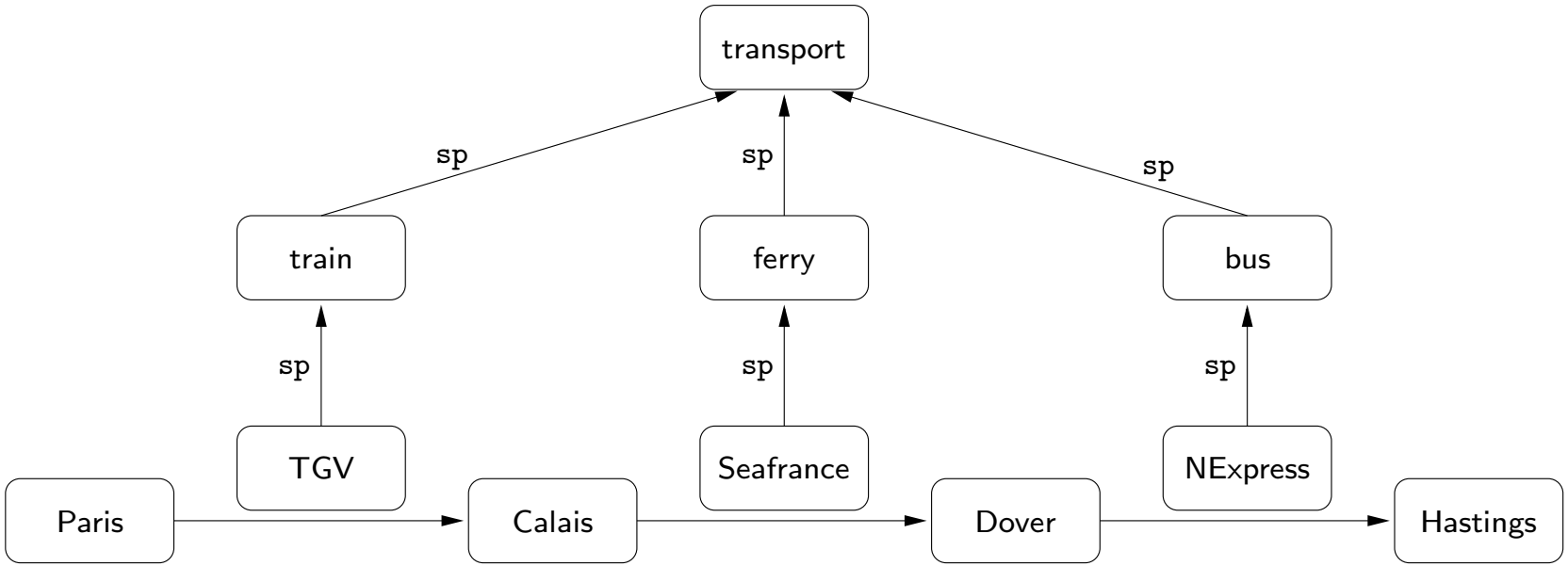
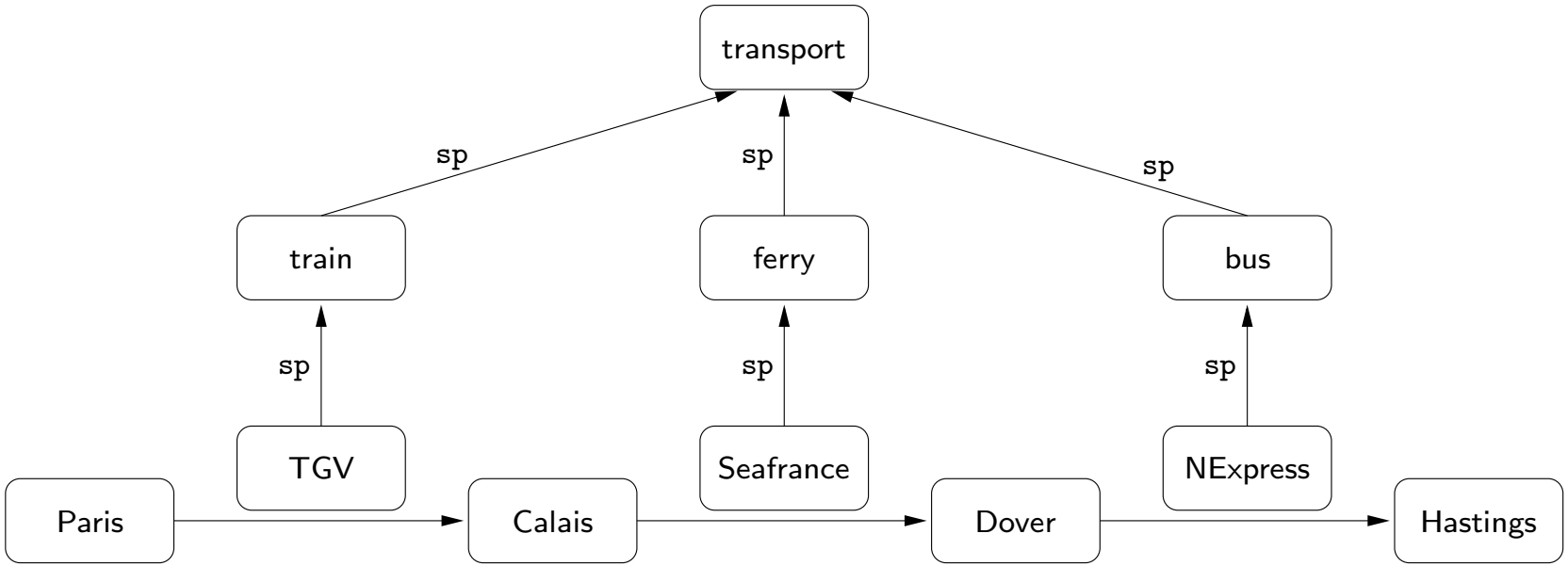
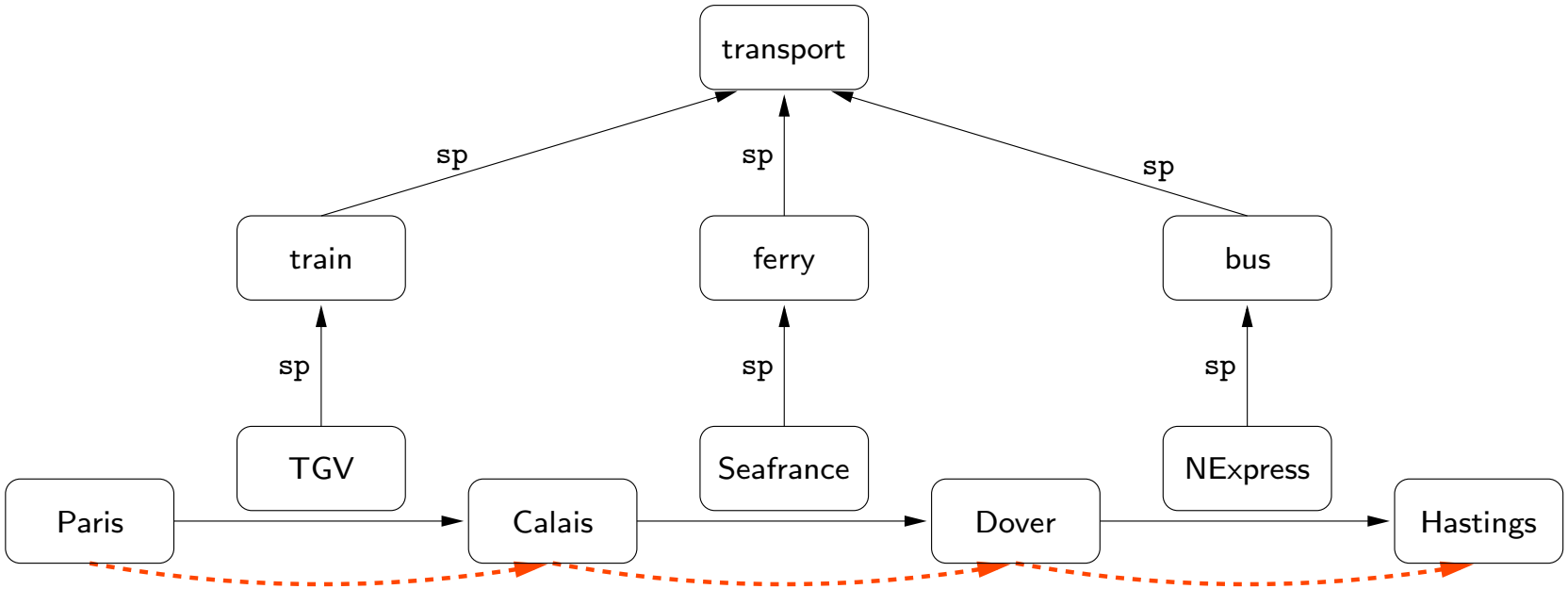
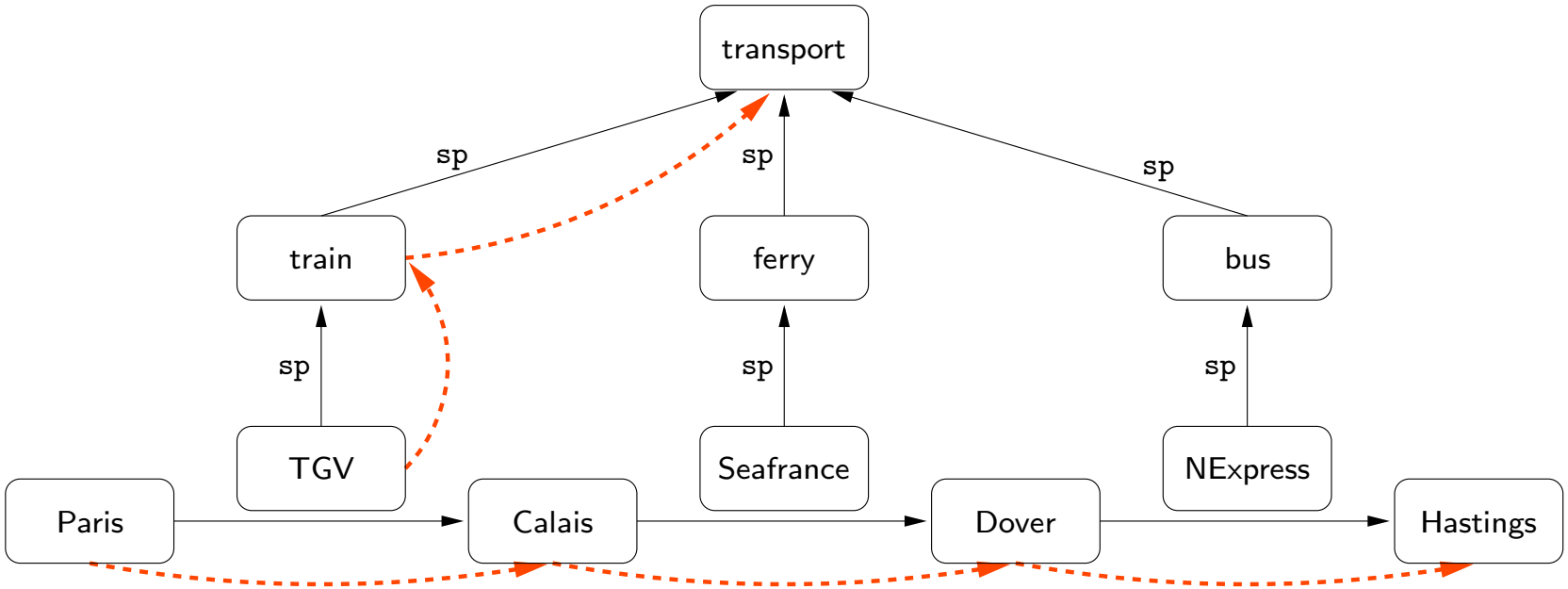

transport
sp
sp
sp
train
ferry
bus
sp
sp
sp
TGV
Seafrance
NExpress
Paris
Calais
Dover
Hastings



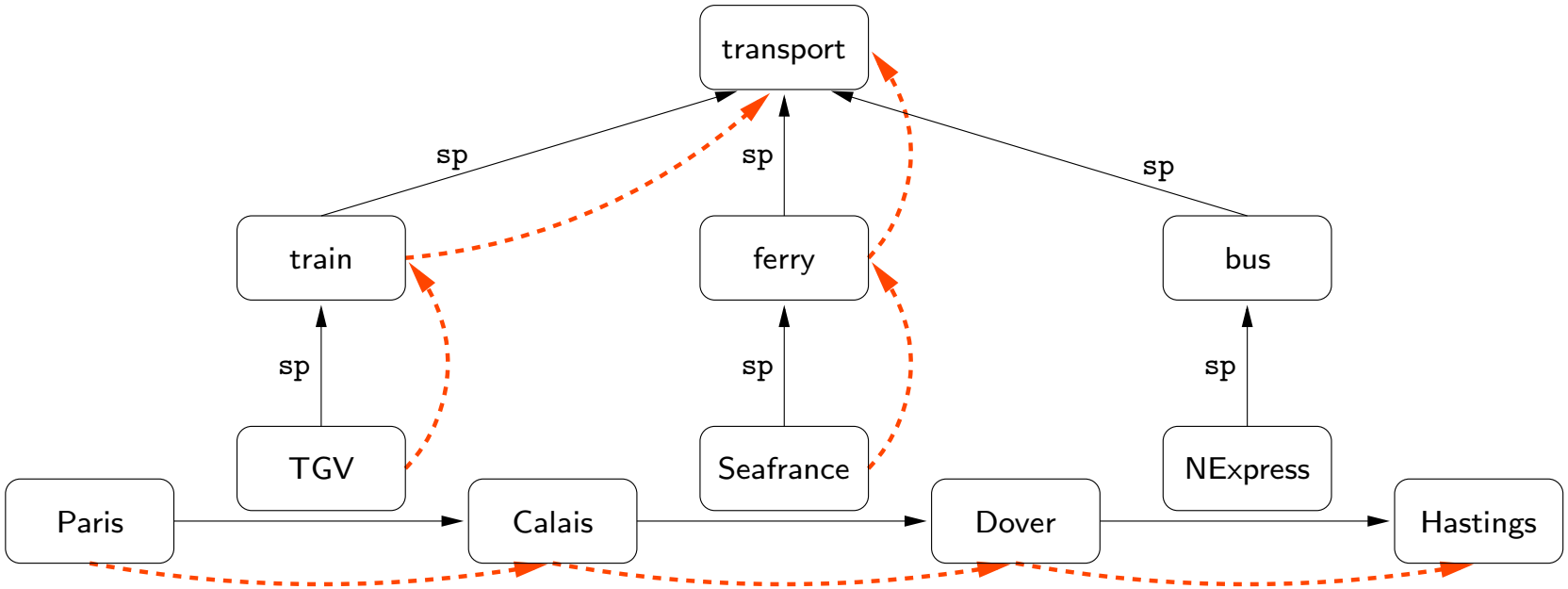
Are Paris and Hastings connected
by a sequence of transportation services?



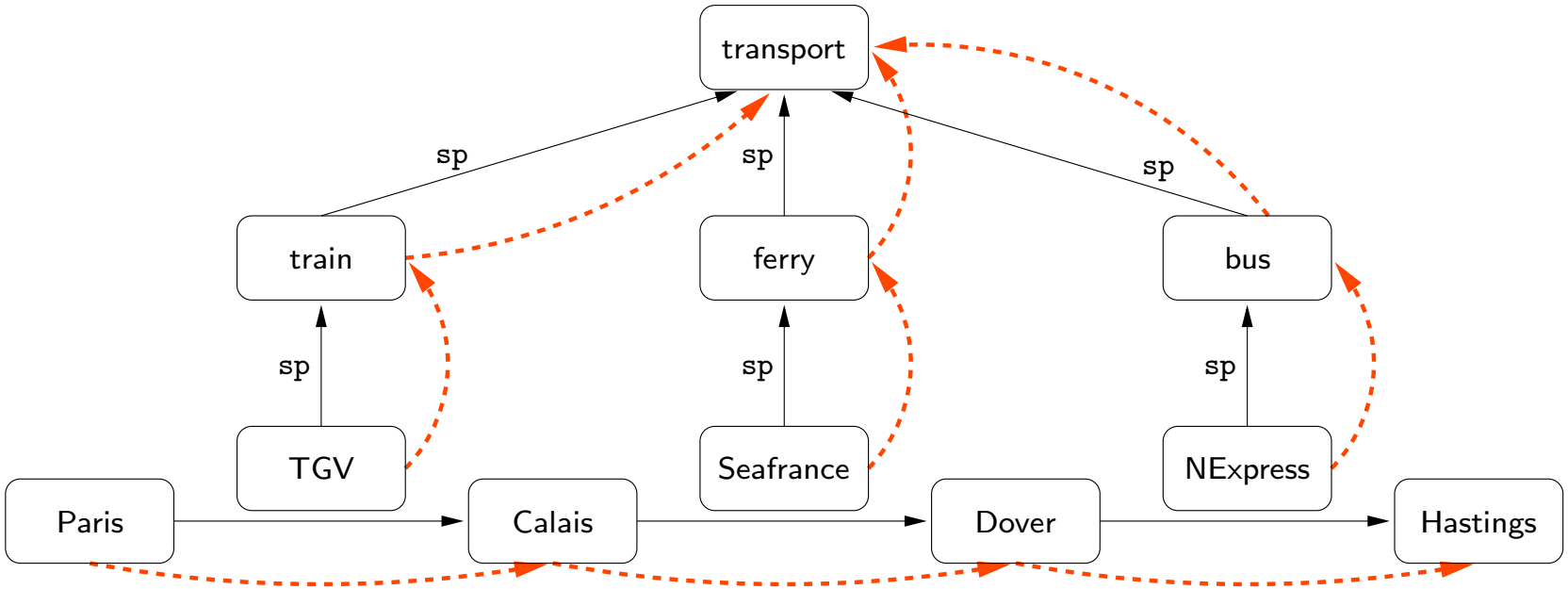
Are Paris and Hastings connected
by a sequence of transportation services?



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

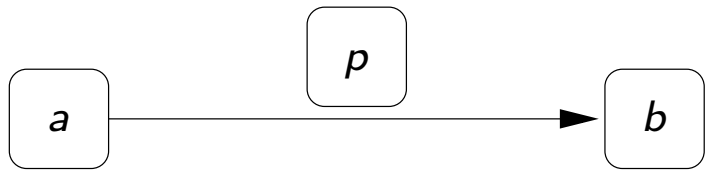
We incorporate these features in SPARQL

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

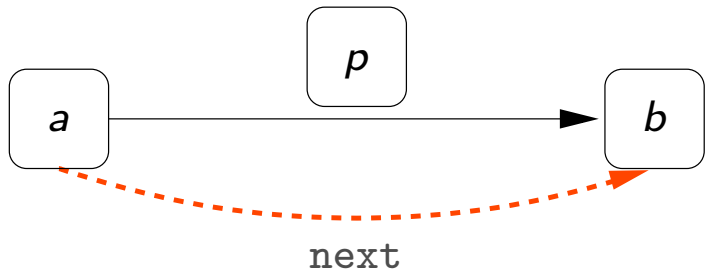
Outline

Outline

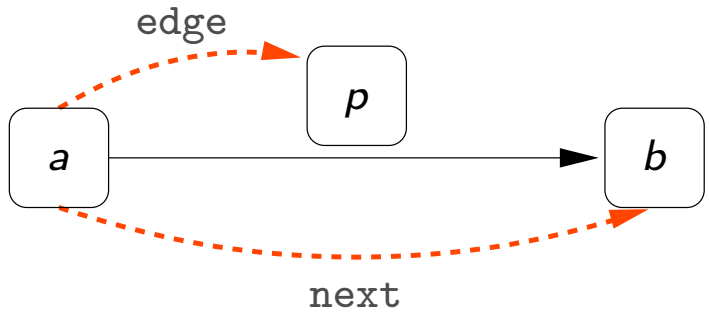
Navigational axes for RDF triple (a, p, b)



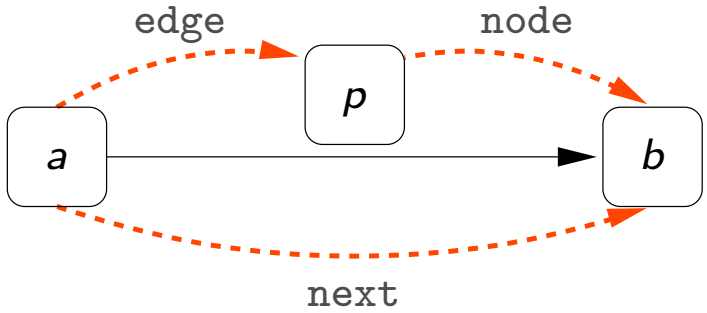
Navigational axes for RDF triple (a, p, b)



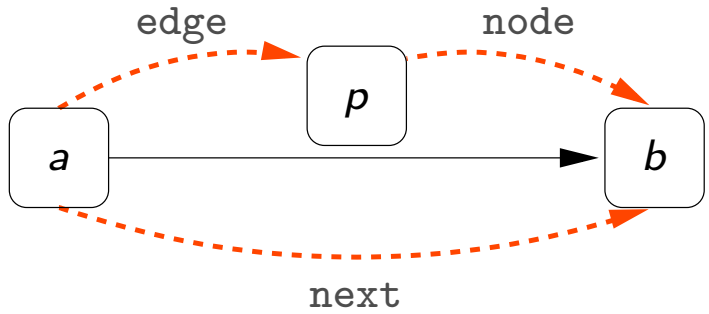
Navigational axes for RDF triple (a, p, b)

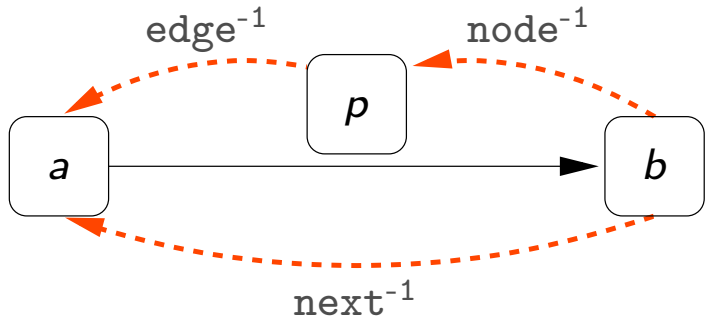


Navigational axes for RDF triple (a, p, b)

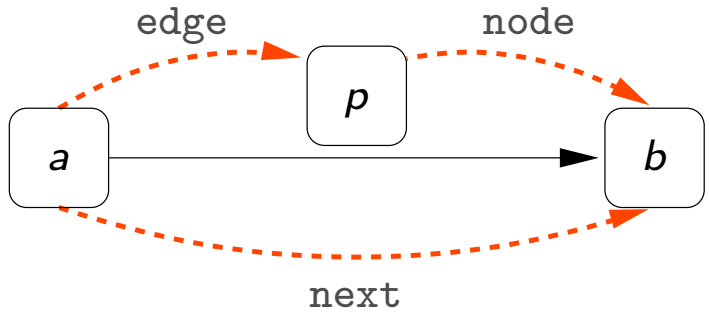


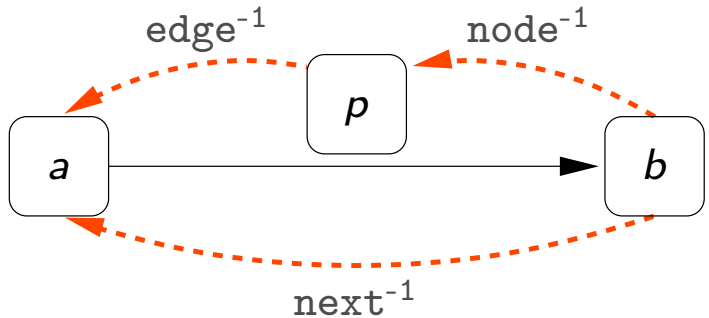
Navigational axes for RDF triple (a, p, b)





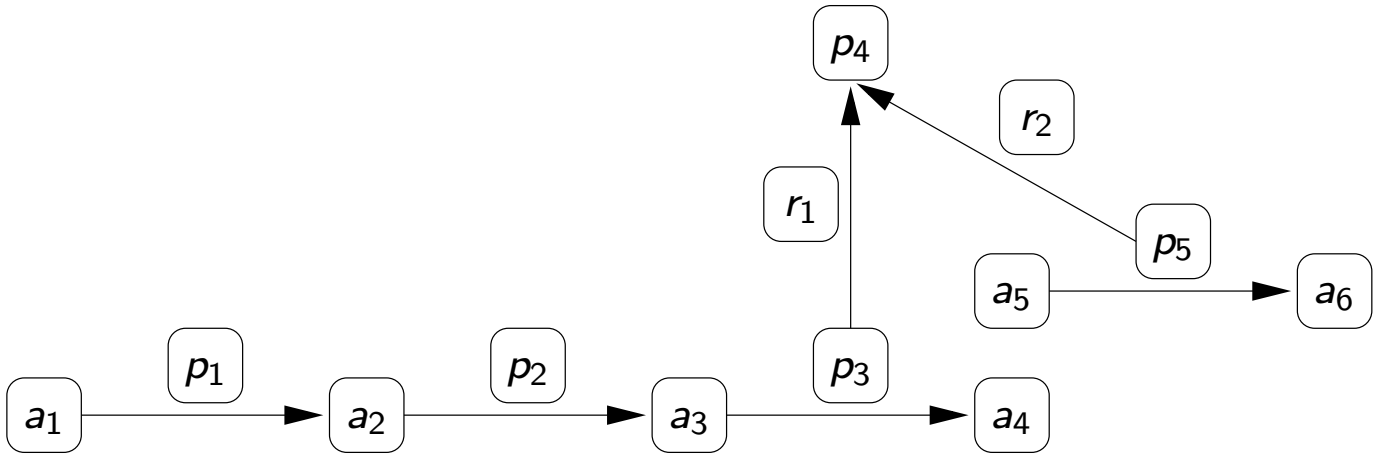
Navigational axes for RDF triple (a, p, b)



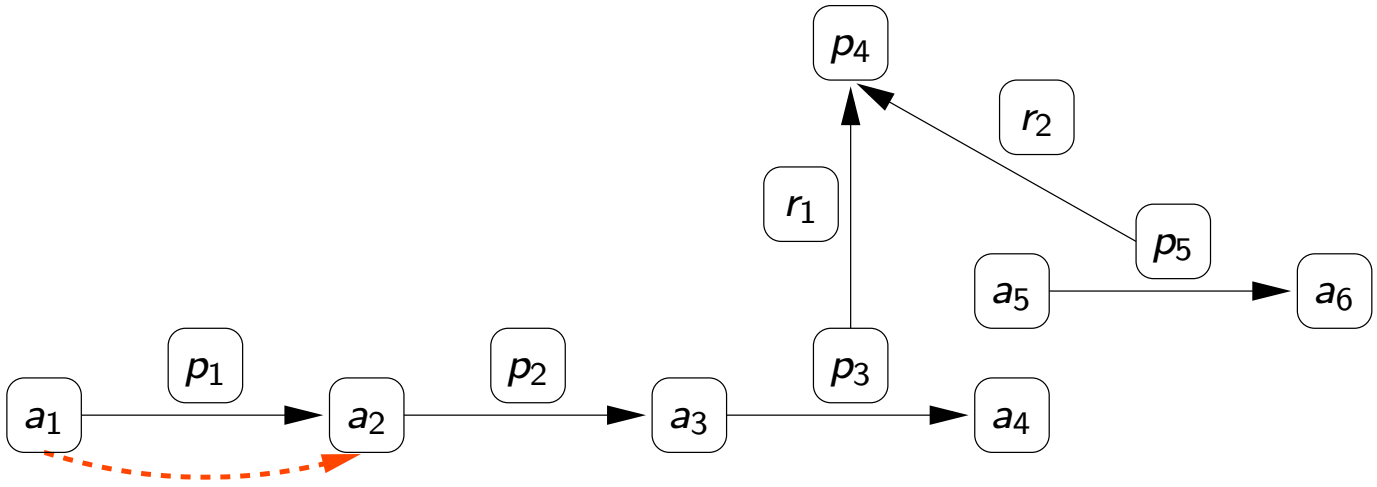


plus `self` axis

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

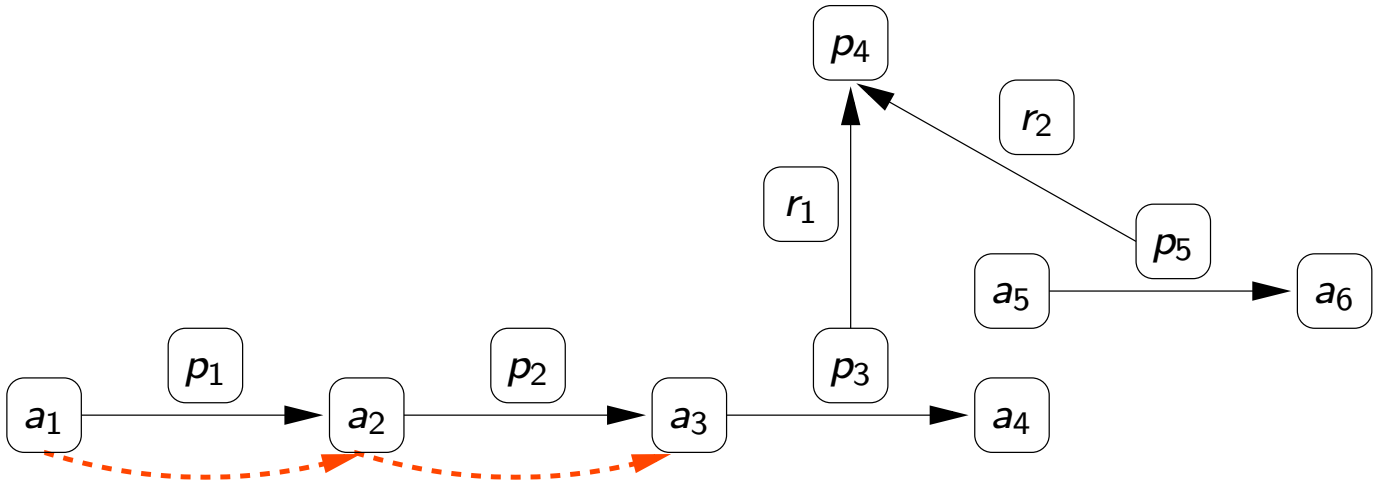


A sequence of axes defines a path in an RDF graph.



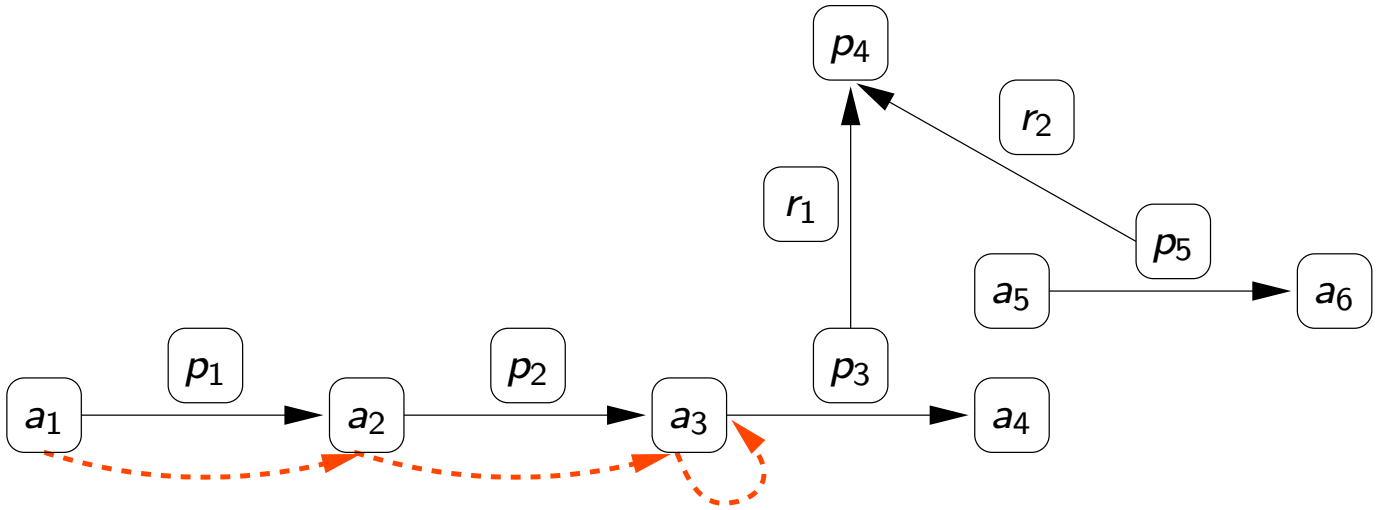
`next`

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



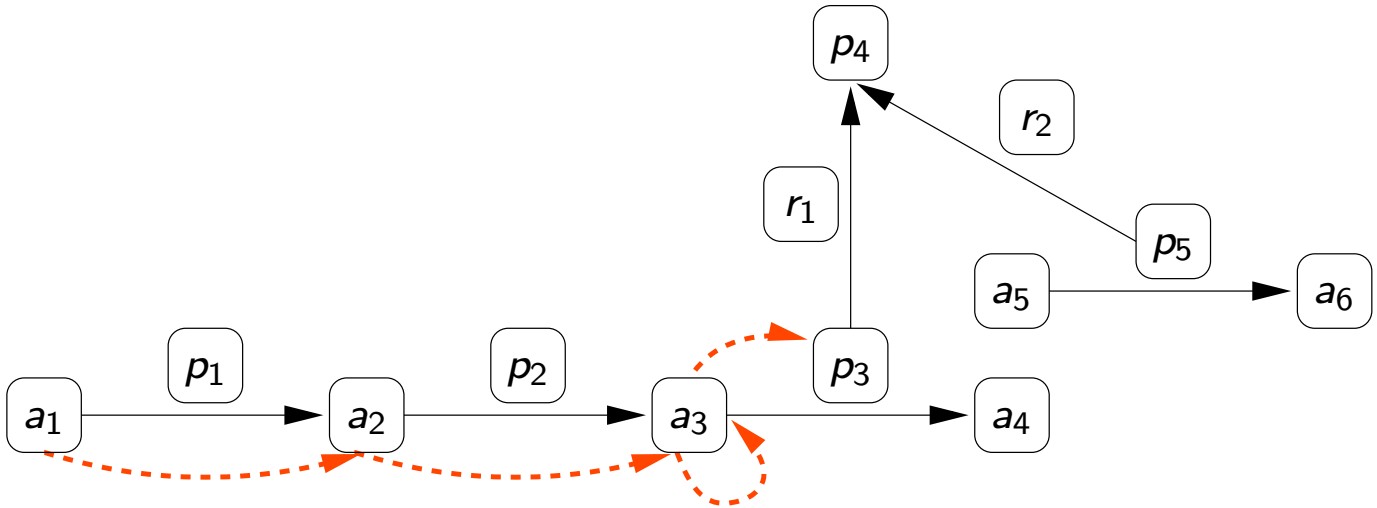
`next/next`

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



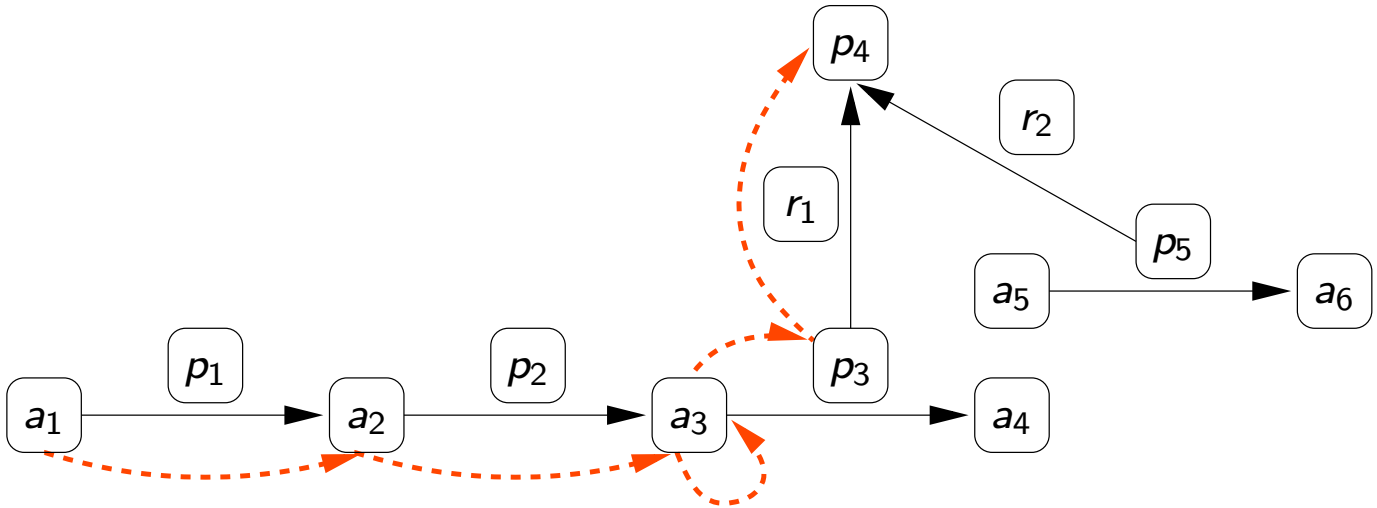
`next/next/self`

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



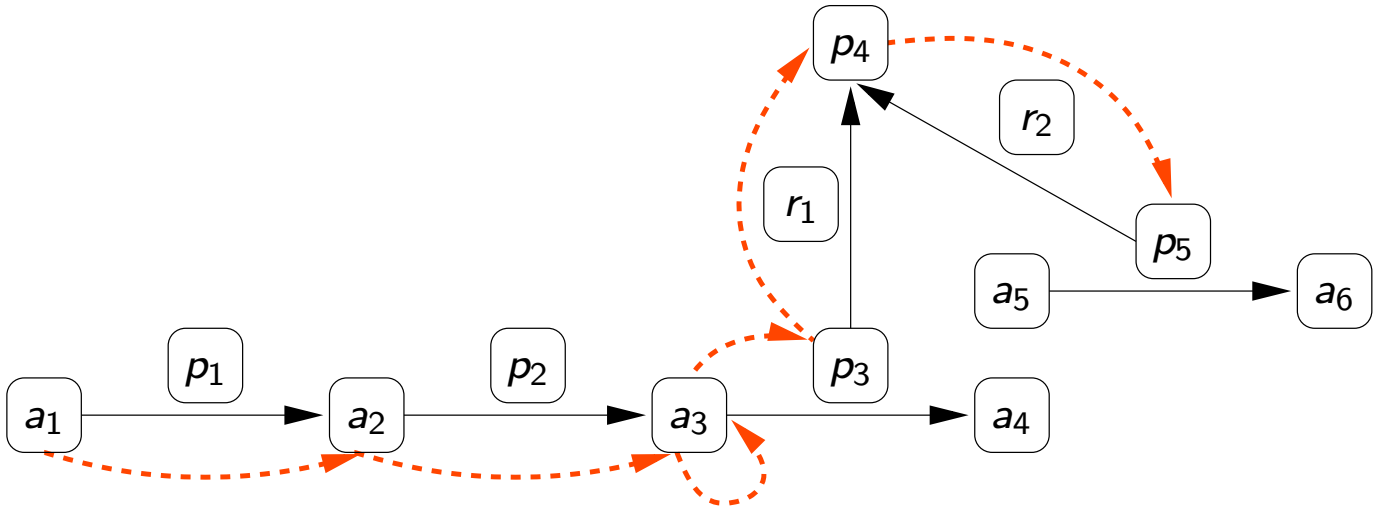
`next/next/self/edge`

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



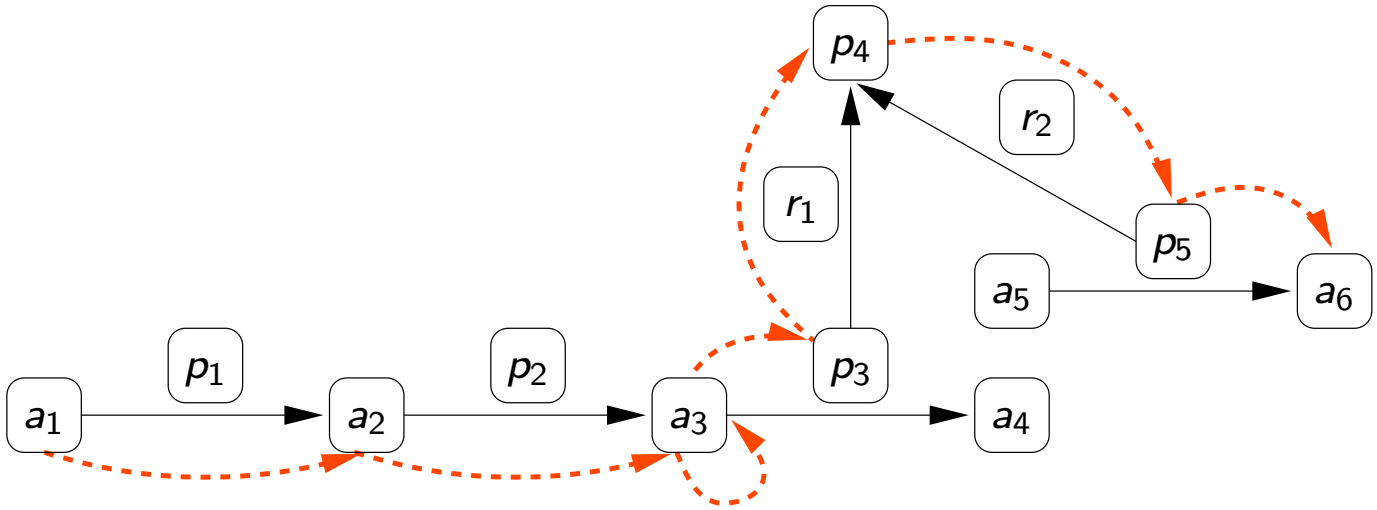
`next/next/self/edge/next`

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:

$$exp \quad :=$$

Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$$exp \;:=\; \text{axis}$$

Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

$$exp \;:=\; \text{axis}$$

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

Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

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

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

Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

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

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

Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

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

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

Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

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

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

Syntax of nested regular expressions over RDF

Syntax of nested regular expressions over RDF:

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

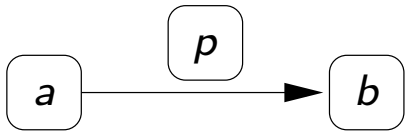
- axis $\in \{\texttt{self}, \texttt{next}, \texttt{next}^{-1}, \texttt{edge}, \texttt{edge}^{-1}, \texttt{node}, \texttt{node}^{-1}\}$
- a is a URI
- $[exp]$ is a *nesting* expression

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*.

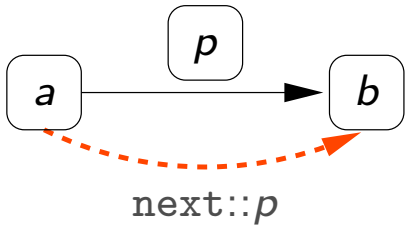
Semantics of nested regular expressions over RDF

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*.



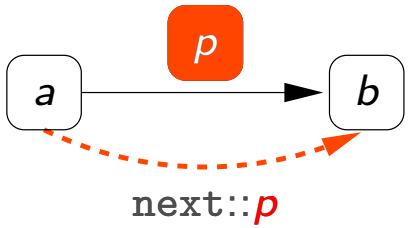
Semantics of nested regular expressions over RDF

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*.



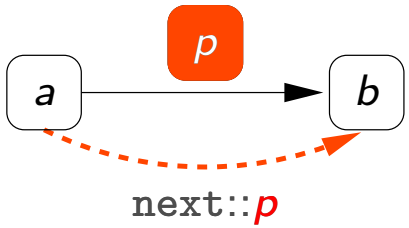
Semantics of nested regular expressions over RDF

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

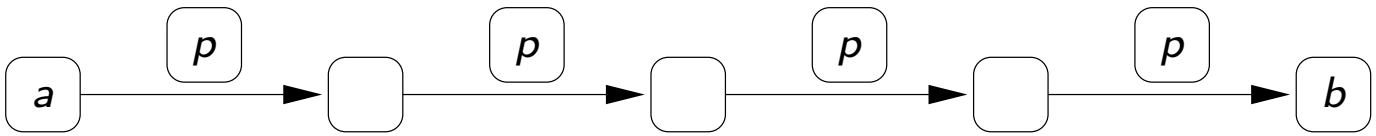
The *evaluation* of an expression *exp* is a set of pairs (a, b) of nodes such that b is reachable from a by following *exp*.



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

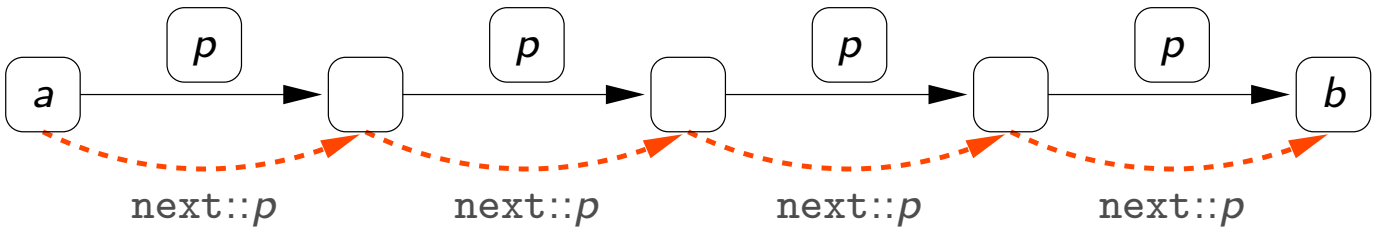
Semantics of nested regular expressions over RDF

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*.



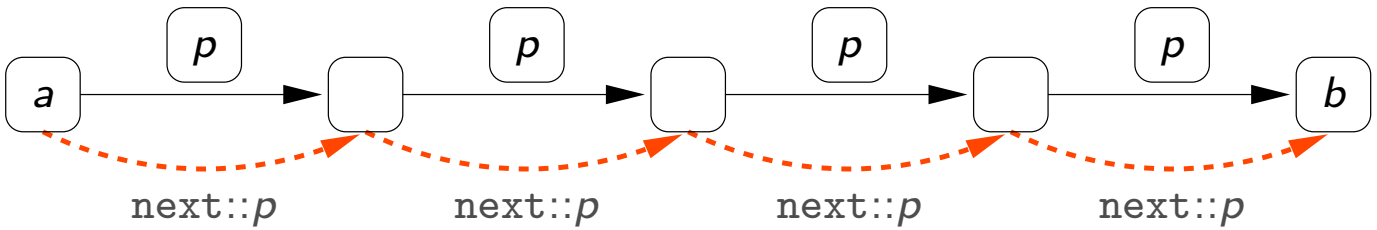
Semantics of nested regular expressions over RDF

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

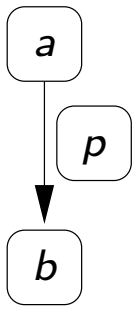
The *evaluation* of an expression *exp* is a set of pairs (a, b) of nodes such that b is reachable from a by following *exp*.



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

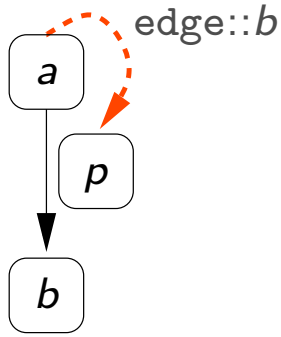
Semantics of nested regular expressions over RDF

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*.



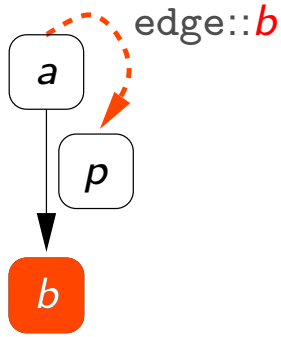
Semantics of nested regular expressions over RDF

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*.



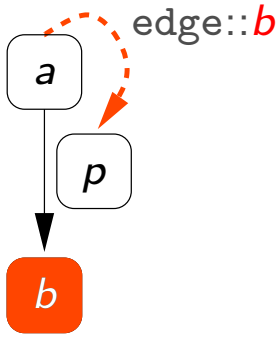
Semantics of nested regular expressions over RDF

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

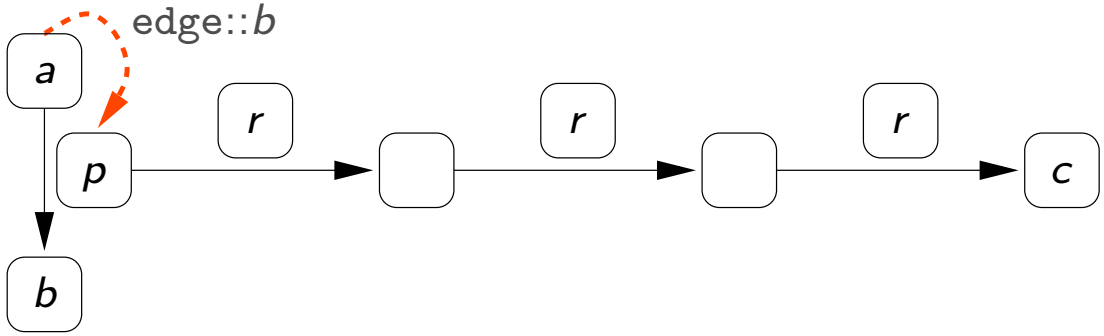
The *evaluation* of an expression *exp* is a set of pairs (a, b) of nodes such that b is reachable from a by following *exp*.



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

Semantics of nested regular expressions over RDF

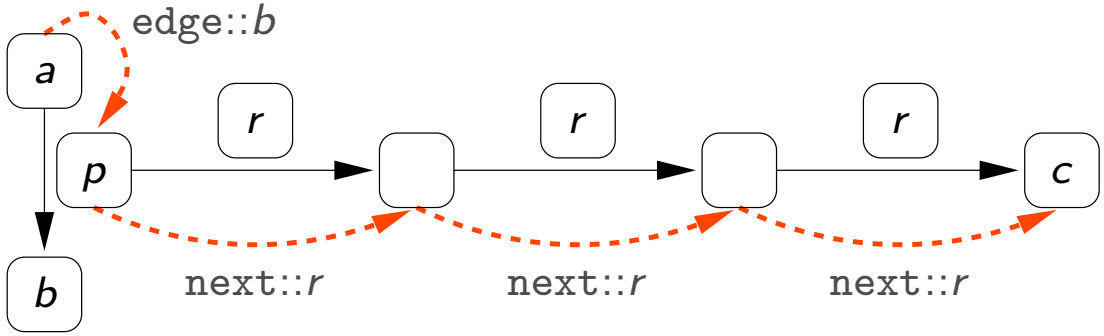
The *evaluation* of an expression *exp* is a set of pairs (a, b) of nodes such that b is reachable from a by following *exp*.



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

Semantics of nested regular expressions over RDF

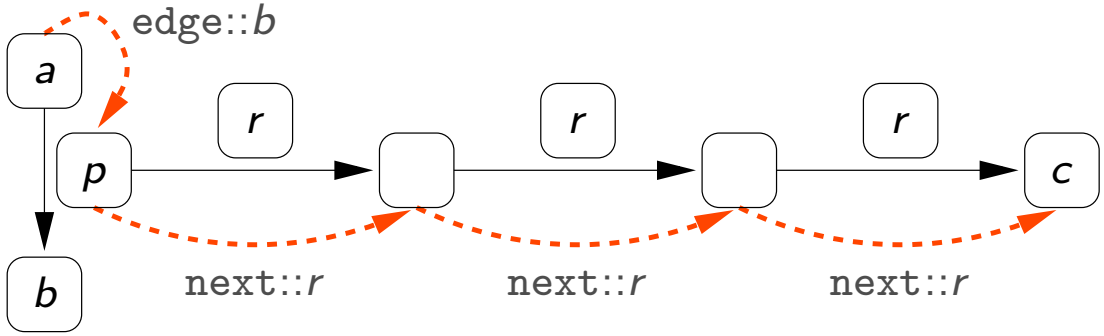
The *evaluation* of an expression *exp* is a set of pairs (a, b) of nodes such that b is reachable from a by following *exp*.



(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 $\texttt{edge::}b$

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

Semantics of nested regular expressions over RDF graphs

Definition

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

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

Semantics of nested regular expressions over RDF graphs

Definition

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

$$
\begin{array}{lcl}
[\![\texttt{self}]\!]_G & = & \{(x,x) \mid x \in \mathrm{voc}(G)\} \\
[\![\texttt{next}]\!]_G & = & \{(x,y) \mid \text{ exists } z \text{ s.t. } (x,z,y) \in G\}
\end{array}
$$

Semantics of nested regular expressions over RDF graphs

Definition

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

$$
\begin{array}{rcl}
[\![\texttt{self}]\!]_G & = & \{(x,x) \mid x \in \mathrm{voc}(G)\} \\
[\![\texttt{next}]\!]_G & = & \{(x,y) \mid \text{ exists } z \text{ s.t. } (x,z,y) \in G\} \\
[\![\texttt{self::}a]\!]_G & = & \{(a,a)\}
\end{array}
$$

Semantics of nested regular expressions over RDF graphs

Definition

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

$$
\begin{array}{rcl}
[\![\texttt{self}]\!]_G & = & \{(x,x) \mid x \in \mathrm{voc}(G)\} \\
[\![\texttt{next}]\!]_G & = & \{(x,y) \mid \text{ exists } z \text{ s.t. } (x,z,y) \in G\} \\
[\![\texttt{self::}a]\!]_G & = & \{(a,a)\} \\
[\![\texttt{next::}a]\!]_G & = & \{(x,y) \mid (x,a,y) \in G\}
\end{array}
$$

Semantics of nested regular expressions over RDF graphs

Definition

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

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

Semantics of nested regular expressions over RDF graphs

Definition

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

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

Semantics of nested regular expressions over RDF graphs

Definition

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

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

Semantics of nested regular expressions over RDF graphs

Definition

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

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

Semantics of nested regular expressions over RDF graphs

Definition

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

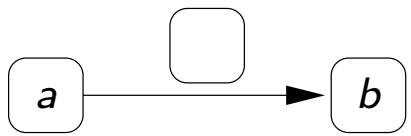
$$
\begin{aligned}
[\![\texttt{self}]\!]_G &= \{(x,x) \mid x \in \mathrm{voc}(G)\} \\
[\![\texttt{next}]\!]_G &= \{(x,y) \mid \text{ exists } z \text{ s.t. } (x,z,y) \in G\} \\
[\![\texttt{self::}a]\!]_G &= \{(a,a)\} \\
[\![\texttt{next::}a]\!]_G &= \{(x,y) \mid (x,a,y) \in G\} \\
[\![exp_1/exp_2]\!]_G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

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

Nesting allows for existential checking.

Intuition:

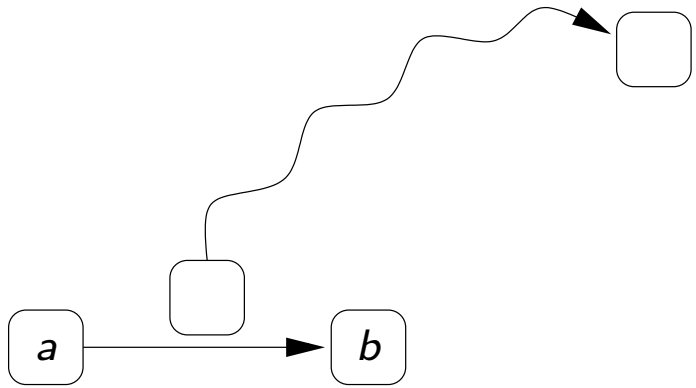
`next`::[*exp*] navigates from node *a* to node *b* if they are connected with an edge that satisfies a path condition given by *exp*.



Nesting allows for existential checking.

Intuition:

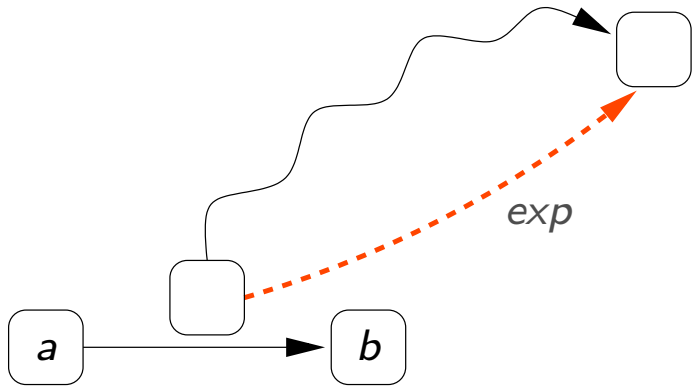
`next`::[*exp*] navigates from node *a* to node *b* if they are connected with an edge that satisfies a path condition given by *exp*.



Nesting allows for existential checking.

Intuition:

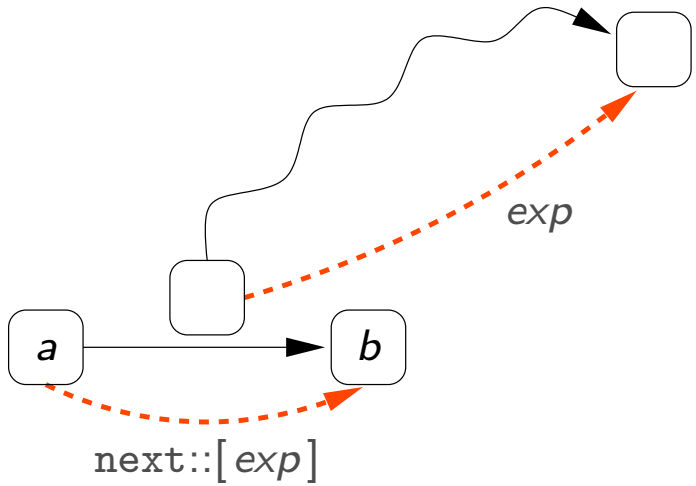
`next`::[*exp*] navigates from node *a* to node *b* if they are connected with an edge that satisfies a path condition given by *exp*.



Nesting allows for existential checking.

Intuition:

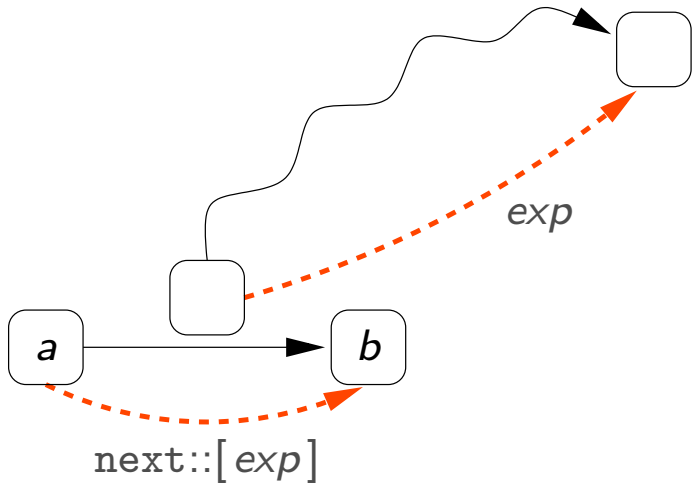
`next::[`*exp*`]` navigates from node *a* to node *b* if they are connected with an edge that satisfies a path condition given by *exp*.



Nesting allows for existential checking.

Intuition:

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

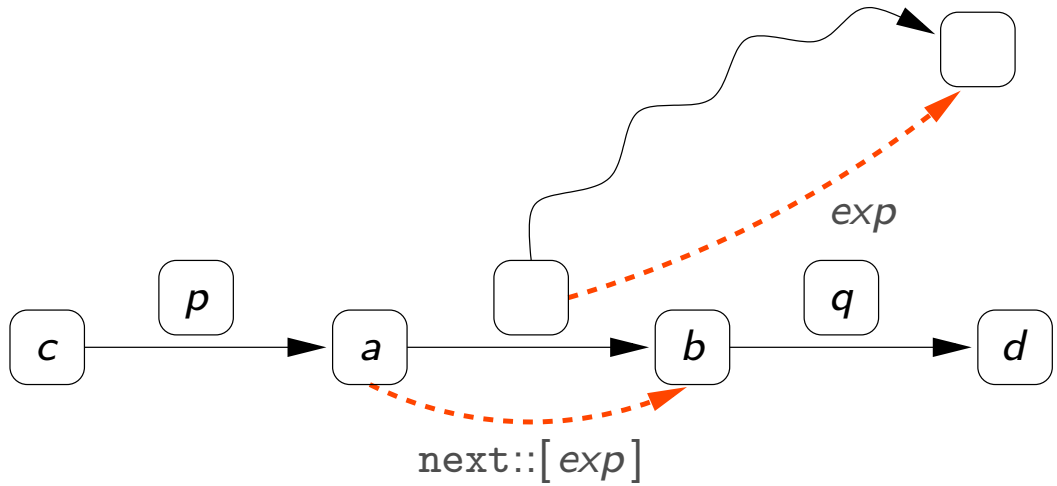


(a, b) is in the evaluation of `next::[`*exp*`]`

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*.

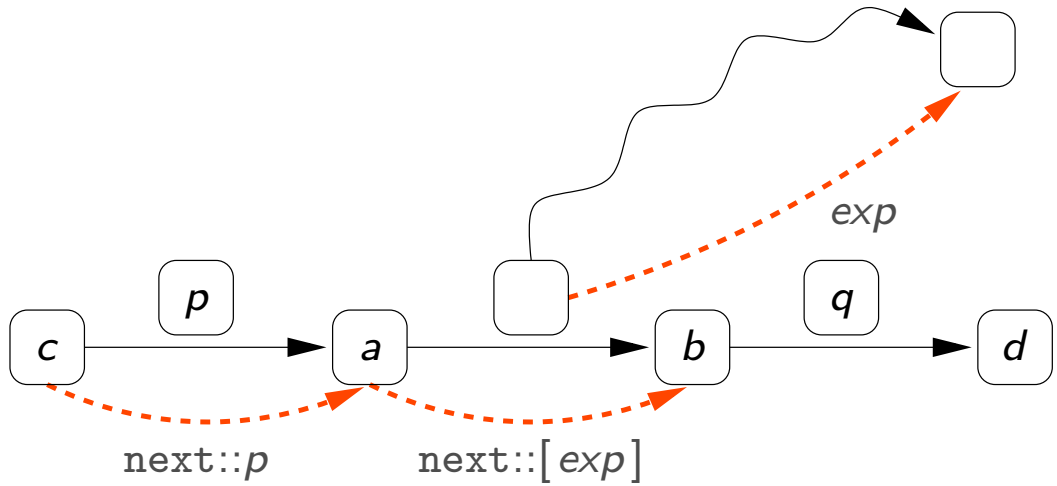


(a, b) is in the evaluation of `next::[`*exp*`]`

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.

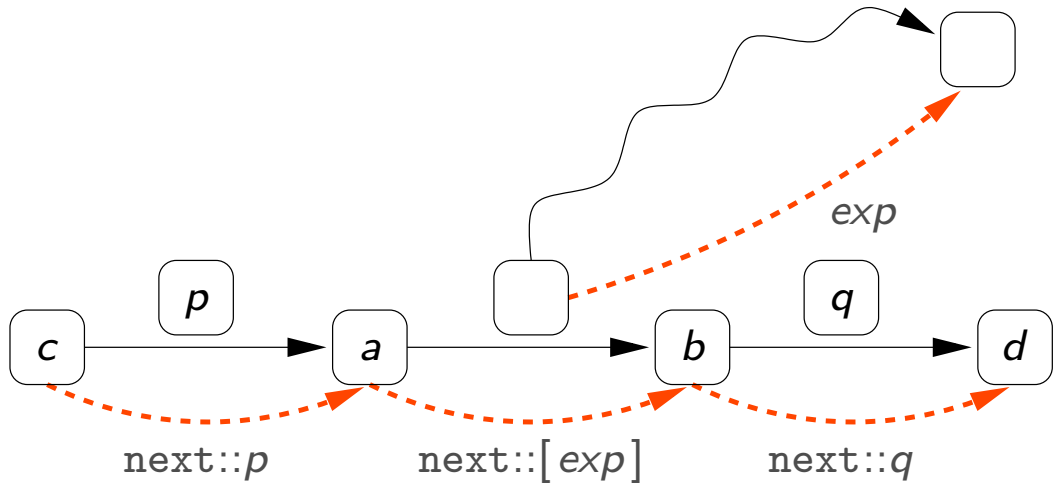


(a, b) is in the evaluation of `next::[exp]`

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.

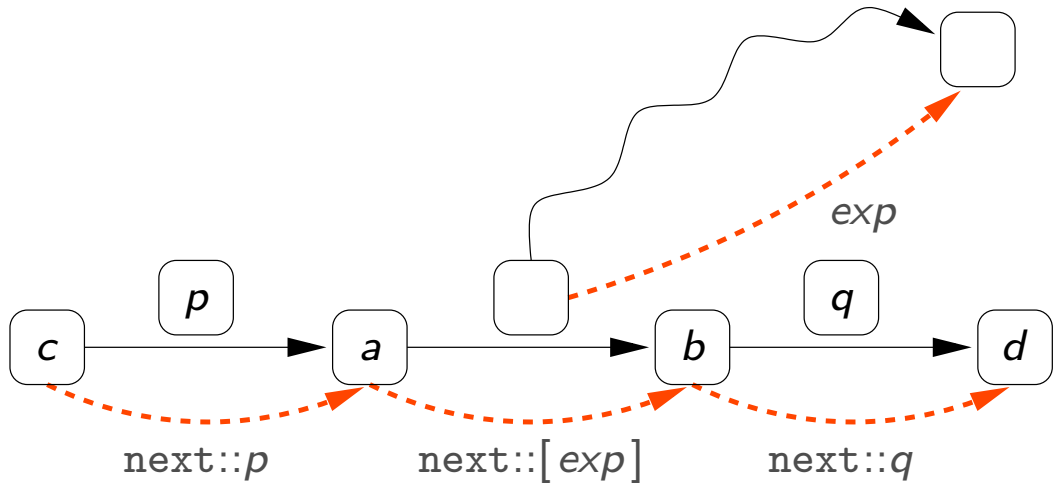


(a, b) is in the evaluation of `next::[exp]`

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*.



(a, b) is in the evaluation of `next::[`*exp*`]`
(c, d) is in the evaluation of `next::`*p*`/next::[`*exp*`]/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:

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

Semantics of nested regular expressions over RDF

Definition

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

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

Semantics of nested regular expressions over RDF

Definition

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

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

Outline

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?

$$(a, b) \in [\![exp]\!]_G?$$

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* $[\![exp]\!]_G$ *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* $[\![exp]\!]_G$ *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* $[\![exp]\!]_G$ *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* $[\![exp]\!]_G$ *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* $[\![exp]\!]_G$ *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* $[\![exp]\!]_G$ *can be done in time*

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

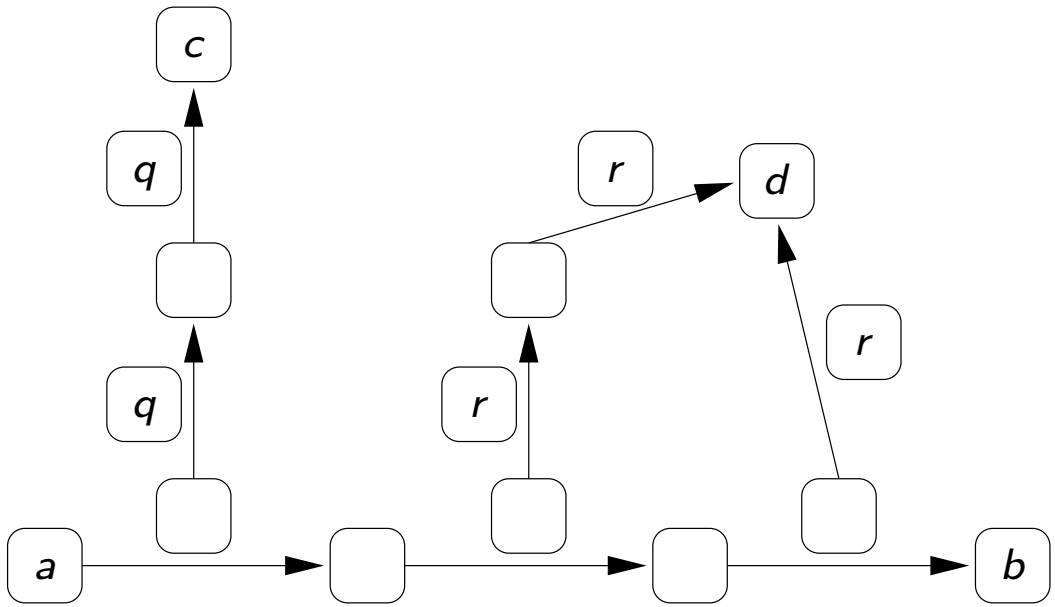
Algorithm (in the style of *temporal logic*)

- 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.

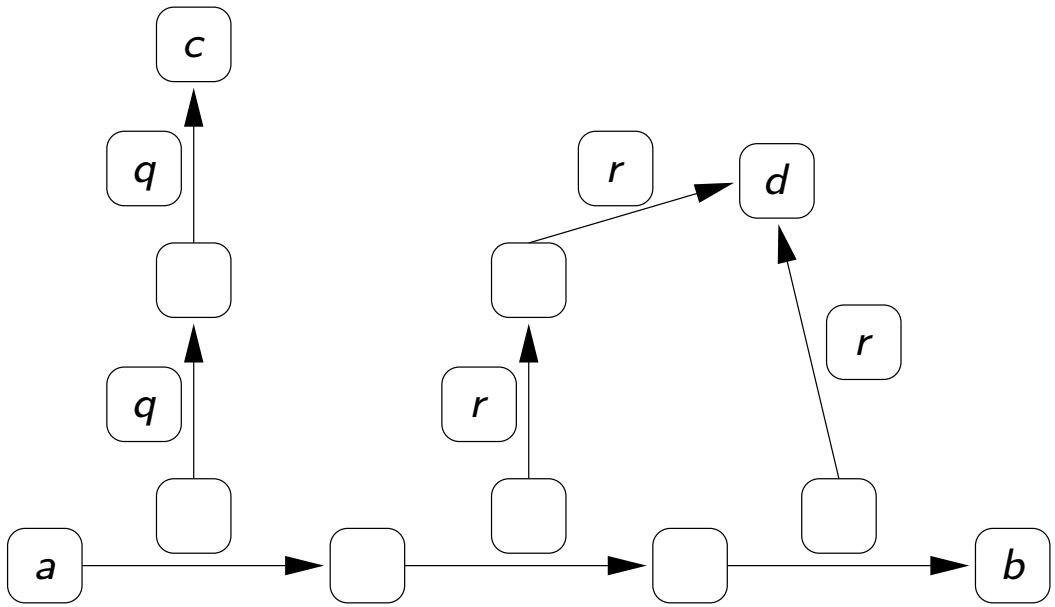
A closer look at the algorithm

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



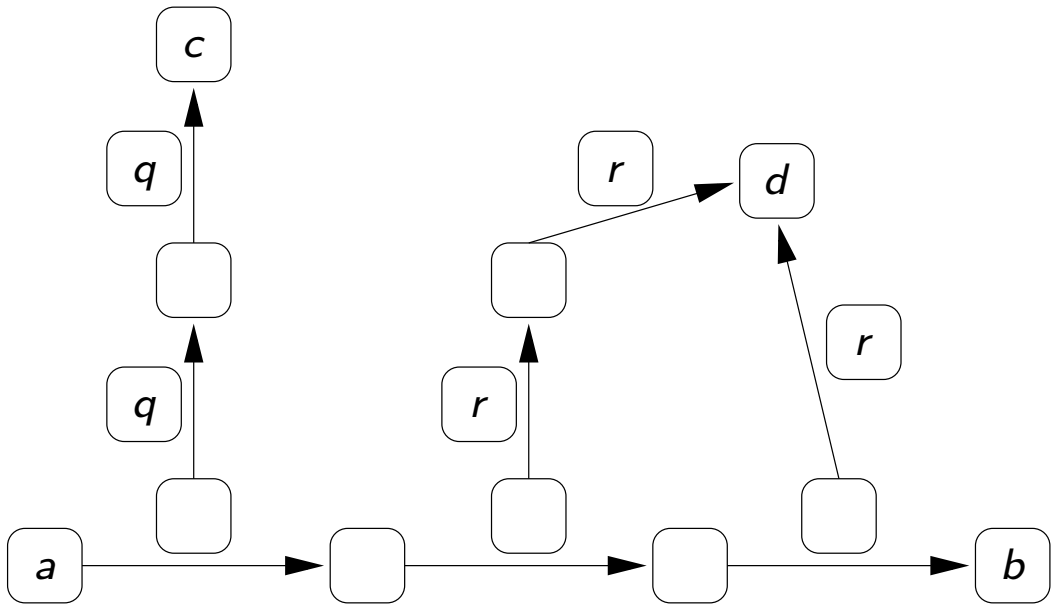
A closer look at the algorithm

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



A closer look at the algorithm

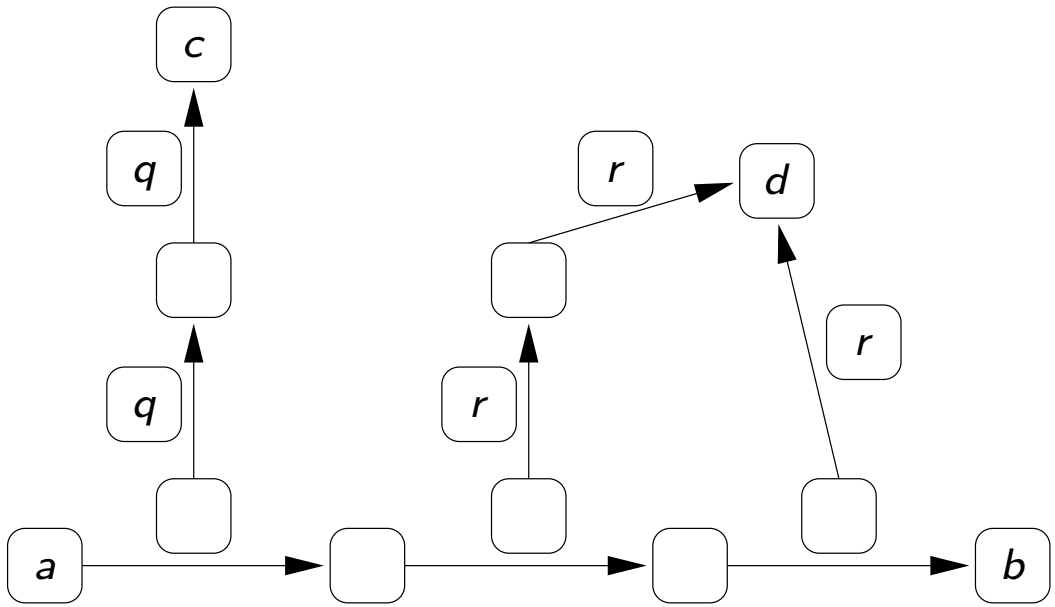
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

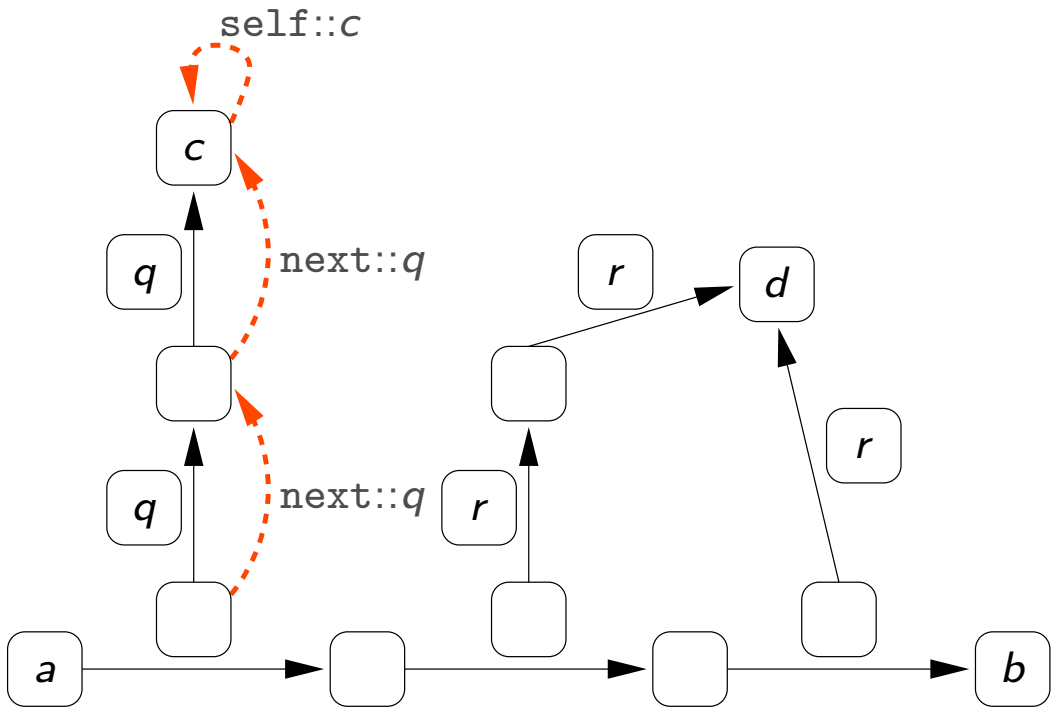
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

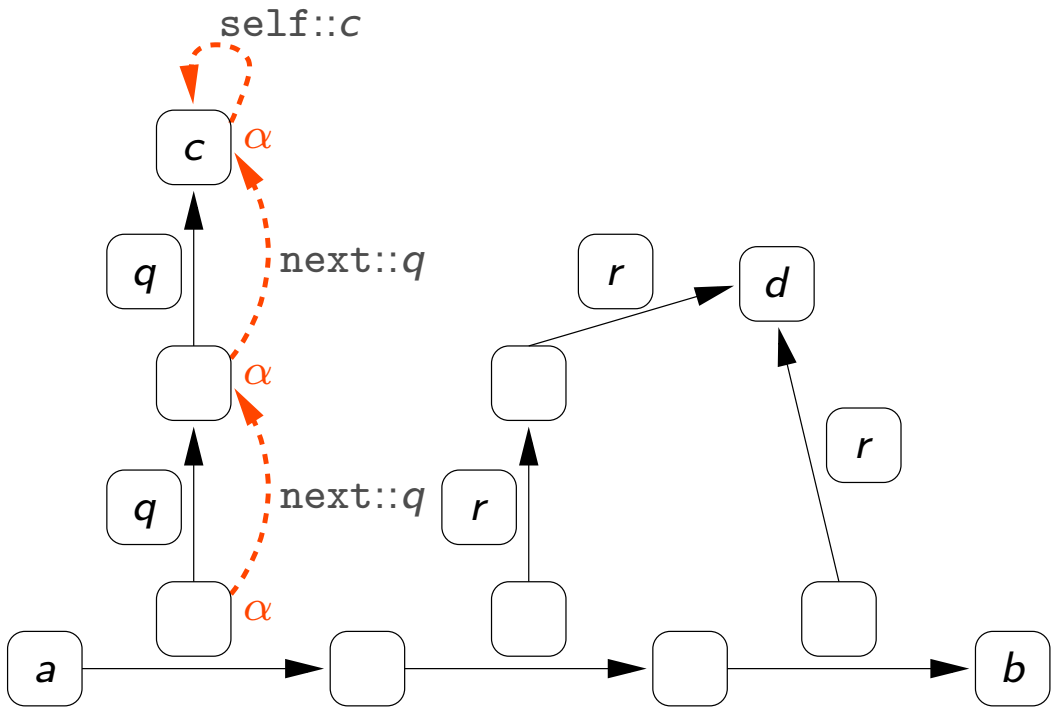
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

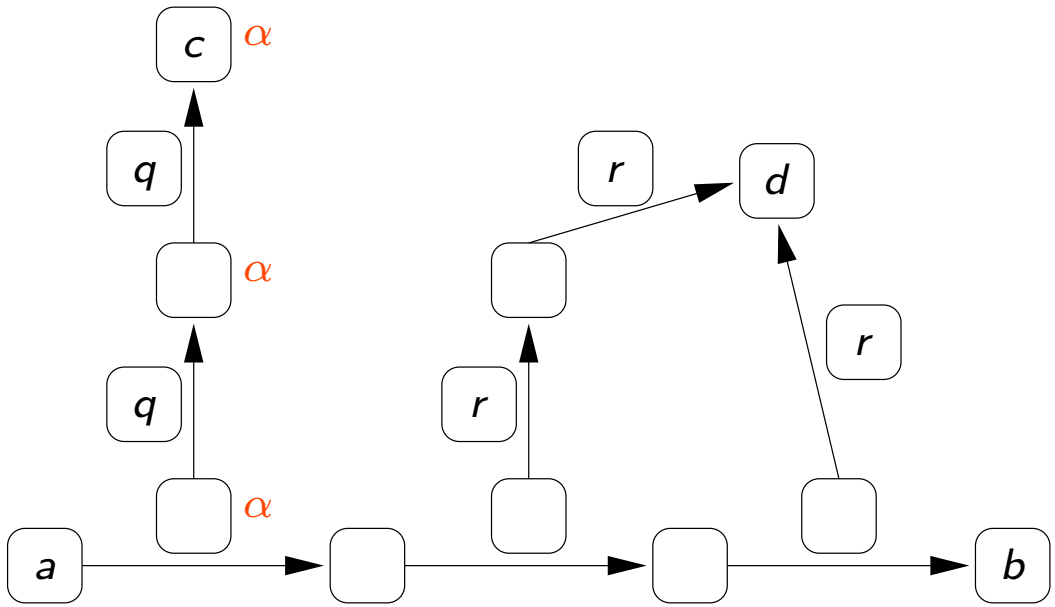
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

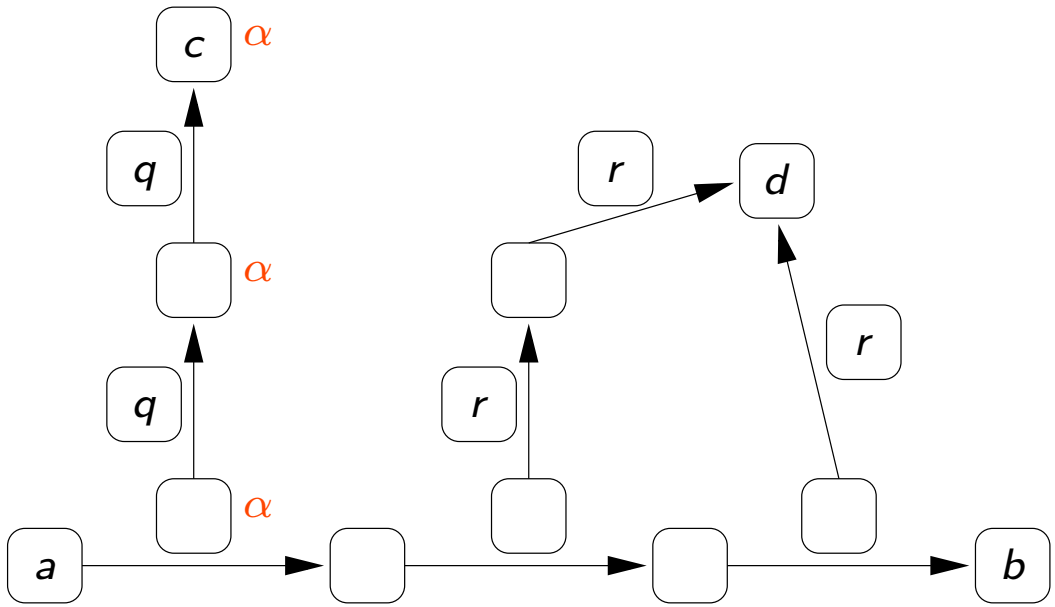
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

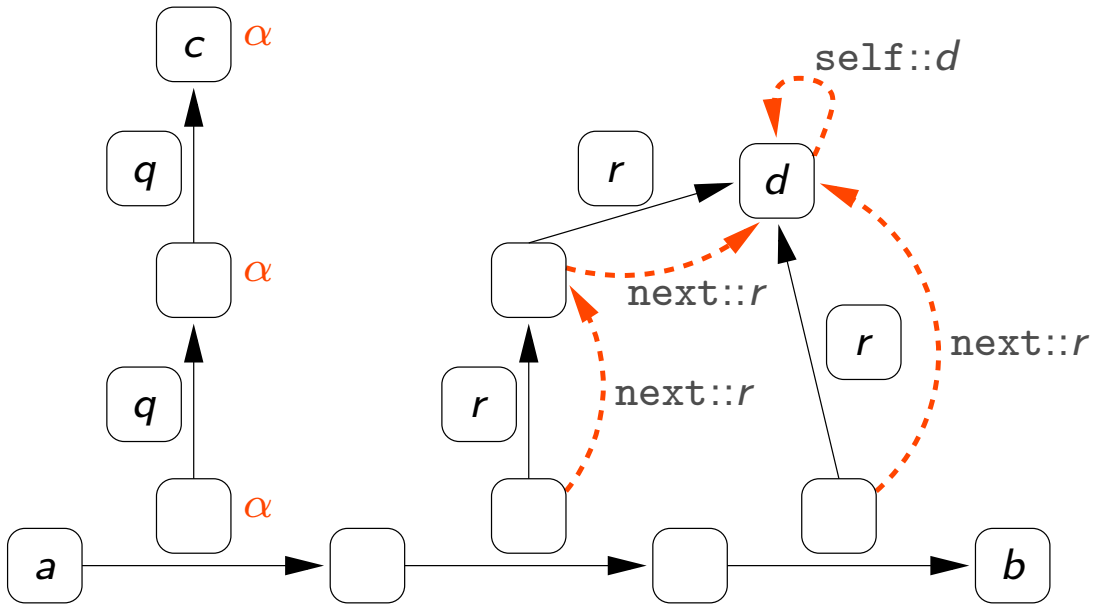
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

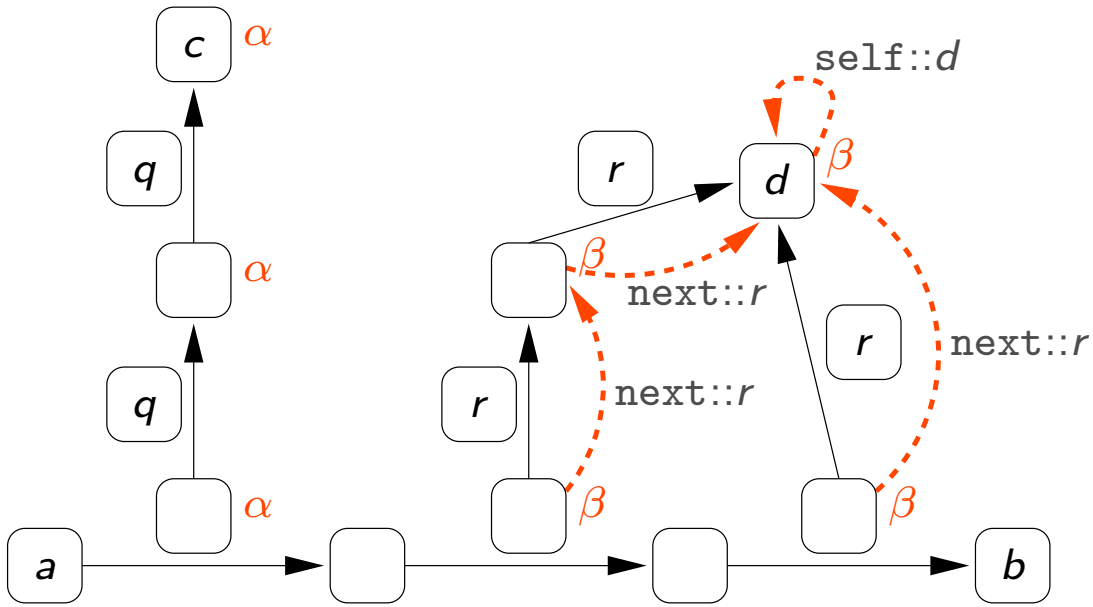
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

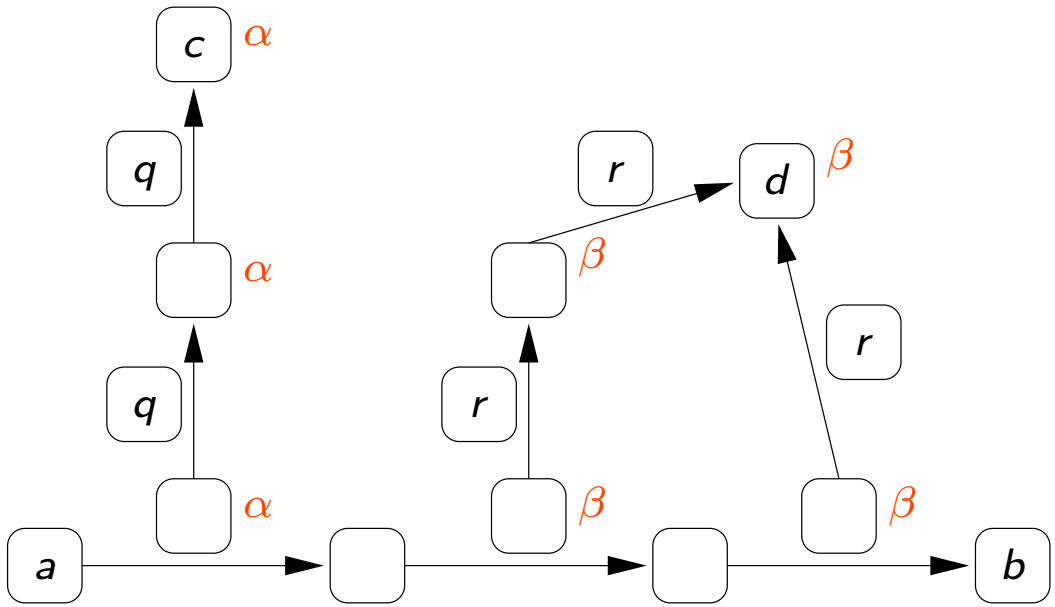
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

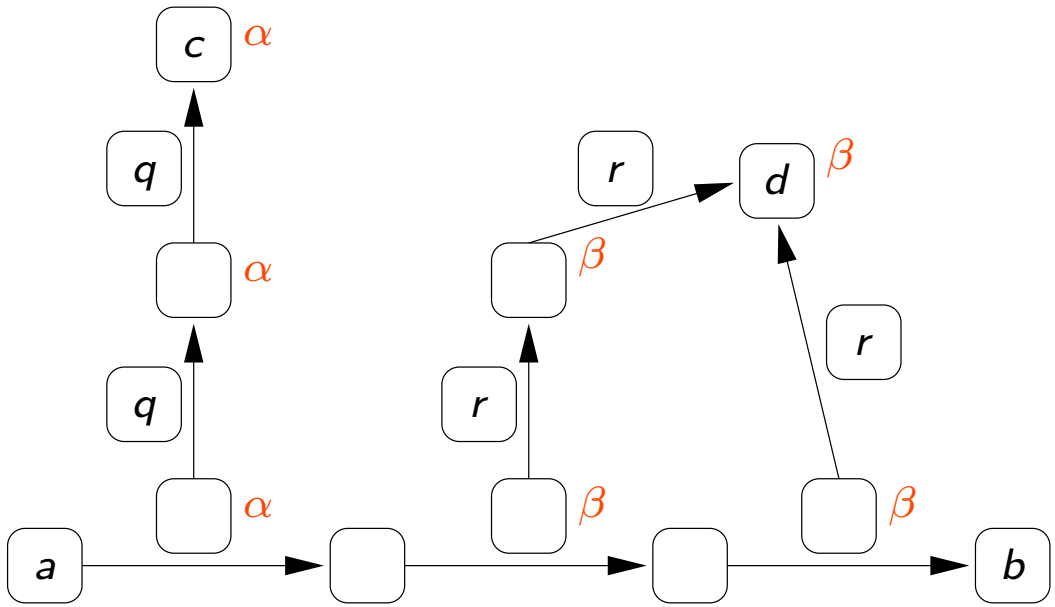
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



$$\alpha \quad = \quad (\texttt{next::}q)^*/\texttt{self::}c$$
$$\beta \quad = \quad (\texttt{next::}r)^*/\texttt{self::}d$$

A closer look at the algorithm

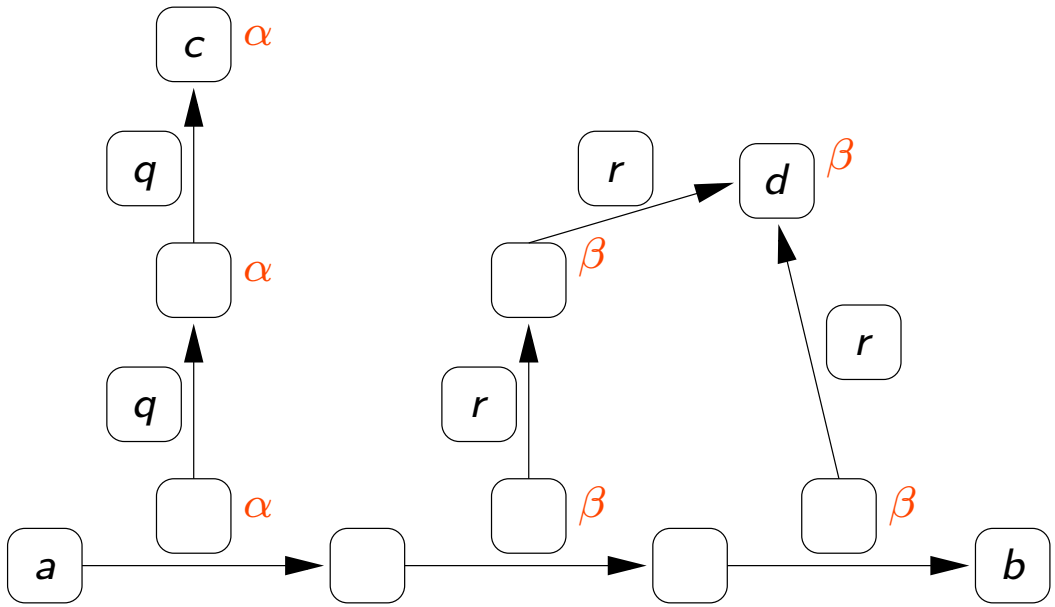
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



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

A closer look at the algorithm

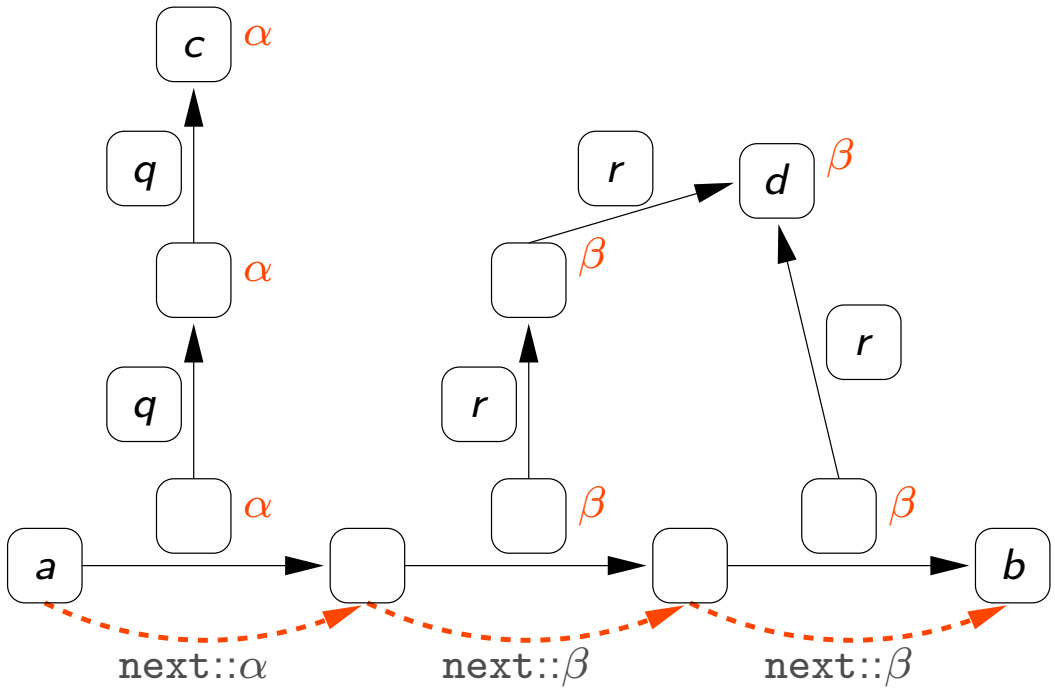
$$exp \quad = \quad \texttt{next::}[(\texttt{next::}q)^*/\texttt{self::}c]/(\texttt{next::}[(\texttt{next::}r)^*/\texttt{self::}d])^*$$



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

A closer look at the algorithm

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



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

Outline

SPARQL

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

original SPARQL syntax	algebraic syntax
`?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 ) }`	$(P_1 \text{ FILTER } R)$

SPARQL

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

original SPARQL syntax	algebraic syntax
`?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 ) }`	$(P_1 \text{ FILTER } R)$

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

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

nSPARQL: semantics

Given an RDF graph G

$\{?X \to a, ?Y \to b\}$ 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

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

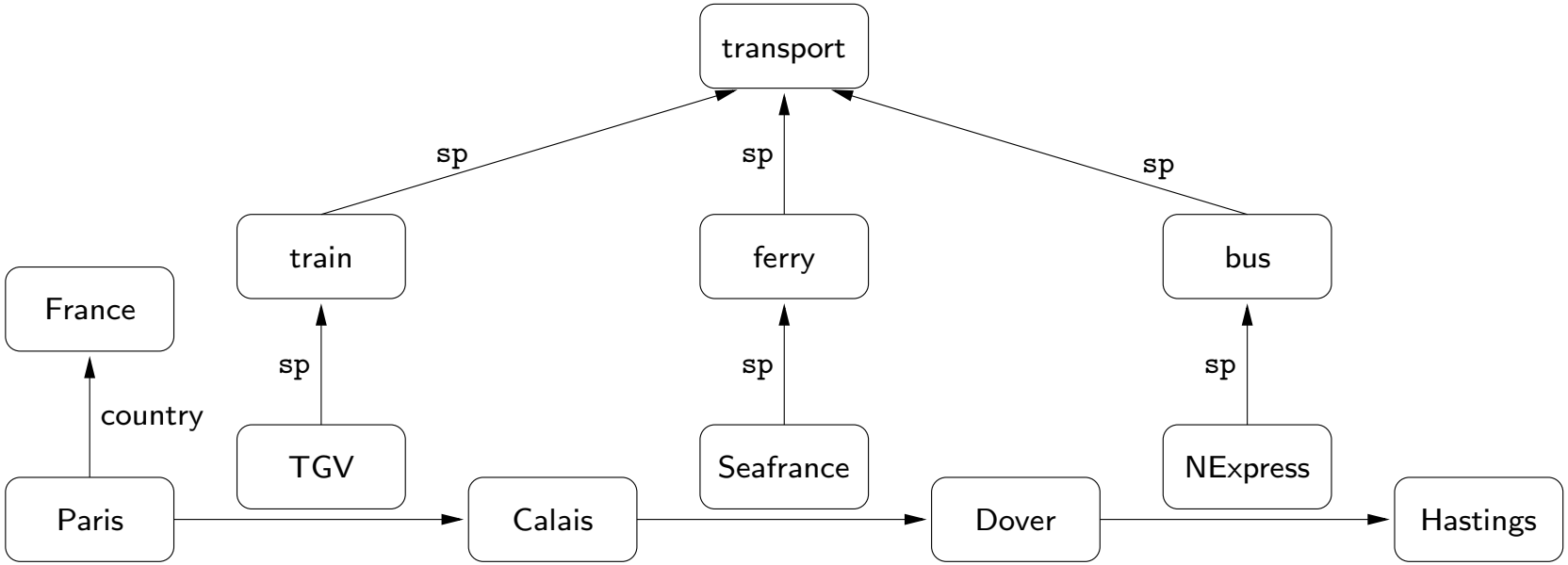
The semantics of AND, OPT, UNION, FILTER, is *inherited* from SPARQL.

Note:

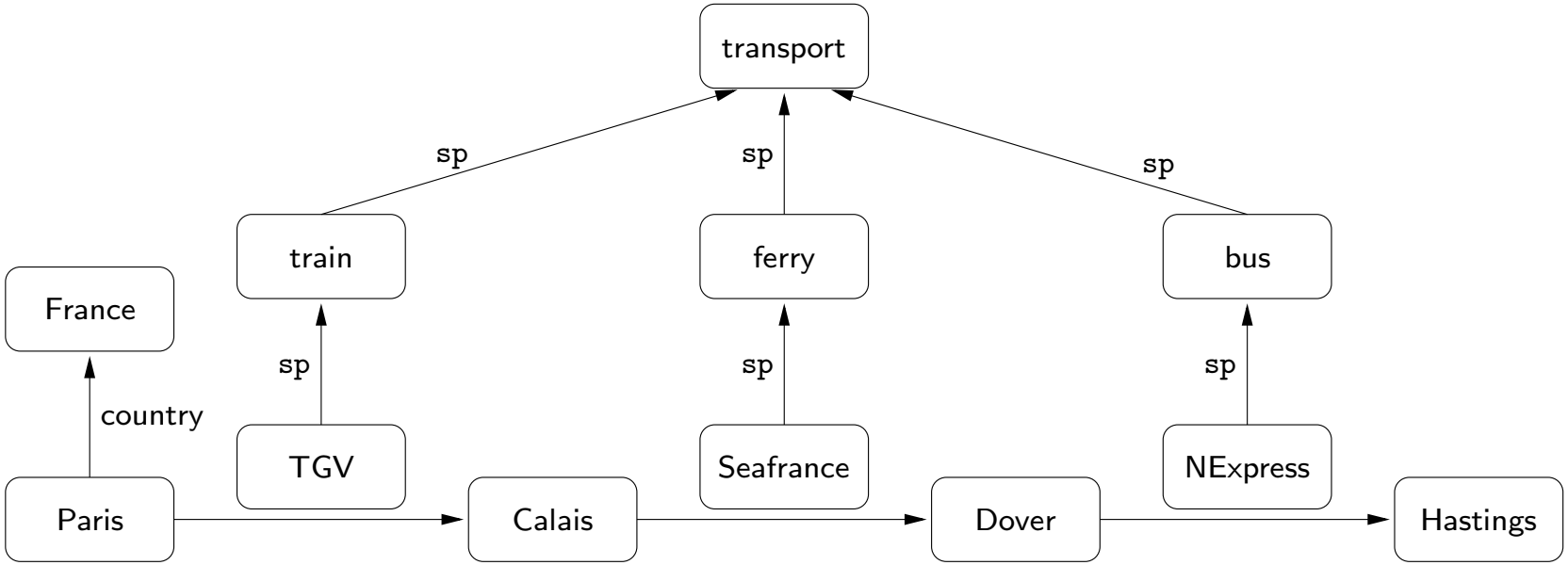
Nested regular expressions do not add extra complexity.

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

nSPARQL examples

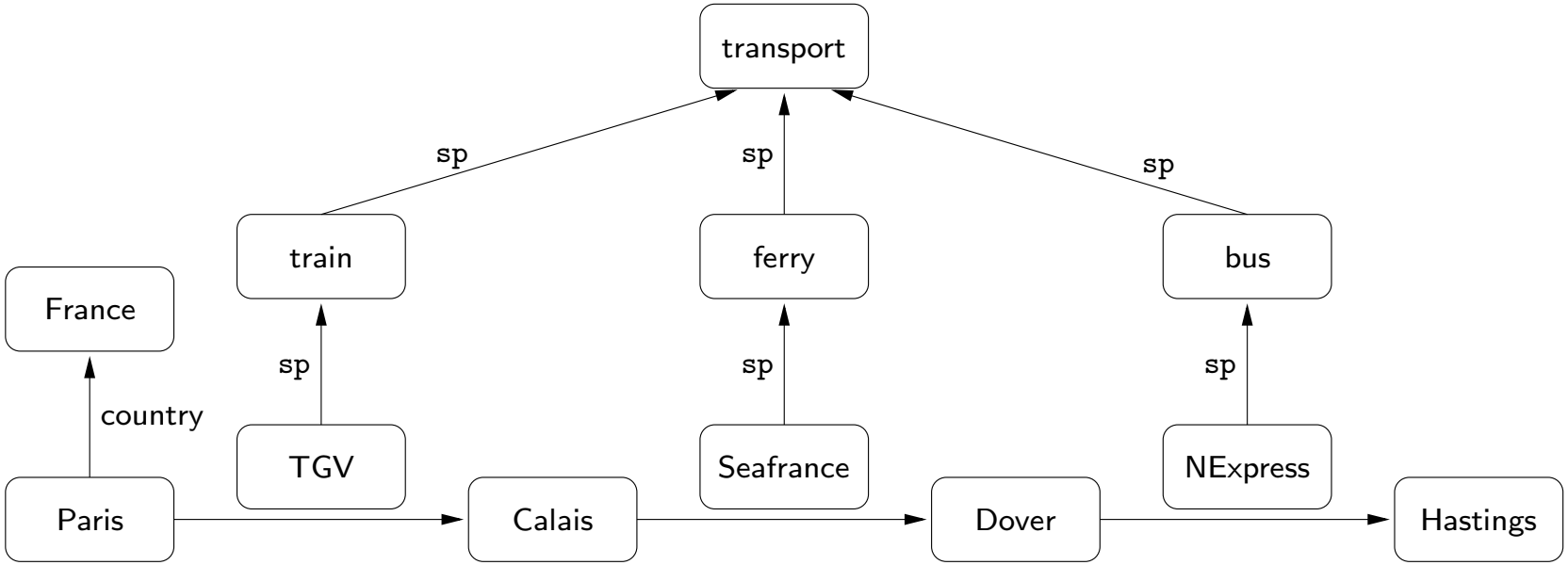


nSPARQL examples



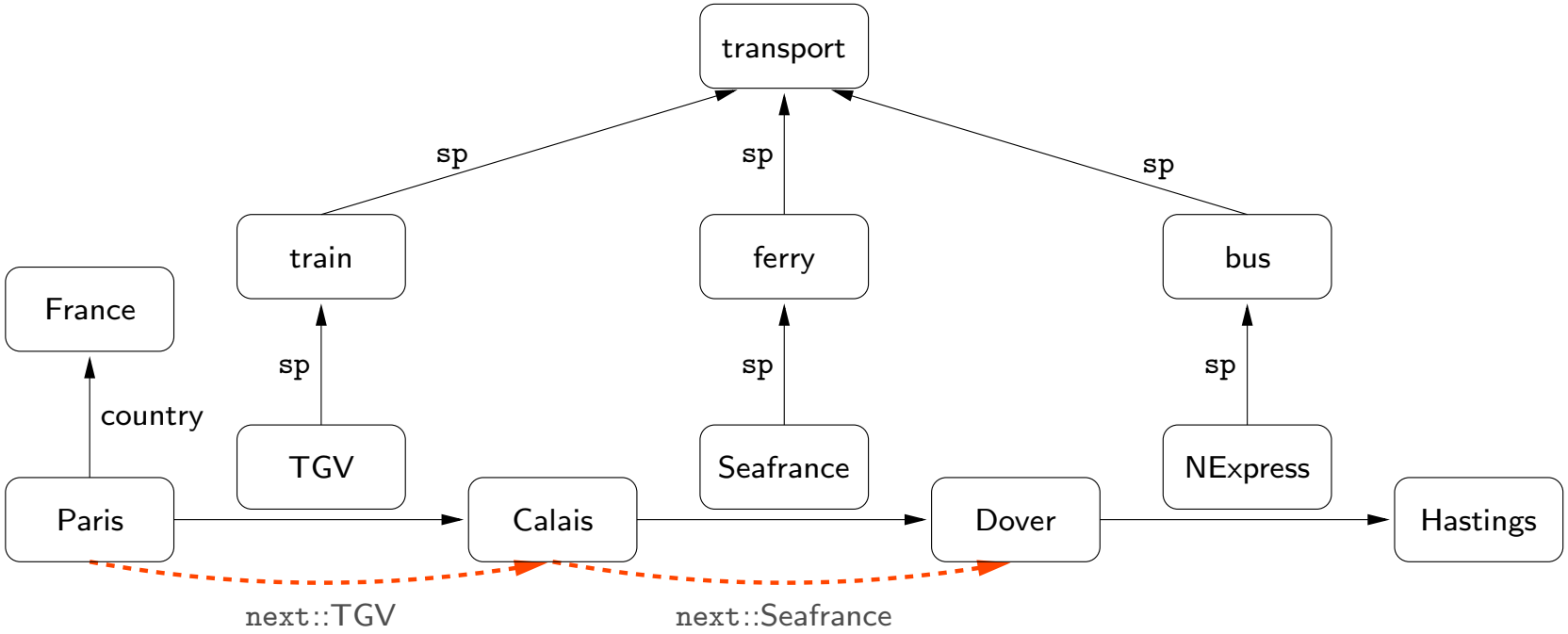
$(?X,\ (\texttt{next}::\text{TGV} \mid \texttt{next}::\text{SeaFrance})^{+},\ \text{Dover})\ \text{AND}\ (?X,\ \texttt{next}::\text{country},\ ?Y)$

nSPARQL examples



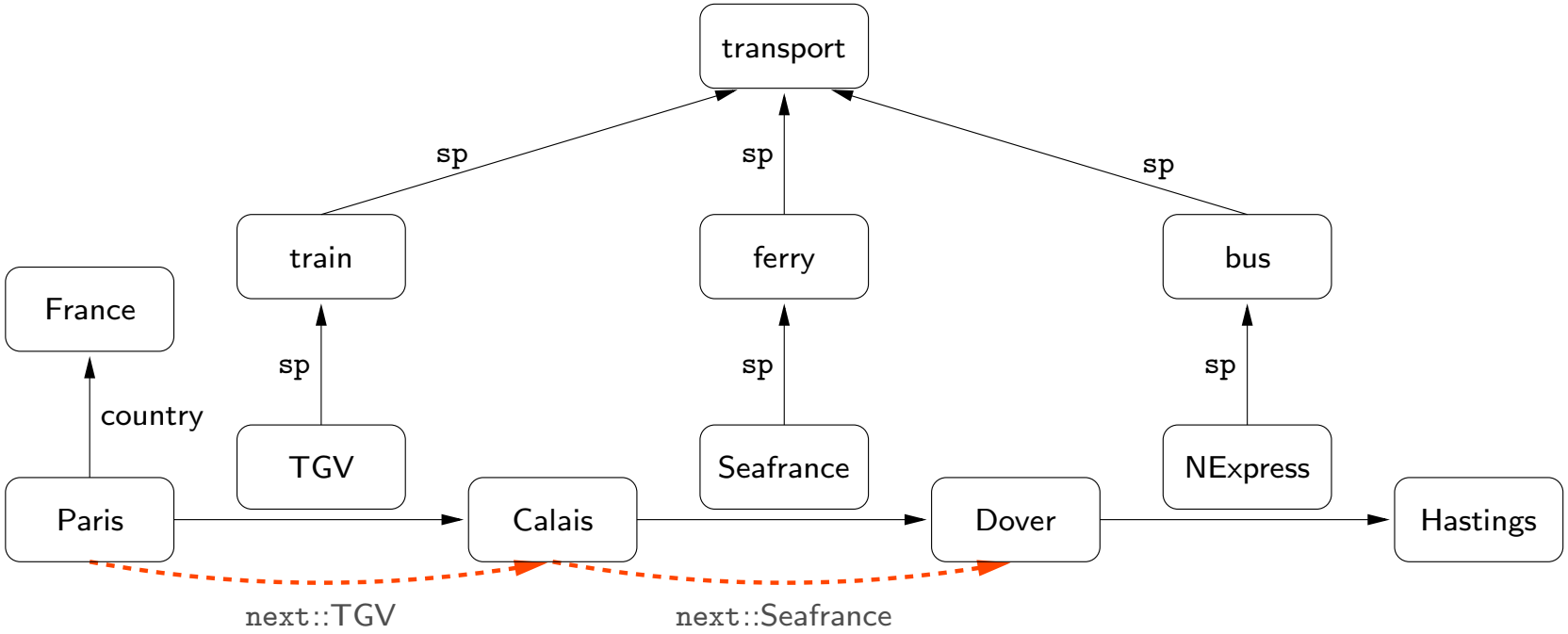
$(?X,\ (\texttt{next::TGV} \mid \texttt{next::SeaFrance})^{+},\ \text{Dover})\ \text{AND}\ (?X,\ \texttt{next::country},\ ?Y)$

nSPARQL examples



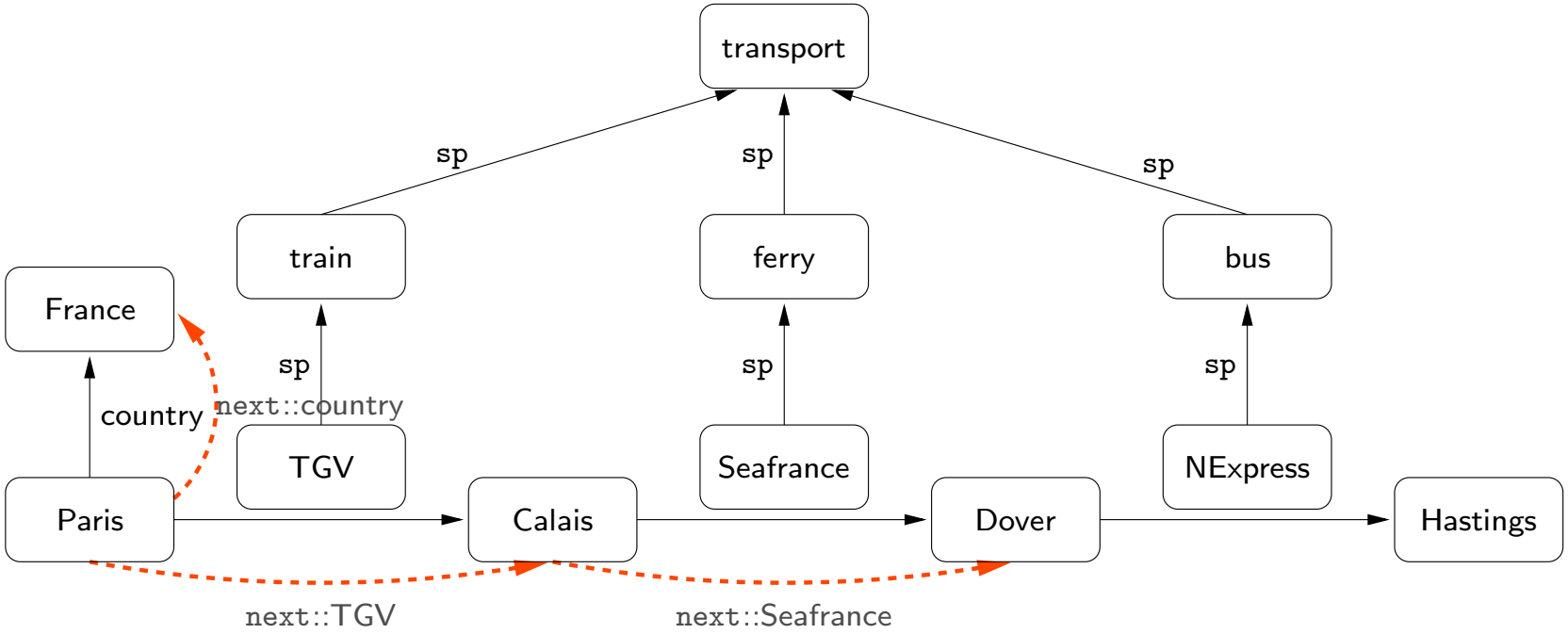
$(?X,\ (\texttt{next::TGV} \mid \texttt{next::SeaFrance})^{+},\ \text{Dover})\ \text{AND}\ (?X,\ \texttt{next::country},\ ?Y)$

nSPARQL examples



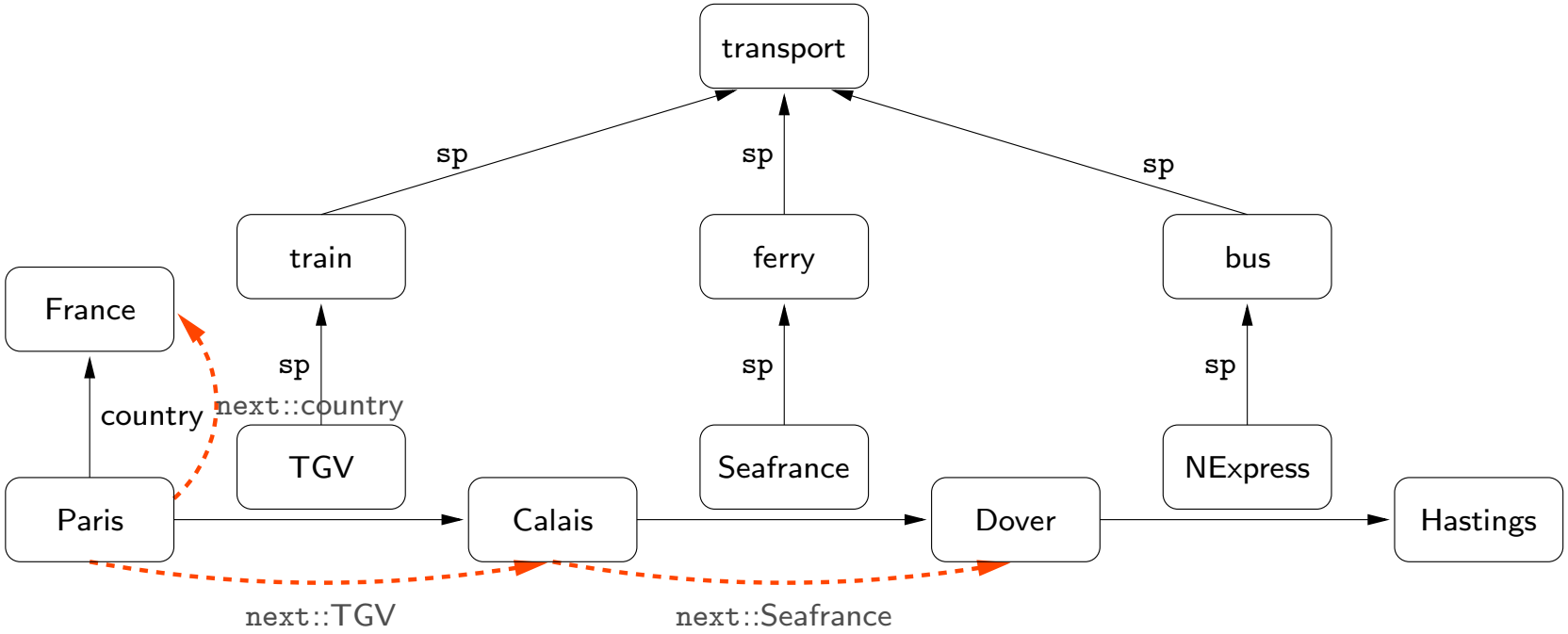
$(?X,\ (\texttt{next::TGV} \mid \texttt{next::SeaFrance})^{+},\ \text{Dover})\ \text{AND}\ (?X,\ \texttt{next::country},\ ?Y)$

nSPARQL examples



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

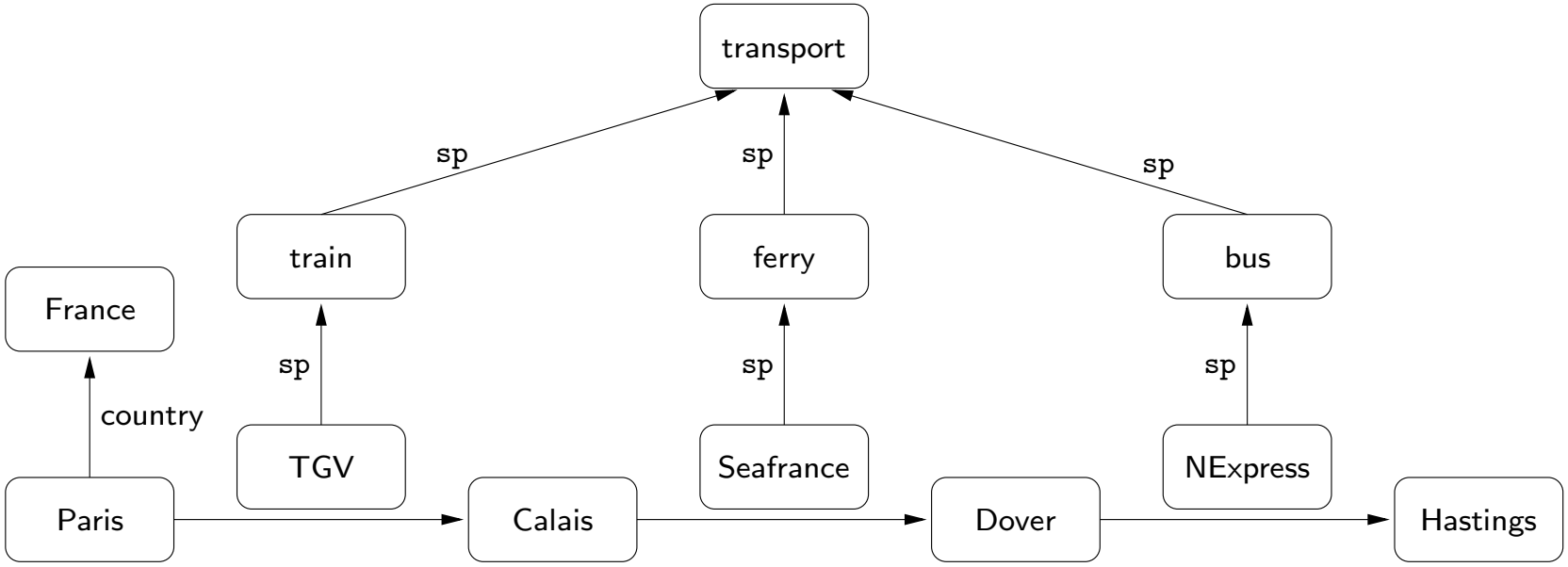
nSPARQL examples



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

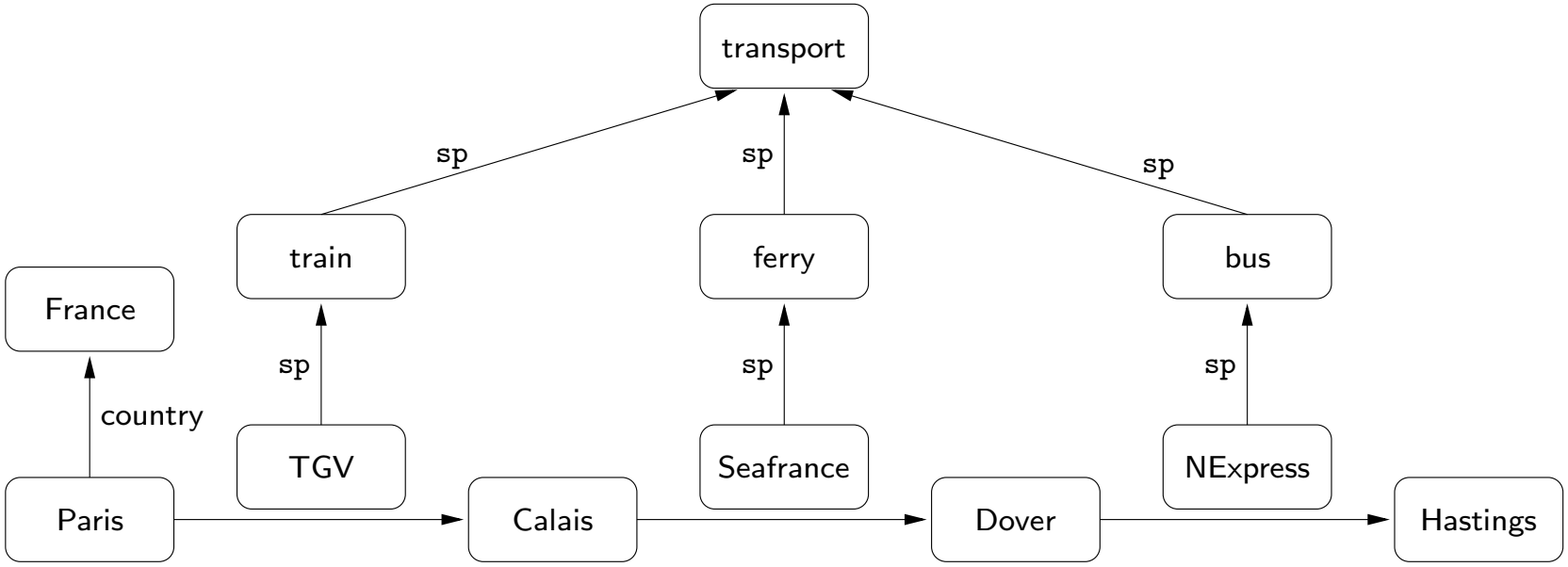
$\{?X \rightarrow \text{Paris}, ?Y \rightarrow \text{France}\}$

nSPARQL examples



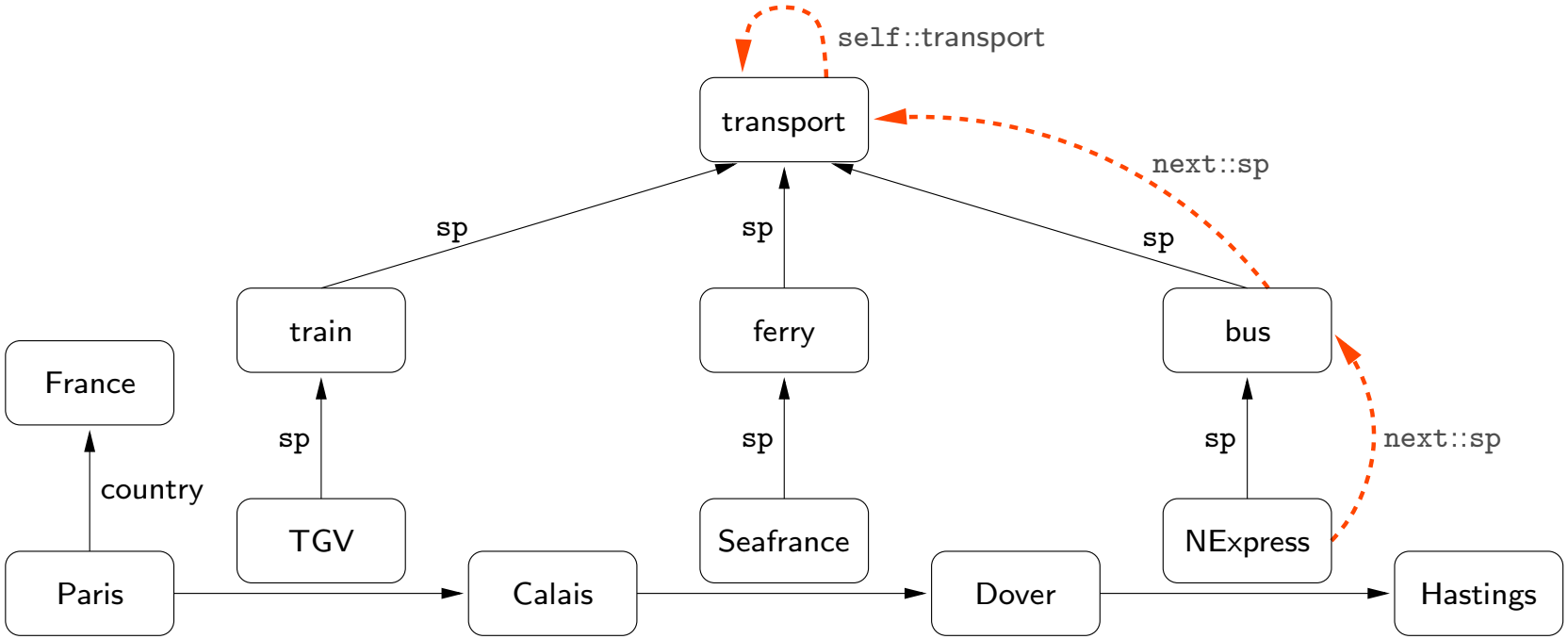
$(?X,\ \texttt{next::[(next::sp)*/self::transport]},\ \text{Hastings})$

nSPARQL examples



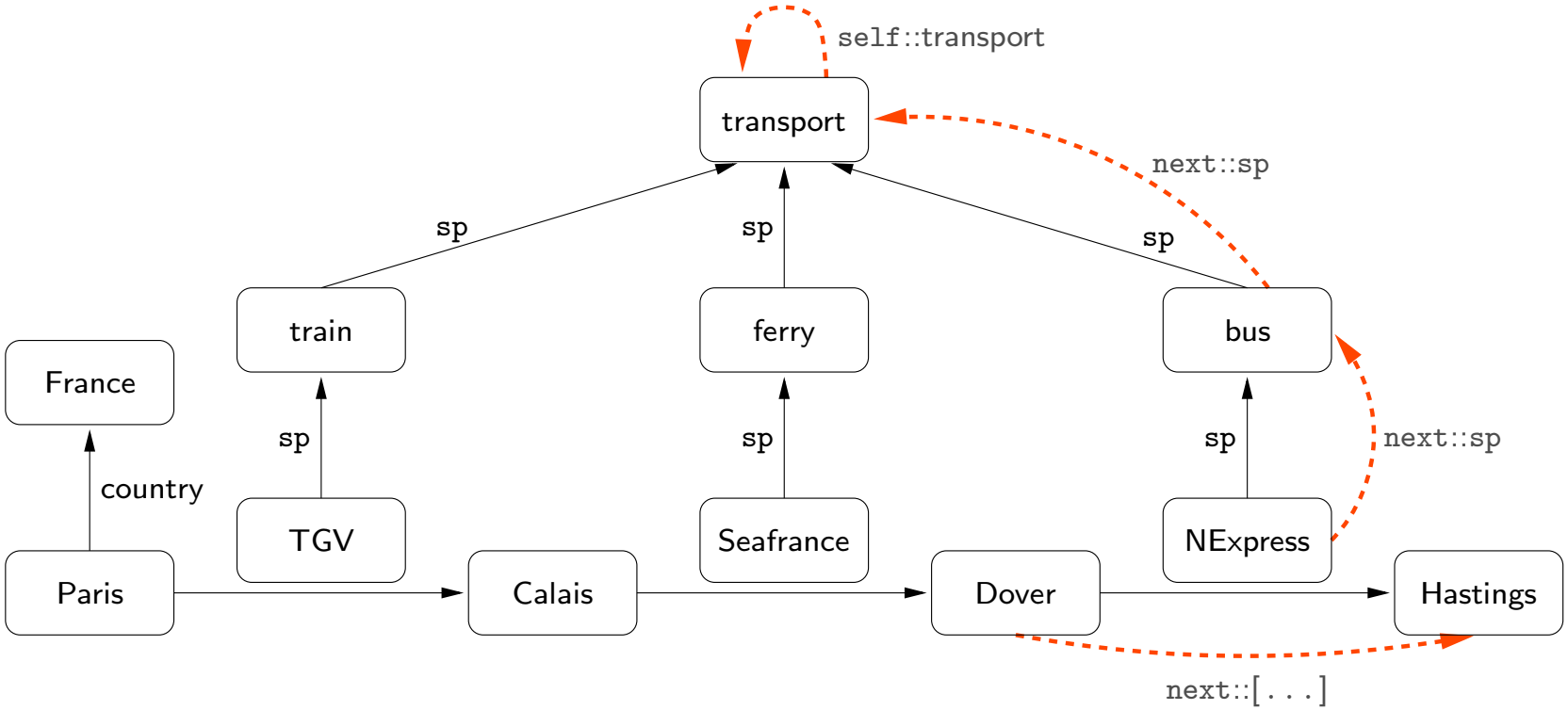
$(?X,\ \texttt{next::[(next::sp)*/self::transport]},\ \text{Hastings})$

nSPARQL examples



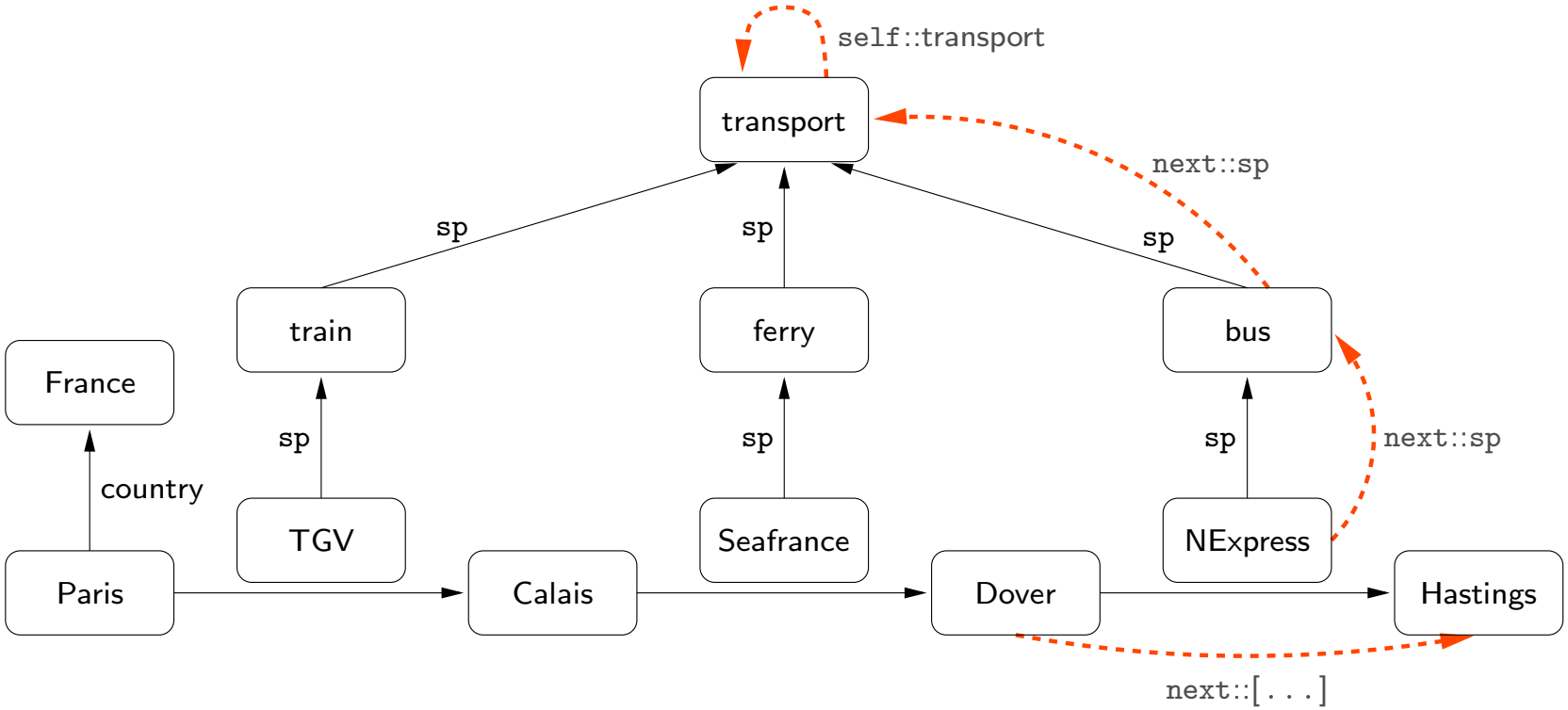
$(?X,\ \texttt{next::}[(\texttt{next::sp})^*/\texttt{self::transport}],\ \text{Hastings})$

nSPARQL examples



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

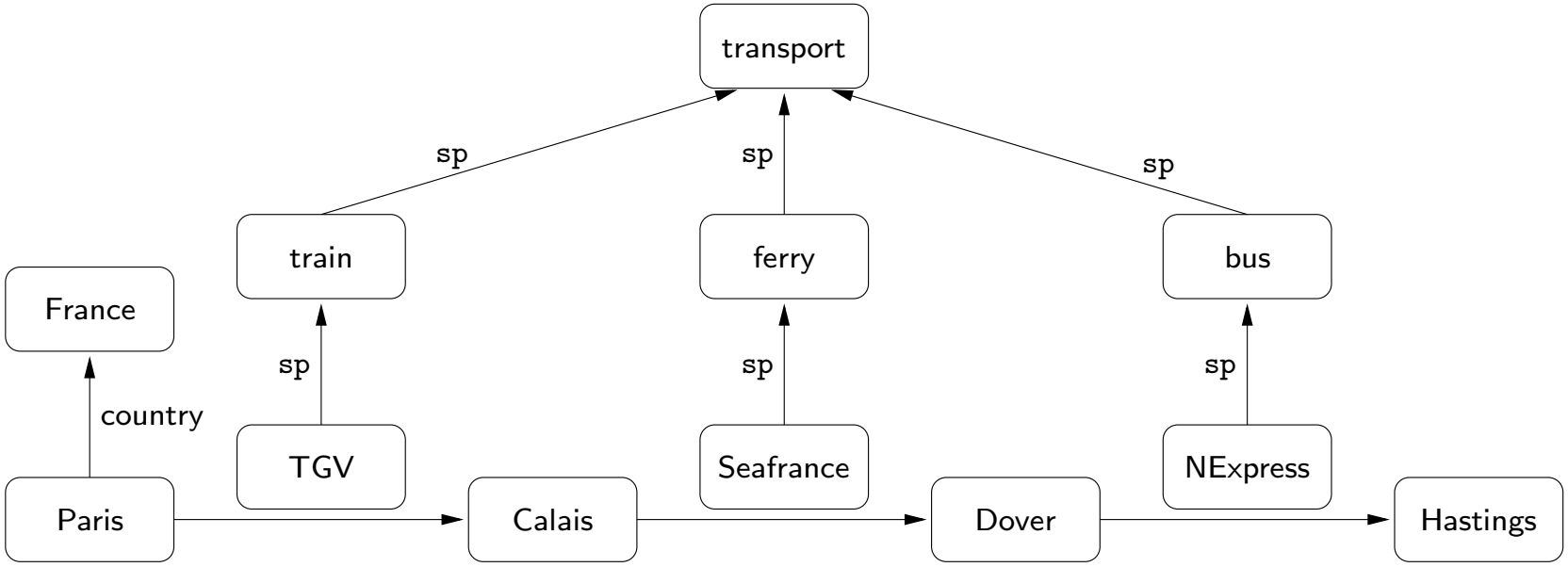
nSPARQL examples



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

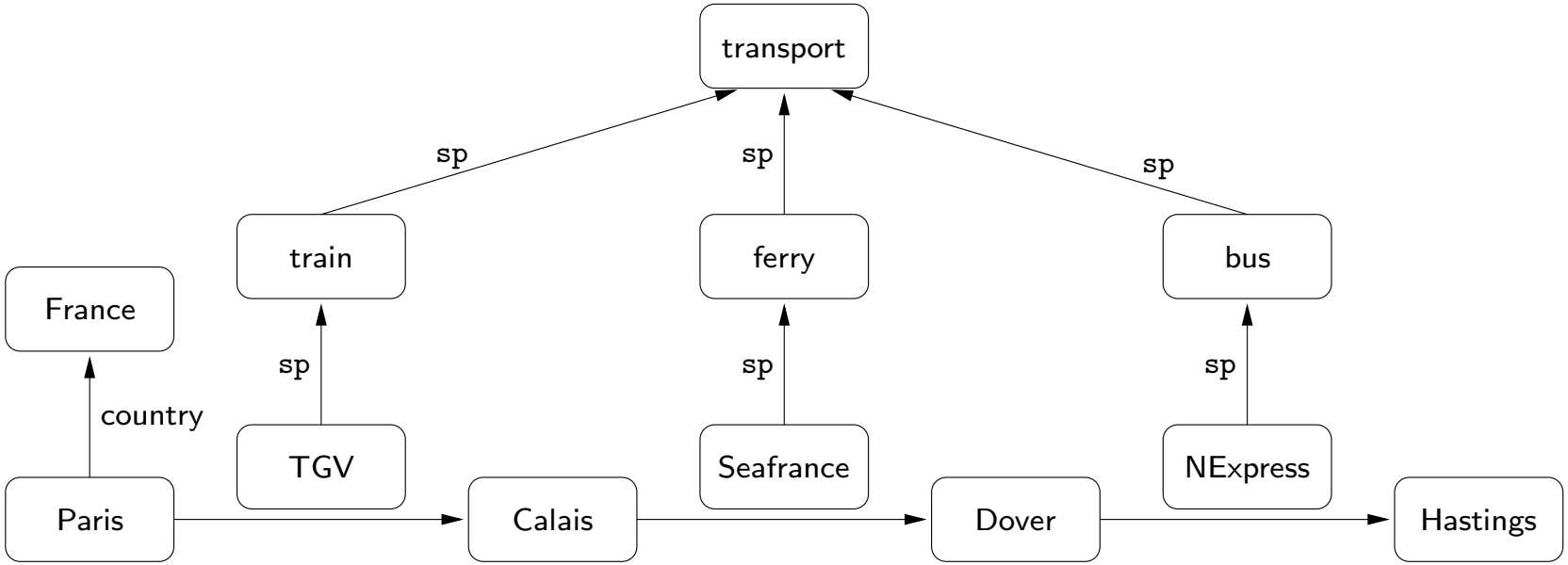
$\{?X \rightarrow \text{Dover}\}$

nSPARQL examples



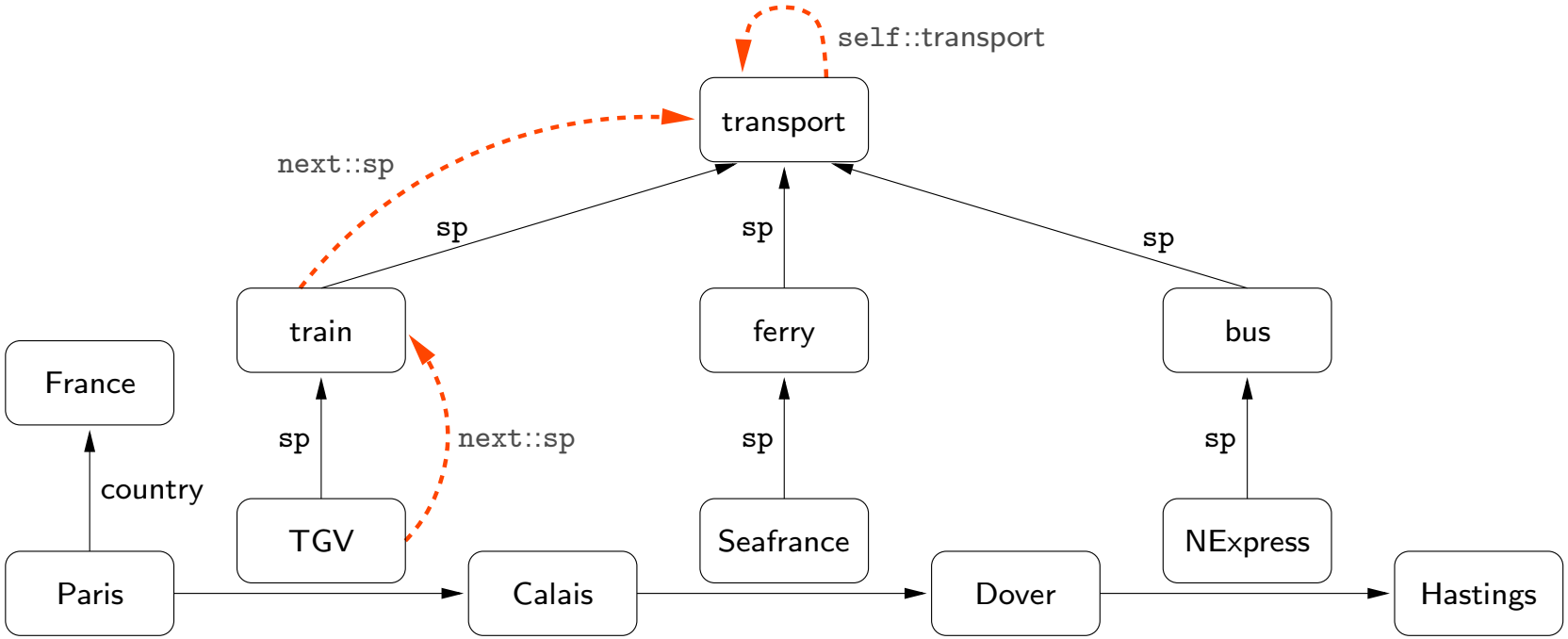
$$(?X,\ \ (\texttt{next::[(next::sp)}^*\texttt{/self::transport]})^+,\ \text{Hastings})$$

nSPARQL examples



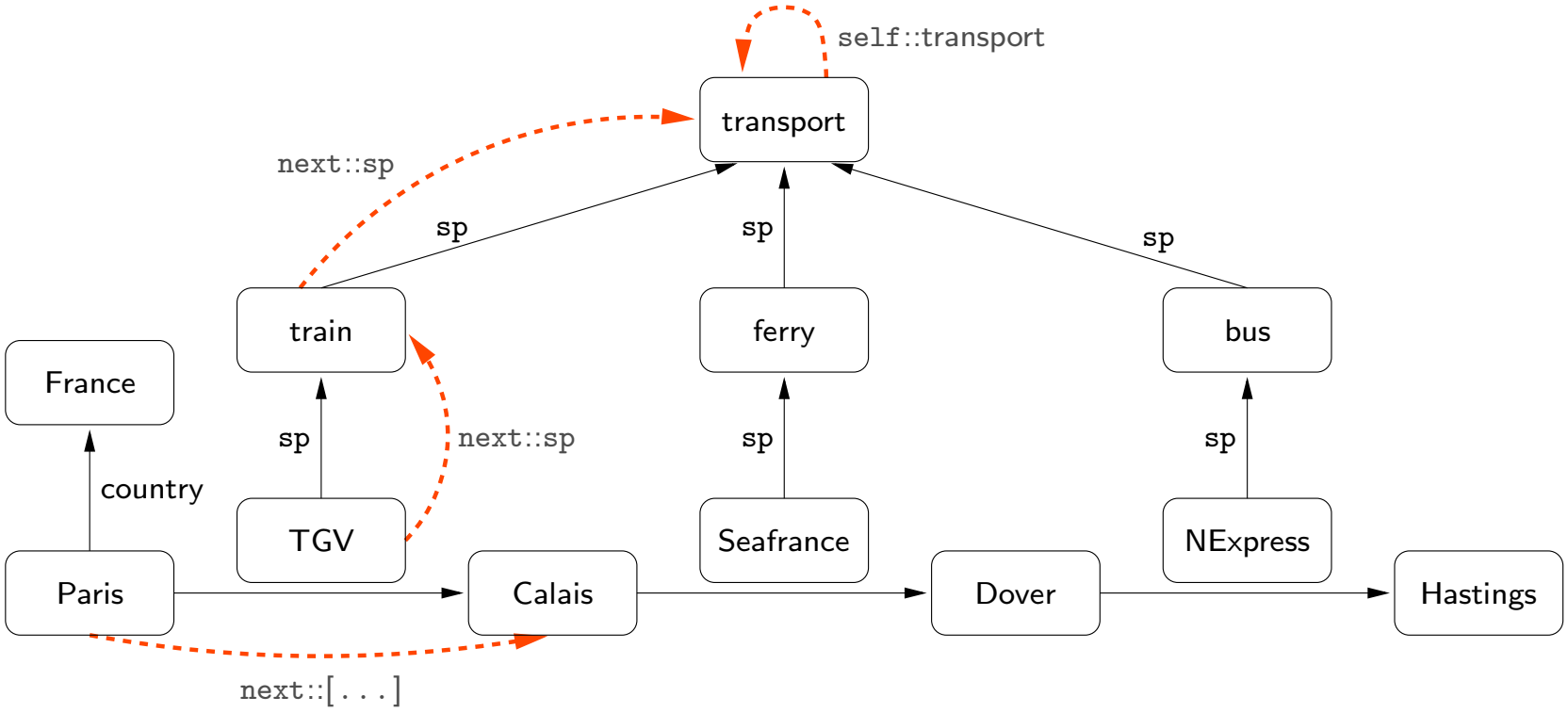
$$(?X,\ \ (\texttt{next::[(next::sp)}^{*}\texttt{/self::transport]})^{+},\ \text{Hastings})$$

nSPARQL examples



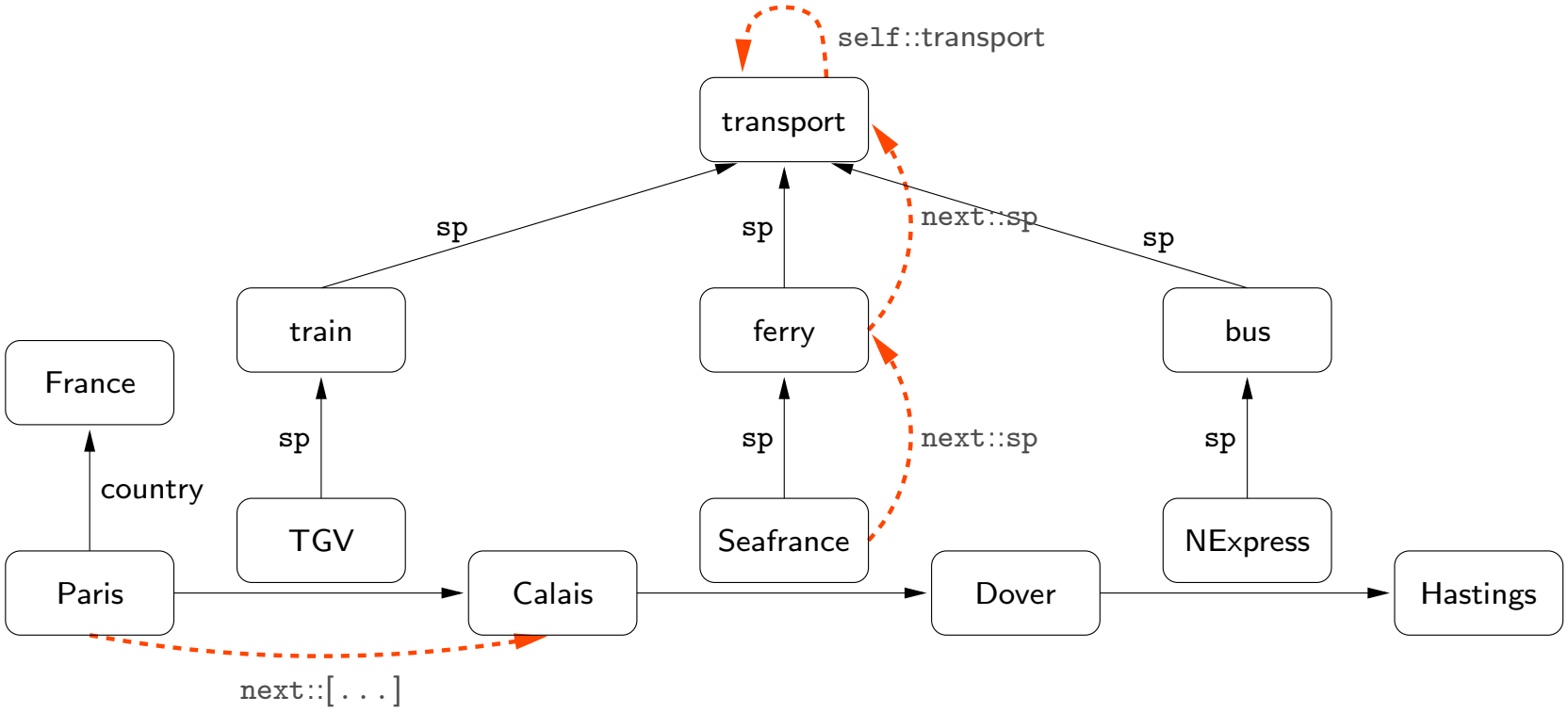
$$(?X,\ \ (\texttt{next::}[(\texttt{next::sp})^*/\texttt{self::transport}])^+,\ \text{Hastings})$$

nSPARQL examples



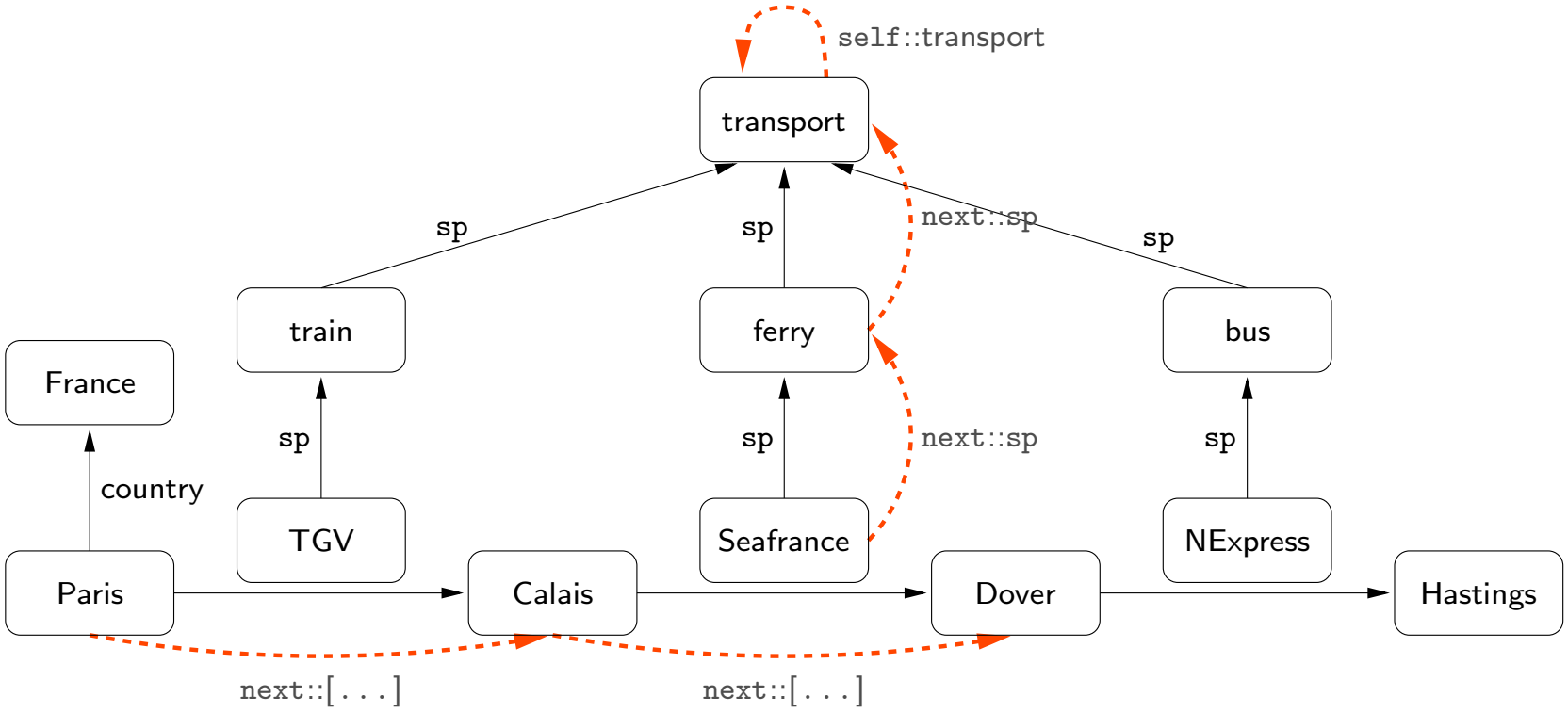
$$(?X,\ \ (\texttt{next::[(next::sp)}^*\texttt{/self::transport]})^+,\ \text{Hastings})$$

nSPARQL examples



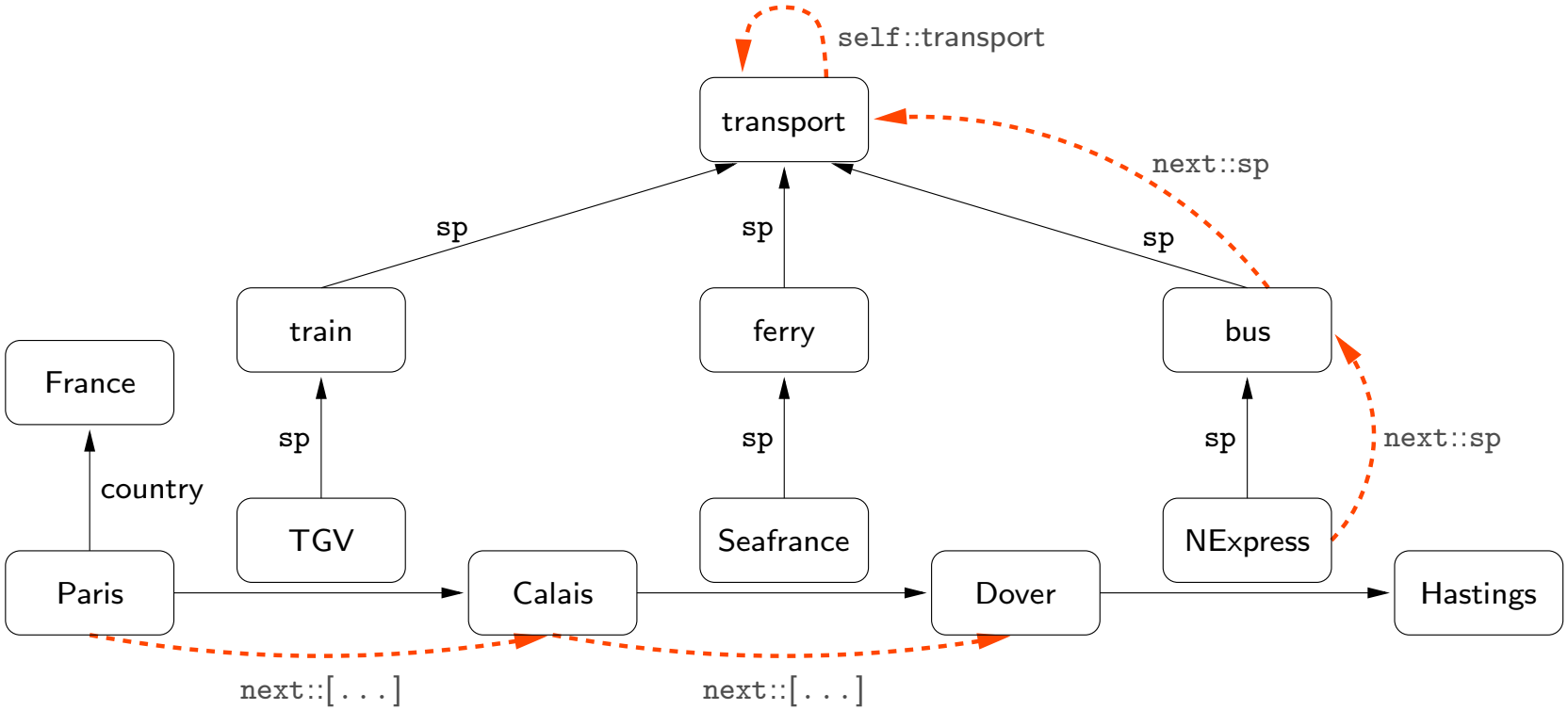
$$(?X,\ \ (\texttt{next::}[(\texttt{next::sp})^*/\texttt{self::transport}])^+,\ \text{Hastings})$$

nSPARQL examples



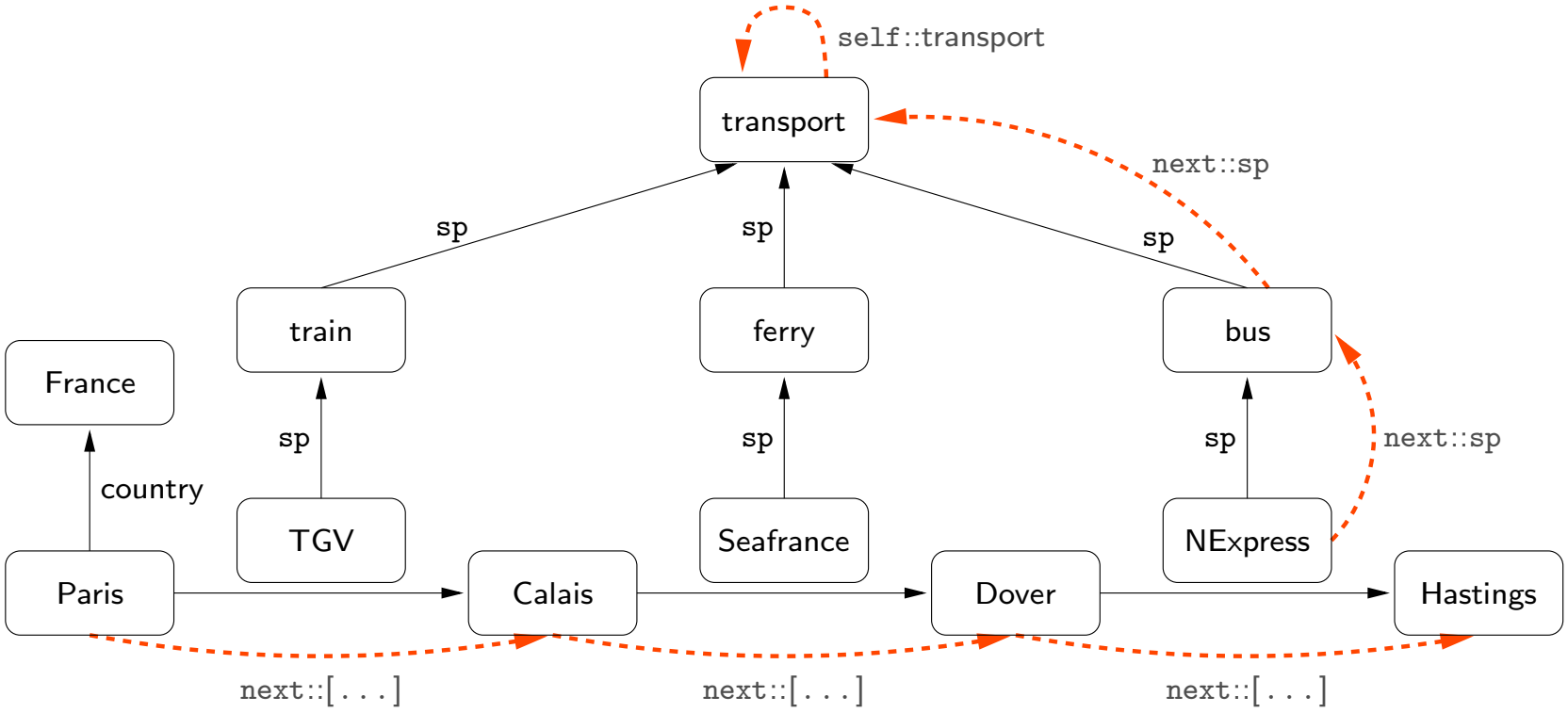
$$(?X,\ \ (\texttt{next::}[(\texttt{next::sp})^*/\texttt{self::transport}])^+,\ \text{Hastings})$$

nSPARQL examples



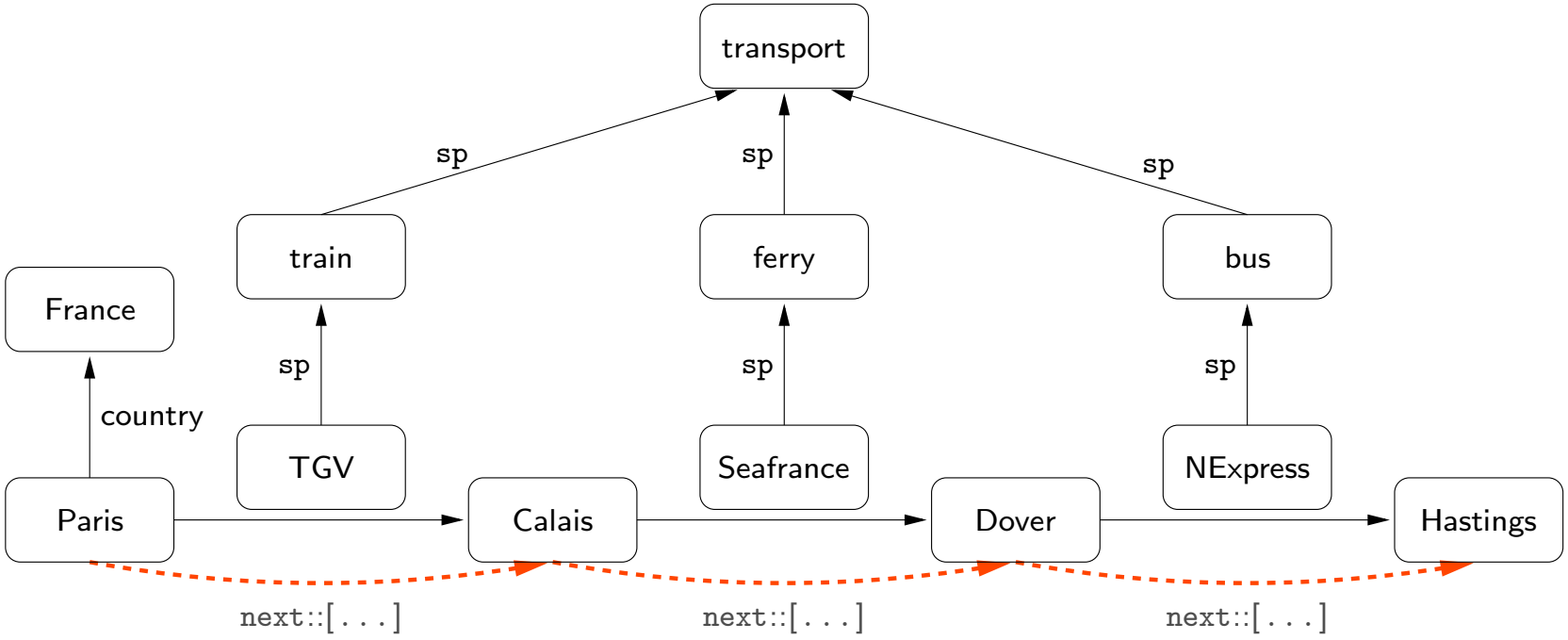
$$(?X,\ \ (\texttt{next::}[(\texttt{next::sp})^*/\texttt{self::transport}])^+,\ \ \text{Hastings})$$

nSPARQL examples



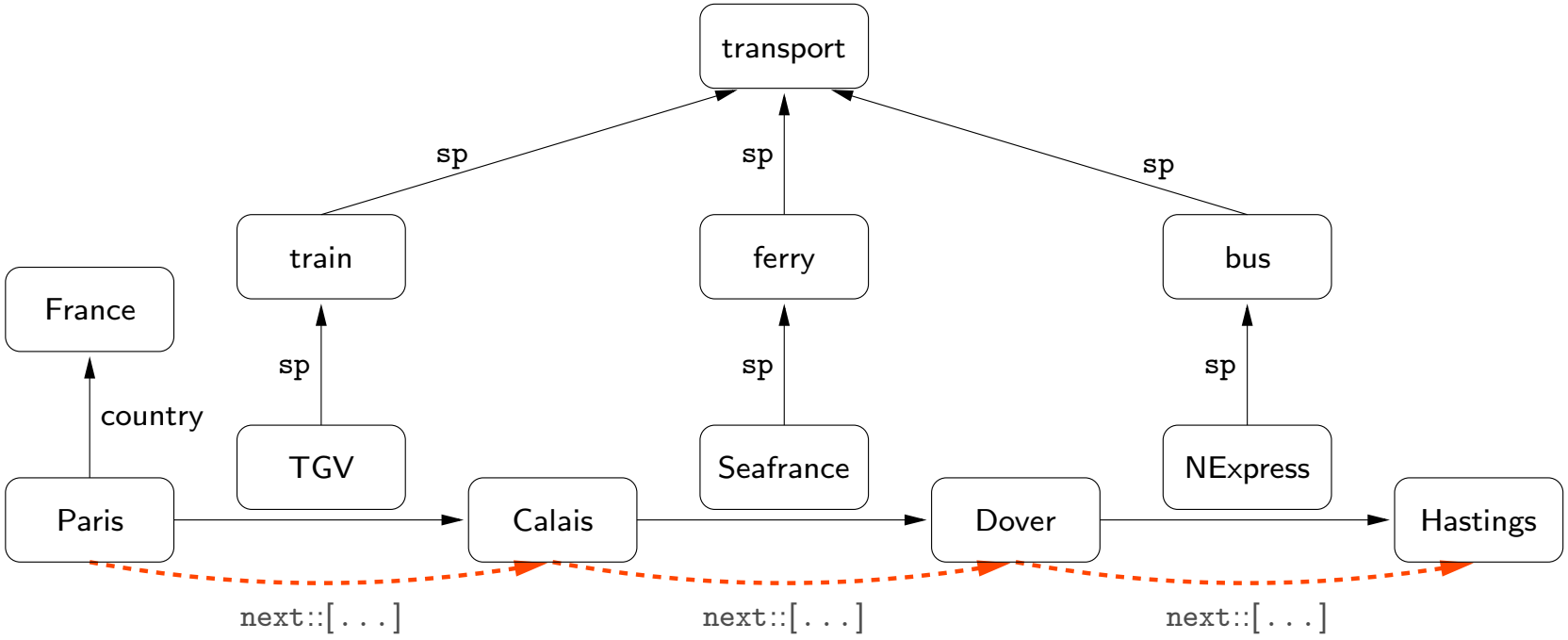
$$(?X,\ \ (\texttt{next::[(next::sp)}^*\texttt{/self::transport]})^+,\ \text{Hastings})$$

nSPARQL examples



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

nSPARQL examples



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

$$\{?X \to \text{Paris}\}$$
$$\{?X \to \text{Calais}\}$$
$$\{?X \to \text{Dover}\}$$

Outline

RDFS evaluation via the closure

Consider the sub-vocabulary of RDFS $\{\texttt{sp}, \texttt{sc}, \texttt{dom}, \texttt{range}, \texttt{type}\}$ 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 $\{\texttt{sp}, \texttt{sc}, \texttt{dom}, \texttt{range}, \texttt{type}\}$ 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* $r = (?X, \mathsf{trans}(a), ?Y)$ *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* $r = (?X, \mathsf{trans}(a), ?Y)$ *such that*

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

Proof idea

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

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

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

Concluding remarks

nested regular expressions:

- natural navigation trough 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`