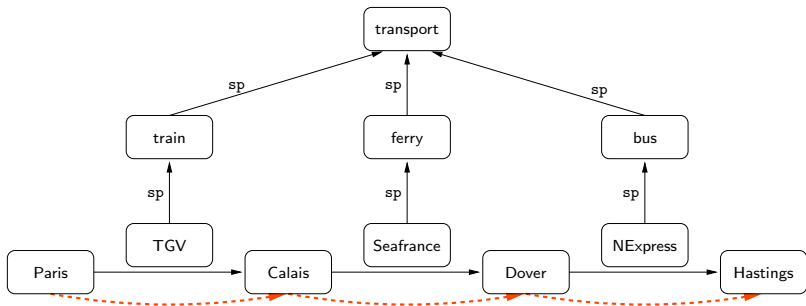
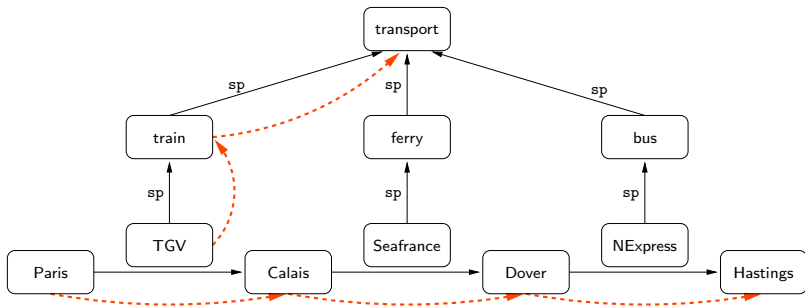


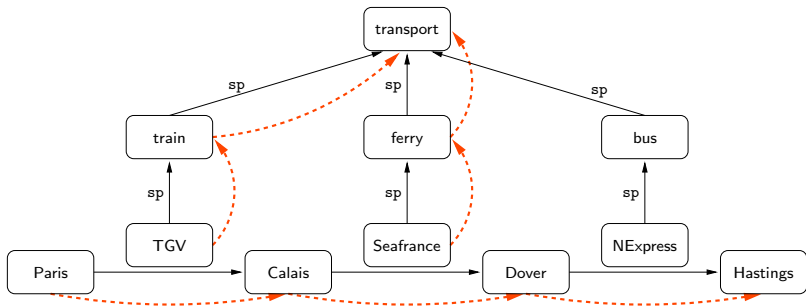
Are Paris and Hastings connected  
by a sequence of transportation services?



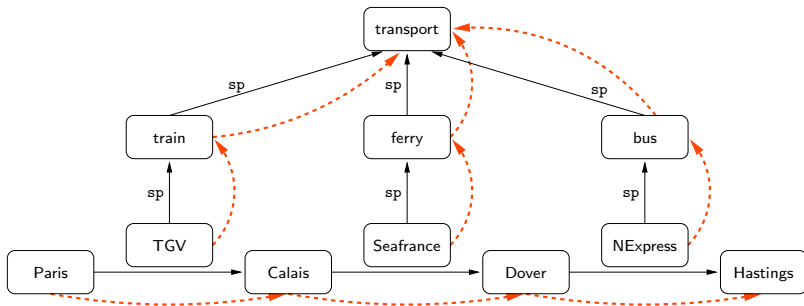
Are Paris and Hastings connected  
by a sequence of transportation services?



Are Paris and Hastings connected  
by a sequence of transportation services?



Are Paris and Hastings connected  
by a sequence of transportation services?



Are Paris and Hastings connected  
by a sequence of transportation services?

# nSPARQL: A Navigational Language for RDF

Jorge Pérez   Marcelo Arenas   Claudio Gutierrez

Departamento de Ciencia de la Computación  
Pontificia Universidad Católica de Chile  
Universidad de Chile

*Khipu group*  
<http://www.khipu.cl>

# Our contributions

- ▶ define a navigational language for RDF
- ▶ show that it can be efficiently evaluated



# Our contributions

- ▶ define a navigational language for RDF
- ▶ show that it can be efficiently evaluated

We incorporate these features in SPARQL

- ▶ show that the language can express interesting RDFS queries
- ▶ study the expressive power of this language

# Outline

## Nested regular expressions

- Syntax and semantics

- Complexity

## nSPARQL

- Syntax and semantics

- nSPARQL and RDFS evaluation

- Expressiveness

## Concluding remarks

# Outline

## Nested regular expressions

Syntax and semantics

Complexity

## nSPARQL

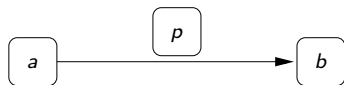
Syntax and semantics

nSPARQL and RDFS evaluation

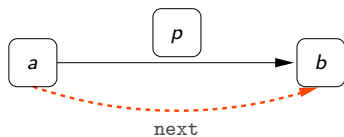
Expressiveness

## Concluding remarks

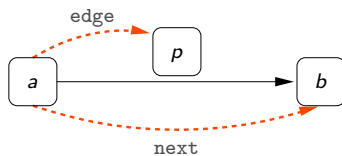
## *Navigational axes* for RDF triple $(a, p, b)$



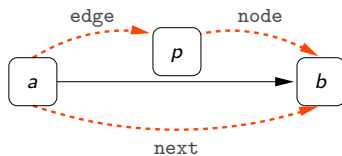
# *Navigational axes for RDF triple (a, p, b)*



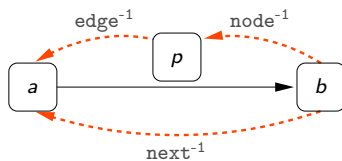
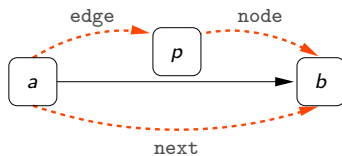
# *Navigational axes for RDF triple $(a, p, b)$*



## *Navigational axes for RDF triple $(a, p, b)$*

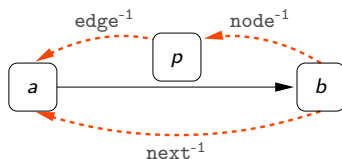
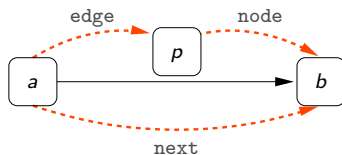


# *Navigational axes for RDF triple (a, p, b)*



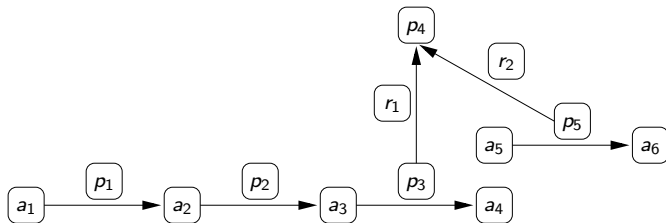


# *Navigational axes for RDF triple (a, p, b)*

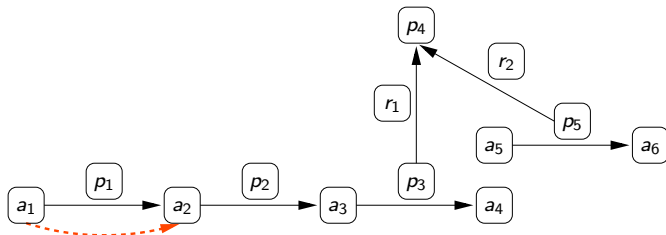


plus self axis

A sequence of axes defines a path in an RDF graph.

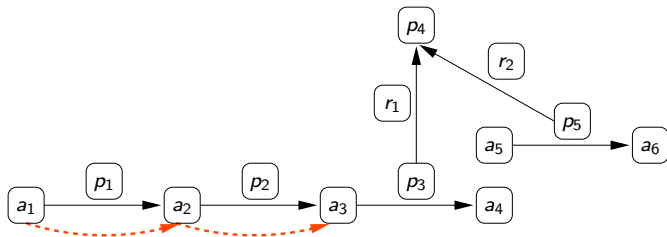


A sequence of axes defines a path in an RDF graph.



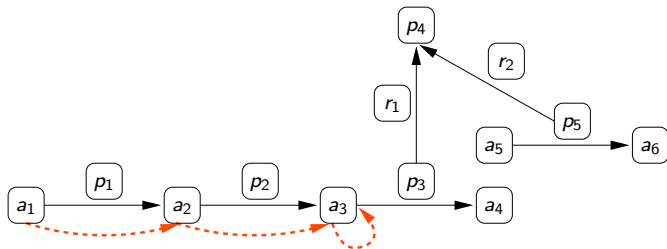
next

A sequence of axes defines a path in an RDF graph.



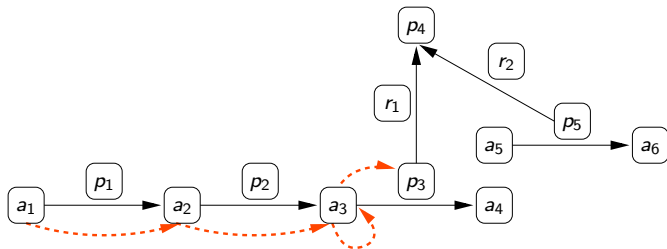
next/next

A sequence of axes defines a path in an RDF graph.



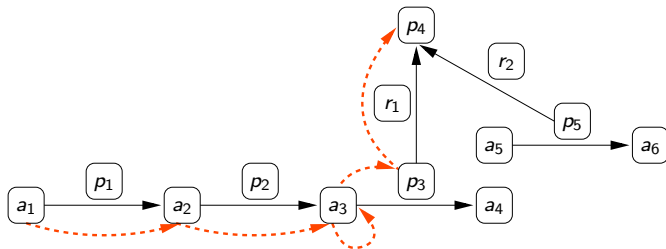
`next/next/self`

A sequence of axes defines a path in an RDF graph.



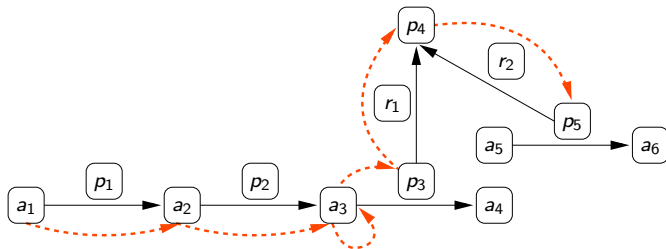
`next/next/self/edge`

A sequence of axes defines a path in an RDF graph.



next/next/self/edge/next

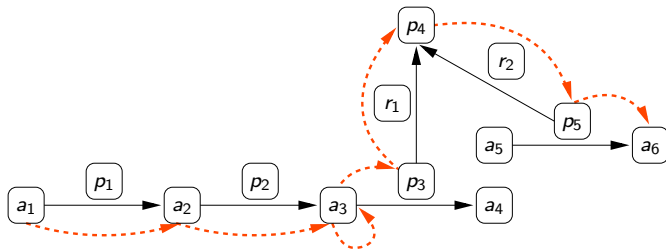
A sequence of axes defines a path in an RDF graph.



`next/next/self/edge/next/next-1`



A sequence of axes defines a path in an RDF graph.



`next/next/self/edge/next/next-1/node`

# Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

*exp* :=

# Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$exp := axis$

# Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$exp := axis$

- ▶  $axis \in \{self, next, next^{-1}, edge, edge^{-1}, node, node^{-1}\}$

# Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$$\text{exp} ::= \text{axis} \mid \text{axis}::a$$

▶  $\text{axis} \in \{\text{self}, \text{next}, \text{next}^{-1}, \text{edge}, \text{edge}^{-1}, \text{node}, \text{node}^{-1}\}$

# Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$exp := axis \mid axis::a$

- ▶  $axis \in \{self, next, next^{-1}, edge, edge^{-1}, node, node^{-1}\}$
- ▶  $a$  is a URI

# Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$$exp := axis \mid axis::a \mid exp/exp \mid exp|exp \mid exp^*$$

- ▶  $axis \in \{self, next, next^{-1}, edge, edge^{-1}, node, node^{-1}\}$
- ▶  $a$  is a URI

# Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$exp := axis \mid axis::a \mid exp/exp \mid exp|exp \mid exp^* \mid axis::[exp]$

- ▶  $axis \in \{self, next, next^{-1}, edge, edge^{-1}, node, node^{-1}\}$
- ▶  $a$  is a URI



# Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$$\text{exp} ::= \text{axis} \mid \text{axis}::a \mid \text{exp}/\text{exp} \mid \text{exp}|\text{exp} \mid \text{exp}^* \mid \text{axis}::[\text{exp}]$$

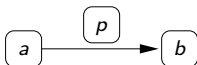
- ▶  $\text{axis} \in \{\text{self}, \text{next}, \text{next}^{-1}, \text{edge}, \text{edge}^{-1}, \text{node}, \text{node}^{-1}\}$
- ▶  $a$  is a URI
- ▶  $[\text{exp}]$  is a *nesting* expression

# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .

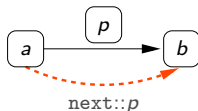
# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



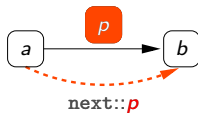
# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



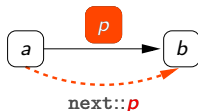
# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



# Semantics of nested regular expressions over RDF

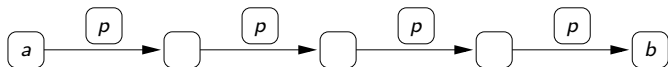
The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



$(a, b)$  is in the evaluation of `next::p`

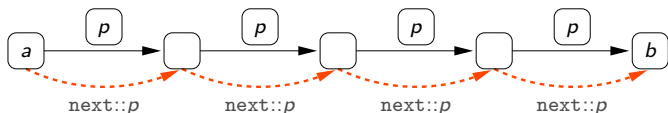
# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



# Semantics of nested regular expressions over RDF

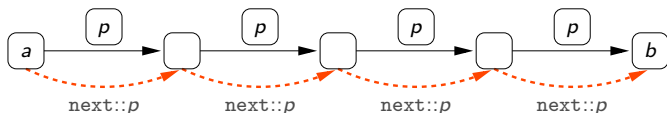
The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .





# Semantics of nested regular expressions over RDF

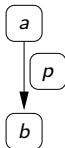
The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



$(a, b)$  is in the evaluation of  $(next::p)^*$

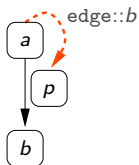
# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



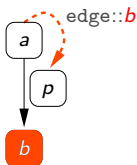
# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



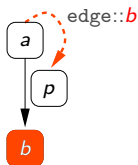
# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



# Semantics of nested regular expressions over RDF

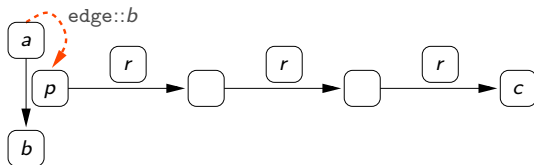
The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



$(a, p)$  is in the evaluation of  $edge::b$

# Semantics of nested regular expressions over RDF

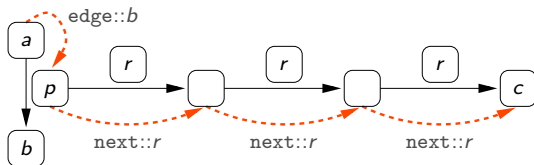
The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



$(a, p)$  is in the evaluation of  $edge::b$

# Semantics of nested regular expressions over RDF

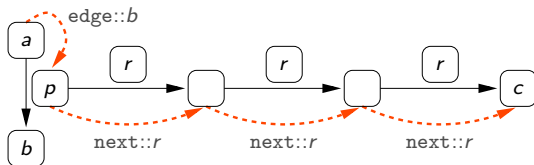
The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



$(a, p)$  is in the evaluation of  $edge::b$

# Semantics of nested regular expressions over RDF

The *evaluation* of an expression  $exp$  is a set of pairs  $(a, b)$  of nodes such that  $b$  is reachable from  $a$  by following  $exp$ .



$(a, p)$  is in the evaluation of  $edge::b$

$(a, c)$  is in the evaluation of  $edge::b/(next::r)^*$



# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

$$\llbracket \text{self} \rrbracket_G = \{(x, x) \mid x \in \text{voc}(G)\}$$

# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

$$\begin{aligned}\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G)\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z, y) \in G\}\end{aligned}$$

# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

$$\begin{aligned}\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G)\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z, y) \in G\} \\ \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\}\end{aligned}$$

# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

$$\begin{aligned}\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G)\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z, y) \in G\} \\ \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\} \\ \llbracket \text{next}::a \rrbracket_G &= \{(x, y) \mid (x, a, y) \in G\}\end{aligned}$$

# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

$$\begin{aligned}\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G)\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z, y) \in G\} \\ \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\} \\ \llbracket \text{next}::a \rrbracket_G &= \{(x, y) \mid (x, a, y) \in G\} \\ \llbracket exp_1/exp_2 \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z) \in \llbracket exp_1 \rrbracket_G \text{ and} \\ &\quad (z, y) \in \llbracket exp_2 \rrbracket_G\}\end{aligned}$$

# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

$$\begin{aligned}\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G)\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z, y) \in G\} \\ \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\} \\ \llbracket \text{next}::a \rrbracket_G &= \{(x, y) \mid (x, a, y) \in G\} \\ \llbracket exp_1/exp_2 \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z) \in \llbracket exp_1 \rrbracket_G \text{ and} \\ &\quad (z, y) \in \llbracket exp_2 \rrbracket_G\} \\ \llbracket exp_1|exp_2 \rrbracket_G &= \llbracket exp_1 \rrbracket_G \cup \llbracket exp_2 \rrbracket_G\end{aligned}$$

# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

$$\begin{aligned}\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G)\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z, y) \in G\} \\ \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\} \\ \llbracket \text{next}::a \rrbracket_G &= \{(x, y) \mid (x, a, y) \in G\} \\ \llbracket exp_1/exp_2 \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z) \in \llbracket exp_1 \rrbracket_G \text{ and} \\ &\quad (z, y) \in \llbracket exp_2 \rrbracket_G\} \\ \llbracket exp_1|exp_2 \rrbracket_G &= \llbracket exp_1 \rrbracket_G \cup \llbracket exp_2 \rrbracket_G \\ \llbracket exp^* \rrbracket_G &= \llbracket \text{self} \rrbracket_G \cup \llbracket exp \rrbracket_G \cup \llbracket exp/exp \rrbracket_G \cup \\ &\quad \llbracket exp/exp/exp \rrbracket_G \cup \dots\end{aligned}$$



# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

$$\begin{aligned}\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G)\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z, y) \in G\} \\ \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\} \\ \llbracket \text{next}::a \rrbracket_G &= \{(x, y) \mid (x, a, y) \in G\} \\ \llbracket exp_1/exp_2 \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z) \in \llbracket exp_1 \rrbracket_G \text{ and} \\ &\quad (z, y) \in \llbracket exp_2 \rrbracket_G\} \\ \llbracket exp_1|exp_2 \rrbracket_G &= \llbracket exp_1 \rrbracket_G \cup \llbracket exp_2 \rrbracket_G \\ \llbracket exp^* \rrbracket_G &= \llbracket \text{self} \rrbracket_G \cup \llbracket exp \rrbracket_G \cup \llbracket exp/exp \rrbracket_G \cup \\ &\quad \llbracket exp/exp/exp \rrbracket_G \cup \dots \\ &\dots\end{aligned}$$

# Semantics of nested regular expressions over RDF graphs

## Definition

The *evaluation* of expression  $exp$  over an RDF graph  $G$ , denoted by  $\llbracket exp \rrbracket_G$ , is defined recursively as a *binary relation*:

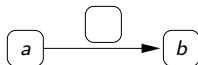
$$\begin{aligned}\llbracket \text{self} \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G)\} \\ \llbracket \text{next} \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z, y) \in G\} \\ \llbracket \text{self}::a \rrbracket_G &= \{(a, a)\} \\ \llbracket \text{next}::a \rrbracket_G &= \{(x, y) \mid (x, a, y) \in G\} \\ \llbracket exp_1/exp_2 \rrbracket_G &= \{(x, y) \mid \text{exists } z \text{ s.t. } (x, z) \in \llbracket exp_1 \rrbracket_G \text{ and} \\ &\quad (z, y) \in \llbracket exp_2 \rrbracket_G\} \\ \llbracket exp_1|exp_2 \rrbracket_G &= \llbracket exp_1 \rrbracket_G \cup \llbracket exp_2 \rrbracket_G \\ \llbracket exp^* \rrbracket_G &= \llbracket \text{self} \rrbracket_G \cup \llbracket exp \rrbracket_G \cup \llbracket exp/exp \rrbracket_G \cup \\ &\quad \llbracket exp/exp/exp \rrbracket_G \cup \dots \\ &\dots\end{aligned}$$

$$(exp^+ \equiv exp/exp^*)$$

# Nesting allows for existential checking.

## Intuition:

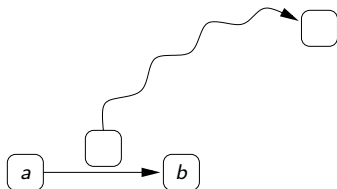
`next::[exp]` navigates from node *a* to node *b* if they are connected with an edge that satisfies a path condition given by *exp*.



# Nesting allows for existential checking.

## Intuition:

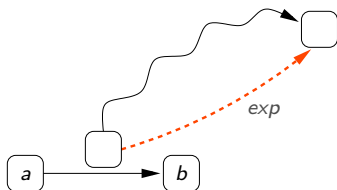
`next::[exp]` navigates from node *a* to node *b* if they are connected with an edge that satisfies a path condition given by *exp*.



# Nesting allows for existential checking.

## Intuition:

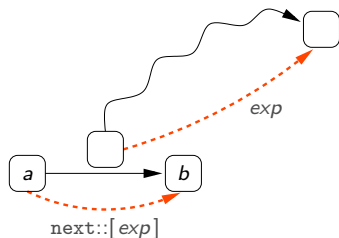
$\text{next}::[\text{exp}]$  navigates from node  $a$  to node  $b$  if they are connected with an edge that satisfies a path condition given by  $\text{exp}$ .



# Nesting allows for existential checking.

## Intuition:

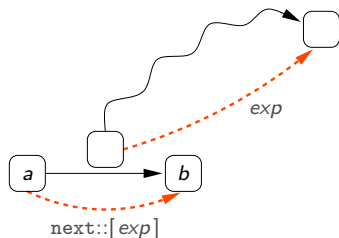
$\text{next}::[\text{exp}]$  navigates from node  $a$  to node  $b$  if they are connected with an edge that satisfies a path condition given by  $\text{exp}$ .



# Nesting allows for existential checking.

## Intuition:

$\text{next}::[\text{exp}]$  navigates from node  $a$  to node  $b$  if they are connected with an edge that satisfies a path condition given by  $\text{exp}$ .

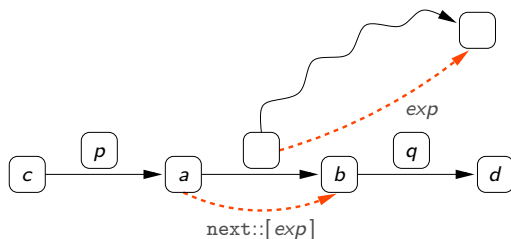


$(a, b)$  is in the evaluation of  $\text{next}::[\text{exp}]$

# Nesting allows for existential checking.

## Intuition:

$\text{next}::[\text{exp}]$  navigates from node  $a$  to node  $b$  if they are connected with an edge that satisfies a path condition given by  $\text{exp}$ .



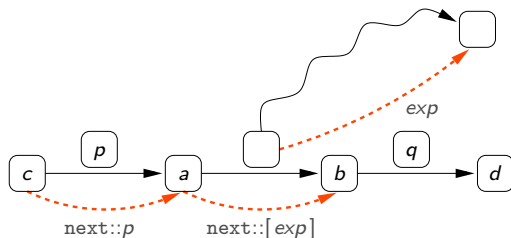
$(a, b)$  is in the evaluation of  $\text{next}::[\text{exp}]$



# Nesting allows for existential checking.

## Intuition:

$\text{next}::[\text{exp}]$  navigates from node  $a$  to node  $b$  if they are connected with an edge that satisfies a path condition given by  $\text{exp}$ .

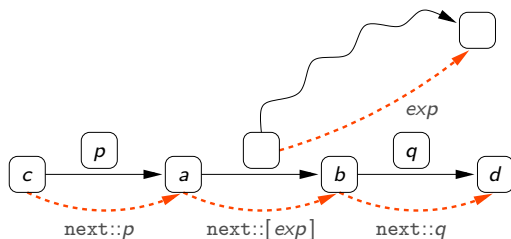


$(a, b)$  is in the evaluation of  $\text{next}::[\text{exp}]$

# Nesting allows for existential checking.

## Intuition:

$\text{next}::[\text{exp}]$  navigates from node  $a$  to node  $b$  if they are connected with an edge that satisfies a path condition given by  $\text{exp}$ .

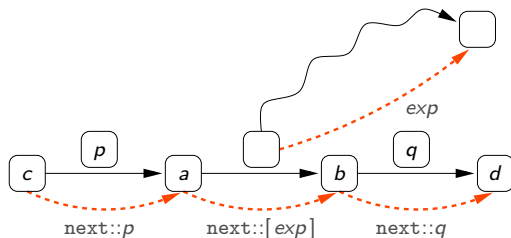


$(a, b)$  is in the evaluation of  $\text{next}::[\text{exp}]$

# Nesting allows for existential checking.

## Intuition:

$\text{next}::[\text{exp}]$  navigates from node  $a$  to node  $b$  if they are connected with an edge that satisfies a path condition given by  $\text{exp}$ .



$(a, b)$  is in the evaluation of  $\text{next}::[\text{exp}]$

$(c, d)$  is in the evaluation of  $\text{next}::p/\text{next}::[\text{exp}]/\text{next}::q$

# Semantics of nested regular expressions over RDF

## Definition

The *evaluation* of expression  $axis::[exp]$  over an RDF graph  $G$  is defined as follows:

# Semantics of nested regular expressions over RDF

## Definition

The *evaluation* of expression `axis::[exp]` over an RDF graph  $G$  is defined as follows:

$$\llbracket \text{self}::[\text{exp}] \rrbracket_G = \{(x, x) \mid x \in \text{voc}(G) \text{ and exists } z \text{ s.t.} \\ (x, z) \in \llbracket \text{exp} \rrbracket_G\}$$

# Semantics of nested regular expressions over RDF

## Definition

The *evaluation* of expression `axis::[exp]` over an RDF graph  $G$  is defined as follows:

$$\begin{aligned} \llbracket \text{self}::[\text{exp}] \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G) \text{ and exists } z \text{ s.t.} \\ &\quad (x, z) \in \llbracket \text{exp} \rrbracket_G\} \\ \llbracket \text{next}::[\text{exp}] \rrbracket_G &= \{(x, y) \mid \text{exist } z, w \text{ s.t. } (x, z, y) \in G \text{ and} \\ &\quad (z, w) \in \llbracket \text{exp} \rrbracket_G\} \end{aligned}$$

# Semantics of nested regular expressions over RDF

## Definition

The *evaluation* of expression  $\text{axis}::[\text{exp}]$  over an RDF graph  $G$  is defined as follows:

$$\begin{aligned} \llbracket \text{self}::[\text{exp}] \rrbracket_G &= \{(x, x) \mid x \in \text{voc}(G) \text{ and exists } z \text{ s.t.} \\ &\quad (x, z) \in \llbracket \text{exp} \rrbracket_G\} \\ \llbracket \text{next}::[\text{exp}] \rrbracket_G &= \{(x, y) \mid \text{exist } z, w \text{ s.t. } (x, z, y) \in G \text{ and} \\ &\quad (z, w) \in \llbracket \text{exp} \rrbracket_G\} \\ &\dots \end{aligned}$$

# Outline

## Nested regular expressions

Syntax and semantics

Complexity

## nSPARQL

Syntax and semantics

nSPARQL and RDFS evaluation

Expressiveness

## Concluding remarks



# The evaluation problem for nested regular expressions

We consider the associated decision problem:

## Input

- ▶ RDF graph  $G$
- ▶ nested regular expression  $exp$
- ▶ a pair of elements  $(a, b)$

# The evaluation problem for nested regular expressions

We consider the associated decision problem:

## Input

- ▶ RDF graph  $G$
- ▶ nested regular expression  $exp$
- ▶ a pair of elements  $(a, b)$

## Question

Is  $(a, b)$  in the evaluation of  $exp$  over  $G$ ?

# The evaluation problem for nested regular expressions

We consider the associated decision problem:

## Input

- ▶ RDF graph  $G$
- ▶ nested regular expression  $exp$
- ▶ a pair of elements  $(a, b)$

## Question

Is  $(a, b)$  in the evaluation of  $exp$  over  $G$ ?

$$(a, b) \in \llbracket exp \rrbracket_G?$$

The evaluation problem for nested regular expressions can be decided in *linear time*.

### Theorem

Given a nested regular expression  $exp$  and an RDF graph  $G$ , testing whether a pair  $(a, b)$  is in  $\llbracket exp \rrbracket_G$  can be done in time

$$O(|G| \cdot |exp|).$$

The evaluation problem for nested regular expressions can be decided in *linear time*.

### Theorem

Given a nested regular expression  $exp$  and an RDF graph  $G$ , testing whether a pair  $(a, b)$  is in  $\llbracket exp \rrbracket_G$  can be done in time

$$O(|G| \cdot |exp|).$$

Algorithm (in the style of *temporal logic*)

The evaluation problem for nested regular expressions can be decided in *linear time*.

### Theorem

Given a nested regular expression  $exp$  and an RDF graph  $G$ , testing whether a pair  $(a, b)$  is in  $\llbracket exp \rrbracket_G$  can be done in time

$$O(|G| \cdot |exp|).$$

Algorithm (in the style of *temporal logic*)

- ▶ Recursively consider the nested subexpressions of  $exp$ ,

The evaluation problem for nested regular expressions can be decided in *linear time*.

### Theorem

Given a nested regular expression  $exp$  and an RDF graph  $G$ , testing whether a pair  $(a, b)$  is in  $\llbracket exp \rrbracket_G$  can be done in time

$$O(|G| \cdot |exp|).$$

Algorithm (in the style of *temporal logic*)

- ▶ Recursively consider the nested subexpressions of  $exp$ ,
- ▶ *mark* nodes according to what subexpressions they satisfy,

The evaluation problem for nested regular expressions can be decided in *linear time*.

### Theorem

Given a nested regular expression  $exp$  and an RDF graph  $G$ , testing whether a pair  $(a, b)$  is in  $\llbracket exp \rrbracket_G$  can be done in time

$$O(|G| \cdot |exp|).$$

Algorithm (in the style of *temporal logic*)

- ▶ Recursively consider the nested subexpressions of  $exp$ ,
- ▶ *mark* nodes according to what subexpressions they satisfy,
- ▶ check if  $b$  is reachable from  $a$  following  $exp$ .



The evaluation problem for nested regular expressions can be decided in *linear time*.

### Theorem

Given a nested regular expression  $exp$  and an RDF graph  $G$ , testing whether a pair  $(a, b)$  is in  $\llbracket exp \rrbracket_G$  can be done in time

$$O(|G| \cdot |exp|).$$

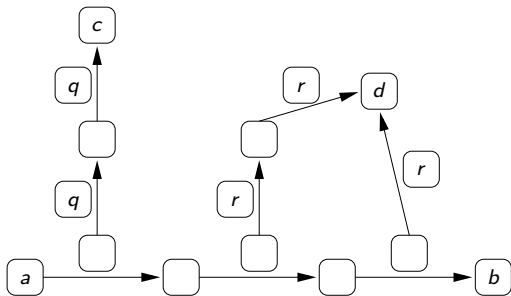
Algorithm (in the style of *temporal logic*)

- ▶ Recursively consider the nested subexpressions of  $exp$ ,
- ▶ *mark* nodes according to what subexpressions they satisfy,
- ▶ check if  $b$  is reachable from  $a$  following  $exp$ .

All can be done in linear time.

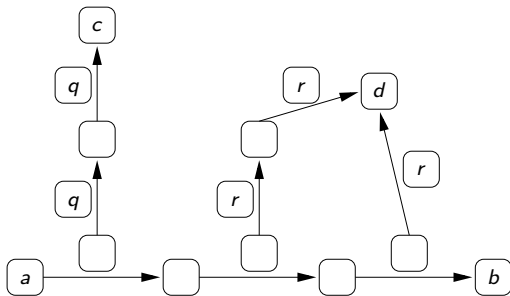
## A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$



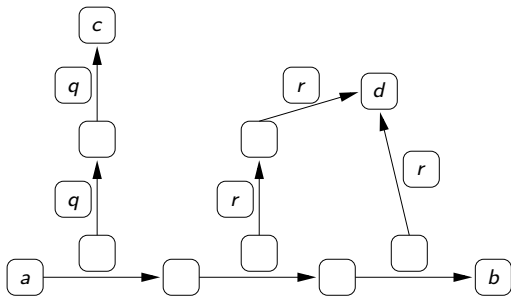
# A closer look at the algorithm

```
exp = next::[(next::q)*self::c]/(next::[(next::r)*self::d])*
```



# A closer look at the algorithm

$exp = next::[(next::q)^*/self::c]/(next::[(next::r)^*/self::d])^*$

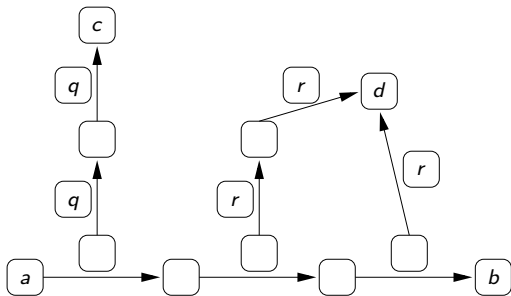


$$\alpha = (next::q)^*/self::c$$

$$\beta = (next::r)^*/self::d$$

## A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$

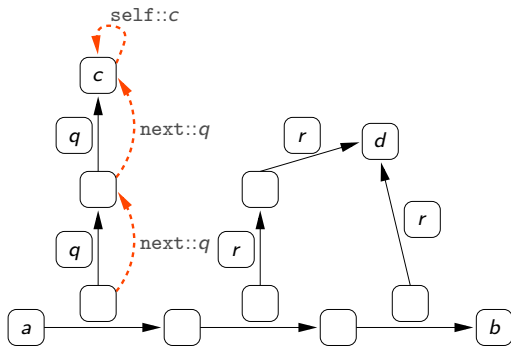


$$\alpha = (\text{next}::q)^*/\text{self}::c$$

$$\beta = (\text{next}::r)^*/\text{self}::d$$

# A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$

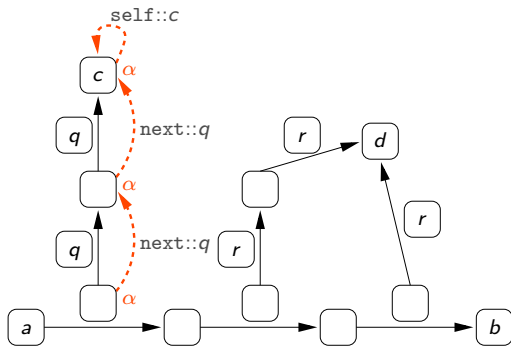


$$\alpha = (\text{next}::q)^*/\text{self}::c$$

$$\beta = (\text{next}::r)^*/\text{self}::d$$

# A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$

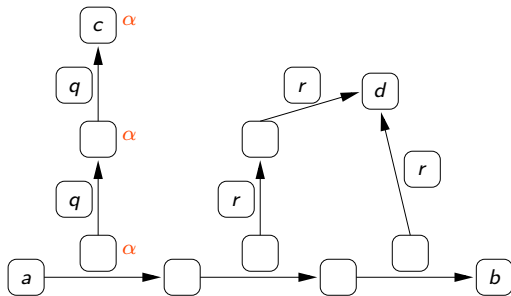


$$\alpha = (\text{next}::q)^*/\text{self}::c$$

$$\beta = (\text{next}::r)^*/\text{self}::d$$

# A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$



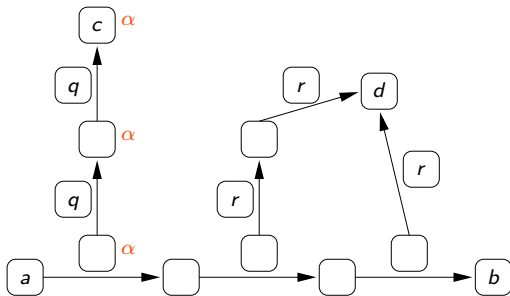
$$\alpha = (\text{next}::q)^*/\text{self}::c$$

$$\beta = (\text{next}::r)^*/\text{self}::d$$



# A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$

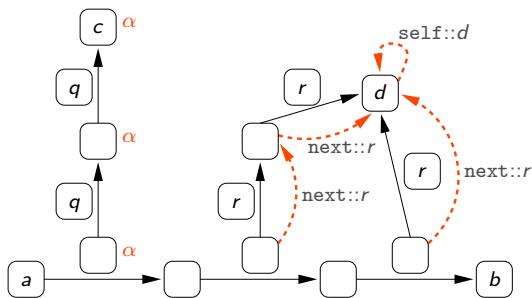


$$\alpha = (\text{next}::q)^*/\text{self}::c$$

$$\beta = (\text{next}::r)^*/\text{self}::d$$

# A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$

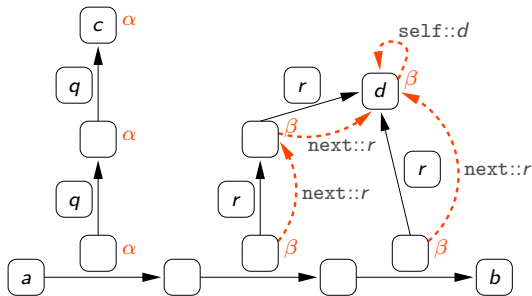


$$\alpha = (\text{next}::q)^*/\text{self}::c$$

$$\beta = (\text{next}::r)^*/\text{self}::d$$

# A closer look at the algorithm

$$exp = next::[(next::q)^*/self::c]/(next::[(next::r)^*/self::d])^*$$

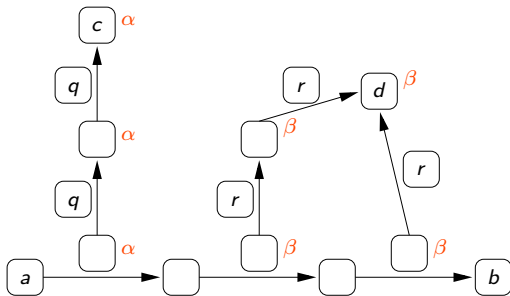


$$\alpha = (next::q)^*/self::c$$

$$\beta = (next::r)^*/self::d$$

# A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$

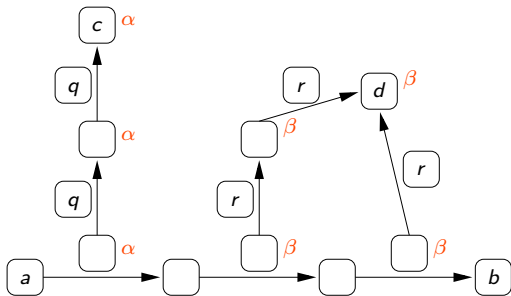


$$\alpha = (\text{next}::q)^*/\text{self}::c$$

$$\beta = (\text{next}::r)^*/\text{self}::d$$

# A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$

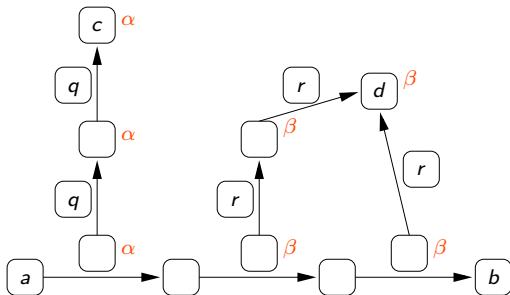


$$\alpha = (\text{next}::q)^*/\text{self}::c$$

$$\beta = (\text{next}::r)^*/\text{self}::d$$

# A closer look at the algorithm

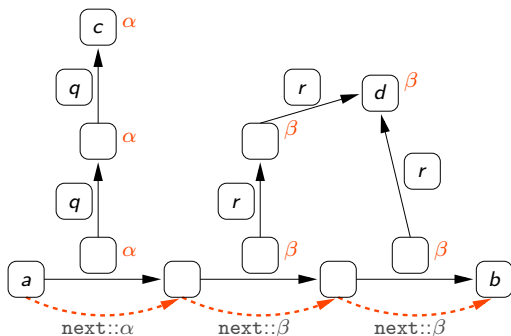
$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$



$$\begin{aligned}\alpha &= (\text{next}::q)^*/\text{self}::c \\ \beta &= (\text{next}::r)^*/\text{self}::d \\ \text{exp}' &= \text{next}::\alpha/(\text{next}::\beta)^*\end{aligned}$$

# A closer look at the algorithm

$$\text{exp} = \text{next}::[(\text{next}::q)^*/\text{self}::c]/(\text{next}::[(\text{next}::r)^*/\text{self}::d])^*$$



$$\begin{aligned}\alpha &= (\text{next}::q)^*/\text{self}::c \\ \beta &= (\text{next}::r)^*/\text{self}::d \\ \text{exp}' &= \text{next}::\alpha/(\text{next}::\beta)^*\end{aligned}$$

# Outline

## Nested regular expressions

Syntax and semantics

Complexity

## nSPARQL

Syntax and semantics

nSPARQL and RDFS evaluation

Expressiveness

## Concluding remarks



# SPARQL

We use the SPARQL formalization proposed in **[PAG06]**

?X :name "john"

(?X, name, john)

{ P1 P2 }

( P<sub>1</sub> AND P<sub>2</sub> )

{ P1 OPTIONAL { P2 } }

( P<sub>1</sub> OPT P<sub>2</sub> )

{ P1 } UNION { P2 }

( P<sub>1</sub> UNION P<sub>2</sub> )

{ P1 FILTER ( R ) }

( P<sub>1</sub> FILTER R )

original SPARQL syntax

algebraic syntax

# SPARQL

We use the SPARQL formalization proposed in **[PAG06]**

`?X :name "john"`

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

`{ P1 P2 }`

$(P_1 \text{ AND } P_2)$

`{ P1 OPTIONAL { P2 } }`

$(P_1 \text{ OPT } P_2)$

`{ P1 } UNION { P2 }`

$(P_1 \text{ UNION } P_2)$

`{ P1 FILTER ( R ) }`

$(P_1 \text{ FILTER } R)$

original SPARQL syntax

algebraic syntax

Example

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

# nSPARQL: navigational SPARQL

*Nested-regular-expression triple* (nre-triple)

$(?X, exp, ?Y)$

- ▶ *exp* is a nested regular expression
- ▶ *?X*, *?Y* are variables (could also be URIs).

# nSPARQL: navigational SPARQL

*Nested-regular-expression triple* (nre-triple)

$(?X, exp, ?Y)$

- ▶ *exp* is a nested regular expression
- ▶ *?X*, *?Y* are variables (could also be URIs).

nSPARQL pattern:

nre-triples combined by using SPARQL operators  
AND, OPT, UNION, FILTER.

# nSPARQL: navigational SPARQL

*Nested-regular-expression triple* (nre-triple)

$(?X, exp, ?Y)$

- ▶ *exp* is a nested regular expression
- ▶ *?X*, *?Y* are variables (could also be URIs).

nSPARQL pattern:

nre-triples combined by using SPARQL operators  
AND, OPT, UNION, FILTER.

Example

$((?X, \text{next::name}, \text{john}) \text{ AND } (?X, \text{next::email}, ?Y))$

# nSPARQL: semantics

Given an RDF graph  $G$

$\{?X \rightarrow a, ?Y \rightarrow b\}$  is in the evaluation of  $(?X, exp, ?Y)$  iff  
 $(a, b)$  is in the evaluation of  $exp$

The semantics of AND, OPT, UNION, FILTER,  
is *inherited* from SPARQL.

# nSPARQL: semantics

Given an RDF graph  $G$

$\{?X \rightarrow a, ?Y \rightarrow b\}$  is in the evaluation of  $(?X, exp, ?Y)$  iff  
 $(a, b)$  is in the evaluation of  $exp$

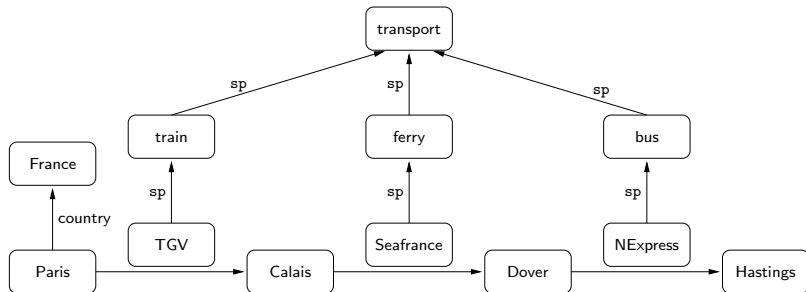
The semantics of AND, OPT, UNION, FILTER,  
is *inherited* from SPARQL.

Note:

Nested regular expressions do not add extra complexity.

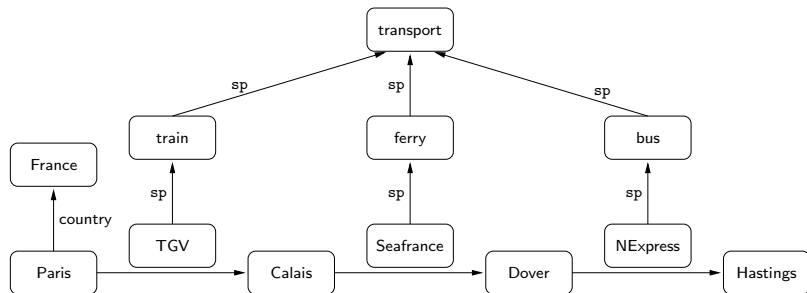
- ▶ An nSPARQL fragment inherits the complexity of the corresponding SPARQL fragment.

# nSPARQL examples



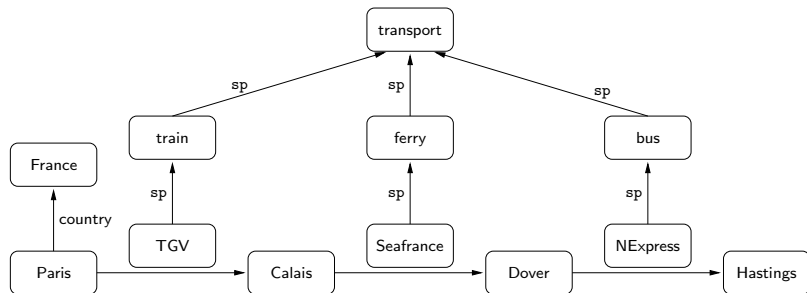


# nSPARQL examples



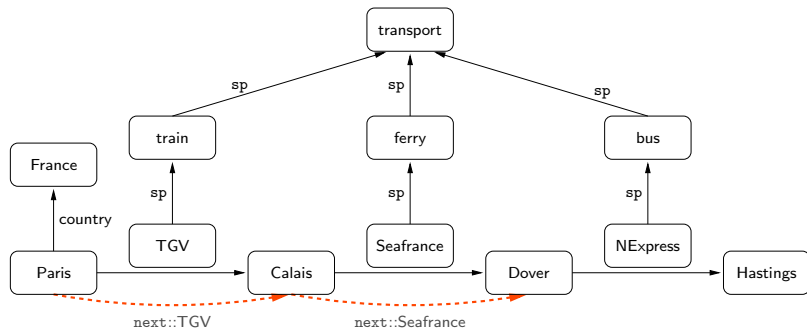
$(?X, (\text{next}::\text{TGV} \mid \text{next}::\text{SeaFrance})^+, \text{Dover}) \text{ AND } (?X, \text{next}::\text{country}, ?Y)$

# nSPARQL examples



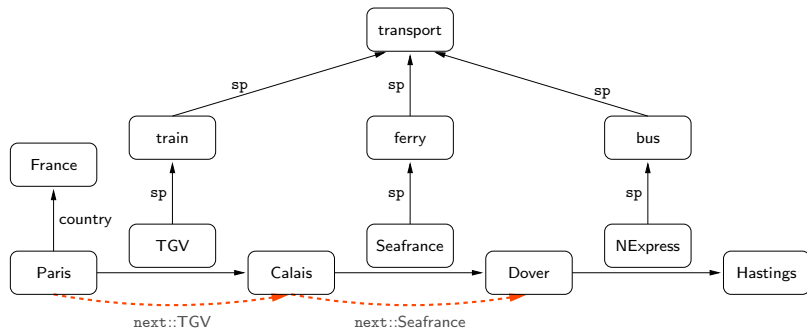
(?X, (next::TGV | next::SeaFrance)<sup>+</sup>, Dover) AND (?X, next::country, ?Y)

# nSPARQL examples



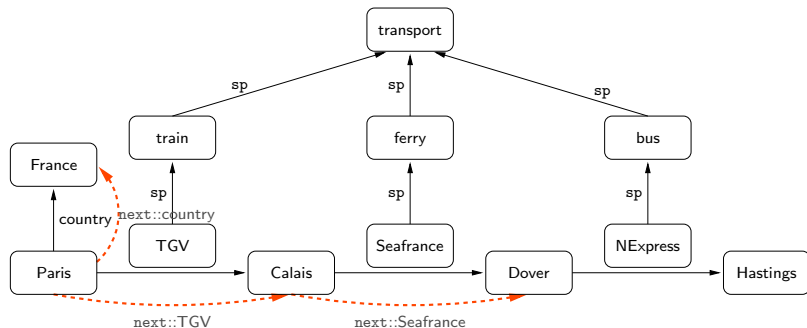
(?X, (next::TGV | next::SeaFrance)<sup>+</sup>, Dover) AND (?X, next::country, ?Y)

# nSPARQL examples



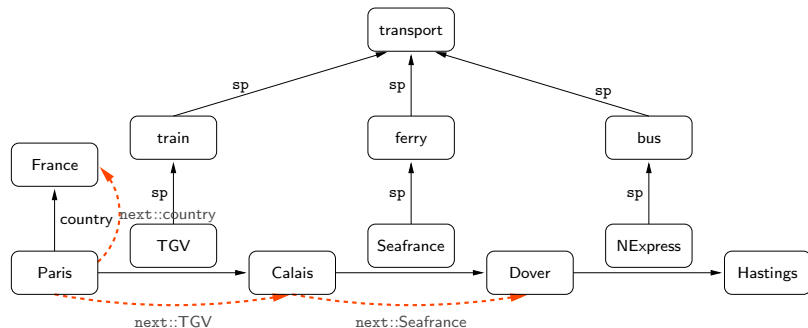
$(?X, (\text{next}::\text{TGV} \mid \text{next}::\text{SeaFrance})^+, \text{Dover}) \text{ AND } (?X, \text{next}::\text{country}, ?Y)$

# nSPARQL examples



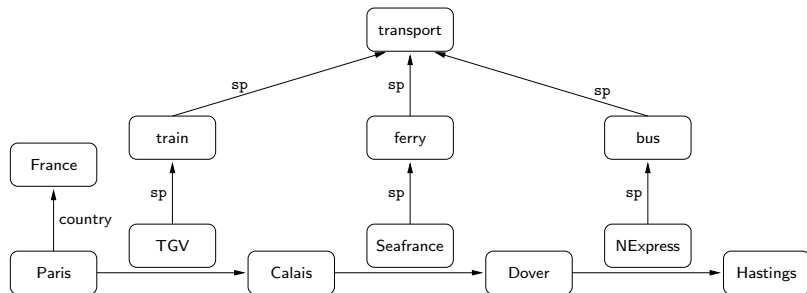
$(?X, (next::TGV \mid next::SeaFrance)^+, Dover) \text{ AND } (?X, next::country, ?Y)$

# nSPARQL examples



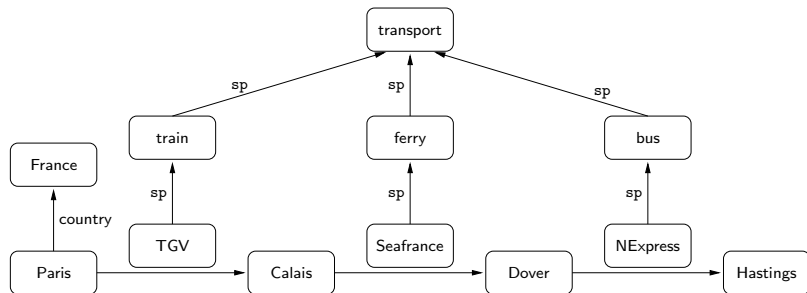
$(?X, (\text{next}::\text{TGV} \mid \text{next}::\text{SeaFrance})^+, \text{Dover}) \text{ AND } (?X, \text{next}::\text{country}, ?Y)$   
 $\{?X \rightarrow \text{Paris}, ?Y \rightarrow \text{France}\}$

# nSPARQL examples



`(?X, next::[(next::sp)*self::transport], Hastings)`

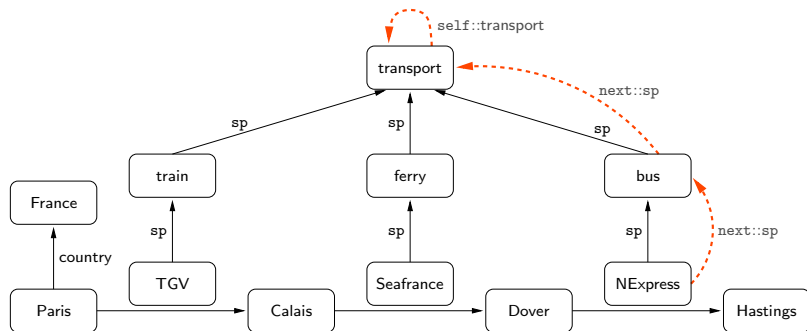
# nSPARQL examples



```
(?X, next::[(next::sp)* /self::transport], Hastings)
```

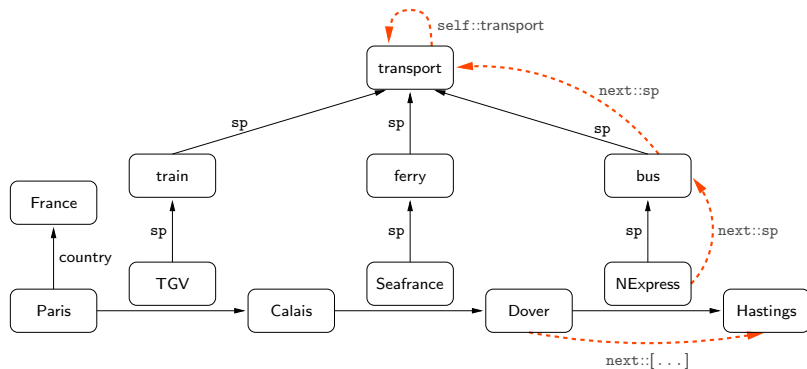


# nSPARQL examples



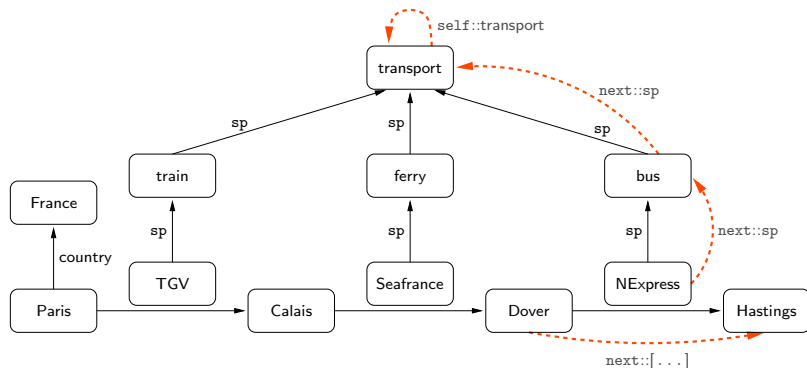
`(?X, next::[(next::sp)* / self::transport], Hastings)`

# nSPARQL examples



`(?X, next::[(next::sp)* / self::transport], Hastings)`

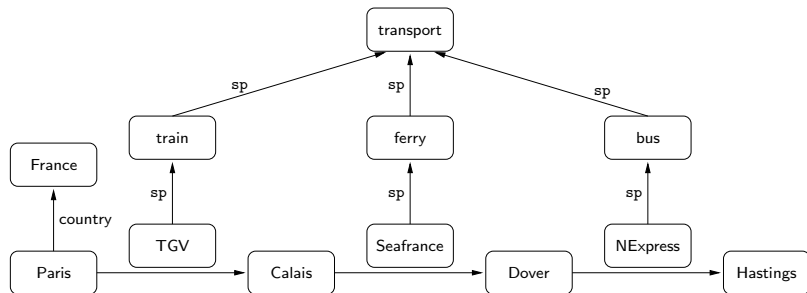
# nSPARQL examples



$(?X, \text{next::}[(\text{next::sp})^*/\text{self::transport}], \text{Hastings})$

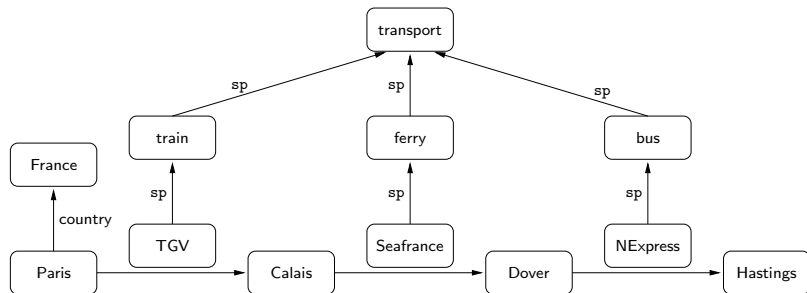
$\{?X \rightarrow \text{Dover}\}$

# nSPARQL examples



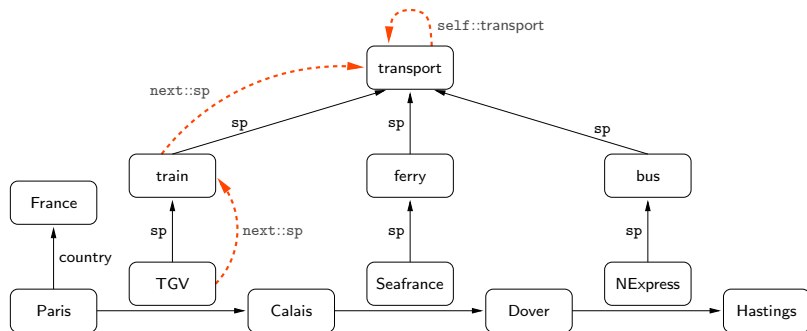
```
(?X, (next::[(next::sp)* /self::transport])+, Hastings)
```

# nSPARQL examples



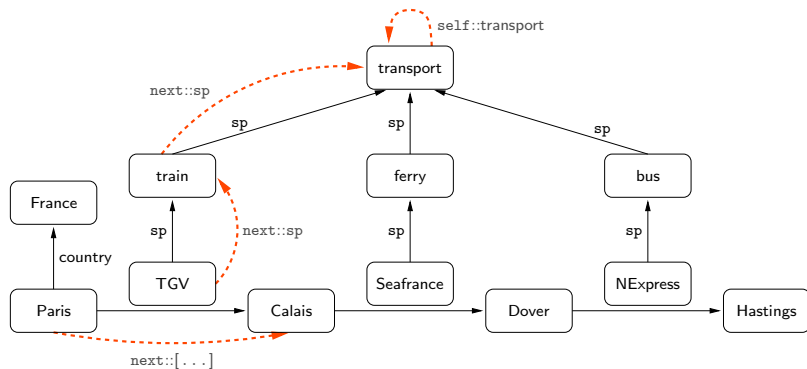
```
(?X, (next::[(next::sp)* /self::transport])+, Hastings)
```

# nSPARQL examples



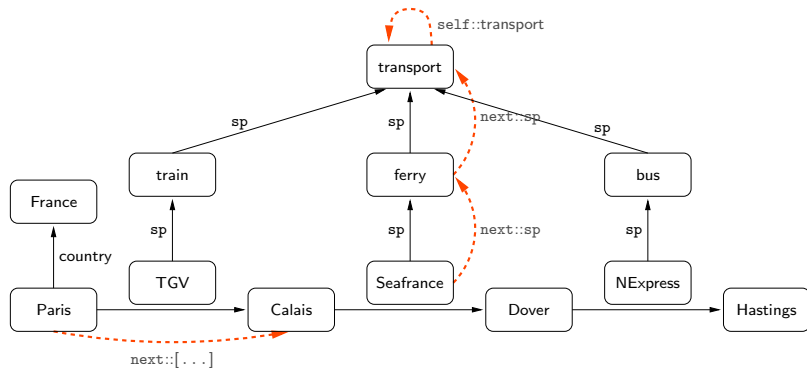
```
(?X, (next::[(next::sp)* / self::transport])+, Hastings)
```

# nSPARQL examples



```
(?X, (next::[(next::sp)* / self::transport])+, Hastings)
```

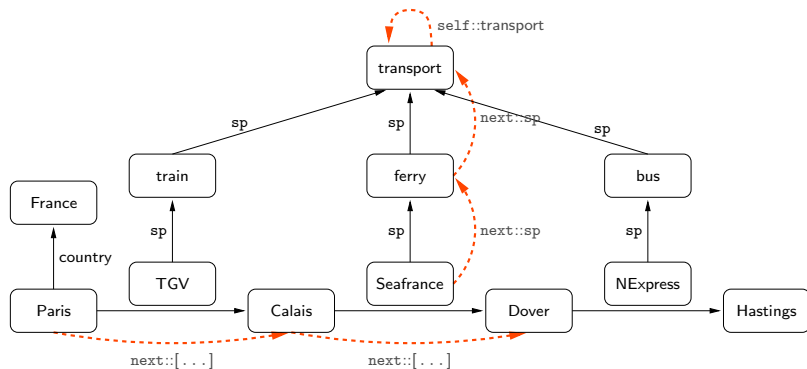
# nSPARQL examples



```
(?X, (next::[(next::sp)* / self::transport])+, Hastings)
```

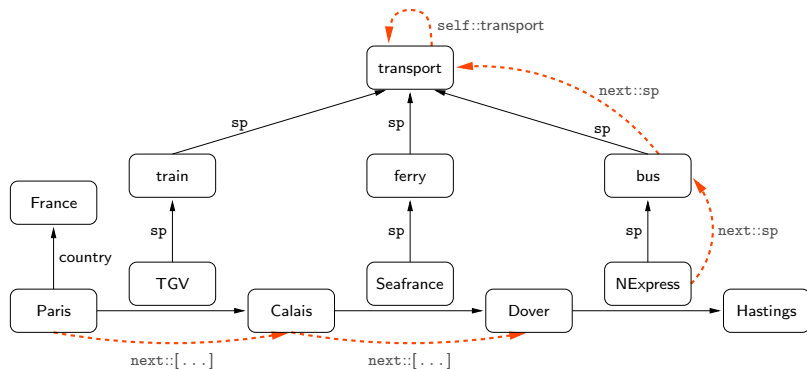


# nSPARQL examples



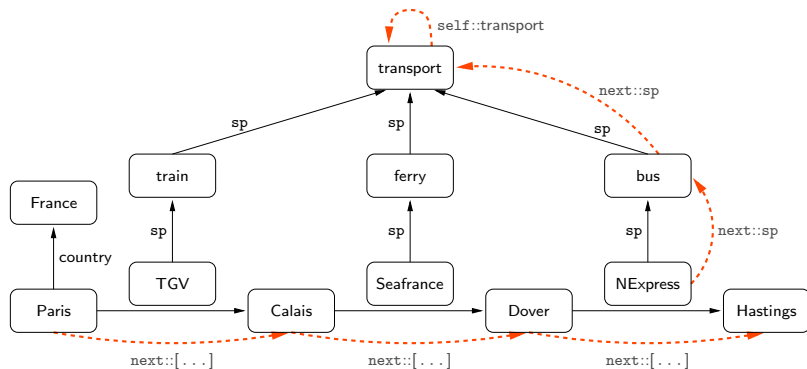
$(?X, (next::((next::sp)^*/self::transport))^+, Hastings)$

# nSPARQL examples



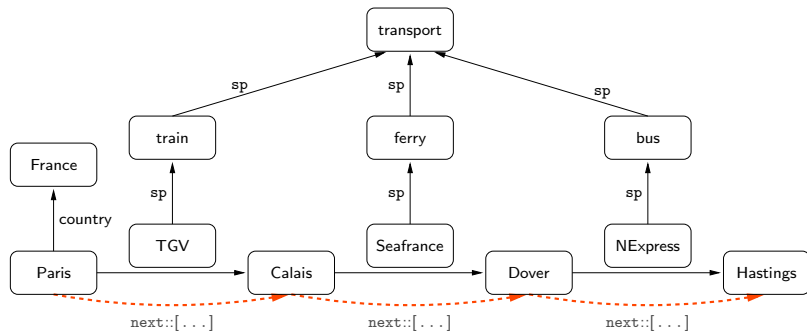
$(?X, (next::[(next::sp)^*/self::transport])^+, Hastings)$

# nSPARQL examples



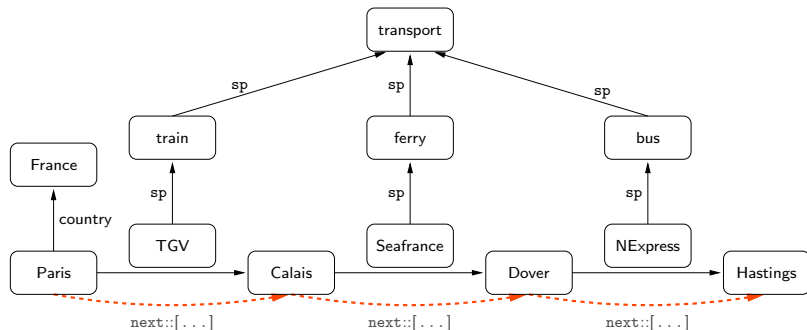
$(?X, (next::[(next::sp)^*/self::transport])^+, Hastings)$

# nSPARQL examples



$(?X, (next::[(next::sp)^*/self::transport])^+, Hastings)$

# nSPARQL examples



$(?X, (next::[(next::sp)^*/self::transport])^+, Hastings)$

$\{?X \rightarrow Paris\}$

$\{?X \rightarrow Calais\}$

$\{?X \rightarrow Dover\}$

# Outline

## Nested regular expressions

Syntax and semantics

Complexity

## nSPARQL

Syntax and semantics

nSPARQL and RDFS evaluation

Expressiveness

## Concluding remarks

# RDFS evaluation via the closure

Consider the sub-vocabulary of RDFS  $\{\text{sp}, \text{sc}, \text{dom}, \text{range}, \text{type}\}$  and its semantics via deductive rules.

*Closure of G*: obtained by applying rules to exhaustion.

# RDFS evaluation via the closure

Consider the sub-vocabulary of RDFS  $\{\text{sp, sc, dom, range, type}\}$  and its semantics via deductive rules.

*Closure of  $G$* : obtained by applying rules to exhaustion.

## Definition

Given a SPARQL pattern  $P$ , its *RDFS evaluation* over a graph  $G$  is the usual evaluation of  $P$  over the *closure* of  $G$ .



nSPARQL is enough for RDFS evaluation of standard SPARQL queries.

### Theorem

*Given a SPARQL triple  $t = (?X, a, ?Y)$ , there exists an nSPARQL triple  $r = (?X, \text{trans}(a), ?Y)$  such that*

*the RDFS evaluation of  $t$  can be obtained by directly evaluating  $r$  over the input graph.*

nSPARQL is enough for RDFS evaluation of standard SPARQL queries.

### Theorem

*Given a SPARQL triple  $t = (?X, a, ?Y)$ , there exists an nSPARQL triple  $r = (?X, \text{trans}(a), ?Y)$  such that*

*the RDFS evaluation of  $t$  can be obtained by directly evaluating  $r$  over the input graph.*

### Proof idea

- ▶  $\text{trans}(\text{sc}) = (\text{next}::\text{sc})^+$
- ▶ ...
- ▶  $\text{trans}(a) = \text{next}::[(\text{next}::\text{sp})^*/\text{self}::a]$  ( $a \notin \{\text{sp}, \text{sc}, \dots\}$ )

$\text{trans}(\text{type})$  uses  $\text{next}$ ,  $\text{edge}$ ,  $\text{node}^{-1}$ .

nSPARQL is enough for RDFS evaluation of standard SPARQL queries.

### Theorem

*Given a SPARQL pattern  $P$ , we can translate it into an nSPARQL pattern  $Q$  such that*

*the RDFS evaluation of  $P$  can be obtained by directly evaluating  $Q$  over the input graph.*

Just rewrite every triple of  $P$ .

nSPARQL without nesting is not enough.

Let rSPARQL the fragment of nSPARQL obtained by forbidding nesting in expressions (usual regular expressions over the axes).

nSPARQL without nesting is not enough.

Let rSPARQL the fragment of nSPARQL obtained by forbidding nesting in expressions (usual regular expressions over the axes).

### Proposition

*The RDFS evaluation of SPARQL triple pattern  $(?X, a, ?Y)$  cannot be obtained with any rSPARQL pattern.*

Nested regular expressions and  
SPARQL operators are complementary.

- ▶ nSPARQL add expressive power to SPARQL

# Nested regular expressions and SPARQL operators are complementary.

- ▶ nSPARQL add expressive power to SPARQL

Do SPARQL operators add expressive power to nSPARQL expressions?

# Nested regular expressions and SPARQL operators are complementary.

- ▶ nSPARQL add expressive power to SPARQL

Do SPARQL operators add expressive power to nSPARQL expressions?

## Theorem

*There is an nSPARQL pattern that is not equivalent to any nSPARQL pattern that does not use OPT.*



# Outline

## Nested regular expressions

Syntax and semantics

Complexity

## nSPARQL

Syntax and semantics

nSPARQL and RDFS evaluation

Expressiveness

## Concluding remarks

# Concluding remarks

nested regular expressions:

- ▶ natural navigation through an RDF graph
- ▶ low complexity of evaluation

nSPARQL:

- ▶ combines the power of SPARQL operators and nre
- ▶ RDFS evaluation by traversing the input graph
- ▶ allow expressing natural queries

# nSPARQL: A Navigational Language for RDF

Jorge Pérez   Marcelo Arenas   Claudio Gutierrez

Departamento de Ciencia de la Computación  
Pontificia Universidad Católica de Chile  
Universidad de Chile

*Khipu group*  
<http://www.khipu.cl>