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

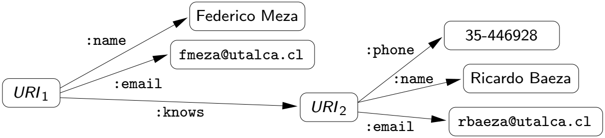
```
SELECT Name, Salary
FROM Employees
WHERE Salary >= 500000
```

$$\pi_{\texttt{Name},\texttt{Salary}}\left(\sigma_{\texttt{Salary}\geq 500000}(\texttt{Employees})\right)$$

RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

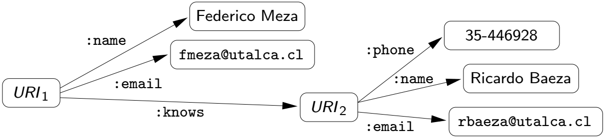
RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

RDF Graph:



RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

RDF Graph:



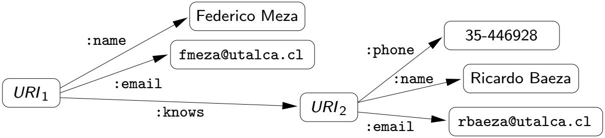
SPARQL Query:

```
SELECT ?N
WHERE
{
  ?X :name ?N .


}
```

RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

RDF Graph:



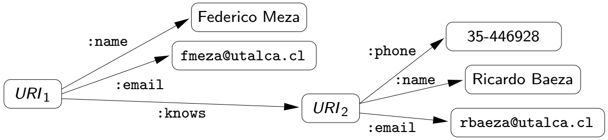
SPARQL Query:

```
SELECT ?N ?E
WHERE
{
  ?X :name ?N .
  ?X :email ?E .

}
```

RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

RDF Graph:



SPARQL Query:

```
SELECT ?N ?E
WHERE
{
  ?X :name ?N .
  ?X :email ?E .
  ?X :knows ?Y . ?Y :name "Ricardo Baeza"
}
```

But things can become more complex...

Interesting features of pattern matching on graphs

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

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping

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


  { P3 .
    P4 }


}
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts

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

  { P3 .
    P4
    OPTIONAL { P7 }  }

}
```

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting

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

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

But things can become more complex...

Interesting features of pattern matching on graphs

- Grouping
- Optional parts
- Nesting
- Union of patterns

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

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

But things can become more complex...

Interesting features of pattern matching on graphs

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

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

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

But things can become more complex...

Interesting features of pattern matching on graphs

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

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

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

But things can become more complex...

Interesting features of pattern matching on graphs

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

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

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

What is the *meaning* of a general SPARQL query?

The SPARQL W3C specification had no formal semantics!

- Specification primarily based on use cases and examples
- Semantics given in *natural language*

The SPARQL W3C specification had no formal semantics!

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

The SPARQL W3C specification had no formal semantics!

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

A formal approach is beneficial to:

- Clarify corner cases
- Help in the implementation process
- Provide sound database foundations

Semantics and Complexity of SPARQL

Jorge Pérez

PhD Student
Computer Science Department, PUC – Chile

Work presented at the *International Semantic Web Conference 2006*

W3C SPARQL specification is currently based on our work

W3C document:

- Oct-2006 back to working draft
- Finally, a recommendation in Jan-2008 incorporating our fomalization

W3C SPARQL specification is currently based on our work

W3C document:

- Oct-2006 back to working draft
- Finally, a recommendation in Jan-2008 incorporating our fomalization

In our work:

- A formal compositional semantics (for graph patterns)
- Complexity bounds
- Normalization and optimization procedures

Outline

Outline

A standard algebraic syntax

- Triple patterns: just triples + variables (V)

original SPARQL syntax	algebraic syntax
`?X :name "john"`	$(?X$, name, john$)$

- Graph patterns: full parenthesized algebra

original SPARQL syntax	algebraic syntax
`{ P1 P2 }`	$(P_1 \text{ AND } P_2)$
`{ P1 OPTIONAL { P2 }}`	$(P_1 \text{ OPT } P_2)$
`{ P1 } UNION { P2 }`	$(P_1 \text{ UNION } P_2)$
`{ P1 FILTER ( R ) }`	$(P_1 \text{ FILTER } R)$

original SPARQL syntax

algebraic syntax

A standard algebraic syntax (cont.)

- Explicit precedence/association

Example

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

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

Mappings: building block for the semantics

Definition

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

$$\mu : \text{Variables} \longrightarrow \text{RDF Terms}$$

Mappings: building block for the semantics

Definition

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

$$\mu : \text{Variables} \longrightarrow \text{RDF Terms}$$

The *evaluation* of a pattern results in a *set of mappings*.

The semantics of triple patterns

Given an RDF graph G and a triple pattern t

Definition

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

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

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 μ 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

graph	*triple*		*evaluation*	
			$?X$	$?Y$
(R_1, name, john)				
(R_1, email, J@ed.ex)	($?X$, name, $?Y$)	μ_1:	R_1	john
(R_2, name, paul)		μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph G and a triple pattern t

Definition

The *evaluation* of t over G is the set of mappings μ 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

graph

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

triple

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

evaluation

	$?X$	$?Y$
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph G and a triple pattern t

Definition

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

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

Example

graph	*triple*	*evaluation*
(R_1, name, john)		
(R_1, email, J@ed.ex)	(?X, name, ?Y)	
(R_2, name, paul)		

	?X	?Y
μ_1:	R_1	john
μ_2:	R_2	paul

Compatible mappings: mappings that can be merged.

Definition

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

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2

Compatible mappings: mappings that can be merged.

Definition

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

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2

Compatible mappings: mappings that can be merged.

Definition

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

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	

Compatible mappings: mappings that can be merged.

Definition

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

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	

Compatible mappings: mappings that can be merged.

Definition

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

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

Compatible mappings: mappings that can be merged.

Definition

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

Example

	$?X$	$?Y$	$?Z$	$?V$
μ_1 :	R_1	john		
μ_2 :	R_1		J@edu.ex	
μ_3 :			P@edu.ex	R_2
$\mu_1 \cup \mu_2$:	R_1	john	J@edu.ex	
$\mu_1 \cup \mu_3$:	R_1	john	P@edu.ex	R_2

- μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

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

- $M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2,$ and μ_1, μ_2 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

- $M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2,$ and μ_1, μ_2 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

- $M_1 \bowtie M_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in M_1, \mu_2 \in M_2$, and μ_1, μ_2 are compatible$\}$

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

- $M_1 \smallsetminus M_2 = \{\mu_1 \in M_1 \mid$ there is no mapping $\mu_2 \in M_2$ compatible with $\mu_1\}$

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

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

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

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

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

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

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

- $M_1 ⟕ M_2 = (M_1 \bowtie M_2) \cup (M_1 \smallsetminus M_2)$

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:

$$\begin{aligned} (P_1 \text{ AND } P_2) &\quad \rightarrow \\ (P_1 \text{ UNION } P_2) &\quad \rightarrow \\ (P_1 \text{ OPT } P_2) &\quad \rightarrow \end{aligned}$$

Semantics in terms of operations between evaluations

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

Definition

The evaluation of:

$$\begin{array}{rcl} (P_1 \text{ AND } P_2) & \rightarrow & M_1 \bowtie M_2 \\ (P_1 \text{ UNION } P_2) & \rightarrow & \\ (P_1 \text{ OPT } P_2) & \rightarrow & \end{array}$$

Semantics in terms of operations between evaluations

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

Definition

The evaluation of:

$$\begin{array}{ccc} (P_1 \text{ AND } P_2) & \rightarrow & M_1 \bowtie M_2 \\ (P_1 \text{ UNION } P_2) & \rightarrow & M_1 \cup M_2 \\ (P_1 \text{ OPT } P_2) & \rightarrow & \end{array}$$

Semantics in terms of operations between evaluations

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

Definition

The evaluation of:

$$
\begin{array}{ccc}
(P_1 \text{ AND } P_2) & \rightarrow & M_1 \bowtie M_2 \\
(P_1 \text{ UNION } P_2) & \rightarrow & M_1 \cup M_2 \\
(P_1 \text{ OPT } P_2) & \rightarrow & M_1 ⟕ M_2
\end{array}
$$

Simple example

Example

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

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

Simple example

Example

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

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

Simple example

Example

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

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

$?X$	$?Y$
R_1	john
R_2	paul

Simple example

Example

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

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

$?X$	$?Y$
R_1	john
R_2	paul

Simple example

Example

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

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

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?E$
R_1	J@ed.ex

Simple example

Example

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

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

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?E$
R_1	J@ed.ex

Simple example

Example

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

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

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

Simple example

Example

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

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

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join

Simple example

Example

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

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

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join
- from the Difference

Simple example

Example

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

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

$?X$	$?Y$
R_1	john
R_2	paul

$?X$	$?Y$	$?E$
R_1	john	J@ed.ex
R_2	paul	

$?X$	$?E$
R_1	J@ed.ex

- from the Join
- from the Difference
- from the Union

Boolean filter expressions (value constraints)

In filter expressions we consider

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

Satisfaction of value constraints

A mapping *satisfies*

- $?X = c$ if it gives the value c to variable $?X$
- $?X = ?Y$ if it gives the same value to $?X$ and $?Y$
- bound($?X$) if it is defined for $?X$

Satisfaction of value constraints

A mapping *satisfies*

- $?X = c$ if it gives the value c to variable $?X$
- $?X = ?Y$ if it gives the same value to $?X$ and $?Y$
- bound$(?X)$ if it is defined for $?X$

Definition

The evaluation of (P FILTER R):

- mappings in the evaluation of P that *satisfy* R.

It was not that difficult to define a formal semantics for SPARQL

It was not that difficult to define a formal semantics for SPARQL

Key aspects:

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

Outline

The evaluation decision problem

INPUT:

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

OUTPUT:

Is the *mapping* in the evaluation of the *pattern* over the *graph*?

Evaluation of simple patterns is polynomial.

Theorem

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

the evaluation problem is polynomial:

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

Evaluation of simple patterns is polynomial.

Theorem

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

the evaluation problem is polynomial:

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

Proof idea

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

Evaluation including UNION is NP-complete.

Theorem

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

the evaluation problem is NP-complete.

Evaluation including UNION is NP-complete.

Theorem

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

the evaluation problem is NP-complete.

Proof idea

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

Evaluation including UNION is NP-complete.

Theorem

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

the evaluation problem is NP-complete.

Proof idea

- Reduction from propositional SAT
- The pattern codifies a propositional formula.
- Using $\neg$ bound to codify negation.

A simple normal from

Theorem (UNION Normal Form)

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

$$P_1 \;\text{UNION}\; P_2 \;\text{UNION}\; \cdots \;\text{UNION}\; P_n$$

with P_i UNION*–free.*

A simple normal from

Theorem (UNION Normal Form)

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

$$P_1 \;\; \text{UNION} \;\; P_2 \;\; \text{UNION} \;\; \cdots \;\; \text{UNION} \;\; P_n$$

with P_i UNION–*free.*

Theorem

The evaluation problem for AND-FILTER-UNION *patterns in* UNION *normal form, is polynomial.*

Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT *operator,*

the evaluation problem is PSPACE-complete.

Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT *operator,*

the evaluation problem is PSPACE-complete.

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

Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT *operator,*

the evaluation problem is PSPACE-complete.

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

Proof idea

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

$$\forall x_1 \exists x_2 \forall x_3 \cdots \varphi.$$

Evaluation in general is PSPACE-complete.

Theorem

For general patterns that include OPT *operator,*

the evaluation problem is PSPACE-complete.

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

Proof idea

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

$$\forall x_1 \exists x_2 \forall x_3 \cdots \varphi.$$

- Using nested OPTs to codify quantifier alternations.

Outline

- Motivation
  - Our contributions
- Syntax and semantics of SPARQL graph patterns
  - Syntax
  - Semantics
- Complexity results
- Well–designed graph patterns
  - Complexity
  - Normalization and optimization

Well–designed patterns

Definition

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

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \mathrm{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs

Well–designed patterns

Definition

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

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad \underset{\uparrow}{B} \quad) \quad \cdots\cdots\cdots\cdots)$$

if a variable occurs *inside B*

Well–designed patterns

Definition

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

$$
\begin{array}{ccccccc}
(\cdots\cdots\cdots\cdots & (& A & \mathrm{OPT} & B &) & \cdots\cdots\cdots\cdots) \\
\uparrow & & & & \uparrow & & \uparrow
\end{array}
$$

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

Well–designed patterns

Definition

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

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$
$$\qquad\quad \uparrow \qquad\qquad\qquad \Uparrow \qquad\qquad\qquad \uparrow \qquad\qquad\qquad \uparrow$$

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

Well–designed patterns

Definition

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

$$(\;\cdots\cdots\cdots\cdots\quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$
$$\qquad\uparrow \qquad\qquad\qquad \Uparrow \qquad\qquad\quad \uparrow \qquad\qquad\quad \uparrow$$

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

Example

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

Well–designed patterns

Definition

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

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$

$$\uparrow \qquad\qquad \Uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

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

Example

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

Well–designed patterns

Definition

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

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$
$$\qquad\quad \uparrow \qquad\qquad\qquad \Uparrow \qquad\qquad\qquad \uparrow \qquad\qquad\qquad \uparrow$$

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

Example

$$\big[\, \big[\, (?Y, \text{name}, \text{paul}) \text{ OPT } (?X, \text{email}, ?Z) \,\big] \;\; \text{AND} \;\; (?X, \text{name}, \text{john}) \,\big]$$
$$\qquad\qquad\qquad\qquad\qquad \uparrow \qquad\qquad\qquad\qquad\qquad\qquad \uparrow$$

Well–designed patterns

Definition

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

$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \text{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$
$$\uparrow \qquad\qquad \Uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

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

Example

$$\Big[\,\big[\,(?Y,\ \text{name},\ \text{paul})\ \text{OPT}\ (?X,\ \text{email},\ ?Z)\,\big]\ \ \text{AND}\ \ (?X,\ \text{name},\ \text{john})\,\Big]$$
$$\times \qquad\qquad\qquad \uparrow \qquad\qquad\qquad\qquad \uparrow$$

Well–designed patterns

Definition

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

$$(\;\cdots\cdots\cdots\cdots\;\;(\;\;A\;\;\text{OPT}\;\;B\;\;)\;\;\cdots\cdots\cdots\cdots\;)$$

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

Example

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

- Well-designed patterns initially proposed to show equivalence with a *procedural semantics* (by W3C)

Evaluation of well-designed patterns is in coNP-complete

Theorem

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

the evaluation problem is coNP-complete

Evaluation of well-designed patterns is in coNP-complete

Theorem

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

the evaluation problem is coNP-complete

Corollary

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

the evaluation problem is coNP-complete

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.

Classical optimization is not directly applicable.

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

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are never null–rejecting
  - by definition of compatible mappings.

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are never null–rejecting
  - by definition of compatible mappings.
- Can we use classical optimization in the context of SPARQL?

Classical optimization is not directly applicable.

- Classical optimization assumes null–rejection.
  - null–rejection: the join/outer–join condition must fail in the presence of null.
- SPARQL operations are never null–rejecting
  - by definition of compatible mappings.
- Can we use classical optimization in the context of SPARQL?
  - Well-designed patterns are suitable for reordering, and then for classical optimization.

Well–designed graph patterns and optimization

Consider the following rules:

$$
\begin{aligned}
((P_1 \text{ OPT } P_2) \text{ FILTER } R) &\longrightarrow ((P_1 \text{ FILTER } R) \text{ OPT } P_2) && (1)\\
(P_1 \text{ AND } (P_2 \text{ OPT } P_3)) &\longrightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3) && (2)\\
((P_1 \text{ OPT } P_2) \text{ AND } P_3) &\longrightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2) && (3)
\end{aligned}
$$

Well–designed graph patterns and optimization

Consider the following rules:

$$
\begin{aligned}
((P_1 \text{ OPT } P_2) \text{ FILTER } R) &\longrightarrow ((P_1 \text{ FILTER } R) \text{ OPT } P_2) && (1)\\
(P_1 \text{ AND } (P_2 \text{ OPT } P_3)) &\longrightarrow ((P_1 \text{ AND } P_2) \text{ OPT } P_3) && (2)\\
((P_1 \text{ OPT } P_2) \text{ AND } P_3) &\longrightarrow ((P_1 \text{ AND } P_3) \text{ OPT } P_2) && (3)
\end{aligned}
$$

Proposition

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

Well–designed graph patterns and optimization

Definition

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

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

Well–designed graph patterns and optimization

Definition

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

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

Theorem

Every well-designed pattern is equivalent to a pattern in OPT *normal form.*

Summary

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

Summary

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

Impact:

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

Semantics and Complexity of SPARQL

Jorge Pérez

PhD Student
Computer Science Department, PUC – Chile

Work presented at the *International Semantic Web Conference 2006*