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:

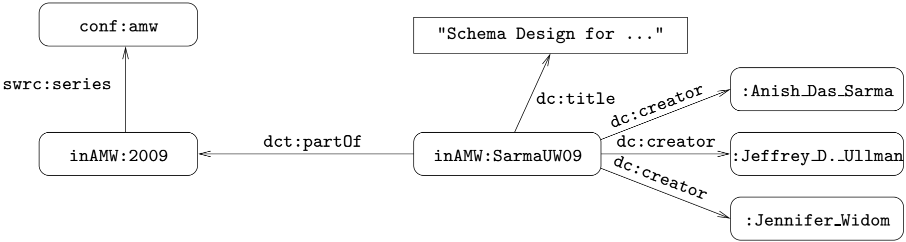
- Build a description language with standard semantics
- Make semantics machine-processable and understandable

W3C proposals:

- *Resource Description Framework (RDF)* as data model
- *SPARQL* as (graph patter 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/>
```



URIs can be used for any abstract resource

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

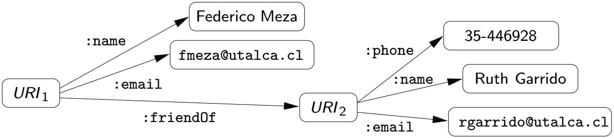
`http://dblp.l3s.de/d2r/resource/authors/Leopoldo_E._Bertossi`

`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)

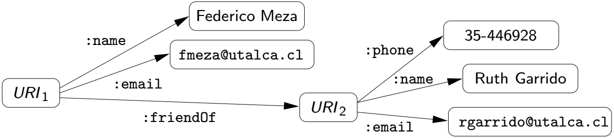
RDF Graph:



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

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

RDF Graph:



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

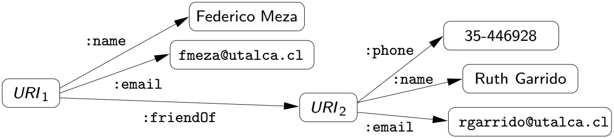
SPARQL Query:

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


}
```

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

RDF Graph:



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

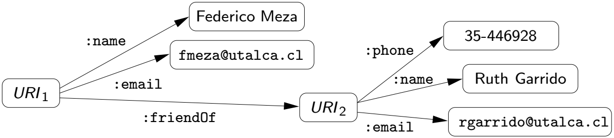
SPARQL Query:

```
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:



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

SPARQL Query:

```
SELECT ?N ?E
WHERE
{
  ?X :name ?N .
  ?X :email ?E .
  ?X :friendOf ?Y . ?Y :name "Ruth Garrido"
}
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

```
SELECT ?Author
```

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
{
  ?Paper      dc:creator     ?Author .


}
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:partOf     ?Conf .

}
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator      ?Author .
  ?Paper      dct:partOf      ?Conf .
  ?Conf       swrc:series     conf:amw .
}
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:partOf     ?Conf .
  ?Conf       swrc:series    conf:amw .
}
```

A SPARQL query consists of a:

SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW

```
SELECT ?Author
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:partOf     ?Conf .
  ?Conf       swrc:series    conf:amw .
}
```

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
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:partOf     ?Conf .
  ?Conf       swrc:series    conf:amw .
}
```

A SPARQL query consists of a:

Head: Processing of the variables

Body: Pattern matching expression

SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW, and their Web pages if this information is available:

```
SELECT ?Author ?WebPage
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:partOf     ?Conf .
  ?Conf       swrc:series    conf:amw .

  OPTIONAL {
      ?Author  foaf:homePage  ?WebPage . }
}
```

SPARQL: A Simple RDF Query Language

Example: Authors that have published in AMW, and their Web pages if this information is available:

```
SELECT ?Author ?WebPage
WHERE
{
  ?Paper      dc:creator     ?Author .
  ?Paper      dct:partOf     ?Conf .
  ?Conf       swrc:series    conf:amw .

  OPTIONAL {
      ?Author  foaf:homePage  ?WebPage . }
}
```

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

```
{ { 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

```
{ { 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

```
{ { 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
- ...

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

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

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

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
```

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

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

A standard algebraic syntax

- Triple patterns: just triples + variables

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

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

A standard algebraic syntax

- Triple patterns: just triples + variables

`?X :name "john"`	$(?X$, name, john$)$

- Graph patterns: full parenthesized algebra

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

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.

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

Mappings: building block for the semantics

Definition

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

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

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: $\text{dom}(\mu) = \text{var}(t)$

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 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*
(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

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*
(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

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		
			$?X$	$?Y$
$(R_1$, 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

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:

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

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:

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

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

$(R_1$, name, john$)$
$(R_1$, email, J@ed.ex$)$
$(R_2$, name, paul$)$

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

Simple example

Example

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

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

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

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

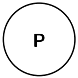
The evaluation decision problem

Evaluation problem for SPARQL patterns

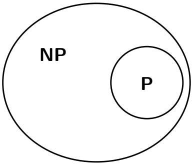
Input: A mapping μ, an RDF graph G
a graph pattern P

Output: Is the mapping μ in the evaluation of pattern P
over the RDF graph G

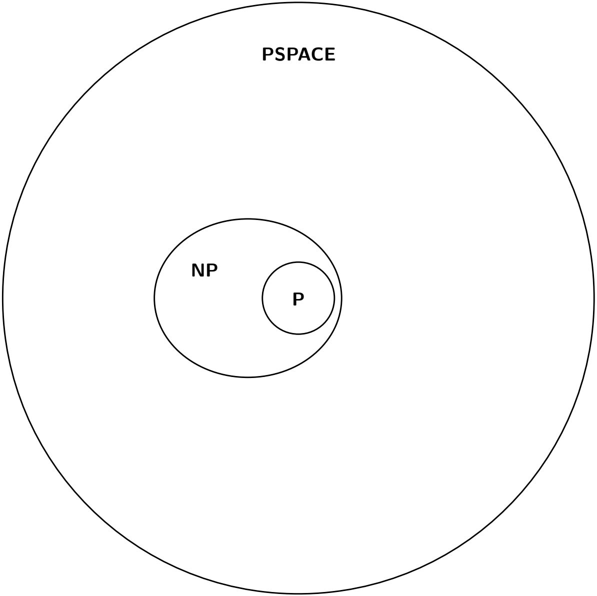
Brief complexity-theory reminder



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

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.*

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

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

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

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

if a variable occurs *inside B*

Well–designed patterns

Can we find a natural fragment with better complexity?

Definition

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

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

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

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

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

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

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

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

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

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

Example

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

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

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

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

Example

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

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

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

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

Example

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

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

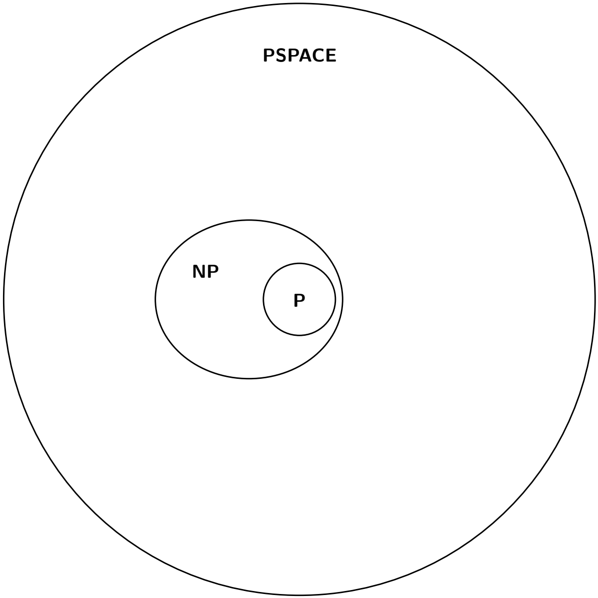
$$(\cdots\cdots\cdots\cdots \quad (\quad A \quad \mathrm{OPT} \quad B \quad) \quad \cdots\cdots\cdots\cdots)$$
$$\uparrow \qquad\qquad \Uparrow \qquad\qquad \uparrow \qquad\qquad \uparrow$$

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

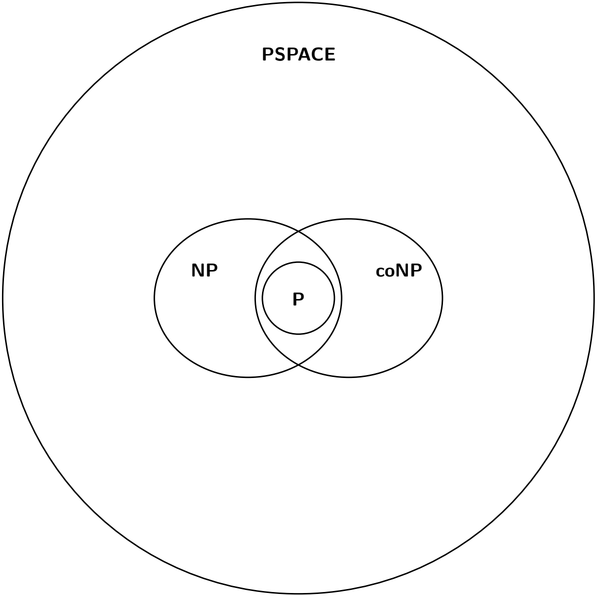
Example

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

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*

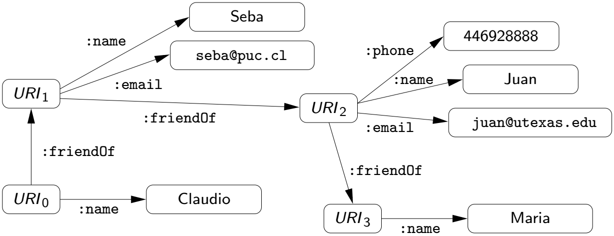
Well-designed patterns also allow to study static analysis:

Theorem (LPPS12)

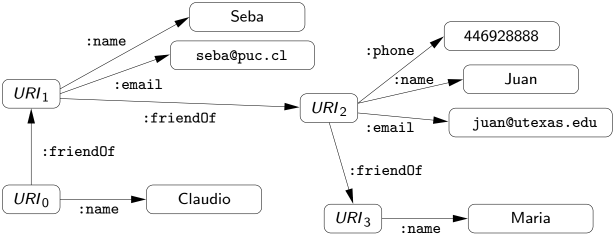
Equivalence of well-designed SPARQL patterns is in **NP**

Outline

SPARQL 1.0 provides limited navigational capabilities

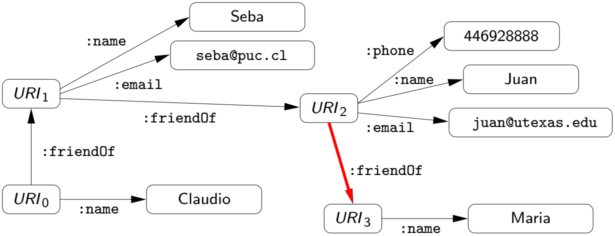


SPARQL 1.0 provides limited navigational capabilities



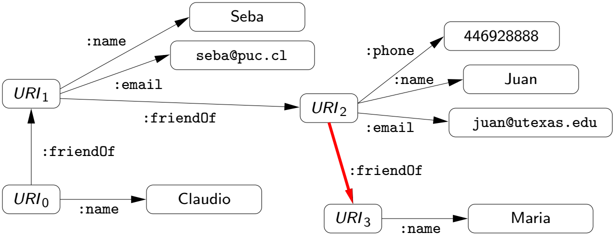

```
SELECT ?X
WHERE
{
   ?X  :friendOf   ?Y .
   ?Y  :name   "Maria" .
}
```

SPARQL 1.0 provides limited navigational capabilities



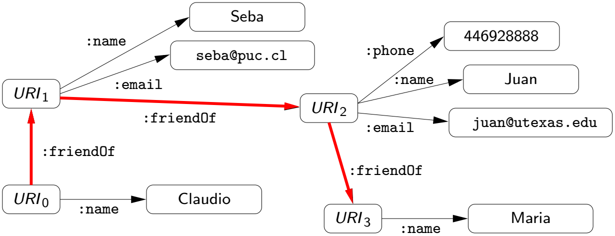

```
SELECT ?X
WHERE
{
   ?X  :friendOf   ?Y .
   ?Y  :name   "Maria" .
}
```

SPARQL 1.0 provides limited navigational capabilities



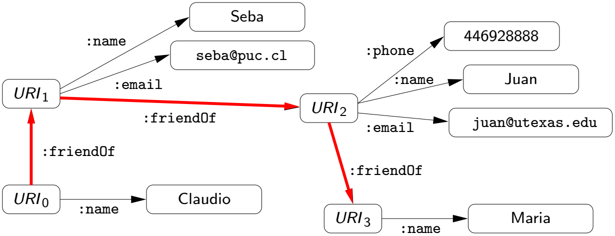

```
SELECT ?X
WHERE
{
   ?X (:friendOf)* ?Y .
   ?Y  :name   "Maria" .
}
```

SPARQL 1.0 provides limited navigational capabilities




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

SPARQL 1.0 provides limited navigational capabilities

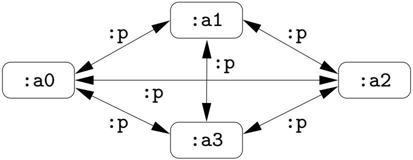



```
SELECT ?X
WHERE
{
   ?X (:friendOf)* ?Y .      ← SPARQL 1.1 property path
   ?Y  :name   "Maria" .
}
```

SPARQL 1.1 implementations had a poor performance

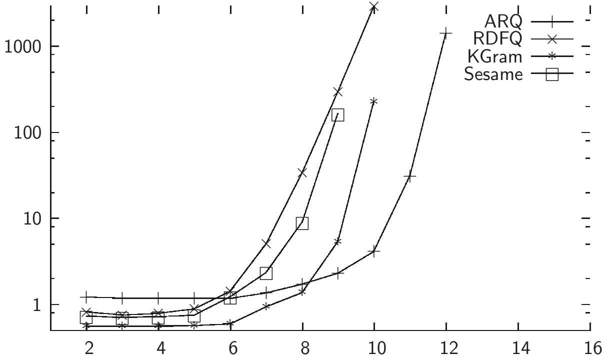
Data:

- *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

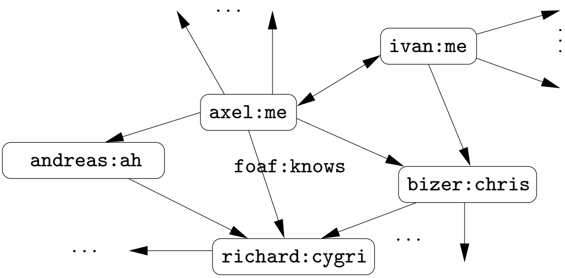



```
SELECT * WHERE { :a0 (:p)* :a1 }
```

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 }
```

Input	ARQ	RDFQ	Kgram	Sesame
9.2KB	5.13	75.70	313.37	–
10.9KB	8.20	325.83	–	–
11.4KB	65.87	–	–	–
13.2KB	292.43	–	–	–
14.8KB	–	–	–	–
17.2KB	–	–	–	–
20.5KB	–	–	–	–
25.8KB	–	–	–	–

(time in seconds, timeout = 1hr)

Poor performance with real Web data of small size

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

Input	ARQ	RDFQ	Kgram	Sesame
9.2KB	5.13	75.70	313.37	–
10.9KB	8.20	325.83	–	–
11.4KB	65.87	–	–	–
13.2KB	292.43	–	–	–
14.8KB	–	–	–	–
17.2KB	–	–	–	–
20.5KB	–	–	–	–
25.8KB	–	–	–	–

(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

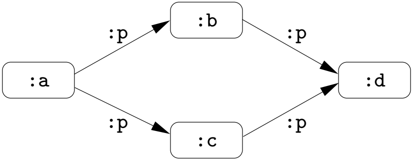
The main sources of complexity is *counting*

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*

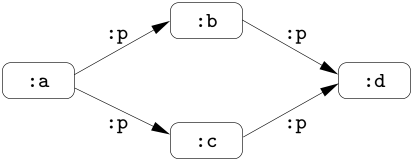
Property paths: regular expressions (/, |, ∗)




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

SPARQL 1.1 property paths match regular expressions but also *count*

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

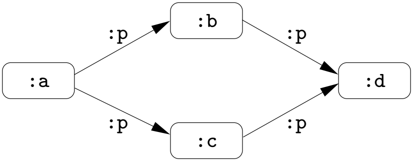



```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

?X
:a
:b
:c
:d
:d

SPARQL 1.1 property paths match regular expressions but also *count*

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

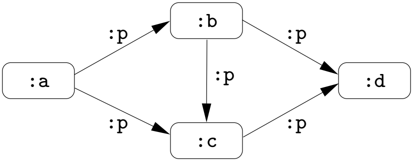



```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

?X
:a
:b
:c
:d
:d

SPARQL 1.1 property paths match regular expressions but also *count*

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

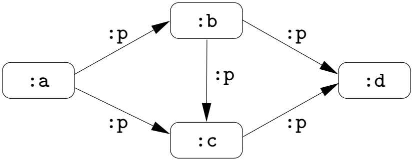



```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

$?X$
:a
:b
:c
:d
:d

SPARQL 1.1 property paths match regular expressions but also *count*

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

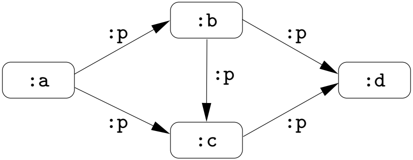



```
SELECT ?X
WHERE { :a  (:p)*  ?X }
```

?X
:a
:b
:c
:d
:d
:c
:d

SPARQL 1.1 property paths match regular expressions but also *count*

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




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

?X
:a
:b
:c
:d
:d
:c
:d

But what if we have cycles?

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

Evaluation of *path*∗

"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 *path*∗

"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)

- W3C document provides a procedure (`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* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size *n*

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

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size *n*

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

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size n

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

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

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

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size n

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

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

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

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

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

every solution is a copy of the *empty mapping* (| | in ARQ)

Counting the number of solutions...

Data: Clique of size n

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

n	# Sol.
9	13,700
10	109,601
11	986,410
12	9,864,101
13	–

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

n	# Sol
2	1
3	6
4	305
5	418,576
6	–

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

n	# Sol.
2	1
3	42
4	–

every solution is a copy of the *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

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

File	# URIs	# Sol.	Output Size
9.2KB	38	29,817	2MB
10.9KB	43	122,631	8.4MB
11.4KB	47	1,739,331	120MB
13.2KB	52	8,511,943	587MB
14.8KB	54	–	–

More on counting the number of solutions...

Data: foaf links crawled from the Web

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

File	# URIs	# Sol.	Output Size
9.2KB	38	29,817	2MB
10.9KB	43	122,631	8.4MB
11.4KB	47	1,739,331	120MB
13.2KB	52	8,511,943	587MB
14.8KB	54	–	–

What is really happening here?

More on counting the number of solutions...

Data: foaf links crawled from the Web

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

File	# URIs	# Sol.	Output Size
9.2KB	38	29,817	2MB
10.9KB	43	122,631	8.4MB
11.4KB	47	1,739,331	120MB
13.2KB	52	8,511,943	587MB
14.8KB	54	–	–

What is really happening here?

Theory can help!

A bit more on complexity classes...

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

NP	**#P**
SAT: is a propositional formula satisfiable?	COUNTSAT: how many assignments satisfy a propositional formula?

A bit more on complexity classes...

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

NP	**#P**
SAT: is a propositional formula satisfiable?	COUNTSAT: how many assignments satisfy a propositional formula?

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*

NP	**#P**
SAT: is a propositional formula satisfiable?	COUNTSAT: how many assignments satisfy a propositional formula?

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

CountW3C

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)

$\textsc{CountW3C}$ *is outside* **#P**

The complexity of property paths is *intractable*

Theorem (ACP12)

$\textsc{CountW3C}$ *is outside* **#P**

$\textsc{CountW3C}$ is hard to solve even if $\mathbf{P} = \mathbf{NP}$

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$(\cdots((\texttt{:p})\texttt{*})\texttt{*})\cdots)\texttt{*}$$

  with s nested stars

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$(\cdots((\texttt{:p})*)*)\cdots)*$$

  with s nested stars

- Let K_n be a clique with n nodes

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$\texttt{(}\cdots\texttt{((:p)*)*)}\cdots\texttt{)*}$$

  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

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$(\cdots((\texttt{:p})*)*)\cdots)*$$

  with s nested stars

- Let K_n be a clique with n nodes
- Let $CountClique(s, n)$ be the number of solutions of $\{\ \texttt{:a0}\ \ path_s\ \ \texttt{:a1}\ \}$ over K_n

Lemma (ACP12)

$$CountClique(s, n) \quad \geq \quad (n-2)!^{(n-1)^{s-1}}$$

A *doubly exponential* lower bound for counting

- Let $path_s$ be a property path of the form

$$\texttt{(}\cdots\texttt{((:p)*)*)}\cdots\texttt{)*}$$

  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)

$$CountClique(s, n) \;\; \geq \;\; (n-2)!^{(n-1)^{s-1}}$$

In [ACP12]: A recursive formula for calculating $CountClique(s, n)$

We can explain the experimental results

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

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

n	# Sol.
2	1
3	6
4	305
5	418,576
6	–
7	–
8	–

We can explain the experimental results

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

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

n	# Sol.	
2	1	✓
3	6	
4	305	
5	418,576	
6	–	
7	–	
8	–	

We can explain the experimental results

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

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

n	# Sol.	
2	1	✓
3	6	✓
4	305	
5	418,576	
6	–	
7	–	
8	–	

We can explain the experimental results

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

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

n	# Sol.	
2	1	✓
3	6	✓
4	305	✓
5	418,576	
6	–	
7	–	
8	–	

We can explain the experimental results

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

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

n	# Sol.	
2	1	✓
3	6	✓
4	305	✓
5	418,576	✓
6	–	
7	–	
8	–	

We can explain the experimental results

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

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

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	←	28×10^9
7	–		
8	–		

We can explain the experimental results

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

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

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	←	28×10^9
7	–	←	144×10^{15}
8	–		

We can explain the experimental results

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

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

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	$\leftarrow$	28×10^9
7	–	$\leftarrow$	144×10^{15}
8	–	$\leftarrow$	79×10^{24}

We can explain the experimental results

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

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

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	$\leftarrow$	28×10^{9}
7	–	$\leftarrow$	144×10^{15}
8	–	$\leftarrow$	79×10^{24}

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*

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

n	# Sol.		
2	1	✓	
3	6	✓	
4	305	✓	
5	418,576	✓	
6	–	←	28×10^9
7	–	←	144×10^{15}
8	–	←	79×10^{24}

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 $\textsc{CountW3C}$ 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 $\textsc{CountW3C}$ 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)

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

Can be proved by using automata theory

Evaluating existential paths is tractable

Theorem (well-known result)

ExistsPath can be solved in $O(|G| \times |path|)$

Can be proved by using automata theory

Theorem (ACP12)

ExistsPath and ExistsW3C are *equivalent* decision problems

Evaluating existential paths is tractable

Theorem (well-known result)

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

Can be proved by using automata theory

Theorem (ACP12)

$\textsc{ExistsPath}$ and $\textsc{ExistsW3C}$ are *equivalent* decision problems

Corollary (ACP12)

$\textsc{ExistsW3C}$ can be solved in $O(|G| \times |path|)$

So there are possibilities for optimization

So there are possibilities for optimization

Corollary

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

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 }
```

Input	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen

SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

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

Input	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen

SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

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

Input	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen
9.2KB	2.24	47.31	2.37	–	0.29	1.39
10.9KB	2.60	204.95	6.43	–	0.30	1.32
11.4KB	6.88	3222.47	80.73	–	0.30	1.34
13.2KB	24.42	–	394.61	–	0.31	1.38
14.8KB	–	–	–	–	0.33	1.38
17.2KB	–	–	–	–	0.35	1.42
20.5KB	–	–	–	–	0.44	1.50
25.8KB	–	–	–	–	0.45	1.52

SPARQL 1.1 implementations do not take advantage of SELECT DISTINCT

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

Input	ARQ	RDFQ	Kgram	Sesame	Psparql	Gleen
9.2KB	2.24	47.31	2.37	–	0.29	1.39
10.9KB	2.60	204.95	6.43	–	0.30	1.32
11.4KB	6.88	3222.47	80.73	–	0.30	1.34
13.2KB	24.42	–	394.61	–	0.31	1.38
14.8KB	–	–	–	–	0.33	1.38
17.2KB	–	–	–	–	0.35	1.42
20.5KB	–	–	–	–	0.44	1.50
25.8KB	–	–	–	–	0.45	1.52

Optimization possibilities can remain hidden
in a complicated specification

Theory can be of help in the design of new stadards

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 stadards

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 stadards

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

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