Foundations of RDF Databases

Marcelo Arenas[1], Claudio Gutierrez[2] and Jorge Pérez[1]

[1] Pontificia Universidad Católica de Chile
[2] Universidad de Chile

Semantic 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.]

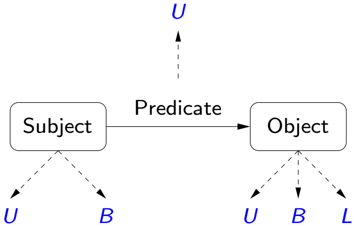
Specific Goals:

- Build a description language with standard semantics
- Make semantics machine-processable and understandable
- Incorporate logical infrastructure to reason about resources
- W3C Proposal: Resource Description Framework (RDF)

RDF in a nutshell

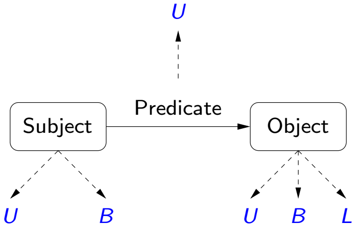
- RDF is the W3C proposal framework for representing information in the Web
- Abstract syntax based on directed labeled graph
- Schema definition language (RDFS): Define new vocabulary (typing, inheritance of classes and properties)
- Extensible URI-based vocabulary
- Formal semantics

RDF formal model



$$
\begin{aligned}
U &= \text{set of Uris} \\
B &= \text{set of Blank nodes} \\
L &= \text{set of Literals}
\end{aligned}
$$

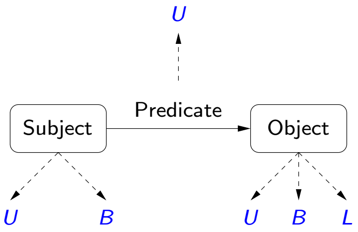
RDF formal model



$$U = \text{set of Uris}$$
$$B = \text{set of Blank nodes}$$
$$L = \text{set of Literals}$$

$(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple

RDF formal model



U = set of Uris

B = set of Blank nodes

L = set of Literals

$(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an RDF triple

A set of RDF triples is called an RDF graph

RDF formal model

Proviso

In this talk, we do distinguish between URIs and literals.

RDF formal model

Proviso

In this talk, we do distinguish between URIs and literals.

- $(s, p, o) \in (U \cup B) \times U \times (U \cup B)$ is called an RDF triple.

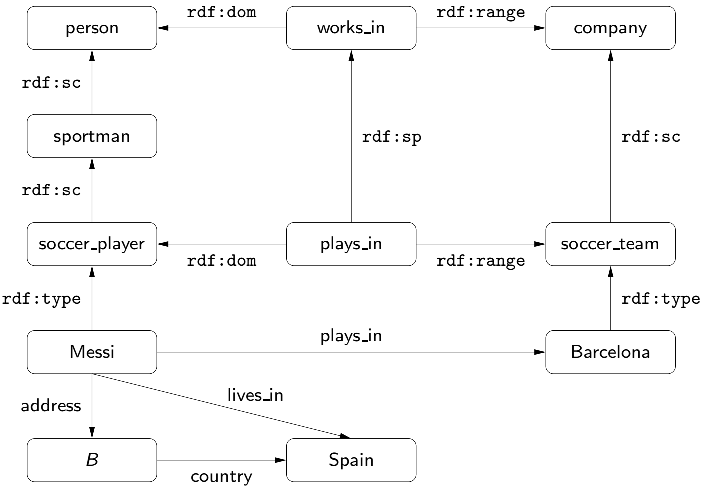
RDF formal model

Proviso

In this talk, we do distinguish between URIs and literals.

- $(s, p, o) \in (U \cup B) \times U \times (U \cup B)$ is called an RDF triple.
- The inclusion of L does not change any of the results presented in this talk.

RDF: An example



Why is RDF interesting from a database point of view?

Some new challenges:

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

Why is RDF interesting from a database point of view?

Some new challenges:

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

Why are database technologies interesting from an RDF point of view?

Why is RDF interesting from a database point of view?

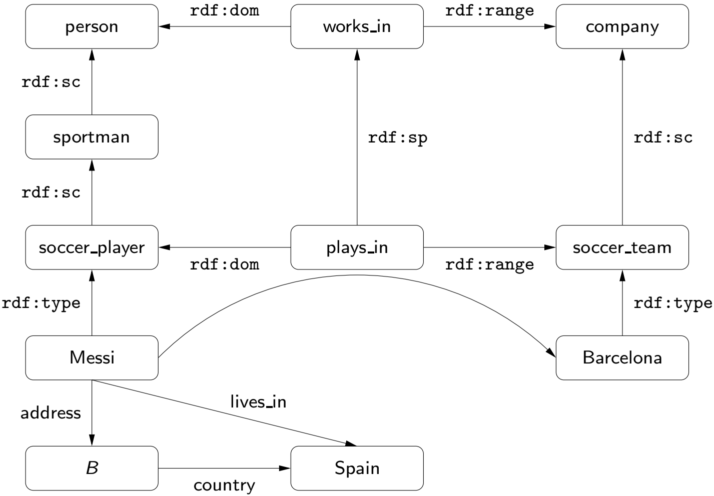
Some new challenges:

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

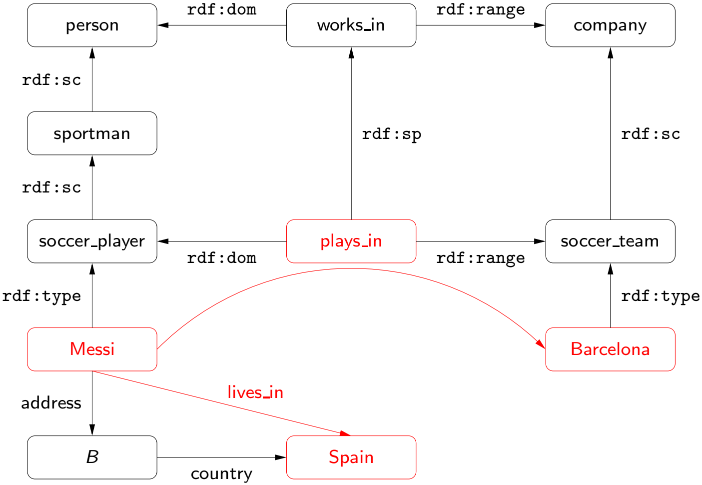
Why are database technologies interesting from an RDF point of view?

- RDF data processing can take advantage of database techniques: Query processing, storing, indexing, ...

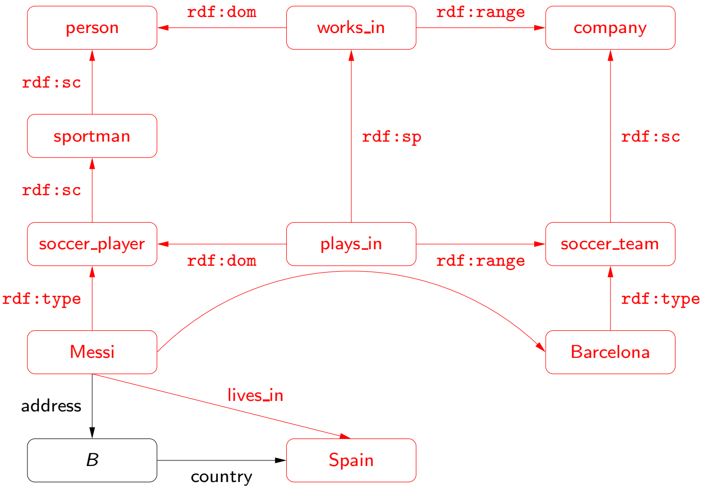
Previous example: A better representation



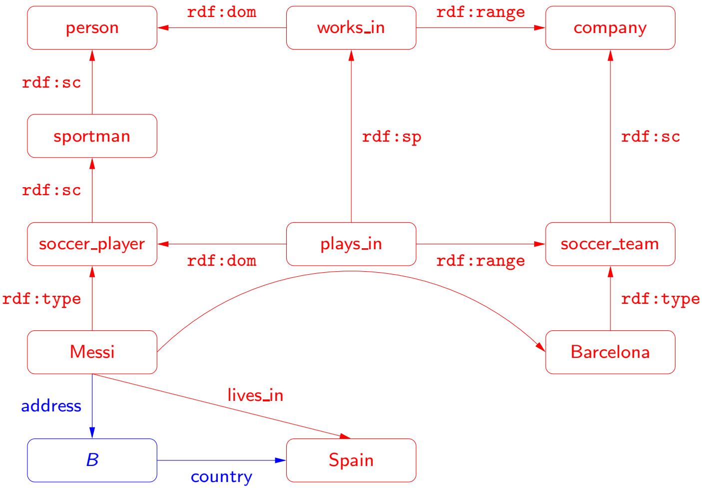
Previous example: A better representation



Previous example: A better representation



Previous example: A better representation



First part: Ground RDF without RDFS vocabulary

SPARQL: A query language for RDF

- Syntax and formal semantics
- Complexity of the evaluation problem
- Optimization methods
- Expressiveness

First part: Ground RDF without RDFS vocabulary

SPARQL: A query language for RDF

- Syntax and formal semantics
- Complexity of the evaluation problem
- Optimization methods
- Expressiveness

Querying RDF: SPARQL

- SPARQL is the W3C recommendation query language for RDF (January 2008).
  - SPARQL is a recursive acronym that stands for *SPARQL Protocol and RDF Query Language*.
- 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, ...

SPARQL in a nutshell

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

SPARQL in a nutshell

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

SPARQL in a nutshell

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

SPARQL in a nutshell

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

SPARQL in a nutshell

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

In general, in a query we have:

$$H \longleftarrow$$

- Head: processing of some variables.

SPARQL in a nutshell

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

In general, in a query we have:

$$H \leftarrow P$$

- Head: processing of some variables.
- Body: pattern matching expression.

SPARQL in a nutshell

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

In general, in a query we have:

$$H \longleftarrow P$$

- Head: processing of some variables.
- Body: pattern matching expression.

We focus on P.

But things can become more complex ...

Interesting features of pattern matching on graphs

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

```
{ P1
  P2 }
```

But things can become more complex ...

Interesting features of pattern matching on graphs

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

```
{ { P1
    P2 }


  { P3
    P4 }


}
```

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 }  }

}
```

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 }  }  }
}
```

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 }
```

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 study of SPARQL

Why is this needed?

- Clarifying corner cases
- Eliminating ambiguities
- Helping in the implementation process
  - Understanding the resources (time/space) needed to implement SPARQL
- Understanding what can/cannot be expressed
  - Discovering what needs to be added (aggregation, navigational capabilities, recursion, ...)

A standard algebraic syntax

- Triple patterns: just triples + variables, without blanks

original SPARQL syntax	algebraic syntax
`?X :name "john"`	$(?X, \text{name}, \text{john})$

- 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

- 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.

$$\mu \quad : \quad \text{Variables} \longrightarrow U$$

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

Mappings: building block for the semantics

Definition

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

$$\mu \quad : \quad \text{Variables} \longrightarrow U$$

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:

- has 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:

- has as domain the variables in t: $\mathrm{dom}(\mu) = \mathrm{var}(t)$
- makes 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:

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

Example

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

	$?X$	$?Y$
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:

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

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:

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

Example

graph	triple	evaluation
$(R_1, \text{name}, \text{john})$		
$(R_1, \text{email}, \text{J@ed.ex})$	$(?X, \text{name}, ?Y)$	
$(R_2, \text{name}, \text{paul})$		

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

Compatible mappings

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

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

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

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

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

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

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

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

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

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

Definition

Mappings μ_1 and μ_2 are compatible if they agree in their common variables:

$$\text{If } ?X \in \text{dom}(\mu_1) \cap \text{dom}(\mu_2), \text{ then } \mu_1(?X) = \mu_2(?X).$$

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 Ω_1 and Ω_2 be sets of mappings.

Definition

Sets of mappings and operations

Let Ω_1 and Ω_2 be sets of mappings.

Definition

Join: extends mappings in Ω_1 with compatible mappings in Ω_2

- $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1,\ \mu_2 \in \Omega_2$ and $\mu_1,\ \mu_2$ are compatible$\}$

Sets of mappings and operations

Let Ω_1 and Ω_2 be sets of mappings.

Definition

Join: extends mappings in Ω_1 with compatible mappings in Ω_2

- $\Omega_1 \bowtie \Omega_2 = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1,\ \mu_2 \in \Omega_2$ and $\mu_1,\ \mu_2$ are compatible$\}$

Difference: selects mappings in Ω_1 that cannot be extended with mappings in Ω_2

- $\Omega_1 \smallsetminus \Omega_2 = \{\mu_1 \in \Omega_1 \mid$ there is no mapping in Ω_2 compatible with $\mu_1\}$

Definition

Sets of mappings and operations

Definition

Union: includes mappings in Ω_1 and in Ω_2

- $\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}$

Sets of mappings and operations

Definition

Union: includes mappings in Ω_1 and in Ω_2

- $\Omega_1 \cup \Omega_2 = \{\mu \mid \mu \in \Omega_1 \text{ or } \mu \in \Omega_2\}$

Left Outer Join: extends mappings in Ω_1 with compatible mappings in Ω_2 **if possible**

- $\Omega_1 ⟕ \Omega_2 = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \smallsetminus \Omega_2)$

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{aligned}
[\![t]\!]_G &= \\
[\![P_1 \text{ AND } P_2]\!]_G &= \\
[\![P_1 \text{ UNION } P_2]\!]_G &= \\
[\![P_1 \text{ OPT } P_2]\!]_G &=
\end{aligned}
$$

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{aligned}
[\![t]\!]_G &= \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\} \\
[\![P_1 \text{ AND } P_2]\!]_G &= \\
[\![P_1 \text{ UNION } P_2]\!]_G &= \\
[\![P_1 \text{ OPT } P_2]\!]_G &=
\end{aligned}
$$

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{aligned}
[\![t]\!]_G &= \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\} \\
[\![P_1 \text{ AND } P_2]\!]_G &= [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \\
[\![P_1 \text{ UNION } P_2]\!]_G &= \\
[\![P_1 \text{ OPT } P_2]\!]_G &=
\end{aligned}
$$

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{aligned}
[\![t]\!]_G &= \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\} \\
[\![P_1 \text{ AND } P_2]\!]_G &= [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \\
[\![P_1 \text{ UNION } P_2]\!]_G &= [\![P_1]\!]_G \cup [\![P_2]\!]_G \\
[\![P_1 \text{ OPT } P_2]\!]_G &=
\end{aligned}
$$

Semantics of SPARQL

Given an RDF graph G.

Definition

$$
\begin{array}{lcl}
[\![t]\!]_G & = & \{\mu \mid \mathrm{dom}(\mu) = \mathrm{var}(t) \text{ and } \mu(t) \in G\} \\
[\![P_1 \text{ AND } P_2]\!]_G & = & [\![P_1]\!]_G \bowtie [\![P_2]\!]_G \\
[\![P_1 \text{ UNION } P_2]\!]_G & = & [\![P_1]\!]_G \cup [\![P_2]\!]_G \\
[\![P_1 \text{ OPT } P_2]\!]_G & = & [\![P_1]\!]_G ⟕ [\![P_2]\!]_G
\end{array}
$$

Semantics of SPARQL: An example

Example

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

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

Semantics of SPARQL: An example

Example

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

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

Semantics of SPARQL: An 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

Semantics of SPARQL: An 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

Semantics of SPARQL: An example

Example

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

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

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

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

Semantics of SPARQL: An 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

Semantics of SPARQL: An 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

Semantics of SPARQL: An 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

Semantics of SPARQL: An 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 Difference

Semantics of SPARQL: An example

Example

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

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

$?X$	$?Y$
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 Union

Filter expressions (value constraints)

Filter expression: P FILTER R

- P is a graph pattern
- R is a built-in condition

We consider in R:

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

We impose a safety condition: $\text{var}(R) \subseteq \text{var}(P)$

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X =?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
- R is $\text{bound}(?X)$ and $?X \in \text{dom}(\mu)$;

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
- R is bound$(?X)$ and $?X \in \text{dom}(\mu)$;
- R is $\neg R_1$ and $\mu \not\models R_1$;

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
- R is bound$(?X)$ and $?X \in \text{dom}(\mu)$;
- R is $\neg R_1$ and $\mu \not\models R_1$;
- R is $R_1 \vee R_2$, and $\mu \models R_1$ or $\mu \models R_2$;

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
- R is $\text{bound}(?X)$ and $?X \in \text{dom}(\mu)$;
- R is $\neg R_1$ and $\mu \not\models R_1$;
- R is $R_1 \vee R_2$, and $\mu \models R_1$ or $\mu \models R_2$;
- R is $R_1 \wedge R_2$, $\mu \models R_1$ and $\mu \models R_2$.

Satisfaction of value constraints

A mapping μ satisfies a condition R ($\mu \models R$) if:

- R is $?X = c$, $?X \in \text{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X, ?Y \in \text{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
- R is bound$(?X)$ and $?X \in \text{dom}(\mu)$;
- R is $\neg R_1$ and $\mu \not\models R_1$;
- R is $R_1 \vee R_2$, and $\mu \models R_1$ or $\mu \models R_2$;
- R is $R_1 \wedge R_2$, $\mu \models R_1$ and $\mu \models R_2$.

Definition

FILTER : selects mappings that satisfy a condition

$$[\![P \text{ FILTER } R]\!]_G \quad = \quad \{\mu \in [\![P]\!]_G \mid \mu \models R\}$$

First part: Ground RDF without RDFS vocabulary

SPARQL: A query language for RDF

- Syntax and formal semantics
- Complexity of the evaluation problem
- Optimization methods
- Expressiveness

The evaluation problem

Input:

A mapping μ, a graph pattern P, and an RDF graph G

Question:

Does μ belong to the evaluation of P over G?

$$\text{Does } \mu \in [\![P]\!]_G?$$

The evaluation problem

Input:

A mapping μ, a graph pattern P, and an RDF graph G

Question:

Does μ belong to the evaluation of P over G?

Does $\mu \in [\![P]\!]_G$?

We study the *combined complexity* of the evaluation problem.

- μ, P and G are part of the input.

Evaluation of simple patterns is polynomial

Theorem (PAG06)

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

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

Evaluation of simple patterns is polynomial

Theorem (PAG06)

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

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

Proof sketch

- 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)

The evaluation problem is NP-complete for AND-FILTER-UNION *expressions.*

Evaluation including UNION is NP-complete

Theorem (PAG06)

The evaluation problem is NP-complete for AND-FILTER-UNION *expressions.*

Proof sketch of hardness

- Reduction from 3SAT.
- $\neg$ bound is used to verify that a satisfying truth assignment is well defined.

Evaluation including UNION is NP-complete: Idea of the reduction

Let $\varphi = (q \vee r \vee \neg s) \wedge (\neg q \vee r \vee \neg t)$

We construct a mapping μ, a graph pattern P and an RDF graph G such that:

$$\varphi \text{ is satisfiable iff } \mu \in [\![P]\!]_G$$

G is defined as $\{(1, \mathtt{is}, \mathtt{true})\}$

P includes the variables $?Q$, $?\overline{Q}$, $?R$, $?\overline{R}$, $?S$, $?\overline{S}$, $?T$ and $?\overline{T}$.

- $\mu(?Q) = 1$ indicates that q is assigned value $\mathtt{true}$,
- $\mu(?\overline{Q}) = 1$ indicates that $\neg q$ is assigned value $\mathtt{true}$.

Evaluation including UNION is NP-complete: Idea of the reduction

Thus, the following pattern P_φ almost does the job.

$$\Big[(?Q, \texttt{is}, \texttt{true}) \text{ UNION } (?R, \texttt{is}, \texttt{true}) \text{ UNION } (?\overline{S}, \texttt{is}, \texttt{true})\Big] \text{ AND}$$
$$\Big[(?\overline{Q}, \texttt{is}, \texttt{true}) \text{ UNION } (?R, \texttt{is}, \texttt{true}) \text{ UNION } (?\overline{T}, \texttt{is}, \texttt{true})\Big]$$

Why does it fail?

Evaluation including UNION is NP-complete: Idea of the reduction

Thus, the following pattern P_φ almost does the job.

$$\Big[(?Q, \texttt{is}, \texttt{true}) \text{ UNION } (?R, \texttt{is}, \texttt{true}) \text{ UNION } (?\overline{S}, \texttt{is}, \texttt{true})\Big] \text{ AND }$$

$$\Big[(?\overline{Q}, \texttt{is}, \texttt{true}) \text{ UNION } (?R, \texttt{is}, \texttt{true}) \text{ UNION } (?\overline{T}, \texttt{is}, \texttt{true})\Big]$$

Why does it fail?

- It can assign value 1 to $?Q$ and $?\overline{Q}$.

Evaluation including UNION is NP-complete: Idea of the reduction

Thus, the following pattern P_φ almost does the job.

$$\Big[(?Q, \texttt{is}, \texttt{true}) \text{ UNION } (?R, \texttt{is}, \texttt{true}) \text{ UNION } (?\overline{S}, \texttt{is}, \texttt{true})\Big] \text{ AND}$$
$$\Big[(?\overline{Q}, \texttt{is}, \texttt{true}) \text{ UNION } (?R, \texttt{is}, \texttt{true}) \text{ UNION } (?\overline{T}, \texttt{is}, \texttt{true})\Big]$$

Why does it fail?

- It can assign value 1 to $?Q$ and $?\overline{Q}$.

Solution: consider the following condition R:

$$(\neg\,\text{bound}(?Q) \vee \neg\,\text{bound}(?\overline{Q})) \wedge (\neg\,\text{bound}(?R) \vee \neg\,\text{bound}(?\overline{R})) \wedge$$
$$(\neg\,\text{bound}(?S) \vee \neg\,\text{bound}(?\overline{S})) \wedge (\neg\,\text{bound}(?T) \vee \neg\,\text{bound}(?\overline{T}))$$

Evaluation including UNION is NP-complete: Idea of the reduction

Then (P_φ FILTER R) does the job.

Evaluation including UNION is NP-complete: Idea of the reduction

Then (P_φ FILTER R) does the job.

- But how do we define μ?

Evaluation including UNION is NP-complete: Idea of the reduction

Then (P_φ FILTER R) does the job.

- But how do we define μ?

Final step: define P as:

$$\Big[(?Q, \mathtt{is}, \mathtt{true}) \text{ AND } (?\overline{Q}, \mathtt{is}, \mathtt{true}) \text{ AND } (?R, \mathtt{is}, \mathtt{true}) \text{ AND }$$
$$(?\overline{R}, \mathtt{is}, \mathtt{true}) \text{ AND } (?S, \mathtt{is}, \mathtt{true}) \text{ AND } (?\overline{S}, \mathtt{is}, \mathtt{true}) \text{ AND }$$
$$(?T, \mathtt{is}, \mathtt{true}) \text{ AND } (?\overline{T}, \mathtt{is}, \mathtt{true})\Big] \text{ AND } \Big[P_\varphi \text{ FILTER } R\Big]$$

and μ as:

$?Q$	$?\overline{Q}$	$?R$	$?\overline{R}$	$?S$	$?\overline{S}$	$?T$	$?\overline{T}$
1	1	1	1	1	1	1	1

In general: Evaluation problem is PSPACE-complete

Theorem (PAG06)

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

In general: Evaluation problem is PSPACE-complete

Theorem (PAG06)

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

Can we evaluate SPARQL queries in practice efficiently?

In general: Evaluation problem is PSPACE-complete

Theorem (PAG06)

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

Can we evaluate SPARQL queries in practice efficiently?

- We need to understand how the complexity depends on the operators of SPARQL.

A simple normal from

Proposition (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.*

A simple normal from

Proposition (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.*

Graph pattern expressions are usually in this normal form.

A simple normal from

Proposition (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.*

Graph pattern expressions are usually in this normal form.

Corollary

The evaluation problem is polynomial for AND-FILTER-UNION *expressions in the UNION normal form.*

PSPACE-completeness: A stronger lower bound

Theorem (PAG06)

The evaluation problem remains PSPACE-complete for AND-FILTER-OPT *expressions.*

PSPACE-completeness: A stronger lower bound

Theorem (PAG06)

The evaluation problem remains PSPACE-complete for AND-FILTER-OPT *expressions.*

Proof sketch of hardness

- Reduction from QBF: A pattern encodes a quantified propositional formula

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \psi.$$

PSPACE-completeness: A stronger lower bound

Theorem (PAG06)

The evaluation problem remains PSPACE-complete for AND-FILTER-OPT *expressions.*

Proof sketch of hardness

- Reduction from QBF: A pattern encodes a quantified propositional formula

$$\forall x_1 \exists y_1 \forall x_2 \exists y_2 \cdots \psi.$$

- Nested OPTs are used to encode quantifier alternation.

PSPACE-hardness: A closer look

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

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

G :

R_ψ :

P_ψ :

P_φ :

μ_0 :

PSPACE-hardness: A closer look

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

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

G : $\{(a, \mathtt{tv}, 0),\ (a, \mathtt{tv}, 1),\ (a, \mathtt{false}, 0),\ (a, \mathtt{true}, 1)\}$

R_ψ :

P_ψ :

P_φ :

μ_0 :

PSPACE-hardness: A closer look

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

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

G : $\{(a, \mathtt{tv}, 0),\ (a, \mathtt{tv}, 1),\ (a, \mathtt{false}, 0),\ (a, \mathtt{true}, 1)\}$

R_ψ : $((?X_1 = 1 \vee ?Y_1 = 0)\ \wedge\ (?X_1 = 0 \vee ?Y_1 = 1))$

P_ψ :

P_φ :

μ_0 :

PSPACE-hardness: A closer look

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

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

G : $\{(a, \mathtt{tv}, 0),\ (a, \mathtt{tv}, 1),\ (a, \mathtt{false}, 0),\ (a, \mathtt{true}, 1)\}$

R_ψ : $((?X_1 = 1 \vee ?Y_1 = 0)\ \wedge\ (?X_1 = 0 \vee ?Y_1 = 1))$

P_ψ : $(((a, \mathtt{tv}, ?X_1)$ AND $(a, \mathtt{tv}, ?Y_1))$ FILTER $R_\psi)$

P_φ :

μ_0 :

PSPACE-hardness: A closer look

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

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

$$G \quad : \quad \{(a, \texttt{tv}, 0),\ (a, \texttt{tv}, 1),\ (a, \texttt{false}, 0),\ (a, \texttt{true}, 1)\}$$

$$R_\psi \quad : \quad ((?X_1 = 1 \vee ?Y_1 = 0)\ \wedge\ (?X_1 = 0 \vee ?Y_1 = 1))$$

$$P_\psi \quad : \quad (((a, \texttt{tv}, ?X_1)\ \text{AND}\ (a, \texttt{tv}, ?Y_1))\ \text{FILTER}\ R_\psi)$$

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

$$\mu_0 \quad :$$

PSPACE-hardness: A closer look

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

We generate G, P_φ and μ_0 such that μ_0 belongs to the answer of P_φ over G iff φ is valid:

$$G \quad : \quad \{(a, \mathtt{tv}, 0),\ (a, \mathtt{tv}, 1),\ (a, \mathtt{false}, 0),\ (a, \mathtt{true}, 1)\}$$

$$R_\psi \quad : \quad ((?X_1 = 1 \vee ?Y_1 = 0)\ \wedge\ (?X_1 = 0 \vee ?Y_1 = 1))$$

$$P_\psi \quad : \quad (((a, \mathtt{tv}, ?X_1)\ \text{AND}\ (a, \mathtt{tv}, ?Y_1))\ \text{FILTER}\ R_\psi)$$

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

$$\mu_0 \quad : \quad \{?B_0 \mapsto 1\}$$

PSPACE-hardness: A closer look

$$\begin{array}{lcl}
\varphi & : & \forall x_1 \exists y_1 \, (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \\
P_\psi & : & (((a, \texttt{tv}, ?X_1) \ \text{AND} \ (a, \texttt{tv}, ?Y_1)) \ \text{FILTER} \\
 & & \ ((?X_1 = 1 \vee ?Y_1 = 0) \ \wedge \ (?X_1 = 0 \vee ?Y_1 = 1))) \\
P_\varphi & : & (a, \texttt{true}, ?B_0) \ \text{OPT} \ (P_1 \ \text{OPT} \ (Q_1 \ \text{AND} \ P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \ \text{AND} \ (a, \texttt{tv}, ?Y_1) \ \text{AND} \ (a, \texttt{false}, ?B_0)
\end{array}$$

PSPACE-hardness: A closer look

$$\begin{array}{lcl}
\varphi & : & \forall x_1 \exists y_1 \, (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \\
P_\psi & : & (((a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1)) \text{ FILTER} \\
 & & \quad ((?X_1 = 1 \vee ?Y_1 = 0) \ \wedge \ (?X_1 = 0 \vee ?Y_1 = 1))) \\
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}$$

$?B_0 \mapsto 1$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
\varphi & : & \forall x_1 \exists y_1\, (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \\
P_\psi & : & (((a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1)) \text{ FILTER} \\
& & \;((?X_1 = 1 \vee ?Y_1 = 0) \;\wedge\; (?X_1 = 0 \vee ?Y_1 = 1))) \\
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}
$$

P_1

$?X_1 \mapsto 0$

$?B_0 \mapsto 1$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
\varphi & : & \forall x_1 \exists y_1\, (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \\
P_\psi & : & (((a, \mathtt{tv}, ?X_1) \text{ AND } (a, \mathtt{tv}, ?Y_1)) \text{ FILTER} \\
 & & \;((?X_1 = 1 \vee ?Y_1 = 0) \;\wedge\; (?X_1 = 0 \vee ?Y_1 = 1))) \\
P_\varphi & : & (a, \mathtt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \mathtt{tv}, ?X_1) \\
Q_1 & : & (a, \mathtt{tv}, ?X_1) \text{ AND } (a, \mathtt{tv}, ?Y_1) \text{ AND } (a, \mathtt{false}, ?B_0)
\end{array}
$$

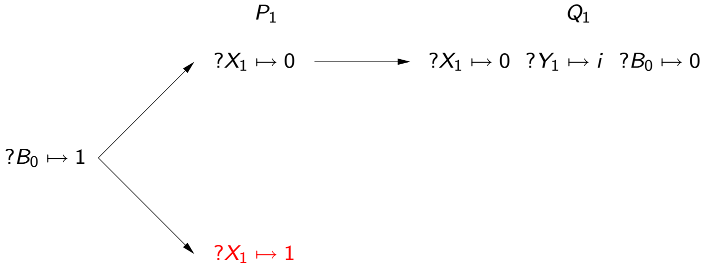
P_1 $\qquad$ Q_1

$?X_1 \mapsto 0 \longrightarrow ?X_1 \mapsto 0 \;\; ?Y_1 \mapsto i \;\; ?B_0 \mapsto 0$

$?B_0 \mapsto 1$

PSPACE-hardness: A closer look

$$
\begin{array}{lcl}
\varphi & : & \forall x_1 \exists y_1\, (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \\
P_\psi & : & (((a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1)) \text{ FILTER} \\
 & & \;((?X_1 = 1 \vee ?Y_1 = 0) \;\wedge\; (?X_1 = 0 \vee ?Y_1 = 1))) \\
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}
$$



$$\begin{array}{lcl}
\varphi & : & \forall x_1 \exists y_1 \, (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1) \\
P_\psi & : & (((a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1)) \text{ FILTER} \\
 & & \;((?X_1 = 1 \vee ?Y_1 = 0) \;\wedge\; (?X_1 = 0 \vee ?Y_1 = 1))) \\
P_\varphi & : & (a, \texttt{true}, ?B_0) \text{ OPT } (P_1 \text{ OPT } (Q_1 \text{ AND } P_\psi)) \\
P_1 & : & (a, \texttt{tv}, ?X_1) \\
Q_1 & : & (a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1) \text{ AND } (a, \texttt{false}, ?B_0)
\end{array}$$

$$\begin{tikzcd}
 & P_1 & & Q_1 \\
 & ?X_1 \mapsto 0 \arrow[rr] & & ?X_1 \mapsto 0 \quad ?Y_1 \mapsto i \quad ?B_0 \mapsto 0 \\
?B_0 \mapsto 1 \arrow[ur] \arrow[dr] & & & \\
 & \color{red}{?X_1 \mapsto 1} \arrow[rr] & & \color{red}{?X_1 \mapsto 1 \quad ?Y_1 \mapsto j \quad ?B_0 \mapsto 0}
\end{tikzcd}$$

What is the source of the high complexity?

Theorem (SML08)

The evaluation problem remains PSPACE-complete for OPT expressions

The use of the OPT operator makes the evaluation problem harder.

- How can we deal with this operator? How can we reduce the complexity?

What is the source of the high complexity?

Theorem (SML08)

The evaluation problem remains PSPACE-complete for OPT expressions

The use of the OPT operator makes the evaluation problem harder.

- How can we deal with this operator? How can we reduce the complexity?
- The formal study has some interesting practical implications.

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in our reduction (and in [SML08]) are not very natural:

$$(a, \texttt{true}, ?B_0) \quad \text{OPT} \quad (P_1 \quad \text{OPT} \quad (Q_1 \quad \text{AND} \quad P_\psi))$$

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in our reduction (and in [SML08]) are not very natural:

$$(a, \texttt{true}, ?B_0) \quad \text{OPT} \quad (P_1 \quad \text{OPT} \quad (\underset{\substack{\uparrow \\ ?B_0}}{Q_1} \quad \text{AND} \quad P_\psi))$$

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in our reduction (and in [SML08]) are not very natural:

$$(a, \texttt{true}, ?B_0) \quad \text{OPT} \quad (P_1 \quad \text{OPT} \quad (Q_1 \quad \text{AND} \quad P_\psi))$$

$?B_0$ $?B_0$

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in our reduction (and in [SML08]) are not very natural:

$$\underset{?B_0}{(a, \texttt{true}, ?B_0)} \quad \text{OPT} \quad (\underset{\times}{P_1} \quad \text{OPT} \quad (\underset{?B_0}{Q_1} \quad \text{AND} \quad P_\psi))$$

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in our reduction (and in [SML08]) are not very natural:

$$(a, \underset{\substack{\uparrow \\ ?B_0}}{\texttt{true}}, ?B_0) \quad \text{OPT} \quad (\underset{\substack{\uparrow \\ \times}}{P_1} \quad \text{OPT} \quad (\underset{\substack{\uparrow \\ ?B_0}}{Q_1} \quad \text{AND} \quad P_\psi))$$

Is $?B_0$ giving optional information for P_1?

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in our reduction (and in [SML08]) are not very natural:

$$\underset{\substack{\uparrow\\ ?B_0}}{(a, \texttt{true}, ?B_0)} \quad \text{OPT} \quad (\underset{\substack{\uparrow\\ \times}}{P_1} \quad \text{OPT} \quad (\underset{\substack{\uparrow\\ ?B_0}}{Q_1} \quad \text{AND} \quad P_\psi))$$

Is $?B_0$ giving optional information for P_1?

- No, $?B_0$ is giving optional information for $(a, \texttt{true}, ?B_0)$?

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in our reduction (and in [SML08]) are not very natural:

$$\underset{\textcolor{red}{?B_0}}{(a, \texttt{true}, ?B_0)} \quad \text{OPT} \quad (\underset{\textcolor{red}{\times}}{P_1} \quad \text{OPT} \quad (\underset{\textcolor{red}{?B_0}}{Q_1} \quad \text{AND} \quad P_\psi))$$

Is $?B_0$ giving optional information for P_1?

- No, $?B_0$ is giving optional information for $(a, \texttt{true}, ?B_0)$?

These patterns rarely occur in practice.

Well–designed patterns

Definition

An AND-FILTER-OPT pattern is well–designed if 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 inside B and anywhere outside the OPT operator, then the variable must also occur inside A.

Well–designed patterns

Definition

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

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

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

Well–designed patterns

Definition

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

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

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

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

Well–designed patterns

Definition

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

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

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

Well–designed patterns

Definition

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

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

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

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

Example

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

Well–designed patterns

Definition

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

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

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

Example

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

Well–designed patterns

Definition

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

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

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

Example

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

Well–designed patterns

Definition

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

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

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

Example

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

AND-FILTER-OPT fragment: Reducing the complexity

Theorem (PAG09)

The evaluation problem is coNP-complete for well-designed AND-FILTER-OPT *patterns.*

AND-FILTER-OPT fragment: Reducing the complexity

Theorem (PAG09)

The evaluation problem is coNP-complete for well-designed AND-FILTER-OPT *patterns.*

Proof sketch of membership

First step: Prove that if P' is obtained by removing some optional parts of P, then P' cannot be more informative than P.

AND-FILTER-OPT fragment: Reducing the complexity

Theorem (PAG09)

The evaluation problem is coNP-complete for well-designed AND-FILTER-OPT *patterns.*

Proof sketch of membership

First step: Prove that if P' is obtained by removing some optional parts of P, then P' cannot be more informative than P.

- This holds for well-designed patterns.

AND-FILTER-OPT fragment: Reducing the complexity

Theorem (PAG09)

The evaluation problem is coNP-complete for well-designed AND-FILTER-OPT *patterns.*

Proof sketch of membership

First step: Prove that if P' is obtained by removing some optional parts of P, then P' cannot be more informative than P.

- This holds for well-designed patterns.
- This does not hold in general: $G = \{(1, \mathtt{a}, \mathtt{b}), (2, \mathtt{c}, \mathtt{d})\}$ and

$$(?X, \mathtt{a}, \mathtt{b}) \text{ OPT } ((?Y, \mathtt{c}, \mathtt{d}) \text{ OPT } (?X, \mathtt{c}, \mathtt{d}))$$

AND-FILTER-OPT fragment: Reducing the complexity

Notation:

AND-FILTER-OPT fragment: Reducing the complexity

Notation:

- $\mu \sqsubseteq \mu'$: μ and μ' are compatible and $\text{dom}(\mu) \subseteq \text{dom}(\mu')$

AND-FILTER-OPT fragment: Reducing the complexity

Notation:

- $\mu \sqsubseteq \mu'$: μ and μ' are compatible and $\text{dom}(\mu) \subseteq \text{dom}(\mu')$
- $\Omega \sqsubseteq \Omega'$: for every $\mu \in \Omega$, there exists $\mu' \in \Omega'$ s.t. $\mu \sqsubseteq \mu'$

AND-FILTER-OPT fragment: Reducing the complexity

Notation:

- $\mu \sqsubseteq \mu'$: μ and μ' are compatible and $\text{dom}(\mu) \subseteq \text{dom}(\mu')$
- $\Omega \sqsubseteq \Omega'$: for every $\mu \in \Omega$, there exists $\mu' \in \Omega'$ s.t. $\mu \sqsubseteq \mu'$
- P' is a reduction of P: P' can be obtained from P by replacing a sub-formula $(P_1 \text{ OPT } P_2)$ of P by P_1

AND-FILTER-OPT fragment: Reducing the complexity

Notation:

- $\mu \sqsubseteq \mu'$: μ and μ' are compatible and $\text{dom}(\mu) \subseteq \text{dom}(\mu')$
- $\Omega \sqsubseteq \Omega'$: for every $\mu \in \Omega$, there exists $\mu' \in \Omega'$ s.t. $\mu \sqsubseteq \mu'$
- P' is a reduction of P: P' can be obtained from P by replacing a sub-formula (P_1 OPT P_2) of P by P_1

$$\begin{aligned} P &= (t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)) \\ P' &= t_1 \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)) \end{aligned}$$

AND-FILTER-OPT fragment: Reducing the complexity

- $\unlhd$: Reflexive and transitive closure of the reduction relation

AND-FILTER-OPT fragment: Reducing the complexity

- $\trianglelefteq$: Reflexive and transitive closure of the reduction relation

$$
\begin{aligned}
P &= (t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)) \\
P' &= t_1 \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)) \\
P'' &= t_1 \text{ AND } t_2
\end{aligned}
$$

AND-FILTER-OPT fragment: Reducing the complexity

- $\trianglelefteq$: Reflexive and transitive closure of the reduction relation

$$
\begin{aligned}
P &= (t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)) \\
P' &= t_1 \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)) \\
P'' &= t_1 \text{ AND } t_2
\end{aligned}
$$

Proposition

If P is a UNION*-free well-designed graph pattern and $P' \trianglelefteq P$, then $[\![P']\!]_G \sqsubseteq [\![P]\!]_G$ for every graph G.*

AND-FILTER-OPT fragment: Reducing the complexity

Second step: Prove that the "compatible" information is not lost by an OPT operator

- Again this holds for well-designed patterns.

AND-FILTER-OPT fragment: Reducing the complexity

Second step: Prove that the "compatible" information is not lost by an OPT operator

- Again this holds for well-designed patterns.

More notation:

AND-FILTER-OPT fragment: Reducing the complexity

Second step: Prove that the "compatible" information is not lost by an OPT operator

- Again this holds for well-designed patterns.

More notation:

- and(P): replace OPT by AND in P

AND-FILTER-OPT fragment: Reducing the complexity

Second step: Prove that the "compatible" information is not lost by an OPT operator

- Again this holds for well-designed patterns.

More notation:

- and(P): replace OPT by AND in P

$$\begin{array}{rcl} P & = & (t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)) \\ \text{and}(P) & = & (t_1 \text{ AND } t_2) \text{ AND } (t_2 \text{ AND } (t_3 \text{ AND } t_4)) \end{array}$$

AND-FILTER-OPT fragment: Reducing the complexity

Second step: Prove that the "compatible" information is not lost by an OPT operator

- Again this holds for well-designed patterns.

More notation:

- and(P): replace OPT by AND in P

$$\begin{array}{rcl} P & = & (t_1 \text{ OPT } t_2) \text{ AND } (t_2 \text{ OPT } (t_3 \text{ AND } t_4)) \\ \text{and}(P) & = & (t_1 \text{ AND } t_2) \text{ AND } (t_2 \text{ AND } (t_3 \text{ AND } t_4)) \end{array}$$

- μ is a partial solution for P over G: there exists $P' \trianglelefteq P$ s.t. $\mu \in [\![\text{and}(P')]\!]_G$.

AND-FILTER-OPT fragment: Reducing the complexity

Let P be a UNION-free well-designed graph pattern and G an RDF graph.

Proposition

$\mu \in [\![P]\!]_G$ if and only if μ is a maximal (w.r.t. $\sqsubseteq$) partial solution for P over G.

AND-FILTER-OPT fragment: Reducing the complexity

Let P be a UNION-free well-designed graph pattern and G an RDF graph.

Proposition

$\mu \in [\![P]\!]_G$ if and only if μ is a maximal (w.r.t. $\sqsubseteq$) partial solution for P over G.

Third step: Show that it can be decided in polynomial time whether μ is a partial solution for P over G

AND-FILTER-OPT fragment: Reducing the complexity

Last step: Combine all the previous results

To verify whether $\mu \notin [\![P]\!]_G$:

(1) Check whether μ is not a partial solution for P over G.
   - If this is the case, then return **true**, else go to (2).

(2) Guess a mapping μ' such that $\mu \sqsubseteq \mu'$ and $\mu' \not\sqsubseteq \mu$.
   (2.1) If μ' is a partial solution for P over G, then return **true**.

A corollary of the previous result

Corollary

The evaluation problem is coNP-complete for patterns of the form P_1 UNION P_2 UNION $\cdots$ UNION P_k, *where each* P_i *is a well-designed* AND-FILTER-OPT *pattern.*

A corollary of the previous result

Corollary

The evaluation problem is coNP-complete for patterns of the form P_1 UNION P_2 UNION $\cdots$ UNION P_k*, where each* P_i *is a well-designed* AND-FILTER-OPT *pattern.*

Can we use this in practice?

- Well-designed patterns are suitable for optimization.

SPARQL: A query language for RDF

- Syntax and formal semantics
- Complexity of the evaluation problem
- Optimization methods
- Expressiveness

Classical optimization

- Classical optimization assumes null-rejection.

Classical optimization

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

Classical optimization

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

- SPARQL operations are not null-rejecting.
  - By definition of compatible mappings.

Classical optimization

- Classical optimization assumes null-rejection.
  - Null-rejection: the join/outer–join condition must fail in the presence of nulls.
- SPARQL operations are not null-rejecting.
  - By definition of compatible mappings.
- Can we use classical optimization in the context of SPARQL?

Classical optimization

- Classical optimization assumes null-rejection.
  - Null-rejection: the join/outer–join condition must fail in the presence of nulls.
- SPARQL operations are not 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:

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

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

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

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 (PAG06)

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

Well–designed graph patterns and optimization

Previous theorem suggests a strategy for evaluating a well-designed pattern P.

- Transform P into an equivalent pattern Q in OPT normal form, and then evaluate Q.

Well–designed graph patterns and optimization

Previous theorem suggests a strategy for evaluating a well-designed pattern P.

- Transform P into an equivalent pattern Q in OPT normal form, and then evaluate Q.

Why this could be a good approach?

Well–designed graph patterns and optimization

Previous theorem suggests a strategy for evaluating a well-designed pattern P.

- Transform P into an equivalent pattern Q in OPT normal form, and then evaluate Q.

Why this could be a good approach?

- FILTER should be applied as soon as possible.

Well–designed graph patterns and optimization

Previous theorem suggests a strategy for evaluating a well-designed pattern P.

- Transform P into an equivalent pattern Q in OPT normal form, and then evaluate Q.

Why this could be a good approach?

- FILTER should be applied as soon as possible.
- AND is better as a filter than OPT:

$$[\![P_1 \text{ AND } P_2]\!]_G \quad \subseteq \quad [\![P_1 \text{ OPT } P_2]\!]_G.$$

Well–designed graph patterns and optimization

Previous theorem suggests a strategy for evaluating a well-designed pattern P.

- Transform P into an equivalent pattern Q in OPT normal form, and then evaluate Q.

Why this could be a good approach?

- FILTER should be applied as soon as possible.
- AND is better as a filter than OPT:

$$[\![P_1 \text{ AND } P_2]\!]_G \quad \subseteq \quad [\![P_1 \text{ OPT } P_2]\!]_G.$$

An experimental evaluation is needed.

- The final strategy will probably have to consider alternative re-orderings (not always the OPT normal form).

First part: Ground RDF without RDFS vocabulary

SPARQL: A query language for RDF

- Syntax and formal semantics
- Complexity of the evaluation problem
- Optimization methods
- Expressiveness

Is SPARQL expressive enough?

How do we study the expressive power of a language?

Is SPARQL expressive enough?

How do we study the expressive power of a language?

- How do we prove that a language has a good expressive power?

Is SPARQL expressive enough?

How do we study the expressive power of a language?

- How do we prove that a language has a good expressive power?

One alternative: Compare it with a well-known language

Is SPARQL expressive enough?

How do we study the expressive power of a language?

- How do we prove that a language has a good expressive power?

One alternative: Compare it with a well-known language

- Relational Algebra is a very good alternative

Is SPARQL expressive enough?

How do we study the expressive power of a language?

- How do we prove that a language has a good expressive power?

One alternative: Compare it with a well-known language

- Relational Algebra is a very good alternative
- We show that SPARQL and Relational Algebra have the same expressive power

But not so fast ...

We first have to say over which class of structure we compare Relational Algebra with SPARQL.

But not so fast ...

We first have to say over which class of structure we compare Relational Algebra with SPARQL.

- Conditional XPath is the same as FO

But not so fast ...

We first have to say over which class of structure we compare Relational Algebra with SPARQL.

- Conditional XPath is the same as FO over trees [M04]

But not so fast ...

We first have to say over which class of structure we compare Relational Algebra with SPARQL.

- Conditional XPath is the same as FO over trees [M04]

We use the following relational schema:

But not so fast ...

We first have to say over which class of structure we compare Relational Algebra with SPARQL.

- Conditional XPath is the same as FO over trees [M04]

We use the following relational schema:

- $triple(\cdot, \cdot, \cdot)$

But not so fast ...

We first have to say over which class of structure we compare Relational Algebra with SPARQL.

- Conditional XPath is the same as FO over trees [M04]

We use the following relational schema:

- *triple*$(\cdot, \cdot, \cdot)$
- **N**$(\cdot)$: It only contains value *null*

But not so fast ...

We first have to say over which class of structure we compare Relational Algebra with SPARQL.

- Conditional XPath is the same as FO over trees [M04]

We use the following relational schema:

- *triple*$(\cdot, \cdot, \cdot)$
- $\mathbf{N}(\cdot)$: It only contains value *null*

Every RDF graph G can be naturally translated into an instance I_G over this schema.

But not so fast ...

We use a language that has the same expressive power as Relational Algebra: nr-Datalog$^\neg$

But not so fast ...

We use a language that has the same expressive power as Relational Algebra: nr-Datalog$^{\neg}$

$$\begin{array}{rcl} \textsc{Answer}(X) & \leftarrow & Q(X,Y), Y = a, \neg R(Y,Y) \\ R(U,V) & \leftarrow & Q(U,Z), Q(Z,V) \end{array}$$

But not so fast ...

We use a language that has the same expressive power as Relational Algebra: nr-Datalog$^\neg$

$$\begin{array}{rcl} \textsc{Answer}(X) & \leftarrow & Q(X,Y), Y = a, \neg R(Y,Y) \\ R(U,V) & \leftarrow & Q(U,Z), Q(Z,V) \end{array}$$

Last point: It is easy to prove that

$$\textsc{Answer}(X) \quad \leftarrow \quad \mathit{triple}(X,Y,Z)$$

is not equivalent to any SPARQL graph pattern.

But not so fast ...

We use a language that has the same expressive power as Relational Algebra: nr-Datalog$^\neg$

$$
\begin{aligned}
\textsc{Answer}(X) &\leftarrow Q(X,Y), Y = a, \neg R(Y,Y) \\
R(U,V) &\leftarrow Q(U,Z), Q(Z,V)
\end{aligned}
$$

Last point: It is easy to prove that

$$\textsc{Answer}(X) \leftarrow \mathit{triple}(X,Y,Z)$$

is not equivalent to any SPARQL graph pattern.

- We need to consider the SELECT operator

SPARQL SELECT language

A SPARQL SELECT query is a tuple (W, P):

- P is an SPARQL graph pattern
- W is subset of $\text{var}(P)$

SPARQL SELECT language

A SPARQL SELECT query is a tuple (W, P):

- P is an SPARQL graph pattern
- W is subset of $\text{var}(P)$

Notation: $\mu_{|W}$ is the restriction of μ to W

- $\text{dom}(\mu_{|W}) = \text{dom}(\mu) \cap W$, and $\mu_{|W}(?X) = \mu(?X)$ for every $?X \in \text{dom}(\mu) \cap W$

SPARQL SELECT language

A SPARQL SELECT query is a tuple (W, P):

- P is an SPARQL graph pattern
- W is subset of $\text{var}(P)$

Notation: $\mu_{|W}$ is the restriction of μ to W

- $\text{dom}(\mu_{|W}) = \text{dom}(\mu) \cap W$, and $\mu_{|W}(?X) = \mu(?X)$ for every $?X \in \text{dom}(\mu) \cap W$

Definition

Given an RDF graph G:

$$[\![(W, P)]\!]_G \quad = \quad \{\mu_{|W} \mid \mu \in [\![P]\!]_G\}$$

SPARQL SELECT $\subseteq$ nr-Datalog$^{\neg}$

Theorem (AG08)

Every query expressible in SPARQL SELECT is expressible in nr-Datalog$^{\neg}$.

SPARQL SELECT $\subseteq$ nr-Datalog$^\neg$

Theorem (AG08)

Every query expressible in SPARQL SELECT is expressible in nr-Datalog$^\neg$.

Example

$((?X, a, b)$ OPT $(?X, c, ?Z))$ is equivalent to:

$$
\begin{array}{rcl}
\textsc{Answer}(X, Z) & \leftarrow & triple(X, a, b), triple(X, c, Z) \\
\textsc{Answer}(X, Z) & \leftarrow & triple(X, a, b), \mathbf{N}(Z), \neg q(X) \\
q(X) & \leftarrow & triple(X, c, V)
\end{array}
$$

nr-Datalog$^\neg$ $\subseteq$ SPARQL SELECT

Theorem (AG08)

Every query over $\{triple\}$ expressible in nr-Datalog$^\neg$ is expressible in SPARQL SELECT.

nr-Datalog$^\neg$ $\subseteq$ SPARQL SELECT

Theorem (AG08)

Every query over $\{triple\}$ expressible in nr-Datalog$^\neg$ is expressible in SPARQL SELECT.

But SPARQL SELECT is so positive!

nr-Datalog$^\neg$ $\subseteq$ SPARQL SELECT

Theorem (AG08)

Every query over $\{triple\}$ expressible in nr-Datalog$^\neg$ is expressible in SPARQL SELECT.

But SPARQL SELECT is so positive!

- Difference operator is definable in SPARQL!

Let MINUS be defined as:

$$[\![P_1 \ \text{MINUS} \ P_2]\!]_G \quad = \quad [\![P_1]\!]_G \smallsetminus [\![P_2]\!]_G$$

MINUS operator

Let MINUS be defined as:

$$[\![P_1 \ \text{MINUS} \ P_2]\!]_G \quad = \quad [\![P_1]\!]_G \smallsetminus [\![P_2]\!]_G$$

Proposition

$(P_1 \ \text{MINUS} \ P_2)$ *is equivalent to:*

$$\Big(P_1 \ \text{OPT} \ (P_2 \ \text{AND} \ (?X_1, ?X_2, ?X_3)) \Big) \ \text{FILTER} \ \neg \, \text{bound}(?X_1),$$

where $?X_1, ?X_2, ?X_3$ *are mentioned neither in* P_1 *nor in* P_2*.*

SPARQL SELECT $\subseteq$ nr-Datalog$^\neg$: Example

Consider the following nr-Datalog$^\neg$ program:

$$\begin{array}{rcl} \textsc{Answer}(X) & \leftarrow & \color{red}{triple(X, a, b), \neg q(X)} \\ q(X) & \leftarrow & \color{blue}{triple(X, c, Y), triple(Y, c, Z)} \end{array}$$

SPARQL SELECT $\subseteq$ nr-Datalog$^{\neg}$: Example

Consider the following nr-Datalog$^{\neg}$ program:

$$
\begin{array}{rcl}
\textsc{Answer}(X) & \leftarrow & triple(X, a, b), \neg q(X) \\
q(X) & \leftarrow & triple(X, c, Y), triple(Y, c, Z)
\end{array}
$$

This program is equivalent to:

$$(?X, a, b) \ \text{MINUS} \ \Big((?X, c, ?Y) \ \text{AND} \ (?Y, c, ?Z)\Big)$$

Second part: Ground RDF with RDFS vocabulary

- Syntax and formal semantics
- Querying RDFS data
  - nSPARQL: A navigational query language for RDFS
  - Expressiveness
  - Complexity of the evaluation problem

Second part: Ground RDF with RDFS vocabulary

- Syntax and formal semantics
- Querying RDFS data
  - nSPARQL: A navigational query language for RDFS
  - Expressiveness
  - Complexity of the evaluation problem

Syntax of RDFS

RDFS extends RDF with a schema vocabulary: subPropertyOf (`rdf:sp`), subClassOf (`rdf:sc`), domain (`rdf:dom`), range (`rdf:range`), type (`rdf:type`).

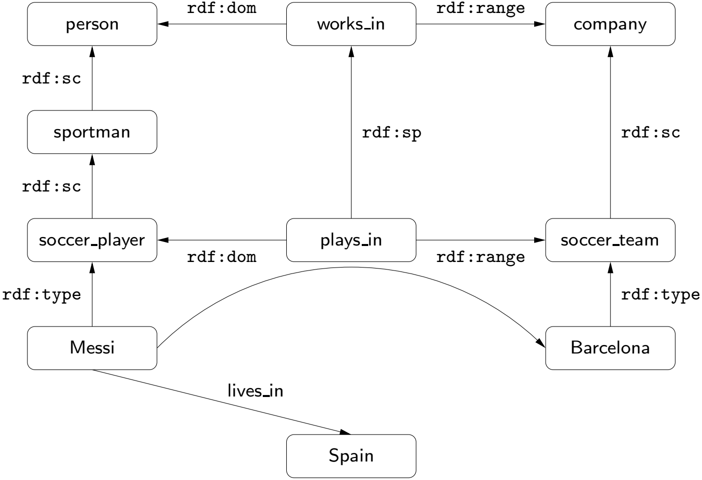
Syntax of RDFS

RDFS extends RDF with a schema vocabulary: subPropertyOf (`rdf:sp`), subClassOf (`rdf:sc`), domain (`rdf:dom`), range (`rdf:range`), type (`rdf:type`).

How can one query RDFS data?

- Evaluating queries which involve this vocabulary is challenging.
- There is not yet consensus in the Semantic Web community on how to define a query language for RDFS.

A simple SPARQL query: (Messi, rdf:type, person)



Semantics of RDFS

Checking whether a triple t is in a graph G is the basic step when answering queries over RDF.

- For the case of RDFS, we need to check whether t is implied by G.

The notion of entailment in RDFS can be defined in terms of classical notions such as model, interpretation, etc.

- As for the case of first-order logic

This notion can also be characterized by a set of inference rules.

An inference system for RDFS

Inference rule: $\frac{R}{R'}$

- R and R' are sequences of RDF triples including symbols $\mathcal{A}$, $\mathcal{X}$, …, to be replaced by elements from U.

Instantiation of a rule: $\frac{\sigma(R)}{\sigma(R')}$

- $\sigma : \{\mathcal{A}, \mathcal{X}, \ldots\} \rightarrow U$

Application of a rule $\frac{R}{R'}$ to an RDF graph G:

- Select an assignment $\sigma : \{\mathcal{A}, \mathcal{X}, \ldots\} \rightarrow U$.
- if $\sigma(R) \subseteq G$, then obtain $G \cup \sigma(R')$

An inference system for RDFS

Sub-property : $$\frac{(\mathcal{A},\texttt{rdf:sp},\mathcal{B}) \ \ (\mathcal{B},\texttt{rdf:sp},\mathcal{C})}{(\mathcal{A},\texttt{rdf:sp},\mathcal{C})}$$

$$\frac{(\mathcal{A},\texttt{rdf:sp},\mathcal{B}) \ \ (\mathcal{X},\mathcal{A},\mathcal{Y})}{(\mathcal{X},\mathcal{B},\mathcal{Y})}$$

Subclass : $$\frac{(\mathcal{A},\texttt{rdf:sc},\mathcal{B}) \ \ (\mathcal{B},\texttt{rdf:sc},\mathcal{C})}{(\mathcal{A},\texttt{rdf:sc},\mathcal{C})}$$

$$\frac{(\mathcal{A},\texttt{rdf:sc},\mathcal{B}) \ \ (\mathcal{X},\texttt{rdf:type},\mathcal{A})}{(\mathcal{X},\texttt{rdf:type},\mathcal{B})}$$

Typing : $$\frac{(\mathcal{A},\texttt{rdf:dom},\mathcal{B}) \ \ (\mathcal{X},\mathcal{A},\mathcal{Y})}{(\mathcal{X},\texttt{rdf:type},\mathcal{B})}$$

$$\frac{(\mathcal{A},\texttt{rdf:range},\mathcal{B}) \ \ (\mathcal{X},\mathcal{A},\mathcal{Y})}{(\mathcal{Y},\texttt{rdf:type},\mathcal{B})}$$

Entailment in RDFS

Theorem (H03,GHM04,MPG07)

The previous system of inference rules characterize the notion of entailment in ground RDFS.

Thus, a triple t can be deduced from an RDF graph G ($G \models t$) if there exists an RDF G' such that:

- $t \in G'$
- G' can be obtained from G by successively applying the rules in the previous system.

Entailment in RDFS: Closure of a graph

Definition

The closure of an RDFS graph G ($\mathrm{cl}(G)$) is the graph obtained by adding to G all the triples that are implied by G.

A basic property of the closure:

- $G \models t$ iff $t \in \mathrm{cl}(G)$

Second part: Ground RDF with RDFS vocabulary

- Syntax and formal semantics
- Querying RDFS data
  - nSPARQL: A navigational query language for RDFS
  - Expressiveness
  - Complexity of the evaluation problem

Querying RDFS data

Basic step for answering queries over RDFS:

- Checking whether a triple t is in $\text{cl}(G)$.

Basic step for answering queries over RDFS:

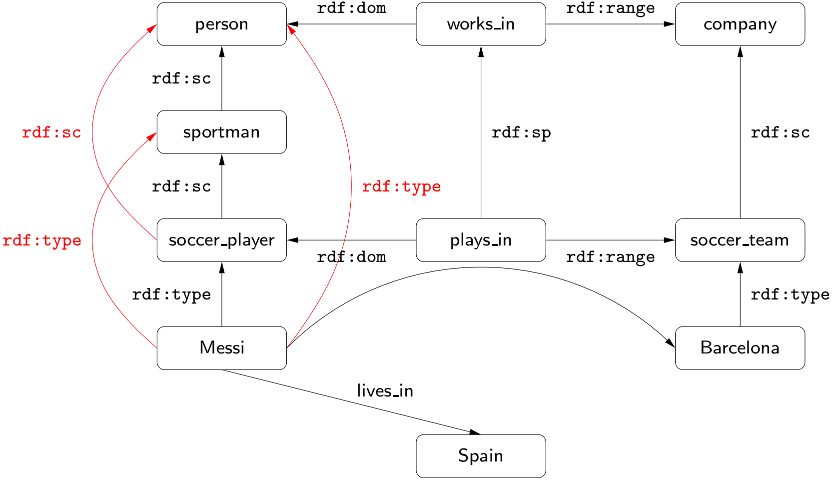
- Checking whether a triple t is in $\mathrm{cl}(G)$.

Definition

The *RDFS-evaluation of a graph pattern P over an RDFS graph G* is defined as the evaluation of P over $\mathrm{cl}(G)$:

$$[\![P]\!]^{\mathrm{rdfs}}_{G} \quad = \quad [\![P]\!]_{\mathrm{cl}(G)}$$

Example: (Messi, rdf:type, person) over the closure



Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G:

- Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF.

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G:

- Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF.

This approach has some drawbacks:

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G:

- Compute $\text{cl}(G)$, and then evaluate P over $\text{cl}(G)$ as for RDF.

This approach has some drawbacks:

- The size of the closure of G can be quadratic in the size of G.

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G:

- Compute $\mathrm{cl}(G)$, and then evaluate P over $\mathrm{cl}(G)$ as for RDF.

This approach has some drawbacks:

- The size of the closure of G can be quadratic in the size of G.
- Once the closure has been computed, all the queries are evaluated over a graph which can be much larger than the original graph.

Answering SPARQL queries over RDFS

A simple approach for answering a SPARQL query P over an RDFS graph G:

- Compute $\mathrm{cl}(G)$, and then evaluate P over $\mathrm{cl}(G)$ as for RDF.

This approach has some drawbacks:

- The size of the closure of G can be quadratic in the size of G.
- Once the closure has been computed, all the queries are evaluated over a graph which can be much larger than the original graph.
- The approach is not goal-oriented.

  When evaluating $(a, \texttt{rdf:sc}, b)$, a goal-oriented approach should not compute $\mathrm{cl}(G)$:

  - It should just verify whether there exists a path from a to b in G where every edge has label `rdf:sc`.

Extending SPARQL with navigational capabilities

The example (a, `rdf:sc`, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Extending SPARQL with navigational capabilities

The example $(a, \texttt{rdf:sc}, b)$ suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

Extending SPARQL with navigational capabilities

The example (a, rdf:sc, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDFS graph G is answered by navigating G (cl(G) is not computed).

Extending SPARQL with navigational capabilities

The example $(a, \texttt{rdf:sc}, b)$ suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDFS graph G is answered by navigating G ($\mathrm{cl}(G)$ is not computed).

This approach has some advantages:

Extending SPARQL with navigational capabilities

The example (a, `rdf:sc`, b) suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDFS graph G is answered by navigating G ($\mathrm{cl}(G)$ is not computed).

This approach has some advantages:

- It is goal-oriented.

Extending SPARQL with navigational capabilities

The example $(a, \texttt{rdf:sc}, b)$ suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

- A query P over an RDFS graph G is answered by navigating G (cl(G) is not computed).

This approach has some advantages:

- It is goal-oriented.
- It has been used to design query languages for XML (e.g., XPath and XQuery). The results for these languages can be used here.

Extending SPARQL with navigational capabilities

The example $(a, \texttt{rdf:sc}, b)$ suggests that a query language with navigational capabilities could be appropriate for RDFS.

Possible approach: Extend SPARQL with navigational capabilities.

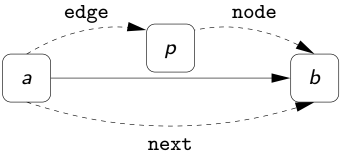
- A query P over an RDFS graph G is answered by navigating G ($\mathrm{cl}(G)$ is not computed).

This approach has some advantages:

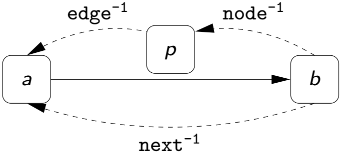
- It is goal-oriented.
- It has been used to design query languages for XML (e.g., XPath and XQuery). The results for these languages can be used here.
- Navigational operators allow to express natural queries that are not expressible in SPARQL over RDFS.

Navigational axes

Forward axes for an RDF triple (a, p, b):



Backward axes for an RDF triple (a, p, b):



A first attempt: rSPARQL

Syntax of navigational expressions:

$$exp \;:=\; \texttt{self} \mid \texttt{self}::a \mid \text{axis} \mid \text{axis}::a \mid exp/exp \mid exp|exp \mid exp^*$$

where $a \in U$ and axis $\in \{\texttt{next}, \texttt{next}^{-1}, \texttt{edge}, \texttt{edge}^{-1}, \texttt{node}, \texttt{node}^{-1}\}$.

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$[\![\texttt{self}]\!]_G \quad = \quad \{(x, x) \mid x \text{ is in } G\}$$

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$\begin{aligned}
[\![\texttt{self}]\!]_G &= \{(x,x) \mid x \text{ is in } G\} \\
[\![\texttt{next}]\!]_G &= \{(x,y) \mid \exists z \in U \; (x,z,y) \in G\}
\end{aligned}$$

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$
\begin{array}{lcl}
[\![\texttt{self}]\!]_G & = & \{(x, x) \mid x \text{ is in } G\} \\
[\![\texttt{next}]\!]_G & = & \{(x, y) \mid \exists z \in U \ (x, z, y) \in G\} \\
[\![\texttt{edge}]\!]_G & = & \{(x, y) \mid \exists z \in U \ (x, y, z) \in G\}
\end{array}
$$

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$\begin{array}{rcl}
[\![\texttt{self}]\!]_G & = & \{(x, x) \mid x \text{ is in } G\} \\
[\![\texttt{next}]\!]_G & = & \{(x, y) \mid \exists z \in U \ (x, z, y) \in G\} \\
[\![\texttt{edge}]\!]_G & = & \{(x, y) \mid \exists z \in U \ (x, y, z) \in G\} \\
[\![\texttt{self}{::}a]\!]_G & = & \{(a, a)\}
\end{array}$$

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$
\begin{aligned}
[\![\texttt{self}]\!]_G &= \{(x, x) \mid x \text{ is in } G\} \\
[\![\texttt{next}]\!]_G &= \{(x, y) \mid \exists z \in U \ (x, z, y) \in G\} \\
[\![\texttt{edge}]\!]_G &= \{(x, y) \mid \exists z \in U \ (x, y, z) \in G\} \\
[\![\texttt{self}::a]\!]_G &= \{(a, a)\} \\
[\![\texttt{next}::a]\!]_G &= \{(x, y) \mid (x, a, y) \in G\}
\end{aligned}
$$

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$
\begin{array}{rcl}
[\![\texttt{self}]\!]_G & = & \{(x,x) \mid x \text{ is in } G\} \\
[\![\texttt{next}]\!]_G & = & \{(x,y) \mid \exists z \in U \ (x,z,y) \in G\} \\
[\![\texttt{edge}]\!]_G & = & \{(x,y) \mid \exists z \in U \ (x,y,z) \in G\} \\
[\![\texttt{self}::a]\!]_G & = & \{(a,a)\} \\
[\![\texttt{next}::a]\!]_G & = & \{(x,y) \mid (x,a,y) \in G\} \\
[\![\texttt{edge}::a]\!]_G & = & \{(x,y) \mid (x,y,a) \in G\}
\end{array}
$$

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$
\begin{array}{rcl}
[\![\texttt{self}]\!]_G & = & \{(x,x) \mid x \text{ is in } G\} \\
[\![\texttt{next}]\!]_G & = & \{(x,y) \mid \exists z \in U\ (x,z,y) \in G\} \\
[\![\texttt{edge}]\!]_G & = & \{(x,y) \mid \exists z \in U\ (x,y,z) \in G\} \\
[\![\texttt{self}::a]\!]_G & = & \{(a,a)\} \\
[\![\texttt{next}::a]\!]_G & = & \{(x,y) \mid (x,a,y) \in G\} \\
[\![\texttt{edge}::a]\!]_G & = & \{(x,y) \mid (x,y,a) \in G\} \\
[\![exp_1/exp_2]\!]_G & = & \{(x,y) \mid \exists z\ (x,z) \in [\![exp_1]\!]_G \text{ and} \\
 & & \qquad\qquad\qquad (z,y) \in [\![exp_2]\!]_G\}
\end{array}
$$

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$
\begin{array}{rcl}
[\![\texttt{self}]\!]_G & = & \{(x,x) \mid x \text{ is in } G\} \\
[\![\texttt{next}]\!]_G & = & \{(x,y) \mid \exists z \in U\ (x,z,y) \in G\} \\
[\![\texttt{edge}]\!]_G & = & \{(x,y) \mid \exists z \in U\ (x,y,z) \in G\} \\
[\![\texttt{self::}a]\!]_G & = & \{(a,a)\} \\
[\![\texttt{next::}a]\!]_G & = & \{(x,y) \mid (x,a,y) \in G\} \\
[\![\texttt{edge::}a]\!]_G & = & \{(x,y) \mid (x,y,a) \in G\} \\
[\![exp_1/exp_2]\!]_G & = & \{(x,y) \mid \exists z\ (x,z) \in [\![exp_1]\!]_G \text{ and} \\
 & & \qquad\qquad\qquad (z,y) \in [\![exp_2]\!]_G\} \\
[\![exp_1|exp_2]\!]_G & = & [\![exp_1]\!]_G \cup [\![exp_2]\!]_G
\end{array}
$$

A first attempt: rSPARQL

Given an RDFS graph G, the semantics of navigational expressions is defined as follows:

$$
\begin{array}{rcl}
[\![\texttt{self}]\!]_G & = & \{(x,x) \mid x \text{ is in } G\} \\
[\![\texttt{next}]\!]_G & = & \{(x,y) \mid \exists z \in U \ (x,z,y) \in G\} \\
[\![\texttt{edge}]\!]_G & = & \{(x,y) \mid \exists z \in U \ (x,y,z) \in G\} \\
[\![\texttt{self}::a]\!]_G & = & \{(a,a)\} \\
[\![\texttt{next}::a]\!]_G & = & \{(x,y) \mid (x,a,y) \in G\} \\
[\![\texttt{edge}::a]\!]_G & = & \{(x,y) \mid (x,y,a) \in G\} \\
[\![exp_1/exp_2]\!]_G & = & \{(x,y) \mid \exists z \ (x,z) \in [\![exp_1]\!]_G \text{ and} \\
 & & \qquad\qquad\qquad (z,y) \in [\![exp_2]\!]_G\} \\
[\![exp_1|exp_2]\!]_G & = & [\![exp_1]\!]_G \cup [\![exp_2]\!]_G \\
[\![exp^*]\!]_G & = & [\![\texttt{self}]\!]_G \cup [\![exp]\!]_G \cup [\![exp/exp]\!]_G \cup \\
 & & \qquad\qquad\qquad [\![exp/exp/exp]\!]_G \cup \cdots
\end{array}
$$

A first attempt: rSPARQL

Syntax of rSPARQL:

- Basic component: A triple of the form (x, exp, y)
  - *exp* is a navigational expression
  - x (resp. y) is either an element from U or a variable
- Operators: AND, FILTER, UNION and OPT

A first attempt: rSPARQL

Syntax of rSPARQL:

- Basic component: A triple of the form (x, exp, y)
  - *exp* is a navigational expression
  - x (resp. y) is either an element from U or a variable
- Operators: AND, FILTER, UNION and OPT

Triple $(?X, ?Y, ?Z)$ is not allowed.

A first attempt: rSPARQL

Syntax of rSPARQL:

- Basic component: A triple of the form (x, exp, y)
  - *exp* is a navigational expression
  - x (resp. y) is either an element from U or a variable
- Operators: AND, FILTER, UNION and OPT

Triple $(?X, ?Y, ?Z)$ is not allowed.

- It computes the closure!

rSPARQL: What can we express?

Example

rSPARQL: What can we express?

Example

- (Messi, `next::lives_in`, Spain): Equivalent to SPARQL pattern (Messi, `lives_in`, Spain)

rSPARQL: What can we express?

Example

- (Messi, `next::lives_in`, Spain): Equivalent to SPARQL pattern (Messi, `lives_in`, Spain)
- ($?X$, `edge`::a, $?Y$): Equivalent to SPARQL pattern ($?X, ?Y, a$)

rSPARQL: What can we express?

Example

- (Messi, `next::lives_in`, Spain): Equivalent to SPARQL pattern (Messi, `lives_in`, Spain)
- $(?X$, edge::a, $?Y)$: Equivalent to SPARQL pattern $(?X, ?Y, a)$
- $(?X$, node::a, $?Y)$: Equivalent to SPARQL pattern $(a, ?X, ?Y)$

rSPARQL: What can we express?

Example

- (Messi, $\texttt{next::lives\_in}$, Spain): Equivalent to SPARQL pattern (Messi, $\texttt{lives\_in}$, Spain)
- $(?X, \texttt{edge::}a, ?Y)$: Equivalent to SPARQL pattern $(?X, ?Y, a)$
- $(?X, \texttt{node::}a, ?Y)$: Equivalent to SPARQL pattern $(a, ?X, ?Y)$
- $(?X, (\texttt{next::(rdf:sc)})^{+}, ?Y)$: Verifies whether $?X$ is a subclass of $?Y$.

A first attempt: rSPARQL

Semantics of rSPARQL: Evaluation of $t = (?X, exp, ?Y)$ over an RDF graph G is the set of mappings μ such that:

A first attempt: rSPARQL

Semantics of rSPARQL: Evaluation of $t = (?X, exp, ?Y)$ over an RDF graph G is the set of mappings μ such that:

- The domain of μ is $\{?X, ?Y\}$, and

A first attempt: rSPARQL

Semantics of rSPARQL: Evaluation of $t = (?X, exp, ?Y)$ over an RDF graph G is the set of mappings μ such that:

- The domain of μ is $\{?X, ?Y\}$, and
- $(\mu(?X), \mu(?Y)) \in [\![exp]\!]_G$

A first attempt: rSPARQL

Semantics of rSPARQL: Evaluation of $t = (?X, exp, ?Y)$ over an RDF graph G is the set of mappings μ such that:

- The domain of μ is $\{?X, ?Y\}$, and
- $(\mu(?X), \mu(?Y)) \in [\![exp]\!]_G$

Example

What does $(?X, (\texttt{next}::\mathrm{KLM} \mid \texttt{next}::\mathrm{AirFrance})^+, ?Y)$ represent?

Is rSPARQL a good language for RDFS?

How do we test whether a language is appropriate for RDFS?

- Can we capture SPARQL over RDFS?

Is rSPARQL a good language for RDFS?

How do we test whether a language is appropriate for RDFS?

- Can we capture SPARQL over RDFS?

For every RDFS graph G and SPARQL pattern P, we would like to find a rSPARQL pattern Q such that:

$$[\![P]\!]_G^{\text{rdfs}} = [\![Q]\!]_G$$

Is rSPARQL a good language for RDFS?

How do we test whether a language is appropriate for RDFS?

- Can we capture SPARQL over RDFS?

For every RDFS graph G and SPARQL pattern P, we would like to find a rSPARQL pattern Q such that:

$$[\![P]\!]_G^{\text{rdfs}} \;=\; [\![Q]\!]_G$$

But we trivially fail because of triple $(?X, ?Y, ?Z)$.

Is rSPARQL a good language for RDFS?

How do we test whether a language is appropriate for RDFS?

- Can we capture SPARQL over RDFS?

For every RDFS graph G and SPARQL pattern P, we would like to find a rSPARQL pattern Q such that:

$$[\![P]\!]_G^{\text{rdfs}} \quad = \quad [\![Q]\!]_G$$

But we trivially fail because of triple $(?X, ?Y, ?Z)$.

- We need to use a fragment of SPARQL.

A good fragment of SPARQL for our study

$\mathcal{T}$: Set of triples (x, y, z) where $x \in U$ or $y \in U$ or $z \in U$.

A good fragment of SPARQL for our study

$\mathcal{T}$: Set of triples (x, y, z) where $x \in U$ or $y \in U$ or $z \in U$.

- $(?X, a, b)$, $(?X, a, ?Y)$ and $(?X, ?Y, a)$

A good fragment of SPARQL for our study

$\mathcal{T}$: Set of triples (x, y, z) where $x \in U$ or $y \in U$ or $z \in U$.

- $(?X, a, b)$, $(?X, a, ?Y)$ and $(?X, ?Y, a)$

$\mathcal{T}$-SPARQL: Fragment of SPARQL where triple patterns are taken from $\mathcal{T}$.

Is rSPARQL a good language for RDFS?

Theorem (PAG08)

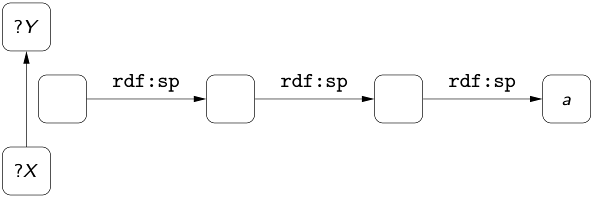
There exists a $\mathcal{T}$-SPARQL pattern P for which there is no rSPARQL pattern Q such that $[\![P]\!]_G^{\mathrm{rdfs}} = [\![Q]\!]_G$ for every RDF graph G.

Is rSPARQL a good language for RDFS?

Theorem (PAG08)

There exists a $\mathcal{T}$-SPARQL pattern P for which there is no rSPARQL pattern Q such that $[\![P]\!]_G^{\text{rdfs}} = [\![Q]\!]_G$ for every RDF graph G.

The previous theorem holds even for $P = (?X, a, ?Y)$:



A successful attempt: Adding nesting

How can we capture $\mathcal{T}$-SPARQL over RDFS?

A successful attempt: Adding nesting

How can we capture $\mathcal{T}$-SPARQL over RDFS?

- We adopt the notion of branching from XPath.

A successful attempt: Adding nesting

How can we capture $\mathcal{T}$-SPARQL over RDFS?

- We adopt the notion of branching from XPath.

Syntax of *nested* regular expressions:

$$\begin{array}{rcl} exp & := & \texttt{self} \mid \texttt{self}::a \mid \text{axis} \mid \text{axis}::a \mid \\ & & \texttt{self}::[exp] \mid \text{axis}::[exp] \mid exp/exp \mid exp|exp \mid exp^{*} \end{array}$$

where $a \in U$ and axis $\in \{\texttt{next}, \texttt{next}^{-1}, \texttt{edge}, \texttt{edge}^{-1}, \texttt{node}, \texttt{node}^{-1}\}$.

A successful attempt: Adding nesting

Given an RDFS graph G, the semantics of nested regular expressions is defined as follows:

A successful attempt: Adding nesting

Given an RDFS graph G, the semantics of nested regular expressions is defined as follows:

$$\begin{aligned}[\![\texttt{next}::[exp]]\!]_G \quad = \quad \{(x, y) \mid \exists z, w \in U \ \ (x, z, y) \in G \text{ and} \\ (z, w) \in [\![exp]\!]_G\}\end{aligned}$$

A successful attempt: Adding nesting

Given an RDFS graph G, the semantics of nested regular expressions is defined as follows:

$$
\begin{aligned}
[\![\texttt{next}::[\mathit{exp}]]\!]_G &= \{(x, y) \mid \exists z, w \in U \;\; (x, z, y) \in G \text{ and } \\
&\qquad\qquad (z, w) \in [\![\mathit{exp}]\!]_G\} \\
[\![\texttt{edge}::[\mathit{exp}]]\!]_G &= \{(x, y) \mid \exists z, w \in U \;\; (x, y, z) \in G \text{ and } \\
&\qquad\qquad (z, w) \in [\![\mathit{exp}]\!]_G\}
\end{aligned}
$$

Capturing $\mathcal{T}$-SPARQL over RDFS

nSPARQL: Defined as rSPARQL but replacing navigational expressions by nested regular expressions.

Capturing $\mathcal{T}$-SPARQL over RDFS

nSPARQL: Defined as rSPARQL but replacing navigational expressions by nested regular expressions.

Example

RDFS evaluation of $(?X, a, ?Y)$ can be obtained by using nSPARQL:

$$(?X, \texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::a)], ?Y)$$

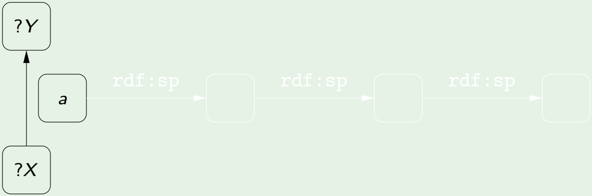
Capturing $\mathcal{T}$-SPARQL over RDFS

nSPARQL: Defined as rSPARQL but replacing navigational expressions by nested regular expressions.

Example

RDFS evaluation of $(?X, a, ?Y)$ can be obtained by using nSPARQL:

$$(?X, \texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::a)], ?Y)$$



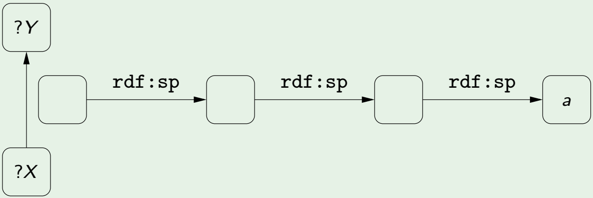
Capturing $\mathcal{T}$-SPARQL over RDFS

nSPARQL: Defined as rSPARQL but replacing navigational expressions by nested regular expressions.

Example

RDFS evaluation of $(?X, a, ?Y)$ can be obtained by using nSPARQL:

$$(?X, \texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::a)], ?Y)$$



Second part: Ground RDF with RDFS vocabulary

- Syntax and formal semantics
- Querying RDFS data
  - nSPARQL: A navigational query language for RDFS
  - Expressiveness
  - Complexity of the evaluation problem

nSPARQL captures $\mathcal{T}$-SPARQL over RDFS

Theorem (PAG08)

For every $\mathcal{T}$-SPARQL pattern P, there exists an nSPARQL pattern Q such that $[\![P]\!]_G^{\text{rdfs}} = [\![Q]\!]_G$ for every RDF graph G.

nSPARQL captures $\mathcal{T}$-SPARQL over RDFS

Theorem (PAG08)

For every $\mathcal{T}$-SPARQL pattern P, there exists an nSPARQL pattern Q such that $[\![P]\!]_G^{\text{rdfs}} = [\![Q]\!]_G$ for every RDF graph G.

Proof sketch

Replace $(?X, a, ?Y)$ by $(?X, \mathit{trans}(a), ?Y)$, where:

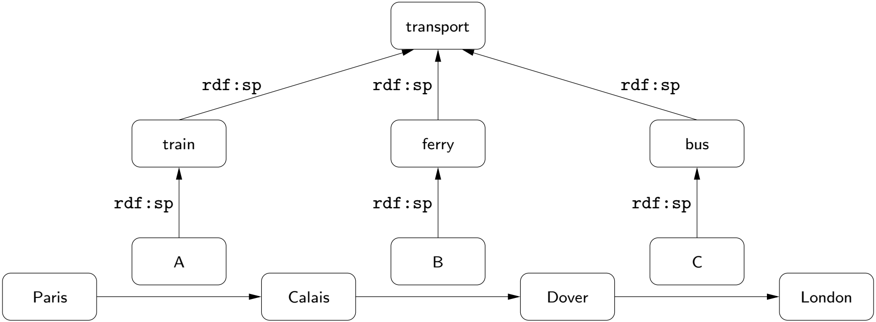
$$
\begin{array}{rcl}
\mathit{trans}(\texttt{rdf:dom}) & = & \texttt{next::(rdf:dom)} \\
\mathit{trans}(\texttt{rdf:range}) & = & \texttt{next::(rdf:range)} \\
\mathit{trans}(\texttt{rdf:sc}) & = & (\texttt{next::(rdf:sc)})^{+} \\
\mathit{trans}(\texttt{rdf:sp}) & = & (\texttt{next::(rdf:sp)})^{+}
\end{array}
$$

nSPARQL: Capturing SPARQL over RDFS

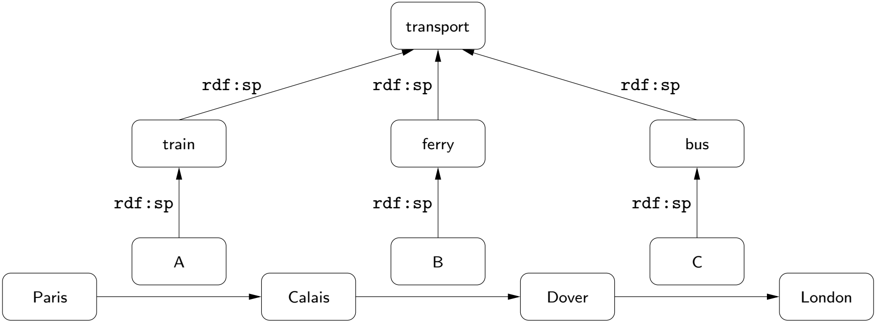
$$\begin{aligned}
\mathit{trans}(\mathtt{rdf:type}) \; = \;& \\
&\mathtt{next}{::}(\mathtt{rdf:type})/(\mathtt{next}{::}(\mathtt{rdf:sc}))^* \mid \\
&\mathtt{edge}/(\mathtt{next}{::}(\mathtt{rdf:sp}))^*/\mathtt{next}{::}(\mathtt{rdf:dom})/(\mathtt{next}{::}(\mathtt{rdf:sc}))^* \mid \\
&\mathtt{node}^{-1}/(\mathtt{next}{::}(\mathtt{rdf:sp}))^*/\mathtt{next}{::}(\mathtt{rdf:range})/(\mathtt{next}{::}(\mathtt{rdf:sc}))^*
\end{aligned}$$

$$\mathit{trans}(p) \; = \; \mathtt{next}{::}[(\mathtt{next}{::}(\mathtt{rdf:sp}))^*/\mathtt{self}{::}p]$$
$$\text{for } p \notin \{\mathtt{rdf:sc}, \mathtt{rdf:sp}, \mathtt{rdf:range}, \mathtt{rdf:dom}, \mathtt{rdf:type}\}$$

The extra expressive power of nSPARQL

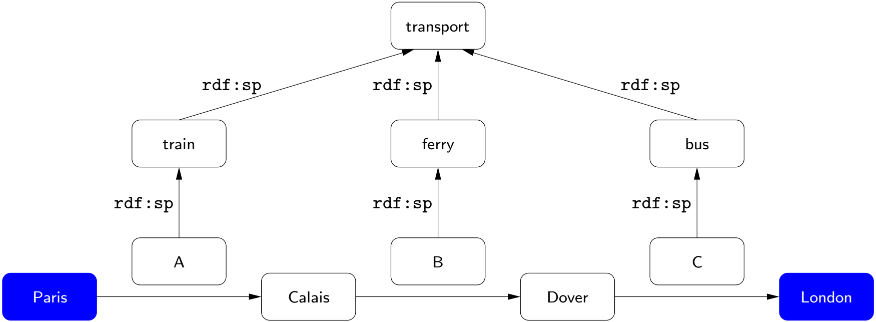


The extra expressive power of nSPARQL



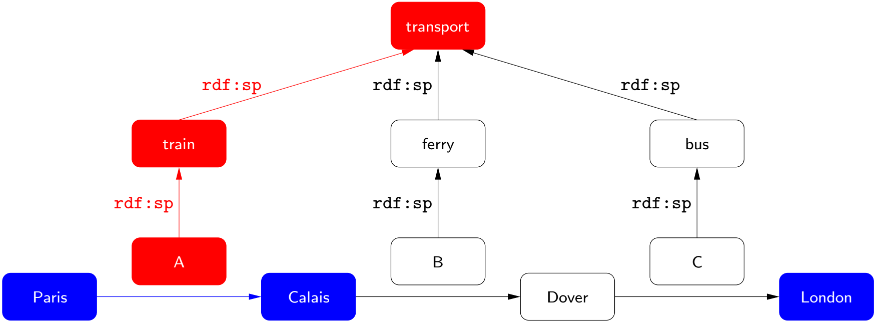
A natural query: $(?X, (\mathtt{next}::[(\mathtt{next}::(\mathtt{rdf:sp}))^*/(\mathtt{self}::\mathsf{travel})])^+, ?Y)$

The extra expressive power of nSPARQL



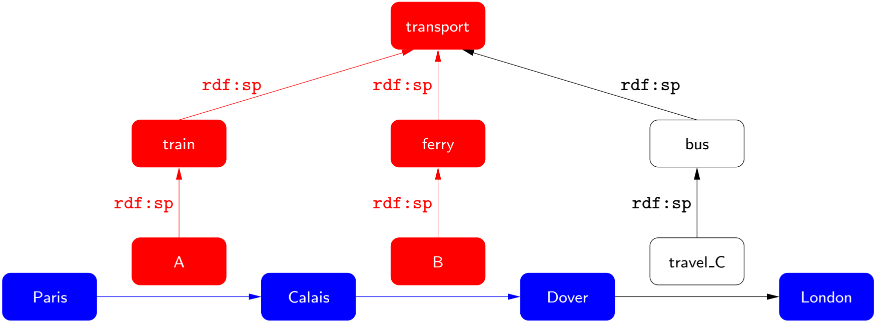
A natural query: $(?X, (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::\text{travel})])^+, ?Y)$

The extra expressive power of nSPARQL



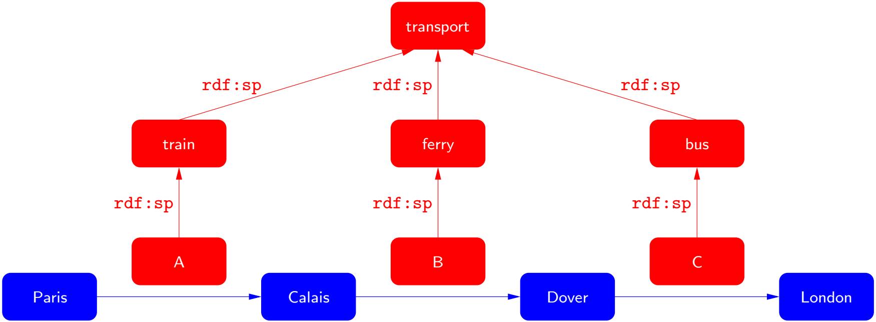
A natural query: $(?X, (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::\text{travel})])^+, ?Y)$

The extra expressive power of nSPARQL



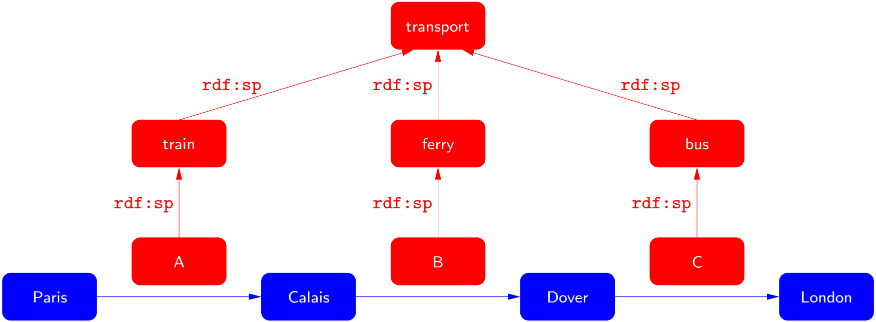
A natural query: $(?X,\ (\mathtt{next}{::}[(\mathtt{next}{::}(\mathtt{rdf:sp}))^{*}/(\mathtt{self}{::}\mathrm{travel})])^{+},\ ?Y)$

The extra expressive power of nSPARQL



A natural query: $(?X,\ (\texttt{next}{::}[(\texttt{next}{::}(\texttt{rdf:sp}))^*/(\texttt{self}{::}\mathsf{travel})])^+,\ ?Y)$

The extra expressive power of nSPARQL



A natural query: $(?X, (\mathtt{next}::[(\mathtt{next}::(\mathtt{rdf:sp}))^*/(\mathtt{self}::\mathrm{travel})])^+, ?Y)$

- This query cannot be expressed in SPARQL over RDFS.

Second part: Ground RDF with RDFS vocabulary

- Syntax and formal semantics
- Querying RDFS data
  - nSPARQL: A navigational query language for RDFS
  - Expressiveness
  - Complexity of the evaluation problem

The evaluation problem for nSPARQL

What is the complexity of the evaluation problem for nSPARQL?

The evaluation problem for nSPARQL

What is the complexity of the evaluation problem for nSPARQL?

- The lower bounds for SPARQL also apply in this case.
  - Just take a look at the proofs

The evaluation problem for nSPARQL

What is the complexity of the evaluation problem for nSPARQL?

- The lower bounds for SPARQL also apply in this case.
    - Just take a look at the proofs

Are there any new problems to consider?

The evaluation problem for nSPARQL

What is the complexity of the evaluation problem for nSPARQL?

- The lower bounds for SPARQL also apply in this case.
  - Just take a look at the proofs

Are there any new problems to consider?

- What is the complexity of evaluating a nested regular expression?

The evaluation problem for nSPARQL

What is the complexity of the evaluation problem for nSPARQL?

- The lower bounds for SPARQL also apply in this case.
  - Just take a look at the proofs

Are there any new problems to consider?

- What is the complexity of evaluating a nested regular expression?
- Can this be done efficiently?

The evaluation problem for nested regular expressions

Input:

A pair $(a, b) \in U \times U$, a nested regular expression *exp* and an RDF graph G

Question:

$$\text{Does } (a, b) \in [\![exp]\!]_G?$$

The evaluation problem for nested regular expressions

Theorem (PAG08)

The evaluation problem for nested regular expressions is solvable in time $O(|G| \cdot |exp|)$.

The evaluation problem for nested regular expressions

Theorem (PAG08)

The evaluation problem for nested regular expressions is solvable in time $O(|G| \cdot |exp|)$.

Proof sketch

Use an efficient evaluation algorithm for PDL.

- There are a few issues that have to be considered.

The evaluation problem for nested regular expressions

Simple example: pair (a, b), RDF graph G and navigational expression exp

The evaluation problem for nested regular expressions

Simple example: pair (a, b), RDF graph G and navigational expression *exp*

(1) Transform *exp* into an ϵ-NFA $\mathcal{A}_{exp}$

Simple example: pair (a, b), RDF graph G and navigational expression *exp*

(1) Transform *exp* into an ϵ-NFA $\mathcal{A}_{exp}$

(2) Construct an automaton $\mathcal{A}_G$ from G:

The evaluation problem for nested regular expressions

Simple example: pair (a, b), RDF graph G and navigational expression *exp*

(1) Transform *exp* into an ϵ-NFA $\mathcal{A}_{exp}$

(2) Construct an automaton $\mathcal{A}_G$ from G:
   - The states of $\mathcal{A}_G$ are the elements mentioned in G

The evaluation problem for nested regular expressions

Simple example: pair (a, b), RDF graph G and navigational expression *exp*

(1) Transform *exp* into an ϵ-NFA $\mathcal{A}_{exp}$

(2) Construct an automaton $\mathcal{A}_G$ from G:

- The states of $\mathcal{A}_G$ are the elements mentioned in G
- For every $(a, b, c) \in G$, automaton $\mathcal{A}_G$ contains:

$$
\begin{array}{lll}
(a,\ \texttt{next}{::}b,\ c) & (a,\ \texttt{edge}{::}c,\ b) & (b,\ \texttt{node}{::}a,\ c) \\
(c,\ \texttt{next}^{-1}{::}b,\ a) & (b,\ \texttt{edge}^{-1}{::}c,\ a) & (c,\ \texttt{node}^{-1}{::}a,\ b)
\end{array}
$$

The evaluation problem for nested regular expressions

Simple example: pair (a, b), RDF graph G and navigational expression exp

(1) Transform exp into an ϵ-NFA $\mathcal{A}_{exp}$

(2) Construct an automaton $\mathcal{A}_G$ from G:

- The states of $\mathcal{A}_G$ are the elements mentioned in G
- For every $(a, b, c) \in G$, automaton $\mathcal{A}_G$ contains:

$$\begin{array}{lll} (a,\ \texttt{next}{::}b,\ c) & (a,\ \texttt{edge}{::}c,\ b) & (b,\ \texttt{node}{::}a,\ c) \\ (c,\ \texttt{next}^{-1}{::}b,\ a) & (b,\ \texttt{edge}^{-1}{::}c,\ a) & (c,\ \texttt{node}^{-1}{::}a,\ b) \end{array}$$

- Every state is final

The evaluation problem for nested regular expressions

Simple example: pair (a, b), RDF graph G and navigational expression exp

(1) Transform exp into an ϵ-NFA $\mathcal{A}_{exp}$

(2) Construct an automaton $\mathcal{A}_G$ from G:

- The states of $\mathcal{A}_G$ are the elements mentioned in G
- For every $(a, b, c) \in G$, automaton $\mathcal{A}_G$ contains:

  $(a, \texttt{next}{::}b, c)$ $\quad (a, \texttt{edge}{::}c, b)$ $\quad (b, \texttt{node}{::}a, c)$
  $(c, \texttt{next}^{-1}{::}b, a)$ $\quad (b, \texttt{edge}^{-1}{::}c, a)$ $\quad (c, \texttt{node}^{-1}{::}a, b)$

- Every state is final

(3) Compute $\mathcal{A}_G \times \mathcal{A}_{exp}$

The evaluation problem for nested regular expressions

Simple example: pair (a, b), RDF graph G and navigational expression exp

(1) Transform exp into an ϵ-NFA $\mathcal{A}_{exp}$

(2) Construct an automaton $\mathcal{A}_G$ from G:

- The states of $\mathcal{A}_G$ are the elements mentioned in G
- For every $(a, b, c) \in G$, automaton $\mathcal{A}_G$ contains:

$$\begin{array}{lll} (a,\, \texttt{next}{::}b,\, c) & (a,\, \texttt{edge}{::}c,\, b) & (b,\, \texttt{node}{::}a,\, c) \\ (c,\, \texttt{next}^{-1}{::}b,\, a) & (b,\, \texttt{edge}^{-1}{::}c,\, a) & (c,\, \texttt{node}^{-1}{::}a,\, b) \end{array}$$

- Every state is final

(3) Compute $\mathcal{A}_G \times \mathcal{A}_{exp}$

(4) Verify whether (b, q_f) is reachable from (a, q_0) in $\mathcal{A}_G \times \mathcal{A}_{exp}$

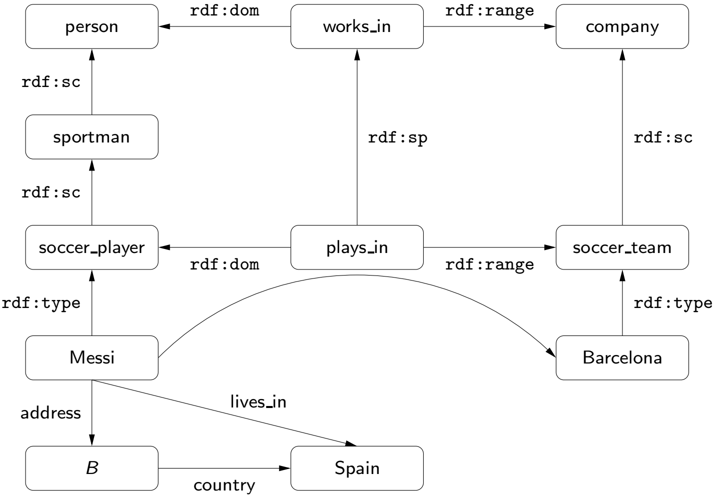
Third part: RDF with RDFS vocabulary

- Formal semantics
- A little bit about complexity

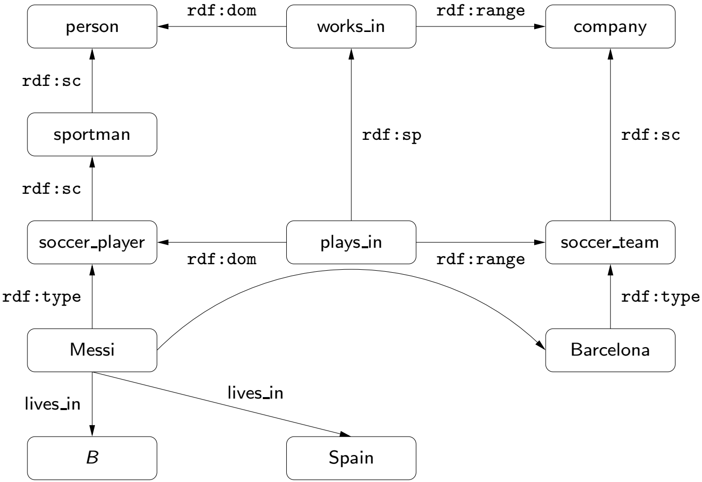
Third part: RDF with RDFS vocabulary

- Formal semantics
- A little bit about complexity

Does the blank node add some information?



What about now?



A fundamental notion: homomorphism

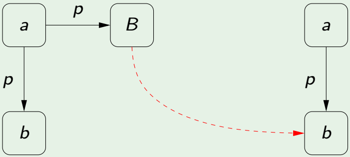
Definition

$h : U \cup B \rightarrow U \cup B$ is a homomorphism from G_1 to G_2 if:

- $h(c) = c$ for every $c \in U$;
- for every $(a, b, c) \in G_1$, $(h(a), h(b), h(c)) \in G_2$

Notation: $G_1 \rightarrow G_2$

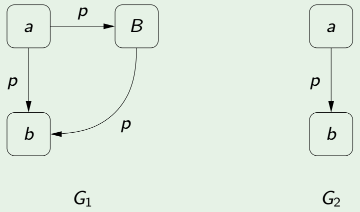
Example



Homomorphism and the notion of entailment

Example

In this case: $G_1 \not\rightarrow G_2$ and $G_2 \rightarrow G_1$



Intuitively: G_1 contains more information than G_2

A general notion of entailment

In this general scenario, entailment can also be defined in terms of classical notions such as model, interpretation, etc.

- As for the case of RDFS graphs without blank nodes

This notion can also be characterized by a set of inference rules.

A general system of inference rules

Existential rule :

Subproperty rules :

Subclass rules :

Typing rules :

Implicit typing :

A general system of inference rules

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules :

Subclass rules :

Typing rules :

Implicit typing :

A general system of inference rules

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\frac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules :

Typing rules :

Implicit typing :

A general system of inference rules

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\dfrac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules : $\dfrac{(a, \texttt{rdf:sc}, b) \quad (b, \texttt{rdf:sc}, c)}{(a, \texttt{rdf:sc}, c)}$

Typing rules :

Implicit typing :

A general system of inference rules

Existential rule : $\dfrac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\dfrac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules : $\dfrac{(a, \texttt{rdf:sc}, b) \quad (b, \texttt{rdf:sc}, c)}{(a, \texttt{rdf:sc}, c)}$

Typing rules : $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing :

A general system of inference rules

Existential rule : $\frac{G_1}{G_2}$ if $G_2 \rightarrow G_1$

Subproperty rules : $\dfrac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules : $\dfrac{(a, \texttt{rdf:sc}, b) \quad (b, \texttt{rdf:sc}, c)}{(a, \texttt{rdf:sc}, c)}$

Typing rules : $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing : $\dfrac{(q, \texttt{rdf:dom}, a) \quad (p, \texttt{rdf:sp}, q) \quad (b, p, c)}{(b, \texttt{rdf:type}, a)}$

A general system of inference rules

Existential rule :

Subproperty rules : $\dfrac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules :

Typing rules : $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing : $\dfrac{(q, \texttt{rdf:dom}, a) \quad (p, \texttt{rdf:sp}, q) \quad (b, p, c)}{(b, \texttt{rdf:type}, a)}$

A general system of inference rules

Existential rule :

Subproperty rules : $\dfrac{(p, \texttt{rdf:sp}, q) \quad (a, p, b)}{(a, q, b)}$

Subclass rules :

Typing rules : $\dfrac{(p, \texttt{rdf:dom}, c) \quad (a, p, b)}{(a, \texttt{rdf:type}, c)}$

Implicit typing : $\dfrac{(B, \texttt{rdf:dom}, a) \quad (p, \texttt{rdf:sp}, B) \quad (b, p, c)}{(b, \texttt{rdf:type}, a)}$

RDFS Entailment

Theorem (H03,GHM04,MPG07)

The previous system of inference rules characterize the notion of entailment in RDFS.

RDFS Entailment

Theorem (H03,GHM04,MPG07)

The previous system of inference rules characterize the notion of entailment in RDFS.

This system can be used to define $\mathrm{cl}(G)$.

RDFS Entailment

Theorem (H03,GHM04,MPG07)

The previous system of inference rules characterize the notion of entailment in RDFS.

This system can be used to define $\mathrm{cl}(G)$.

- This can be used to define the semantics of a query language over RDFS data.

Third part: RDF with RDFS vocabulary

- Formal semantics
- A bit about complexity

A little about complexity

Complexity (GHM04)

RDFS entailment is NP-complete.

A little about complexity

Complexity (GHM04)

RDFS entailment is NP-complete.

Proof sketch

Membership in NP: If $G \models t$, then there exists a polynomial-size proof of this fact.

Thank you!