Semantic Web: Query Languages for RDF and RDFS

Marcelo Arenas

Pontificia Universidad Católica de Chile
Centro de Investigación de la Web

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.]

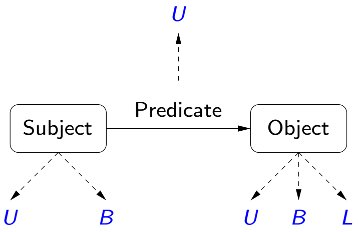
Specific Goals:

- 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

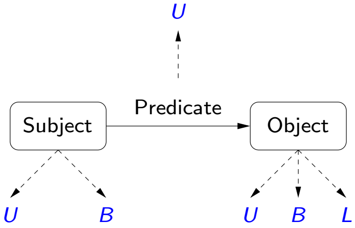


U = set of Uris

B = set of Blank nodes

L = set of Literals

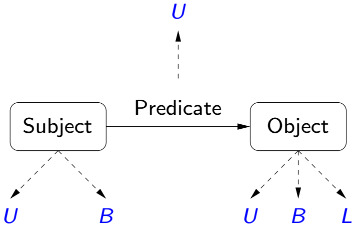
RDF formal model



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

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

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

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.

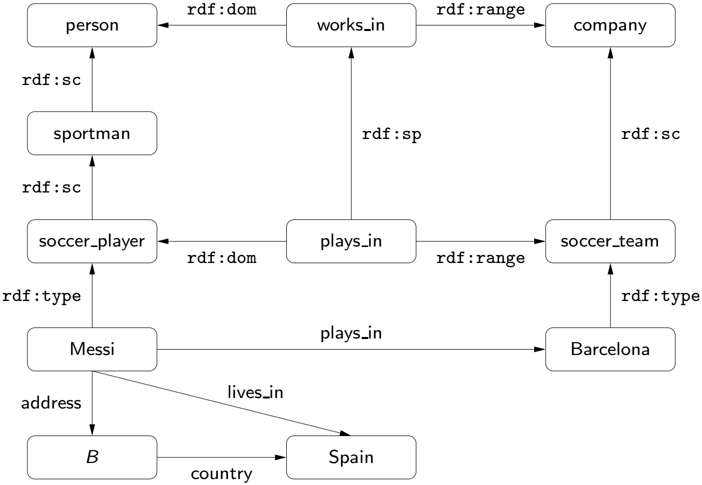
RDF formal model

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?

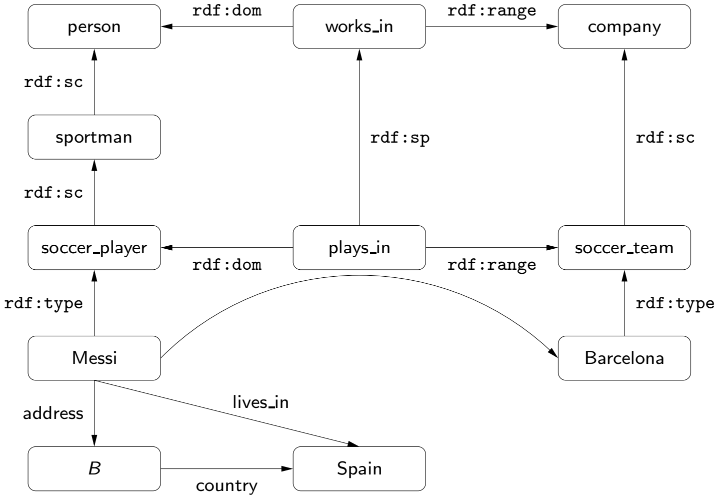
Some new challenges:

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

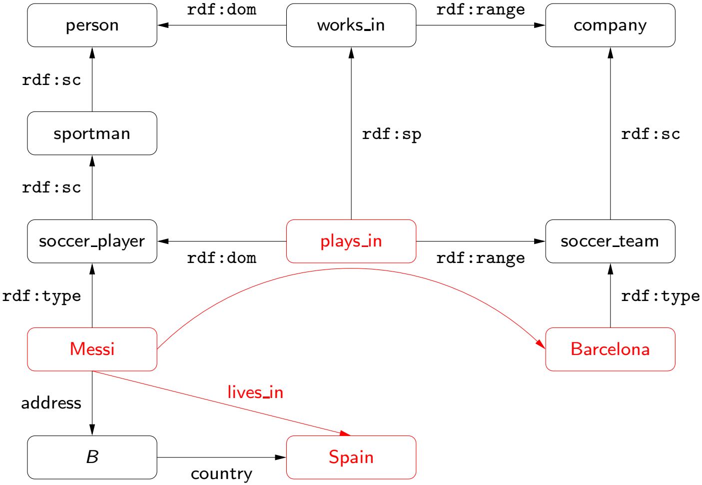
Why are database technologies interesting from an RDF point of view?

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

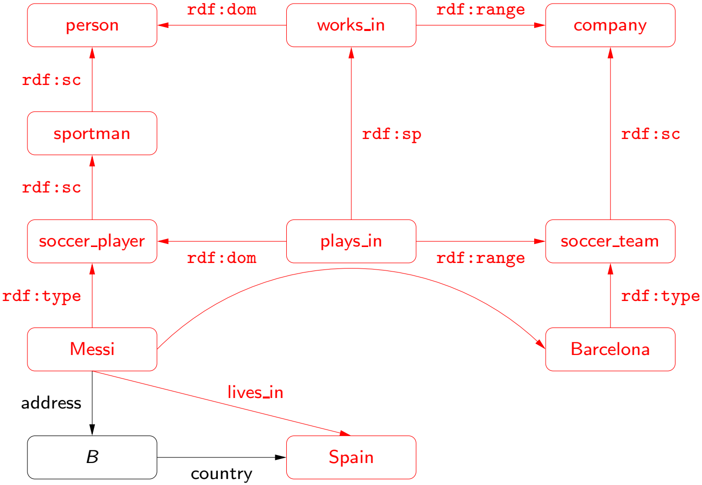
Previous example: A better representation



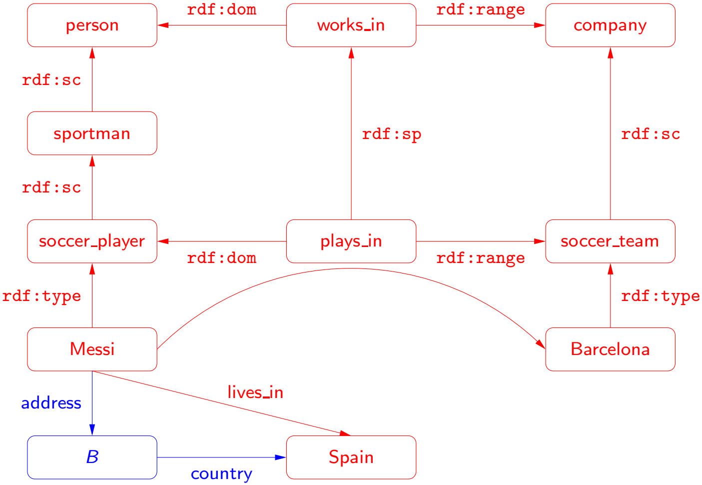
Previous example: A better representation



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

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:

$$H \longleftarrow P$$

- 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:

$$H \leftarrow P$$

- 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

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

- Graph patterns: full parenthesized algebra

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

A standard algebraic syntax

- 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.

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

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.

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

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

Example

graph	triple	evaluation
$(R_1, \text{name}, \text{john})$ $(R_1, \text{email}, \text{J@ed.ex})$ $(R_2, \text{name}, \text{paul})$	$(?X, \text{name}, ?Y)$	see 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: $\text{dom}(\mu) = \text{var}(t)$
- makes t to match the graph: $\mu(t) \in G$

Example

graph	triple	evaluation
(R_1, name, john)		
(R_1, email, J@ed.ex)	(?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, 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

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 $\mu_1\}$

Sets of mappings and operations

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

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

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

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

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

Semantics of SPARQL: An example

Example

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

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

$?X$	$?Y$
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

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

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

$?X$	$?Y$
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

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

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

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

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 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 $?X = c$, $?X \in \mathrm{dom}(\mu)$ and $\mu(?X) = c$;
- R is $?X = ?Y$, $?X, ?Y \in \mathrm{dom}(\mu)$ and $\mu(?X) = \mu(?Y)$;
- R is $\mathrm{bound}(?X)$ and $?X \in \mathrm{dom}(\mu)$;
- 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 $\text{bound}(?X)$ 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 $\mu \models R_2$;

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

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

Second part: Ground RDF with RDFS vocabulary

- Syntax and formal semantics
- Querying RDFS data
  - nSPARQL: A navigational query language for RDFS
  - Expressiveness

Second part: Ground RDF with RDFS vocabulary

- Syntax and formal semantics
- Querying RDFS data
  - nSPARQL: A navigational query language for RDFS
  - Expressiveness

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`).

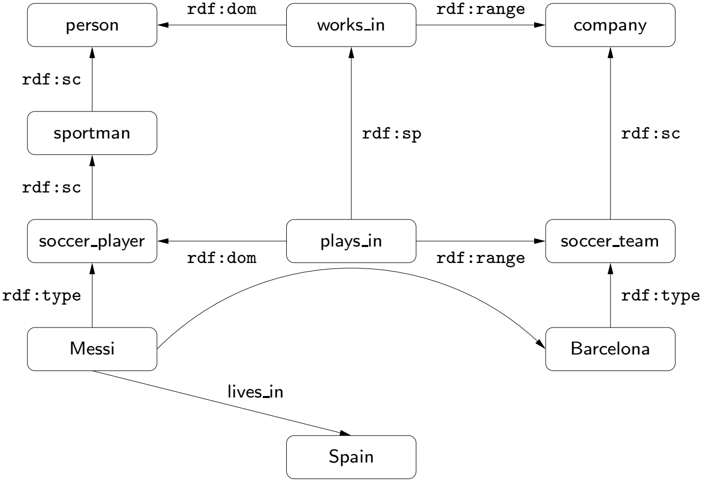
Syntax of RDFS

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: $\frac{\sigma(R)}{\sigma(R')}$

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

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

- Select an assignment $\sigma : \{\mathcal{A}, \mathcal{X}, \ldots\} \to U$.
- 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

Querying RDFS data

Basic step for answering queries over RDFS:

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

Querying RDFS data

Basic step for answering queries over RDFS:

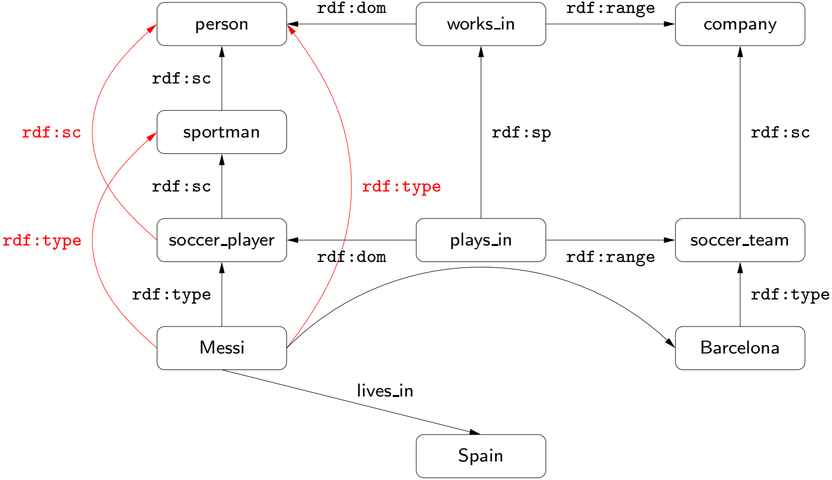
- Checking whether a triple t is in $\mathrm{cl}(G)$.

Definition

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

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

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 $\text{cl}(G)$, 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.
- 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 $\text{cl}(G)$, 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.
- 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 $\text{cl}(G)$:

  - 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 $(a, \texttt{rdf:sc}, b)$ 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).

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).

This approach has some advantages:

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.

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).

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

Possible approach: Extend SPARQL with navigational capabilities.

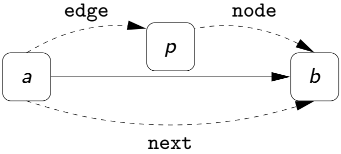
- A query P over an RDFS graph G is answered by navigating G (cl(G) is not computed).

This approach has some advantages:

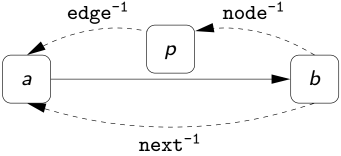
- It is goal-oriented.
- It has been used to design query languages for XML (e.g., XPath and XQuery). The results for these languages can be used here.
- Navigational operators allow to express natural queries that are not expressible in SPARQL over RDFS.

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:

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

A first attempt: rSPARQL

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

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

A first attempt: rSPARQL

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

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

A first attempt: rSPARQL

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

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

A first attempt: rSPARQL

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

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

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:

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

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, `next::lives_in`, Spain): Equivalent to SPARQL pattern (Messi, `lives_in`, Spain)
- $(?X, \texttt{edge}{::}a, ?Y)$: Equivalent to SPARQL pattern $(?X, ?Y, a)$
- $(?X, \texttt{node}{::}a, ?Y)$: Equivalent to SPARQL pattern $(a, ?X, ?Y)$
- $(?X, (\texttt{next}{::}(\texttt{rdf:sc}))^{+}, ?Y)$: 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 $(?X, (\texttt{next}::\text{KLM} \mid \texttt{next}::\text{AirFrance})^+, ?Y)$ 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:

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

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:

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

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:

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

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)

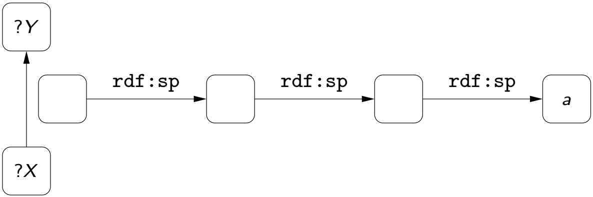
There exists a $\mathcal{T}$-SPARQL pattern P for which there is no rSPARQL pattern Q such that $[\![P]\!]_G^{\text{rdfs}} = [\![Q]\!]_G$ for every RDF graph G.

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:

$$\begin{aligned} exp \quad := \quad & \texttt{self} \mid \texttt{self}::a \mid \mathrm{axis} \mid \mathrm{axis}::a \mid \\ & \texttt{self}::[exp] \mid \mathrm{axis}::[exp] \mid exp/exp \mid exp|exp \mid exp^* \end{aligned}$$

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:

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

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:

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

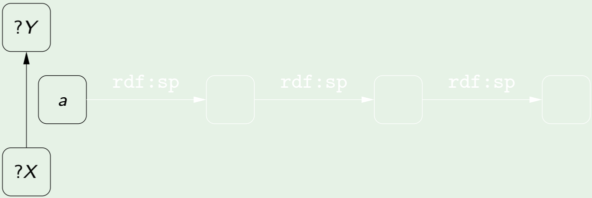
Capturing $\mathcal{T}$-SPARQL over RDFS

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:

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



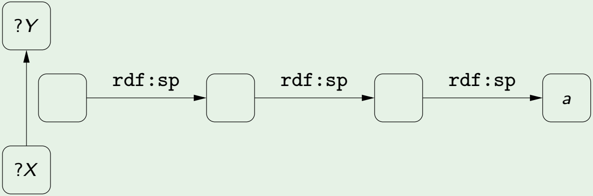
Capturing $\mathcal{T}$-SPARQL over RDFS

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

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:

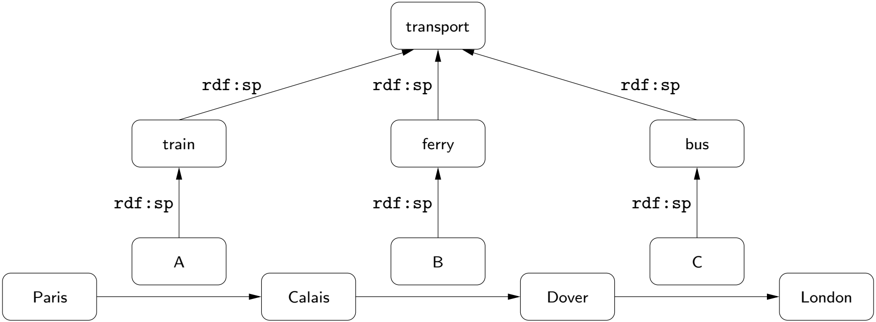
$$\begin{aligned}
\mathit{trans}(\texttt{rdf:dom}) &= \texttt{next::}(\texttt{rdf:dom}) \\
\mathit{trans}(\texttt{rdf:range}) &= \texttt{next::}(\texttt{rdf:range}) \\
\mathit{trans}(\texttt{rdf:sc}) &= (\texttt{next::}(\texttt{rdf:sc}))^+ \\
\mathit{trans}(\texttt{rdf:sp}) &= (\texttt{next::}(\texttt{rdf:sp}))^+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

nSPARQL: Capturing SPARQL over RDFS

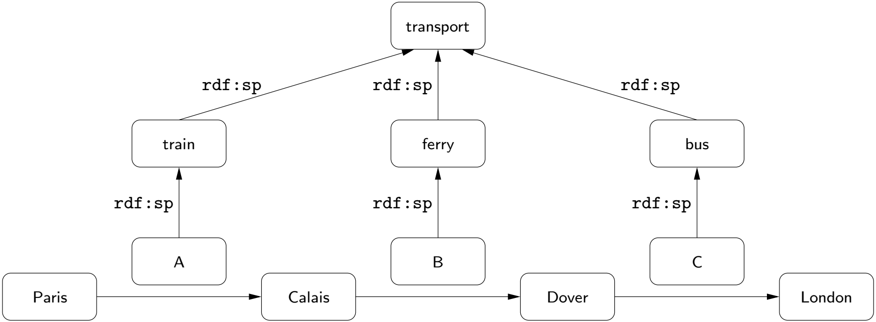
$$
\begin{aligned}
\mathit{trans}(\texttt{rdf:type}) \; = \;& \\
& \texttt{next::(rdf:type)}/(\texttt{next::(rdf:sc)})^* \mid \\
& \texttt{edge}/(\texttt{next::(rdf:sp)})^*/\texttt{next::(rdf:dom)}/(\texttt{next::(rdf:sc)})^* \mid \\
& \texttt{node}^{-1}/(\texttt{next::(rdf:sp)})^*/\texttt{next::(rdf:range)}/(\texttt{next::(rdf:sc)})^*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

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

The extra expressive power of nSPARQL

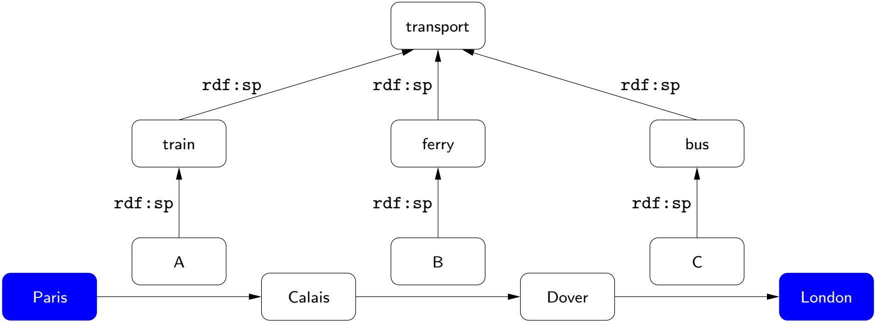


The extra expressive power of nSPARQL



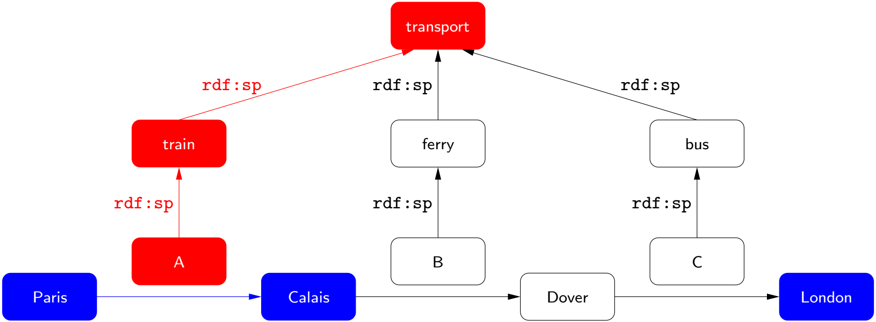
A natural query: $(?X,\ (\mathtt{next}::[(\mathtt{next}::(\mathtt{rdf:sp}))^*/(\mathtt{self}::\mathsf{travel})])^+,\ ?Y)$

The extra expressive power of nSPARQL



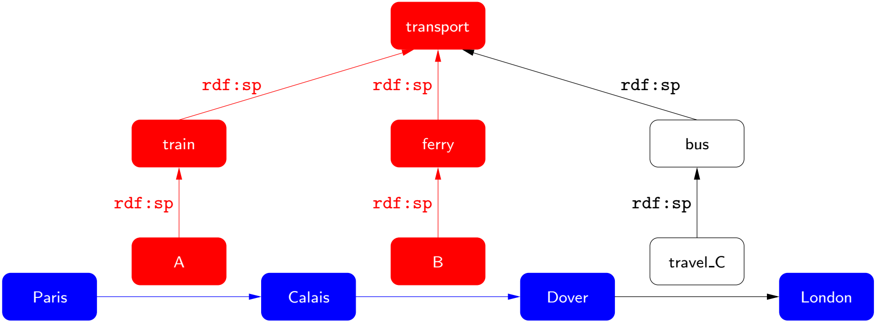
A natural query: $(?X,\ (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::\text{travel})])^+,\ ?Y)$

The extra expressive power of nSPARQL



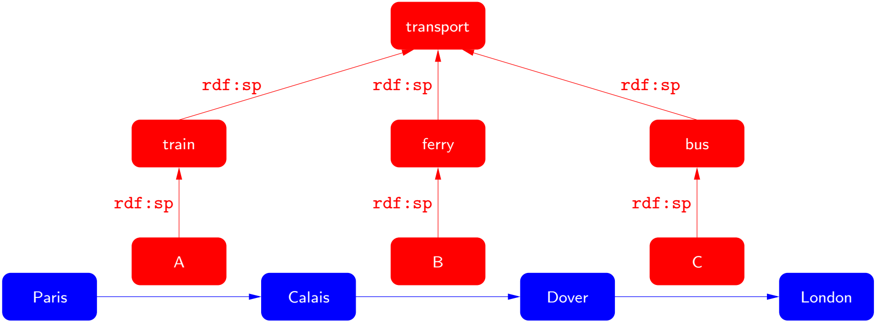
A natural query: $(?X, (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::\text{travel})])^+, ?Y)$

The extra expressive power of nSPARQL



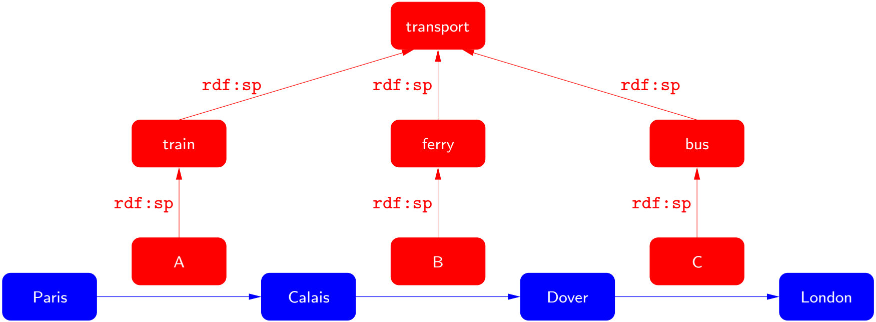
A natural query: $(?X, (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::\text{travel})])^+, ?Y)$

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



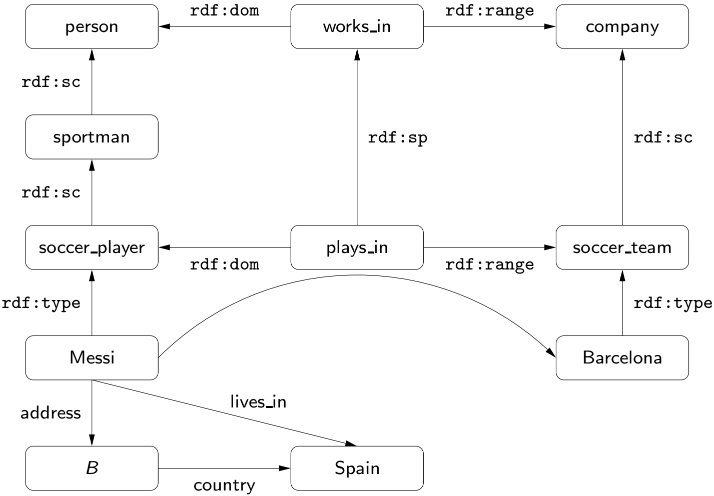
A natural query: $(?X, (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::\text{travel})])^+, ?Y)$

- This query cannot be expressed in SPARQL over RDFS.

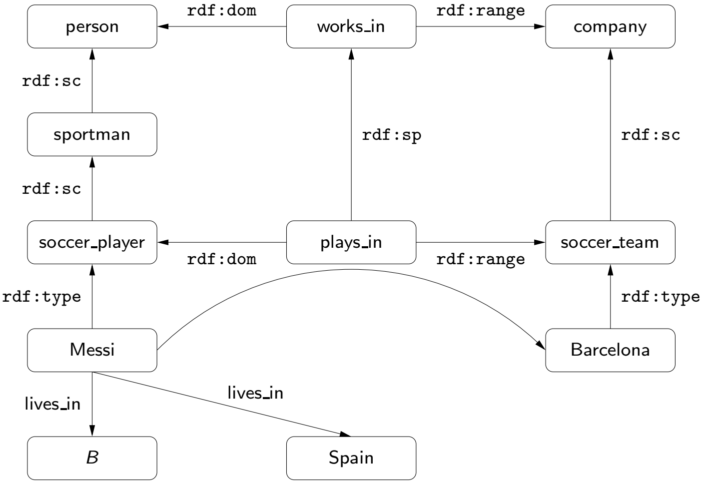
Third part: RDF with RDFS vocabulary

- Formal semantics
- A little bit about complexity

Does the blank node add some information?



What about now?



A fundamental notion: homomorphism

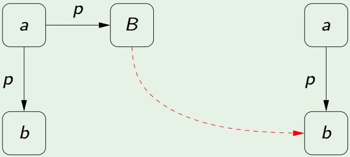
Definition

$h : U \cup B \to U \cup B$ 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: $G_1 \to G_2$

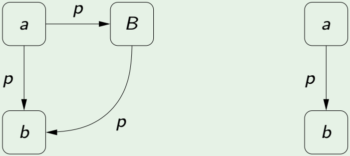
Example



Homomorphism and the notion of entailment

Example

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



G_1 G_2

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

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

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

Implicit typing :

A general system of inference rules

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

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

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

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

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

A general system of inference rules

Existential rule :

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

Subclass rules :

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

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

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!