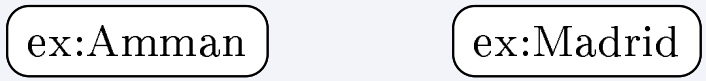
Querying RDF(S) with regular expressions

Faisal Alkhateeb
faisal.alkhateeb@inrialpes.fr

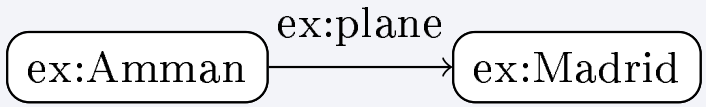
INRIA Rhône-Alpes - LIG

Ph.D Defense – June 2008

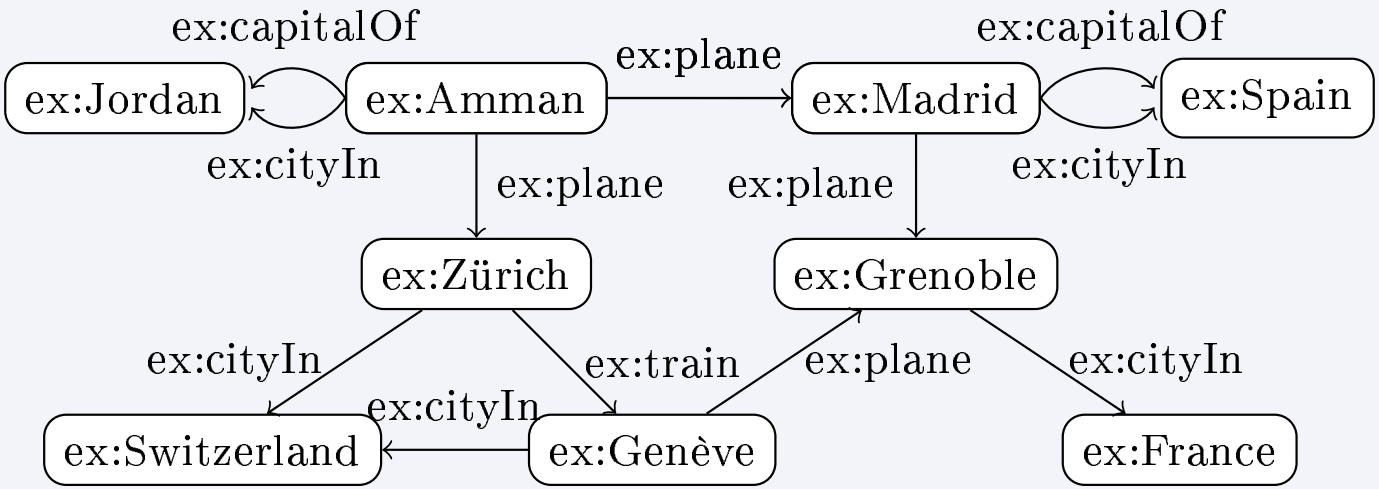
Querying RDF graphs



Querying RDF graphs



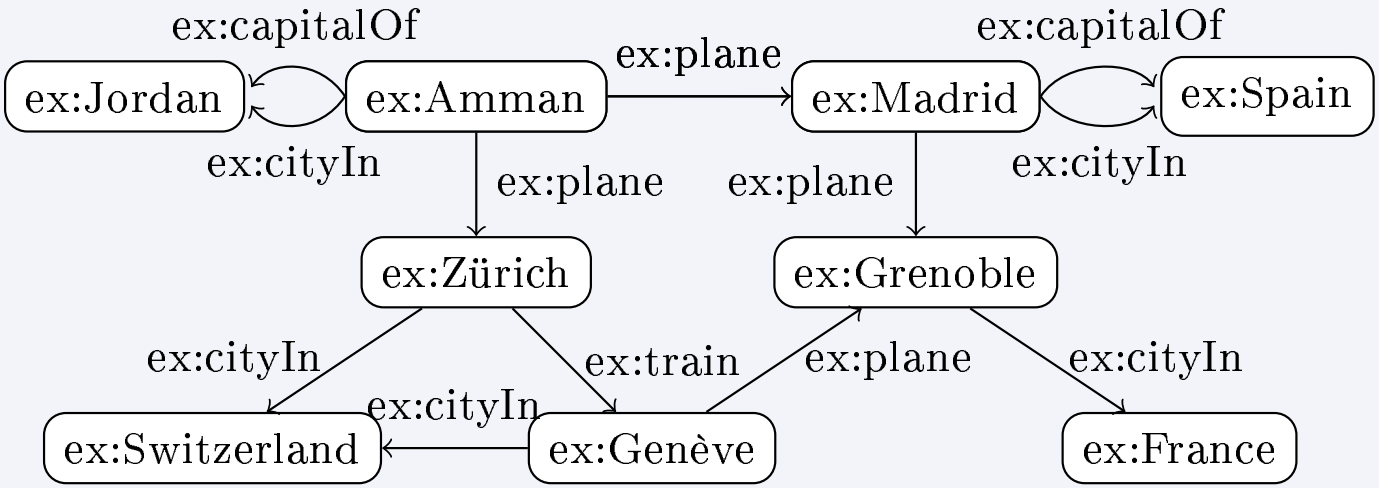
Querying RDF graphs



Querying RDF graphs

SPARQL Query:

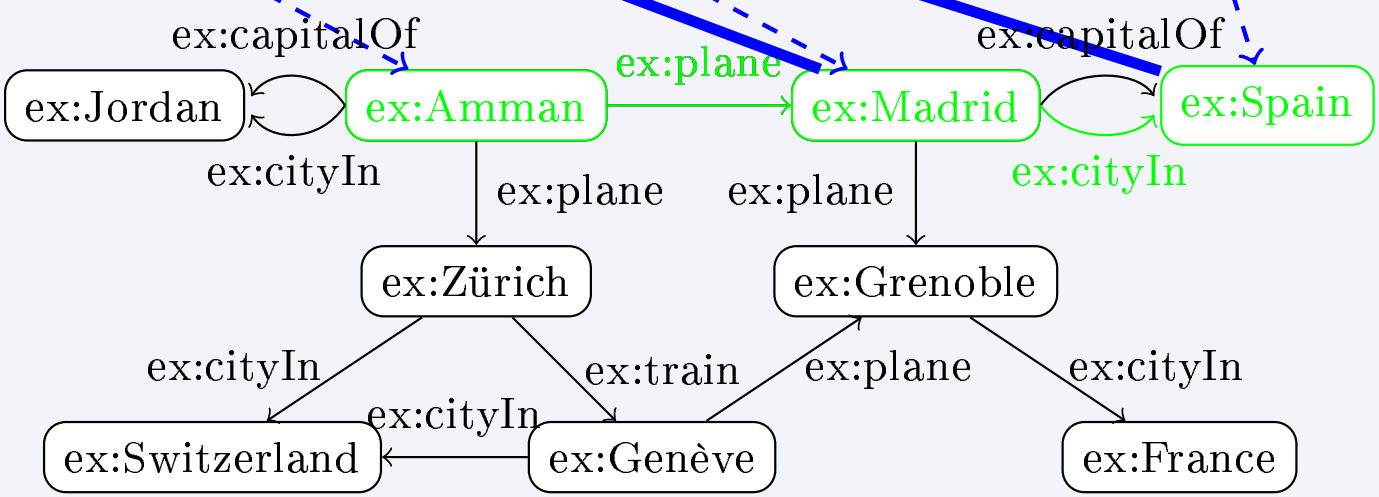
```
SELECT ?City ?Country
FROM <Transport>
WHERE { ex:Amman ex:plane ?City. ?City ex:cityIn ?Country }
```



Querying RDF graphs

SPARQL Query:

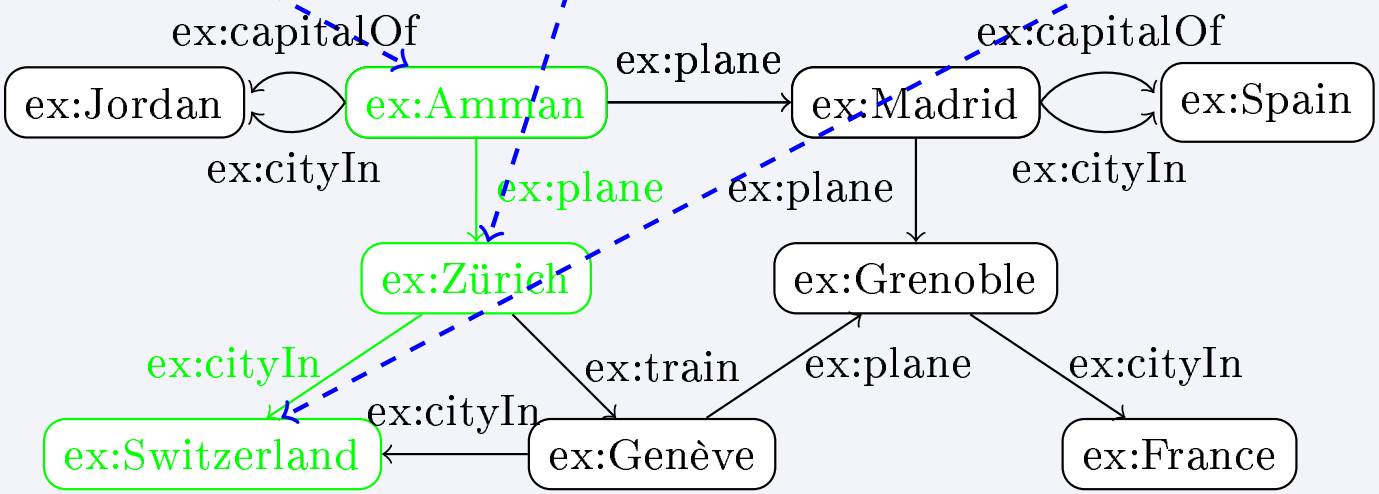
```
SELECT ?City ?Country
FROM <Transport>
WHERE { ex:Amman ex:plane ?City . ?City ex:cityIn ?Country }
```



Querying RDF graphs

SPARQL Query:

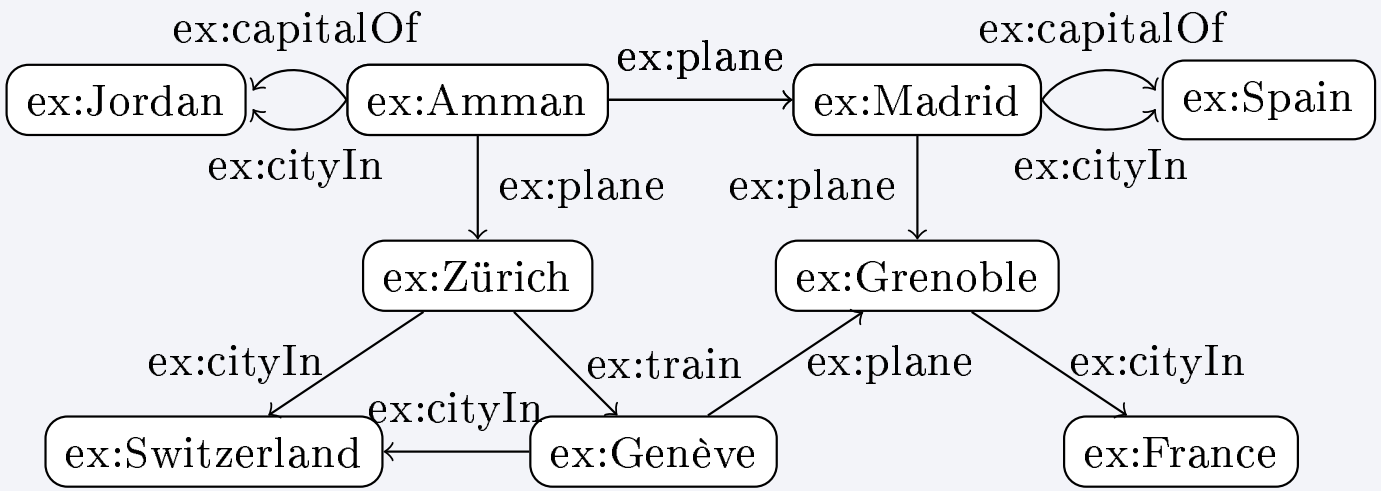
```
SELECT ?City ?Country
FROM <Transport>
WHERE { ex:Amman ex:plane ?City. ?City ex:cityIn ?Country }
```



Querying RDF graphs

PSPARQL Query:

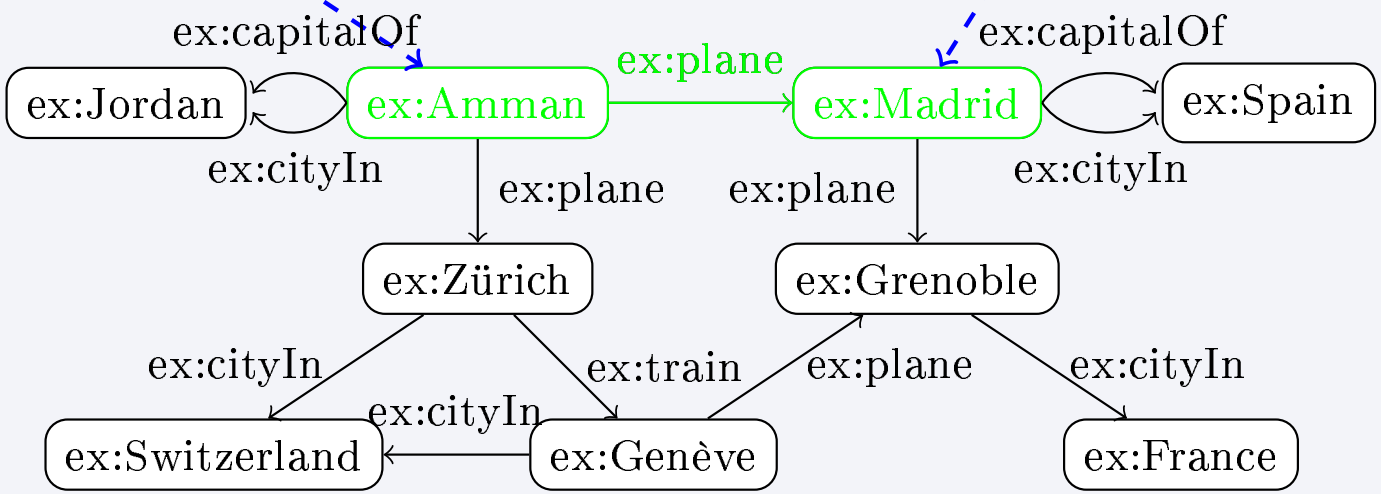
```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman (ex:plane | ex:train)+ ?City . }
```



Querying RDF graphs

PSPARQL Query:

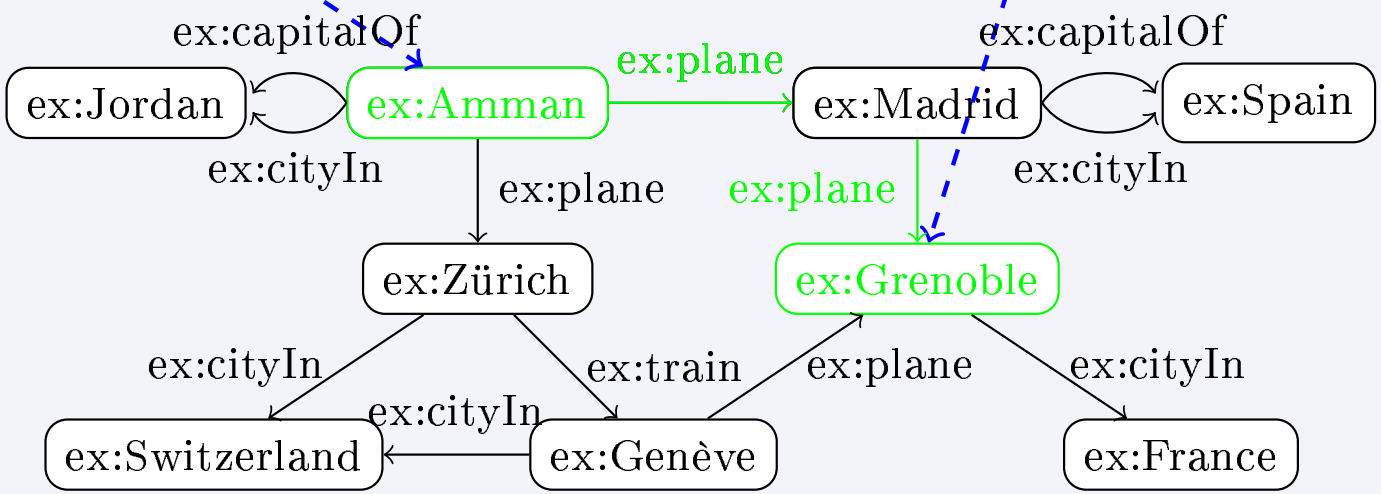
```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman (ex:plane | ex:train)+ ?City . }
```



Querying RDF graphs

PSPARQL Query:

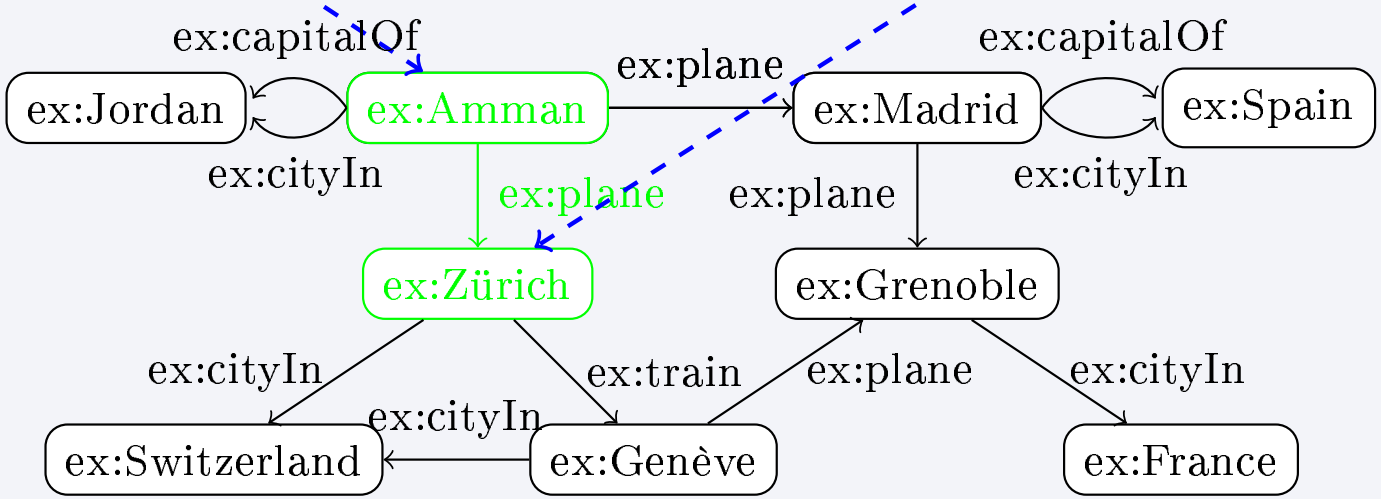
```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman (ex:plane | ex:train)+ ?City . }
```



Querying RDF graphs

PSPARQL Query:

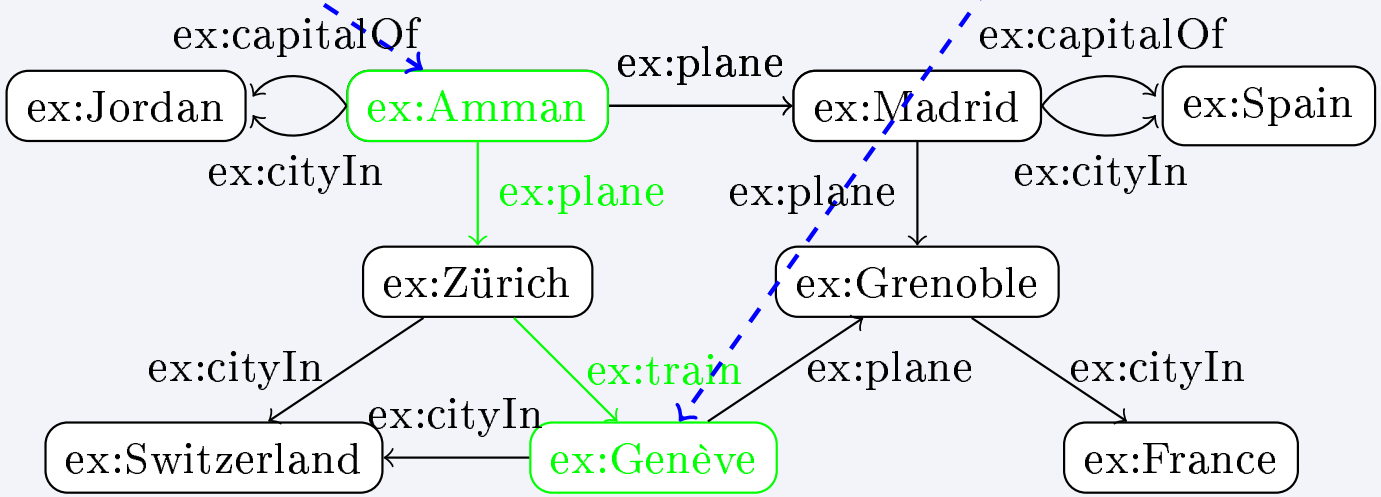
```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman (ex:plane | ex:train)+ ?City . }
```



Querying RDF graphs

PSPARQL Query:

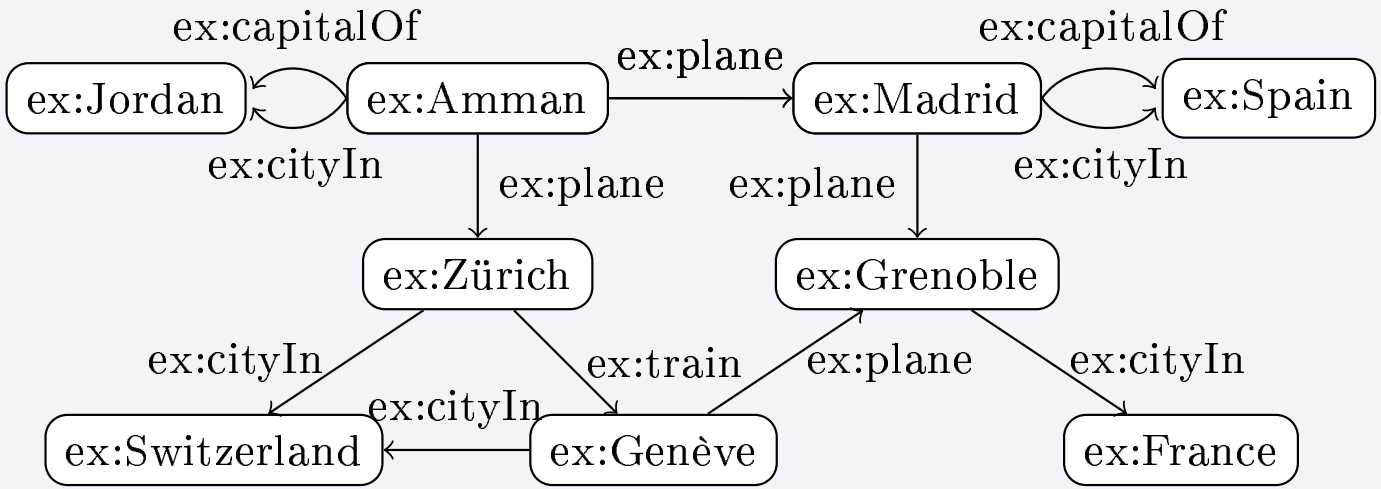
```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman (ex:plane | ex:train)+ ?City . }
```



Querying RDF graphs

CPSPARQL Query:

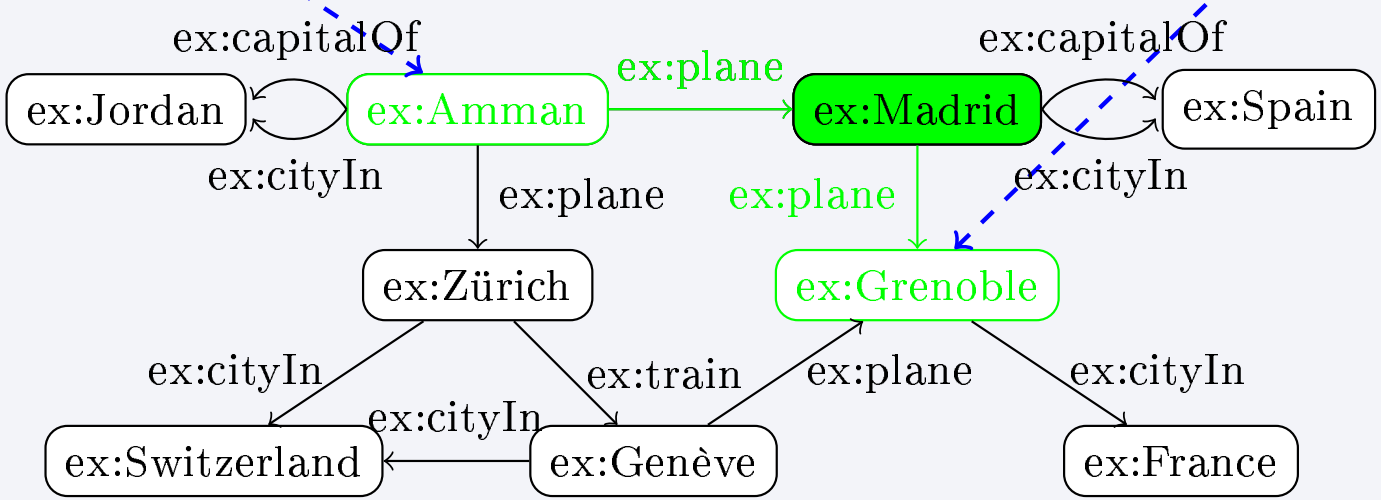
```
SELECT ?City
FROM <Transport>
WHERE { CONSTRAINT const1 ]ALL ?Stop[:
                        { ?Stop ex:capitalOf ?Country }
        ex:Amman (ex:plane | ex:train)+%const1% ?City .}
```



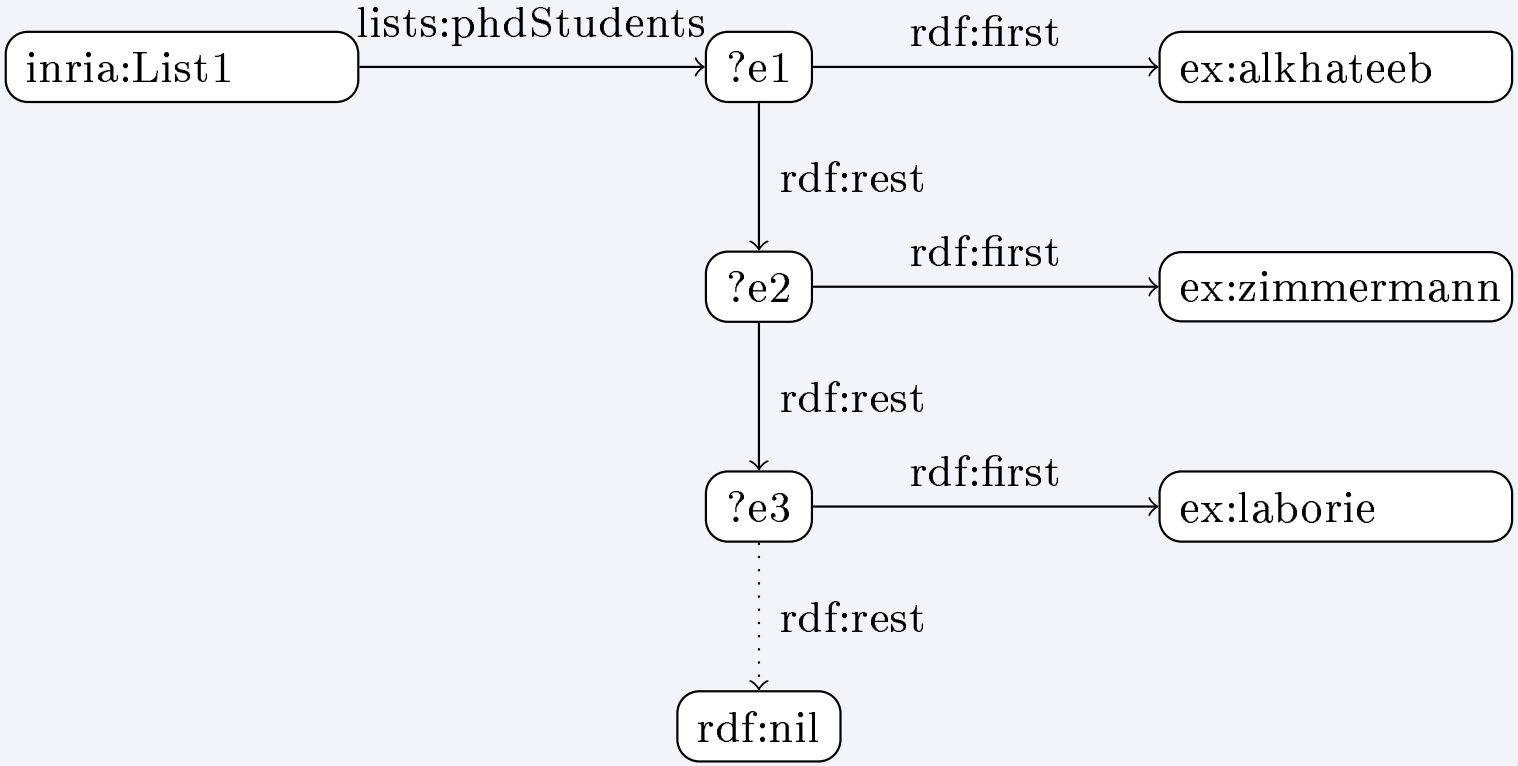
Querying RDF graphs

CPSPARQL Query:

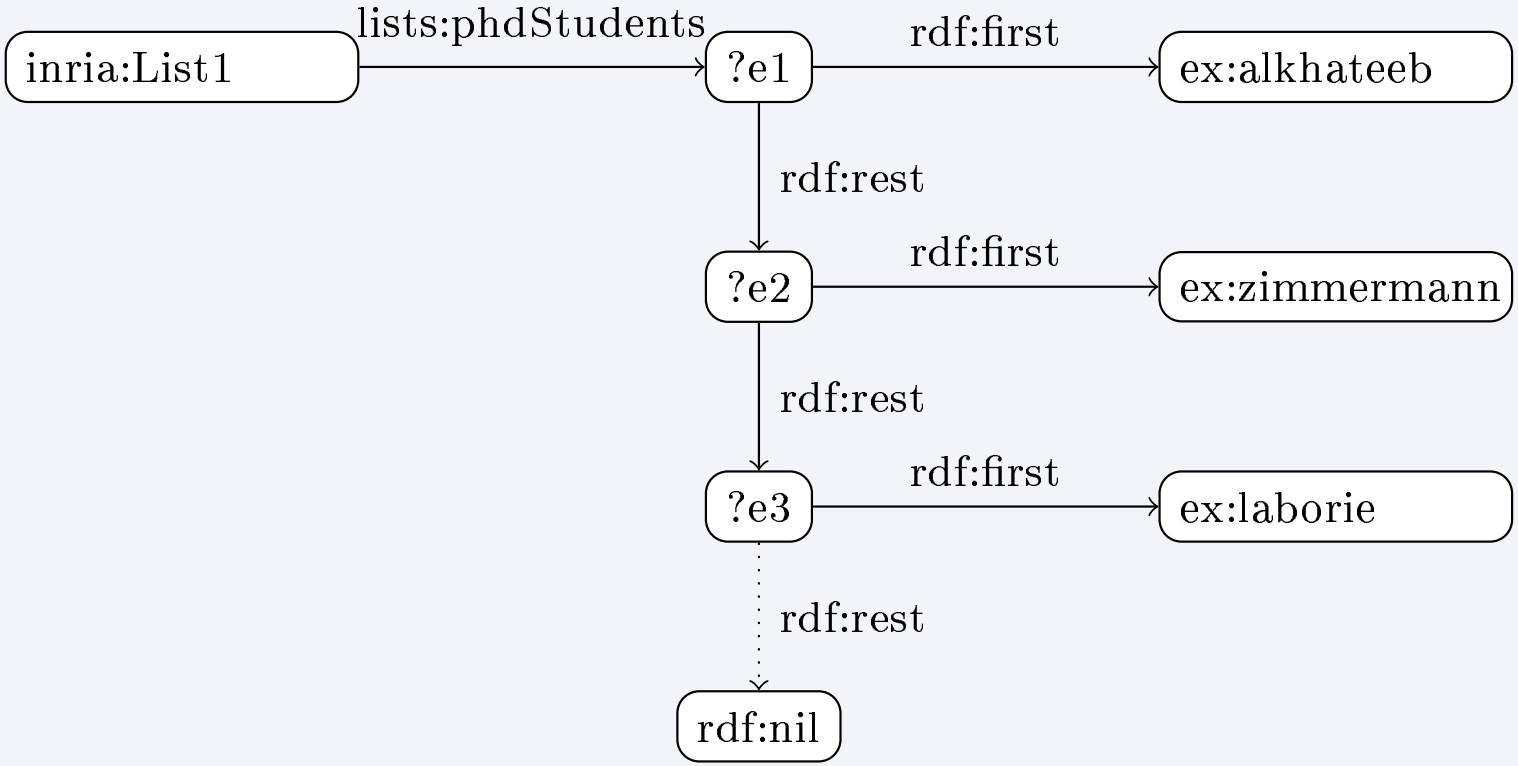
```
SELECT ?City
FROM <Transport>
WHERE { CONSTRAINT const1 ]ALL ?Stop[:
                        { ?Stop ex:capitalOf ?Country }
        ex:Amman (ex:plane | ex:train)+%const1% ?City .}
```



Accessing elements of a collection (a list)



Accessing elements of a collection (a list)




```
SELECT ?Element
WHERE { lists:phdStudents rdfs:range rdf:List .
        inria:List1 lists:phdStudents.rdf:rest*.rdf:first
?Element .
      }
```

Main contributions

- We define two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions

Main contributions

- We define two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We define inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.

Main contributions

- We define two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We define inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.
- We prove the soundness and completeness of (C)PRDF homomorphism ((C)PRDF into RDF) w.r.t $\models_{(C)PRDF}$.

Main contributions

- We define two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We define inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.
- We prove the soundness and completeness of (C)PRDF homomorphism ((C)PRDF into RDF) w.r.t $\models_{(C)PRDF}$.
- We propose:
  - PSPARQL = SPARQL + PRDF
  - CPSPARQL = SPARQL + CPRDF

Main contributions

- We define two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We define inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.
- We prove the soundness and completeness of (C)PRDF homomorphism ((C)PRDF into RDF) w.r.t $\models_{(C)PRDF}$.
- We propose:
  - PSPARQL = SPARQL + PRDF
  - CPSPARQL = SPARQL + CPRDF
- We show that (C)PSPARQL extensions do not increase the overall complexity of SPARQL.

Main contributions

- We define two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We define inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.
- We prove the soundness and completeness of (C)PRDF homomorphism ((C)PRDF into RDF) w.r.t $\models_{(C)PRDF}$.
- We propose:
  - PSPARQL = SPARQL + PRDF
  - CPSPARQL = SPARQL + CPRDF
- We show that (C)PSPARQL extensions do not increase the overall complexity of SPARQL.
- We provide a rewriting method for evaluating SPARQL or (C)PSPARQL queries over RDF(S).

Outline

Outline

PRDF syntax

- RDF syntax
- Regular expressions syntax
- Mixed 2 syntax

RDF syntax

- Terminology $\mathcal{T}$: $\mathcal{U}$: Urirefs, $\mathcal{L}$: Literals, $\mathcal{B}$: Blanks (Variables)
  $\mathcal{V} = \mathcal{U} \cup \mathcal{L}$

RDF syntax

- Terminology $\mathcal{T}$: $\mathcal{U}$: Urirefs, $\mathcal{L}$: Literals, $\mathcal{B}$: Blanks (Variables)
  $\mathcal{V} = \mathcal{U} \cup \mathcal{L}$
- RDF Triple: $\langle subject, predicate, object \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times \mathcal{T}$

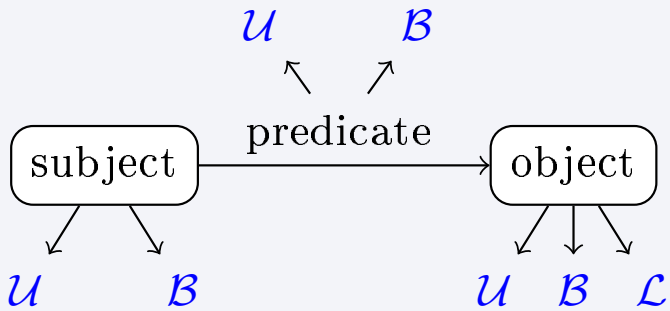
RDF syntax

- Terminology $\mathcal{T}$: $\mathcal{U}$: Urirefs, $\mathcal{L}$: Literals, $\mathcal{B}$: Blanks (Variables)
  $\mathcal{V} = \mathcal{U} \cup \mathcal{L}$
- RDF Triple: $\langle subject, predicate, object \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times \mathcal{T}$
- GRDF Triple: $\langle subject, predicate, object \rangle \in$
  $(\mathcal{U} \cup \mathcal{B}) \times (\mathcal{U} \cup \mathcal{B}) \times \mathcal{T}$

subject predicate object

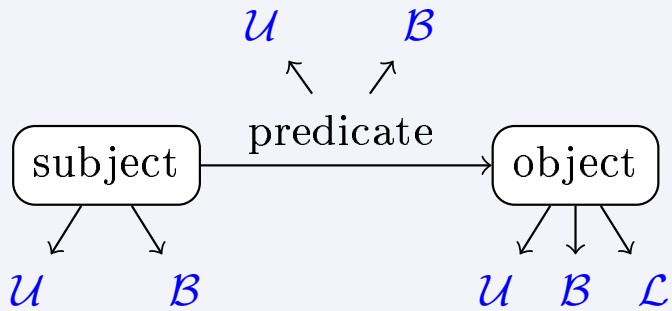
RDF syntax

- Terminology $\mathcal{T}$: $\mathcal{U}$: Urirefs, $\mathcal{L}$: Literals, $\mathcal{B}$: Blanks (Variables)
  $\mathcal{V} = \mathcal{U} \cup \mathcal{L}$
- RDF Triple: $\langle subject, predicate, object \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times \mathcal{T}$
- GRDF Triple: $\langle subject, predicate, object \rangle \in$
  $(\mathcal{U} \cup \mathcal{B}) \times (\mathcal{U} \cup \mathcal{B}) \times \mathcal{T}$



RDF syntax

- Terminology $\mathcal{T}$: $\mathcal{U}$: Urirefs, $\mathcal{L}$: Literals, $\mathcal{B}$: Blanks (Variables)
  $\mathcal{V} = \mathcal{U} \cup \mathcal{L}$
- RDF Triple: $\langle subject, predicate, object \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{U} \times \mathcal{T}$
- GRDF Triple: $\langle subject, predicate, object \rangle \in$
  $(\mathcal{U} \cup \mathcal{B}) \times (\mathcal{U} \cup \mathcal{B}) \times \mathcal{T}$



- (G)RDF graph: a set of (G)RDF triples

Regular Expressions

The set $\mathcal{RE}(\mathcal{U}, B)$ of GRDF regular expressions is inductively defined by:

- $\forall a \in \mathcal{U}$, then a and $!a \in \mathcal{RE}(\mathcal{U}, B)$;

Regular Expressions

The set $\mathcal{RE}(\mathcal{U}, B)$ of GRDF regular expressions is inductively defined by:

- $\forall a \in \mathcal{U}$, then a and $!a \in \mathcal{RE}(\mathcal{U}, B)$;
- $\forall a \in \mathcal{B}$, then $a \in \mathcal{RE}(\mathcal{U}, B)$;
- $\# \in \mathcal{RE}(\mathcal{U}, B)$;

Regular Expressions

The set $\mathcal{RE}(\mathcal{U}, B)$ of GRDF regular expressions is inductively defined by:

- $\forall a \in \mathcal{U}$, then a and $!a \in \mathcal{RE}(\mathcal{U}, B)$;
- $\forall a \in \mathcal{B}$, then $a \in \mathcal{RE}(\mathcal{U}, B)$;
- $\# \in \mathcal{RE}(\mathcal{U}, B)$;
- If $A \in \mathcal{R}(\mathcal{U}, B)$ and $B \in \mathcal{RE}(\mathcal{U}, B)$ then:
  - $A|B$, $A \cdot B$ and $A^* \in \mathcal{RE}(\mathcal{U}, B)$ $(R^+ = R \cdot R^*)$.

Regular Expressions

The set $\mathcal{RE}(\mathcal{U}, B)$ of GRDF regular expressions is inductively defined by:

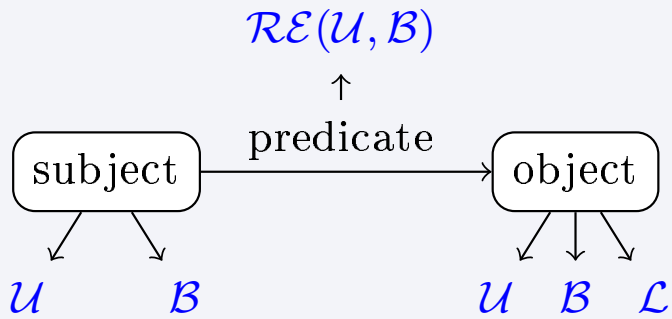
- $\forall a \in \mathcal{U}$, then a and $!a \in \mathcal{RE}(\mathcal{U}, B)$;
- $\forall a \in \mathcal{B}$, then $a \in \mathcal{RE}(\mathcal{U}, B)$;
- $\# \in \mathcal{RE}(\mathcal{U}, B)$;
- If $A \in \mathcal{R}(\mathcal{U}, B)$ and $B \in \mathcal{RE}(\mathcal{U}, B)$ then:
  - $A|B$, $A \cdot B$ and $A^* \in \mathcal{RE}(\mathcal{U}, B)$ $(R^+ = R \cdot R^*)$.

Example (GRDF regular expressions)

$(\texttt{ex:plane} \mid \texttt{ex:train})^+ \cdot \texttt{?b5}$

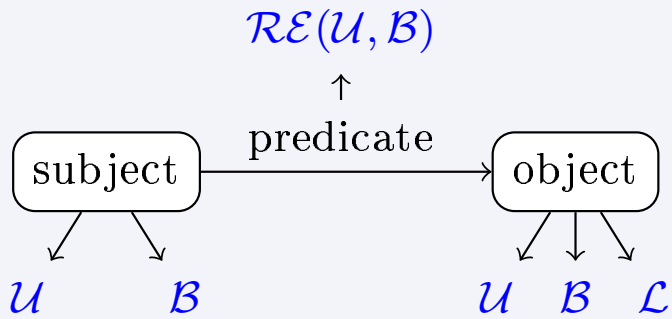
PRDF Syntax

- PRDF Triple: $\langle subject, predicate, object \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{RE}(\mathcal{U}, \mathcal{B}) \times \mathcal{T}$



PRDF Syntax

- PRDF Triple: $\langle subject, predicate, object \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{RE}(\mathcal{U}, \mathcal{B}) \times \mathcal{T}$



- PRDF graph: a set of PRDF triples

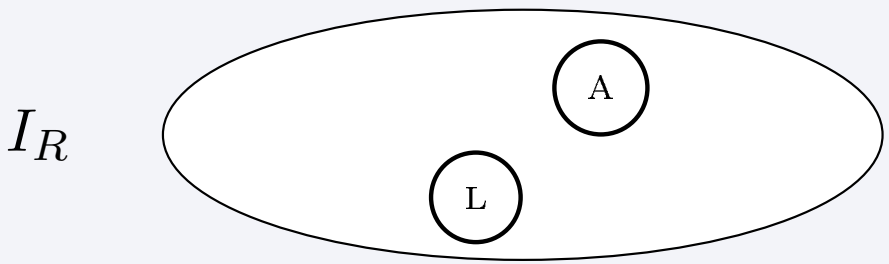
PRDF semantics

- RDF semantics
- Regular expressions semantics
- Mixed 2 semantics

GRDF Semantics

Definition (Interpretation)

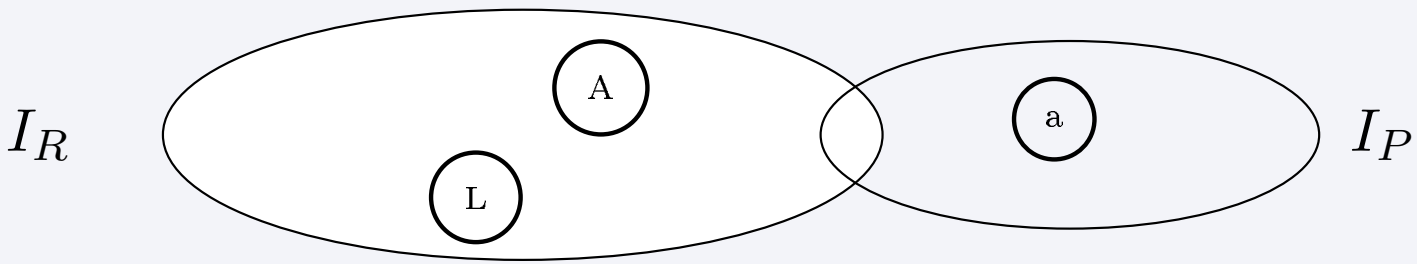
An interpretation of V is a 4-tuple $I = \langle I_R, I_P, \iota, I_{EXT} \rangle$, where:



GRDF Semantics

Definition (Interpretation)

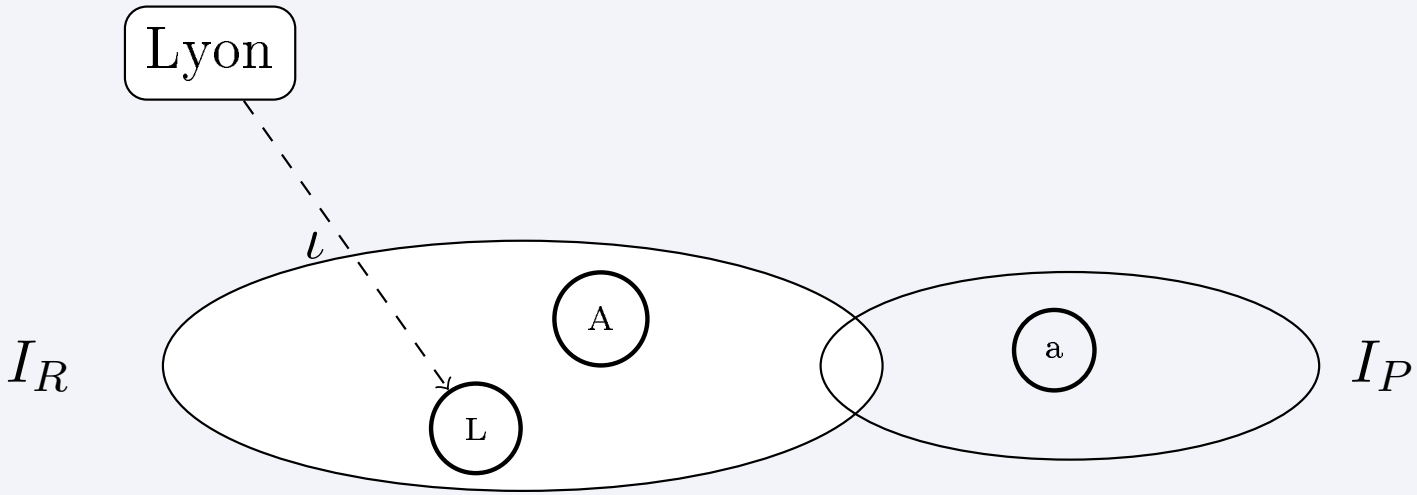
An interpretation of V is a 4-tuple $I = \langle I_R, I_P, \iota, I_{EXT} \rangle$, where:



GRDF Semantics

Definition (Interpretation)

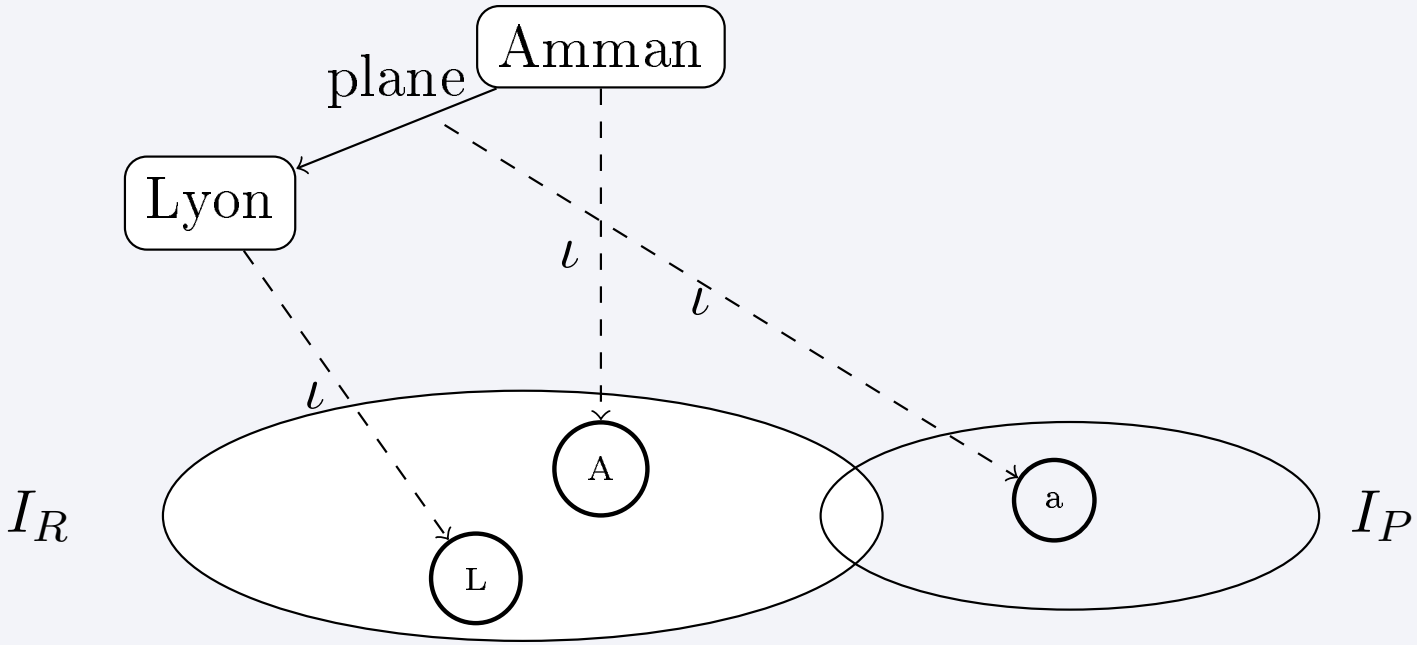
An interpretation of V is a 4-tuple $I = \langle I_R, I_P, \iota, I_{EXT} \rangle$, where:



GRDF Semantics

Definition (Interpretation)

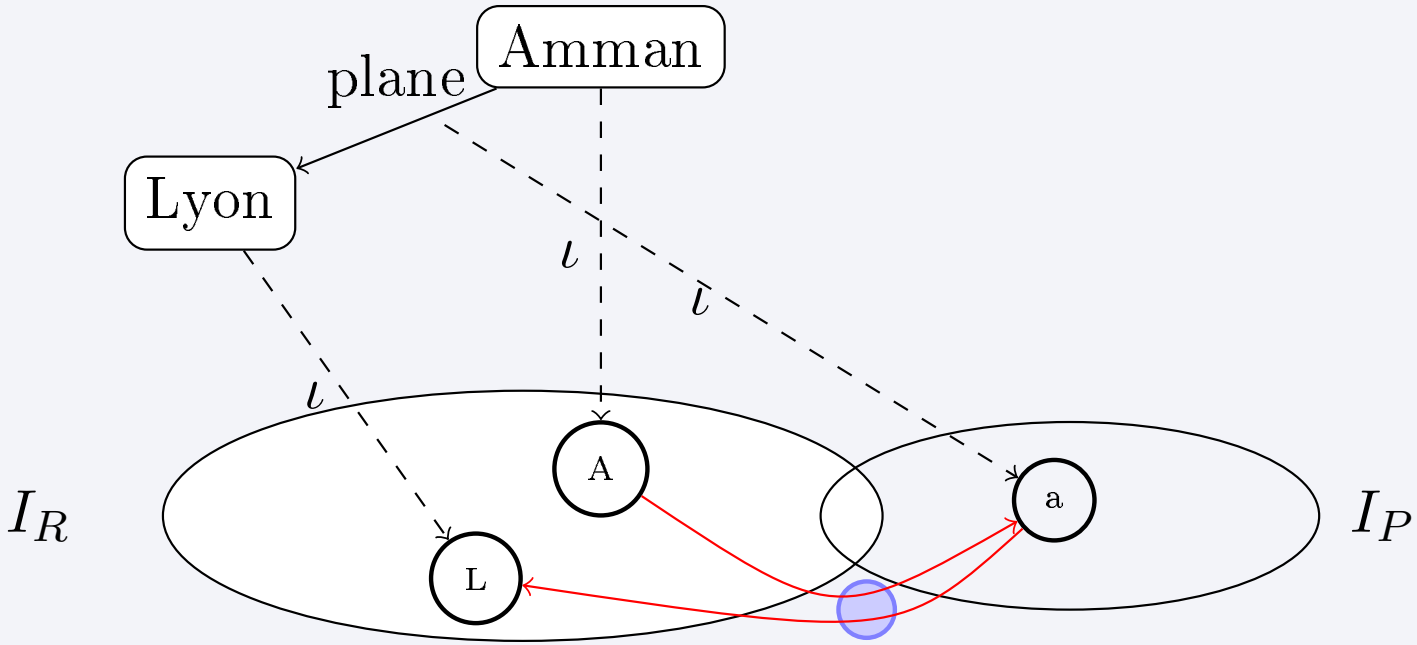
An interpretation of V is a 4-tuple $I = \langle I_R, I_P, \iota, I_{EXT} \rangle$, where:



GRDF Semantics

Definition (Interpretation)

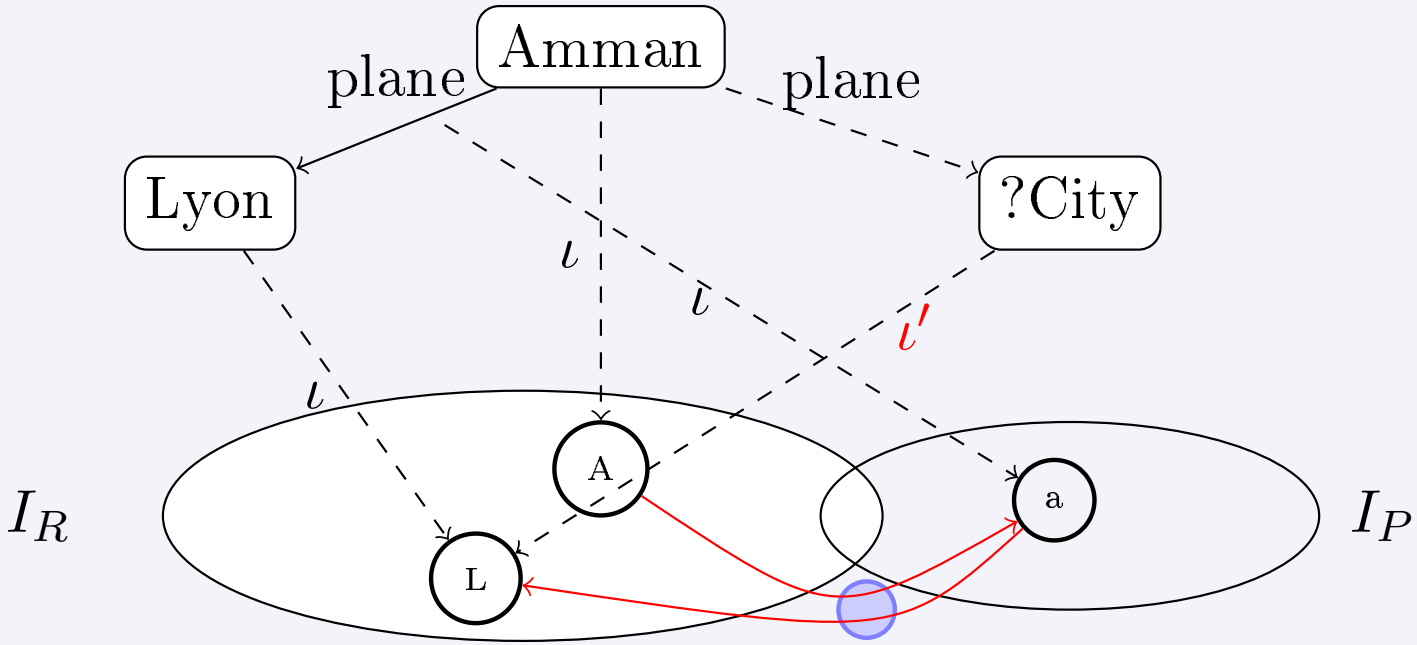
An interpretation of V is a 4-tuple $I = \langle I_R, I_P, \iota, I_{EXT} \rangle$, where:



GRDF Semantics

Definition (Interpretation)

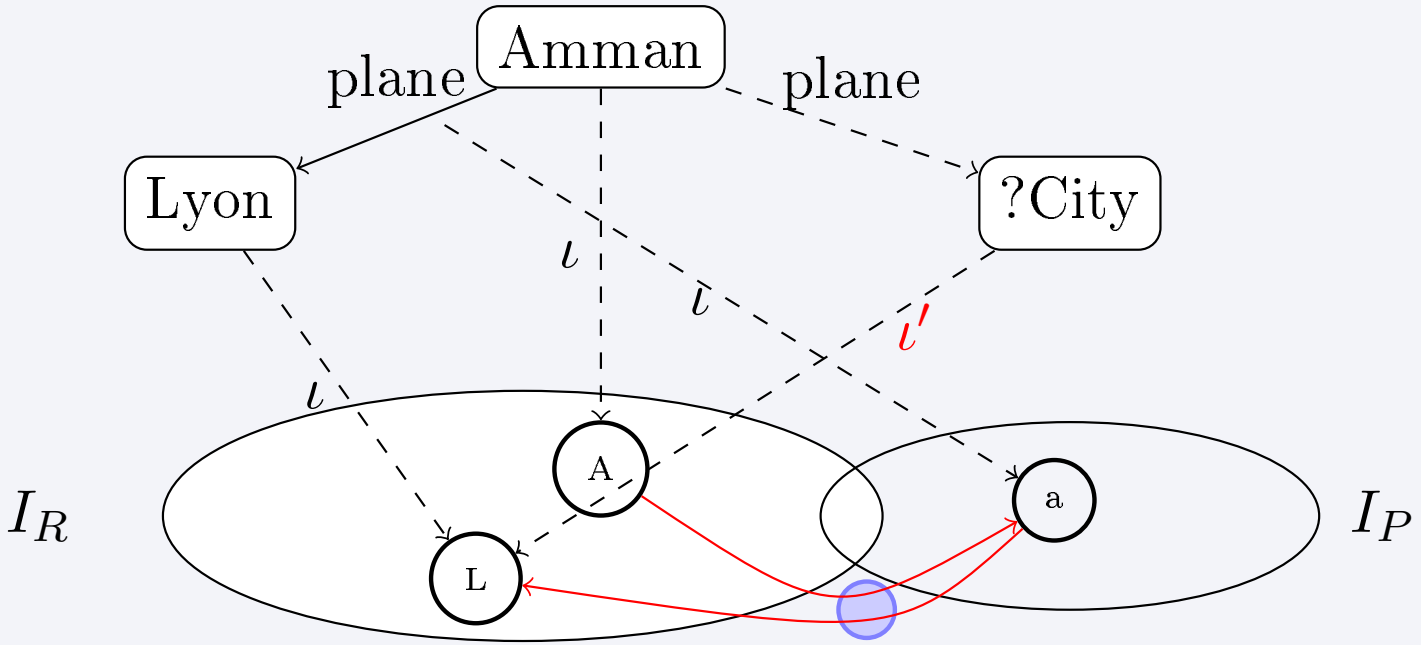
An interpretation of V is a 4-tuple $I = \langle I_R, I_P, \iota, I_{EXT} \rangle$, where:



GRDF Semantics

Definition (Interpretation)

An interpretation of V is a 4-tuple $I = \langle I_R, I_P, \iota, I_{EXT} \rangle$, where:



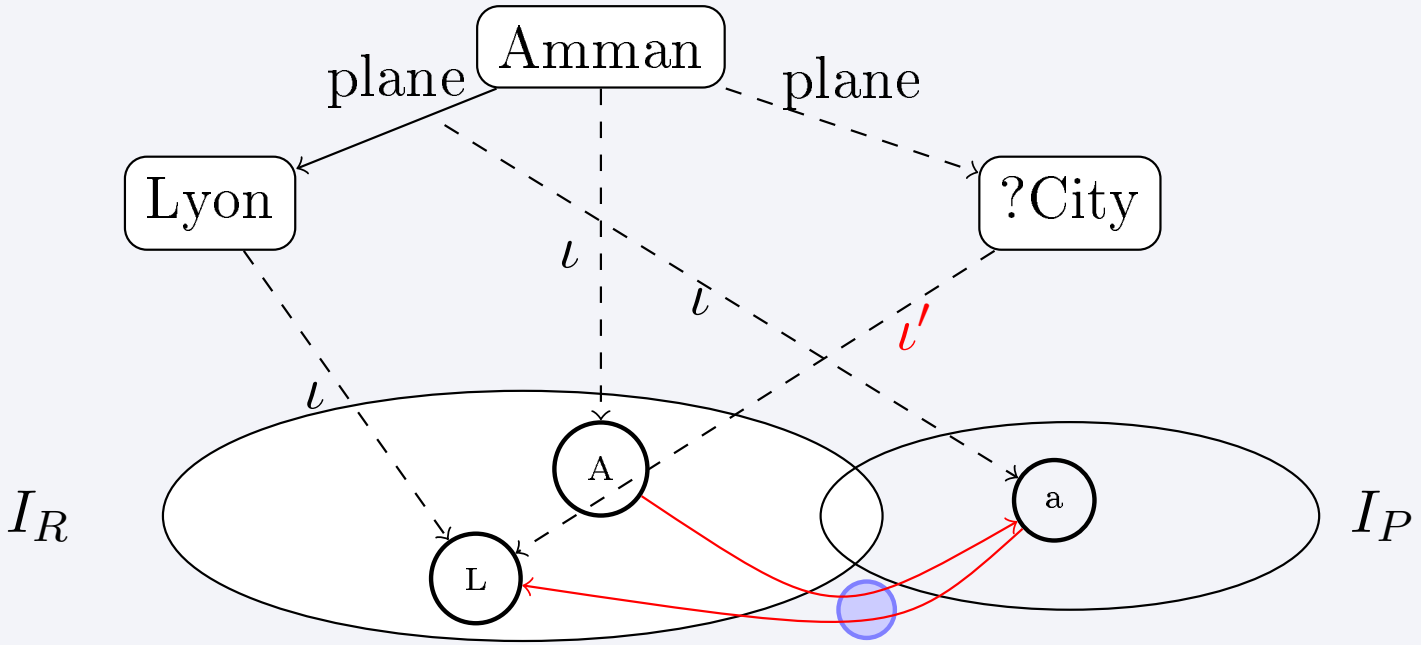
Definition (Models, consequence)

I is a model of a GRDF graph G iff $\exists$ an extension ι' of ι to $\mathcal{B}(G)$ such that $\forall \langle s, p, o \rangle \in G$, $\langle \iota'(s), \iota'(o) \rangle \in I_{EXT}(\iota'(p))$.

GRDF Semantics

Definition (Interpretation)

An interpretation of V is a 4-tuple $I = \langle I_R, I_P, \iota, I_{EXT} \rangle$, where:

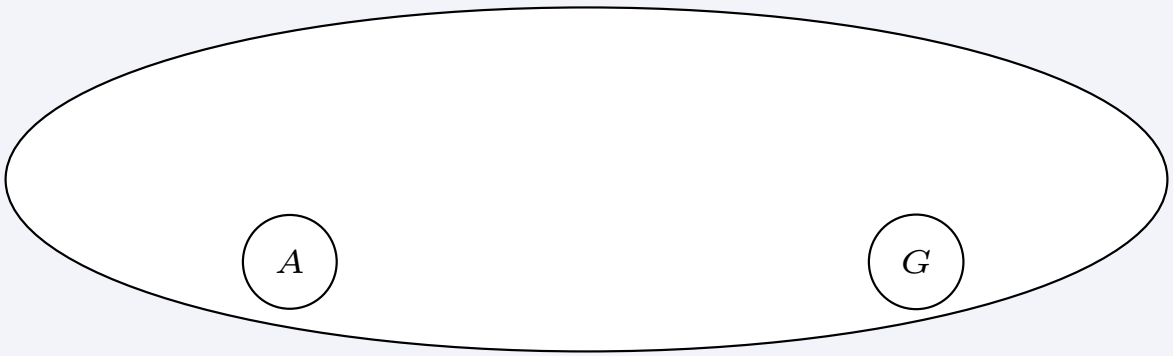


Definition (Models, consequence)

A graph H is a semantic consequence of a graph G, denoted $G \models H$, iff every model of G is a model of H.

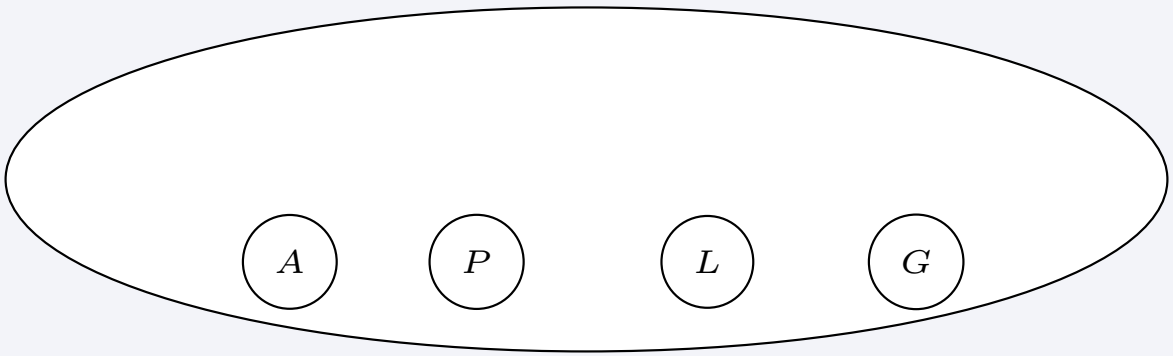
RDF-like semantics for regular expressions

- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})$ in ι'?



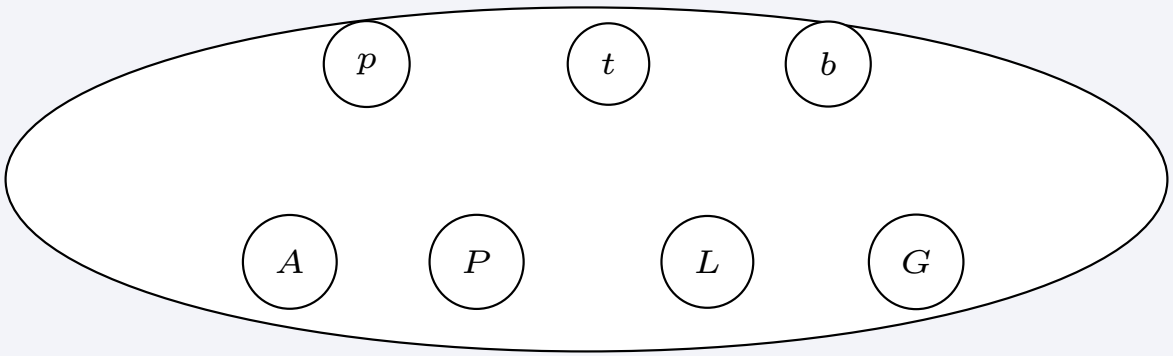
RDF-like semantics for regular expressions

- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})$ in ι'?



RDF-like semantics for regular expressions

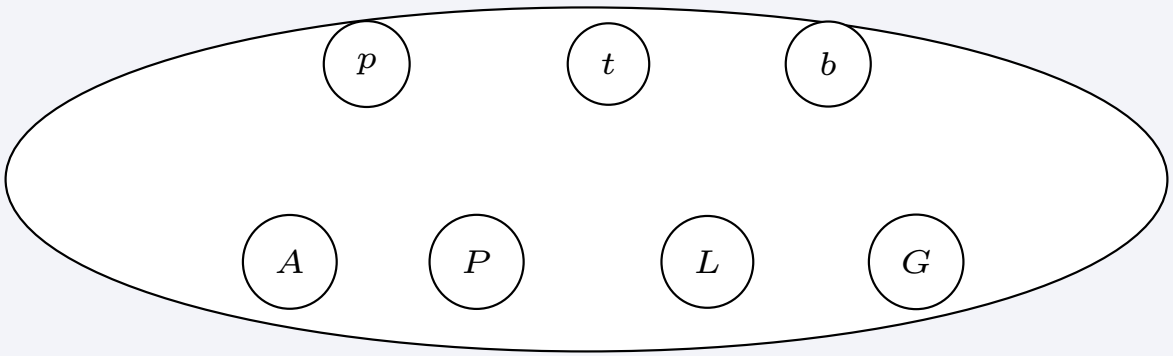
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})$ in ι'?



RDF-like semantics for regular expressions

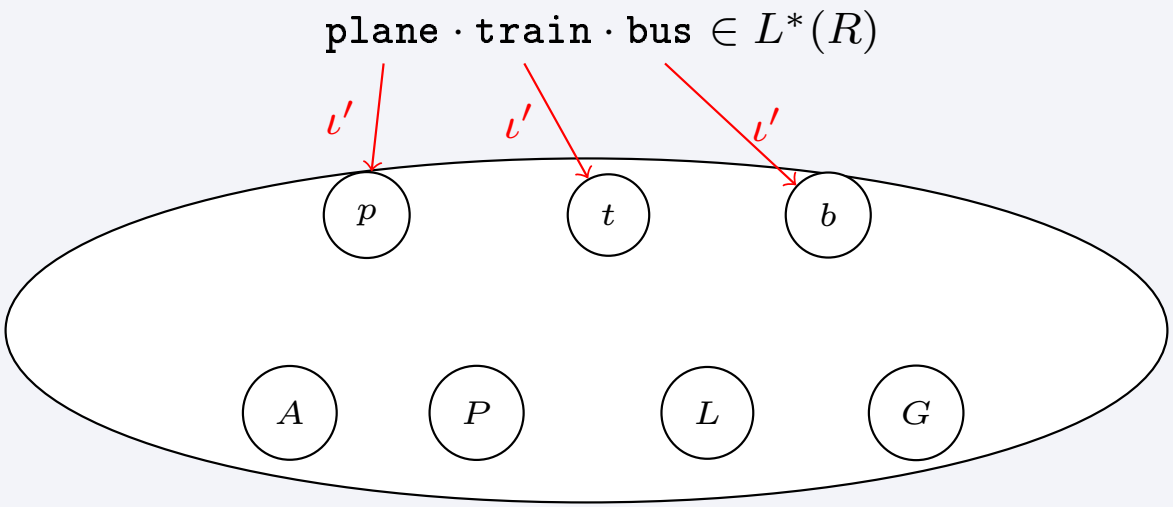
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})$ in ι'?

$$\texttt{plane} \cdot \texttt{train} \cdot \texttt{bus} \in L^*(R)$$



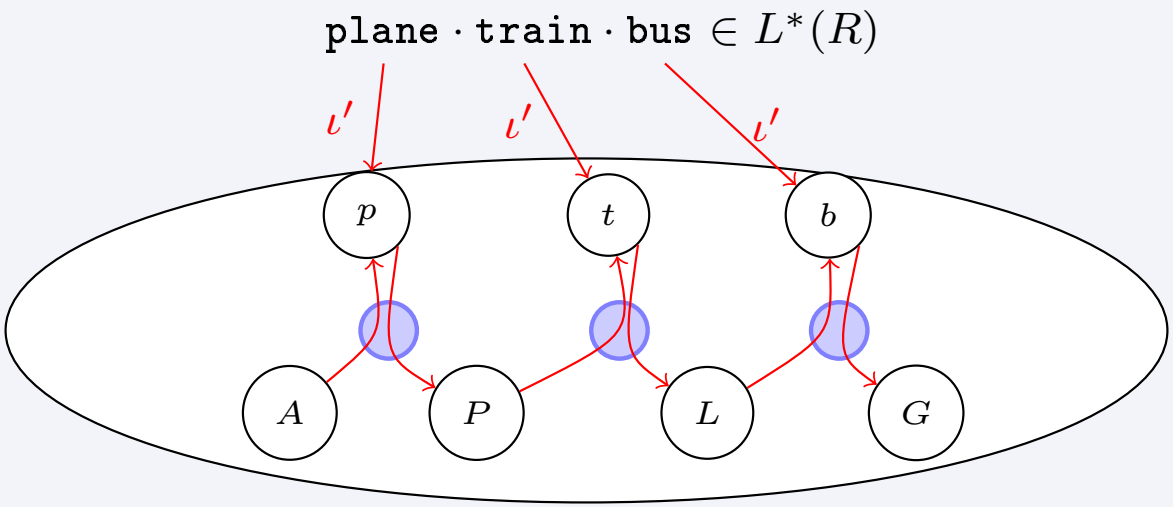
RDF-like semantics for regular expressions

- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})$ in ι'?



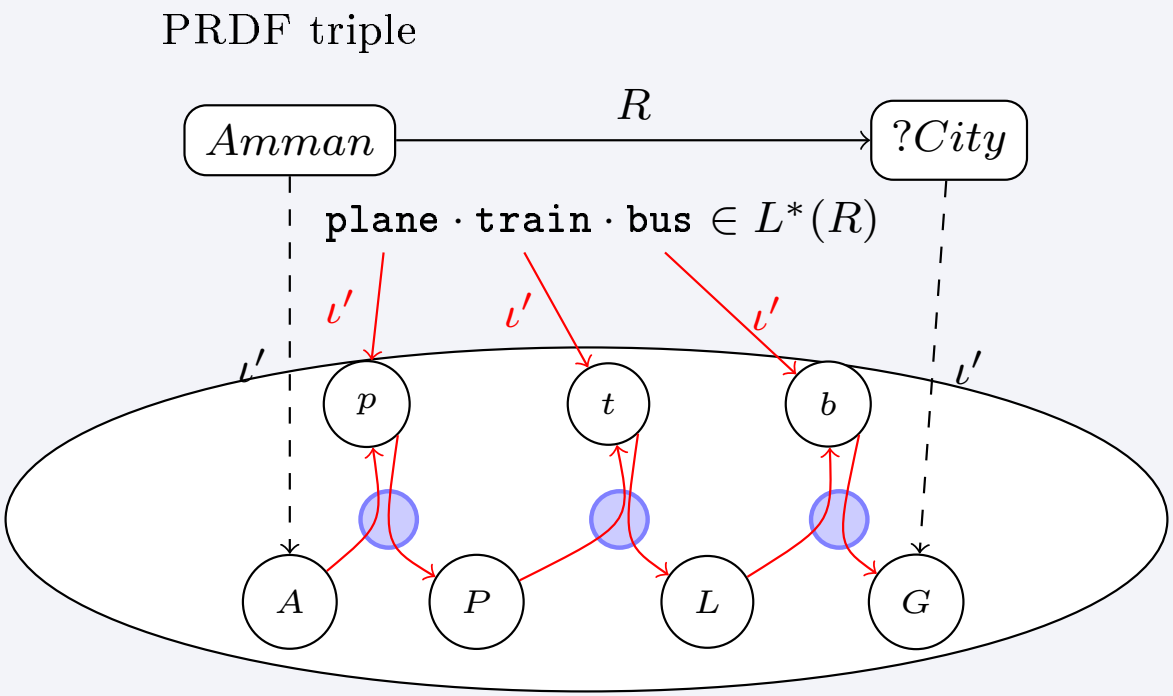
RDF-like semantics for regular expressions

- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})$ in ι'?



RDF-like semantics for regular expressions

- Does $\langle A, G\rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})$ in ι'?



Definition

I is a model of a PRDF G iff $\exists$ an extension ι' of ι to $\mathcal{B}(G)$ such that $\forall\langle s, R, o\rangle \in G$, $\langle \iota'(s), \iota'(o)\rangle$ supports R in ι'.

Inference Mechanism: PRDF Homomorphism

- A PRDF graph H, and a GRDF graph G. Does $G \models_{PRDF} H$?

Inference Mechanism: PRDF Homomorphism

- A PRDF graph H, and a GRDF graph G. Does $G \models_{PRDF} H$?

  - RDF homomorphism
  - Path satisfiability
  - Mixed 2 mechanisms

GRDF Homomorphism

- A map μ is a function from $\mathcal{T} \rightarrow \mathcal{T}$ such that $\forall x \in \mathcal{V}$, $\mu(x) = x$.

GRDF Homomorphism

- A map μ is a function from $\mathcal{T} \rightarrow \mathcal{T}$ such that $\forall x \in \mathcal{V}$, $\mu(x) = x$.
- A GRDF homomorphism from a GRDF graph H into a GRDF graph G is a **map** π such that: $\langle s, p, o \rangle \in H \Longrightarrow \langle \pi(s), \pi(p), \pi(o) \rangle \in G$

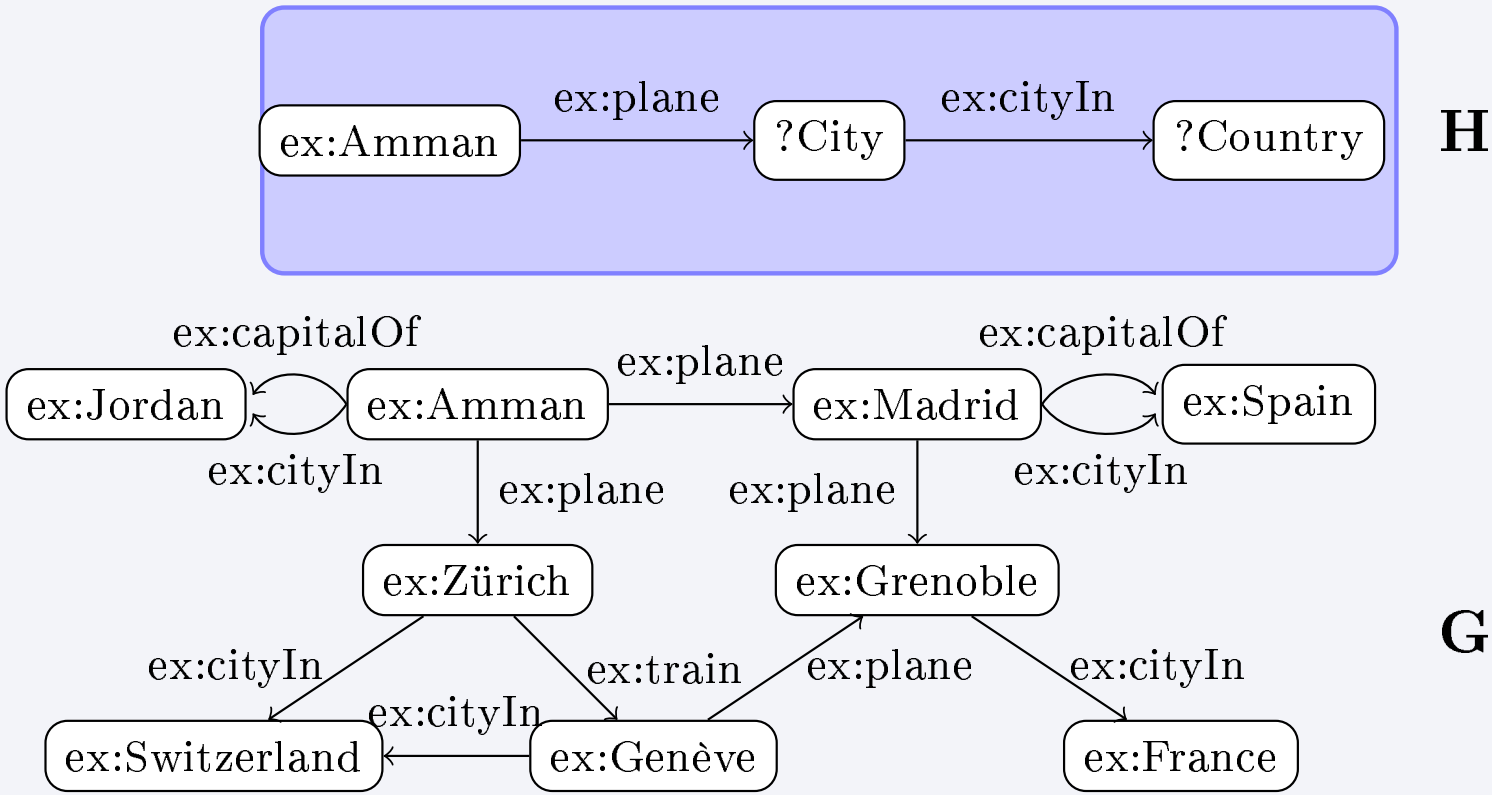
GRDF Homomorphism

- A map μ is a function from $\mathcal{T} \rightarrow \mathcal{T}$ such that $\forall x \in \mathcal{V}$, $\mu(x) = x$.
- A GRDF homomorphism from a GRDF graph H into a GRDF graph G is a **map** π such that: $\langle s, p, o \rangle \in H \Longrightarrow \langle \pi(s), \pi(p), \pi(o) \rangle \in G$

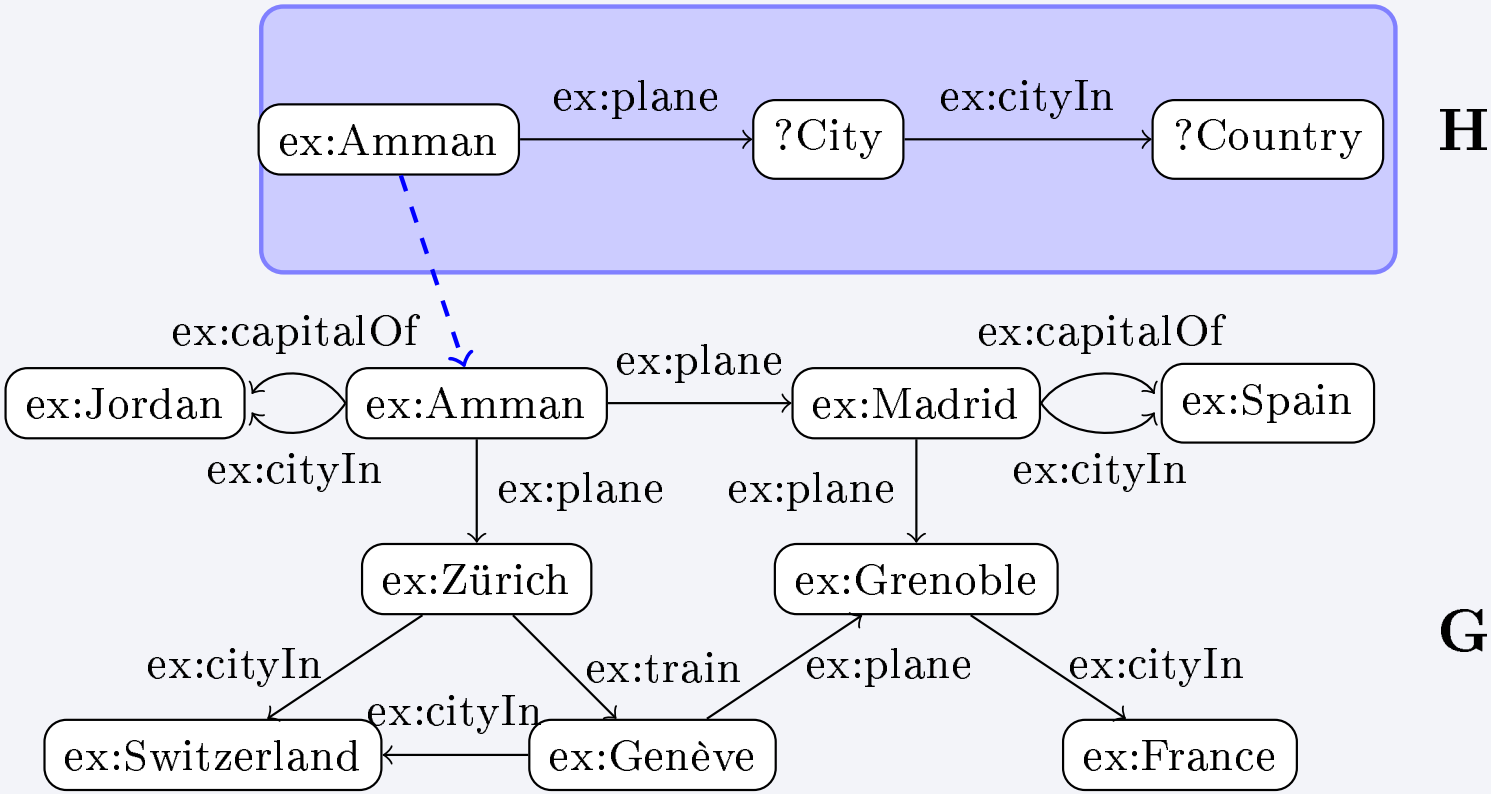
Theorem

Let G and H be two GRDF graphs. Then $G \models_{GRDF} H$ iff there exists a GRDF homomorphism from H into G.

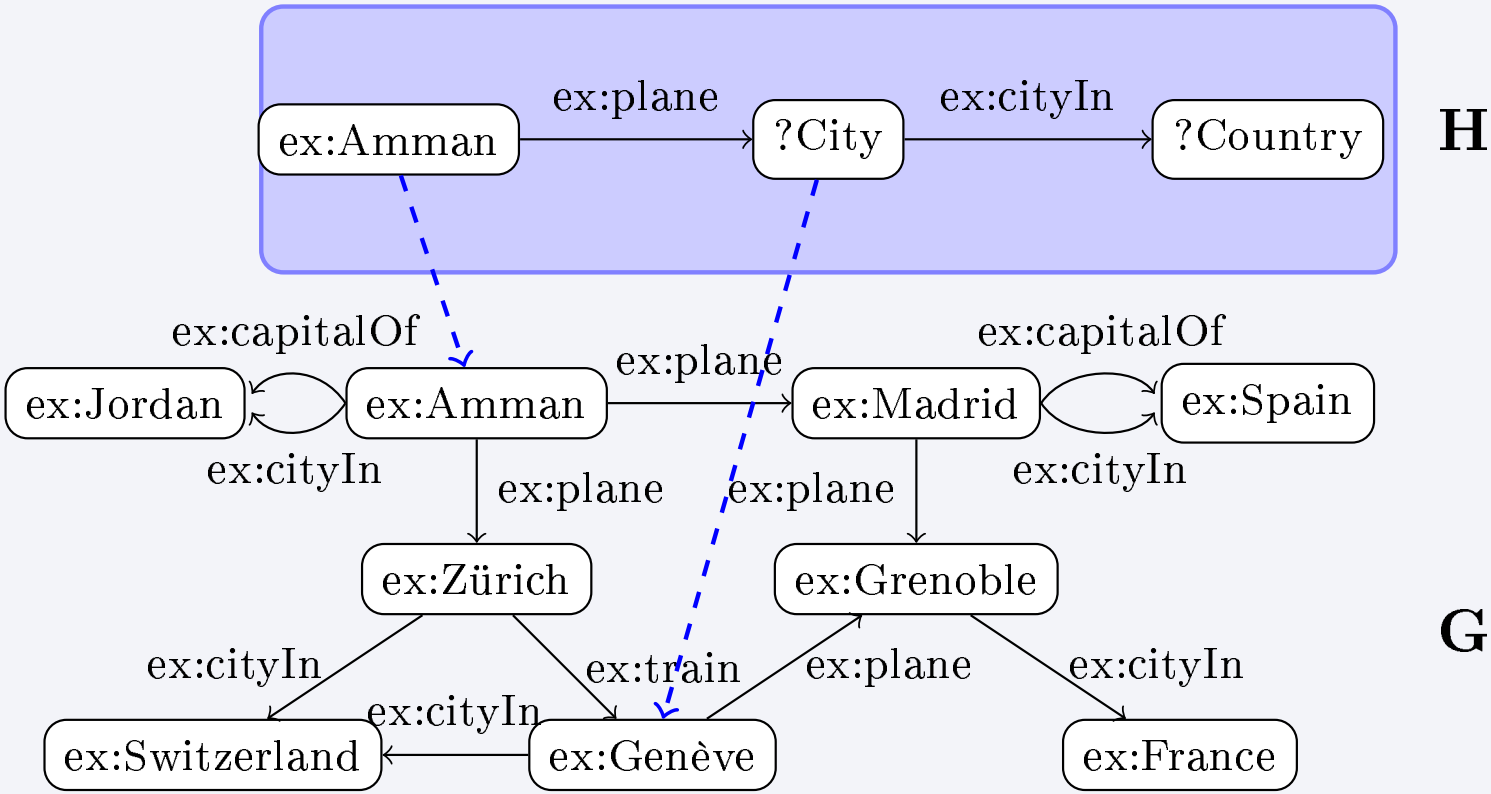
GRDF Homomorphism: example



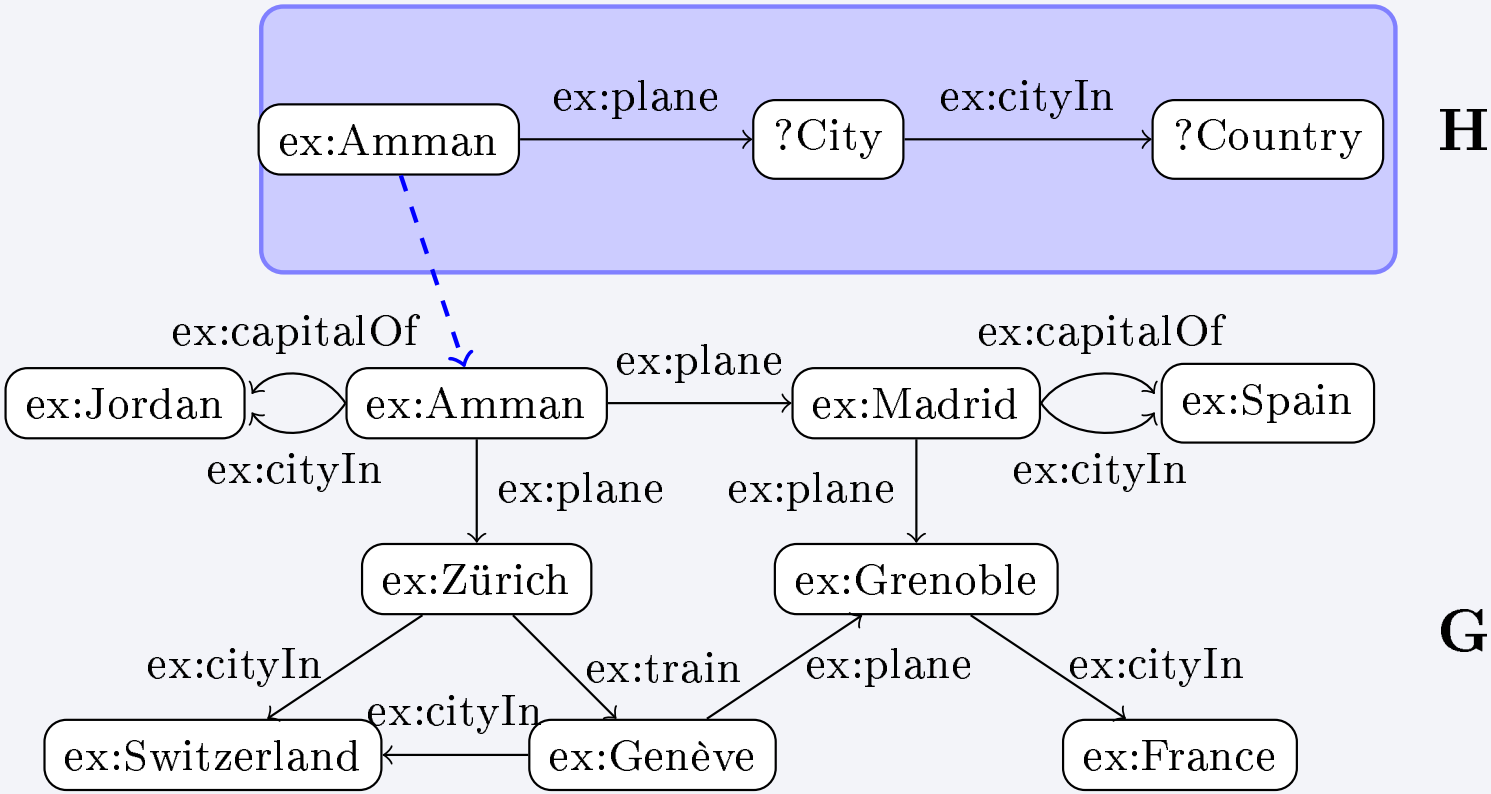
GRDF Homomorphism: example



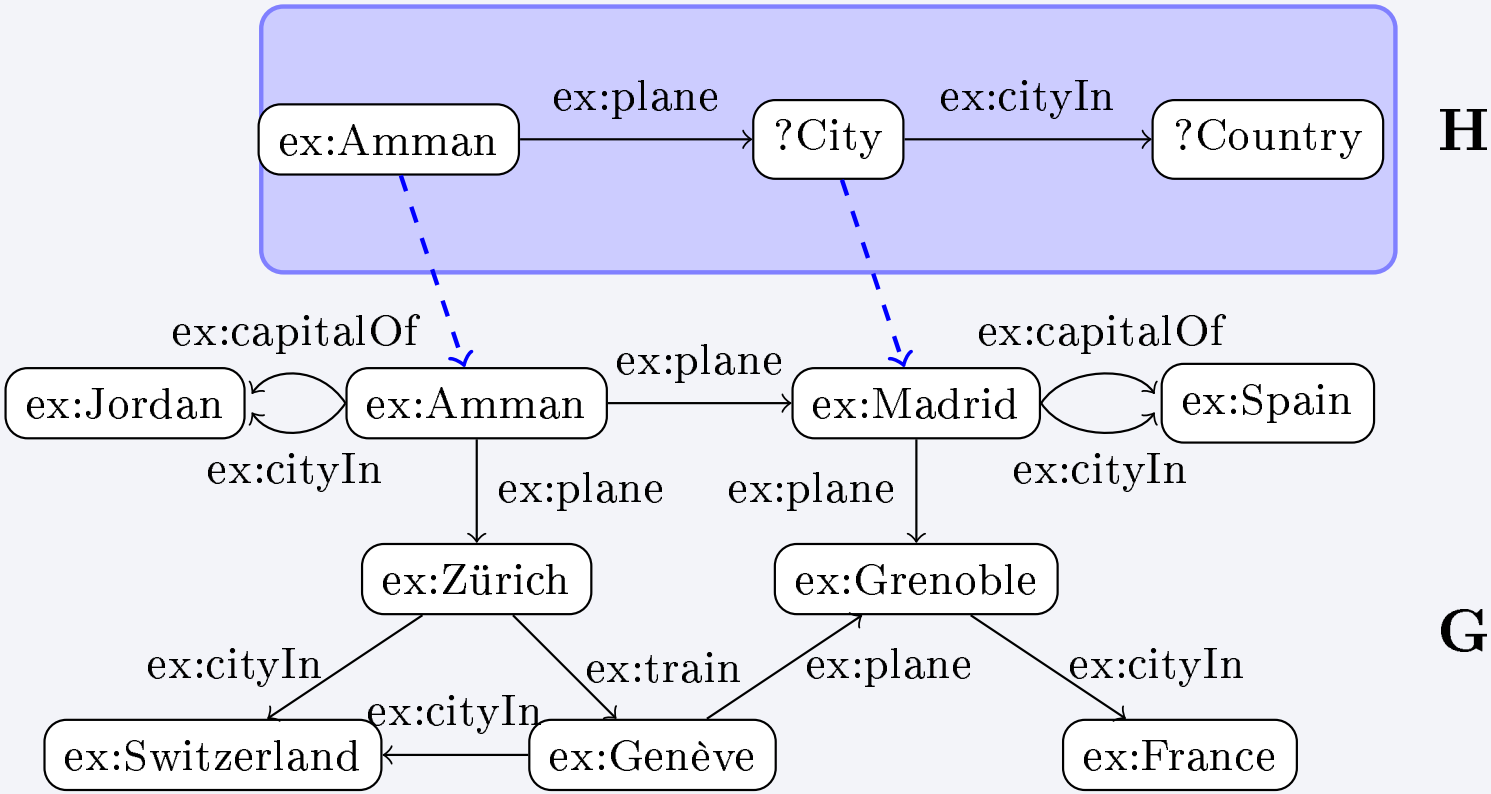
GRDF Homomorphism: example



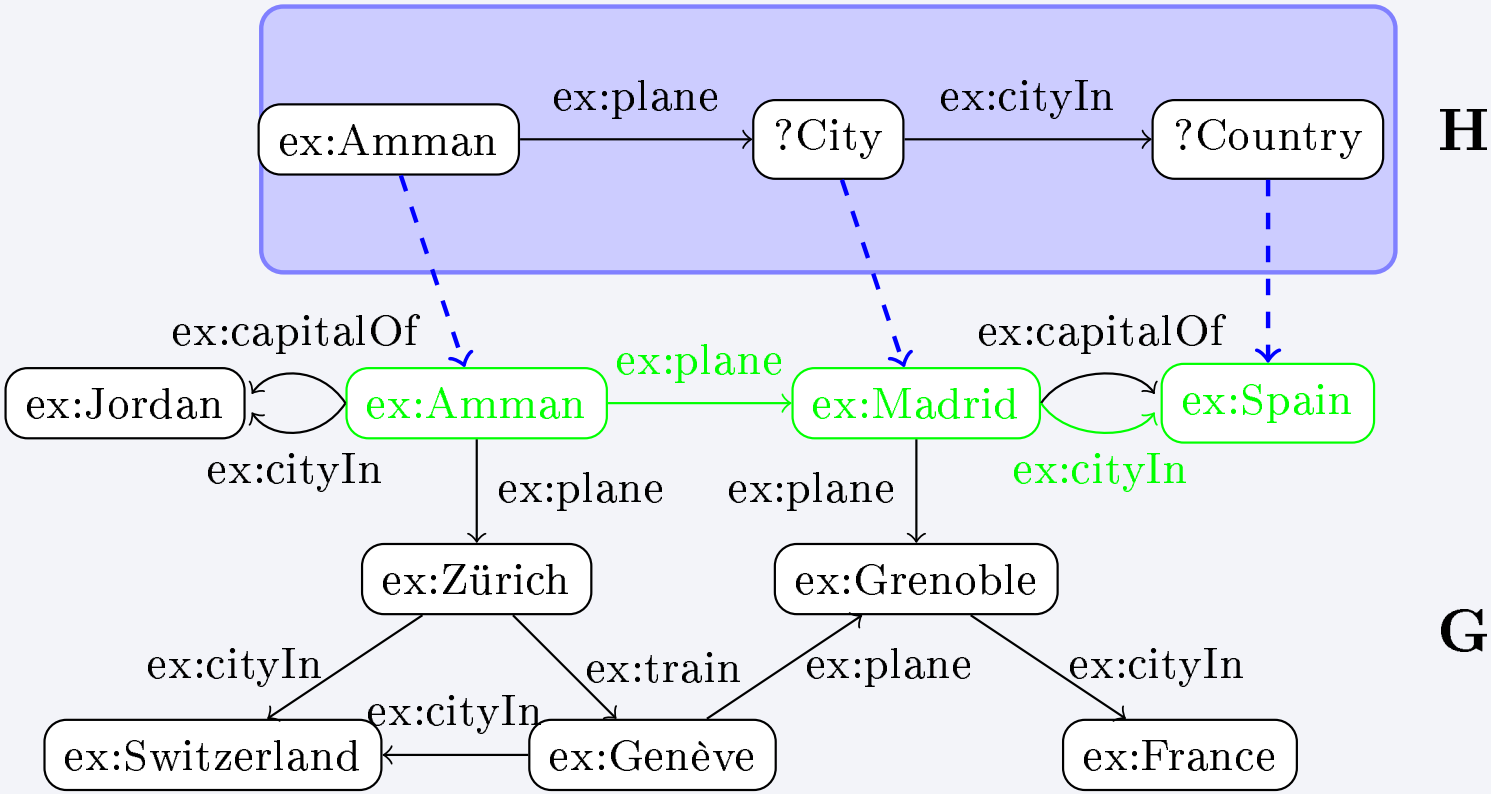
GRDF Homomorphism: example



GRDF Homomorphism: example



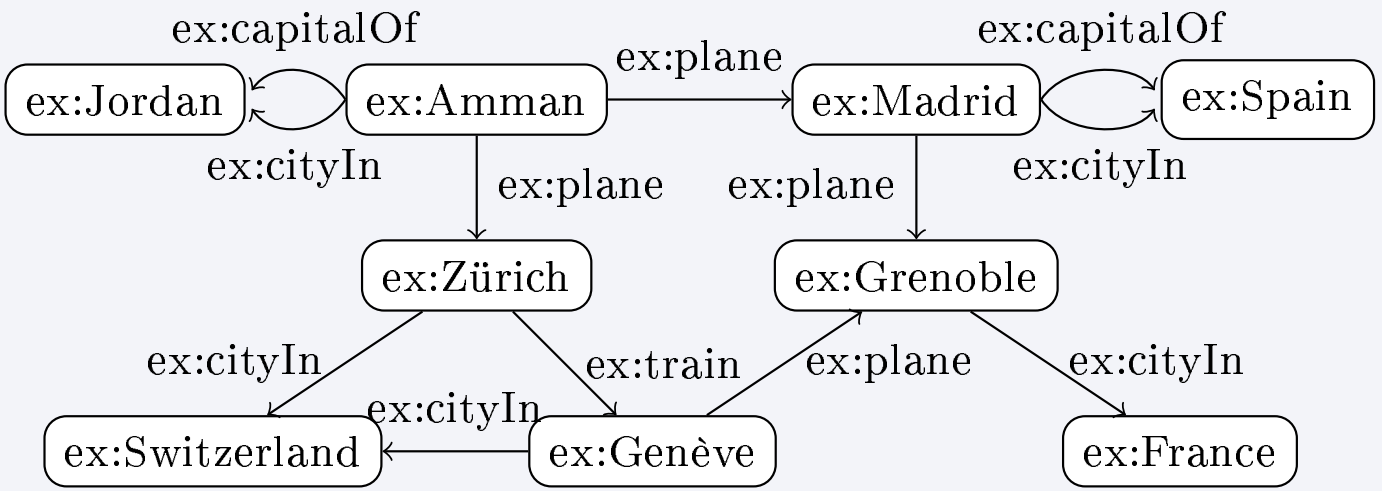
GRDF Homomorphism: example



Path-satisfiability

PATH-SATISFIABILITY

- $R = (\text{ex:plane} \mid \text{ex:train})^+$

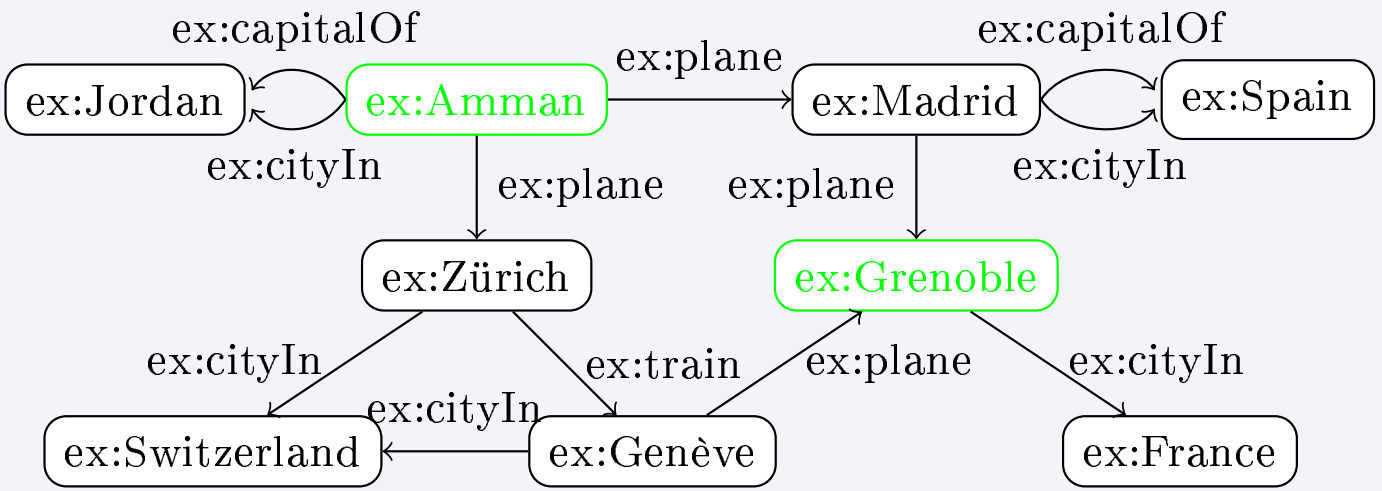


G

Path-satisfiability

PATH-SATISFIABILITY

- $R = (\text{ex:plane} \mid \text{ex:train})^+$

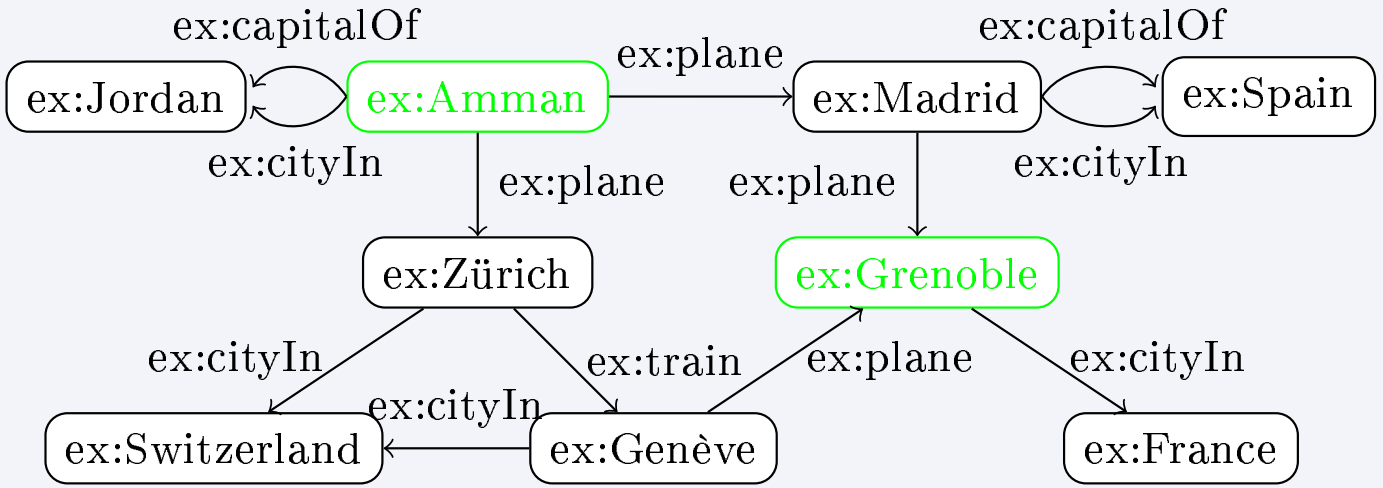


G

Path-satisfiability

PATH-SATISFIABILITY

- R =(ex:plane | ex:train)$^+$
- Does ⟨ex:Amman, ex:Grenoble⟩ satisfies (ex:plane | ex:train)$^+$

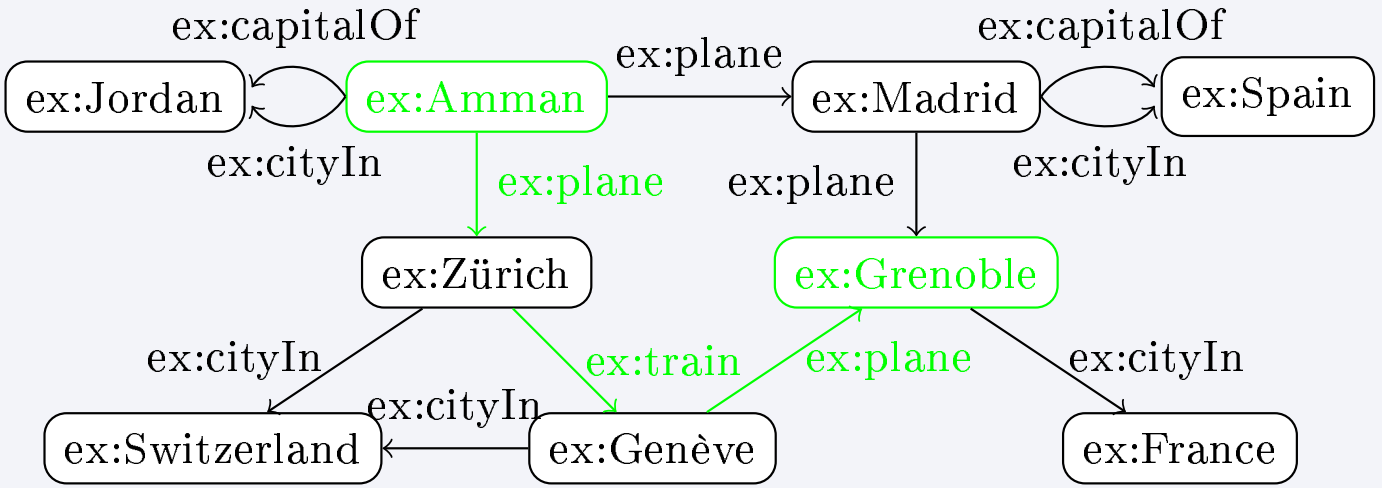


G

Path-satisfiability

PATH-SATISFIABILITY

- R =(ex:plane | ex:train)$^+$
- Does ⟨ex:Amman, ex:Grenoble⟩ satisfies (ex:plane | ex:train)$^+$
- plane·train·plane $\in L^*$((ex:plane | ex:train)$^+$)



G

PRDF Homomorphism

- A PRDF homomorphism from a PRDF graph H into a GRDF graph G is a map π such that $\forall\langle s, R, o\rangle \in H$, $\langle \pi(s), \pi(o)\rangle$ satisfies $\pi(R)$ in G.

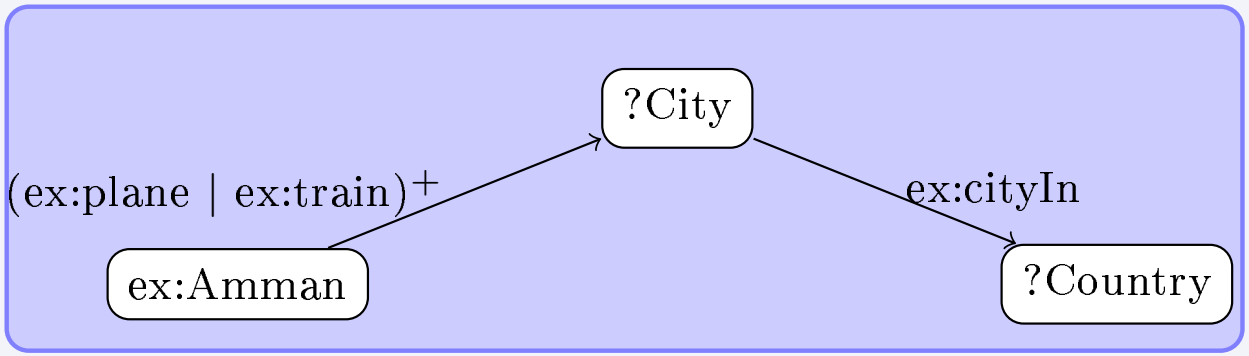
PRDF Homomorphism

- A PRDF homomorphism from a PRDF graph H into a GRDF graph G is a map π such that $\forall\langle s, R, o\rangle \in H$, $\langle \pi(s), \pi(o)\rangle$ satisfies $\pi(R)$ in G.

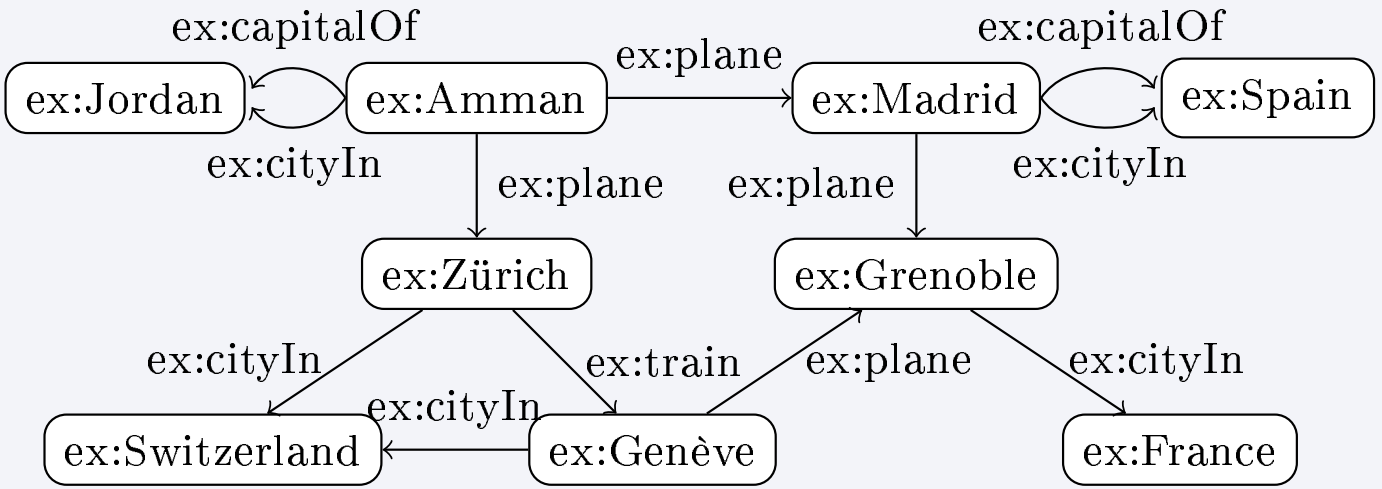
Theorem (Soundness and Completeness)

Let G be a GRDF graph, and H be a PRDF graph. Then there is a PRDF homomorphism from H into G iff $G \models_{PRDF} H$.

PRDF Homomorphism: example

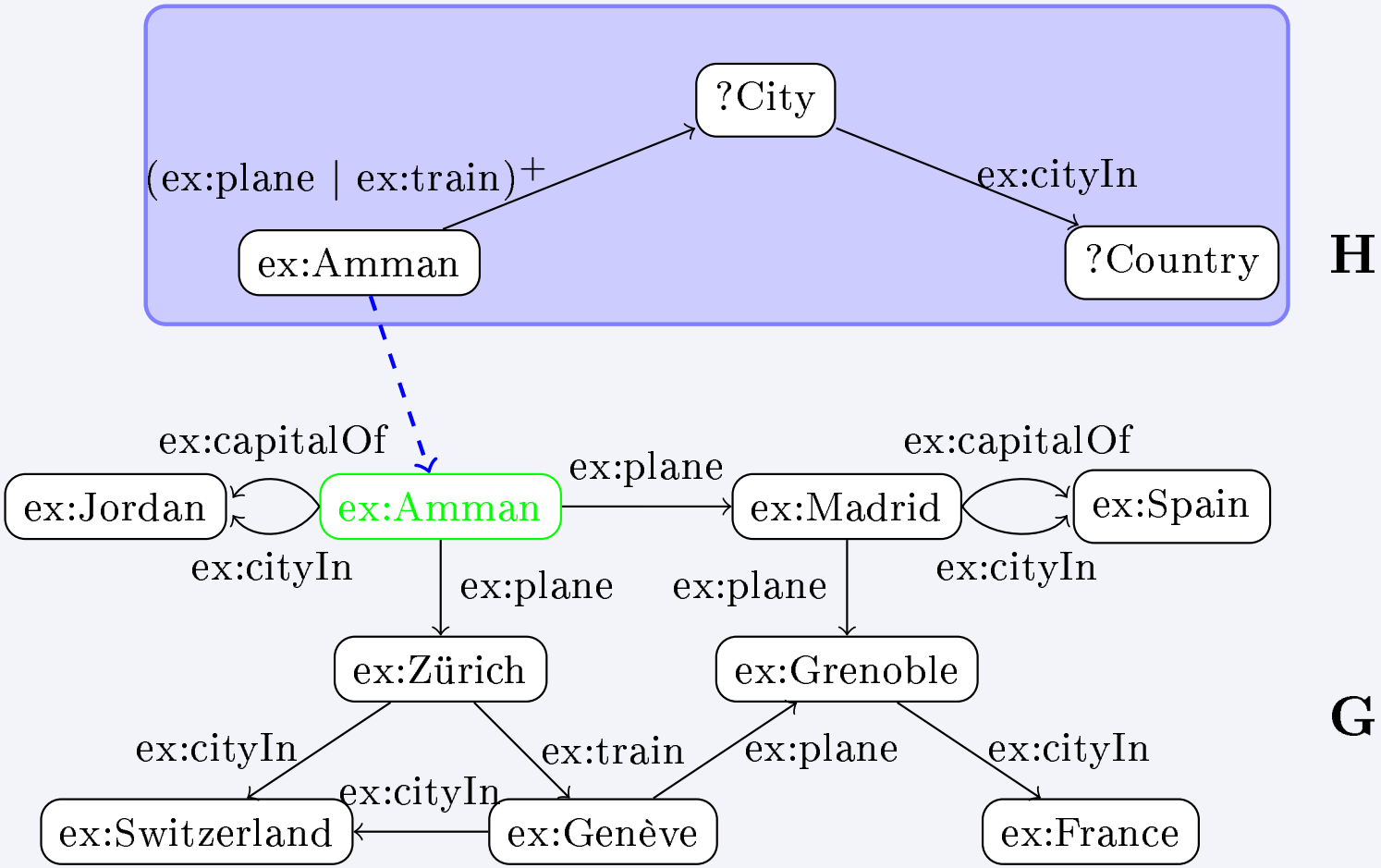


H

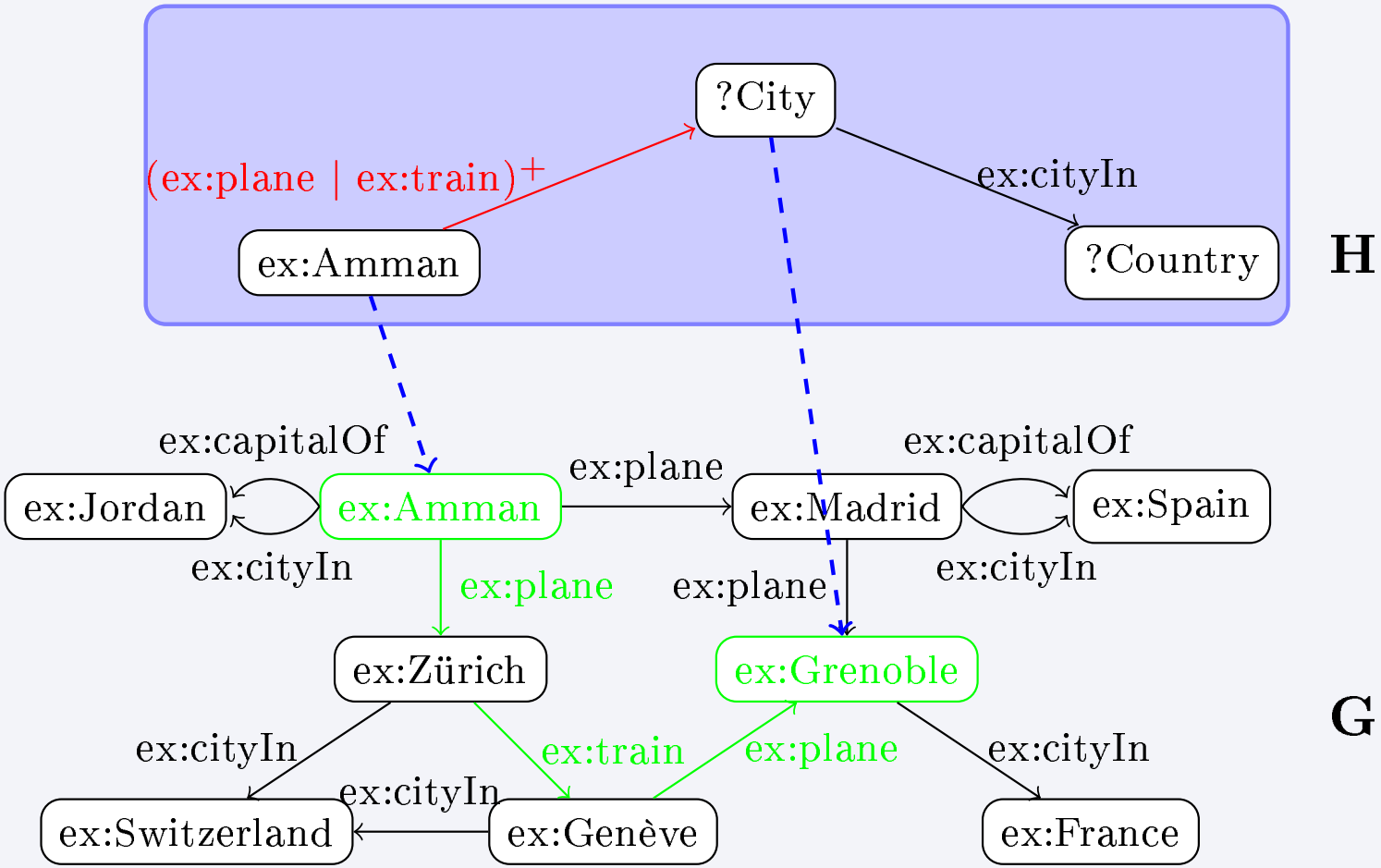


G

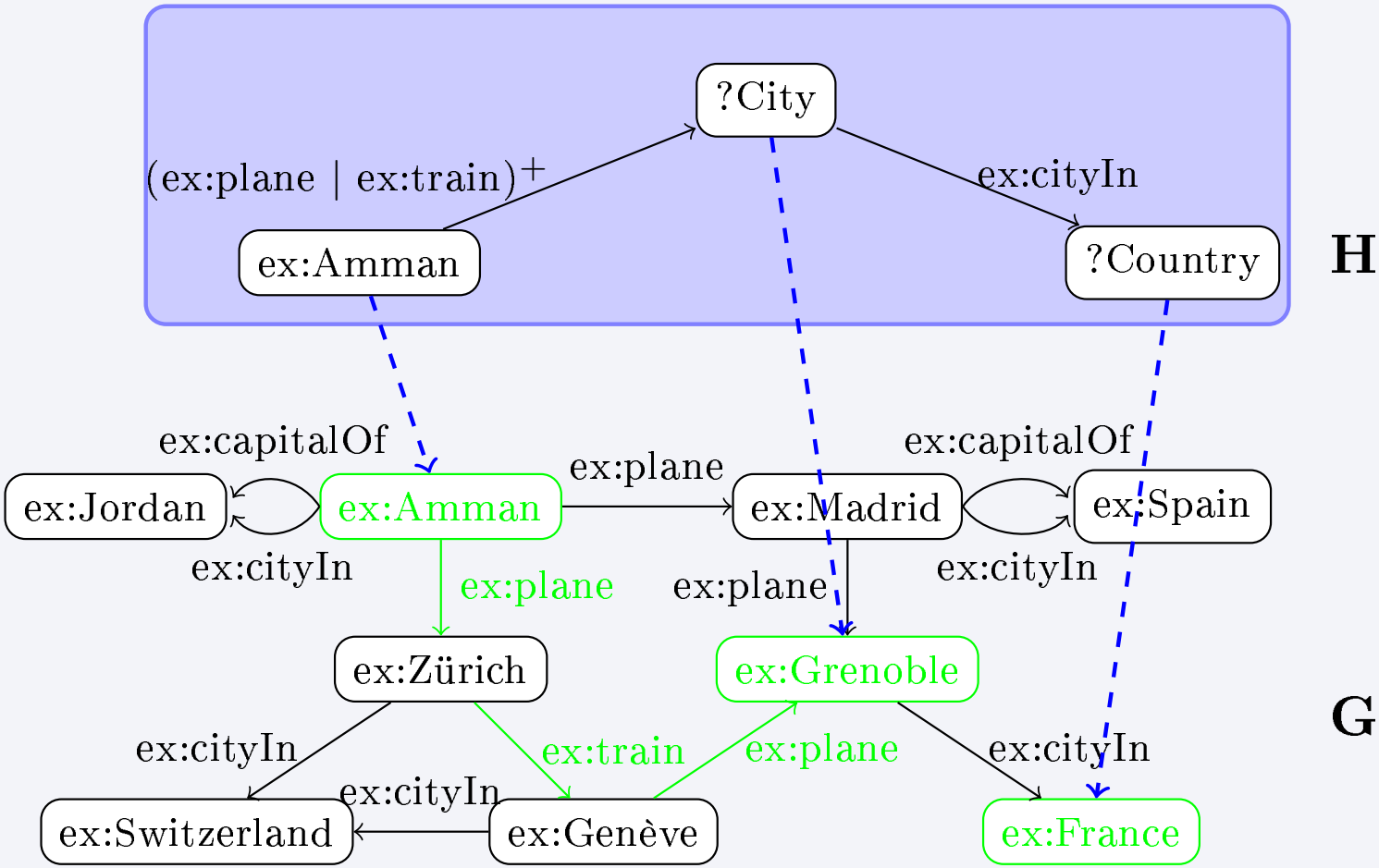
PRDF Homomorphism: example



PRDF Homomorphism: example



PRDF Homomorphism: example



Outline

CPRDF syntax

$$\text{CPRDF} = \text{RDF} + \text{constrained regular expressions}$$

GRDF constraint

Definition

A GRDF constraint is written $\dagger_1 Qx \dagger_2 : C$ where:

GRDF constraint

Definition

A GRDF constraint is written $\dagger_1 Qx \dagger_2 : C$ where:

- C is a GRDF graph.

Example

]ALL ?Stop[: {(?Stop, ex:capitalOf, ?Country)}

GRDF constraint

Definition

A GRDF constraint is written $\dagger_1 Q x \dagger_2 : C$ where:

- x is a variable that occurs in a triple of C,
- C is a GRDF graph.

Example

]ALL ?Stop[: {(?Stop, ex:capitalOf, ?Country)}

GRDF constraint

Definition

A GRDF constraint is written $\dagger_1 Q x \dagger_2 : C$ where:

- Q is either a quantifier `ALL`, `EXISTS` or `EDGE` (constraints applied to edges),
- x is a variable that occurs in a triple of C,
- C is a GRDF graph.

Example

]ALL ?Stop[: {(?Stop, ex:capitalOf, ?Country)}

GRDF constraint

Definition

A GRDF constraint is written $\dagger_1 Q x \dagger_2 : C$ where:

- $\dagger_1$ and $\dagger_2$ are one of the interval delimiters [and],
- Q is either a quantifier `ALL`, `EXISTS` or `EDGE` (constraints applied to edges),
- x is a variable that occurs in a triple of C,
- C is a GRDF graph.

Example

]`ALL` ?Stop[: {(?Stop, ex:capitalOf, ?Country)}

GRDF constraint

Definition

A GRDF constraint is written $\dagger_1 Q x \dagger_2 : C$ where:

- $\dagger_1$ and $\dagger_2$ are one of the interval delimiters [and],
- Q is either a quantifier `ALL`, `EXISTS` or `EDGE` (constraints applied to edges),
- x is a variable that occurs in a triple of C,
- C is a GRDF graph.
- $\Phi = \Phi^E \cup \Phi^N$

Example

]`ALL` ?Stop[: {(?Stop, ex:capitalOf, ?Country)}

Constrained Regular Expressions

Definition (Constrained regular expression)

The set of constrained regular expressions $\mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$ is defined inductively by:

- if $u \in \mathcal{U}$ and $\psi \in \Phi^E$, then u, $!u$, $u\%\psi\%$, $!u\%\psi\%$, u^- and $u^-\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;

Constrained Regular Expressions

Definition (Constrained regular expression)

The set of constrained regular expressions $\mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$ is defined inductively by:

- if $u \in \mathcal{U}$ and $\psi \in \Phi^E$, then u, $!u$, $u\%\psi\%$, $!u\%\psi\%$, u^- and $u^-\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $b \in \mathcal{B}$ and $\psi \in \Phi^E$, then b, $b\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $\psi \in \Phi^E$, then $\#$, $\#\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;

Constrained Regular Expressions

Definition (Constrained regular expression)

The set of constrained regular expressions $\mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$ is defined inductively by:

- if $u \in \mathcal{U}$ and $\psi \in \Phi^E$, then u, $!u$, $u\%\psi\%$, $!u\%\psi\%$, u^- and $u^-\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $b \in \mathcal{B}$ and $\psi \in \Phi^E$, then b, $b\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $\psi \in \Phi^E$, then $\#$, $\#\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $R \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$, then $(R^*) \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;

Constrained Regular Expressions

Definition (Constrained regular expression)

The set of constrained regular expressions $\mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$ is defined inductively by:

- if $u \in \mathcal{U}$ and $\psi \in \Phi^E$, then u, $!u$, $u\%\psi\%$, $!u\%\psi\%$, u^- and $u^-\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $b \in \mathcal{B}$ and $\psi \in \Phi^E$, then b, $b\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $\psi \in \Phi^E$, then $\#$, $\#\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $R \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$, then $(R^*) \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $R_1, R_2 \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$, then $(R_1 \cdot R_2)$, and $(R_1|R_2)$ are elements of $\mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$.

Constrained Regular Expressions

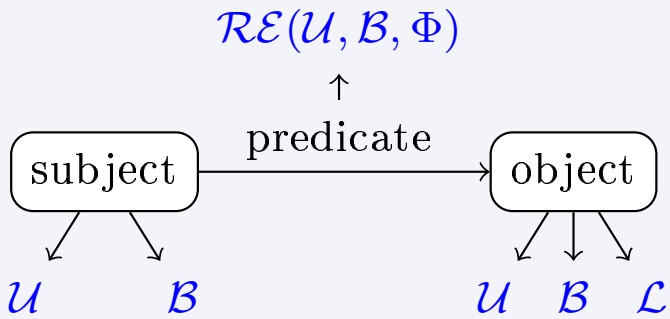
Definition (Constrained regular expression)

The set of constrained regular expressions $\mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$ is defined inductively by:

- if $u \in \mathcal{U}$ and $\psi \in \Phi^E$, then u, $!u$, $u\%\psi\%$, $!u\%\psi\%$, u^- and $u^-\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $b \in \mathcal{B}$ and $\psi \in \Phi^E$, then b, $b\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $\psi \in \Phi^E$, then $\#$, $\#\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $R \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$, then $(R^*) \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$;
- if $R_1, R_2 \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$, then $(R_1 \cdot R_2)$, and $(R_1|R_2)$ are elements of $\mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$.
- if $R \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$, $\psi \in \Phi^N$ is a constraint, then $R\%\psi\% \in \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi)$.

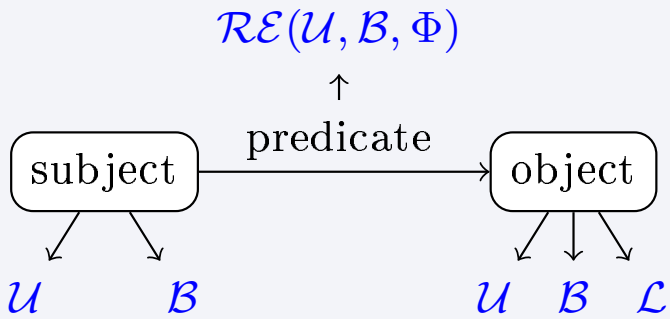
CPRDF Syntax

- CPRDF Triple: $\langle subject, predicate, object \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi) \times \mathcal{T}$



CPRDF Syntax

- CPRDF Triple: $\langle subject, predicate, object \rangle \in (\mathcal{U} \cup \mathcal{B}) \times \mathcal{RE}(\mathcal{U}, \mathcal{B}, \Phi) \times \mathcal{T}$

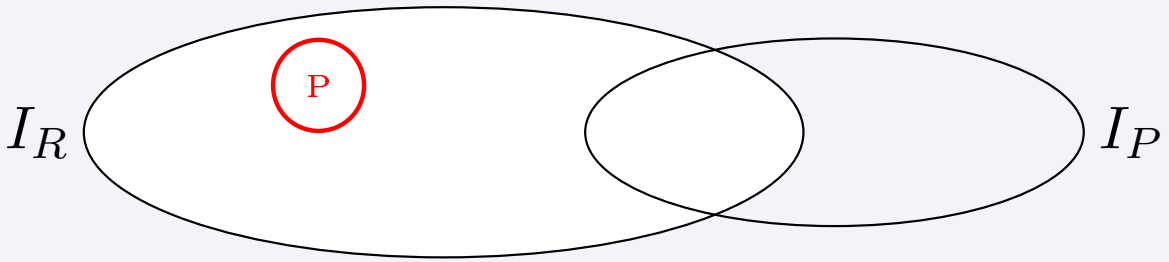


- CPRDF graph: a set of CPRDF triples

Satisfied constraint in an interpretation

Example

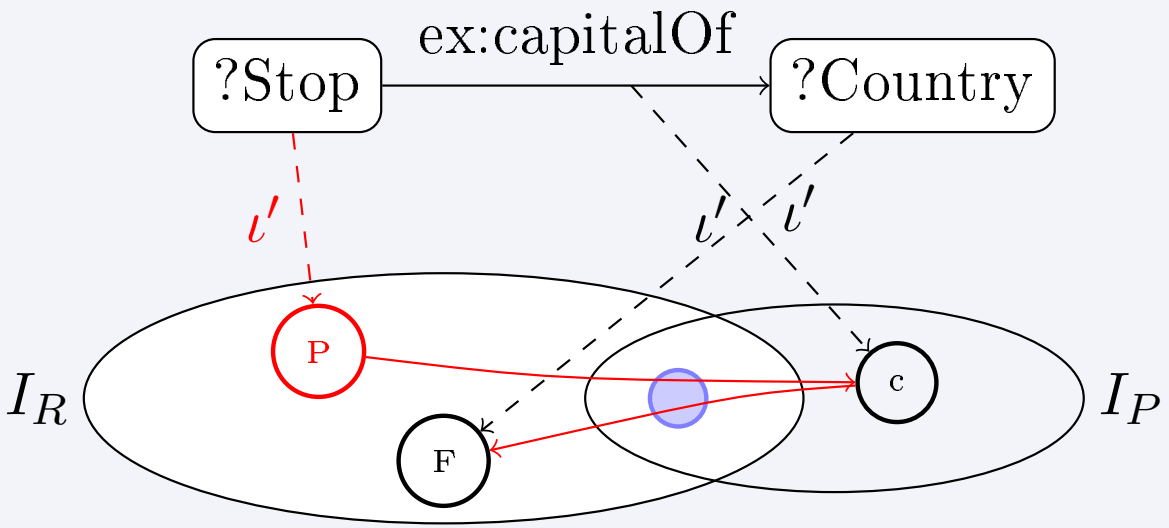
]ALL ?Stop[: {(?Stop, ex:capitalOf, ?Country)}



Satisfied constraint in an interpretation

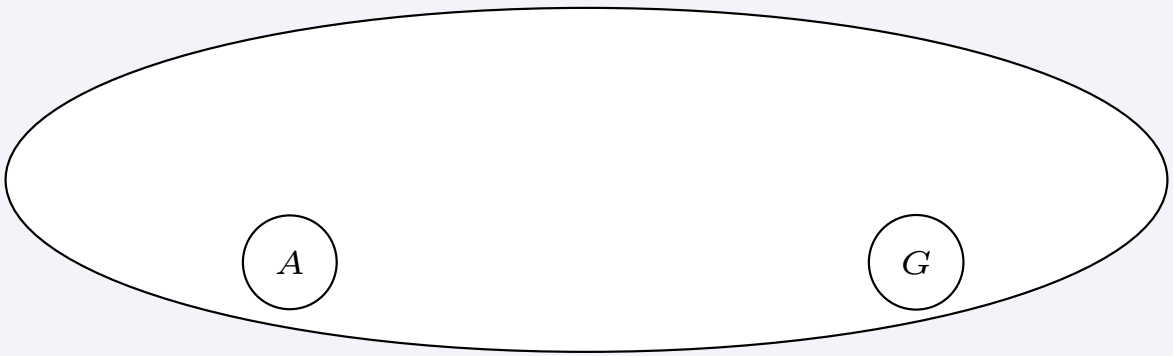
Example

]ALL ?Stop[: {(?Stop, ex:capitalOf, ?Country)}



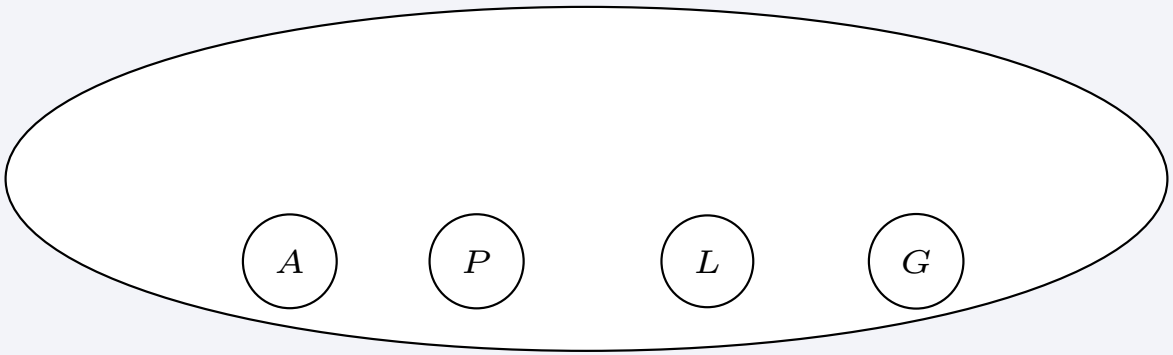
Semantics for constrained regular expressions

- $const1$ =]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^{+} \%const1\% \cdot \texttt{bus})$ in ι'?



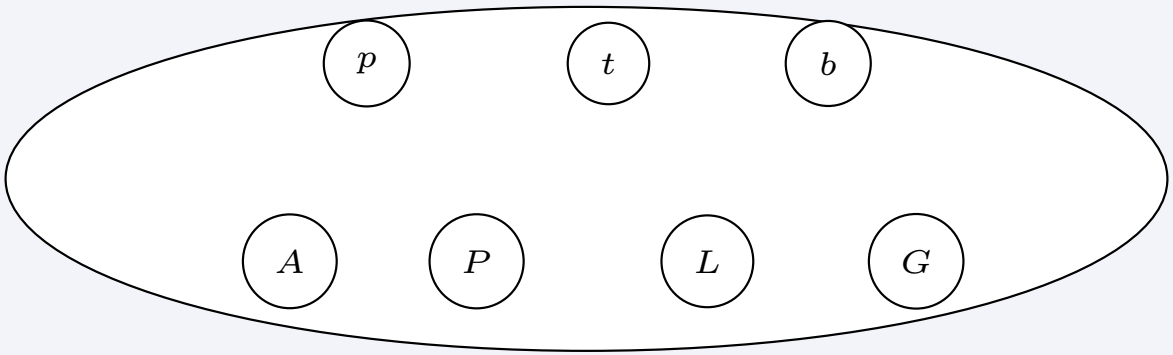
Semantics for constrained regular expressions

- $const1 =$]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^{+} \%const1\% \cdot \texttt{bus})$ in ι'?



Semantics for constrained regular expressions

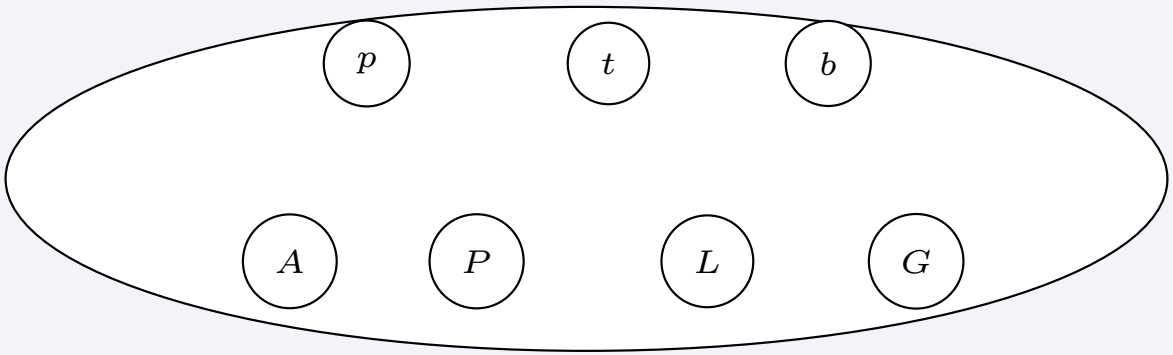
- $const1$ =]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^{+} \%const1\% \cdot \texttt{bus})$ in ι'?



Semantics for constrained regular expressions

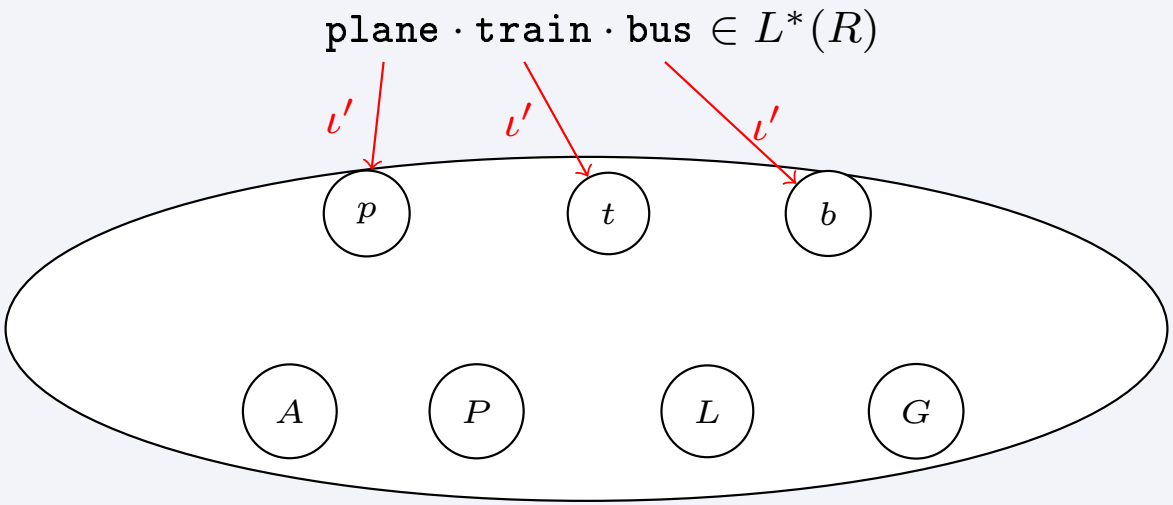
- $const1 =$]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^{+} \%const1\% \cdot \texttt{bus})$ in ι'?

$$\texttt{plane} \cdot \texttt{train} \cdot \texttt{bus} \in L^{*}(R)$$



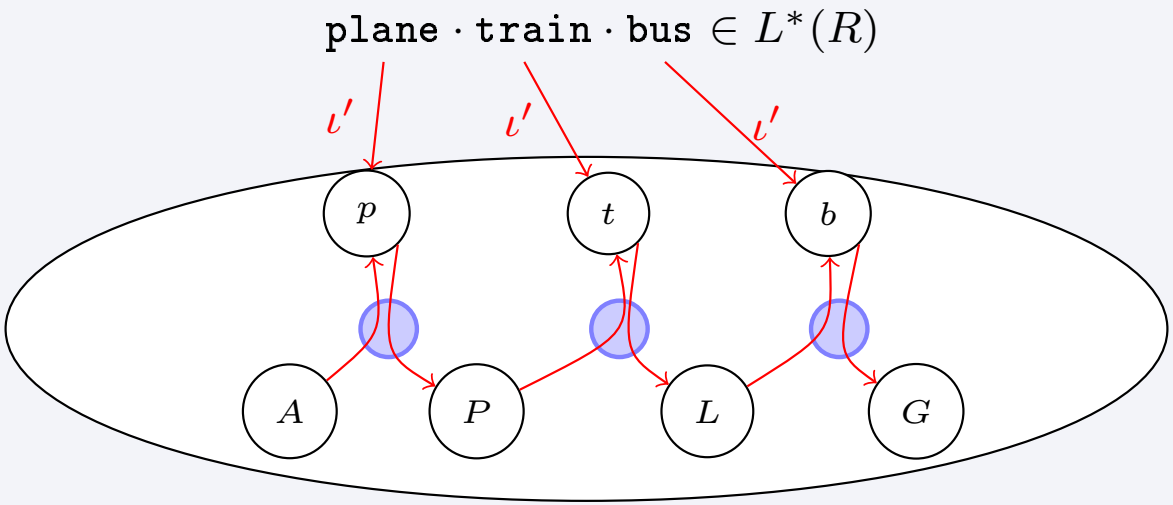
Semantics for constrained regular expressions

- $const1$ =]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^{+} \%const1\% \cdot \texttt{bus})$ in ι'?



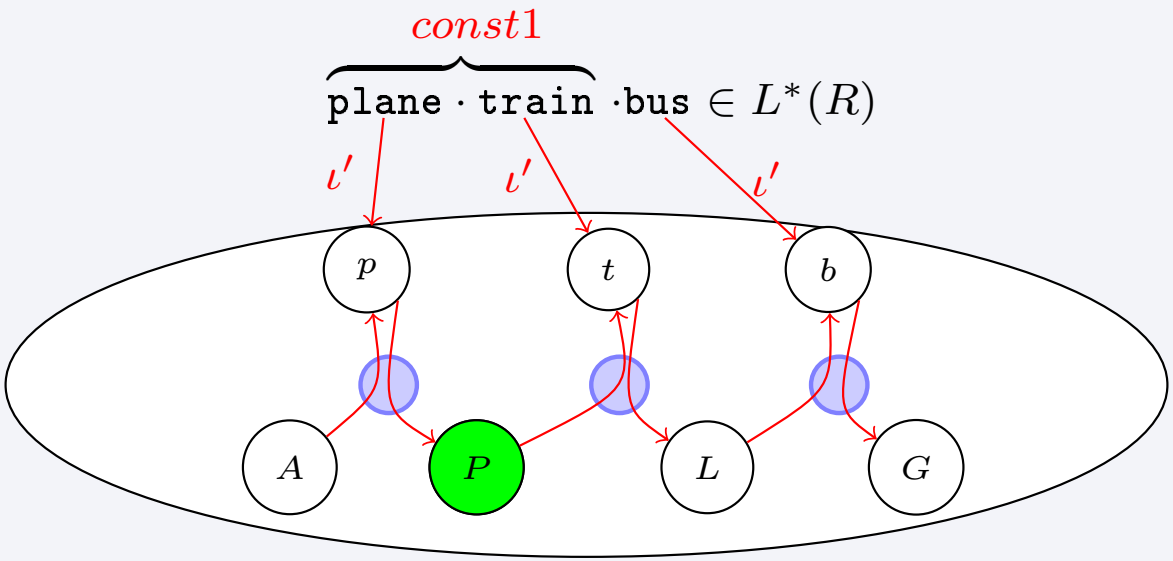
Semantics for constrained regular expressions

- $const1$ =]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^{+} \%const1\% \cdot \texttt{bus})$ in ι'?



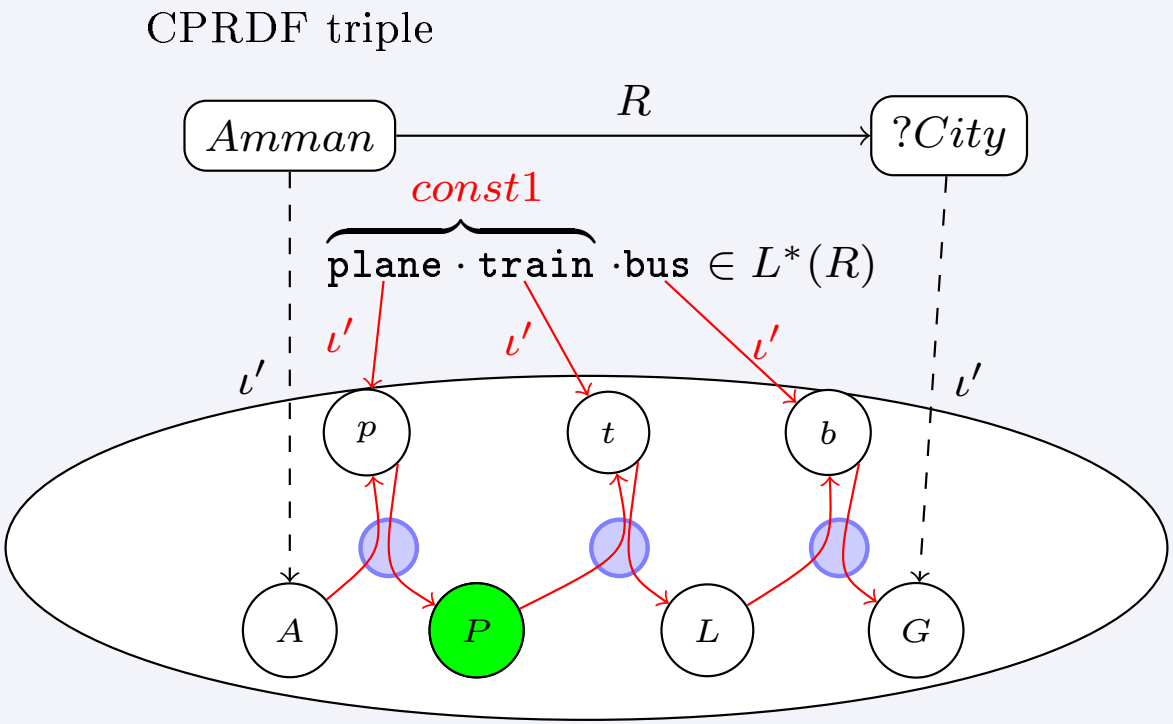
Semantics for constrained regular expressions

- $const1$ =]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^{+} \%const1\% \cdot \texttt{bus})$ in ι'?



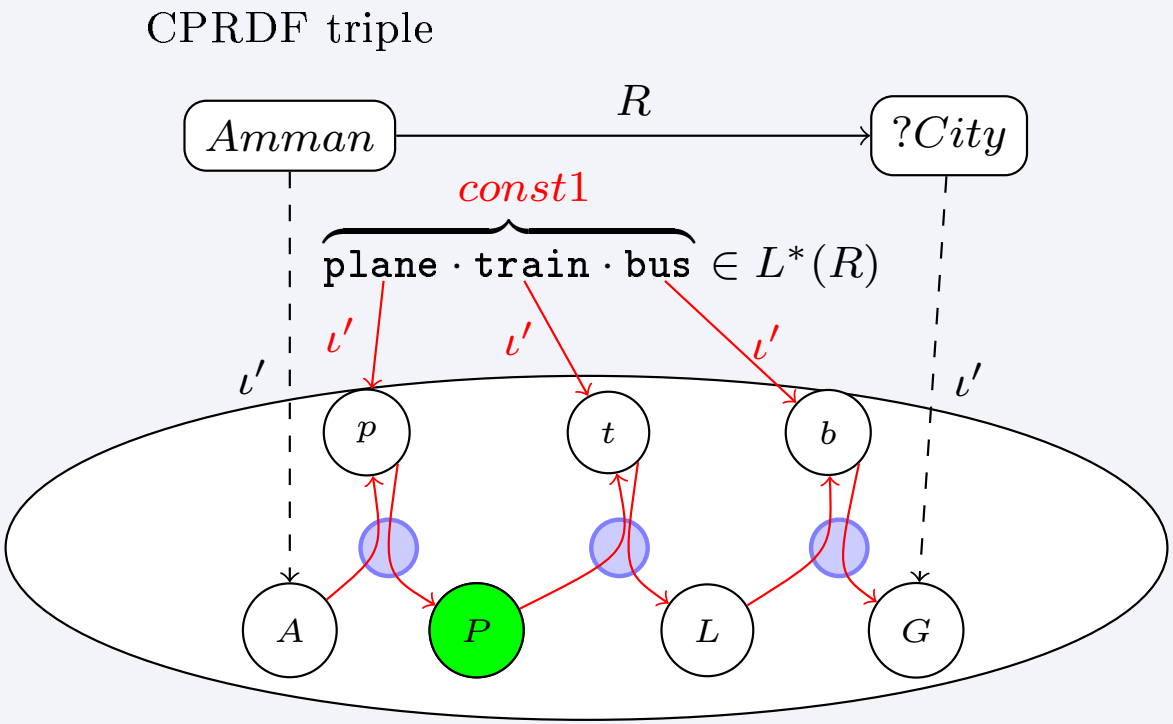
Semantics for constrained regular expressions

- $const1$ =]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G\rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^{+}\%const1\% \cdot \texttt{bus})$ in ι'?



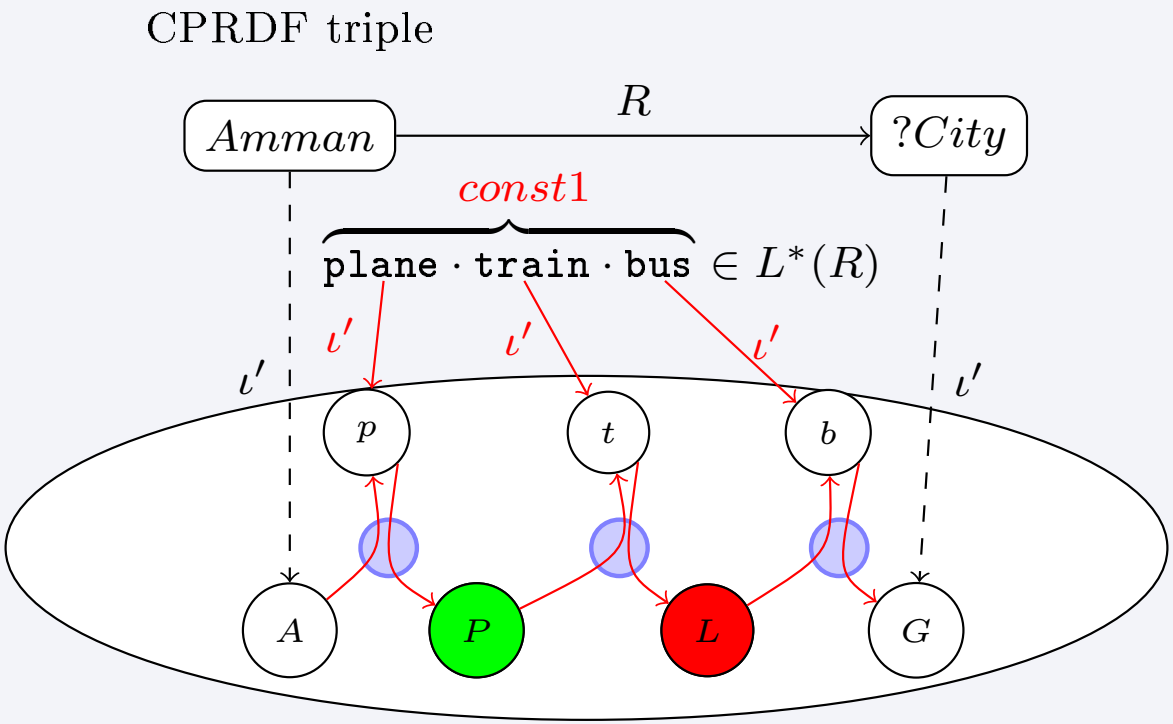
Semantics for constrained regular expressions

- $const1$ =]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G\rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})\%const1\%$ in ι'?



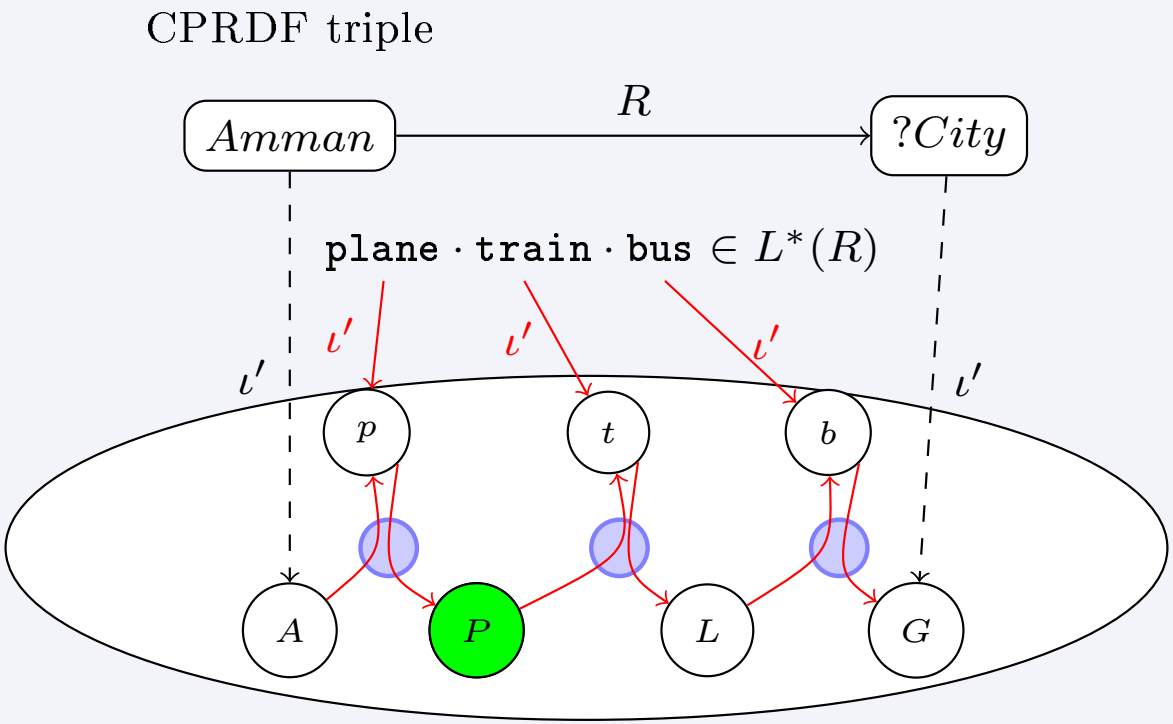
Semantics for constrained regular expressions

- $const1$ =]ALL ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G\rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})\%const1\%$ in ι'?



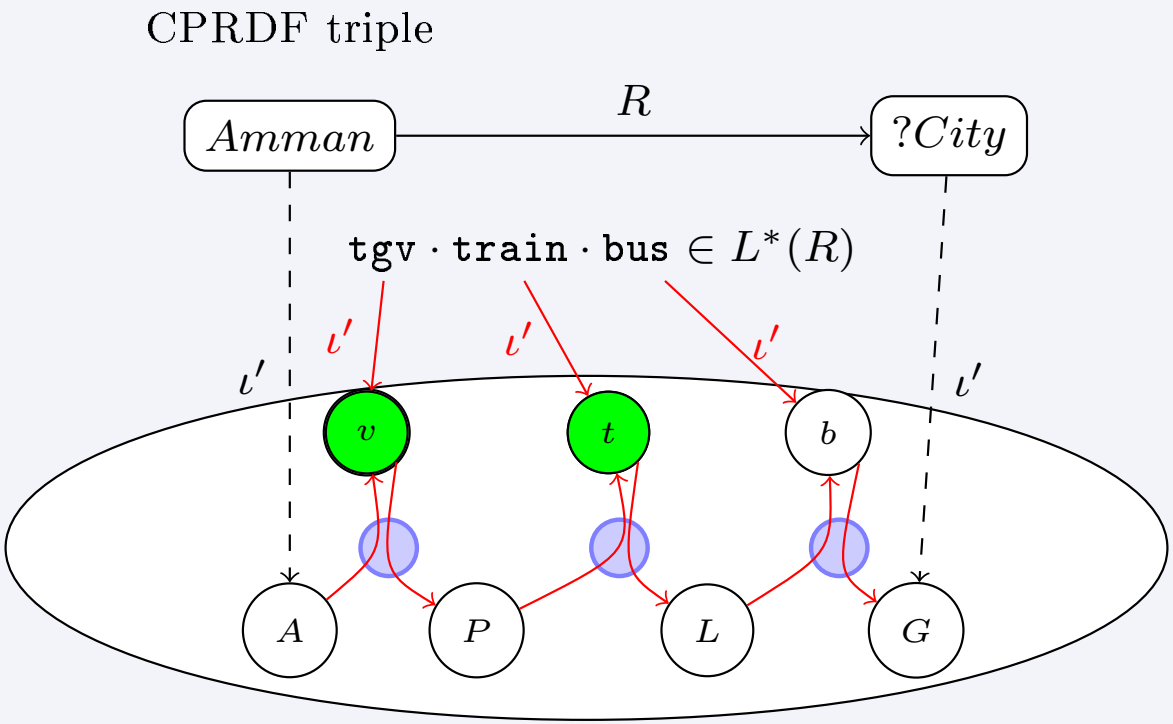
Semantics for constrained regular expressions

- $const1 =$]EXISTS ?Stop [:{(?Stop, ex:capitalOf, ?Country)}
- Does $\langle A, G \rangle$ supports $R = ((\texttt{train} \mid \texttt{plane})^+ \cdot \texttt{bus})\%const1\%$ in ι'?



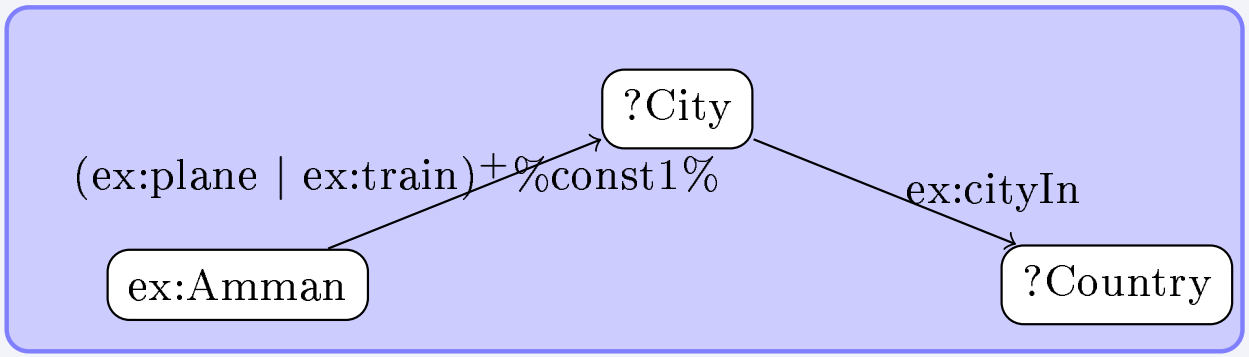
Semantics for constrained regular expressions

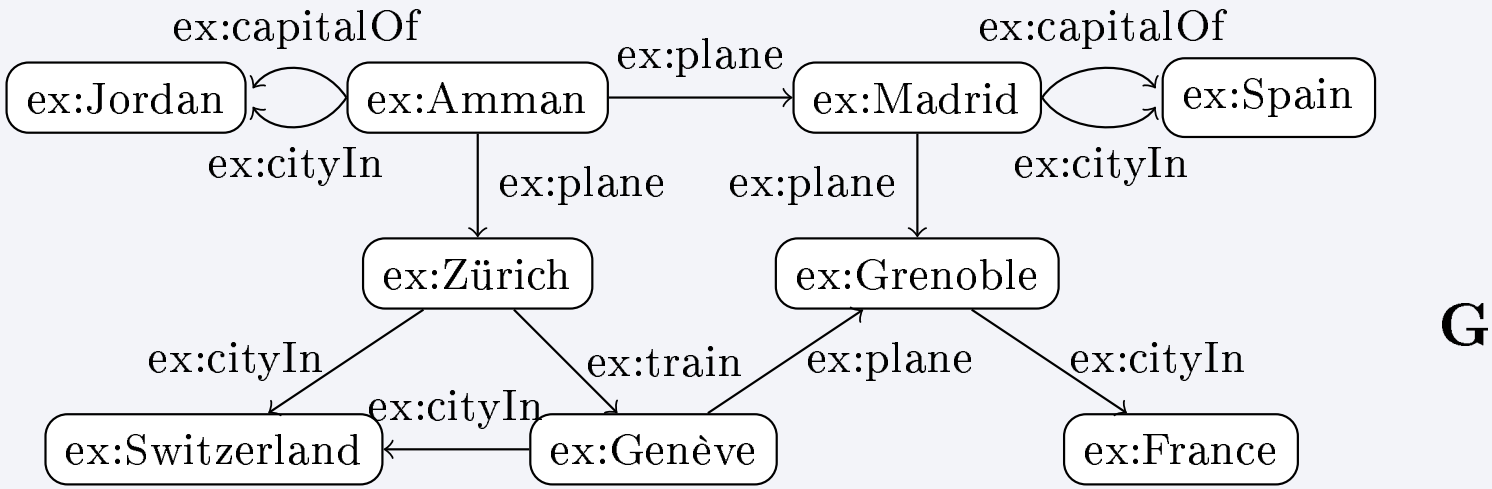
- $const1 =$ [EDGE ?Stop]:{(?Stop, subPropertyOf, train)}
- Does $\langle A, G \rangle$ supports $R = ((\#\%const1\%)^+ \cdot \texttt{bus})$ in ι'?



CPRDF Homomorphism: example

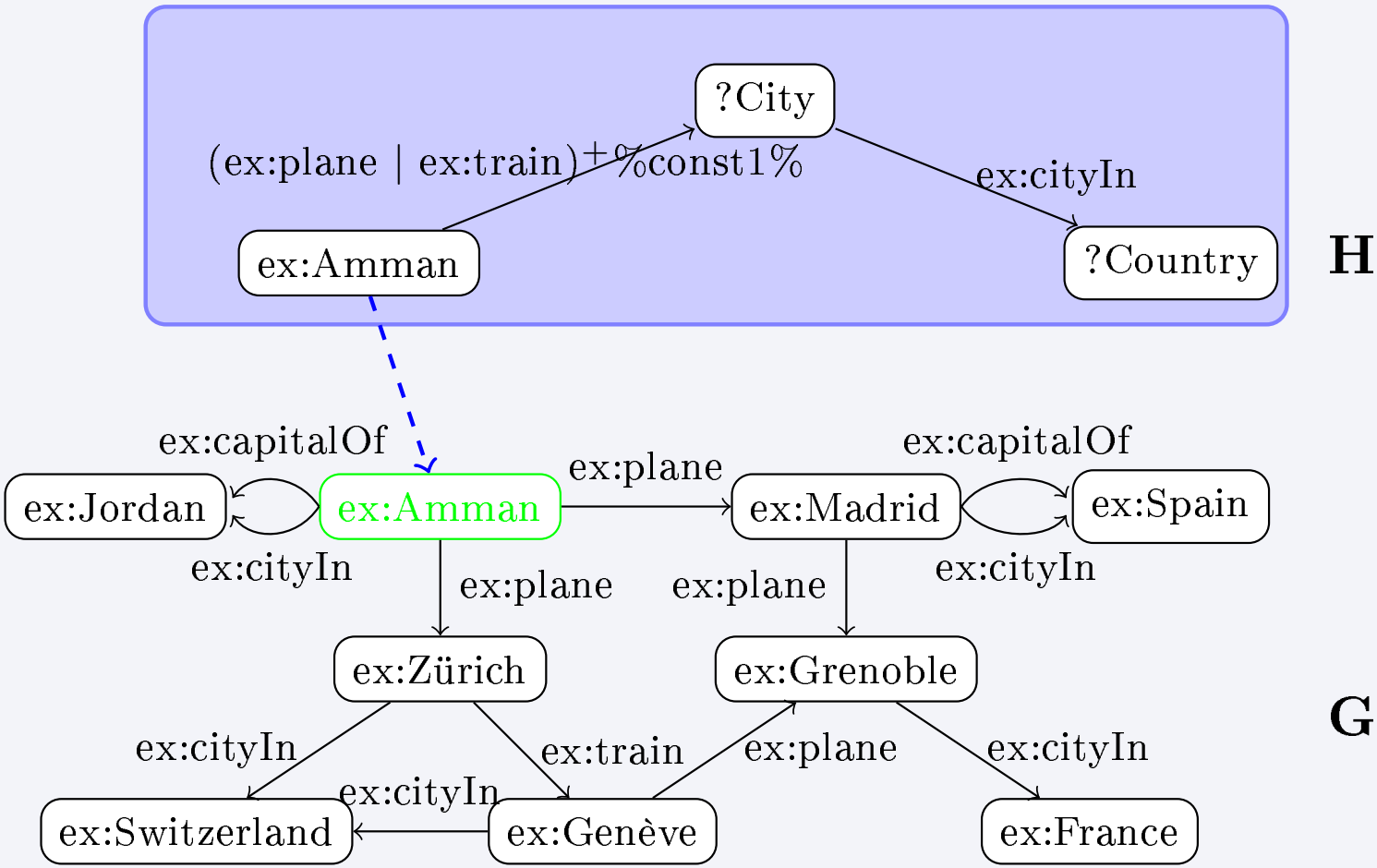
- const1 =]ALL ?Stop[: {?Stop ex:capitalOf ?Country }





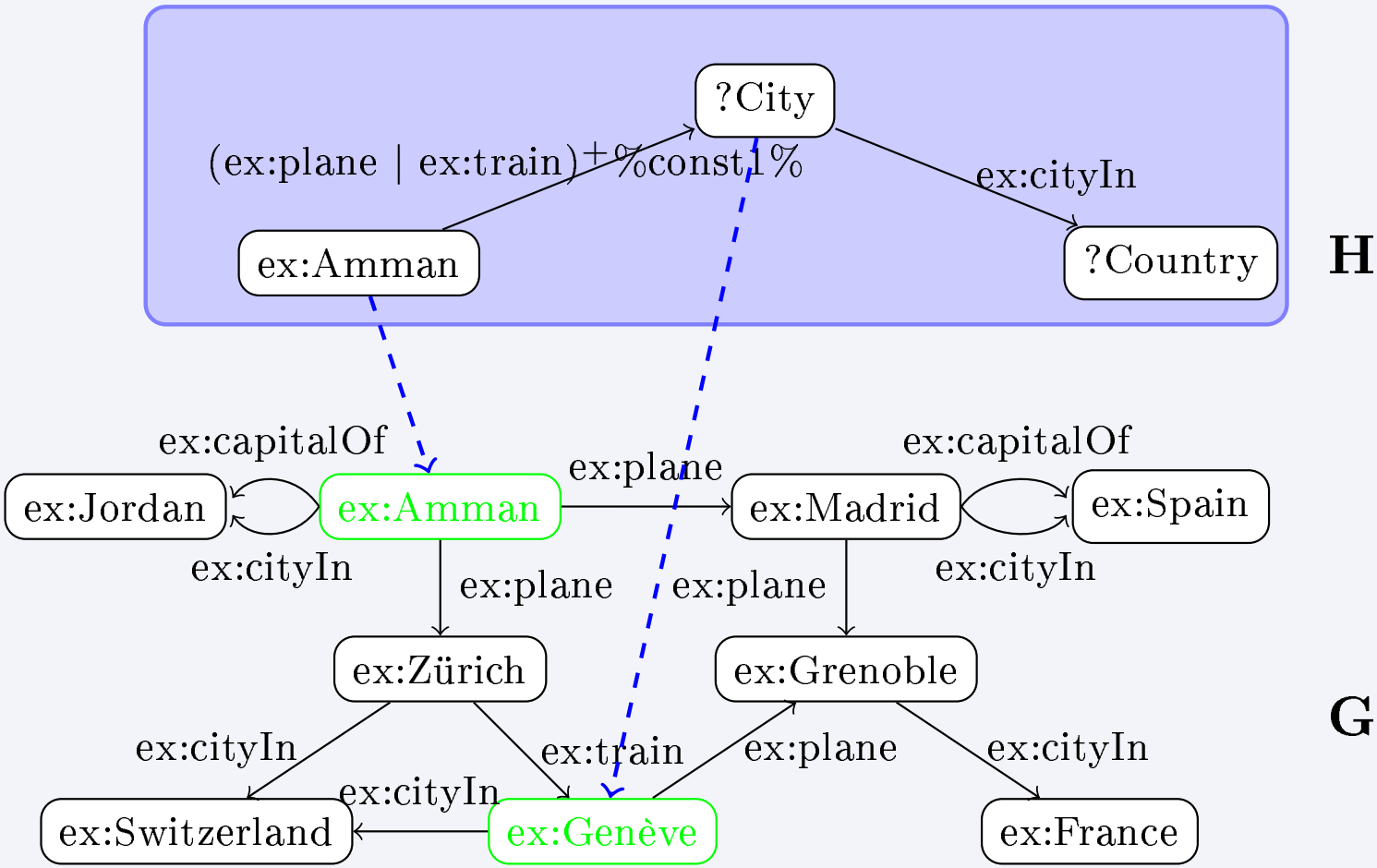
CPRDF Homomorphism: example

- const1 =]ALL ?Stop[: {?Stop ex:capitalOf ?Country }



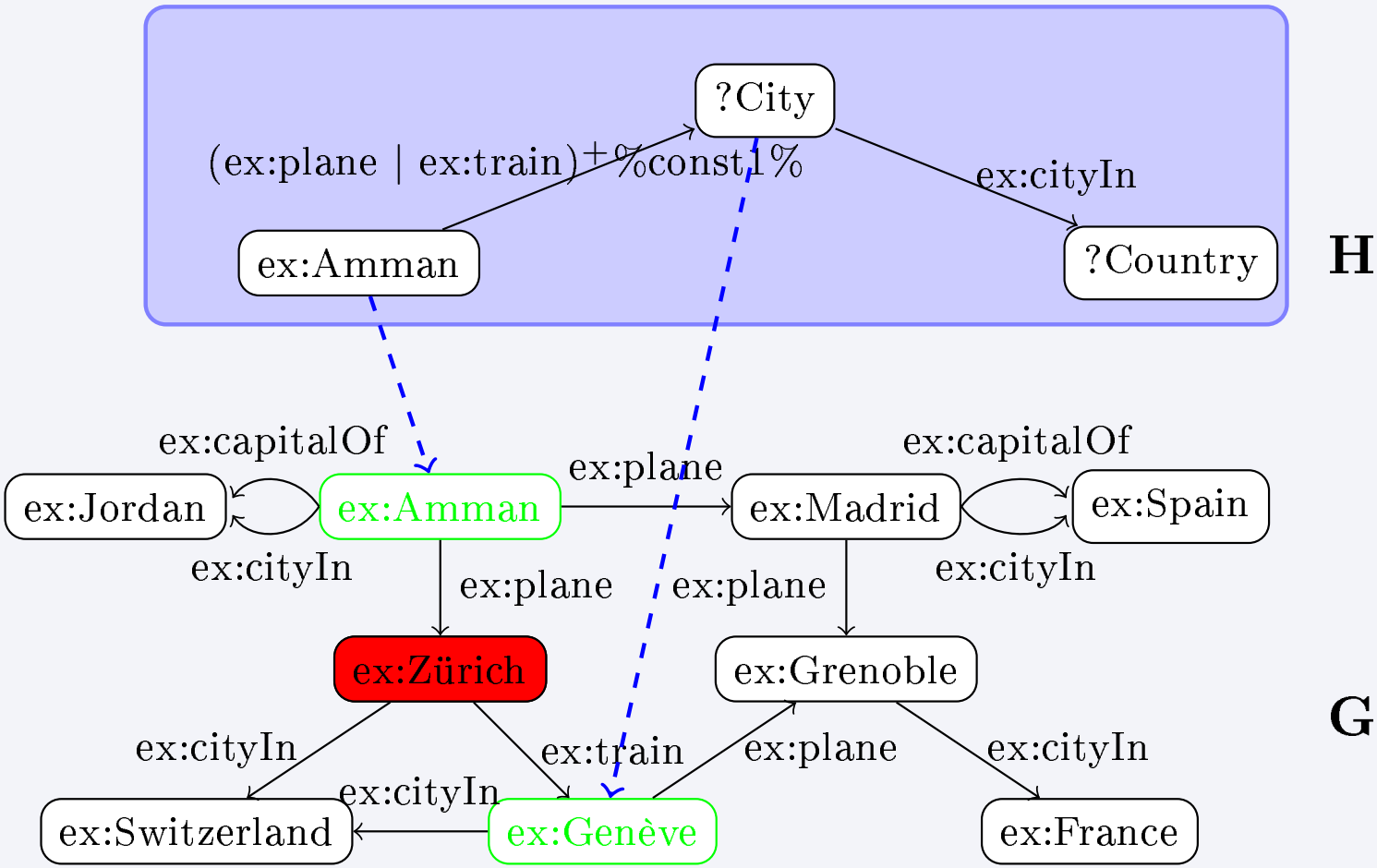
CPRDF Homomorphism: example

- const1 =]ALL ?Stop[: {?Stop ex:capitalOf ?Country }



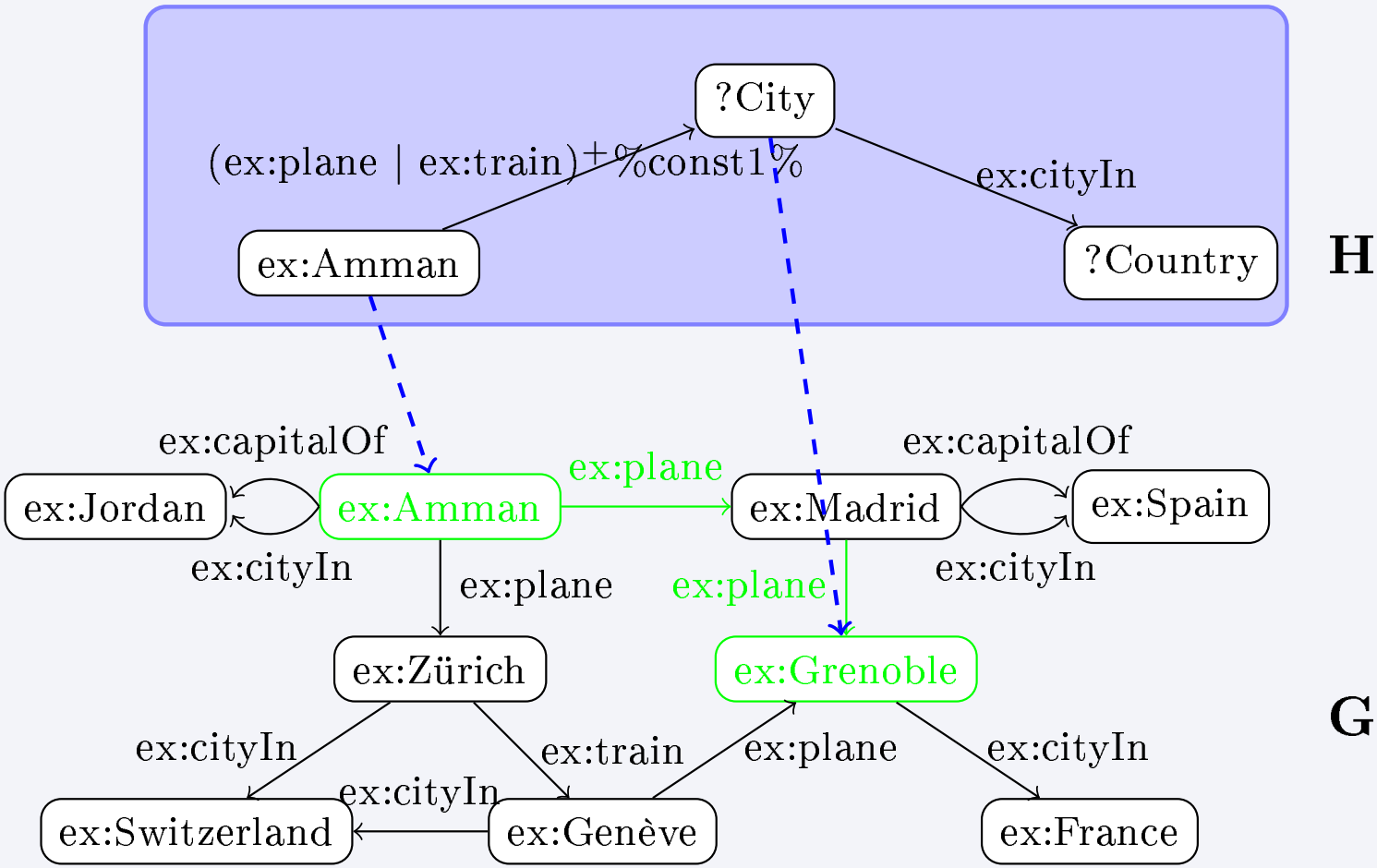
CPRDF Homomorphism: example

- const1 =]ALL ?Stop[: {?Stop ex:capitalOf ?Country }



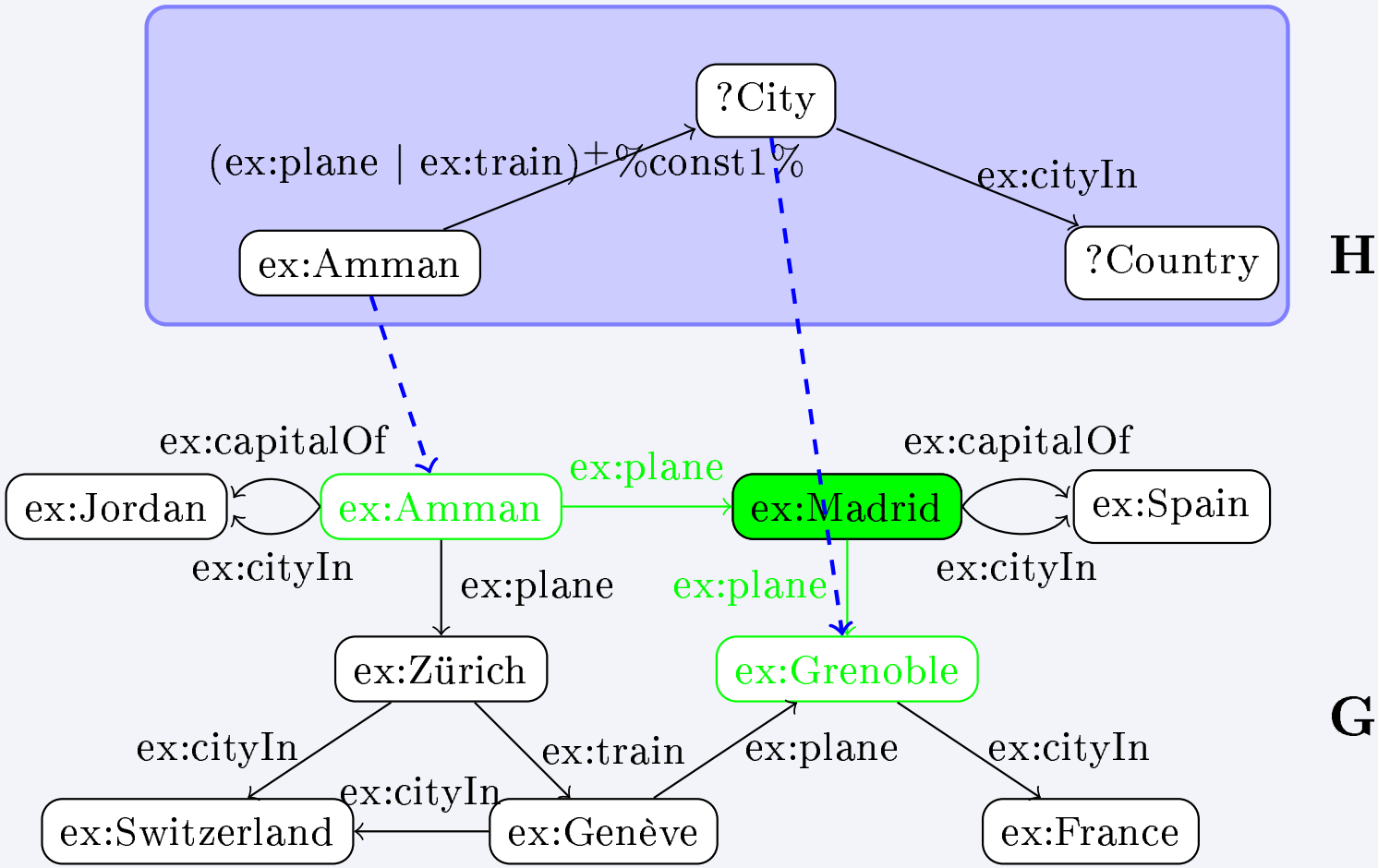
CPRDF Homomorphism: example

- const1 =]ALL ?Stop[: {?Stop ex:capitalOf ?Country }



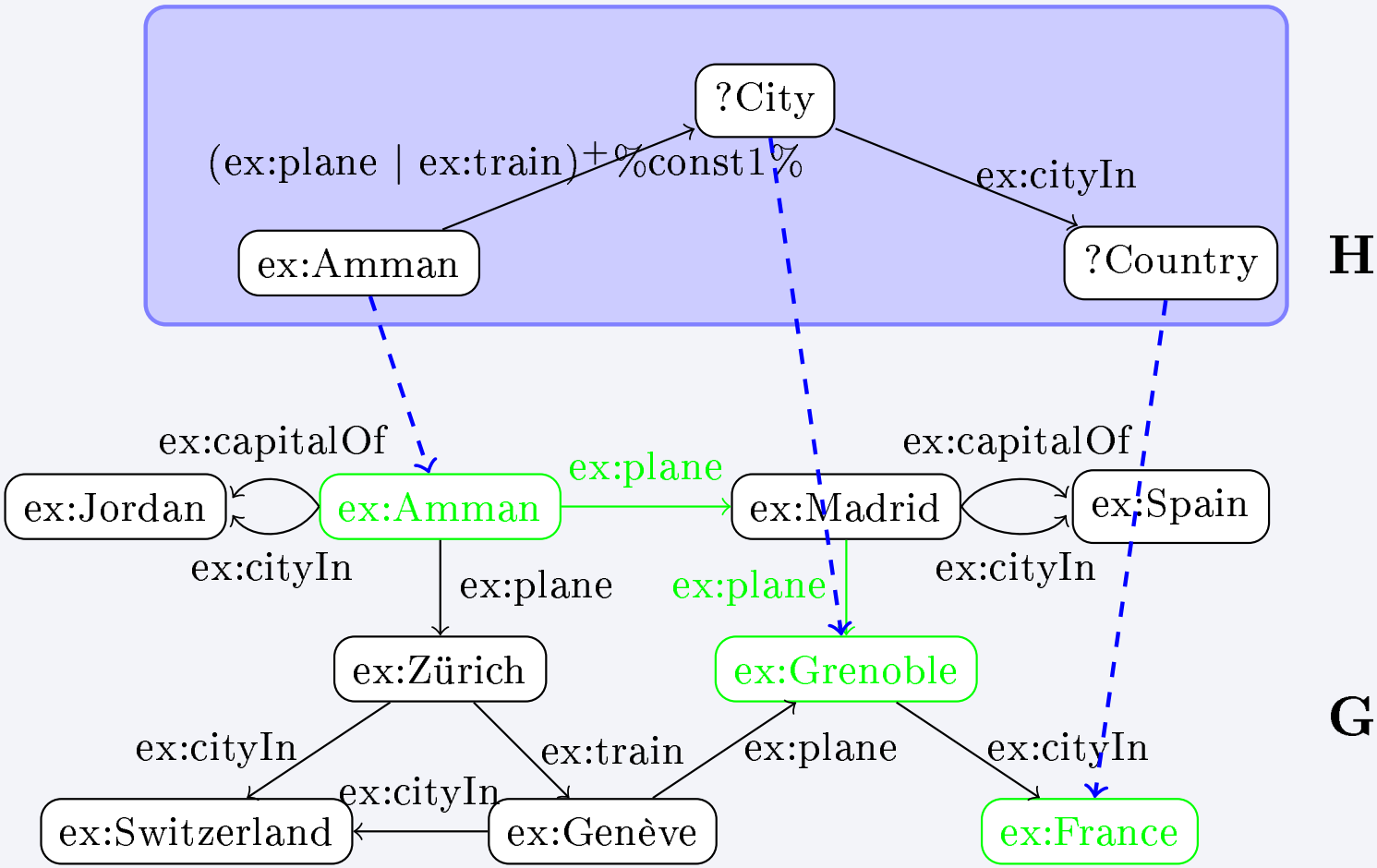
CPRDF Homomorphism: example

- const1 =]ALL ?Stop[: {?Stop ex:capitalOf ?Country }



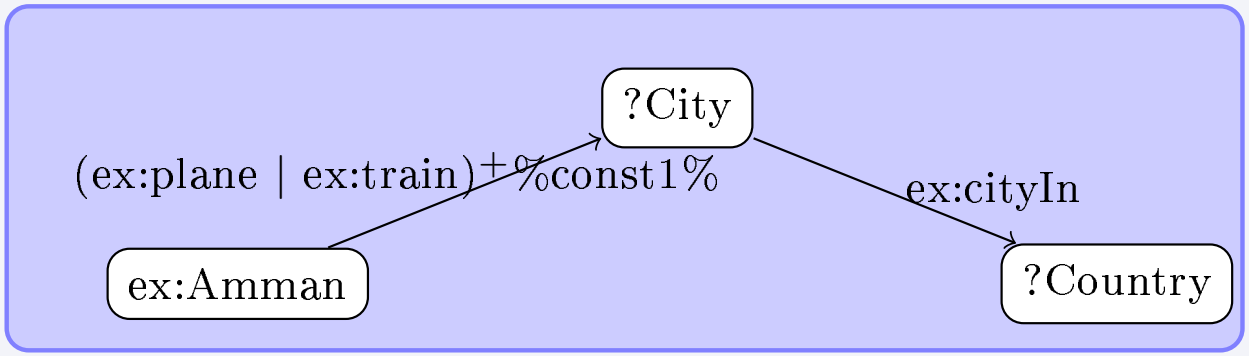
CPRDF Homomorphism: example

- const1 =]ALL ?Stop[: {?Stop ex:capitalOf ?Country }

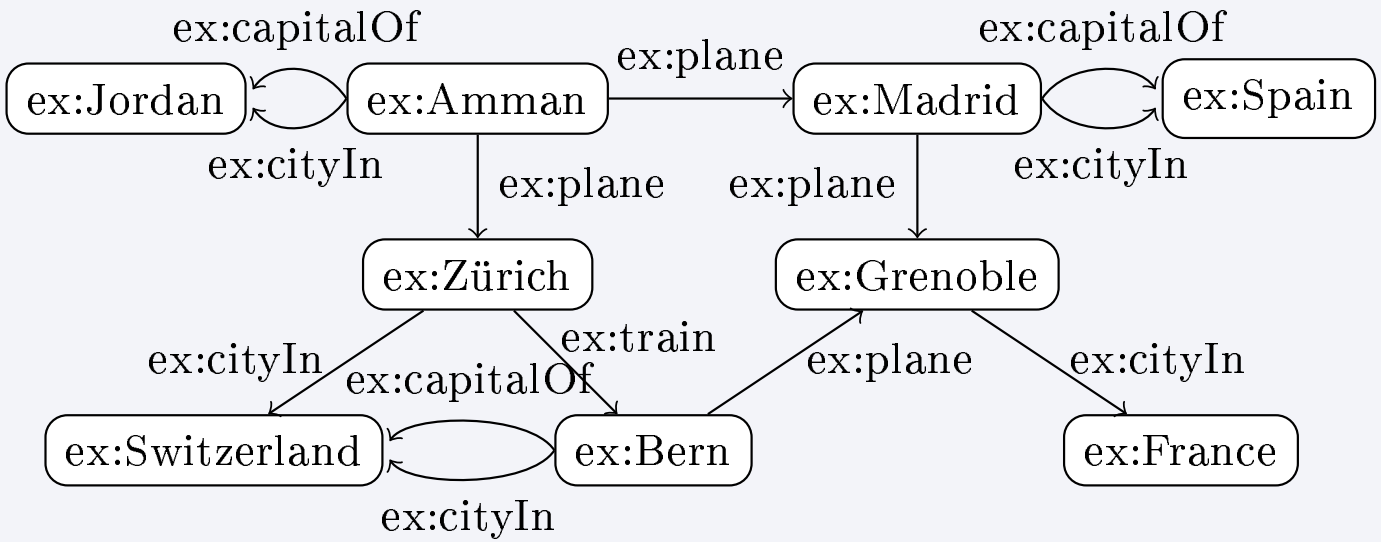


CPRDF Homomorphism: example

- const1 =]EXISTS ?Stop[: {?Stop ex:capitalOf ?Country }



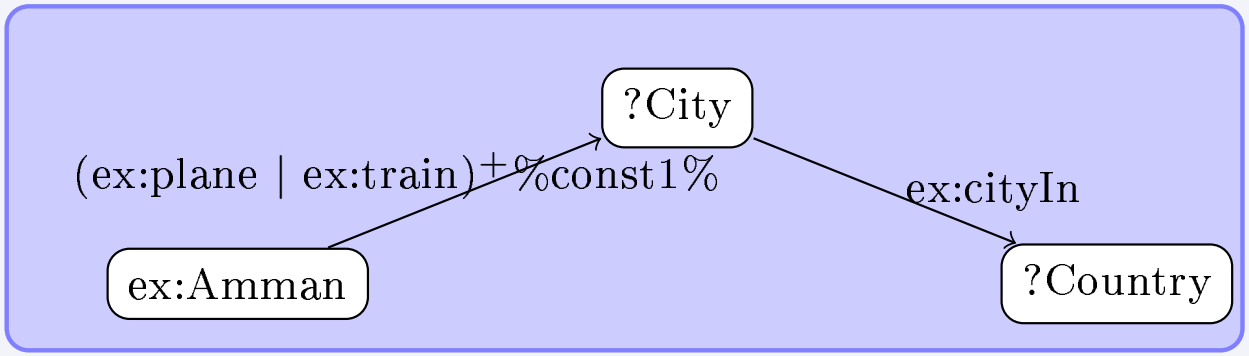
H



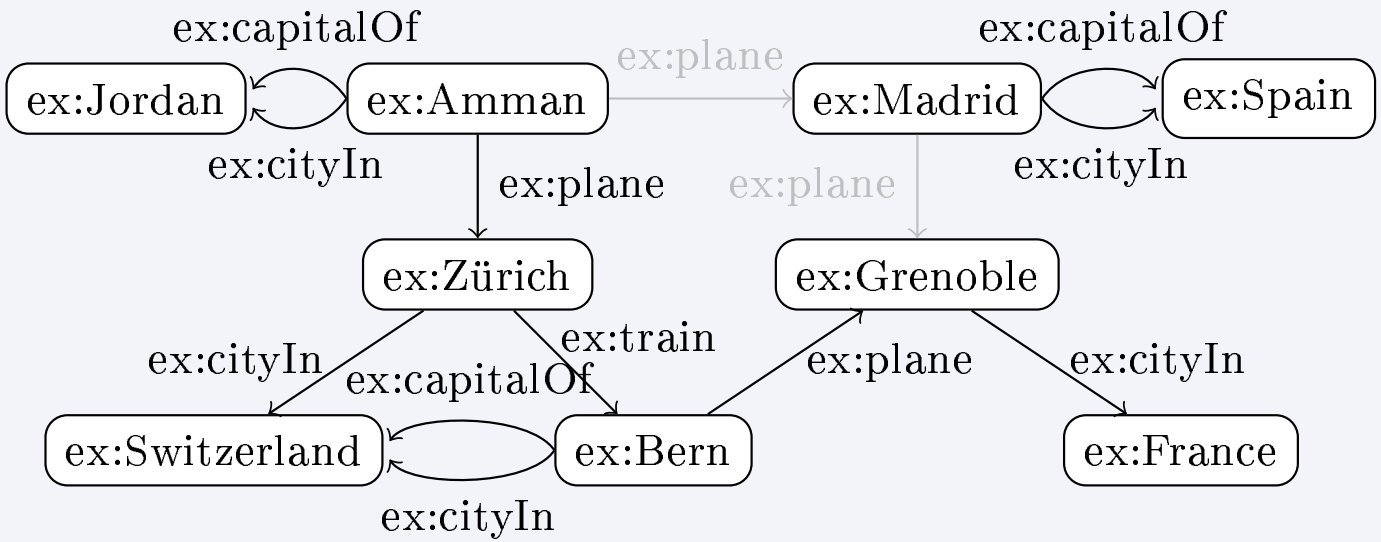
G

CPRDF Homomorphism: example

- const1 =]EXISTS ?Stop[: {?Stop ex:capitalOf ?Country }



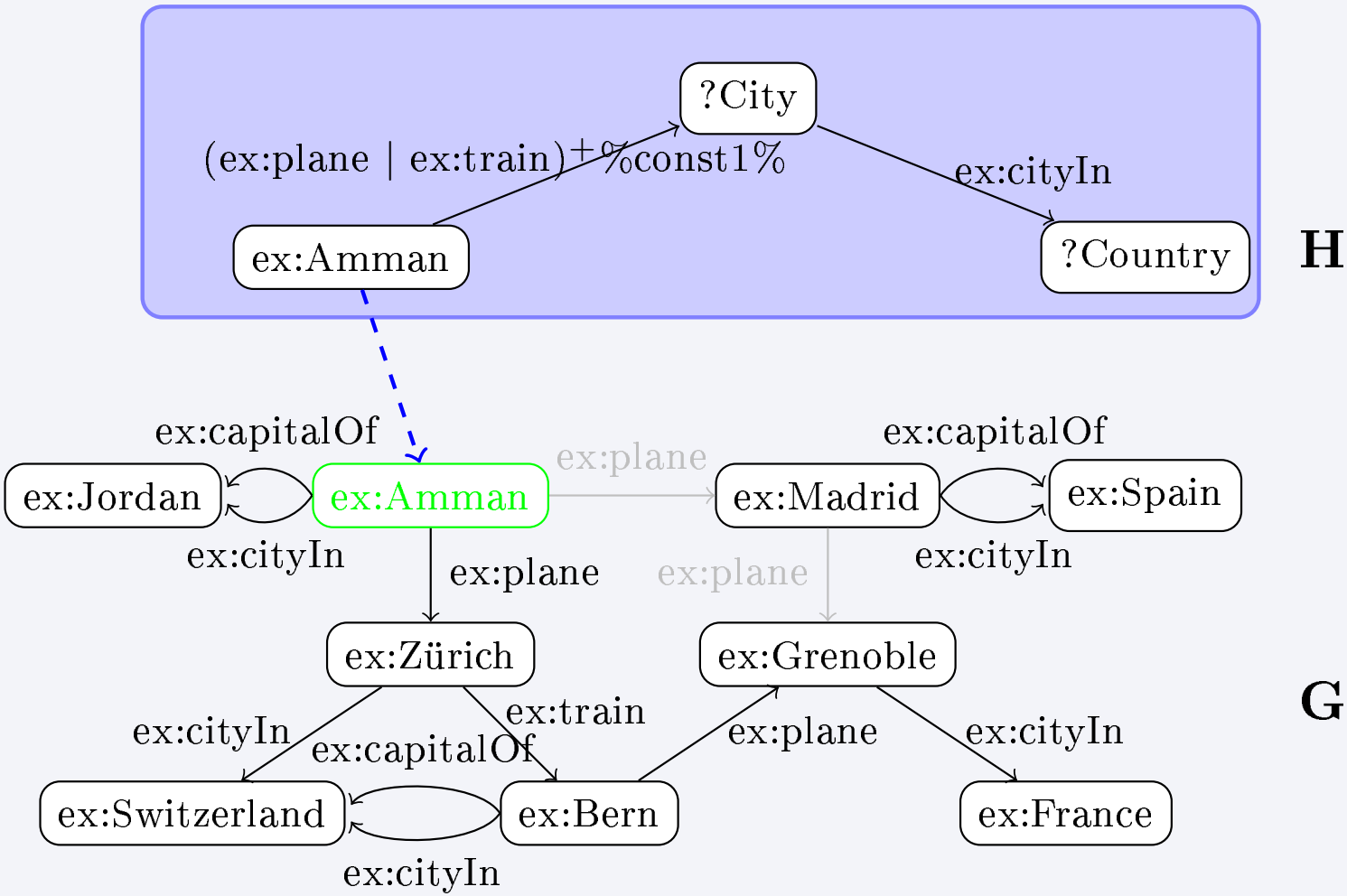
H



G

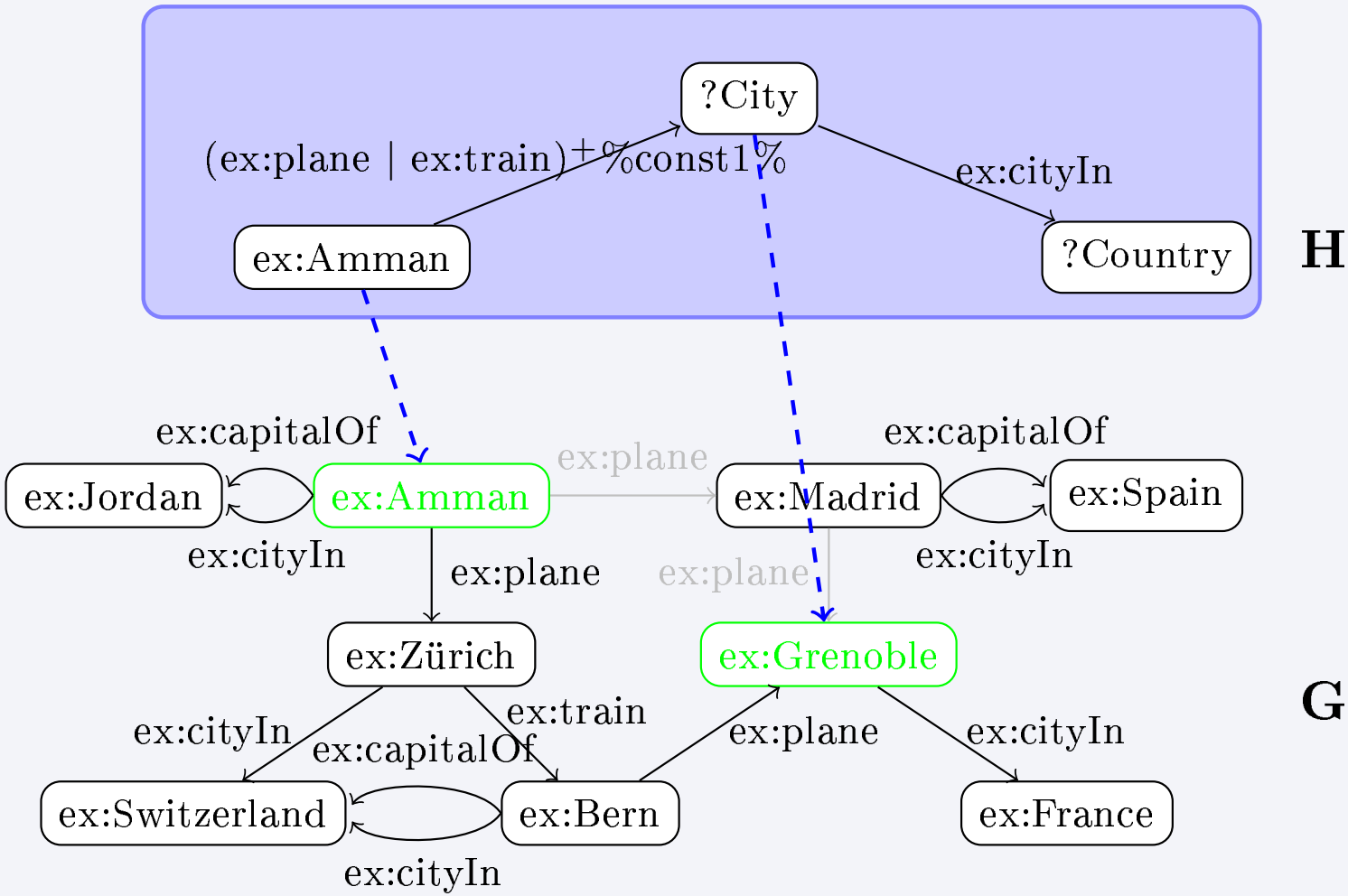
CPRDF Homomorphism: example

- const1 =]EXISTS ?Stop[: {?Stop ex:capitalOf ?Country }



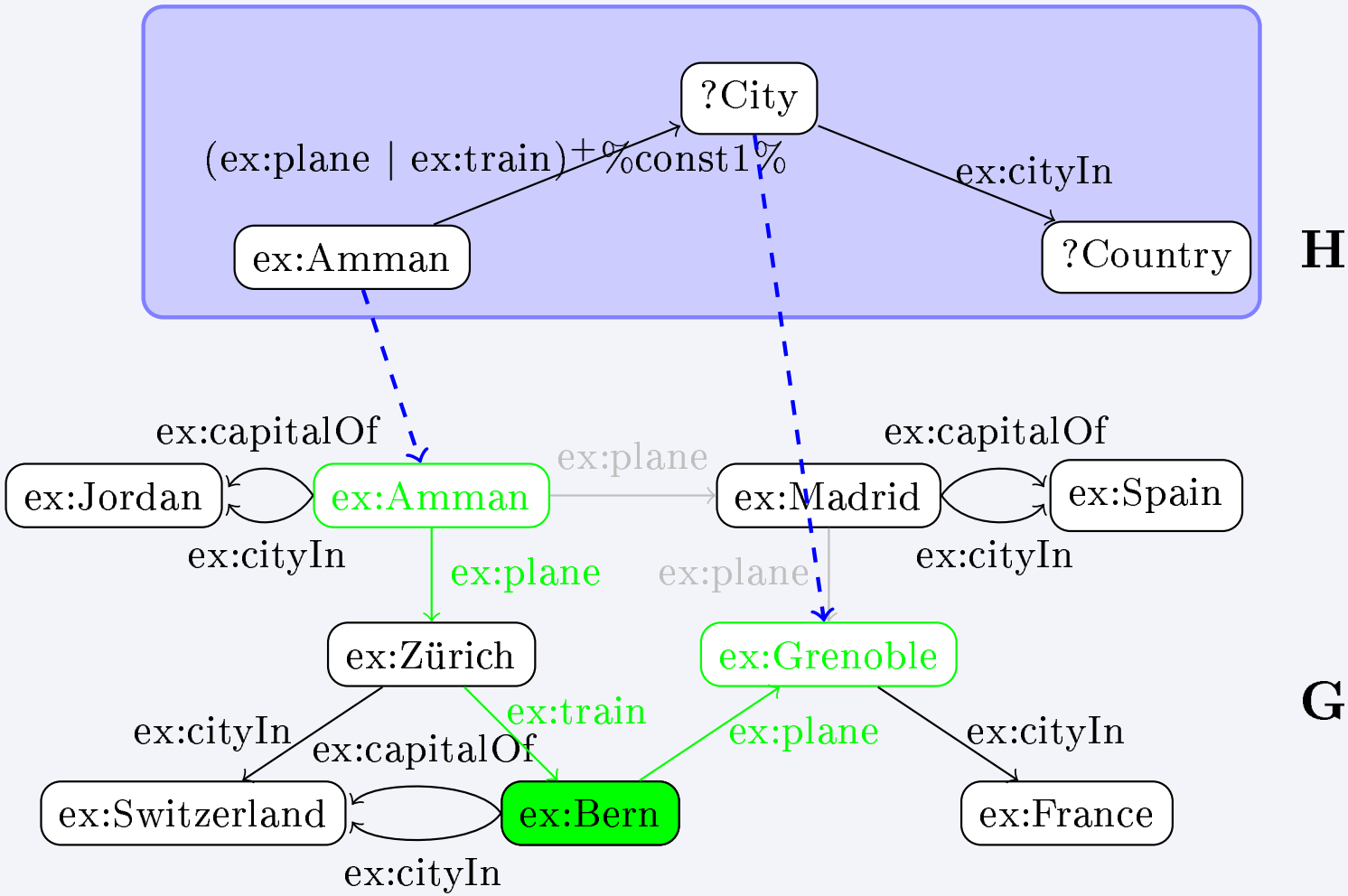
CPRDF Homomorphism: example

- const1 =]EXISTS ?Stop[: {?Stop ex:capitalOf ?Country }



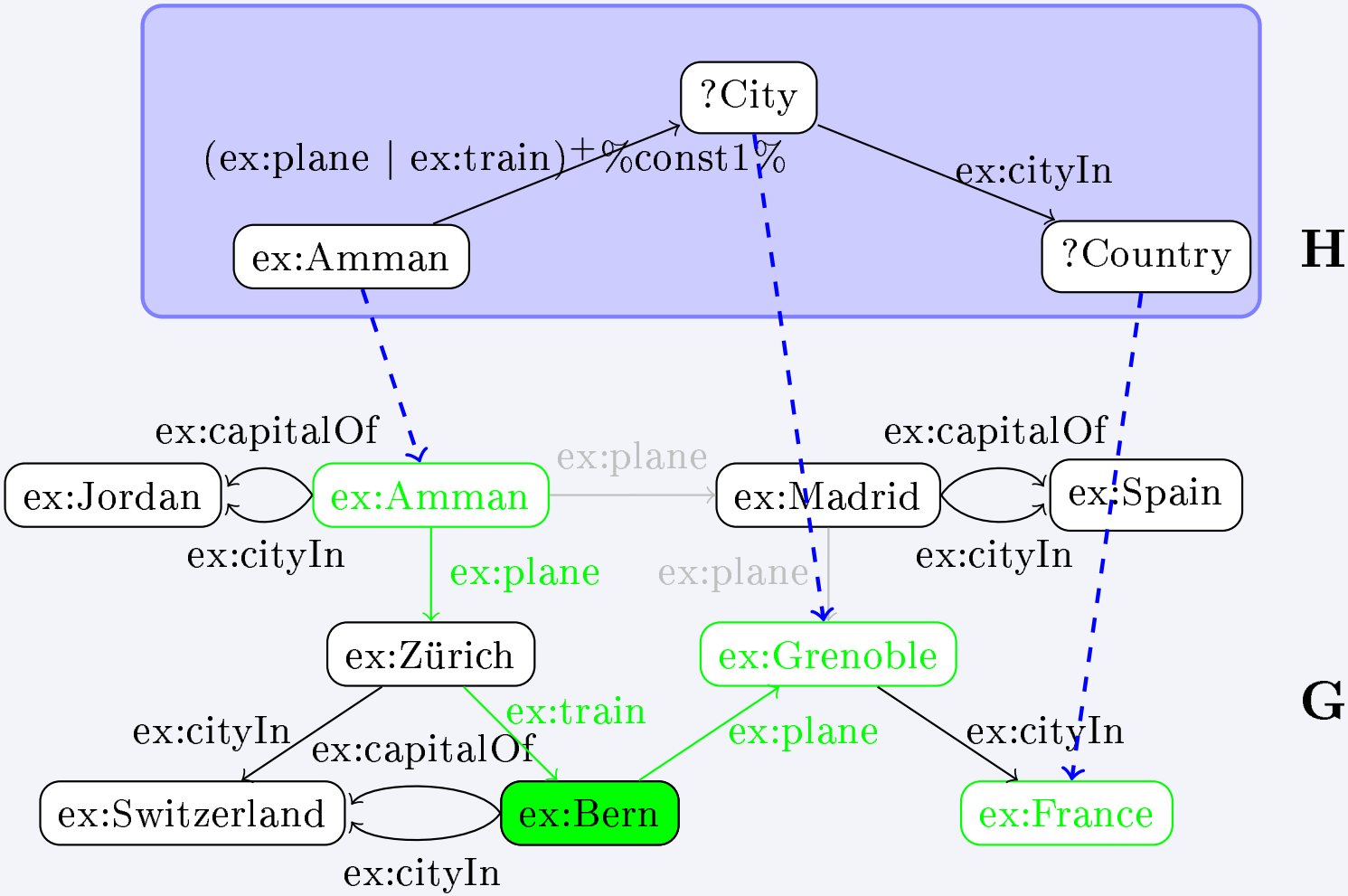
CPRDF Homomorphism: example

- const1 =]EXISTS ?Stop[: {?Stop ex:capitalOf ?Country }



CPRDF Homomorphism: example

- const1 =]EXISTS ?Stop[: {?Stop ex:capitalOf ?Country }



Outline

Graph patterns

SPARQL graph patterns

every GRDF graph is a graph pattern

Graph patterns

SPARQL graph patterns
every GRDF graph is a graph pattern
$(P_1$ `AND` $P_2)$ is a graph pattern
$(P_1$ `UNION` $P_2)$ is a graph pattern
$(P_1$ `OPT` $P_2)$ is a graph pattern
$(P_1$ `FILTER` $C)$ is a graph pattern

Graph patterns

PSPARQL graph patterns
every PRDF graph is a graph pattern
$(P_1$ `AND` $P_2)$ is a graph pattern
$(P_1$ `UNION` $P_2)$ is a graph pattern
$(P_1$ `OPT` $P_2)$ is a graph pattern
$(P_1$ `FILTER` $C)$ is a graph pattern

Graph patterns

CPSPARQL graph patterns
every CPRDF graph is a graph pattern
$(P_1$ `AND` $P_2)$ is a graph pattern
$(P_1$ `UNION` $P_2)$ is a graph pattern
$(P_1$ `OPT` $P_2)$ is a graph pattern
$(P_1$ `FILTER` $C)$ is a graph pattern

Queries

SPARQL query

`SELECT` $\vec{B}$
`FROM` u
`WHERE` P, where P is a SPARQL graph pattern.

Queries

PSPARQL query

`SELECT` $\vec{B}$
`FROM` u
`WHERE` P, where P is a PSPARQL graph pattern.

Queries

CPSPARQL query

`SELECT` $\vec{B}$
`FROM` u
`WHERE` P, where P is a CPSPARQL graph pattern.

Example

SPARQL Query:

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ex:plane ?City . }
```

Example

PSPARQL Query:

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman (ex:plane | ex:train)+ ?City . }
```

Example

CPSPARQL Query:

```
SELECT ?City
FROM <Transport>
WHERE {
      CONSTRAINT const1 ]ALL ?Stop[:
                        { ?Stop ex:capitalOf ?Country }
        ex:Amman (ex:plane | ex:train)+%const1% ?City .}
```

Answers to (CP/P)SPARQL graph patterns

- The set $\mathcal{S}(P, G)$ of answers to a SPARQL graph pattern P in a GRDF graph G is defined inductively in the following way:

P	$\mathcal{S}(P, G)$
GRDF graph	$\{\mu \mid \mu$ is a GRDF homomorphism$\}$
$(P_1$ `AND` $P_2)$	$\mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G)$
$(P_1$ `UNION` $P_2)$	$\mathcal{S}(P_1, G) \cup \mathcal{S}(P_2, G)$
$(P_1$ `OPT` $P_2)$	$(\mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G))$ $\cup(\mathcal{S}(P_1, G) \setminus \mathcal{S}(P_2, G))$
$(P_1$ `FILTER` $C)$	$\{\mu \in \mathcal{S}(P_1, G) \mid \mu(C) = \top\}$

Answers to (CP/P)SPARQL graph patterns

- The set $\mathcal{S}(P, G)$ of answers to a PSPARQL graph pattern P in a GRDF graph G is defined inductively in the following way:

P	$\mathcal{S}(P, G)$
PRDF graph	$\{\mu \mid \mu$ is a PRDF homomorphism$\}$
$(P_1$ `AND` $P_2)$	$\mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G)$
$(P_1$ `UNION` $P_2)$	$\mathcal{S}(P_1, G) \cup \mathcal{S}(P_2, G)$
$(P_1$ `OPT` $P_2)$	$(\mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G))$ $\cup(\mathcal{S}(P_1, G) \setminus \mathcal{S}(P_2, G))$
$(P_1$ `FILTER` $C)$	$\{\mu \in \mathcal{S}(P_1, G) \mid \mu(C) = \top\}$

Answers to (CP/P)SPARQL graph patterns

- The set $\mathcal{S}(P, G)$ of answers to a CPSPARQL graph pattern P in a GRDF graph G is defined inductively in the following way:

P	$\mathcal{S}(P, G)$
CPRDF graph	$\{\mu \mid \mu$ is a CPRDF homomorphism$\}$
$(P_1$ `AND` $P_2)$	$\mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G)$
$(P_1$ `UNION` $P_2)$	$\mathcal{S}(P_1, G) \cup \mathcal{S}(P_2, G)$
$(P_1$ `OPT` $P_2)$	$(\mathcal{S}(P_1, G) \bowtie \mathcal{S}(P_2, G))$ $\cup(\mathcal{S}(P_1, G) \setminus \mathcal{S}(P_2, G))$
$(P_1$ `FILTER` $C)$	$\{\mu \in \mathcal{S}(P_1, G) \mid \mu(C) = \top\}$

Answers to queries

- If $Q = \texttt{SELECT}\ \vec{B}\ \texttt{FROM}\ u\ \texttt{WHERE}\ P$ be a SPARQL/(C)PSPARQL query

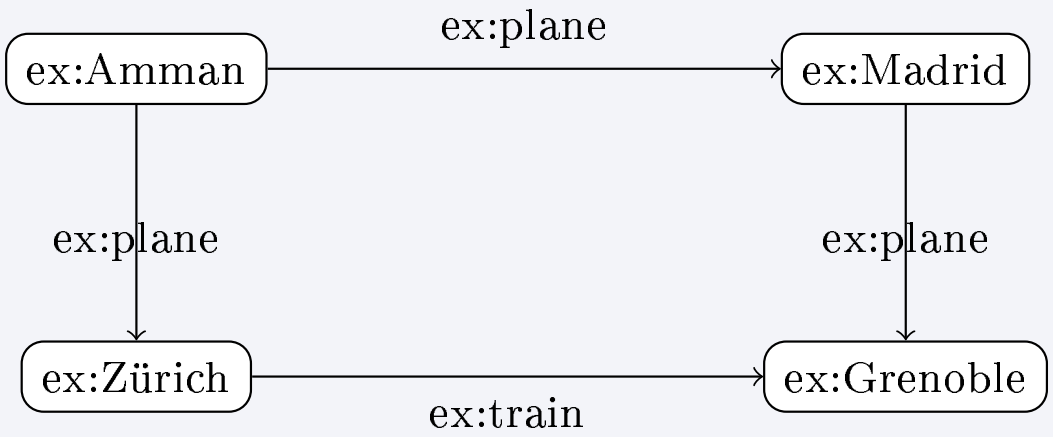
Answers to queries

- If $Q = \texttt{SELECT}\ \vec{B}\ \texttt{FROM}\ u\ \texttt{WHERE}\ P$ be a SPARQL/(C)PSPARQL query
- The answers to Q are: $\{\mu(\vec{B}) \mid \mu \in \mathcal{S}(P, G)\}$

Outline

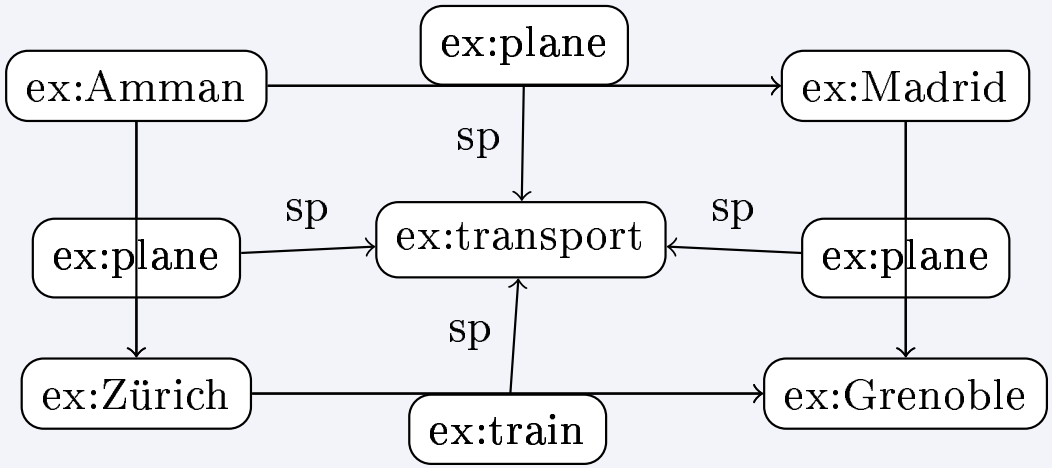
RDFS closure

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ex:transport ?City . }
```



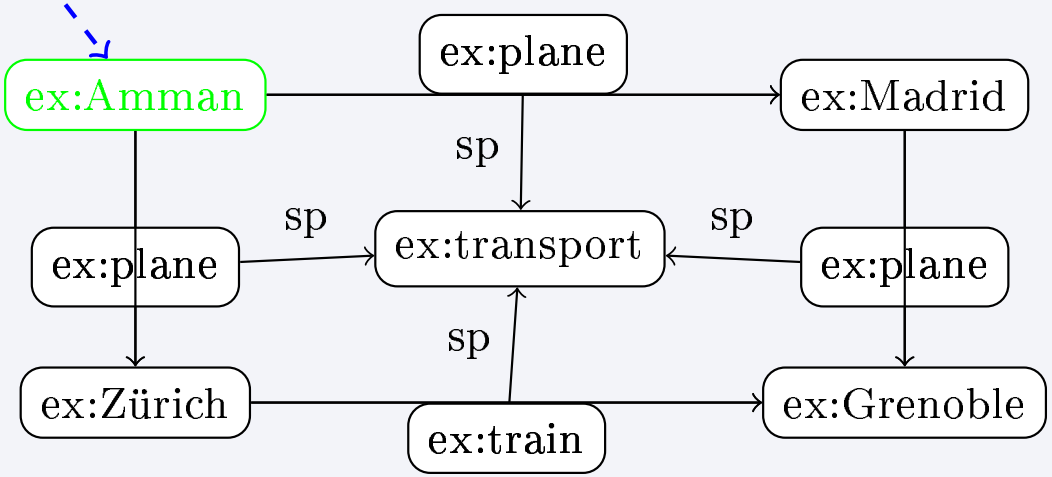
RDFS closure

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ex:transport ?City . }
```



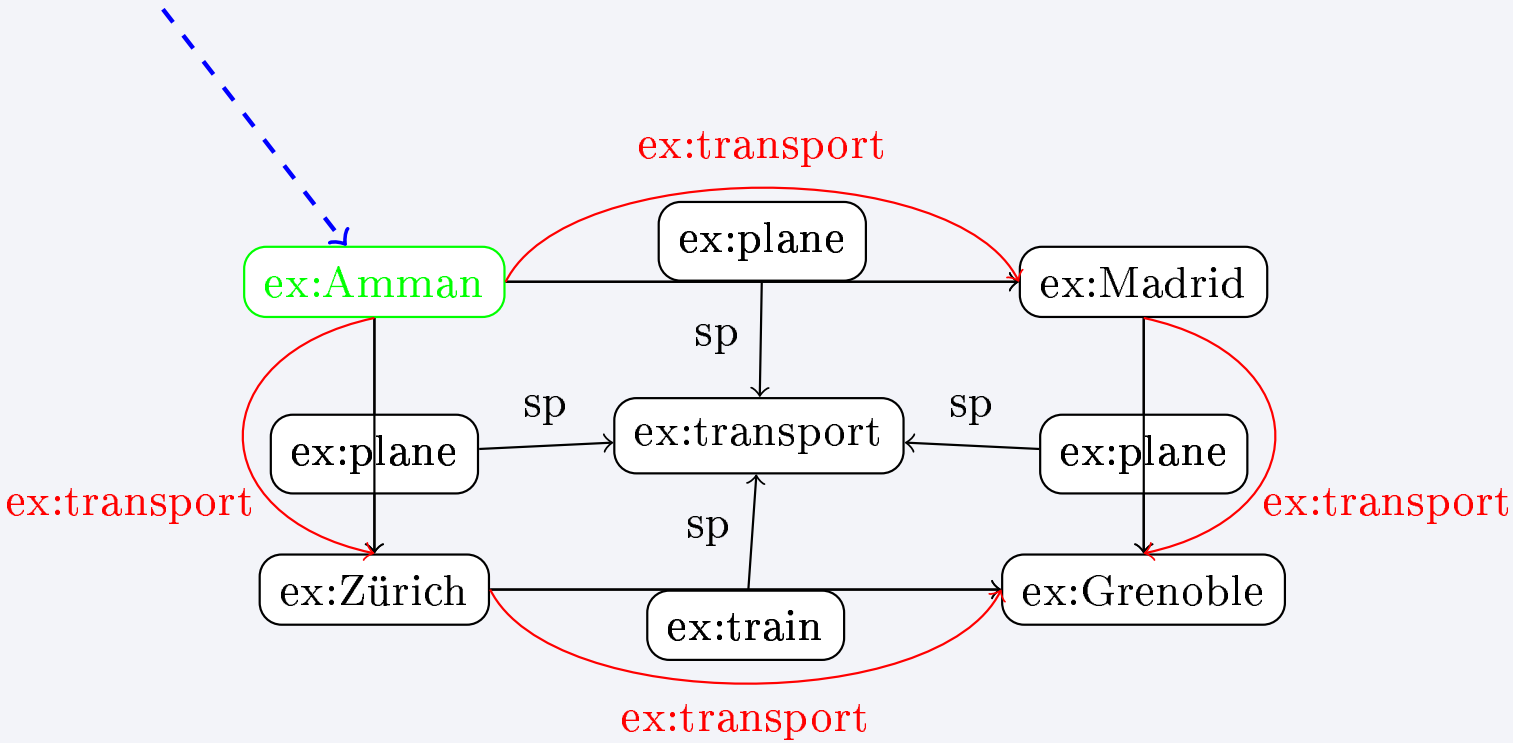
RDFS closure

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ex:transport ?City . }
```



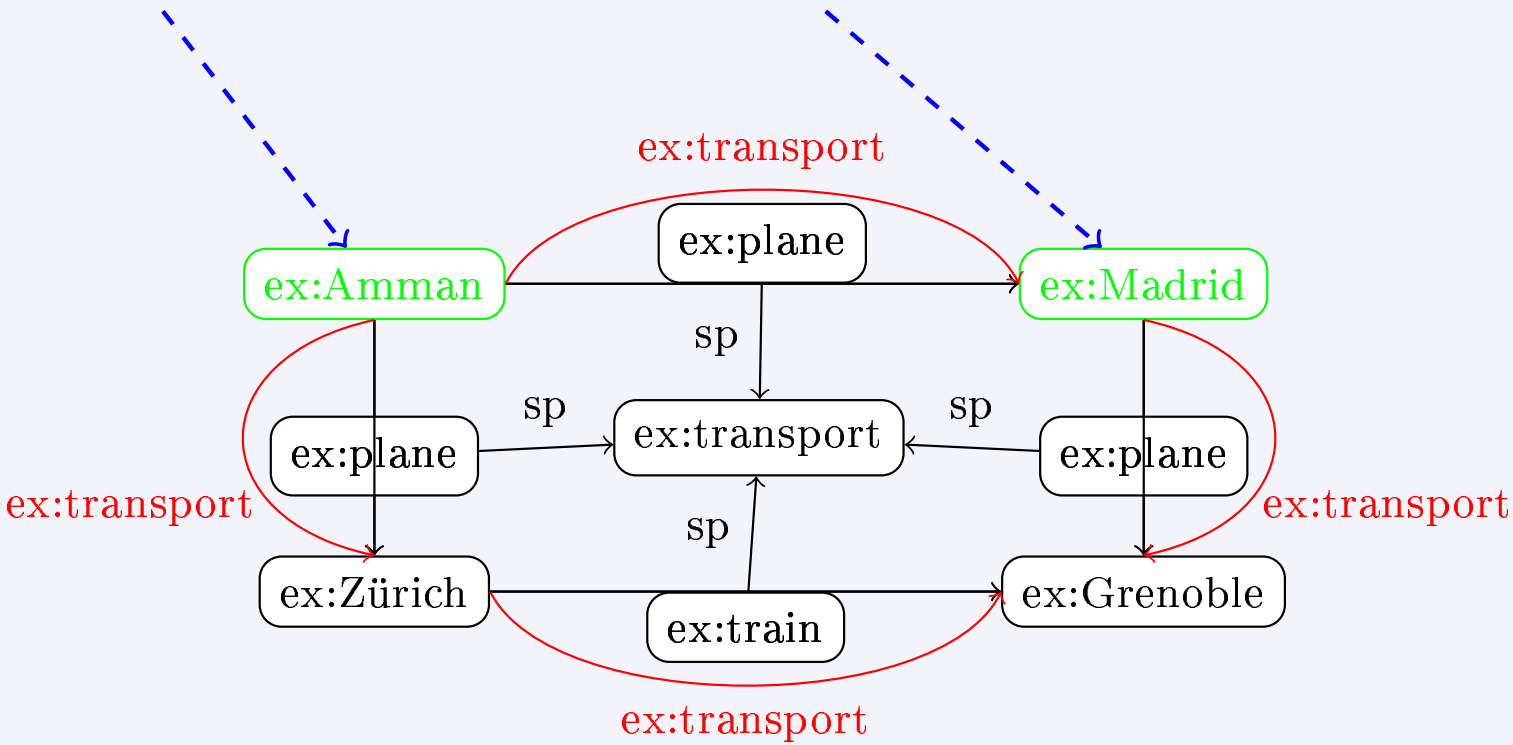
RDFS closure

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ex:transport ?City . }
```



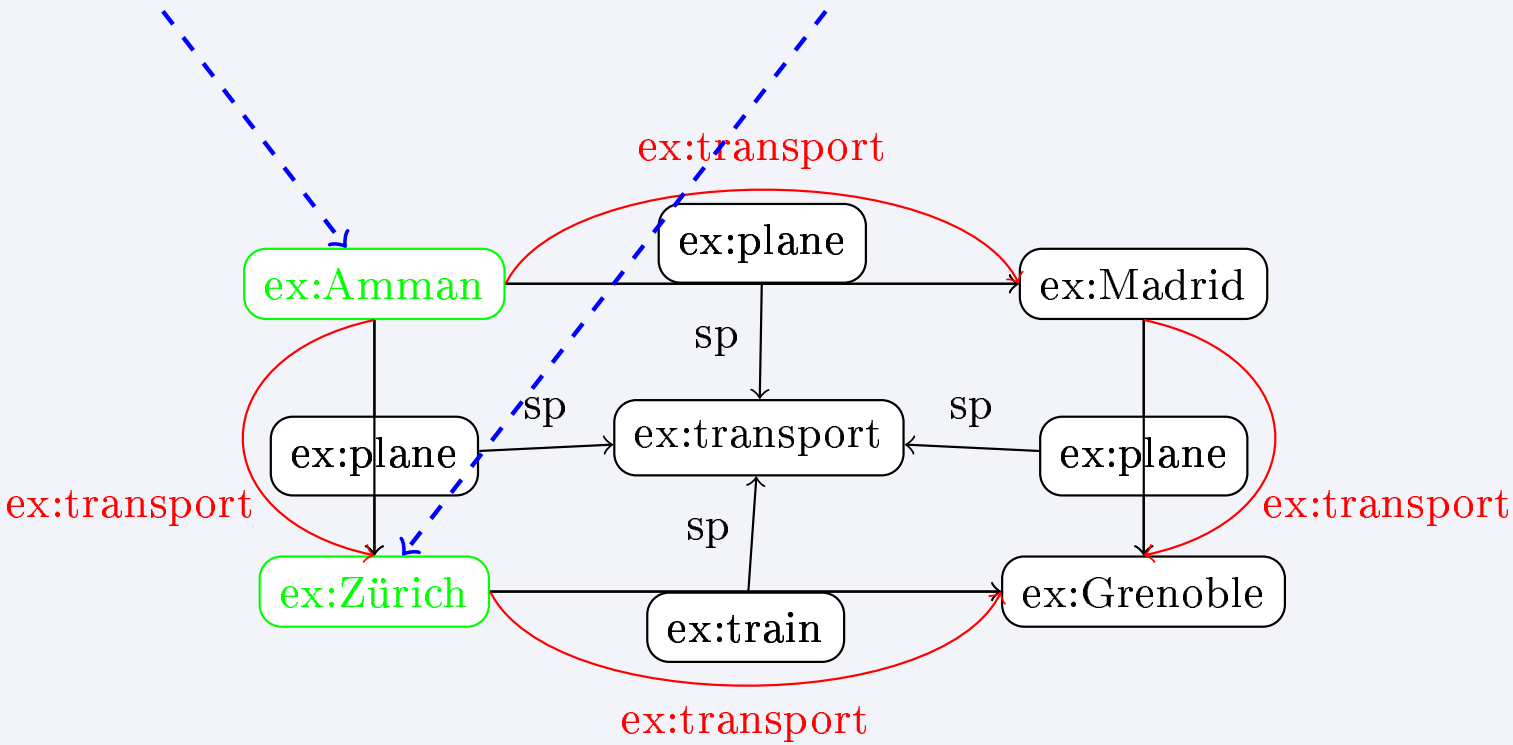
RDFS closure

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ex:transport ?City . }
```



RDFS closure

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ex:transport ?City . }
```



From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$
Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$\tau(s, \mathtt{type}, o) \quad = \quad \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\}$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$$\tau(s, \mathtt{type}, o) = \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\}$$

$$\frac{\langle s,\mathtt{type},z\rangle,\langle z,\mathtt{sc}^*,o\rangle}{\langle s,\mathtt{type},o\rangle} = \frac{\langle s,\mathtt{type}\cdot\mathtt{sc}^*,o\rangle}{\langle s,\mathtt{type},o\rangle}$$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$$\begin{array}{lll} \tau(s, \mathtt{type}, o) & = & \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\} \\ & \mathtt{UNION} & \{\langle s, ?p_1, ?y\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{dom} \cdot \mathtt{sc}^*, o\rangle\} \end{array}$$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$$\tau(s, \mathtt{type}, o) = \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\}$$

$$\mathtt{UNION}\ \{\langle s, ?p_1, ?y\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{dom} \cdot \mathtt{sc}^*, o\rangle\}$$

$$\frac{\langle s,?p_1,?y\rangle, \langle ?p_1, \mathtt{dom}, o\rangle}{\langle s, \mathtt{type}, o\rangle}$$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$$\tau(s, \mathtt{type}, o) = \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\}$$
$$\mathtt{UNION}\quad \{\langle s, ?p_1, ?y\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{dom} \cdot \mathtt{sc}^*, o\rangle\}$$
$$\frac{\langle s,?p_1,?y\rangle,\langle ?p_1,\mathtt{dom}\cdot\mathtt{sc}^*,o\rangle}{\langle s,\mathtt{type},o\rangle}$$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$$\tau(s, \mathtt{type}, o) = \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\}$$
$$\mathtt{UNION} \quad \{\langle s, ?p_1, ?y\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{dom} \cdot \mathtt{sc}^*, o\rangle\}$$

$$\frac{\langle s,?p_1,?y\rangle,\langle ?p_1,\mathtt{sp}^*,?p_2\rangle,\langle ?p_2,\mathtt{dom}\cdot\mathtt{sc}^*,o\rangle}{\langle s,\mathtt{type},o\rangle}$$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$$\begin{array}{lll} \tau(s, \mathtt{type}, o) & = & \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\} \\ & \texttt{UNION} & \{\langle s, ?p_1, ?y\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{dom} \cdot \mathtt{sc}^*, o\rangle\} \\ & \texttt{UNION} & \{\langle ?y, ?p_1, s\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{range} \cdot \mathtt{sc}^*, o\rangle\} \end{array}$$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$$\begin{array}{lll} \tau(s, \mathtt{type}, o) & = & \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\} \\ & \texttt{UNION} & \{\langle s, ?p_1, ?y\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{dom} \cdot \mathtt{sc}^*, o\rangle\} \\ & \texttt{UNION} & \{\langle ?y, ?p_1, s\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{range} \cdot \mathtt{sc}^*, o\rangle\} \end{array}$$

Domain rule: $\tau(\langle s, \mathtt{dom}, o\rangle) = \langle s, \mathtt{dom}, o\rangle$

Range rule: $\tau(\langle s, \mathtt{range}, o\rangle) = \langle s, \mathtt{range}, o\rangle$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$

Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$

Typing rule:

$$\begin{array}{lll} \tau(s, \mathtt{type}, o) & = & \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\} \\ & \texttt{UNION} & \{\langle s, ?p_1, ?y\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{dom} \cdot \mathtt{sc}^*, o\rangle\} \\ & \texttt{UNION} & \{\langle ?y, ?p_1, s\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{range} \cdot \mathtt{sc}^*, o\rangle\} \end{array}$$

Domain rule: $\tau(\langle s, \mathtt{dom}, o\rangle) = \langle s, \mathtt{dom}, o\rangle$

Range rule: $\tau(\langle s, \mathtt{range}, o\rangle) = \langle s, \mathtt{range}, o\rangle$

Any property: $\tau(\langle s, p, o\rangle) = \{\langle s, ?x, o\rangle, \langle ?x, \mathtt{sp}^*, p\rangle\}$

From SPARQL/RDFS to PSPARQL/RDF

SubClass: $\tau(\langle s, \mathtt{sc}, o\rangle) = \{\langle s, \mathtt{sc}^+, o\rangle\}$
Subproperty: $\tau(\langle s, \mathtt{sp}, o\rangle) = \{\langle s, \mathtt{sp}^+, o\rangle\}$
Typing rule:

$$\begin{array}{lll} \tau(s, \mathtt{type}, o) & = & \{\langle s, \mathtt{type} \cdot \mathtt{sc}^*, o\rangle\} \\ & \mathtt{UNION} & \{\langle s, ?p_1, ?y\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{dom} \cdot \mathtt{sc}^*, o\rangle\} \\ & \mathtt{UNION} & \{\langle ?y, ?p_1, s\rangle, \langle ?p_1, \mathtt{sp}^*, ?p_2\rangle, \langle ?p_2, \mathtt{range} \cdot \mathtt{sc}^*, o\rangle\} \end{array}$$

Domain rule: $\tau(\langle s, \mathtt{dom}, o\rangle) = \langle s, \mathtt{dom}, o\rangle$
Range rule: $\tau(\langle s, \mathtt{range}, o\rangle) = \langle s, \mathtt{range}, o\rangle$
Any property: $\tau(\langle s, p, o\rangle) = \{\langle s, ?x, o\rangle, \langle ?x, \mathtt{sp}^*, p\rangle\}$

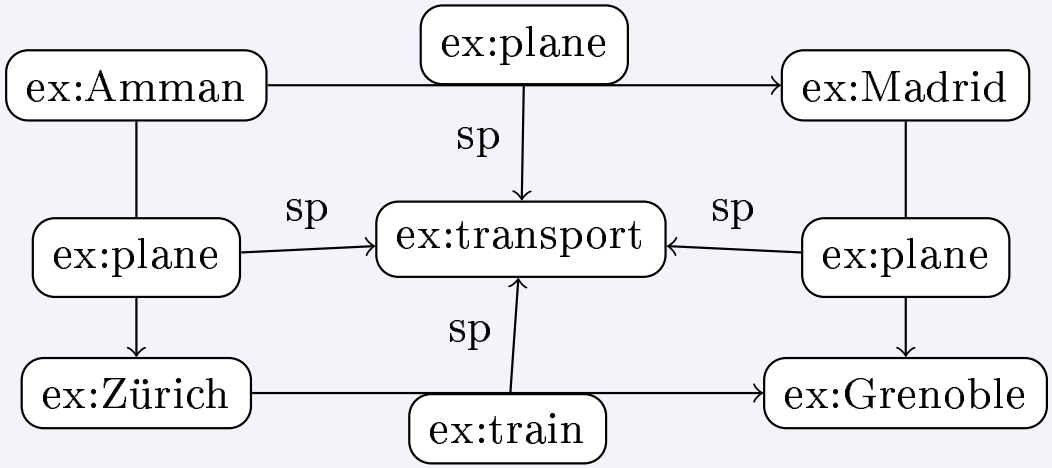
$$\rho\text{df [munoz2007]} = \{\mathtt{sc}, \mathtt{sp}, \mathtt{type}, \mathtt{dom}, \mathtt{range}\}$$

Theorem

Let G and P be two ρdf graphs, then $Eval(P, \hat{G}, \Omega)$ is equivalent to $Eval(\tau(P), G, \Omega)$.

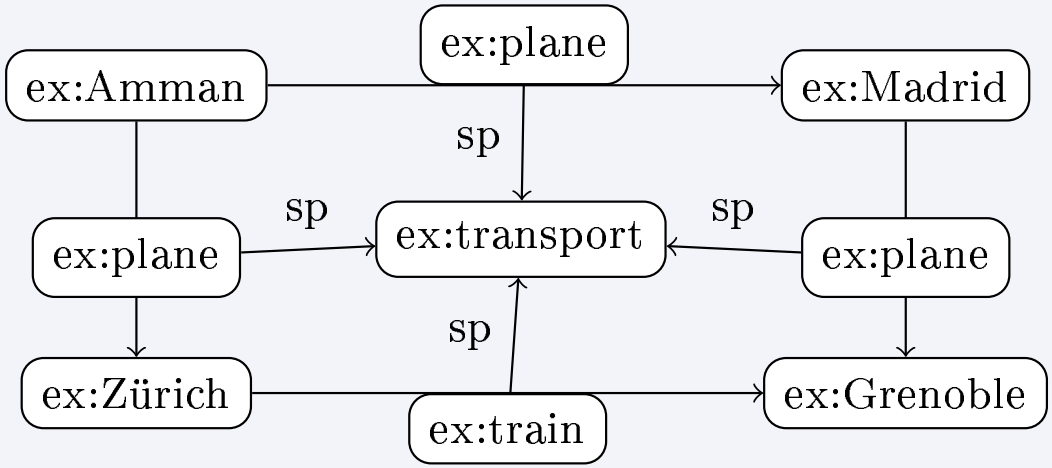
From SPARQL/RDFS to PSPARQL/RDF: example

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ex:transport ?City . }
```



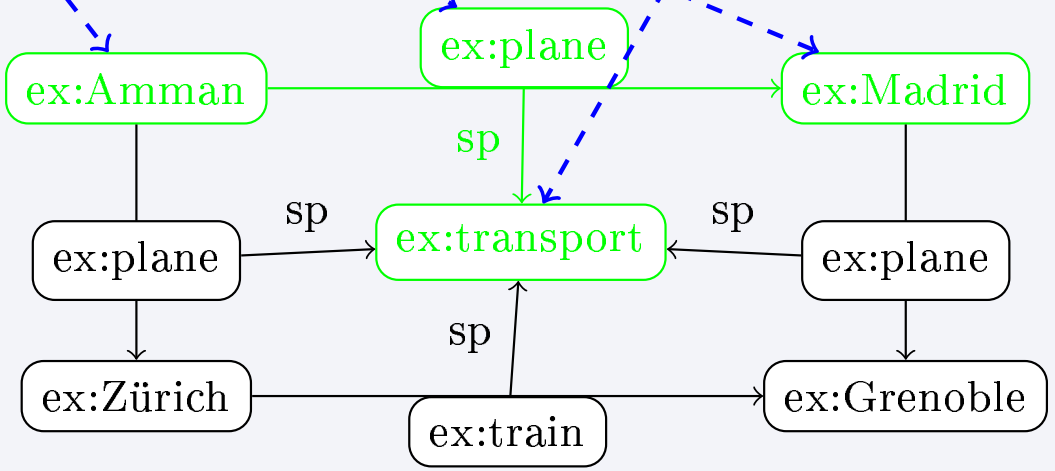
From SPARQL/RDFS to PSPARQL/RDF: example

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ?P ?City. ?P sp* ex:transport . }
```



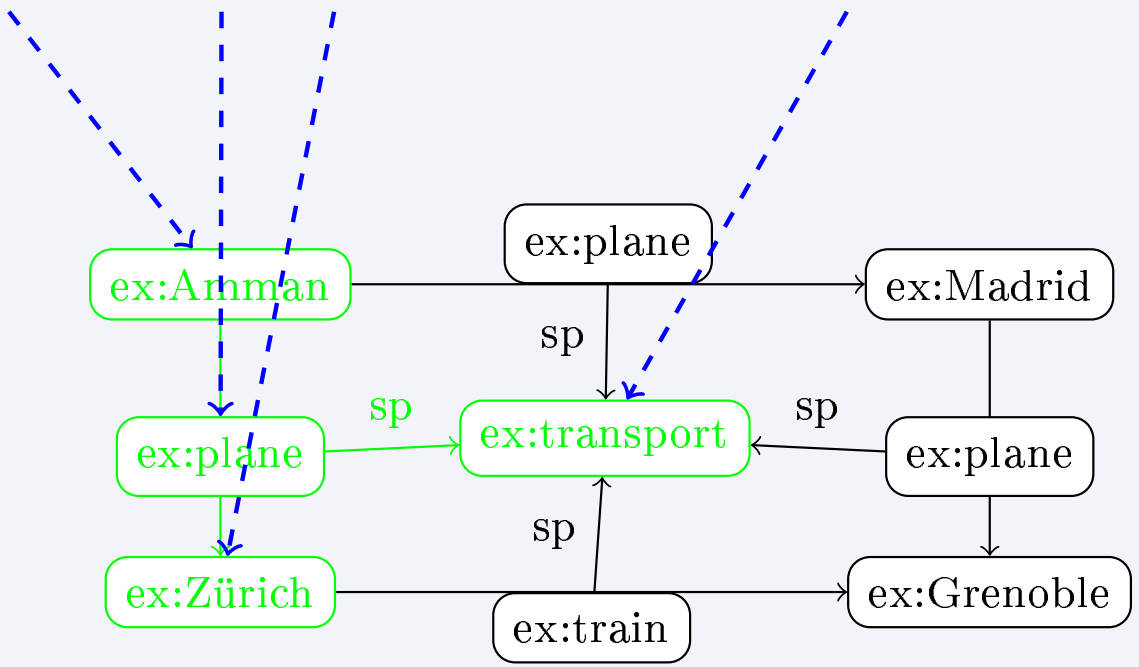
From SPARQL/RDFS to PSPARQL/RDF: example

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ?P ?City. ?P sp* ex:transport . }
```



From SPARQL/RDFS to PSPARQL/RDF: example

```
SELECT ?City
FROM <Transport>
WHERE { ex:Amman ?P ?City. ?P sp* ex:transport . }
```

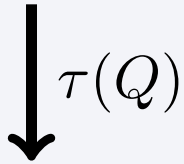


From PSPARQL/RDFS to CPSPARQL/RDF

```
SELECT ?City
WHERE { ex:Amman ex:transport+ ?City . }
```

From PSPARQL/RDFS to CPSPARQL/RDF

```
SELECT ?City
WHERE { ex:Amman ex:transport+ ?City . }
```



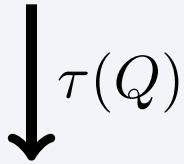

```
SELECT ?City
WHERE {
CONSTRAINT const1 [EDGE ?P]: { ?P sp* ex:transport . }
ex:Amman (# % const1 %)+ ?City .
}
```

From PSPARQL/RDFS to CPSPARQL/RDF

```
SELECT ?City
WHERE { ex:Amman (!ex:train)+ ?City . }
```

From PSPARQL/RDFS to CPSPARQL/RDF

```
SELECT ?City
WHERE { ex:Amman (!ex:train)+ ?City . }
```




```
SELECT ?City
WHERE {
CONSTRAINT const1 [EDGE ?P]: { ?P (!sp)* ex:train . }
ex:Amman (# % const1 %)+ ?City .
}
```

Outline

Summary

- We have defined two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions

Summary

- We have defined two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We have defined inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.

Summary

- We have defined two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We have defined inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.
- We have proved the soundness and completeness of (C)PRDF homomorphism ((C)PRDF into RDF) w.r.t $\models_{(C)PRDF}$.

Summary

- We have defined two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We have defined inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.
- We have proved the soundness and completeness of (C)PRDF homomorphism ((C)PRDF into RDF) w.r.t $\models_{(C)PRDF}$.
- We have proposed:
  - PSPARQL = SPARQL + PRDF
  - CPSPARQL = SPARQL + CPRDF

Summary

- We have defined two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We have defined inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.
- We have proved the soundness and completeness of (C)PRDF homomorphism ((C)PRDF into RDF) w.r.t $\models_{(C)PRDF}$.
- We have proposed:
  - PSPARQL = SPARQL + PRDF
  - CPSPARQL = SPARQL + CPRDF
- We have provided a rewriting method for evaluating SPARQL or (C)PSPARQL queries over RDF(S).

Summary

- We have defined two extensions to RDF:
  - PRDF = RDF + regular expressions
  - CPRDF = PRDF + constrained regular expressions
- We have defined inference mechanisms ((C)PRDF homomorphism) for querying RDF graphs.
- We have proved the soundness and completeness of (C)PRDF homomorphism ((C)PRDF into RDF) w.r.t $\models_{(C)PRDF}$.
- We have proposed:
  - PSPARQL = SPARQL + PRDF
  - CPSPARQL = SPARQL + CPRDF
- We have provided a rewriting method for evaluating SPARQL or (C)PSPARQL queries over RDF(S).
- We have implemented (C)PSPARQL (`psparql.inrialpes.fr`).

Further work

- Processing alignment with query languages

Further work

- Processing alignment with query languages
- Consider advanced database optimization and graph indexing techniques:
  - clustering graphs to minimize external path length [Diwan et al., 1996].
  - Fast Practical Indexing and Query of Very Large Graphs [Tribl et al., 2007] .

Further work

- Processing alignment with query languages
- Consider advanced database optimization and graph indexing techniques:
  - clustering graphs to minimize external path length [Diwan et al., 1996].
  - Fast Practical Indexing and Query of Very Large Graphs [Tribl et al., 2007] .
- Using SPARQL for document generation and adaptation [alkhateeb et al., 2008]

Thanks

Questions and Comments?