SPARQL over RDF, and its possible extensions to RDFS

Marcelo Arenas

Department of Computer Science
Pontificia Universidad Católica de Chile
&
Center for Web Research
Universidad de Chile

Outline

- RDF and RDFS: A brief introduction
- SPARQL: A query language for RDF
  - Formal semantics
  - Complexity of the evaluation problem
  - Optimization methods
- SPARQL as a query language for RDFS
  - Formal semantics and the closure of an RDFS graph
- NAV-SPARQL: A navigational query language for RDFS

Outline

- RDF and RDFS: A brief introduction
- SPARQL: A query language for RDF
  - Formal semantics
  - Complexity of the evaluation problem
  - Optimization methods
- SPARQL as a query language for RDFS
  - Formal semantics and the closure of an RDFS graph
- NAV-SPARQL: A navigational query language for RDFS

This is joint work with Claudio Gutierrez and Jorge Pérez.

Outline

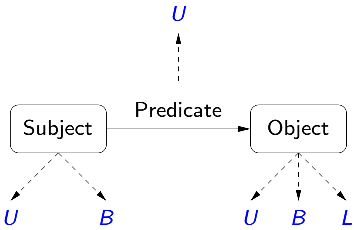
- RDF and RDFS: A brief introduction
- SPARQL: A query language for RDF
  - Formal semantics
  - Complexity of the evaluation problem
  - Optimization methods
- SPARQL as a query language for RDFS
  - Formal semantics and the closure of an RDFS graph
- NAV-SPARQL: A navigational query language for RDFS

This is joint work with Claudio Gutierrez and Jorge Pérez.

RDF in a nutshell

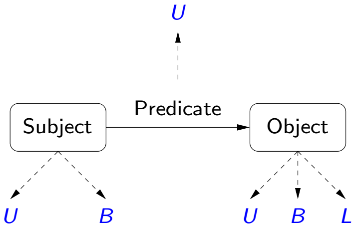
- RDF is the W3C proposal framework for representing information in the Web.
- Abstract syntax based on directed labeled graph.
- Schema definition language (RDFS): Define new vocabulary (typing, inheritance of classes and properties).
- Formal semantics.

RDF formal model



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

RDF formal model



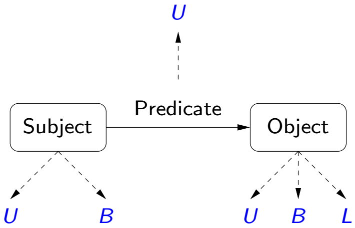
U = set of Uris

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

RDF formal model



U = set of Uris

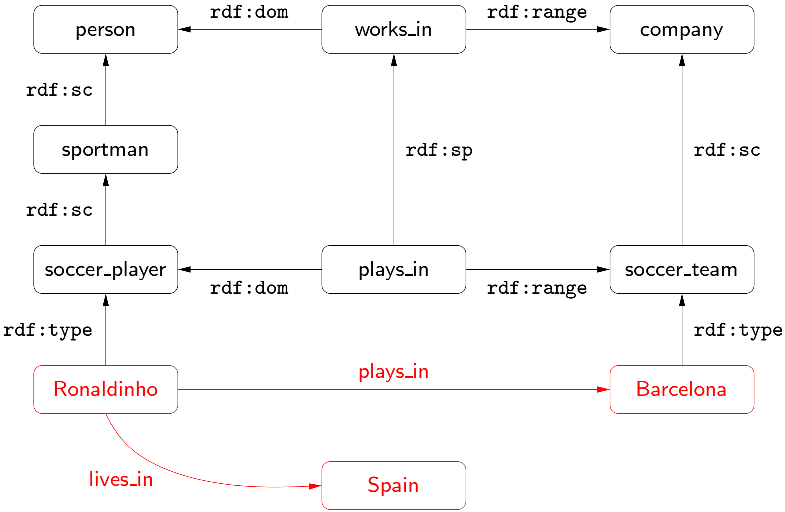
B = set of Blank nodes

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: An example



Outline

- RDF and RDFS: A brief introduction
- SPARQL: A query language for RDF
  - Formal semantics
  - Complexity of the evaluation problem
  - Optimization methods
- SPARQL as a query language for RDFS
  - Formal semantics and the closure of an RDFS graph
- NAV-SPARQL: A navigational query language for RDFS

Querying RDF: SPARQL

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

A simple RDF query language

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

A simple RDF query language

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

A simple RDF query language

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

A simple RDF query language

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

A simple RDF query language

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

In general, in a query we have:

$$H \longleftarrow$$

- Head: processing of some variables.

A simple RDF query language

```
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.

A simple RDF query language

```
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 semantics for SPARQL

A formal approach would be beneficial for:

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

A formal semantics for SPARQL

A formal approach would be beneficial for:

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

In this presentation:

- A formal compositional semantics for SPARQL
- A study of the complexity of evaluating SPARQL
- Optimization procedures

A standard algebraic syntax

- Triple patterns: just triples + variables, without blanks

`?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 ) }`	$(P_1 \text{ FILTER } R)$

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.

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.

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

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

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

	?X	?Y
μ_1:	R_1	john
μ_2:	R_2	paul

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

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 table below

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

The semantics of triple patterns

Given an RDF graph and a triple pattern t

Definition

The evaluation of t is the set of mappings that

- make t to match the graph
- have as domain the variables in t.

Example

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

	?X	?Y
μ_1:	R_1	john
μ_2:	R_2	paul

Compatible mappings

Definition

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

Example

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

Compatible mappings

Definition

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

Example

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

Compatible mappings

Definition

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

Example

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

Compatible mappings

Definition

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

Example

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

Compatible mappings

Definition

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

Example

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

Compatible mappings

Definition

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

Example

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

- μ_2 and μ_3 are not compatible

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \smallsetminus M_2$

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

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \smallsetminus M_2$

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

Union: $M_1 \cup M_2$

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

Sets of mappings and operations

Let M_1 and M_2 be sets of mappings:

Definition

Join: $M_1 \bowtie M_2$

- extending mappings in M_1 with compatible mappings in M_2

Difference: $M_1 \smallsetminus M_2$

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

Union: $M_1 \cup M_2$

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

Definition

Left Outer Join: $M_1 ⟕ M_2 = (M_1 \bowtie M_2) \cup (M_1 \smallsetminus M_2)$

Semantics of SPARQL operators

Let M_1 and M_2 be the result of evaluating P_1 and P_2.

Definition

The evaluation of:

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

Semantics of SPARQL operators

Let M_1 and M_2 be the result of evaluating P_1 and P_2.

Definition

The evaluation of:

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

Semantics of SPARQL operators

Let M_1 and M_2 be the result of evaluating P_1 and P_2.

Definition

The evaluation of:

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

Semantics of SPARQL operators

Let M_1 and M_2 be the result of evaluating P_1 and P_2.

Definition

The evaluation of:

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

Simple example

Example

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

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

Simple example

Example

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

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

Simple example

Example

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

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

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

Simple example

Example

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

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

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

Simple example

Example

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

$((?X,$ name, $?Y)$ OPT $(?X,$ email, $?E))$

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

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

Simple example

Example

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

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

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

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

Simple example

Example

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

$((?X,$ name, $?Y)$ OPT $(?X,$ email, $?E))$

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

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

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

Simple example

Example

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

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

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

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

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

- from the Join

Simple example

Example

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

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

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

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

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

- from the Difference

Simple example

Example

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

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

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

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

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

- from the Union

Boolean filter expressions (value constraints)

In filter expressions we consider:

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

Satisfaction of value constraints

A mapping satisfies:

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

Definition

Evaluation of (P FILTER R): Set of mappings in the evaluation of P that satisfy R.

The evaluation problem

Input:

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

Question:

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

Evaluation of simple patterns is polynomial

Theorem

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

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 idea

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

Evaluation including UNION is NP-complete

Theorem

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

Evaluation including UNION is NP-complete

Theorem

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

Proof idea

- *Reduction from 3SAT.*
- $\neg$ bound *is used to encode negation.*

In general: Evaluation problem is PSPACE-complete

Theorem

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

In general: Evaluation problem is PSPACE-complete

Theorem

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

Can we efficiently evaluate SPARQL queries in practice?

In general: Evaluation problem is PSPACE-complete

Theorem

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

Can we efficiently evaluate SPARQL queries in practice?

- 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

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

PSPACE-completeness: A stronger lower bound

Theorem

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

Proof idea

- *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

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

Proof idea

- *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* OPT*s 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 : $\{(a, \texttt{tv}, 0),\ (a, \texttt{tv}, 1),\ (a, \texttt{false}, 0),\ (a, \texttt{true}, 1)\}$

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

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

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:

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

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

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

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

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

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

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

$?B_0 \mapsto 1$

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

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

P_1 $\qquad\qquad$ Q_1

$?B_0 \mapsto 1 \longrightarrow ?X_1 \mapsto 0 \longrightarrow ?X_1 \mapsto 0 \quad ?Y_1 \mapsto i \quad ?B_0 \mapsto 0$

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 $\qquad\qquad$ Q_1

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

$?B_0 \mapsto 1$

$?X_1 \mapsto 1$

PSPACE-hardness: A closer look

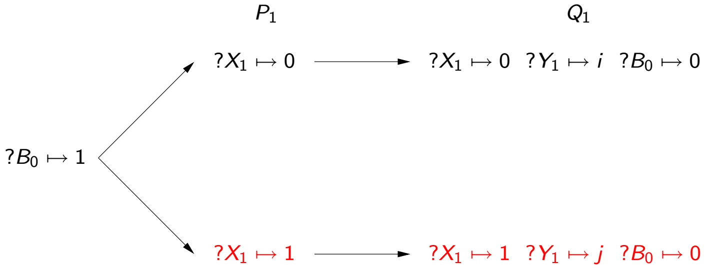
$\varphi \ : \ \forall x_1 \exists y_1 \, (x_1 \vee \neg y_1) \wedge (\neg x_1 \vee y_1)$

$P_\psi \ : \ (((a, \texttt{tv}, ?X_1) \text{ AND } (a, \texttt{tv}, ?Y_1)) \text{ FILTER}$
$\quad ((?X_1 = 1 \vee ?Y_1 = 0) \ \wedge \ (?X_1 = 0 \vee ?Y_1 = 1)))$

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

$P_1 \ : \ (a, \texttt{tv}, ?X_1)$

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



AND-FILTER-OPT fragment: Reducing the complexity

Patterns in the reduction 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 the reduction 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 the reduction are not very natural:

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

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in the reduction are not very natural:

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

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in the reduction 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 the reduction 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?

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

AND-FILTER-OPT fragment: Reducing the complexity

Patterns in the reduction 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?

- 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:

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

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:

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

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:

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

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

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:

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

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

if a variable occurs inside B and anywhere outside the OPT 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:

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

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

Example

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

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

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

Well–designed patterns

Definition

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

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

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

Example

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

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

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

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

AND-FILTER-OPT fragment: Reducing the complexity

Theorem

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

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.*

AND-FILTER-OPT fragment: Reducing the complexity

Theorem

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

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.

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.

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.

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, ..., Q_k such that:

P is constructed from Q_1, ..., 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, ..., Q_k such that:

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

Theorem

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

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

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

Patterns in OPT normal form can be evaluated more efficiently:

- AND-FILTER expressions are evaluated first, and then the results are combined using the OPT operator.

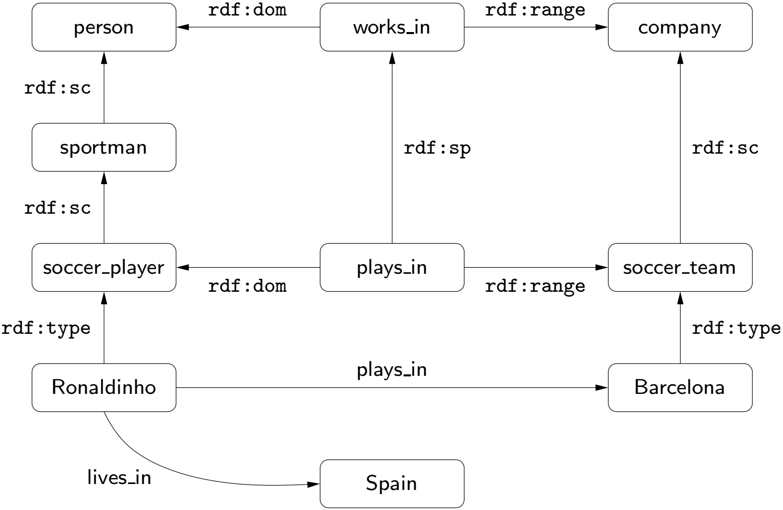
Outline

- RDF and RDFS: A brief introduction
- SPARQL: A query language for RDF
  - Formal semantics
  - Complexity of the evaluation problem
  - Optimization methods
- SPARQL as a query language for RDFS
  - Formal semantics and the closure of an RDFS graph
- NAV-SPARQL: A navigational query language for RDFS

Querying RDFS data

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



SPARQL over 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 model, interpretation, etc.

- As for the case of first-order logic.

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

Entailment in RDFS

There are inference systems characterizing the notion of entailment in RDFS:

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

...

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

SPARQL over RDFS: Closure of a graph

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

Basic step for answering queries over RDFS:

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

SPARQL over RDFS: Closure of a graph

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

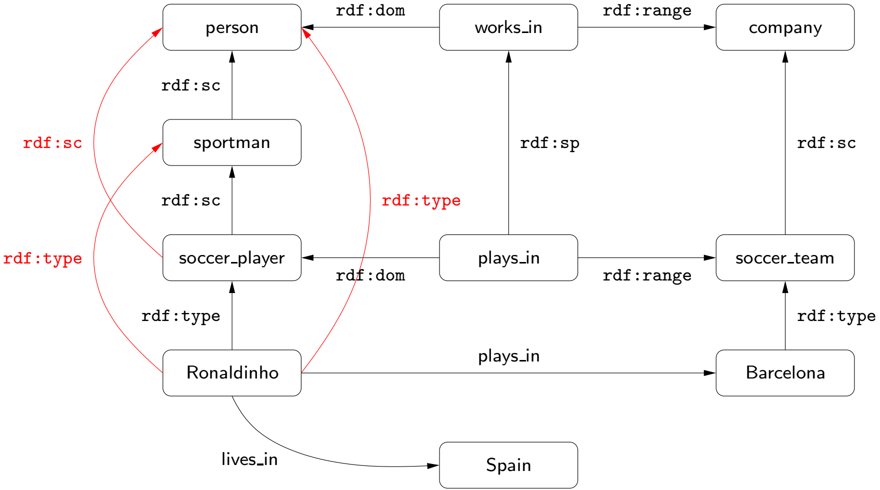
Basic step for answering queries over RDFS:

- Checking whether a tripe 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 $\mathrm{cl}(G)$.

Example: (Ronaldinho, 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 $\mathrm{cl}(G)$, and then evaluate P over $\mathrm{cl}(G)$ 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 $\mathrm{cl}(G)$, 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.

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

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

Our 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.

Our 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.

Our approach: Extend SPARQL with navigational capabilities.

- A query P over an RDFS graph G is answered by navigating G ($\text{cl}(G)$ 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.

Our 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.

Our approach: Extend SPARQL with navigational capabilities.

- A query P over an RDFS graph G is answered by navigating G ($\text{cl}(G)$ 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.

Our approach: Extend SPARQL with navigational capabilities.

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

This approach has some advantages:

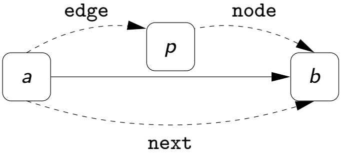
- It is goal-oriented.
- It has been used to design query languages for XML (e.g., XPath and XQuery). The results for these languages can be used here.
- Navigational operators allow to express natural queries that are not expressible in SPARQL over RDFS.

Outline

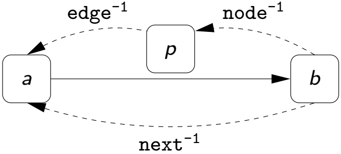
- RDF and RDFS: A brief introduction
- SPARQL: A query language for RDF
  - Formal semantics
  - Complexity of the evaluation problem
  - Optimization methods
- SPARQL as a query language for RDFS
  - Formal semantics and the closure of an RDFS graph
- NAV-SPARQL: A navigational query language for RDFS

Navigational axes

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



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



A first attempt: 0-NAV-SPARQL

Syntax of navigational expressions:

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

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

A first attempt: 0-NAV-SPARQL

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

A first attempt: 0-NAV-SPARQL

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: 0-NAV-SPARQL

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: 0-NAV-SPARQL

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

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

A first attempt: 0-NAV-SPARQL

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: 0-NAV-SPARQL

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

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

A first attempt: 0-NAV-SPARQL

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: 0-NAV-SPARQL

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

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

A first attempt: 0-NAV-SPARQL

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: 0-NAV-SPARQL

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

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

A first attempt: 0-NAV-SPARQL

Syntax of 0-NAV-SPARQL: SPARQL extended with triples of the form (x, exp, y), where exp is a navigational expression.

- Examples: (Ronaldinho, $\texttt{next::lives\_in}$, Spain) and $(?X, (\texttt{next::(rdf:sc)})^+, ?Y)$.

Semantics of 0-NAV-SPARQL: The evaluation of $t = (?X, exp, ?Y)$ over an RDFS graph G is the set of mappings μ such that

A first attempt: 0-NAV-SPARQL

Syntax of 0-NAV-SPARQL: SPARQL extended with triples of the form (x, exp, y), where exp is a navigational expression.

- Examples: (Ronaldinho, `next::lives_in`, Spain) and $(?X, (\texttt{next::(rdf:sc)})^{+}, ?Y)$.

Semantics of 0-NAV-SPARQL: The evaluation of $t = (?X, exp, ?Y)$ over an RDFS graph G is the set of mappings μ such that

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

A first attempt: 0-NAV-SPARQL

Syntax of 0-NAV-SPARQL: SPARQL extended with triples of the form (x, exp, y), where exp is a navigational expression.

- Examples: (Ronaldinho, $\texttt{next::lives\_in}$, Spain) and $(?X, (\texttt{next::(rdf:sc)})^+, ?Y)$.

Semantics of 0-NAV-SPARQL: The evaluation of $t = (?X, exp, ?Y)$ over an RDFS 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: 0-NAV-SPARQL

Syntax of 0-NAV-SPARQL: SPARQL extended with triples of the form (x, exp, y), where exp is a navigational expression.

- Examples: (Ronaldinho, `next::lives_in`, Spain) and $(?X, (\texttt{next::(rdf:sc)})^+, ?Y)$.

Semantics of 0-NAV-SPARQL: The evaluation of $t = (?X, exp, ?Y)$ over an RDFS 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: $(?X, (\texttt{next::Iberia})^+, ?Y)$ AND $(?X, (\texttt{next::AirFrance})^+, ?Y)$

Is 0-NAV-SPARQL a good language for RDFS?

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

- Can we capture SPARQL over RDFS?

Is 0-NAV-SPARQL 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 0-NAV-SPARQL pattern Q such that:

- RDFS-evaluation of P over G $=$ evaluation of Q over G.

Is 0-NAV-SPARQL a good language for RDFS?

Theorem

There is a SPARQL pattern P for which there is no 0-NAV-SPARQL pattern Q such that, for every RDFS graph G:

$$\text{RDFS-evaluation of } P \text{ over } G \quad = \quad \text{evaluation of } Q \text{ over } G,$$

Is 0-NAV-SPARQL a good language for RDFS?

Theorem

There is a SPARQL pattern P for which there is no 0-NAV-SPARQL pattern Q such that, for every RDFS graph G:

$$\textit{RDFS-evaluation of } P \textit{ over } G \quad = \quad \textit{evaluation of } Q \textit{ over } G,$$

The previous theorem holds even for $P = (?X, a, ?Y)$, where a is an arbitrary element in U.

Is 0-NAV-SPARQL a good language for RDFS?

Theorem

There is a SPARQL pattern P for which there is no 0-NAV-SPARQL pattern Q such that, for every RDFS graph G:

$$\textit{RDFS-evaluation of } P \textit{ over } G \quad = \quad \textit{evaluation of } Q \textit{ over } G,$$

The previous theorem holds even for $P = (?X, a, ?Y)$, where a is an arbitrary element in U.

How can we capture SPARQL over RDFS?

Is 0-NAV-SPARQL a good language for RDFS?

Theorem

There is a SPARQL pattern P for which there is no 0-NAV-SPARQL pattern Q such that, for every RDFS graph G:

$$\text{RDFS-evaluation of } P \text{ over } G \quad = \quad \text{evaluation of } Q \text{ over } G,$$

The previous theorem holds even for $P = (?X, a, ?Y)$, where a is an arbitrary element in U.

How can we capture SPARQL over RDFS?

- We adopt the notion of branching from XPath.

A successful attempt: NAV-SPARQL

Syntax of navigational expressions:

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

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

A successful attempt: NAV-SPARQL

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

A successful attempt: NAV-SPARQL

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

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

A successful attempt: NAV-SPARQL

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

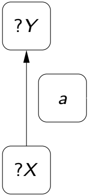
$$
\begin{aligned}
[\![\texttt{next}::[exp]]\!]_G &= \{(x, y) \mid \exists z, w \in U \ \ (x, z, y) \in G \text{ and} \\
&\qquad\qquad\qquad\qquad\qquad\qquad (z, w) \in [\![exp]\!]_G\} \\
[\![\texttt{edge}::[exp]]\!]_G &= \{(x, y) \mid \exists z, w \in U \ \ (x, y, z) \in G \text{ and} \\
&\qquad\qquad\qquad\qquad\qquad\qquad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

NAV-SPARQL: Capturing SPARQL over RDFS

Example: $(?X, a, ?Y)$ over RDFS is equivalent to NAV-SPARQL pattern $(?X, \texttt{next}{::}[(\texttt{next}{::}(\texttt{rdf:sp}))^*/(\texttt{self}{::}a)], ?Y)$.

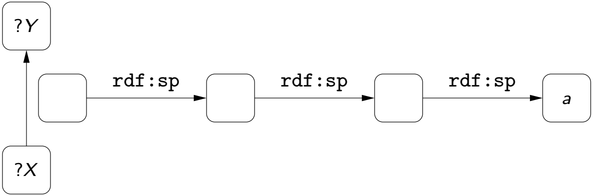
NAV-SPARQL: Capturing SPARQL over RDFS

Example: $(?X, a, ?Y)$ over RDFS is equivalent to NAV-SPARQL pattern $(?X, \mathtt{next}{::}[(\mathtt{next}{::}(\mathtt{rdf:sp}))^*/(\mathtt{self}{::}a)], ?Y)$.



NAV-SPARQL: Capturing SPARQL over RDFS

Example: $(?X, a, ?Y)$ over RDFS is equivalent to NAV-SPARQL pattern $(?X, \texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::a)], ?Y)$.



NAV-SPARQL: Capturing SPARQL over RDFS

Theorem

For every SPARQL pattern P, there exists a NAV-SPARQL pattern Q such that, for every RDFS graph G:

$$\text{RDFS-evaluation of } P \text{ over } G \quad = \quad \text{evaluation of } Q \text{ over } G,$$

NAV-SPARQL: Capturing SPARQL over RDFS

Theorem

For every SPARQL pattern P, there exists a NAV-SPARQL pattern Q such that, for every RDFS graph G:

$$\textit{RDFS-evaluation of } P \textit{ over } G \quad = \quad \textit{evaluation of } Q \textit{ over } G,$$

Proof idea

Replace $(?X, a, ?Y)$ *by* $(?X, R(a), ?Y)$, *where:*

$$\begin{array}{rcl} R(\mathtt{rdf:sc}) & = & (\mathtt{next}::(\mathtt{rdf:sc}))^+ \\ R(\mathtt{rdf:sp}) & = & (\mathtt{next}::(\mathtt{rdf:sp}))^+ \\ & \cdots & \\ R(b) & = & \mathtt{next}::[(\mathtt{next}::(\mathtt{rdf:sp}))^* / (\mathtt{self}::b)] \end{array}$$

NAV-SPARQL: Capturing SPARQL over RDFS

Theorem

For every SPARQL pattern P, there exists a NAV-SPARQL pattern Q such that, for every RDFS graph G:

$$\textit{RDFS-evaluation of } P \textit{ over } G \quad = \quad \textit{evaluation of } Q \textit{ over } G,$$

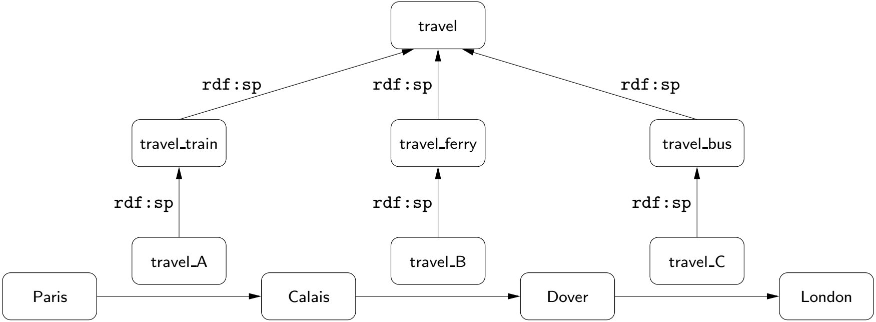
Proof idea

Replace $(?X, a, ?Y)$ *by* $(?X, R(a), ?Y)$, *where:*

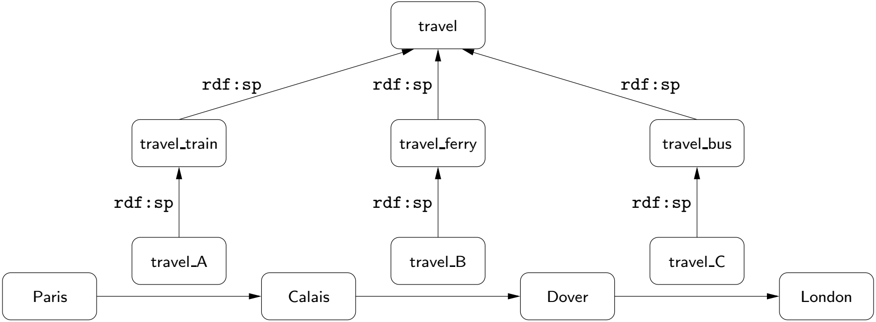
$$
\begin{aligned}
R(\mathtt{rdf:sc}) &= (\mathtt{next}::(\mathtt{rdf:sc}))^+ \\
R(\mathtt{rdf:sp}) &= (\mathtt{next}::(\mathtt{rdf:sp}))^+ \\
&\cdots \\
R(b) &= \mathtt{next}::[(\mathtt{next}::(\mathtt{rdf:sp}))^*/(\mathtt{self}::b)]
\end{aligned}
$$

Note: $R(\mathtt{rdf:type})$ *uses* next, edge *and* $\mathtt{node}^{-1}$.

The extra expressive power of NAV-SPARQL

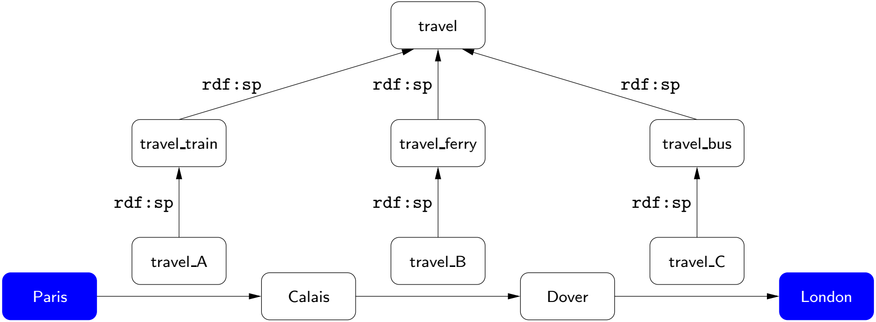


The extra expressive power of NAV-SPARQL



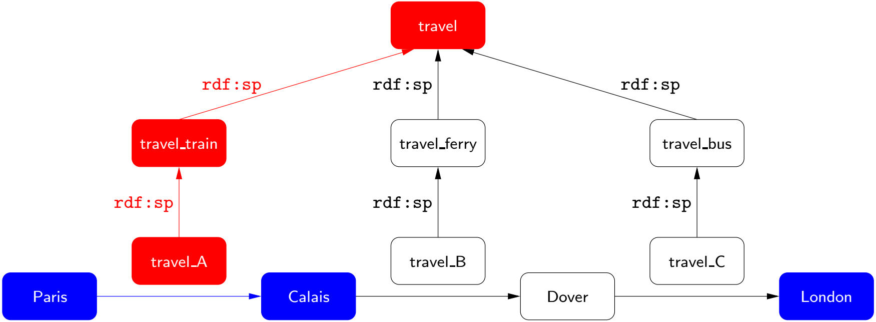
A natural query: $(?X, (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::\text{travel})])^+, ?Y)$

The extra expressive power of NAV-SPARQL



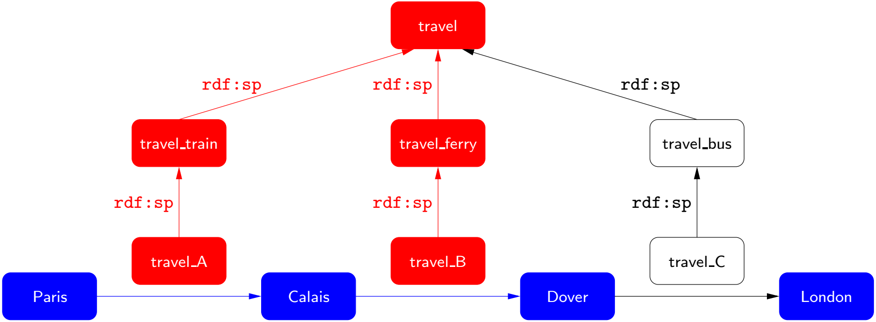
A natural query: $(?X,\ (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^{*}/(\texttt{self}::\text{travel})])^{+},\ ?Y)$

The extra expressive power of NAV-SPARQL



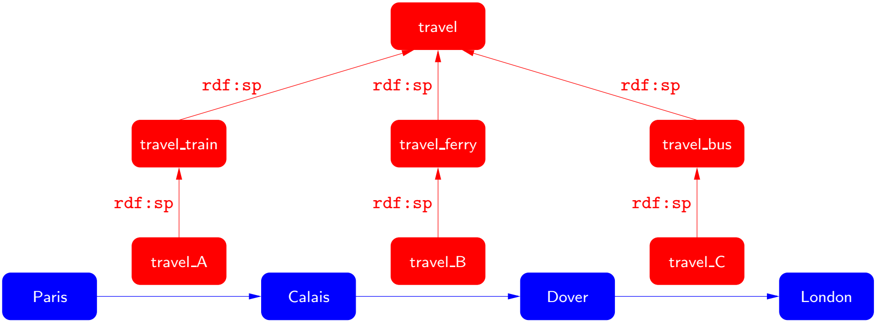
A natural query: $(?X, (\mathtt{next}::[(\mathtt{next}::(\mathtt{rdf:sp}))^{*}/(\mathtt{self}::\mathrm{travel})])^{+}, ?Y)$

The extra expressive power of NAV-SPARQL



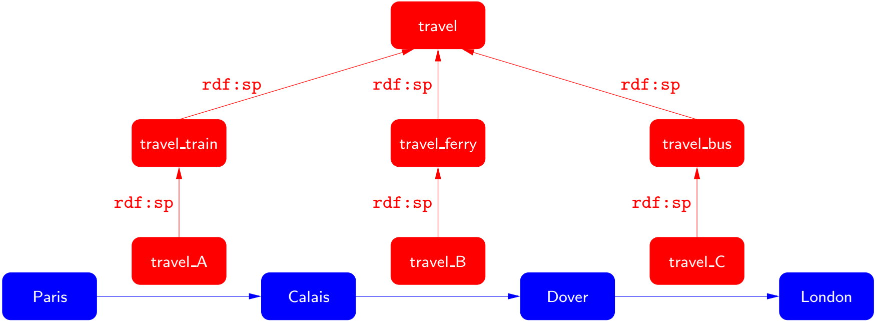
A natural query: $(?X,\ (\texttt{next}::[(\texttt{next}::(\texttt{rdf:sp}))^*/(\texttt{self}::\text{travel})])^+,\ ?Y)$

The extra expressive power of NAV-SPARQL



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

The extra expressive power of NAV-SPARQL



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.

Ongoing work

Ongoing work

- Implementation of SPARQL.

Ongoing work

- Implementation of SPARQL.
  - How useful are the optimization rules in practice?

Ongoing work

- Implementation of SPARQL.
  - How useful are the optimization rules in practice?
- Implementation of NAV-SPARQL.

Ongoing work

- Implementation of SPARQL.
  - How useful are the optimization rules in practice?
- Implementation of NAV-SPARQL.
  - Can this language be implemented efficiently?

Ongoing work

- Implementation of SPARQL.
  - How useful are the optimization rules in practice?
- Implementation of NAV-SPARQL.
  - Can this language be implemented efficiently? Can this language be used over large RDFS graphs?

Ongoing work

- Implementation of SPARQL.
  - How useful are the optimization rules in practice?
- Implementation of NAV-SPARQL.
  - Can this language be implemented efficiently? Can this language be used over large RDFS graphs?
  - Is the extra expressive power of NAV-SPARQL useful in practice?

Ongoing work

- Implementation of SPARQL.
  - How useful are the optimization rules in practice?
- Implementation of NAV-SPARQL.
  - Can this language be implemented efficiently? Can this language be used over large RDFS graphs?
  - Is the extra expressive power of NAV-SPARQL useful in practice?
  - Is there a fragment of NAV-SPARQL which is also appropriate for RDFS?

Ongoing work

- Implementation of SPARQL.
  - How useful are the optimization rules in practice?
- Implementation of NAV-SPARQL.
  - Can this language be implemented efficiently? Can this language be used over large RDFS graphs?
  - Is the extra expressive power of NAV-SPARQL useful in practice?
  - Is there a fragment of NAV-SPARQL which is also appropriate for RDFS? One level of nesting is enough to capture SPARQL over RDFS.