Static Analysis and Optimization of Semantic Web Queries

Andrés Letelier Jorge Pérez Reinhard Pichler Sebastian Skritek

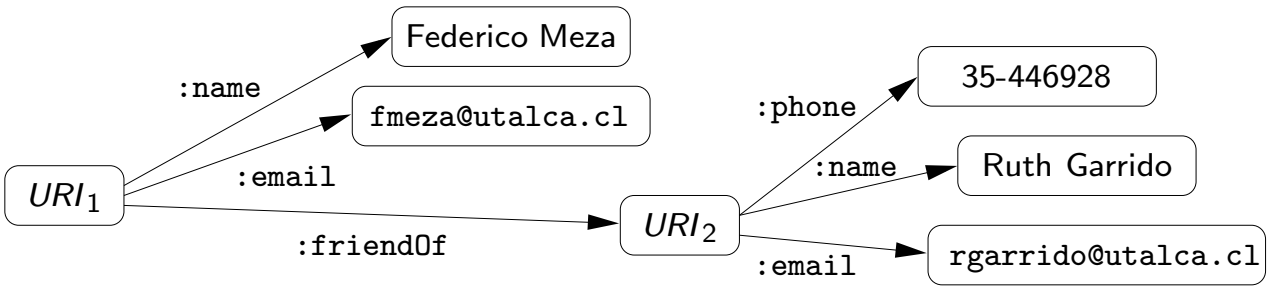
PUC Chile Universidad de Chile TU Vienna

RDF data model for the Semantic Web

SPARQL query language for RDF (W3C initiatives)

RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

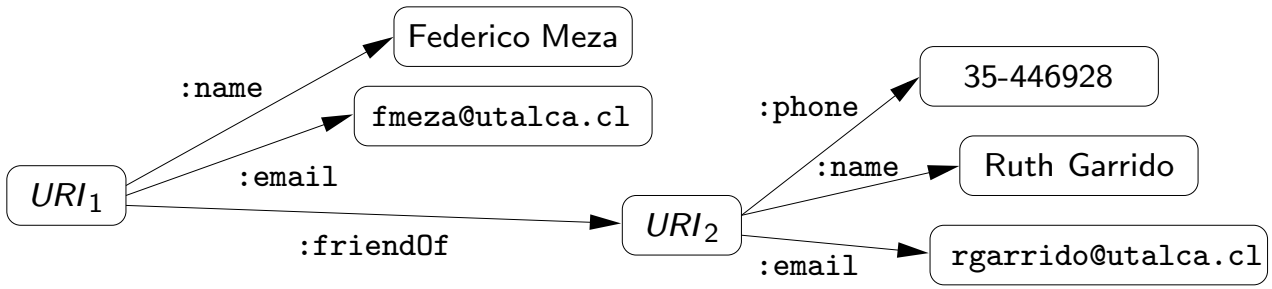
RDF Graph:



RDF-triples: (URI_2, `:email`, `rgarrido@utalca.cl`)

RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

RDF Graph:



RDF-triples: (URI_2, `:email`, `rgarrido@utalca.cl`)

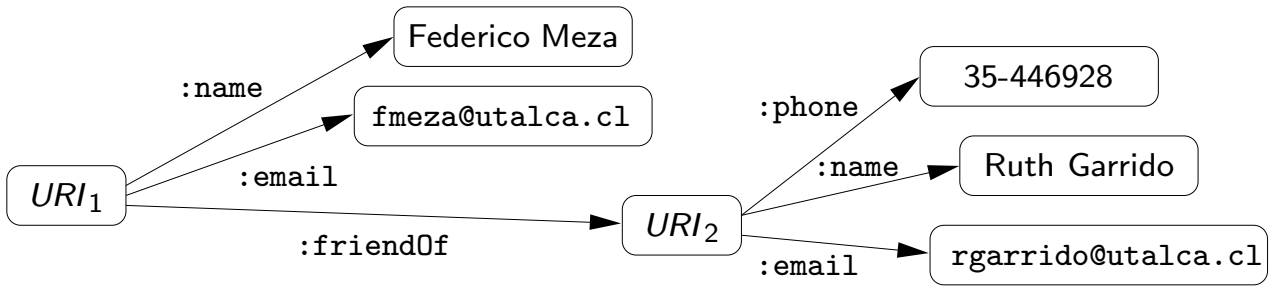
SPARQL Query:

```
SELECT ?N ?E
WHERE
{
  ?X :name ?N .
  ?X :email ?E .

}
```

RDF data model for the Semantic Web
SPARQL query language for RDF (W3C initiatives)

RDF Graph:



RDF-triples: (URI_2, `:email`, `rgarrido@utalca.cl`)

SPARQL Query:

```
SELECT ?N ?E
WHERE
{
  ?X :name ?N .
  ?X :email ?E .
  ?X :friendOf ?Y . ?Y :name "Ruth Garrido"
}
```

SPARQL static query analysis have been hardly considered so far

- Foundations of SPARQL have been laid
- Optimization rules have been proposed

SPARQL static query analysis have been hardly considered so far

- Foundations of SPARQL have been laid
- Optimization rules have been proposed

But we lack sound and complete SPARQL static analysis results in the spirit of *C&M-77* for CQs

SPARQL static query analysis have been hardly considered so far

- Foundations of SPARQL have been laid
- Optimization rules have been proposed

But we lack sound and complete SPARQL static analysis results in the spirit of *C&M-77* for CQs

Our goals:

- Static analysis of SPARQL
- Apply CQ tractable results to SPARQL

Main contributuions

Conceptual contribution: *Pattern trees*

- Tree representation for a fragment of SPARQL graph patterns
- First step towards an algebra for SPARQL *query plans*

Main contributuions

Conceptual contribution: *Pattern trees*

- Tree representation for a fragment of SPARQL graph patterns
- First step towards an algebra for SPARQL *query plans*

Technical contributions:

- Homomorphism-based redundancy elimination and equivalence
- Complexity of testing equivalence (NP) and subsumption (Π_2^P)
- Tractable enumeration & complexity of counting

Static Analysis and Optimization of Semantic Web Queries

Andrés Letelier Jorge Pérez Reinhard Pichler Sebastian Skritek

PUC Chile Universidad de Chile TU Vienna

Outline

Outline

SPARQL: a simple RDF query language

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

SPARQL: a simple RDF query language

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

In general, in a query we have:

$$H \leftarrow$$

- Head: processing of some variables (projection, order, ...)

SPARQL: a simple RDF query language

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

In general, in a query we have:

$$H \leftarrow P$$

- Head: processing of some variables (projection, order, ...)
- Body: pattern matching expression (graph patterns)

SPARQL: a simple RDF query language

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

In general, in a query we have:

$$H \leftarrow P$$

- Head: processing of some variables (projection, order, ...)
- Body: pattern matching expression (graph patterns)

We focus on P

SPARQL: a simple RDF query language

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

In general, in a query we have:

$$H \leftarrow P$$

- Head: processing of some variables (projection, order, ...)
- Body: pattern matching expression (graph patterns)

We focus on P

(i.e. we do not consider projection)

Graph patterns: interesting features of pattern matching on graphs

- Conjunctions

```
{ P1 .
  P2 }
```

Graph patterns: interesting features of pattern matching on graphs

- Conjunctions
- Grouping

```
{ { P1 .
    P2 }

  { P3 .
    P4 }


}
```

Graph patterns: interesting features of pattern matching on graphs

- Conjunctions
- Grouping
- Optional parts

```
{ { P1 .
    P2
    OPTIONAL { P5 }  }
  { P3 .
    P4
    OPTIONAL { P7 }  }

}
```

Graph patterns: interesting features of pattern matching on graphs

- Conjunctions
- Grouping
- Optional parts
- Nesting of optionals

```
{ { P1 .
    P2
    OPTIONAL { P5 }  }
  { P3 .
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
```

Graph patterns: interesting features of pattern matching on graphs

- Conjunctions
- Grouping
- Optional parts
- Nesting of optionals
- Union of patterns

```
{ { P1 .
    P2
    OPTIONAL { P5 }  }
  { P3 .
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9 }
```

Graph patterns: interesting features of pattern matching on graphs

- Conjunctions
- Grouping
- Optional parts
- Nesting of optionals
- Union of patterns
- Filtering

```
{ { P1 .
    P2
    OPTIONAL { P5 }  }
  { P3 .
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9 FILTER ( R ) }
```

Graph patterns: interesting features of pattern matching on graphs

- Conjunctions
- Grouping
- Optional parts
- Nesting of optionals
- Union of patterns
- Filtering
- ...

```
{ { P1 .
    P2
    OPTIONAL { P5 }  }
  { P3 .
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9 FILTER ( R ) }
```

Graph patterns: interesting features of pattern matching on graphs

- Conjunctions
- Grouping
- Optional parts
- Nesting of optionals
- Union of patterns
- Filtering
- ...
- new version: regular expressions, federated queries, sub-queries, etc.

```
{ { P1 .
    P2
    OPTIONAL { P5 }  }
  { P3 .
    P4
    OPTIONAL { P7
      OPTIONAL { P8 }  }  }
}
UNION
{ P9 FILTER ( R ) }
```

A standard algebraic syntax

- Triple patterns: just triples + variables ($?X, ?Y, \ldots$)

original SPARQL syntax	algebraic syntax
`?X :name "john"`	$(?X, \text{name}, \text{john})$

- Graph patterns: algebraic expressions over triple patterns

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

A standard algebraic syntax

- Triple patterns: just triples + variables ($?X, ?Y, \ldots$)

original SPARQL syntax	algebraic syntax
`?X :name "john"`	$(?X, \text{name}, \text{john})$

- Graph patterns: algebraic expressions over triple patterns

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

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

$(R_1, \text{name}, \text{john})$	$(R_2, \text{name}, \text{paul})$	$(R_3, \text{name}, \text{ringo})$
$(R_1, \text{email}, \text{J@ed.ex})$		$(R_3, \text{email}, \text{R@ed.ex})$
		$(R_3, \text{wPage}, \text{w3.rin.ed})$

$$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{wPage}, ?W))$$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

$$(\,(?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{wPage},\ ?W)\,)$$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

$$(\,(?X,\ \text{name},\ ?Y)\ \text{OPT}\ (?X,\ \text{wPage},\ ?W)\,)$$

$\{?X \to R_1,\ ?Y \to \text{john}\}$
$\{?X \to R_2,\ ?Y \to \text{paul}\}$
$\{?X \to R_3,\ ?Y \to \text{ringo}\}$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

$$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{wPage}, ?W))$$

$\{?X \to R_1, ?Y \to \text{john}\}$
$\{?X \to R_2, ?Y \to \text{paul}\}$
$\{?X \to R_3, ?Y \to \text{ringo}\}$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

$$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{wPage}, ?W))$$

$\{?X \to R_1, ?Y \to \text{john}\}$
$\{?X \to R_2, ?Y \to \text{paul}\}$
$\{?X \to R_3, ?Y \to \text{ringo}\}$

$\{?X \to R_3, ?W \to \text{w3.rin.ed}\}$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

$$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{wPage}, ?W))$$

$\{?X \to R_1, ?Y \to \text{john}\}$
$\{?X \to R_2, ?Y \to \text{paul}\}$
$\{?X \to R_3, ?Y \to \text{ringo}\}$

$\{?X \to R_3, ?W \to \text{w3.rin.ed}\}$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

$$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{wPage}, ?W))$$

$\{?X \to R_1, ?Y \to \text{john}\}$
$\{?X \to R_2, ?Y \to \text{paul}\}$
$\{?X \to R_3, ?Y \to \text{ringo}\}$

$\{?X \to R_3, ?W \to \text{w3.rin.ed}\}$

$\{?X \to R_1, ?Y \to \text{john}\}$
$\{?X \to R_2, ?Y \to \text{paul}\}$
$\{?X \to R_3, ?Y \to \text{ringo}, ?W \to \text{w3.rin.ed}\}$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

$((?X, \text{name}, ?Y) \text{ OPT } (?X, \text{wPage}, ?W))$

$\{?X \to R_1, ?Y \to \text{john}\}$
$\{?X \to R_2, ?Y \to \text{paul}\}$
$\{?X \to R_3, ?Y \to \text{ringo}, ?W \to \text{w3.rin.ed}\}$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

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

$\{?X \to R_1,\ ?Y \to \text{john}\}$
$\{?X \to R_2,\ ?Y \to \text{paul}\}$
$\{?X \to R_3,\ ?Y \to \text{ringo},\ ?W \to \text{w3.rin.ed}\}$

Evaluating SPARQL patterns: a simple example

Evaluation formalized in terms of *(partial) mappings*

$$\mu : \text{Variables} \longrightarrow \text{RDF-Terms}$$

Example

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

$(R_2, \text{name}, \text{paul})$

$(R_3, \text{name}, \text{ringo})$
$(R_3, \text{email}, \text{R@ed.ex})$
$(R_3, \text{wPage}, \text{w3.rin.ed})$

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

$\{?X \to R_1,\ ?Y \to \text{john}, ?E \to J@ed.ex\}$
$\{?X \to R_2,\ ?Y \to \text{paul}\}$
$\{?X \to R_3,\ ?Y \to \text{ringo},\ ?W \to \text{w3.rin.ed}, ?E \to R@ed.ex\}$

Optional parts lead to high complexity of evaluation

Evaluation for the AND-OPT fragment is PSPACE-complete

- high complexity: unrestricted use of (several nested) OPTs

Optional parts lead to high complexity of evaluation

Evaluation for the AND-OPT fragment is PSPACE-complete

- high complexity: unrestricted use of (several nested) OPTs

A graph pattern is *well–designed* iff for every OPT in the pattern

$$(\ \cdots\cdots\cdots\ \ (\ \ A\ \ \text{OPT}\ \ B\ \ )\ \ \cdots\cdots\cdots\ )$$

if a variable occurs

Optional parts lead to high complexity of evaluation

Evaluation for the AND-OPT fragment is PSPACE-complete

- high complexity: unrestricted use of (several nested) OPTs

A graph pattern is *well–designed* iff for every OPT in the pattern

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

if a variable occurs inside B

Optional parts lead to high complexity of evaluation

Evaluation for the AND-OPT fragment is PSPACE-complete

- high complexity: unrestricted use of (several nested) OPTs

A graph pattern is *well–designed* iff for every OPT in the pattern

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

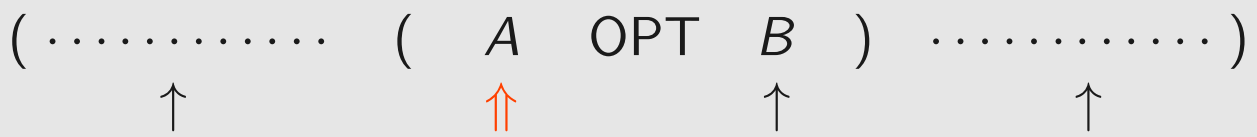
if a variable occurs inside B and anywhere outside the OPT,

Optional parts lead to high complexity of evaluation

Evaluation for the AND-OPT fragment is PSPACE-complete

- high complexity: unrestricted use of (several nested) OPTs

A graph pattern is *well–designed* iff for every OPT in the pattern



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

Optional parts lead to high complexity of evaluation

Evaluation for the AND-OPT fragment is PSPACE-complete

- high complexity: unrestricted use of (several nested) OPTs

A graph pattern is *well–designed* iff for every OPT in the pattern

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

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

$$\big(\; (?X, \text{name}, \text{john}) \; \text{OPT} \; \big[(?Y, \text{name}, \text{ringo}) \; \text{OPT} \; (?X, \text{wPage}, ?W) \big] \; \big)$$

Optional parts lead to high complexity of evaluation

Evaluation for the AND-OPT fragment is PSPACE-complete

- high complexity: unrestricted use of (several nested) OPTs

A graph pattern is *well–designed* iff for every OPT in the pattern

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

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

$$\big(\ (?X, \text{name}, \text{john})\ \text{OPT}\ \big[\ (?Y, \text{name}, \text{ringo})\ \text{OPT}\ (?X, \text{wPage}, ?W)\ \big]\ \big)$$

Optional parts lead to high complexity of evaluation

Evaluation for the AND-OPT fragment is PSPACE-complete

- high complexity: unrestricted use of (several nested) OPTs

A graph pattern is *well–designed* iff for every OPT in the pattern

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

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

$$\big(\ (?X, \text{name}, \text{john})\ \text{OPT}\ \big[\ (?Y, \text{name}, \text{ringo})\ \text{OPT}\ (?X, \text{wPage}, ?W)\ \big]\ \big)$$

[PAG09] evaluation for the well-designed fragment is *coNP*

Outline

Extending CQs to capture SPARQL graph patterns

- Conjunctions (AND) of triple patterns are essentially CQs
- We want to extend static analysis of CQs to SPARQL

Extending CQs to capture SPARQL graph patterns

- Conjunctions (AND) of triple patterns are essentially CQs
- We want to extend static analysis of CQs to SPARQL

Idea

Represent AND-OPT patterns as an extension of CQs:

- graph patterns represented by *(unordered & rooted) trees*
- nodes representing conjunctions of triple patterns (CQs)
- tree structure representing *optionality*

Extending CQs to capture SPARQL graph patterns

- Conjunctions (AND) of triple patterns are essentially CQs
- We want to extend static analysis of CQs to SPARQL

Idea

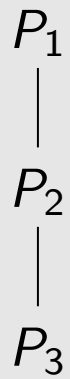
Represent AND-OPT patterns as an extension of CQs:

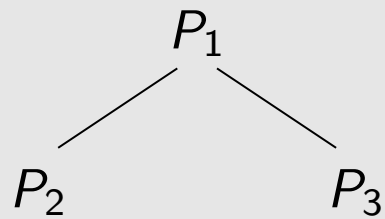
- graph patterns represented by *(unordered & rooted) trees*
- nodes representing conjunctions of triple patterns (CQs)
- tree structure representing *optionality*

Pattern Trees

Pattern tree example

Intuition





$$\Big(P_1 \text{ OPT } (P_2 \text{ OPT } P_3)\Big) \qquad \Big((P_1 \text{ OPT } P_2) \text{ OPT } P_3\Big)$$

Quasi-well designed pattern trees (QWDPTs)

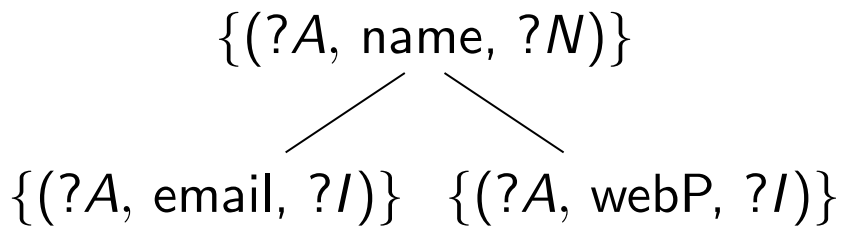
Definition

A pattern tree is *quasi-well desgined* if for every two nodes mentioning $?X$, there is a common ancestor that also mentions $?X$

Quasi-well designed pattern trees (QWDPTs)

Definition

A pattern tree is *quasi-well desgined* if for every two nodes mentioning $?X$, there is a common ancestor that also mentions $?X$

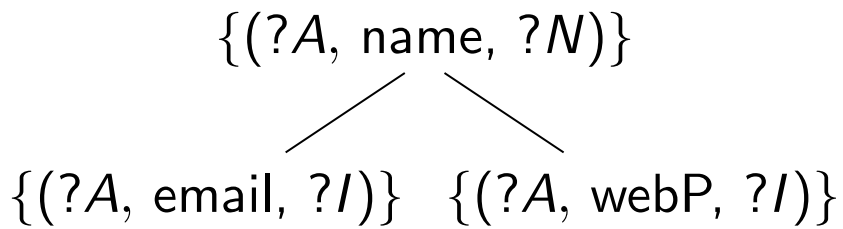


non QWDPT (because of $?I$)

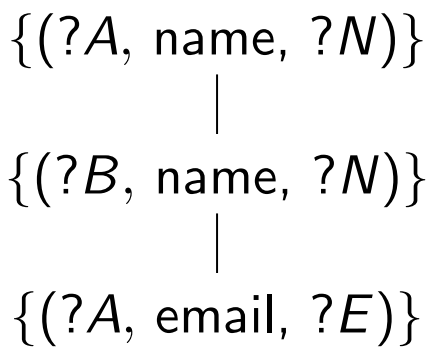
Quasi-well designed pattern trees (QWDPTs)

Definition

A pattern tree is *quasi-well desgined* if for every two nodes mentioning $?X$, there is a common ancestor that also mentions $?X$



non QWDPT (because of $?I$)

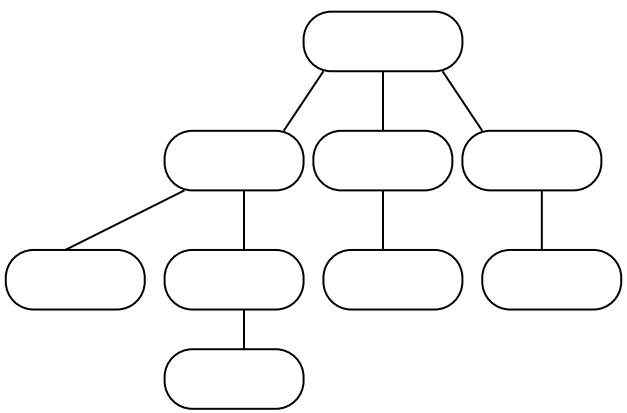


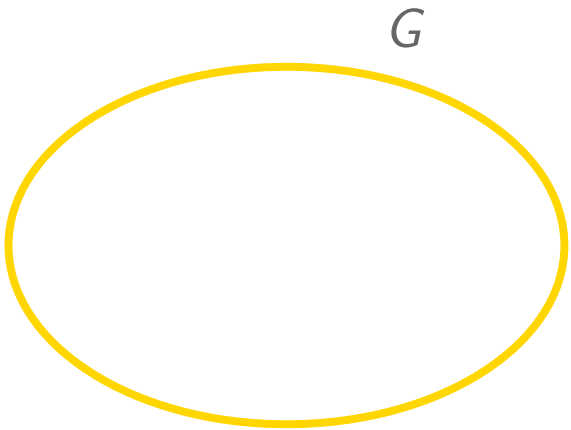
QWDPT

Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible

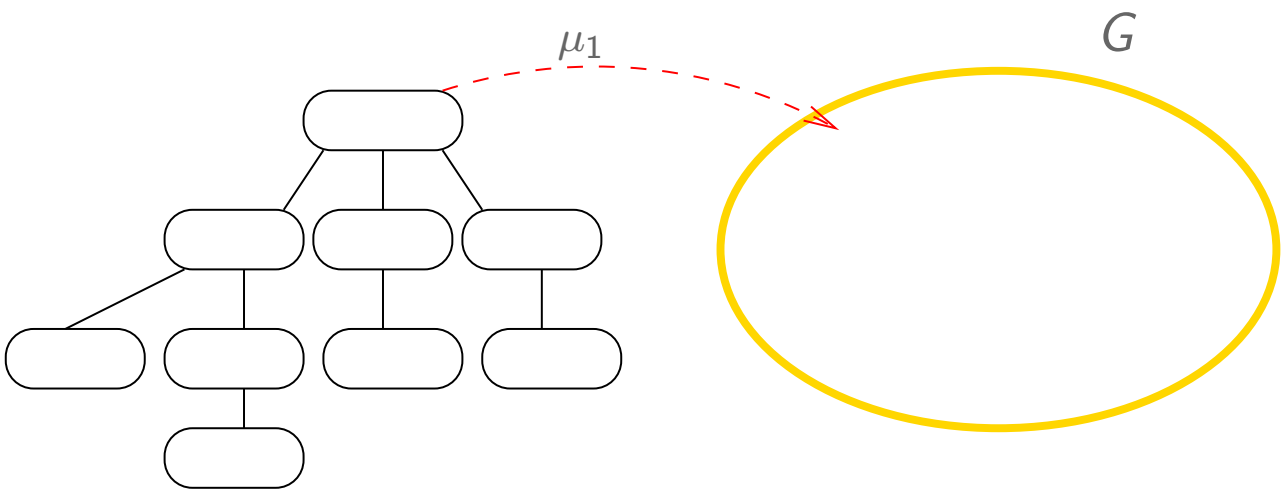




Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

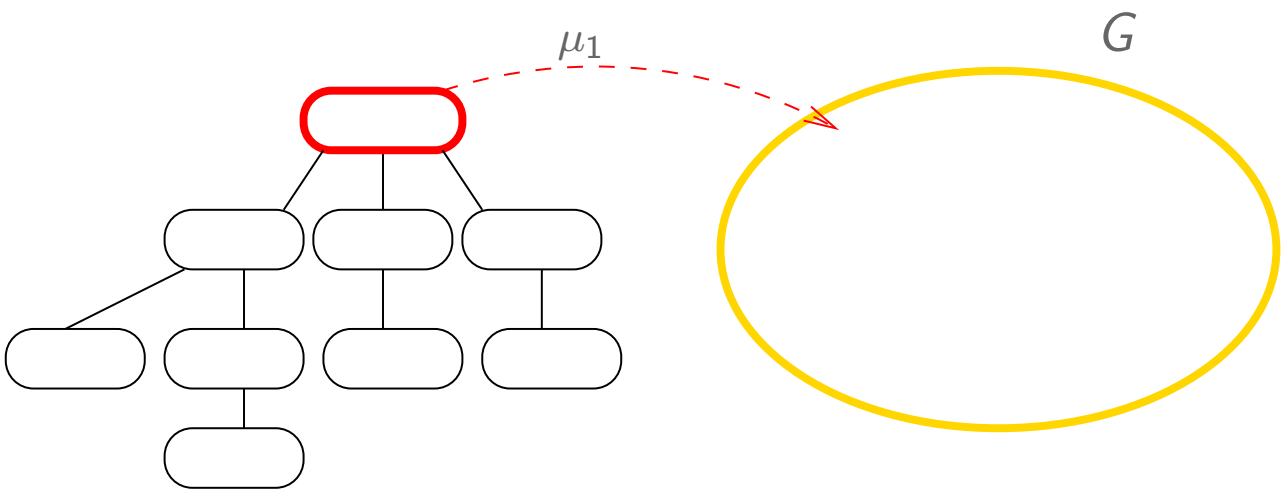
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

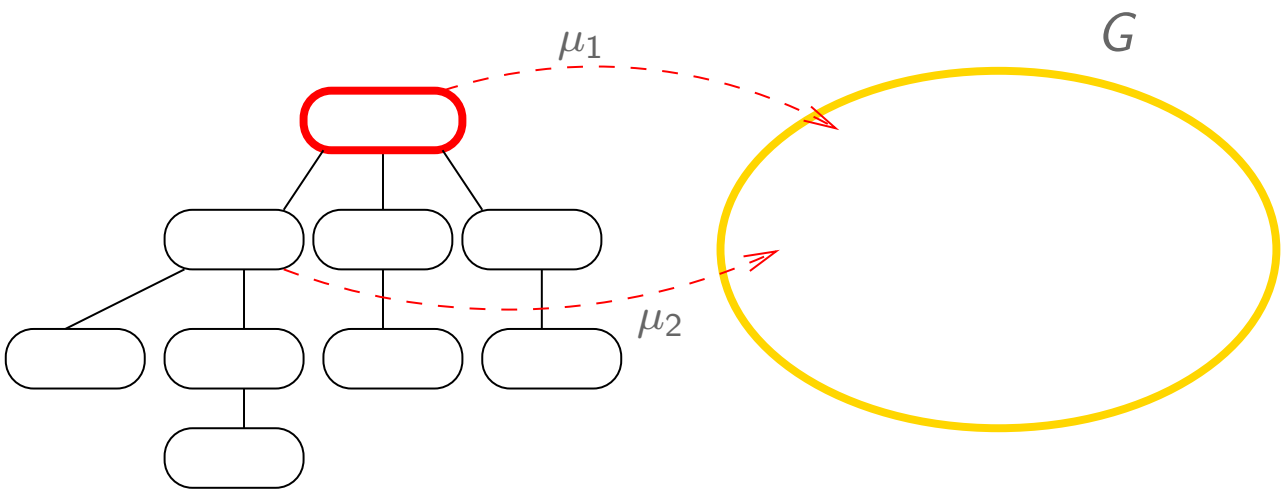
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

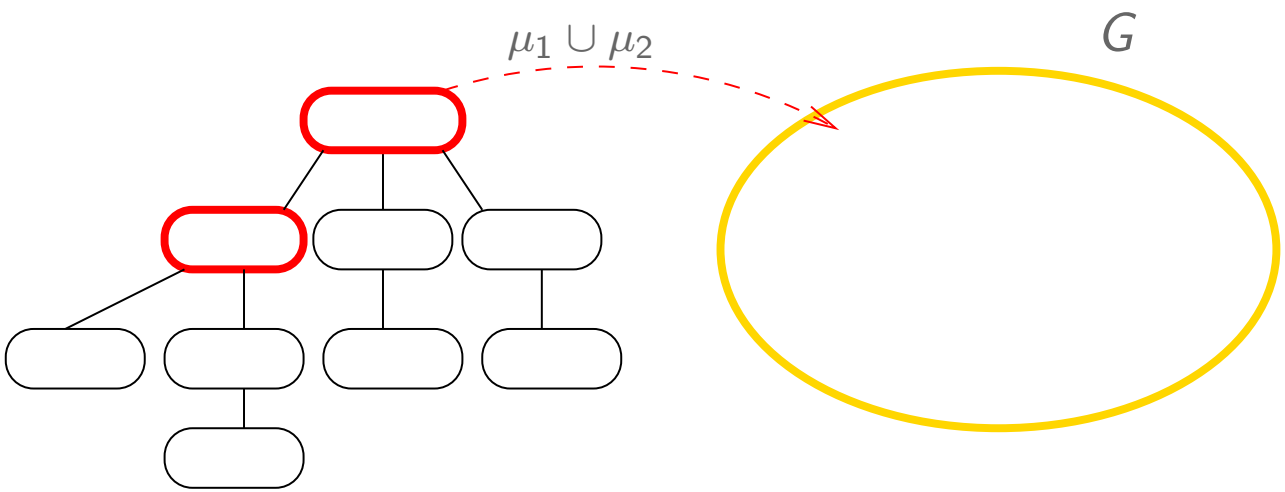
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

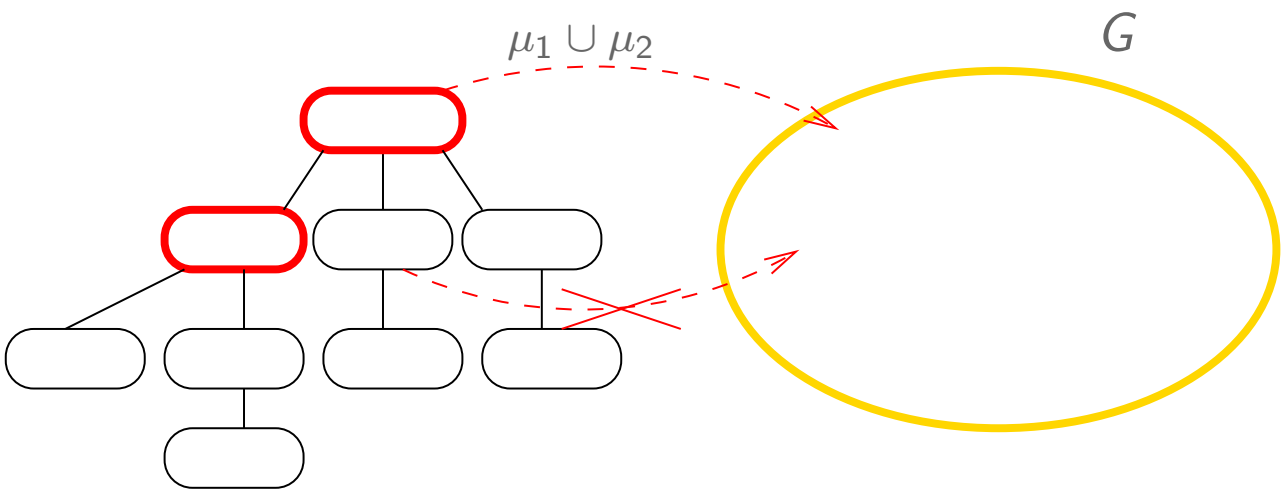
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

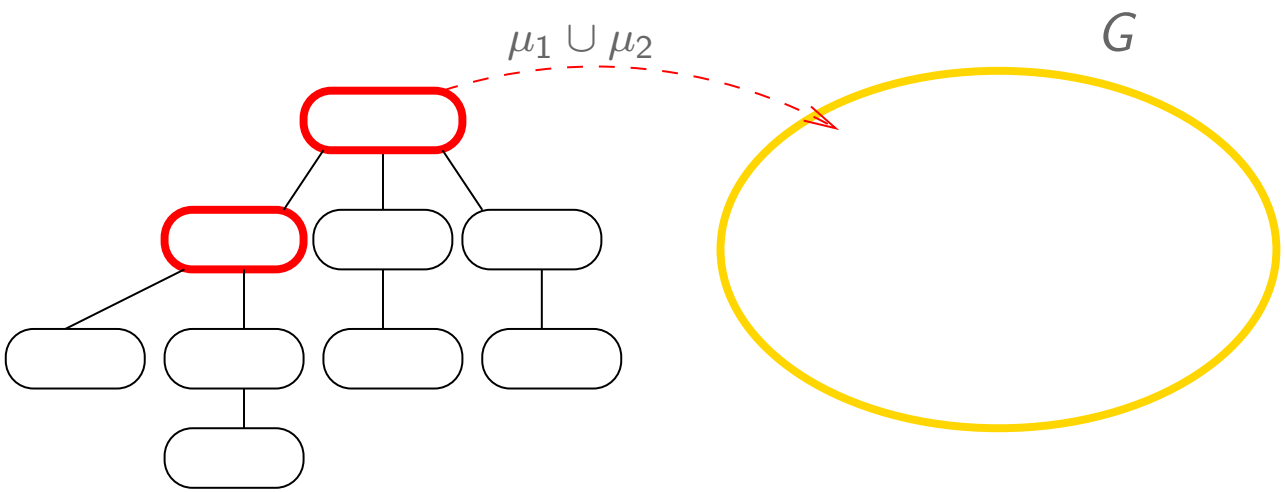
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

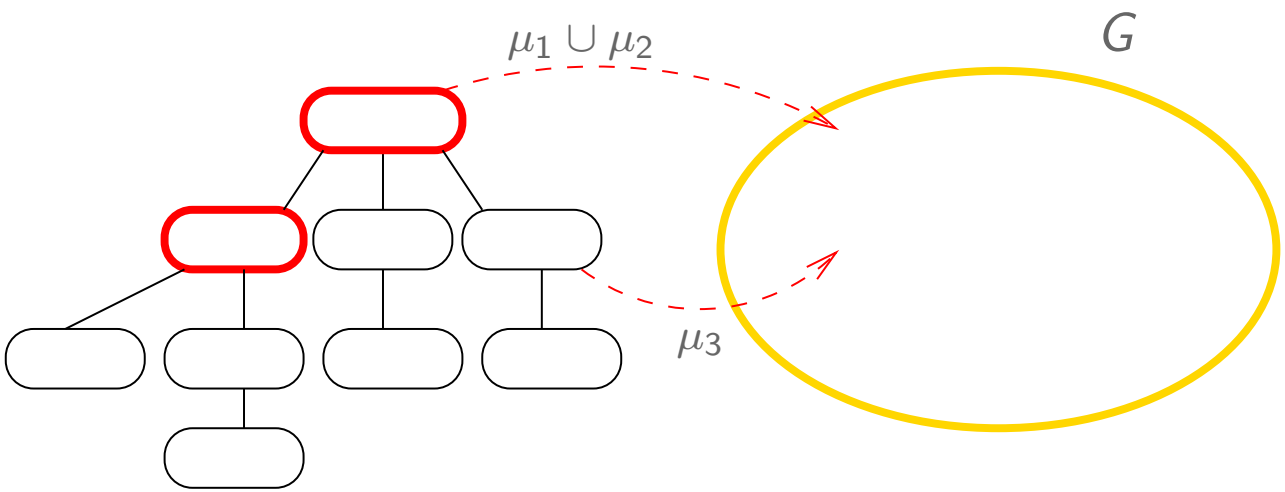
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

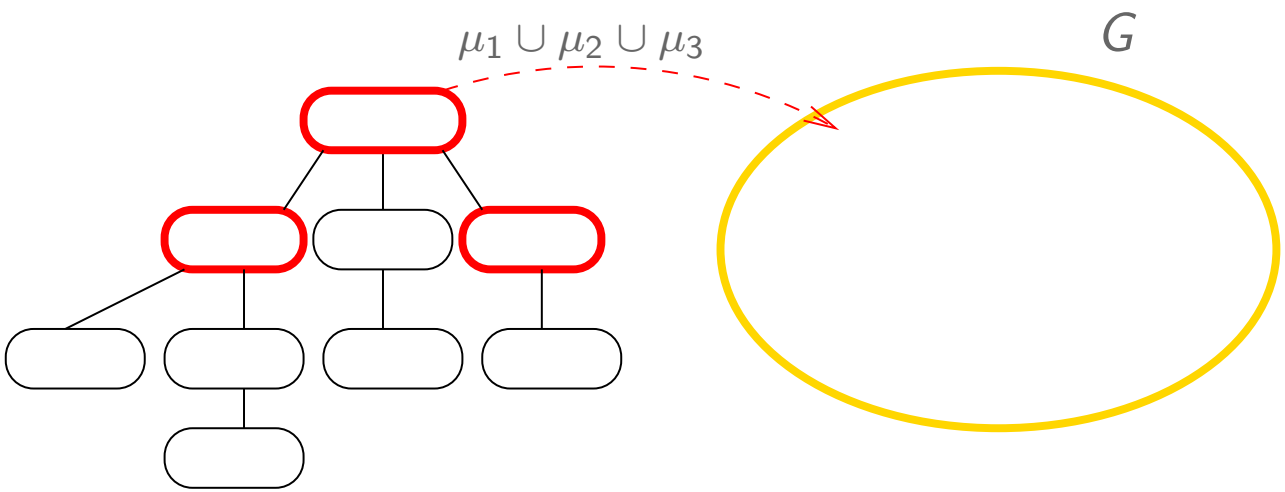
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

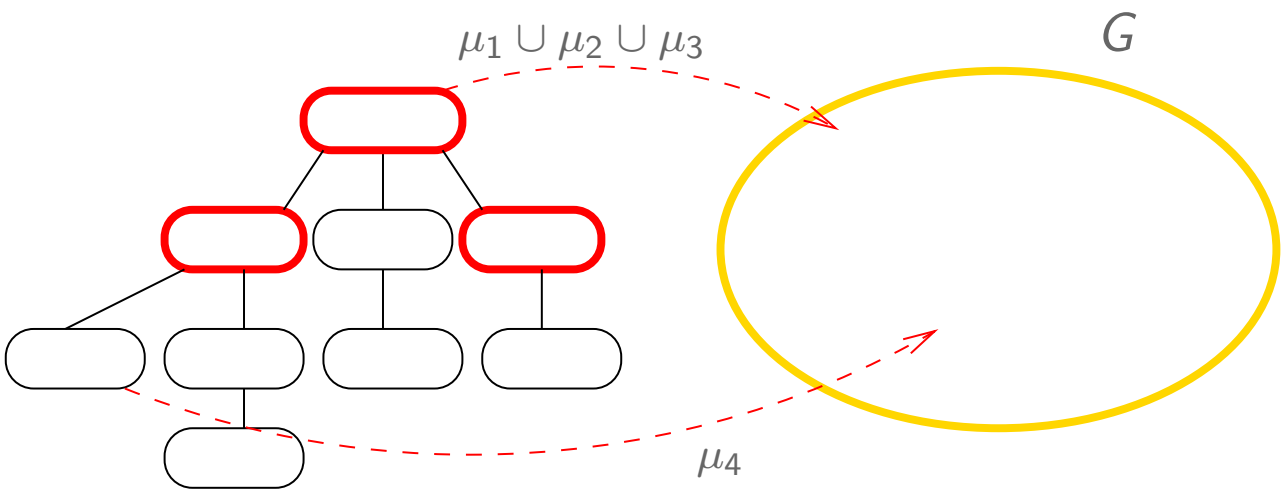
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

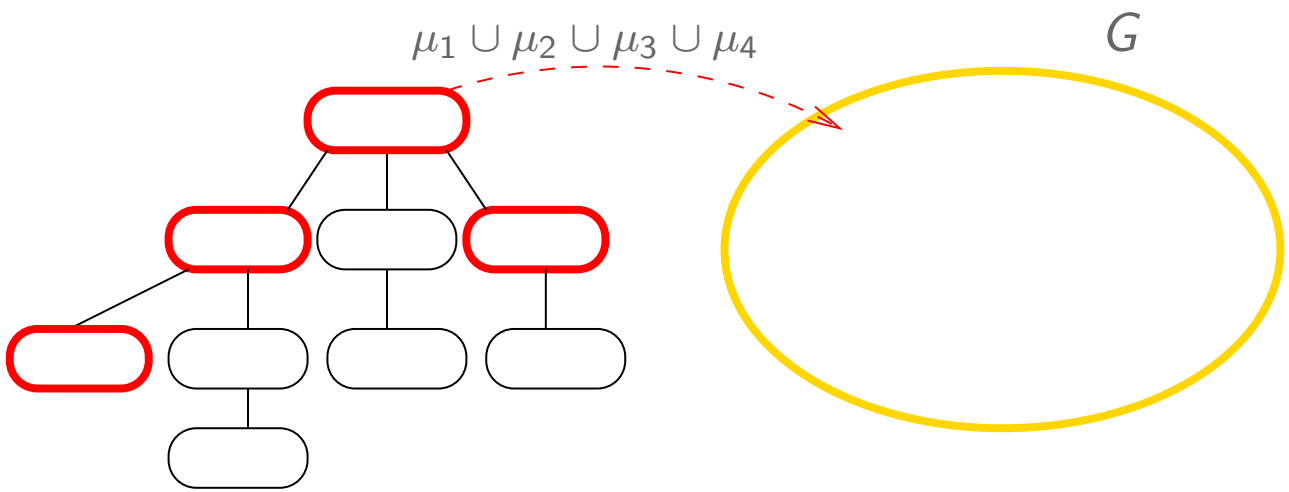
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

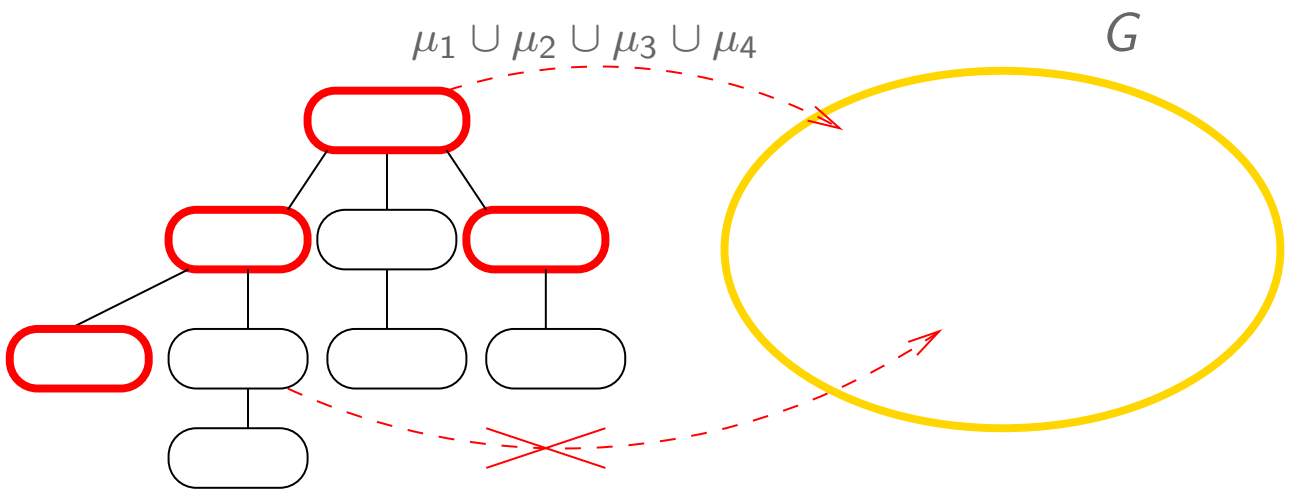
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

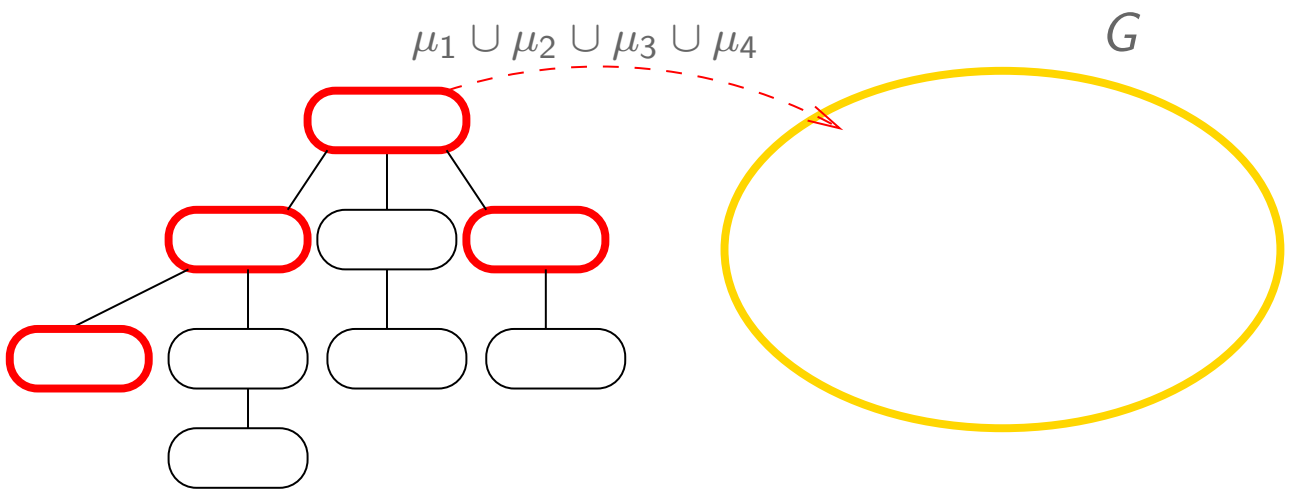
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

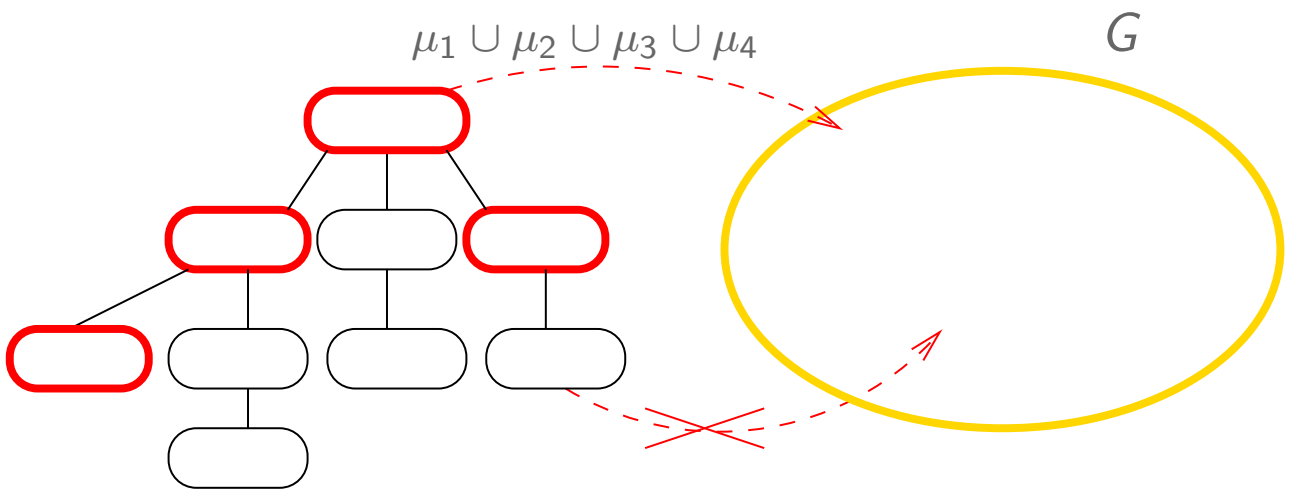
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

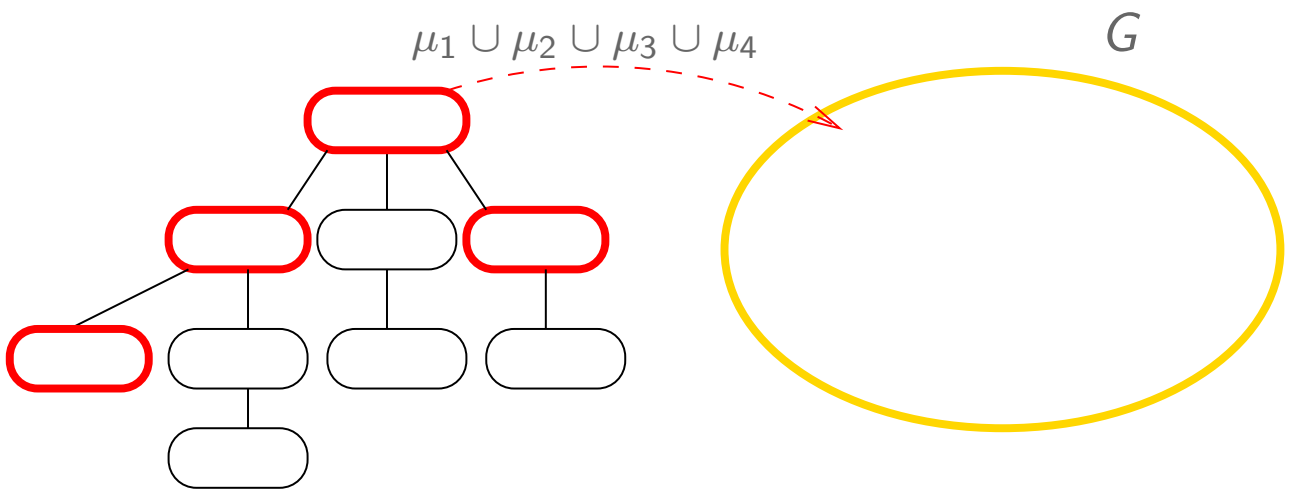
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

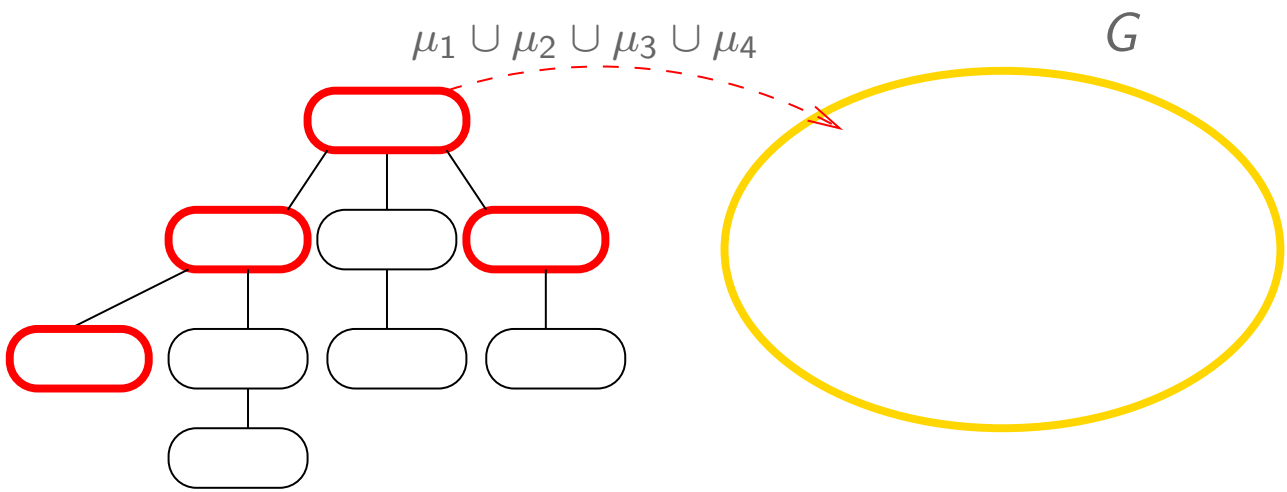
- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



Top-down evaluation of pattern trees

QWDPTs allow for a *natural evaluation* from the root to the leaves:

- compute a mapping for a given node
- then try to extend it with the evaluation of its children
- stop in a branch if no further extension is possible



$\mu_1 \cup \mu_2 \cup \mu_3 \cup \mu_4$ is a solution over G

QWDPTs represent well-designed SPARQL graph patterns

Theorem

For every well-designed pattern P there exists a QWDPT $\mathcal{T}$ s.t.

$$\textit{top-down evaluation of } \mathcal{T} \equiv \textit{algebraic evaluation of } P$$

QWDPTs represent well-designed SPARQL graph patterns

Theorem

For every well-designed pattern P there exists a QWDPT $\mathcal{T}$ s.t.

$$\textit{top-down evaluation of } \mathcal{T} \equiv \textit{algebraic evaluation of } P$$

And the other way around

Theorem

For every QWDPT $\mathcal{T}$ there exists a well-designed pattern P s.t.

$$\textit{top-down evaluation of } \mathcal{T} \equiv \textit{algebraic evaluation of } P$$

QWDPTs represent well-designed SPARQL graph patterns

Theorem

For every well-designed pattern P there exists a QWDPT $\mathcal{T}$ s.t.

$$\textit{top-down evaluation of } \mathcal{T} \equiv \textit{algebraic evaluation of } P$$

And the other way around

Theorem

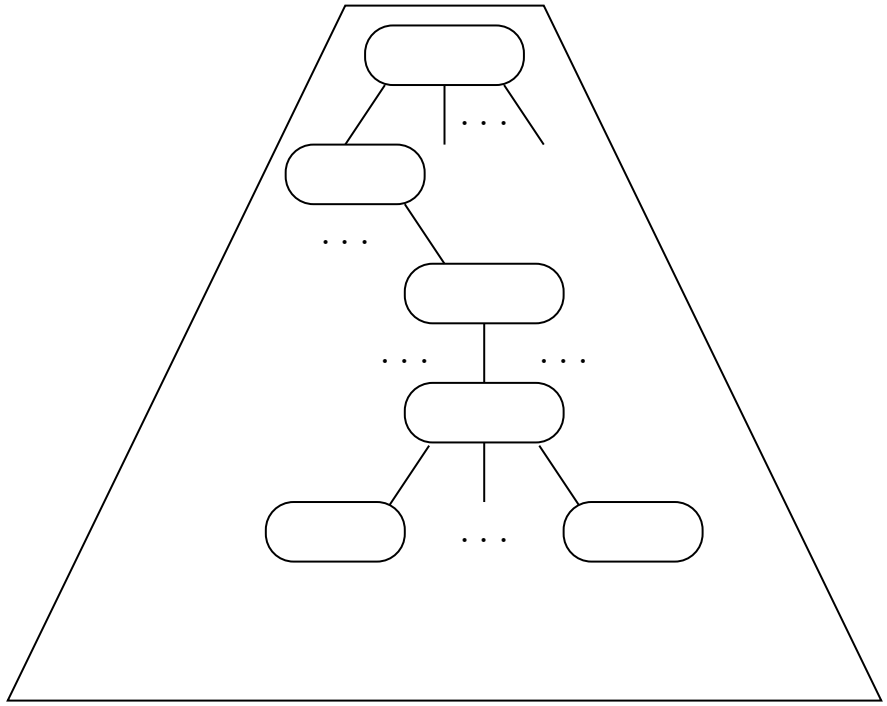
For every QWDPT $\mathcal{T}$ there exists a well-designed pattern P s.t.

$$\textit{top-down evaluation of } \mathcal{T} \equiv \textit{algebraic evaluation of } P$$

Moreover, we can go from one representation
into the other in polynomial time

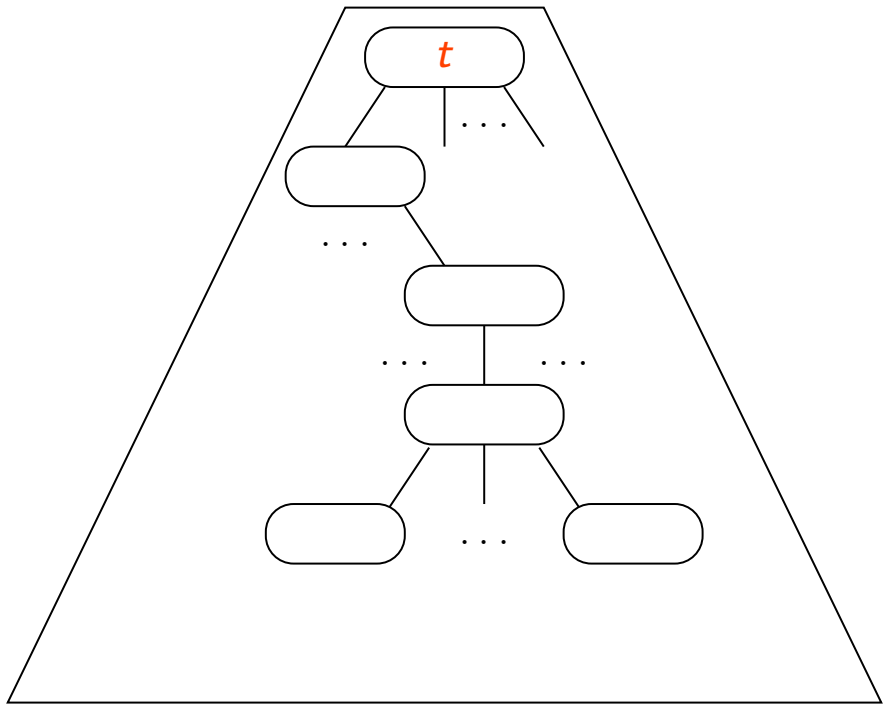
Outline

Transformation rules: deleting redundant triples



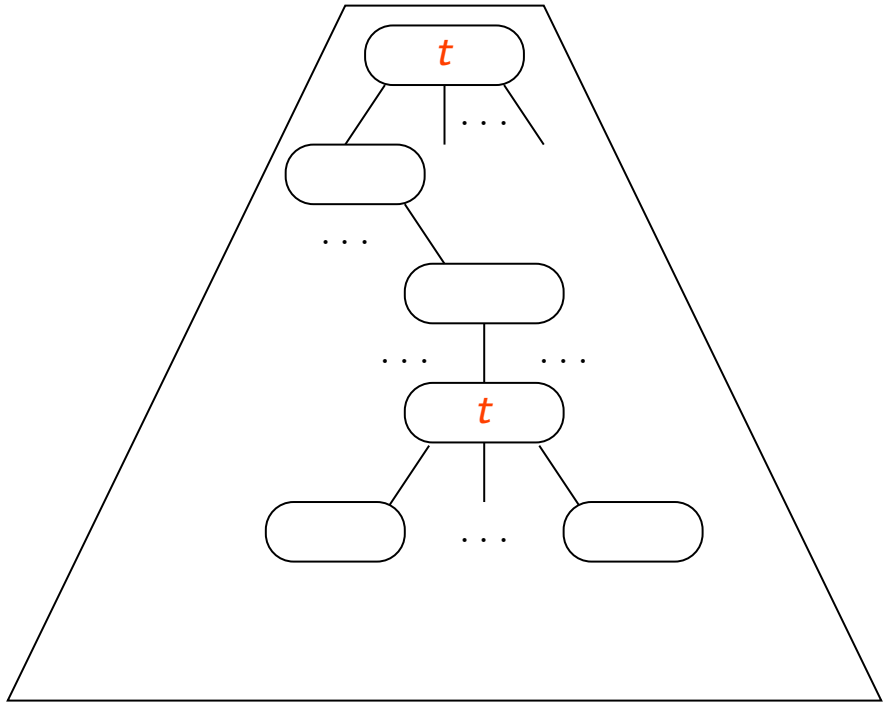
R1: If a triple pattern t belongs to node n

Transformation rules: deleting redundant triples



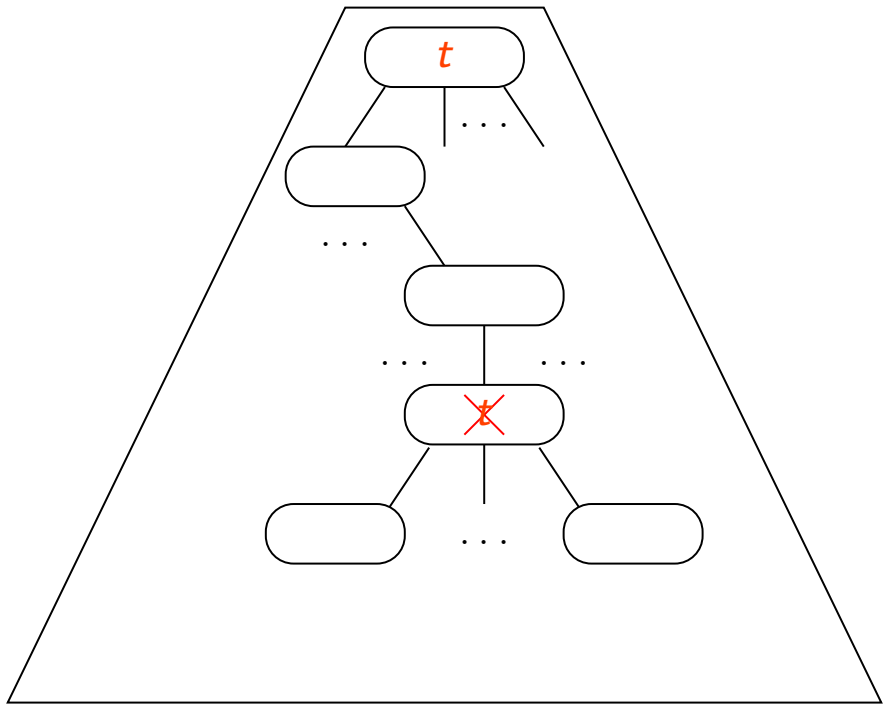
R1: If a triple pattern t belongs to node n

Transformation rules: deleting redundant triples



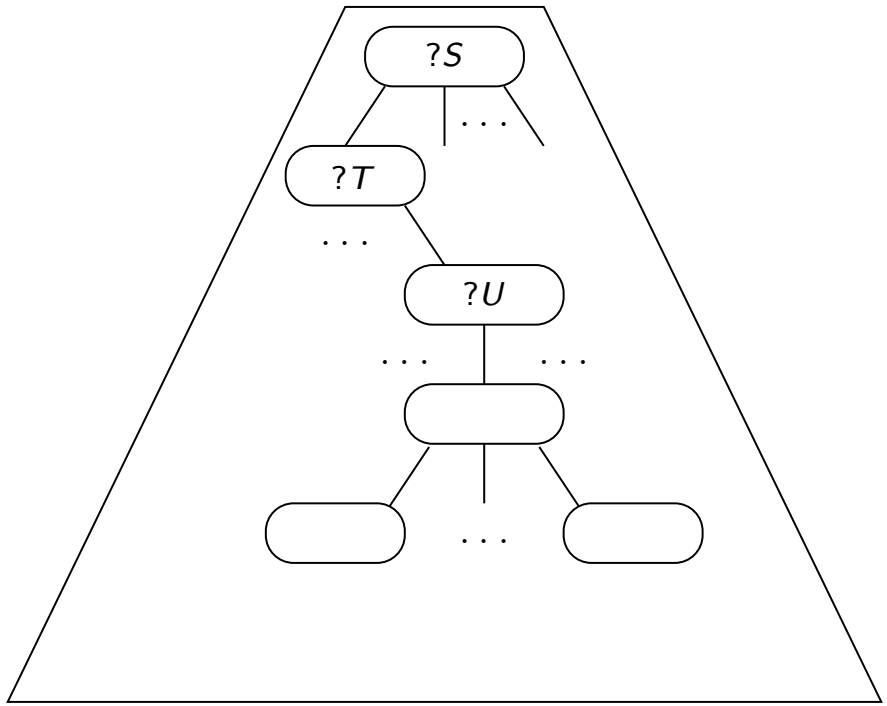
R1: If a triple pattern t belongs to node n and to a descendant n' of n,

Transformation rules: deleting redundant triples



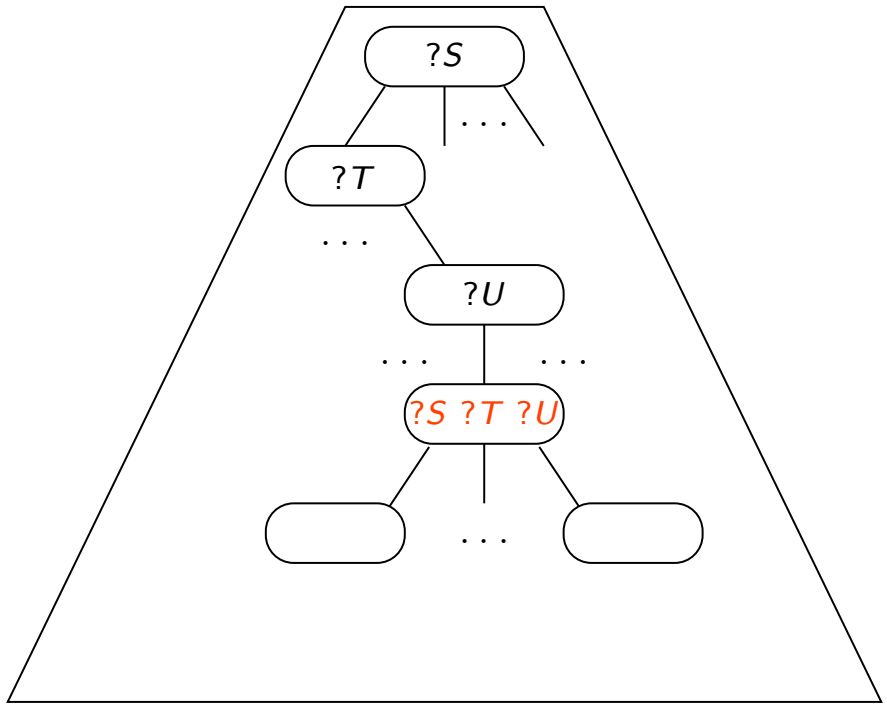
R1: If a triple pattern t belongs to node n and to a descendant n' of n, then delete t from n'.

Transformation rules: deleting redundant nodes



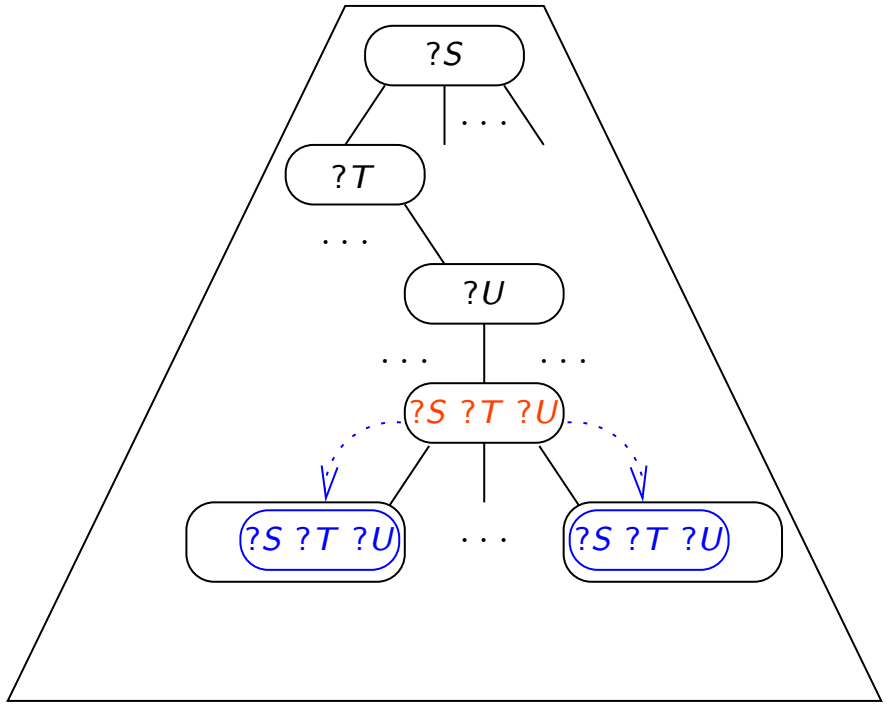
R2: If node n

Transformation rules: deleting redundant nodes



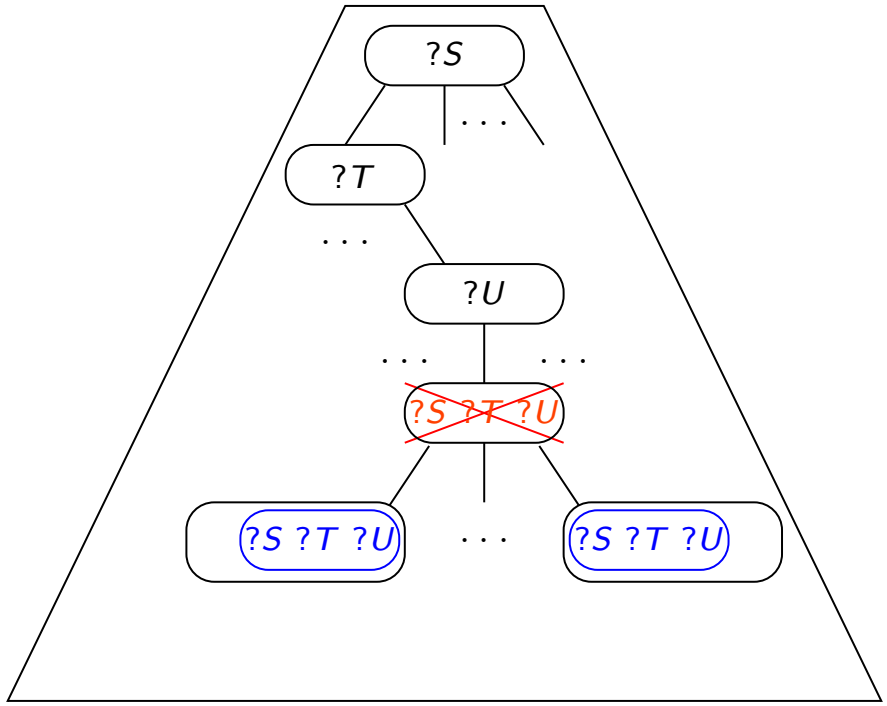
R2: If node n does not introduce any new variable w.r.t. its ancestors,

Transformation rules: deleting redundant nodes



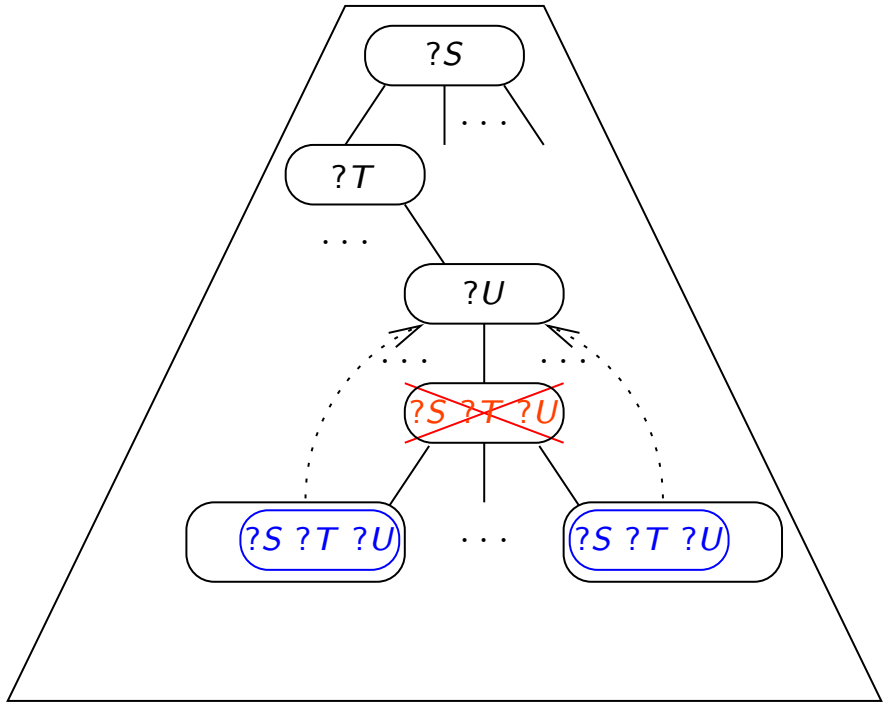
R2: If node n does not introduce any new variable w.r.t. its ancestors, then *push* copies of n into its children,

Transformation rules: deleting redundant nodes



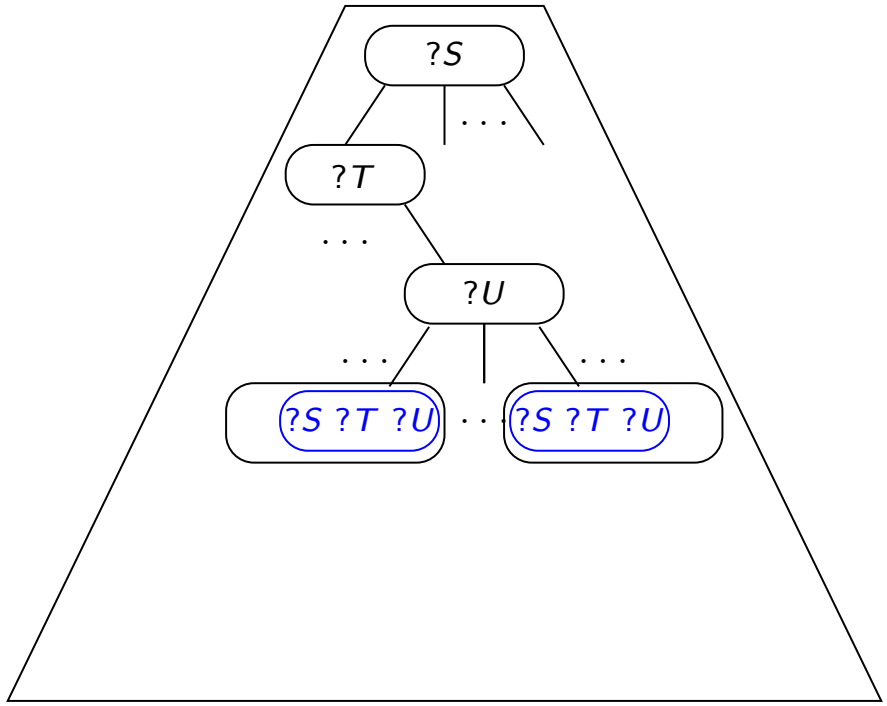
R2: If node n does not introduce any new variable w.r.t. its ancestors, then *push* copies of n into its children, and delete the node.

Transformation rules: deleting redundant nodes



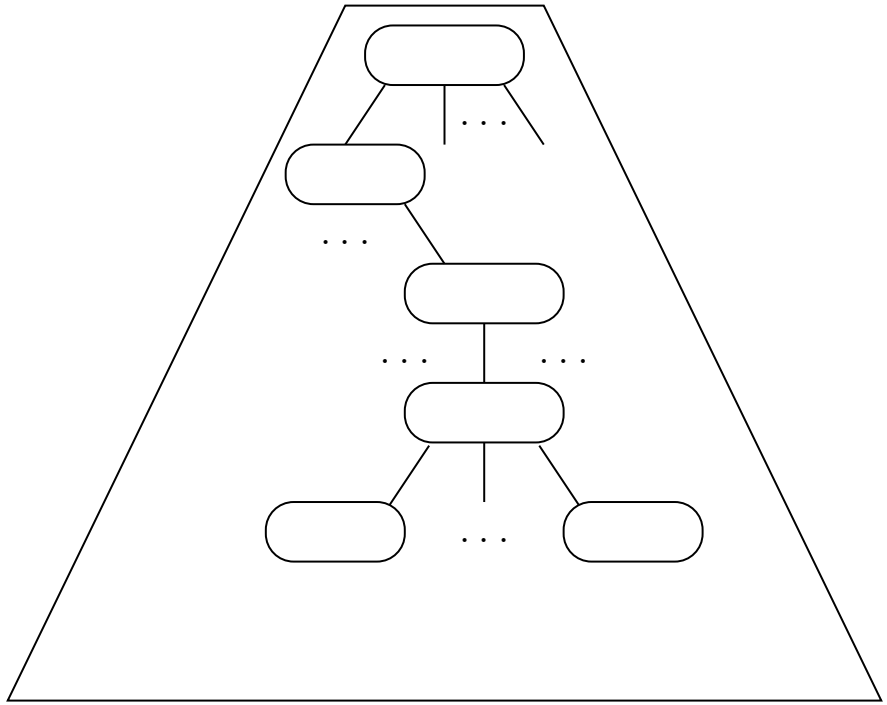
R2: If node n does not introduce any new variable w.r.t. its ancestors, then *push* copies of n into its children, and delete the node.

Transformation rules: deleting redundant nodes



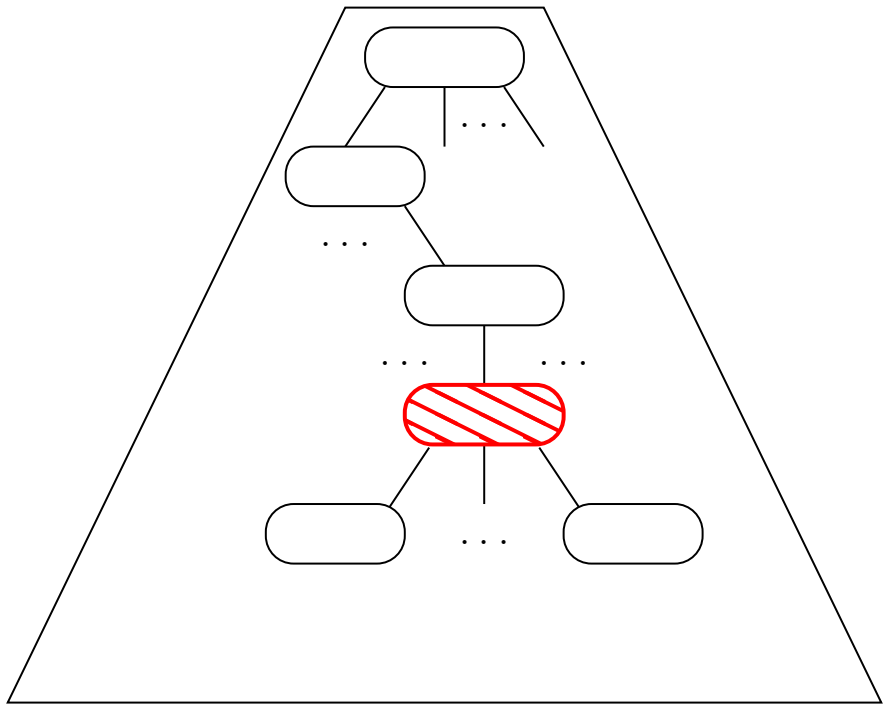
R2: If node n does not introduce any new variable w.r.t. its ancestors, then *push* copies of n into its children, and delete the node.

Transformation rules: homomorphism upwards



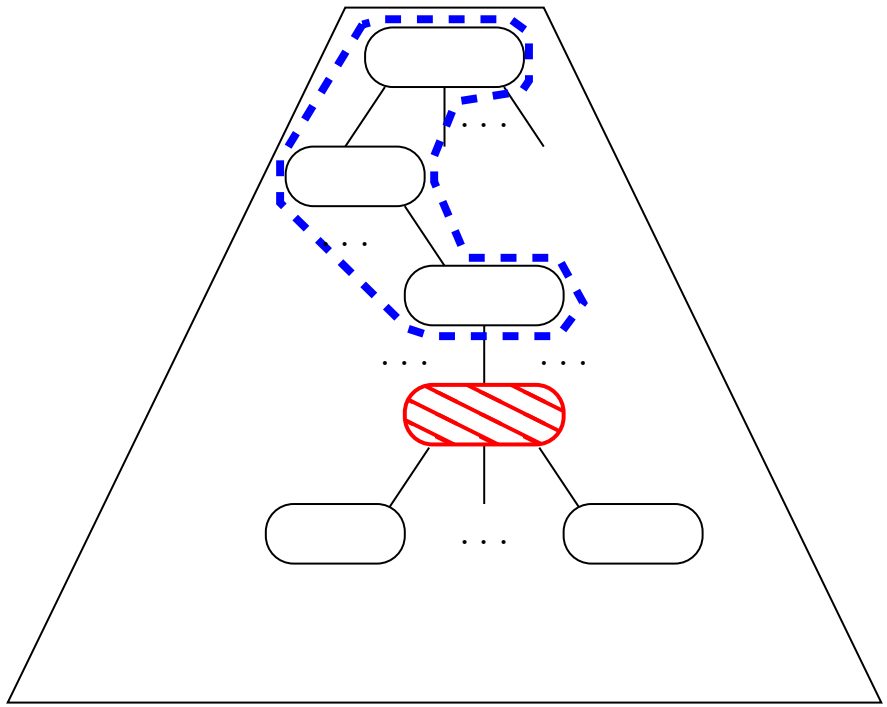
R3: If there is a *homomorphism*

Transformation rules: homomorphism upwards



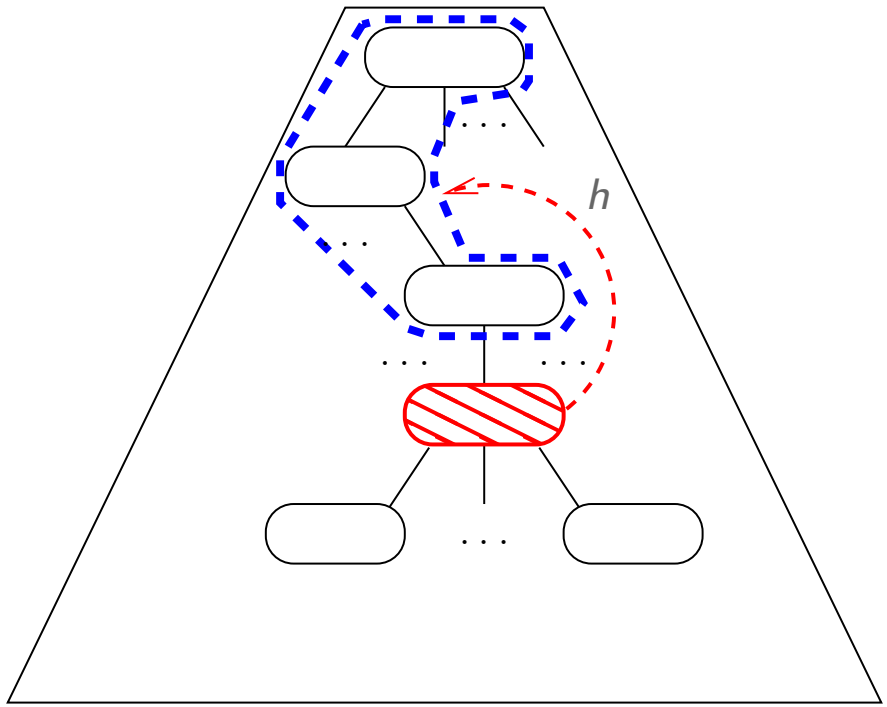
R3: If there is a *homomorphism* from node n

Transformation rules: homomorphism upwards



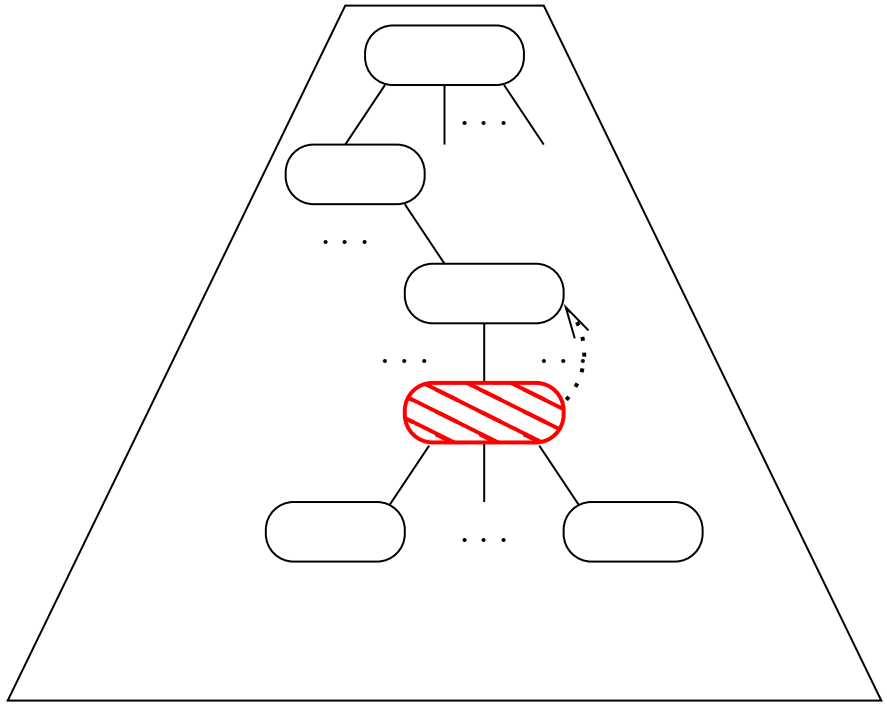
R3: If there is a *homomorphism* from node n to the branch of ancestors of n,

Transformation rules: homomorphism upwards



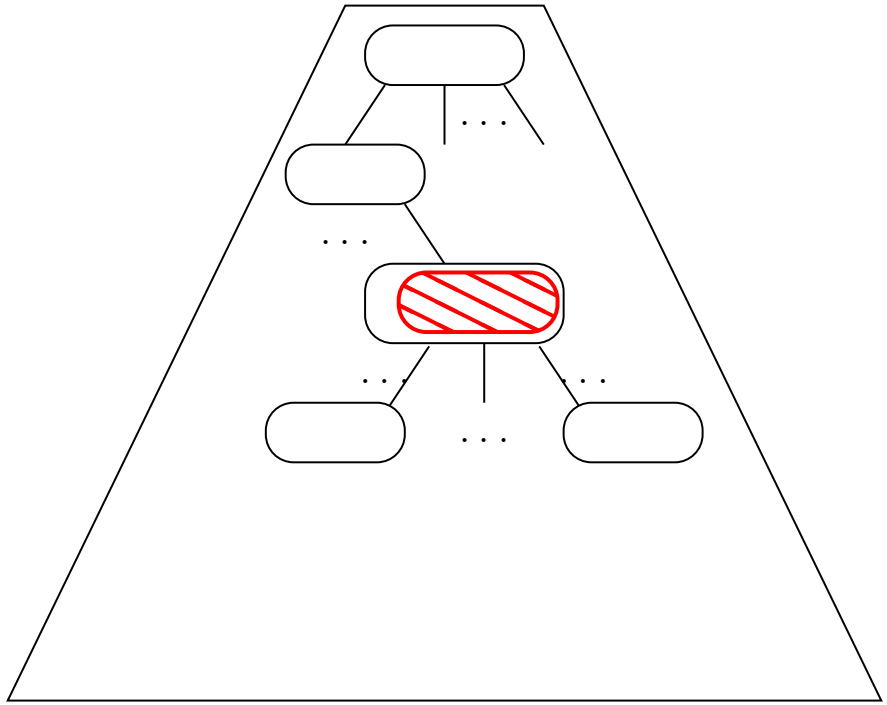
R3: If there is a *homomorphism* from node n to the branch of ancestors of n,

Transformation rules: homomorphism upwards



R3: If there is a *homomorphism* from node n to the branch of ancestors of n, then merge node n with its parent.

Transformation rules: homomorphism upwards



R3: If there is a *homomorphism* from node n to the branch of ancestors of n, then merge node n with its parent.

Normal forms

NR-normal-form: Apply R1 and R2 in any order

R3-normal-form: Apply R3 to pattern trees in NR-normal-form

Normal forms

NR-normal-form: Apply R1 and R2 in any order

R3-normal-form: Apply R3 to pattern trees in NR-normal-form

- NR-normal-form is *cheap*

Normal forms

NR-normal-form: Apply R1 and R2 in any order

R3-normal-form: Apply R3 to pattern trees in NR-normal-form

- NR-normal-form is *cheap*
- R3-normal-form requires a *coNP* check (to stop)

Normal forms

NR-normal-form: Apply R1 and R2 in any order

R3-normal-form: Apply R3 to pattern trees in NR-normal-form

- NR-normal-form is *cheap*
- R3-normal-form requires a *coNP* check (to stop)

Impact for optimization

Rules application *eliminate optional parts*

Also useful when testing equivalence of QWDPT

Outline

Subsumption

Definition

Mapping μ_1 is *subsumed* by μ_2, denoted by $\mu_1 \sqsubseteq \mu_2$, if μ_2 extends μ_1 with more variable bindings

Definition

$\mathcal{T}_1$ is *subsumed* by $\mathcal{T}_2$, denoted by $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, if for ever G

- every mapping in the evaluation of $\mathcal{T}_1$ over G, is subsumed by a mapping in the evaluation of $\mathcal{T}_2$ over G

Subsumption

Definition

Mapping μ_1 is *subsumed* by μ_2, denoted by $\mu_1 \sqsubseteq \mu_2$, if μ_2 extends μ_1 with more variable bindings

Definition

$\mathcal{T}_1$ is *subsumed* by $\mathcal{T}_2$, denoted by $\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$, if for ever G

- every mapping in the evaluation of $\mathcal{T}_1$ over G, is subsumed by a mapping in the evaluation of $\mathcal{T}_2$ over G

$\sqsubseteq$ is a natural notion of *containment* for SPARQL

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

Subsumption is Π_2^P-complete

Theorem

$\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ iff for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$, and a homomorphism from triples in $\mathcal{T}_2'$ to triples in $\mathcal{T}_1'$ (which is the identity over variables in $\mathcal{T}_1'$)

Subsumption is Π_2^P-complete

Theorem

$\mathcal{T}_1 \sqsubseteq \mathcal{T}_2$ iff for every subtree $\mathcal{T}_1'$ of $\mathcal{T}_1$ there exists a subtree $\mathcal{T}_2'$ of $\mathcal{T}_2$, and a homomorphism from triples in $\mathcal{T}_2'$ to triples in $\mathcal{T}_1'$ (which is the identity over variables in $\mathcal{T}_1'$)

Theorem

Checking subsumption of QWDPTs is Π_2^P-complete

Equivalence & subsumption

Pattern trees $\mathcal{T}_1$ and $\mathcal{T}_2$ are equivalent, denoted by $\mathcal{T}_1 \equiv \mathcal{T}_2$ if they give the same result evaluated over every RDF graph

Proposition

For QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$

$$\mathcal{T}_1 \equiv \mathcal{T}_2 \quad \textit{iff} \quad \mathcal{T}_1 \sqsubseteq \mathcal{T}_2 \textit{ and } \mathcal{T}_2 \sqsubseteq \mathcal{T}_1.$$

Equivalence & subsumption

Pattern trees $\mathcal{T}_1$ and $\mathcal{T}_2$ are equivalent, denoted by $\mathcal{T}_1 \equiv \mathcal{T}_2$ if they give the same result evaluated over every RDF graph

Proposition

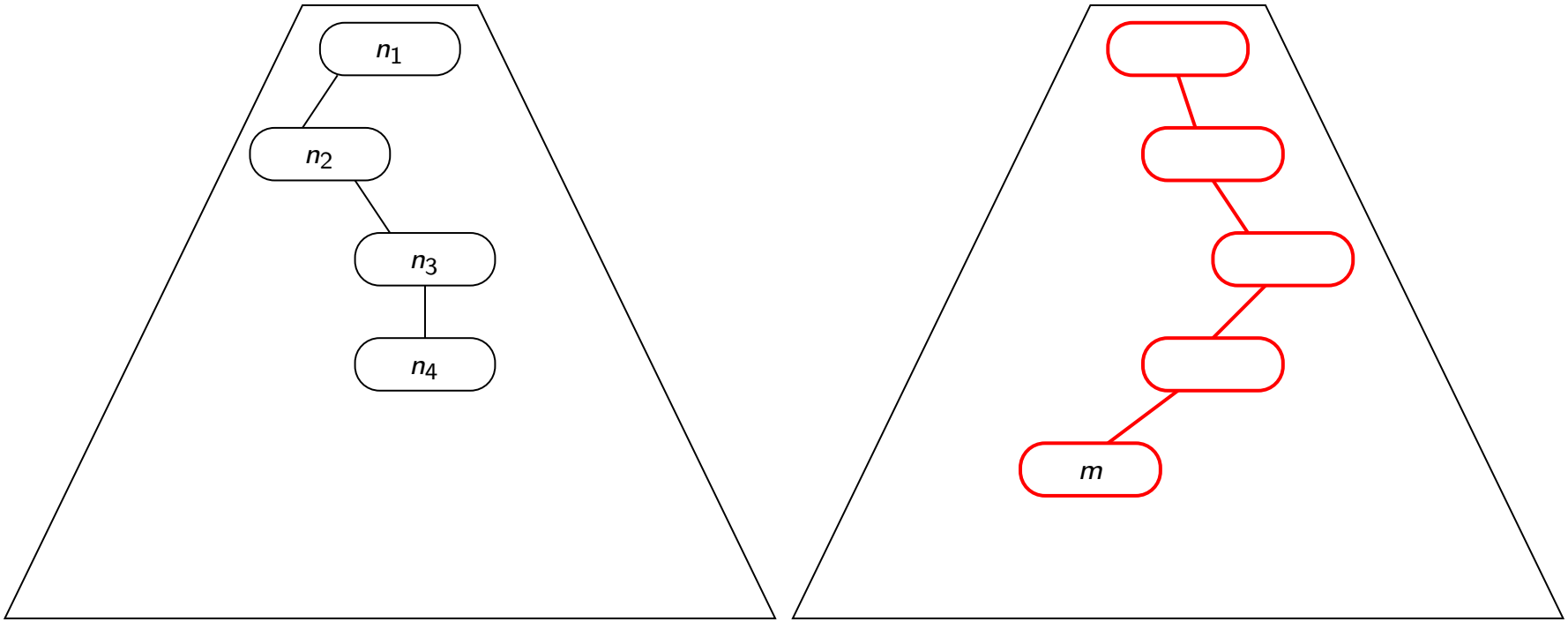
For QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$

$$\mathcal{T}_1 \equiv \mathcal{T}_2 \quad \textit{iff} \quad \mathcal{T}_1 \sqsubseteq \mathcal{T}_2 \textit{ and } \mathcal{T}_2 \sqsubseteq \mathcal{T}_1.$$

This gives us with a Π_2^P equivalence test, but we can do better!

Strong homomorphisms: key concept for equivalence test

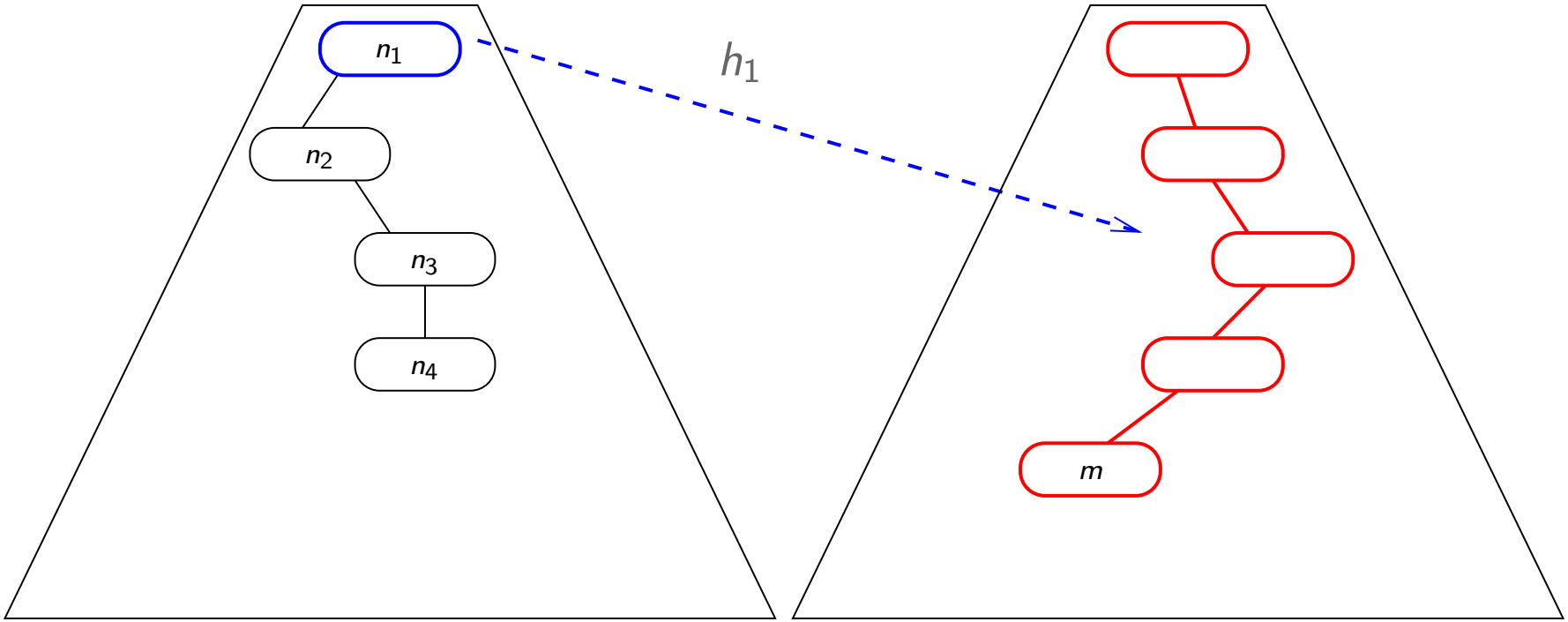
Strong homomorphism among branches: a set of homomorphisms



h_i is the identity over variables not introduced in n_i

Strong homomorphisms: key concept for equivalence test

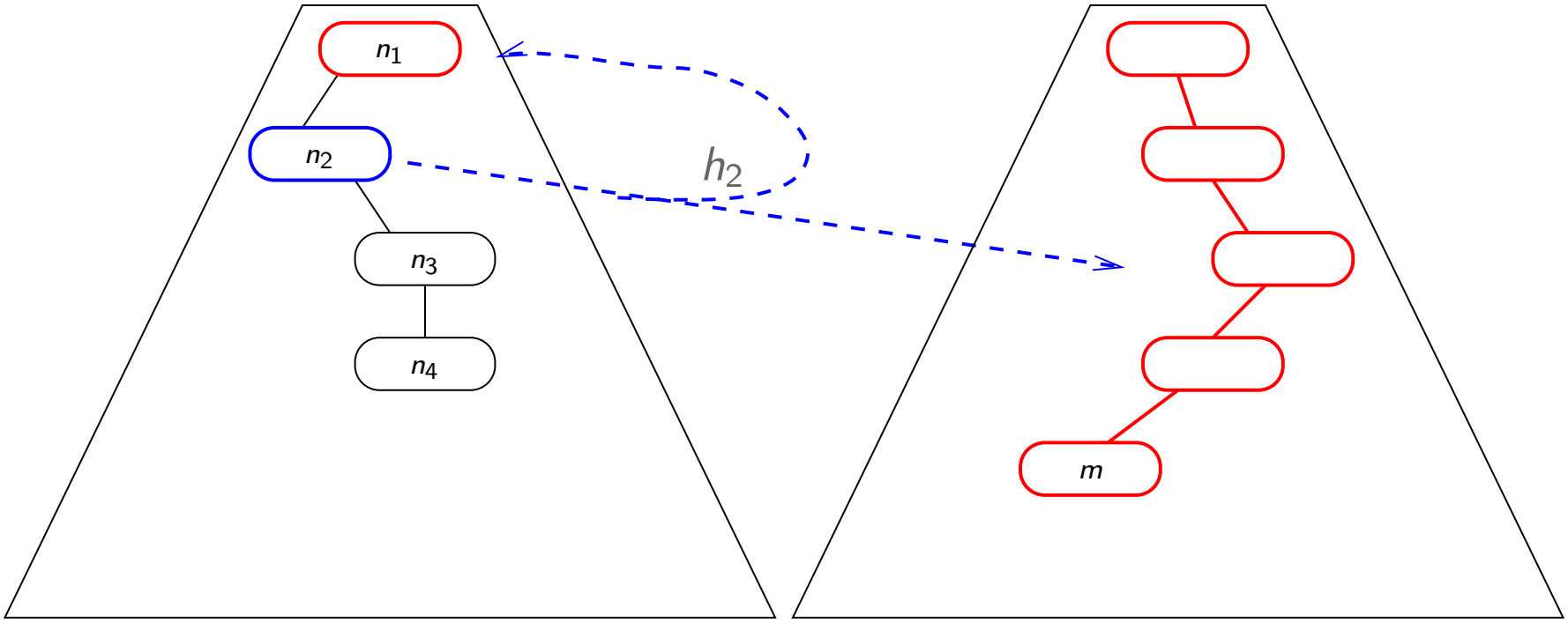
Strong homomorphism among branches: a set of homomorphisms



h_i is the identity over variables not introduced in n_i

Strong homomorphisms: key concept for equivalence test

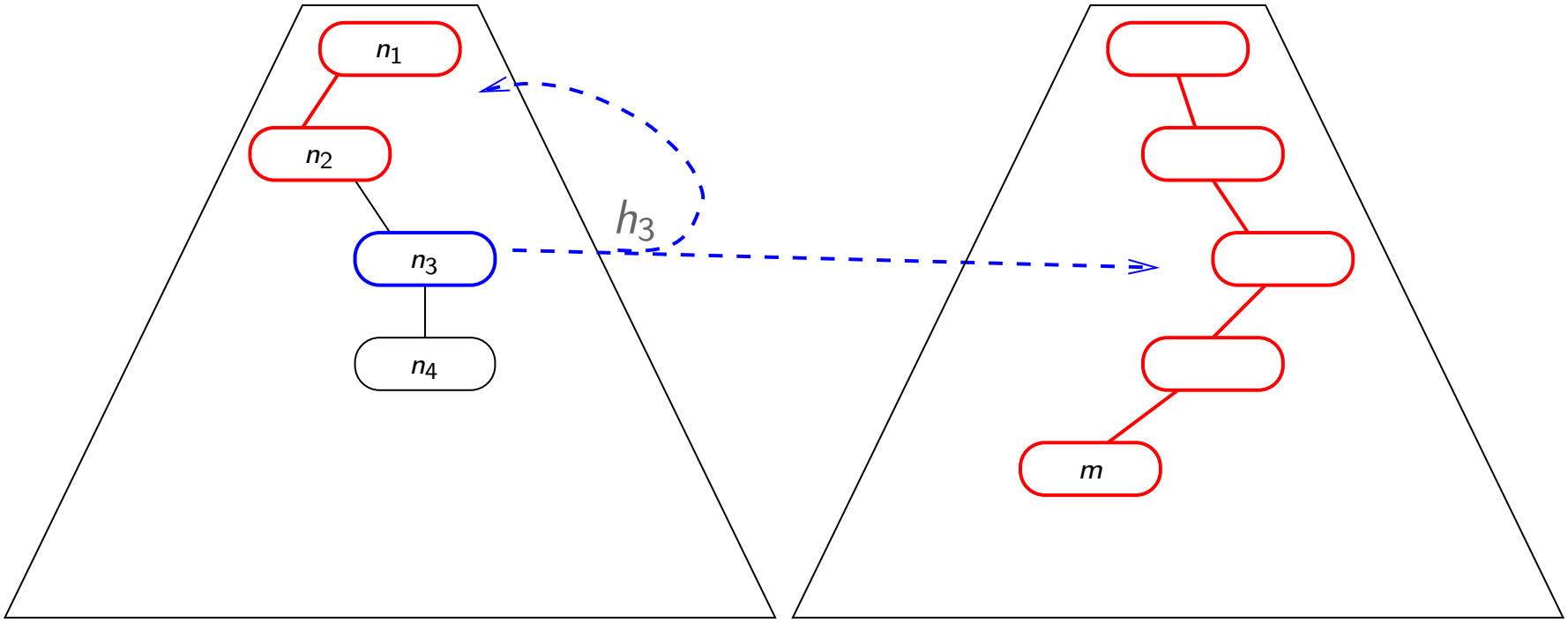
Strong homomorphism among branches: a set of homomorphisms



h_i is the identity over variables not introduced in n_i

Strong homomorphisms: key concept for equivalence test

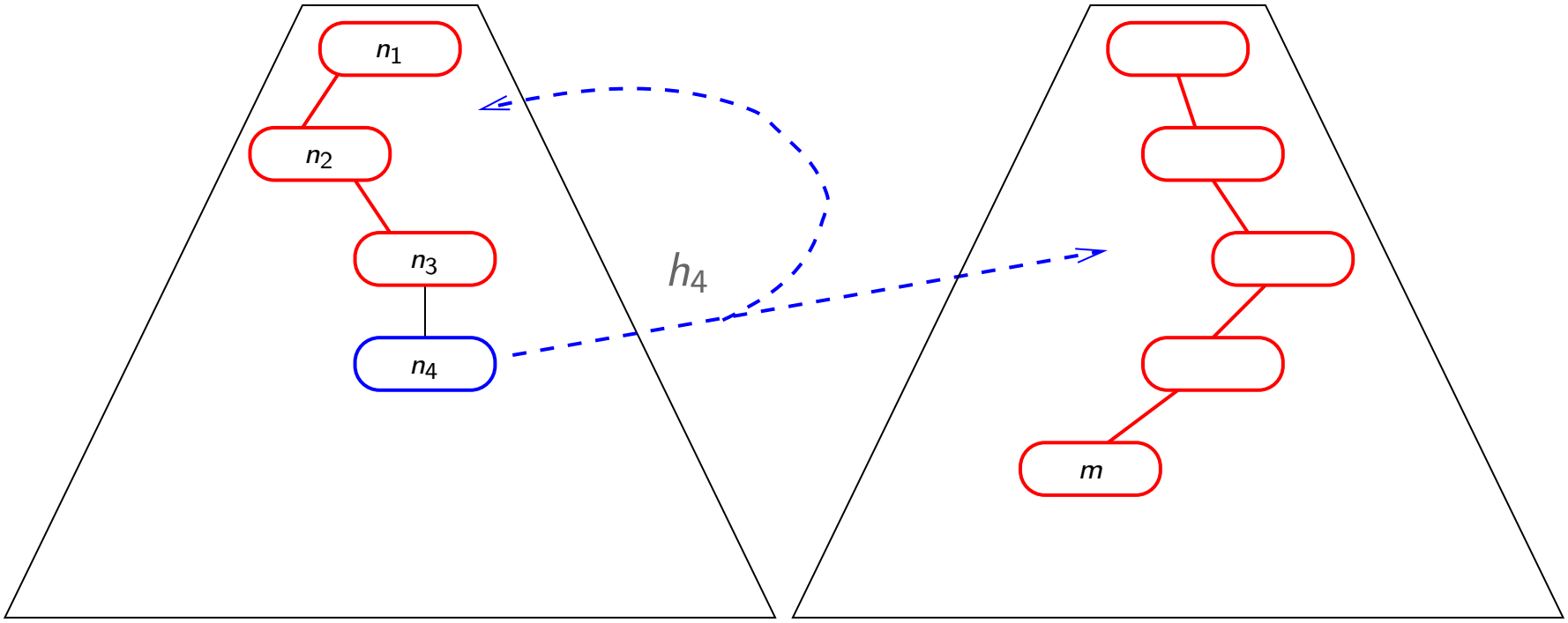
Strong homomorphism among branches: a set of homomorphisms



h_i is the identity over variables not introduced in n_i

Strong homomorphisms: key concept for equivalence test

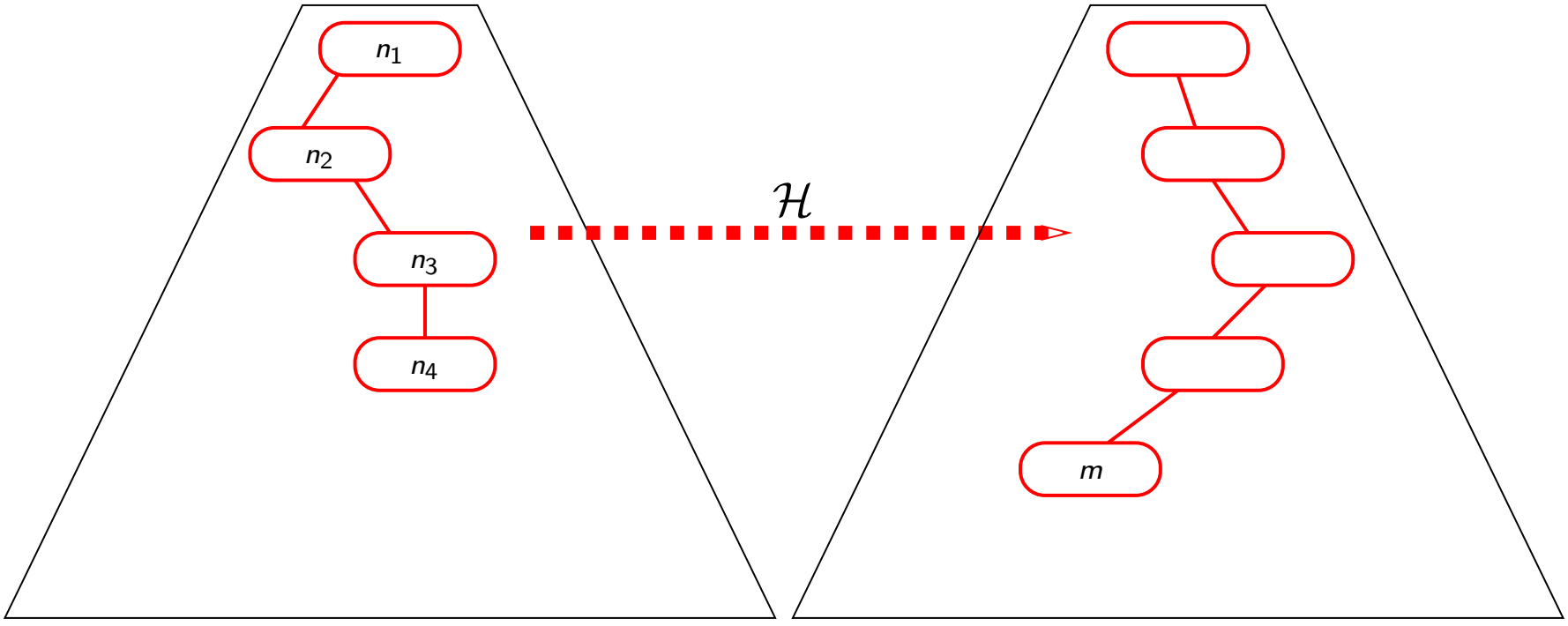
Strong homomorphism among branches: a set of homomorphisms



h_i is the identity over variables not introduced in n_i

Strong homomorphisms: key concept for equivalence test

Strong homomorphism among branches: a set of homomorphisms

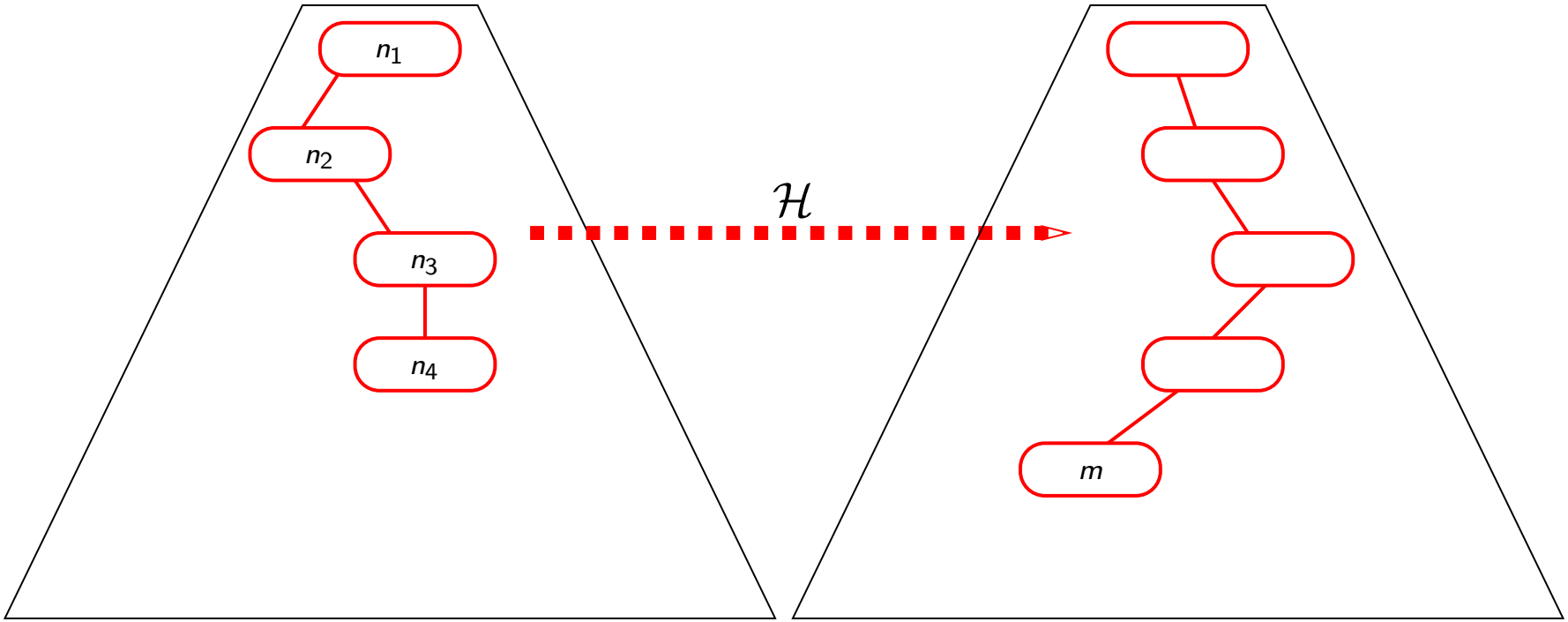


h_i is the identity over variables not introduced in n_i

$\mathcal{H} = \{h_1, h_2, h_3, h_4\}$ is a strong homorphism
$\mathcal{H} : \text{branch}(n_4) \rightarrow \text{branch}(m)$.

Strong homomorphisms: key concept for equivalence test

Strong homomorphism among branches: a set of homomorphisms



h_i is the identity over variables not introduced in n_i

$\mathcal{H} = \{h_1, h_2, h_3, h_4\}$ is a strong homorphism
$\mathcal{H} : \text{branch}(n_4) \rightarrow \text{branch}(m)$.

Strong homomorphisms in both direction:

strongly homomorphically equivalent branches

Equivalence is NP-complete

Theorem

QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$ in R3-normal form *are equivalent iff*

- *$\mathcal{T}_1$ and $\mathcal{T}_2$ have the same triple patterns, the same root, and*
- *if n_1 in $\mathcal{T}_1$ and n_2 in $\mathcal{T}_2$* introduce the same variable, *then* $\mathsf{branch}(n_1)$ *and* $\mathsf{branch}(n_2)$ *are strongly hom. equiv.*

Equivalence is NP-complete

Theorem

QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$ in R3-normal form *are equivalent iff*

- *$\mathcal{T}_1$ and $\mathcal{T}_2$ have the same triple patterns, the same root, and*
- *if n_1 in $\mathcal{T}_1$ and n_2 in $\mathcal{T}_2$* introduce the same variable, *then* $\mathsf{branch}(n_1)$ *and* $\mathsf{branch}(n_2)$ *are strongly hom. equiv.*

Key for *NP*-test: R3-normal form does not need to be computed!
It is enough to guess a polynomial number of applications of R3

Equivalence is NP-complete

Theorem

QWDPTs $\mathcal{T}_1$ and $\mathcal{T}_2$ in R3-normal form *are equivalent iff*

- *$\mathcal{T}_1$ and $\mathcal{T}_2$ have the same triple patterns, the same root, and*
- *if n_1 in $\mathcal{T}_1$ and n_2 in $\mathcal{T}_2$* introduce the same variable, *then* branch(n_1) *and* branch(n_2) *are strongly hom. equiv.*

Key for *NP*-test: R3-normal form does not need to be computed!
It is enough to guess a polynomial number of applications of R3

Theorem

Checking equivalence of QWDPTs is NP-complete

Outline

CQ-tractable cases to tractable SPARQL

Each node of a QWDPT is a CQ $\Rightarrow$ natural question:

how do tractable fragments of CQ carry over?

CQ-tractable cases to tractable SPARQL

Each node of a QWDPT is a CQ $\Rightarrow$ natural question:

how do tractable fragments of CQ carry over?

- Evaluation:

CQ-tractable cases to tractable SPARQL

Each node of a QWDPT is a CQ $\Rightarrow$ natural question:

how do tractable fragments of CQ carry over?

- Evaluation: *yes!* (straightforward)

CQ-tractable cases to tractable SPARQL

Each node of a QWDPT is a CQ $\Rightarrow$ natural question:

how do tractable fragments of CQ carry over?

- Evaluation: *yes!* (straightforward)
- Polynomial delay enumeration:

CQ-tractable cases to tractable SPARQL

Each node of a QWDPT is a CQ $\Rightarrow$ natural question:

how do tractable fragments of CQ carry over?

- Evaluation: *yes!* (straightforward)
- Polynomial delay enumeration: *yes!* (not so straightforward)

CQ-tractable cases to tractable SPARQL

Each node of a QWDPT is a CQ $\Rightarrow$ natural question:

how do tractable fragments of CQ carry over?

- Evaluation: *yes!* (straightforward)
- Polynomial delay enumeration: *yes!* (not so straightforward)
- Counting:

CQ-tractable cases to tractable SPARQL

Each node of a QWDPT is a CQ $\Rightarrow$ natural question:

how do tractable fragments of CQ carry over?

- Evaluation: *yes!* (straightforward)
- Polynomial delay enumeration: *yes!* (not so straightforward)
- Counting: *no* :-(

CQ-tractable cases to tractable SPARQL

Each node of a QWDPT is a CQ $\Rightarrow$ natural question:

how do tractable fragments of CQ carry over?

- Evaluation: *yes!* (straightforward)
- Polynomial delay enumeration: *yes!* (not so straightforward)
- Counting: *no* :-(

  $\#\cdot coNP$ in general

  $\#P$-hard even if every node is an acyclic CQ

Concluding remarks

Evaluation of SPARQL graph patterns via pattern trees

- abstract representation for well-designed SPARQL
- resemble relational query plans (+ transformation rules)
- help in studying static analysis (equivalence, subsumption)

Concluding remarks

Evaluation of SPARQL graph patterns via pattern trees

- abstract representation for well-designed SPARQL
- resemble relational query plans (+ transformation rules)
- help in studying static analysis (equivalence, subsumption)

Future work

- include more features (projection, union)
- more transformation rules
- tests and empirical results

Static Analysis and Optimization of Semantic Web Queries

Andrés Letelier Jorge Pérez Reinhard Pichler Sebastian Skritek

PUC Chile Universidad de Chile TU Vienna