\]
**Example (OPT)**

$$G : (R_1, \text{name, john}) \quad (R_2, \text{name, paul}) \quad (R_3, \text{name, ringo})$$

$$(R_1, \text{email, J@ed.ex}) \quad (R_3, \text{email, R@ed.ex}) \quad (R_3, \text{webPage, www.ringo.com})$$

$$[[\{(?X, \text{name, ?N})\} \ \text{OPT} \ \{(?X, \text{email, ?E})\}]_G$$

$$[[\{(?X, \text{name, ?N})\}]_G \ \text{∪} \ [[\{(?X, \text{email, ?E})\}]_G$$

<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>$?X$</td>
</tr>
<tr>
<td>$R_2$</td>
<td>paul</td>
<td>$?E$</td>
</tr>
<tr>
<td>$R_3$</td>
<td>ringo</td>
<td>$?X$</td>
</tr>
</tbody>
</table>

$$\mu_1 \ \text{∪} \ \mu_4$$

<table>
<thead>
<tr>
<th>$\mu_2$</th>
<th>$\mu_3$</th>
<th>$\mu_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_1$</td>
<td>john</td>
<td>$R_1$</td>
</tr>
<tr>
<td>$R_3$</td>
<td>ringo</td>
<td>$R_3$</td>
</tr>
<tr>
<td>$R_2$</td>
<td>paul</td>
<td>$R_3$</td>
</tr>
</tbody>
</table>

**Example (UNION)**

$$G : (R_1, \text{name, john}) \quad (R_2, \text{name, paul}) \quad (R_3, \text{name, ringo})$$

$$(R_1, \text{email, J@ed.ex}) \quad (R_3, \text{email, R@ed.ex}) \quad (R_3, \text{webPage, www.ringo.com})$$

$$[[\{(?X, \text{email, ?Info})\} \ \text{UNION} \ \{(?X, \text{webPage, ?Info})\}]_G$$

$$[[\{(?X, \text{email, ?Info})\}]_G \ \text{∪} \ [[\{(?X, \text{webPage, ?Info})\}]_G$$

<table>
<thead>
<tr>
<th>$\mu_1$</th>
<th>$\mu_2$</th>
<th>$\mu_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$?X$</td>
<td>$?Info$</td>
<td>$R_1$</td>
</tr>
<tr>
<td>$R_1$</td>
<td><a href="mailto:J@ed.ex">J@ed.ex</a></td>
<td></td>
</tr>
<tr>
<td>$R_3$</td>
<td><a href="mailto:R@ed.ex">R@ed.ex</a></td>
<td></td>
</tr>
</tbody>
</table>

$$\mu_1 \ \text{∪} \ \mu_3$$

<table>
<thead>
<tr>
<th>$\mu_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R_3$</td>
</tr>
<tr>
<td><a href="http://www.ringo.com">www.ringo.com</a></td>
</tr>
</tbody>
</table>
Boolean filter expressions (value constraints)

In filter expressions we consider

- the equality = among variables and RDF terms
- a unary predicate bound
- boolean combinations (\(\land\), \(\lor\), \(\neg\))

A mapping \(\mu\) satisfies

- \(?X = c\) if \(\mu(?X) = c\)
- \(?X = ?Y\) if \(\mu(?X) = \mu(?Y)\)
- bound(?X) if \(\mu\) is defined in ?X, i.e. \(?X \in \text{dom}(\mu)\)

Satisfaction of value constraints

- If \(P\) is a graph pattern and \(R\) is a value constraint then \((P \text{ FILTER } R)\) is also a graph pattern.

Definition

Given a graph \(G\)

\[
\|\{(P \text{ FILTER } R)\}\|_G = \{\mu \in \|P\|_G \mid \mu \text{ satisfies } R\}
\]

i.e. mappings in the evaluation of \(P\) that satisfy \(R\).
Example (FILTER)

\[
G : (R_1, \text{name, john}) \quad (R_2, \text{name, paul}) \quad (R_3, \text{name, ringo}) \\
(R_1, \text{email, J@ed.ex}) \quad (R_3, \text{email, R@ed.ex}) \quad (R_3, \text{webPage, www.ringo.com})
\]

\[
\llbracket \{ (\exists X, \text{name, } ?N) \} \text{ FILTER } (?N = \text{ringo} \lor ?N = \text{paul}) \rrbracket_G
\]

\[
\begin{array}{c|c}
?X & ?N \\
\hline
R_1 & \text{john} \\
R_2 & \text{paul} \\
R_3 & \text{ringo} \\
\end{array}
\]

\[
\mu_1 \cup \mu_4 \\
\mu_2 \\
\mu_3 \cup \mu_5 \\
\mu_2
\]

\[
\begin{array}{c|c|c}
\hline
R_1 & \text{j@ed.ex} \\
R_2 & \text{paul} \\
R_3 & \text{R@ed.ex} \\
\end{array}
\]

\[
\neg \text{bound}(?E)
\]

Example (FILTER)

\[
G : (R_1, \text{name, john}) \quad (R_2, \text{name, paul}) \quad (R_3, \text{name, ringo}) \\
(R_1, \text{email, J@ed.ex}) \quad (R_3, \text{email, R@ed.ex}) \quad (R_3, \text{webPage, www.ringo.com})
\]

\[
\llbracket \{ (\exists X, \text{name, } ?N) \} \text{ OPT } \{ (\exists X, \text{email, } ?E) \} \text{ FILTER } \neg \text{bound}(?E) \rrbracket_G
\]

\[
\begin{array}{c|c|c}
\hline
R_1 & \text{j@ed.ex} \\
R_3 & \text{R@ed.ex} \\
\end{array}
\]

\[
\mu_2 \\
\mu_2
\]
**FILTER:** differences with the official specification

- We restrict to the case in which all variables in $R$ are mentioned in $P$.
- This restriction is not imposed in the official specification by W3C.
- The semantics without the restriction does not modify the expressive power of the language.

**SPARQL Datasets**

- One of the interesting features of SPARQL is that a query may retrieve data from different sources.

**Definition**

A SPARQL dataset is a set

\[ \mathcal{D} = \{G_0, \langle u_1, G_1 \rangle, \langle u_2, G_2 \rangle, \ldots, \langle u_n, G_n \rangle \} \]

- $G_0$ is the default graph, $\langle u_i, G_i \rangle$ are named graphs
- $\text{name}(\mathcal{D}) = \{u_1, u_2, \ldots, u_n\}$
- $d_{\mathcal{D}}$ is a function such that $d_{\mathcal{D}}(u_i) = G_i$. 
The **GRAPH** operator

if $u$ is an IRI, $?X$ is a variable and $P$ is a graph pattern, then

- $(u \ \text{GRAPH} \ P)$ is a graph pattern
- $(?X \ \text{GRAPH} \ P)$ is a graph pattern

GRAPH will permit us to dynamically change the graph against which our pattern is evaluated.

---

**Semantics of GRAPH**

**Definition**

Given a dataset $\mathcal{D}$ and a graph pattern $P$

$$
\llbracket (u \ \text{GRAPH} \ P) \rrbracket_G = \llbracket P \rrbracket_{D(u)}
$$

$$
\llbracket (?X \ \text{GRAPH} \ P) \rrbracket_G = \bigcup_{u \in \text{name}(\mathcal{D})} \left( \llbracket P \rrbracket_{D(u)} \bowtie \{X \rightarrow u\} \right)
$$

**Definition**

The evaluation of a general pattern $P$ against a dataset $\mathcal{D}$, denoted by $\llbracket P \rrbracket_\mathcal{D}$, is the set $\llbracket P \rrbracket_{G_0}$ where $G_0$ is the default graph in $\mathcal{D}$.
Example (GRAPH)

\[ D \]

\[ \langle \text{tb}, G_1: (R_1, \text{name}, \text{john}) (R_2, \text{name}, \text{paul}) \rangle \]

\[ \langle \text{trs}, G_2: (R_4, \text{name}, \text{mick}) (R_5, \text{name}, \text{keith}) (R_4, \text{email}, \text{M@ed.ex}) (R_5, \text{email}, \text{K@ed.ex}) \rangle \]

\[ \[\{\{?X, \text{name}, \text{?N}\}\}\] \]

M. Arenas, C. Gutierrez, J. Pérez – SPARQL Formalization 27 / 34

Example (GRAPH)

\[ D \]

\[ \langle \text{tb}, G_1: (R_1, \text{name}, \text{john}) (R_2, \text{name}, \text{paul}) \rangle \]

\[ \langle \text{trs}, G_2: (R_4, \text{name}, \text{mick}) (R_5, \text{name}, \text{keith}) (R_4, \text{email}, \text{M@ed.ex}) (R_5, \text{email}, \text{K@ed.ex}) \rangle \]

\[ \[\{\{?X, \text{name}, \text{?N}\}\}\] \]

\[ \[\{\{?X, \text{name}, \text{?N}\}\}\] \]

\[ \\mu_1 \]

\[ \mu_2 \]

\[ R_4 \text{ mick} \]

\[ R_5 \text{ keith} \]

M. Arenas, C. Gutierrez, J. Pérez – SPARQL Formalization 28 / 34
SELECT

- Up to this point we have concentrated in the body of a SPARQL query, i.e. in the graph pattern matching expression.
- A query can also process the values of the variables. The most simple processing operation is the selection of some variables appearing in the query.

**Definition**
- A SELECT query is a tuple \((W, P)\) where \(P\) is a graph pattern and \(W\) is a set of variable.
- The answer of a SELECT query against a dataset \(D\) is

\[
\{ \mu_W \mid \mu \in \llbracket P \rrbracket_D \}
\]

where \(\mu_W\) is the restriction of \(\mu\) to domain \(W\).

CONSTRUCT

- A query can also output an RDF graph.
- The construction of the output graph is based on a template.
- A template is a set of triple patterns possibly with bnodes.

**Example**

\[ T_1 = \{ (?X, \text{name}, ?Y), (?X, \text{info}, ?I), (?X, \text{addr}, B) \} \]

with \(B\) a bnode

**Definition**
- A CONSTRUCT query is a tuple \((T, P)\) where \(P\) is a graph pattern and \(T\) is a template.
CONSTRUCT: Semantics

**Definition**

The answer of a CONSTRUCT query \((T, P)\) against a dataset \(\mathcal{D}\) is obtained by

- for every \(\mu \in \llbracket P \rrbracket_{\mathcal{D}}\) create a template \(T_{\mu}\) with fresh bnodes
- take the union of \(\mu(T_{\mu})\) for every \(\mu \in \llbracket P \rrbracket_{\mathcal{D}}\)
- discard the not valid RDF triples
  - some variables have not been instantiated.
  - bnodes in predicate positions

Blank nodes in graph patterns

- We allow now bnodes in triple patterns.
- Bnodes act as existentials scoped to the basic graph pattern.

**Definition**

The evaluation of the BGP \(P\) with bnodes over the graph \(G\) denoted \(\llbracket P \rrbracket_G\), is the set of all mappings \(\mu\) such that:

- \(\text{dom}(\mu)\) is exactly the set of variables occurring in \(P\),
- there exists a function \(\theta\) from bnodes of \(P\) to \(G\) such that
  \[ \mu(\theta(P)) \subseteq G. \]

- A natural extension of BGPs without bnodes.
- The algebra remains the same.
Bag/Multiset semantics

- In a bag, a mapping can have cardinality greater than one.
- Every mapping \( \mu \) in a bag \( M \) is annotated with an integer \( c_M(\mu) \) that represents its cardinality (\( c_M(\mu) = 0 \) if \( \mu \not\in M \)).
- Operations between sets of mappings can be extended to bags maintaining duplicates:

\[
\begin{align*}
\mu \in M &= M_1 \times M_2, \quad c_M(\mu) = \sum_{\mu_1 \cup \mu_2 = \mu} c_{M_1}(\mu_1) \cdot c_{M_2}(\mu_2), \\
\mu \in M &= M_1 \cup M_2, \quad c_M(\mu) = c_{M_1}(\mu) + c_{M_2}(\mu), \\
\mu \in M &= M_1 \setminus M_2, \quad c_M(\mu) = c_{M_1}(\mu).
\end{align*}
\]

- Intuition: we simply do not discard duplicates.

References

Unit 4: SPARQL Foundations
RDF and SPARQL: Database Foundations

Marcelo Arenas, Claudio Gutierrez, Jorge Pérez

Department of Computer Science
Pontificia Universidad Católica de Chile
Universidad de Chile

Center for Web Research
http://www.cwr.cl/

Outline

- Part I: Querying RDF Data
  - The RDF data model
  - Querying: The simple and the ideal
  - Querying: Semantics and Complexity

- Part II: Querying Data with SPARQL
  - Decisions taken
  - Decisions to be taken

- Conclusions
RDF in a nutshell

- RDF is the W3C proposal framework for representing information in the Web.
- Abstract syntax based on directed labeled graph.
- Schema definition language (RDFS): Define new vocabulary (typing, inheritance of classes and properties).
- Extensible URI-based vocabulary.
- Support use of XML schema datatypes.
- Formal semantics.

RDF formal model

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

A set of RDF triples is called an RDF graph
RDFS: An example

```
RDFS: An example

Some difficulties:
- Existential variables as datavalues
- Built-in vocabulary with fixed semantics (RDFS)
- Graph model where nodes may also be edge labels

RDF data processing can take advantage of database techniques:
- Query processing
- Storing
- Indexing
```
Entailment of RDF graphs

Entailment of RDF graphs:
- Can be defined in terms of classical notions such as model, interpretation, etc
  - As for the case of first order logic
  - Has a graph characterization via homomorphisms.

Homomorphism

A function \( h : U \cup B \cup L \to U \cup B \cup L \) is a homomorphism \( h \) from \( G_1 \) to \( G_2 \) if:
- \( h(c) = c \) for every \( c \in U \cup L \);
- for every \( (a, b, c) \in G_1 \), \( (h(a), h(b), h(c)) \in G_2 \)

Notation: \( G_1 \to G_2 \)

Example: \( h = \{ B \mapsto b \} \)
Entailment

**Theorem (CM77)**

$G_1 \models G_2$ if and only if there is a homomorphism $G_2 \rightarrow G_1$.

![Diagram](image)

**Complexity**

Entailment for RDF is NP-complete

Graphs with RDFS vocabulary

Previous characterization of entailment is not enough to deal with RDFS vocabulary: *(Ronaldinho, rdf:type, person)*

![RDFS Diagram](image)
Graphs with RDFS vocabulary

Built-in predicates have pre-defined semantics:
- `rdf:sc`: transitive
- `rdf:sp`: transitive

More complicated interactions: \[
\begin{align*}
(p, \text{rdf:dom}, c) & \quad (a, p, b) \\
(a, \text{rdf:type}, c)
\end{align*}
\]

RDFS-entailment can be characterized by a set of rules

- An Existential rule
- Subproperty rules
- Subclass rules
- Typing rules
- Implicit typing

Graphs with RDFS vocabulary: Inference rules

Inference system in [MPG07] has 14 rules:

**Existential rule** : \[
\frac{G_2}{G_1} \text{ if } G_2 \rightarrow G_1
\]

**Subproperty rules** : \[
\frac{(p, \text{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}
\]

**Subclass rules** : \[
\frac{(a, \text{rdf:sc}, b) \quad (b, \text{rdf:sc}, c)}{(a, \text{rdf:sc}, c)}
\]

**Typing rules** : \[
\frac{(p, \text{rdf:dom}, c) \quad (a, p, b)}{(a, \text{rdf:type}, c)}
\]

**Implicit typing** : \[
\frac{(q, \text{rdf:dom}, a) \quad (p, \text{rdf:sp}, q) \quad (b, p, c)}{(b, \text{rdf:type}, a)}
\]
RDFS Entailment

Theorem (H04,GHM04,MPG07)

\[ G_1 \models G_2 \text{ iff there is a proof of } G_2 \text{ from } G_1 \text{ using the system of 14 inference rules.} \]

Complexity

RDFS-entailment is NP-complete.

Proof idea

Membership in NP: If \( G_1 \models G_2 \), then there exists a polynomial-size proof of this fact.

Closure of an RDF Graph

Notation:

\[
\begin{align*}
ground(G) & : \text{Graph obtained by replacing every blank } B \text{ in } G \text{ by a constant } c_B. \\
ground^{-1}(G) & : \text{Graph obtained by replacing every constant } c_B \text{ in } G \text{ by } B.
\end{align*}
\]

Closure of an RDF graph \( G \) (denoted by \( \text{closure}(G) \)):

\[
G \cup \{ t \in (U \cup B) \times U \times (U \cup B \cup L) \ | \\
\text{there exists a ground tuple } t' \text{ such that} \\
ground(G) \models t' \text{ and } t = \text{ground}^{-1}(t') \}
\]
Closure of an RDF Graph: Example

\[
\text{rdf : sc} \\
\text{b} \\
\text{a}
\]

\[
\text{rdf : sc} \\
\text{b} \\
\text{a}
\]

Closure of an RDF graph: complexity

**Proposition (H04,GHM04,MPG07)**

\[G_1 \models G_2 \iff G_2 \text{ → } \text{closure}(G_1)\]

**Complexity**

*The closure of G can be computed in time* \(O(|G|^4 \cdot \log |G|)\).

Can the closure be used in practice?

- Can we use an alternative materialization?
- Can we materialize a small part of the closure?
Core of an RDF Graph

An RDF Graph $G$ is a core if there is no homomorphism from $G$ to a proper subgraph of it.

Theorem (HN92,FKP03,GHM04)

- Each RDF graph $G$ has a unique core (denoted by $\text{core}(G)$).
- Deciding if $G$ is a core is coNP-complete.
- Deciding if $G = \text{core}(G')$ is DP-complete.

Core and RDFS

For RDF graphs with RDFS vocabulary, the core of $G$ may contain redundant information:

\[ \text{rdf:s} : \text{sc} \]

\[ \text{rdf:s} : \text{sc} \]

\[ \text{rdf:s} : \text{sc} \]

\[ \text{rdf:s} : \text{sc} \]

\[ \text{rdf:s} : \text{sc} \]

\[ \text{rdf:s} : \text{sc} \]
A normal form for RDF graphs

To reduce the size of the materialization, we can combine both core and closure.

\[ \text{nf}(G) = \text{core} ( \text{closure}(G)) \]

Theorem (GHM04)

\[ G_1 \text{ is equivalent to } G_2 \text{ iff } \text{nf}(G_1) \approx \text{nf}(G_2). \]

\[ G_1 \models G_2 \text{ iff } G_2 \rightarrow \text{nf}(G_1) \]

Complexity

The problem of deciding if \( G_1 = \text{nf}(G_2) \) is DP-complete.

Querying RDF data: Desiderata

Let \( D \) be a database, \( Q \) a query, and \( Q(D) \) the answer.

\[ \begin{align*}
\text{Outputs should belong to the same family of objects as inputs} \\
\text{If } D \equiv D', \text{ then } Q(D) = Q(D') \\
\text{(Weaker) If } D \equiv D', \text{ then } Q(D) \cong Q(D') \\
Q(D) \text{ should have no (or minimal) redundancies} \\
The framework should be extensible to RDFS \\
\text{(Should the framework be extensible to OWL?)} \\
\text{Incorporate to the framework the notion of entailment}
\end{align*} \]
Querying RDF data: Desiderata

Outputs should belong to the same family of objects as inputs

- Allows compositionality of queries
- Allows defining views
- Allows rewriting

In RDF, the natural objects of input/output are RDF graphs.

If $D \equiv D'$, then $Q(D) = Q(D')$
(Weaker) If $D \equiv D'$, then $Q(D) \cong Q(D')$

- Outputs are syntactic or semantic objects?
- Need a notion of "equivalent" databases ($\equiv$)
  (In RDF, there is a standard notion of logical equivalence)
- One could just ask logical equivalence in the output
- In RDF there is an intermediate notion: graph isomorphism
Querying RDF data: Desiderata

\( Q(D) \) should have no (or minimal) redundancies

- Desirable to avoid inconsistencies
- Desirable to improve processing time and space
- Standard requirement for exchange information

The framework should be extensible to RDFS (Should the framework be extensible to OWL?)

- A basic requirement of the Semantic Web Architecture
- Extension to OWL are not trivial because of the known mismatch
- Not necessarily related to the type of semantics given (logical framework, graph matching, etc.)
Querying RDF data: Desiderata

Incorporate to the framework the notion of entailment

- RDF graphs are not purely syntactic objects
- Would like to incorporate KB framework
- Beware of the complexity issues! RDF navigates on the Web
- Find the good compromise

Querying RDF data: Definitions

A conjunctive query $Q$ is a pair of RDF graphs $H, B$ where some resources have been replaced by variables $\bar{X}, \bar{Y}$ in $V$.

$$Q : \quad H(\bar{X}) \leftarrow B(\bar{X}, \bar{Y})$$

Issues:
- Free variables in $B$ (projection)
- Treatment of blank nodes in $B$
- Treatment of blank nodes in $H$
A valuation is a function $v : V \rightarrow U \cup B \cup L$

A matching of a graph $B$ in the database $D$ is a valuation $v$ such that $v(B) \subseteq D$.

A pre-answer to $Q$ over $D$ is the set $\text{preans}(Q, D) = \{ v(H) : v \text{ is a matching of } B \text{ in } D \}$

A single answer is an element of $\text{preans}(Q, D)$

---

Querying RDF data: Two semantics

**Union:** answer $Q(D)$ is the union of all single answers

\[ \text{ans}_U(Q, D) = \bigcup \text{preans}(Q, D) \]

**Merge:** answer $Q(D)$ is the merge of all single answers

\[ \text{ans}_M(Q, D) = \biguplus \text{preans}(Q, D) \]

**Proposition**

1. For both semantics, if $D \models D'$ then $\text{ans}(Q, D') \models \text{ans}(Q, D)$
2. For all $D$, $\text{ans}_U(Q, D) \models \text{ans}_M(Q, D)$
3. With merge semantics, we cannot represent the identity query
Querying RDF data: refined semantics

Problem
Two non-isomorphic datasets $D, D'$ give different answers to the same query.

A slightly refined semantics:
1. Normalize $D$ before querying
2. Then query as usual over $nf(D)$

**Good** News: if $D \equiv D'$ then $Q(D) \cong Q(D')$
**Bad** News: computing $nf(D)$ is hard

Querying RDF data: refined semantics (cont.)

The news as formal results:

**Theorem (MPG07)**

*Do not need to compute the normal form.*

**Theorem (FG06)**

*If a query language has the following two properties:*
  1. for all $Q$, if $D \equiv D'$ then $Q(D) = Q(D')$,
  2. can represent the identity query,

*then the complexity of evaluation is NP-hard (in data complexity).*
A query $Q$ contains a query $Q'$, denoted $Q \subseteq Q'$ iff $\text{ans}(Q, D)$ comprises all the information of $\text{ans}(Q', D)$.

In classical DB: $\text{ans}(Q, D) \subseteq \text{ans}(Q', D)$

In our setting we have two versions:

- $\text{ans}(Q', D) \subseteq \text{ans}(Q, D)$ ($Q \subseteq_p Q'$)
- $\text{preans}(Q, D) \subseteq \text{preans}(Q', D)$ (modulo iso) ($Q \subseteq_m Q'$)

For ground RDF both notions coincide.

Query complexity version: The evaluation problem is NP-complete

Data complexity version: The evaluation problem is polynomial
Querying with SPARQL

- SPARQL is the W3C candidate recommendation query language for RDF.
- SPARQL is a graph-matching query language.
- A SPARQL query consists of three parts:
  - Pattern matching: optional, union, nesting, filtering.
  - Solution modifiers: projection, distinct, order, limit, offset.
  - Output part: construction of new triples, ... .

Recall the formalization from Unit-2

Syntax:
- Triple patterns: RDF triple + variables (no bnodes)
- Operators between triple patterns: AND, UNION, OPT.
- Filtering of solutions: FILTER.
- A full parenthesized algebra.
Recall the formalization from Unit-2

Semantics:

- Based on mappings, partial functions from variables to terms.
- A mapping $\mu$ is a solution of triple pattern $t$ in $G$ iff
  - $\mu(t) \in G$
  - $\text{dom}(\mu) = \text{var}(t)$.
- $\llbracket t \rrbracket_G$ is the evaluation of $t$ in $G$, the set of solutions.

Example

<table>
<thead>
<tr>
<th>$G$</th>
<th>$t$</th>
<th>$\llbracket t \rrbracket_G$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$(R_1, \text{name}, \text{john})$</td>
<td>$?X, \text{name}, ?Y$</td>
<td>$\mu_1: R_1 \text{ john}$</td>
</tr>
<tr>
<td>$(R_1, \text{email}, \text{<a href="mailto:J@edu.ex">J@edu.ex</a>})$</td>
<td></td>
<td>$\mu_2: R_2 \text{ paul}$</td>
</tr>
<tr>
<td>$(R_2, \text{name}, \text{paul})$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Compatible mappings

Definition

Two mappings are compatible if they agree in their shared variables.

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></td>
<td>$R_1$</td>
<td>$R_1$</td>
<td>$R_1$</td>
</tr>
<tr>
<td>$\text{john}$</td>
<td></td>
<td>$\text{<a href="mailto:J@edu.ex">J@edu.ex</a>}$</td>
<td>$\text{john}$</td>
<td>$\text{<a href="mailto:J@edu.ex">J@edu.ex</a>}$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$<a href="mailto:P@edu.ex">P@edu.ex</a>$</td>
<td>$<a href="mailto:P@edu.ex">P@edu.ex</a>$</td>
<td>$<a href="mailto:P@edu.ex">P@edu.ex</a>$</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:

**Definition**

**Join:** $M_1 \Join M_2$

- extending mappings in $M_1$ with compatible mappings in $M_2$

**Difference:** $M_1 \setminus M_2$

- mappings in $M_1$ that cannot be extended with mappings in $M_2$

**Union:** $M_1 \cup M_2$

- mappings in $M_1$ plus mappings in $M_2$ (set theoretical union)

**Definition**

**Left Outer Join:** $M_1 \JoinLeft M_2 = (M_1 \Join M_2) \cup (M_1 \setminus M_2)$

Semantics of general graph patterns

**Definition**

Given a graph $G$ the evaluation of a pattern is recursively defined

- $\llbracket (P_1 \text{ AND } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \Join \llbracket P_2 \rrbracket_G$
- $\llbracket (P_1 \text{ UNION } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \cup \llbracket P_2 \rrbracket_G$
- $\llbracket (P_1 \text{ OPT } P_2) \rrbracket_G = \llbracket P_1 \rrbracket_G \Join \llbracket P_2 \rrbracket_G$
- $\llbracket (P \text{ FILTER } R) \rrbracket_G = \{ \mu \in \llbracket P \rrbracket_G \mid \mu \text{ satisfies } R \}$
Differences with Relational Algebra / SQL

- Not a fixed output schema
  - mappings instead of tables
  - schema is implicit in the domain of mappings
- Too many NULLs
  - mappings with disjoint domains can be joined
  - mappings with distinct domains in output solutions
- SPARQL-to-SQL translations experience this issues
  - need of IS NULL/IS NOT NULL in join/outerjoin conditions
  - need of COALESCE in constructing output schema

SPARQL complexity: the evaluation problem

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

Question:
Is the mapping in the evaluation of the pattern against the graph?

$$\mu \in \llbracket P \rrbracket_G?$$
Evaluation of **AND-FILTER** patterns is polynomial.

**Theorem (PAG06)**

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

\[ O(|P| \times |G|). \]

**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 (PAG06)**

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

**Proof idea**

- Reduction from **3SAT**.
- A pattern encodes the propositional formula.
- \( \neg \) bound is used to encode negation.
Evaluation including \textbf{OPT} is \textPSPACE-completet.\textit{

\begin{center}
\textbf{Theorem (PAG06)}
\end{center}

For patterns using \textbf{AND}, \textbf{FILTER} and \textbf{OPT} operators, the evaluation problem is \textPSPACE-complete.

\begin{center}
\textbf{Proof idea}
\end{center}

\begin{itemize}
\item \textit{Reduction from QBF}
\item A pattern encodes a quantified propositional formula:
\[
\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \psi.
\]
\item \textit{nested OPTs are used to encode quantifier alternation.}
\begin{center}
\textit{(This time, we do not need $\neg$-bound.)}
\end{center}
\end{itemize}

\section*{PSPACE-hardness: A closer look}

Assume $\varphi = \forall x_1 \exists y_1 \psi$, where $\psi = (x_1 \lor \neg y_1) \land (\neg x_1 \lor y_1)$.

We generate $G$, $P_\varphi$ and $\mu_0$ such that $\mu_0$ belongs to the answer of $P_\varphi$ over $G$ iff $\varphi$ is valid:

\begin{align*}
G & : \{(a, \text{tv}, 0), \ (a, \text{tv}, 1), \ (a, \text{false}, 0), \ (a, \text{true}, 1)\} \\

P_\psi & : \ \ ((a, \text{tv}, ?X_1) \ \text{AND} \ (a, \text{tv}, ?Y_1)) \ \text{FILTER} \\
& \quad \quad \ ((?X_1 = 1 \lor ?Y_1 = 0) \land (?X_1 = 0 \lor ?Y_1 = 1)) \\

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

\mu_0 & : \ \{?B_0 \mapsto 1\}
\end{align*}
PSPACE-hardness: A closer look

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

\[
P_1 : (a, \text{tv}, ?X_1)
\]

\[
Q_1 : (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0)
\]

\[
P_1 \rightarrow ?B_0 \mapsto 1
\]

\[
?X_1 \mapsto 1 \quad ?X_1 \mapsto 0 \quad ?Y_1 \mapsto i \quad ?B_0 \mapsto 0
\]

\[
Q_1 \rightarrow ?X_1 \mapsto 1
\]

\[
?X_1 \mapsto 1 \quad ?X_1 \mapsto 0 \quad ?Y_1 \mapsto j \quad ?B_0 \mapsto 0
\]

Data–complexity is polynomial

**Theorem (PAG06)**

*When patterns are consider to be fixed (data complexity), the evaluation problem is in LOGSPACE.*

**Proof idea**

*From data–complexity of first–order logic.*
SPARQL reordering/optimization: a simple normal form

- AND and UNION are commutative and associative.
- AND, OPT, and FILTER distribute over UNION.

Theorem (UNION Normal Form)

Every graph pattern is equivalent to one of the form

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

where each \( P_i \) is UNION–free.

We concentrate in UNION-free patterns.

Well–designed patterns

Definition

A graph pattern is well–designed iff for every OPT in the pattern

\[
\begin{align*}
\downarrow & \quad \downarrow & \quad \downarrow & \quad \uparrow \\
\cdots & \quad (A \text{ OPT } B) & \cdots \\
\end{align*}
\]

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

Example

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

The variable \( \text{?X} \) is inside \( B \) and anywhere outside the OPT, but it does not also occur inside \( A \).
Well–designed patterns and PSPACE-hardness

In the PSPACE-hardness reduction we use this formula:

\[
\begin{align*}
P_\varphi &: (a, \text{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 &: (a, \text{tv}, ?X_1) \\
Q_1 &: (a, \text{tv}, ?X_1) \text{ AND } (a, \text{tv}, ?Y_1) \text{ AND } (a, \text{false}, ?B_0)
\end{align*}
\]

It is not well-designed: \( B_0 \)

Well–designed patterns: reordering/optimization

For well-designed patterns

\[
\begin{align*}
\text{\ding{52}} & \; P_1 \text{ AND } (P_2 \text{ OPT } P_3) \equiv (P_1 \text{ AND } P_2) \text{ OPT } P_3 \\
\text{\ding{52}} & \; (P_1 \text{ OPT } P_2) \text{ OPT } P_3 \equiv (P_1 \text{ OPT } P_3) \text{ OPT } P_2
\end{align*}
\]

Theorem (OPT Normal Form)

Every well–designed pattern is equivalent to one of the form

\[
(\cdots (t_1 \text{ AND } \cdots \text{ AND } t_k) \text{ OPT } O_1)\cdots) \text{ OPT } O_n
\]

where each \( t_i \) is a triple pattern, and each \( O_j \) is a pattern of the same form.
Final remarks

- RDFS can be considered a new data model.
  - It is the W3C’s recommendation for describing Web metadata.

- RDFS can definitely benefit from database technology.
  - RDFS: Formal semantics, entailment of RDFS graphs, normal forms for RDFS graphs (closure and core).
  - SPARQL: Formal semantics, complexity of query evaluation, query optimization.
  - Updating
  - . . .

References