Automated Termination Proving: Contributions to Graph Rewriting via Extended Weighted Type Graphs and Morphism Counting

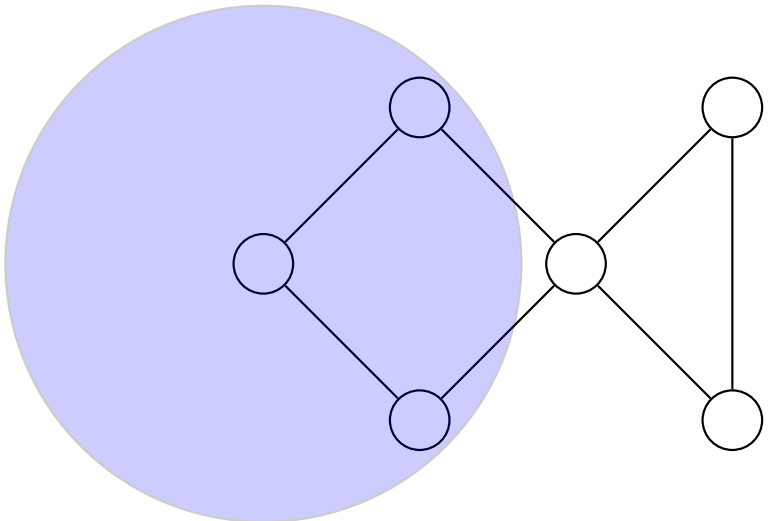
Qi QIU

Supervisor: Xavier URBAIN
LIRIS, UMR 5205 CNRS
Université Claude Bernard Lyon 1, France

Context and Objective of the Thesis

- Ubiquity of distributed and concurrent algorithms
  - Medical devices, transportation systems, ...
- Severe consequences of errors
  - Threats to human life, data loss, ...
- Ensuring correct behaviour of such systems is challenging
- Automated verification methods
  - Mathematically rigorous methods
  - Require little user intervention

Graph Transformation

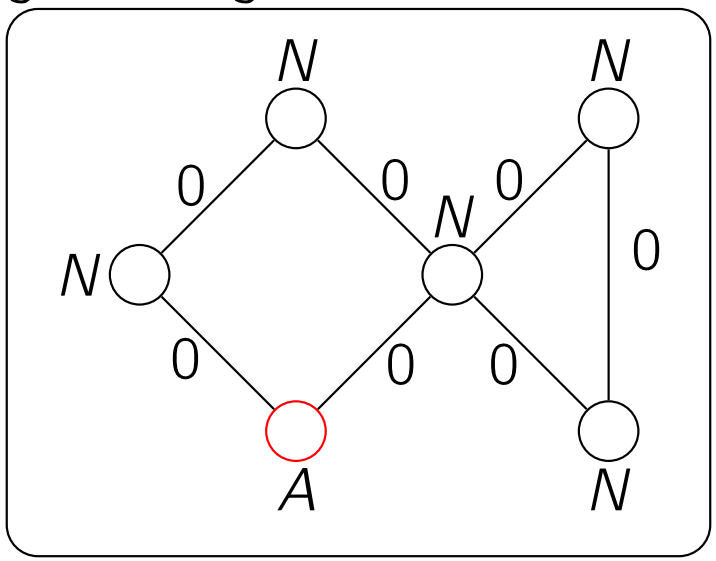


Graph rewriting:

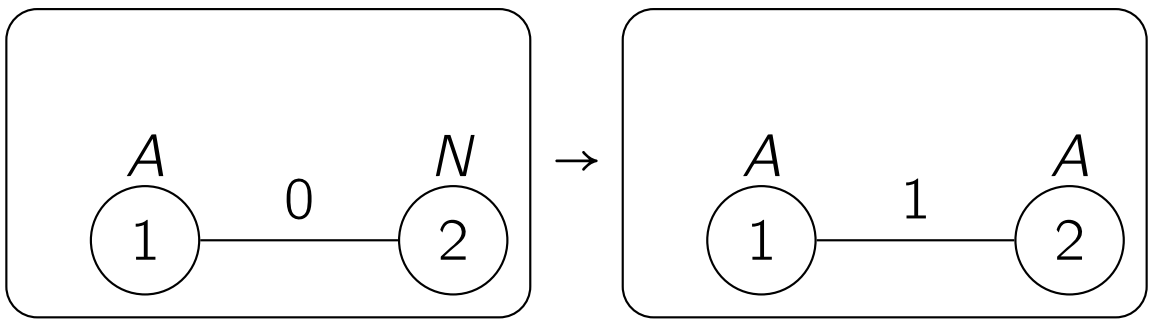
- computational units → nodes
- communication channels → edges
- system states → graphs
- algorithm behaviors → graph transformation rules that depend on the states of neighboring nodes and edges.

Graph Transformation

Graph representing a configuration of a distributed system:

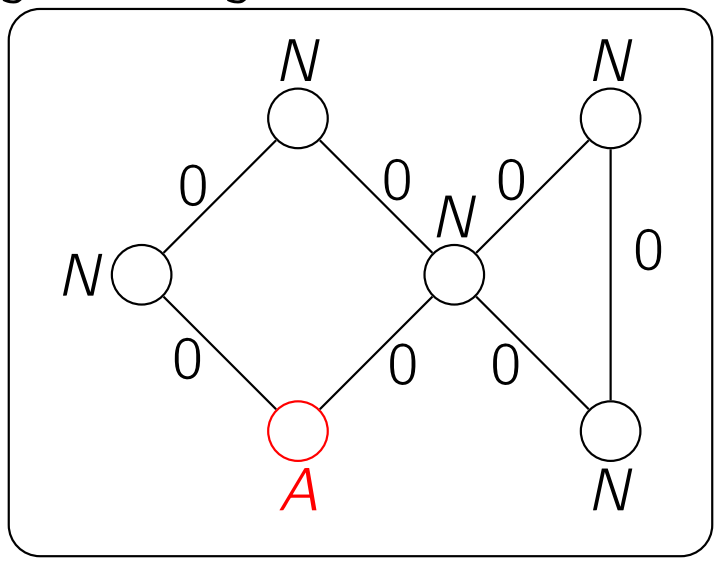


Graph transformation rule:

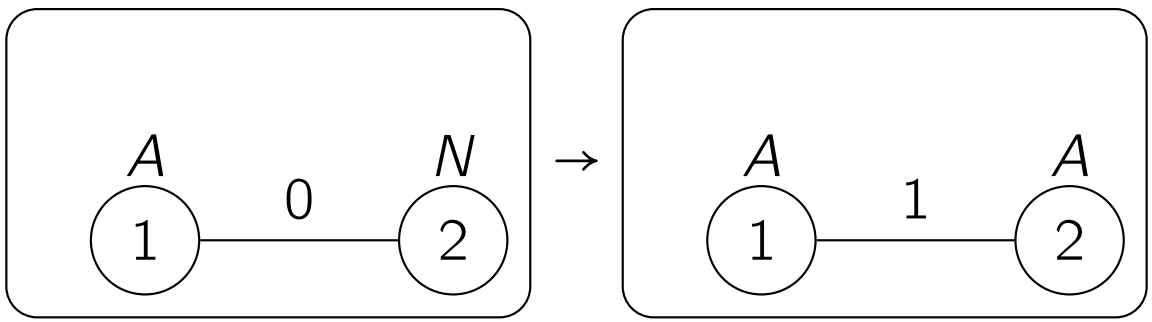


Graph Transformation

Graph representing a configuration of a distributed system:

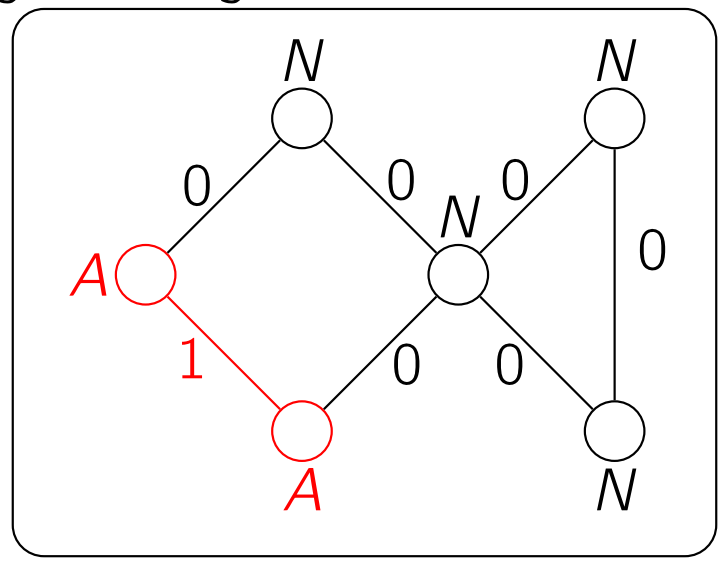


Graph transformation rule:

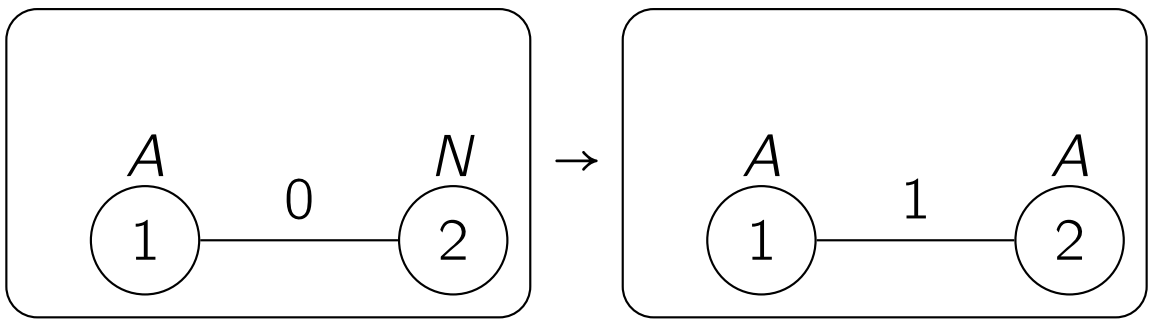


Graph Transformation

Graph representing a configuration of a distributed system:

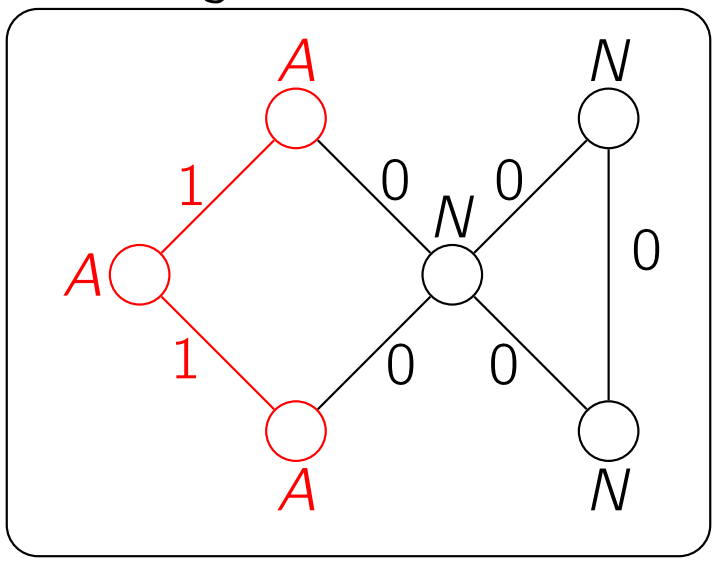


Graph transformation rule:

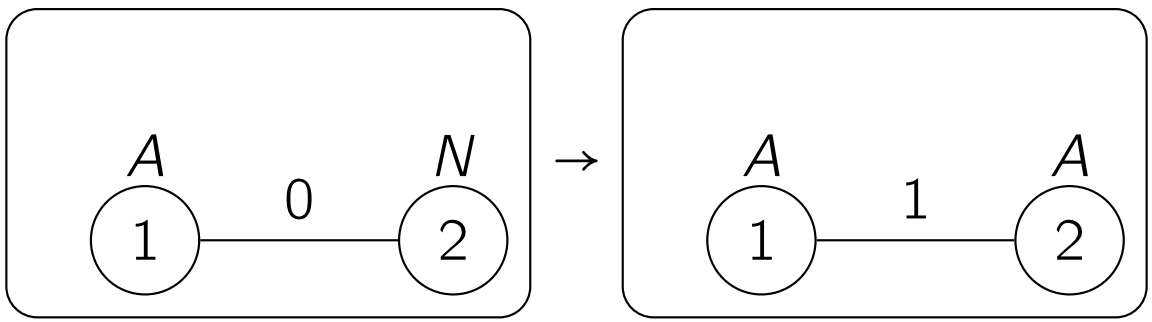


Graph Transformation

Graph representing a configuration of a distributed system:

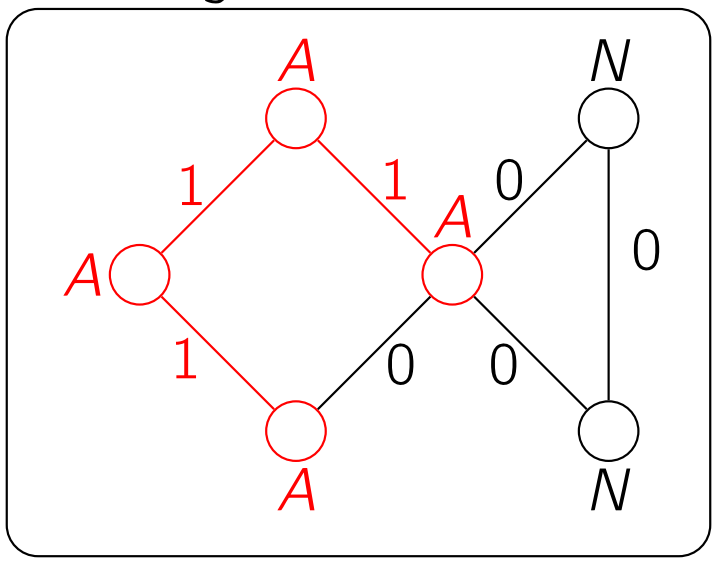


Graph transformation rule:

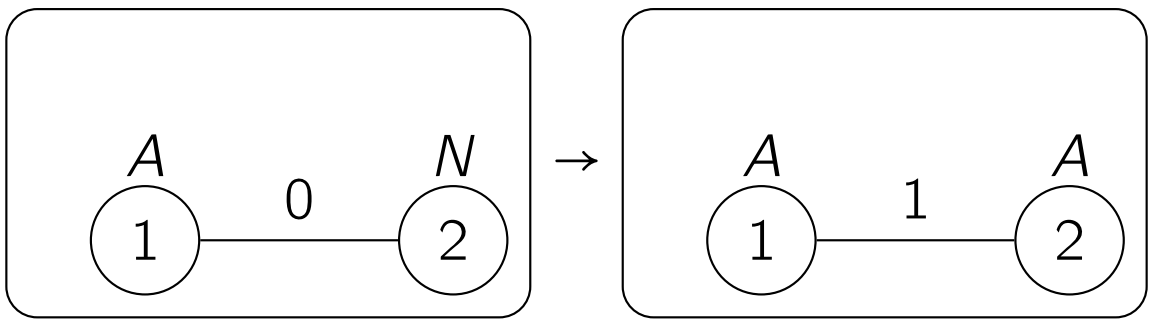


Graph Transformation

Graph representing a configuration of a distributed system:

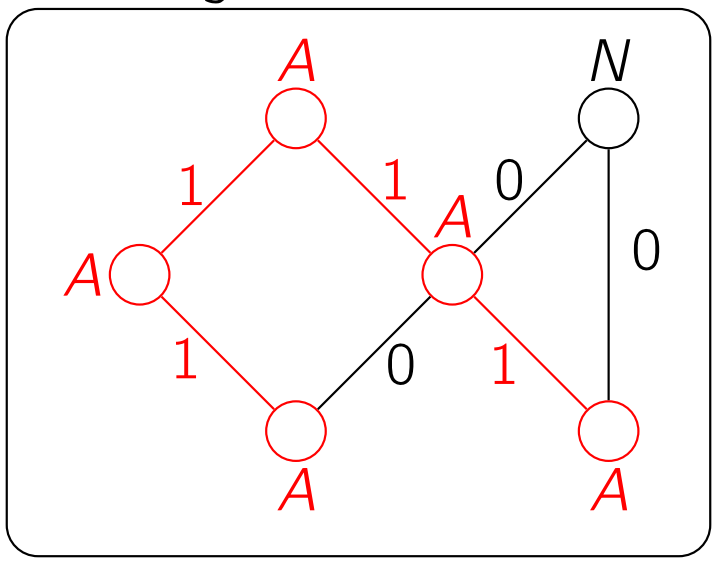


Graph transformation rule:

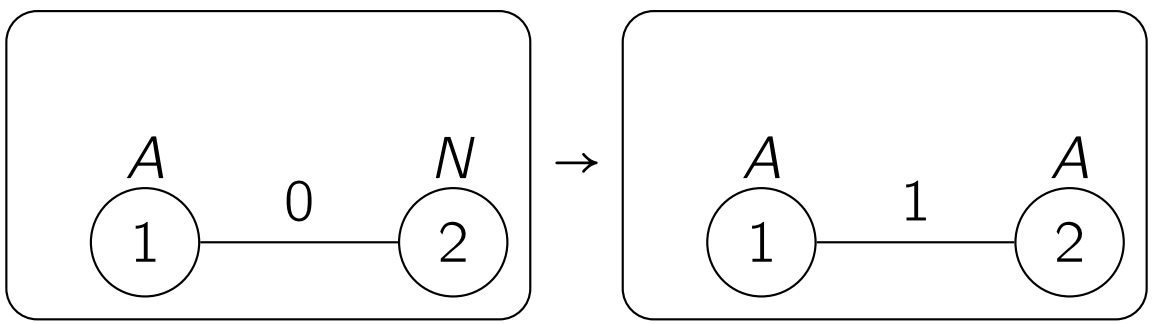


Graph Transformation

Graph representing a configuration of a distributed system:

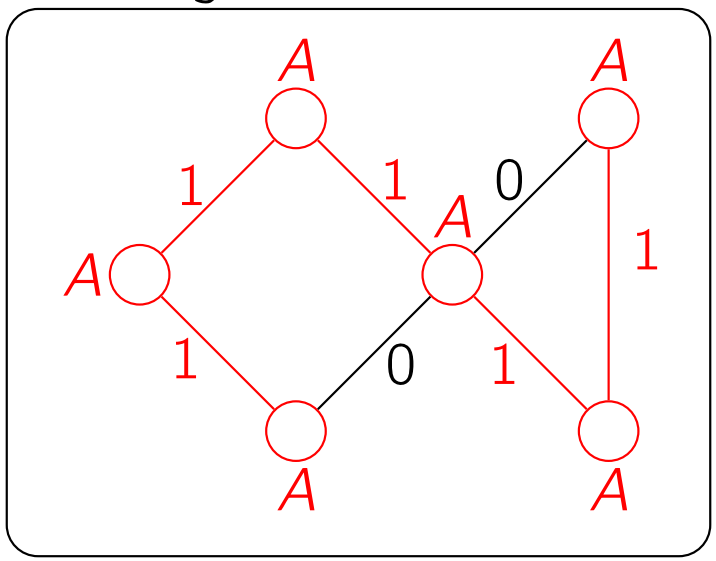


Graph transformation rule:

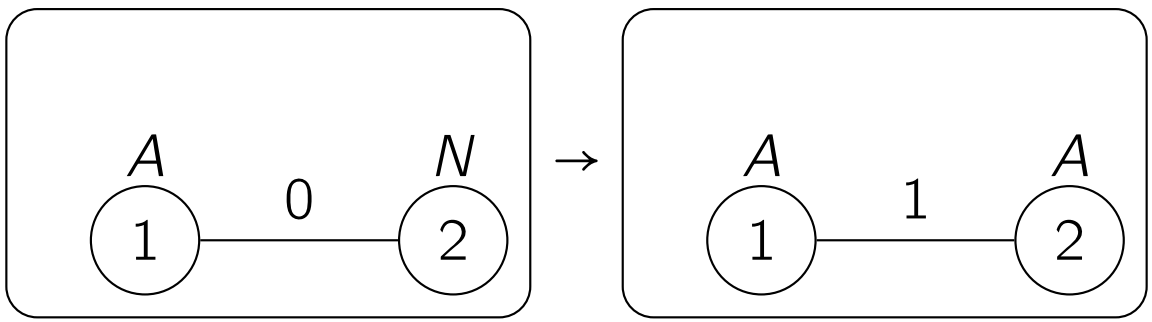


Graph Transformation

Graph representing a configuration of a distributed system:

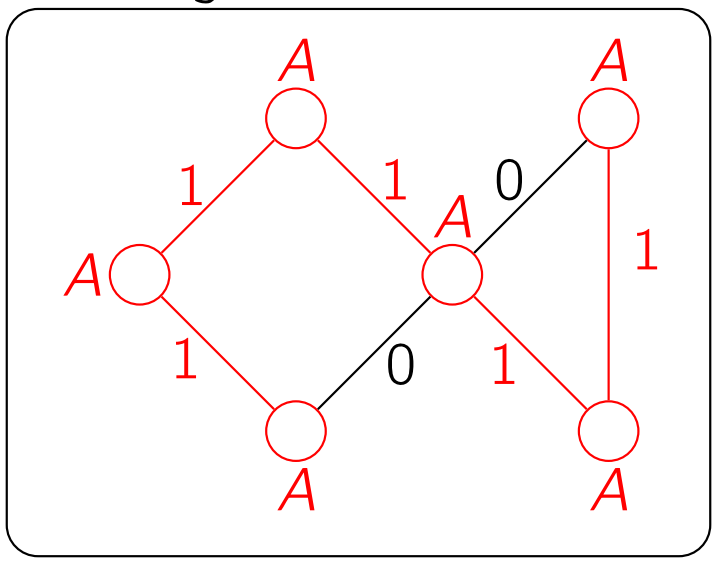


Graph transformation rule:

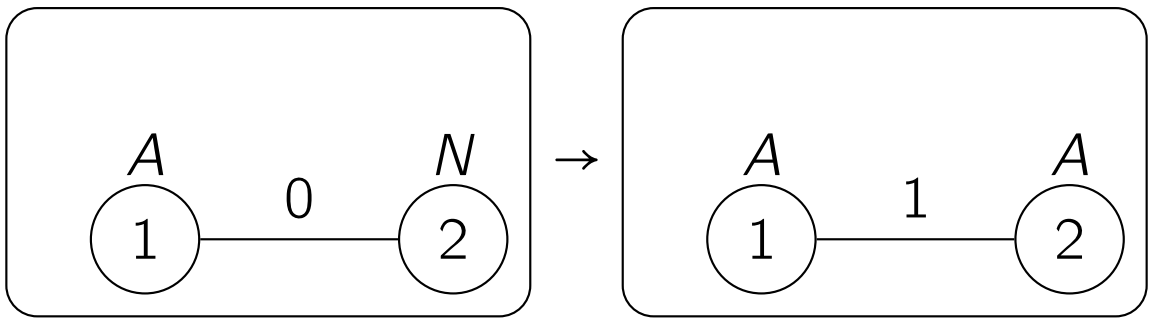


Graph Transformation

Graph representing a configuration of a distributed system:



Graph transformation rule:



Question: does the transformation process terminate from any initial configuration?

Termination of graph transformation systems

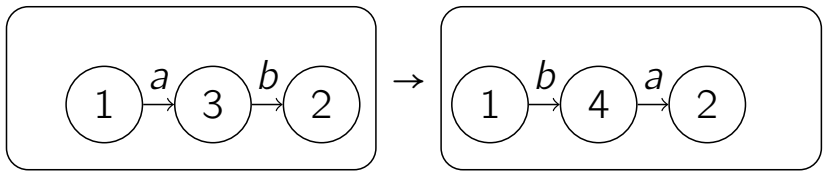
- $\mathcal{R}$: a set of rules
- No graph can be transformed forever when using the non-deterministic strategy

  "apply rules as long as possible"
- Aligns with the standard notion of program termination:

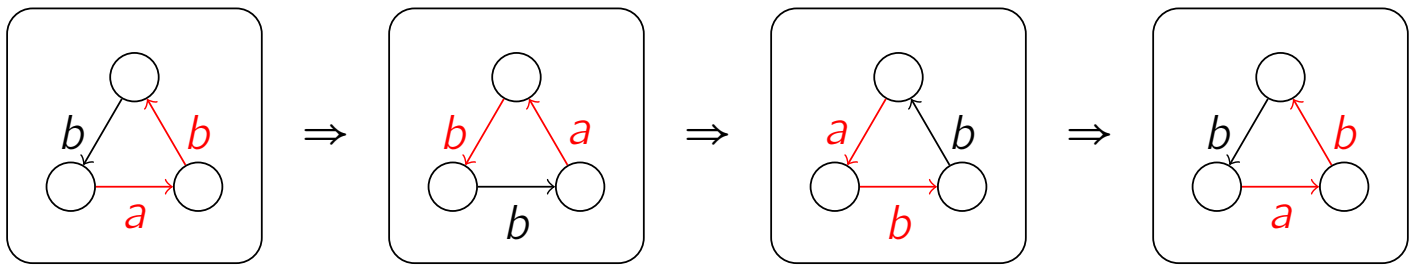
  "every execution (on any input) eventually halts."
- Undecidable in general

One-rule examples of non-termination and termination

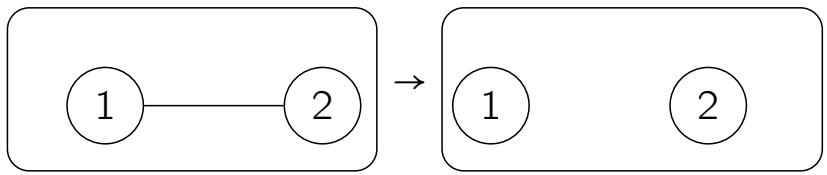
Rule α :



Looping:

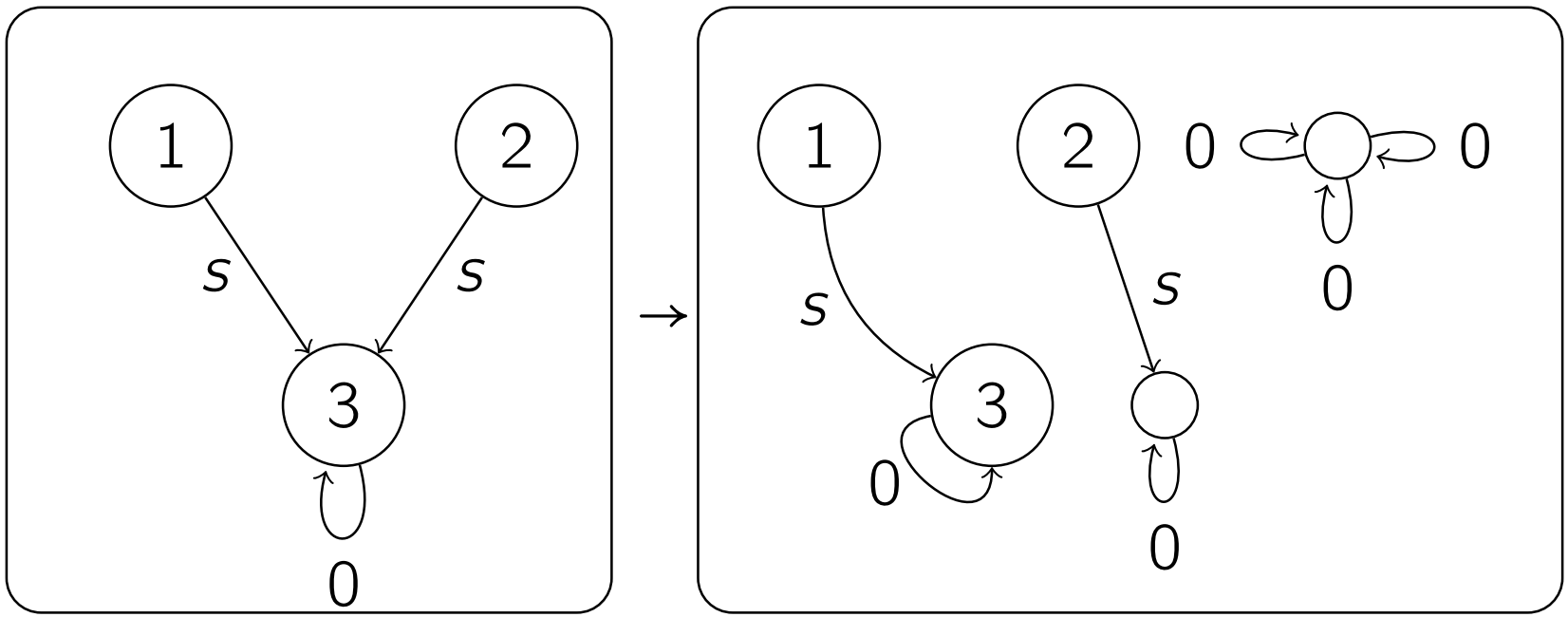


Rule β:



Terminating by the number of edges if we consider finite graphs only.

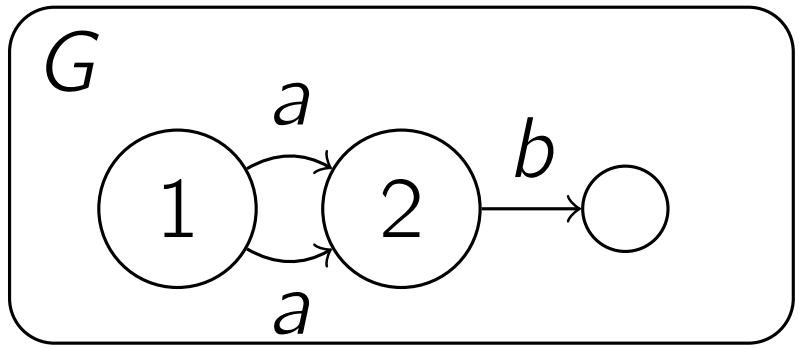
Motivating Rule



Question: does this rule terminate?

Structure of the presentation

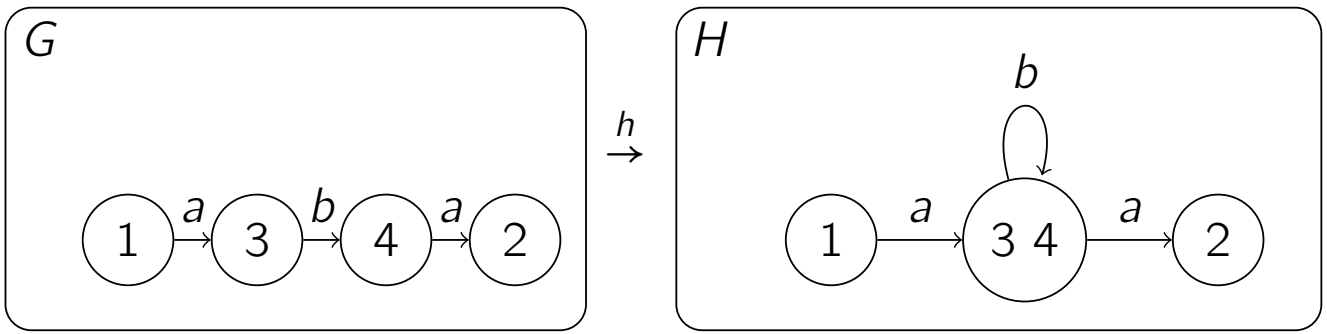
Graphs : finite, directed, edge-labeled multigraphs



Remarks:

- Edges with the same source, target and label are permitted.
- Graph name (G) shown at the top left of the drawing.
- Numbers inside nodes are identifiers (omitted when not relevant).

Graph morphisms: structure-preserving functions



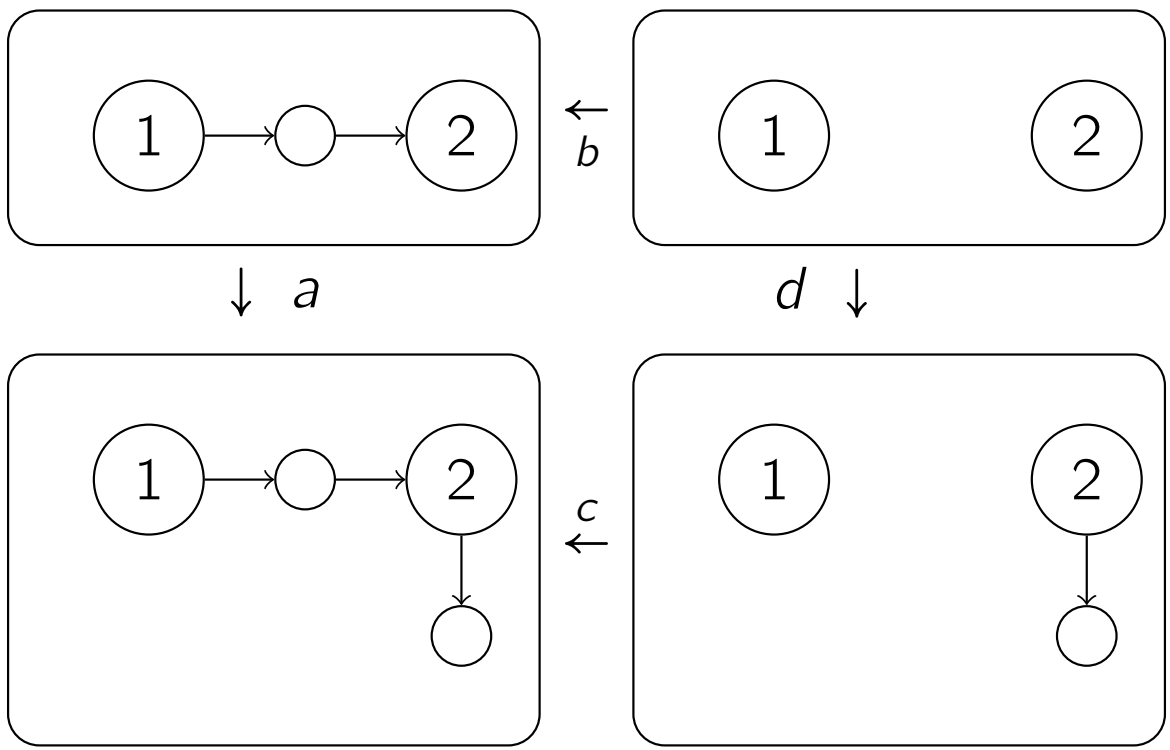
Remarks:

- Nodes of H are labeled with the sets of identifiers of nodes of G that map to them.
- $\overset{h}{\to}$ indicates $h : G \to H$.

Commutative Diagram

$$\begin{array}{ccc} L & \xleftarrow{\quad b \quad} & K \\ \downarrow{\scriptstyle a} & & \downarrow{\scriptstyle d} \\ G & \xleftarrow{\quad c \quad} & C \end{array}$$

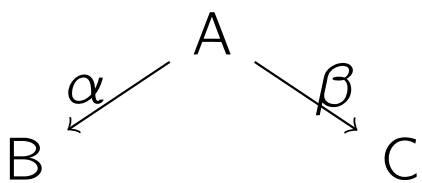
Commutative if $a \circ b = c \circ d$.



Pushouts: gluing graphs along a common part

Definition

The **pushout** of (α, β) is

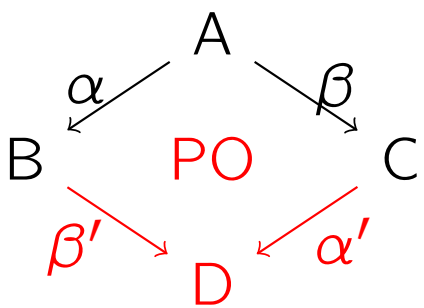


Pushouts: gluing graphs along a common part

Definition

The **pushout** of (α, β) is (β', α') such that

- ABDC is commutative,

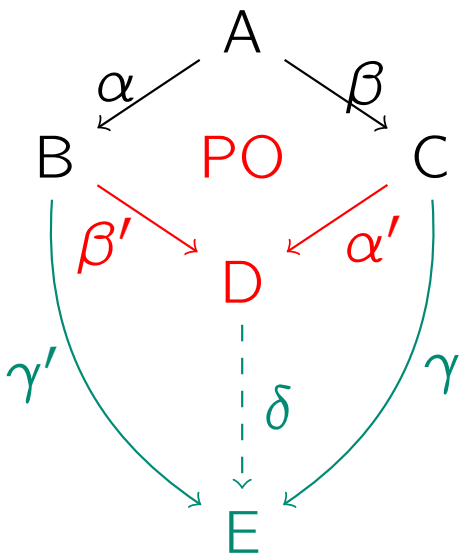


Pushouts: gluing graphs along a common part

Definition

The **pushout** of (α, β) is (β', α') such that

- ABDC is commutative,
- universality: $\forall (\gamma, \gamma')$. ABEC is commutative $\Longrightarrow$ $\exists\,!\,\delta$. BDE & CDE are commutative.

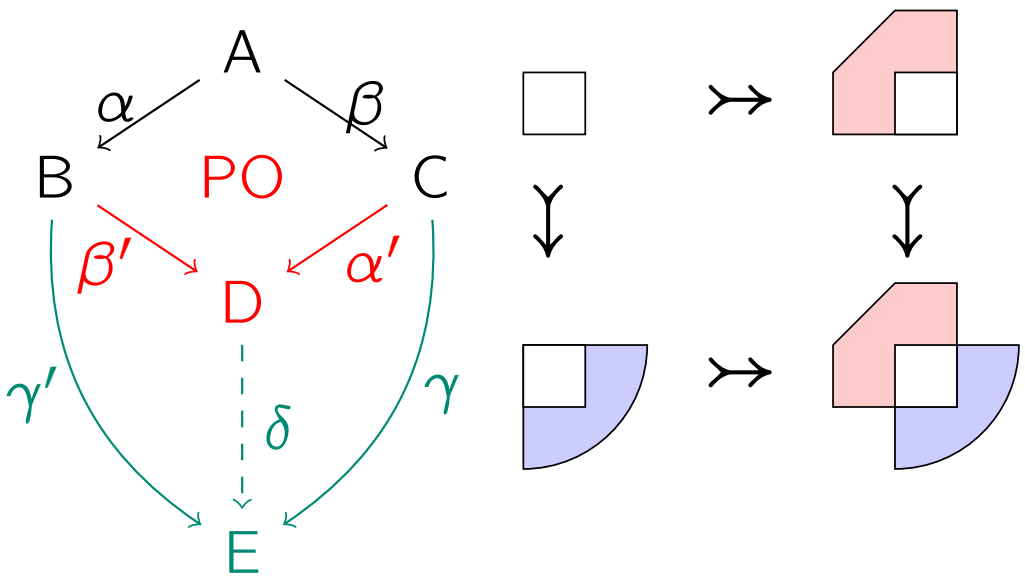


Pushouts: gluing graphs along a common part

Definition

The **pushout** of (α, β) is (β', α') such that

- ABDC is commutative,
- universality: $\forall (\gamma, \gamma')$. ABEC is commutative $\Longrightarrow$ $\exists\,!\,\delta$. BDE & CDE are commutative.

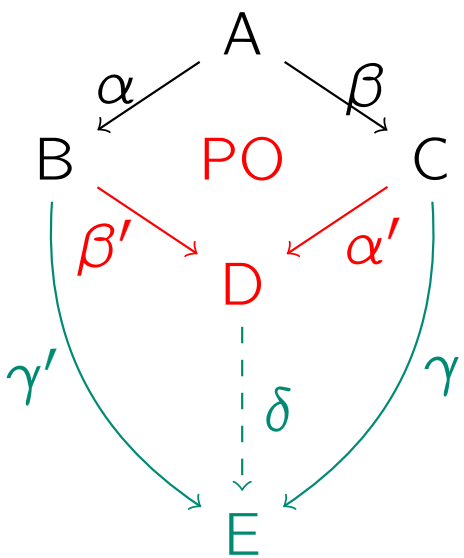


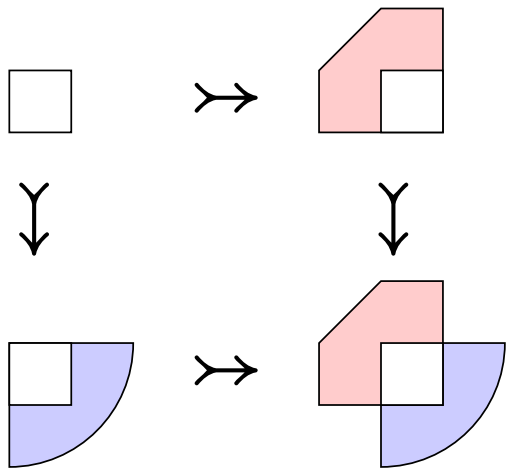
Pushouts: gluing graphs along a common part

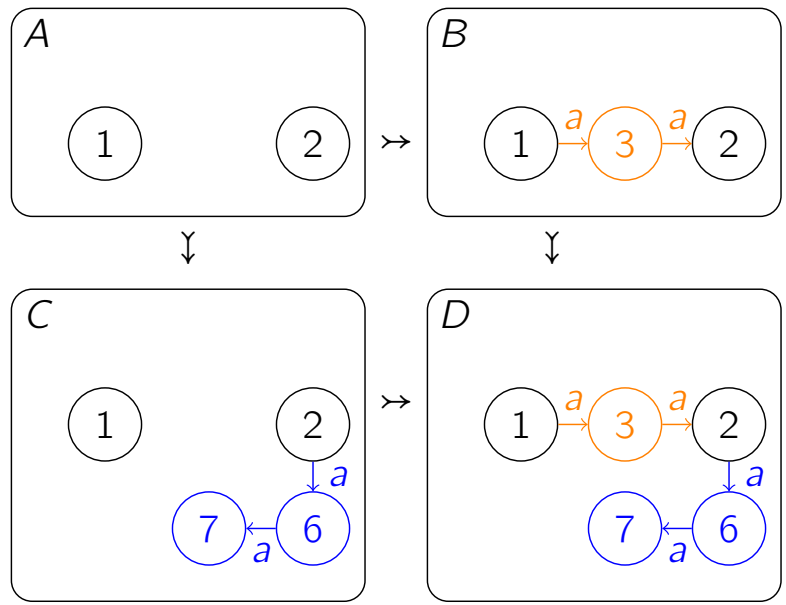
Definition

The **pushout** of (α, β) is (β', α') such that

- ABDC is commutative,
- universality: $\forall (\gamma, \gamma')$. ABEC is commutative $\Longrightarrow$ $\exists\,!\,\delta$. BDE & CDE are commutative.



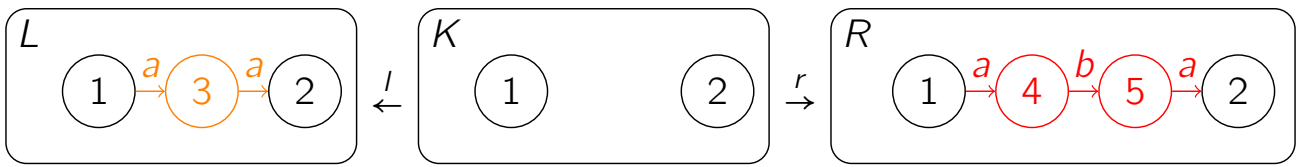




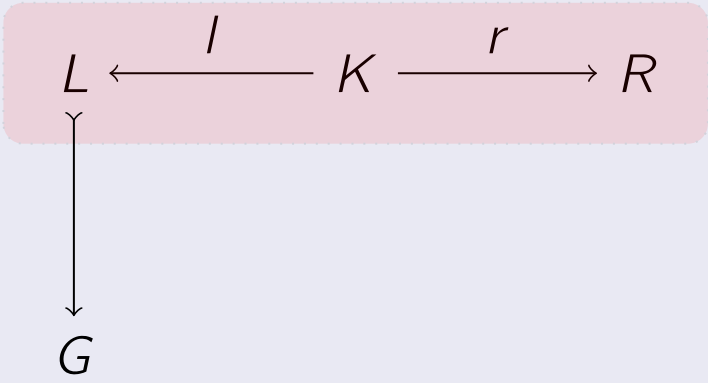
Graph rewriting with double-pushout approach (DPO)

$L \xleftarrow{l} K \xrightarrow{r} R$

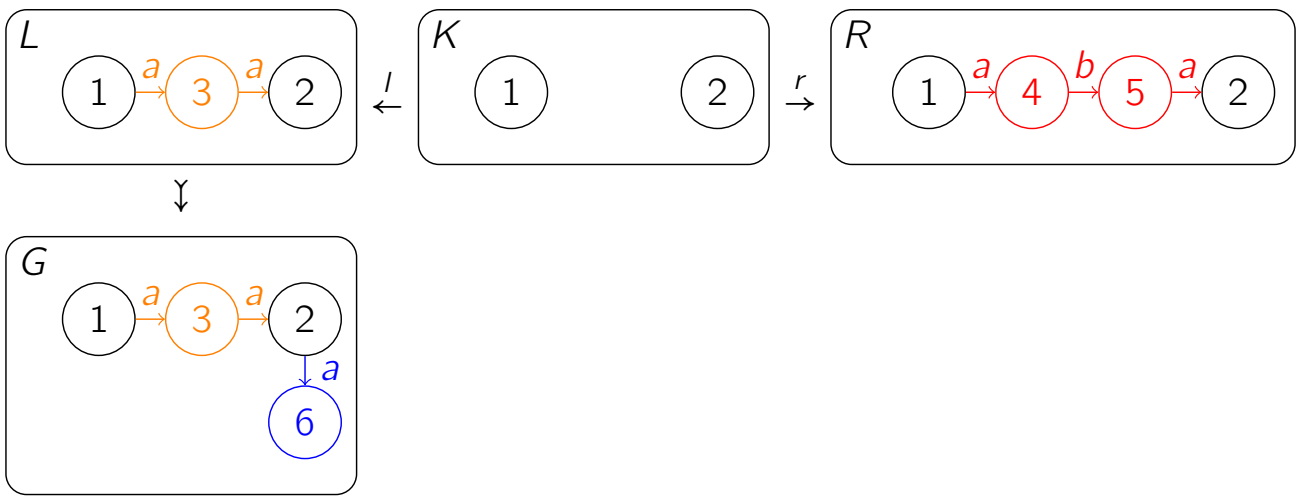
Rewriting rule with interface K



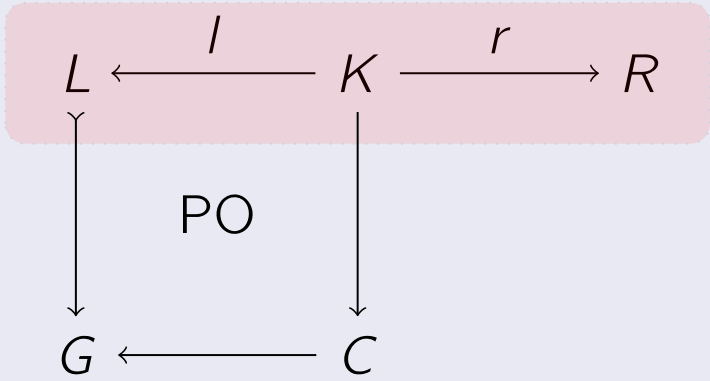
Graph rewriting with double-pushout approach (DPO)



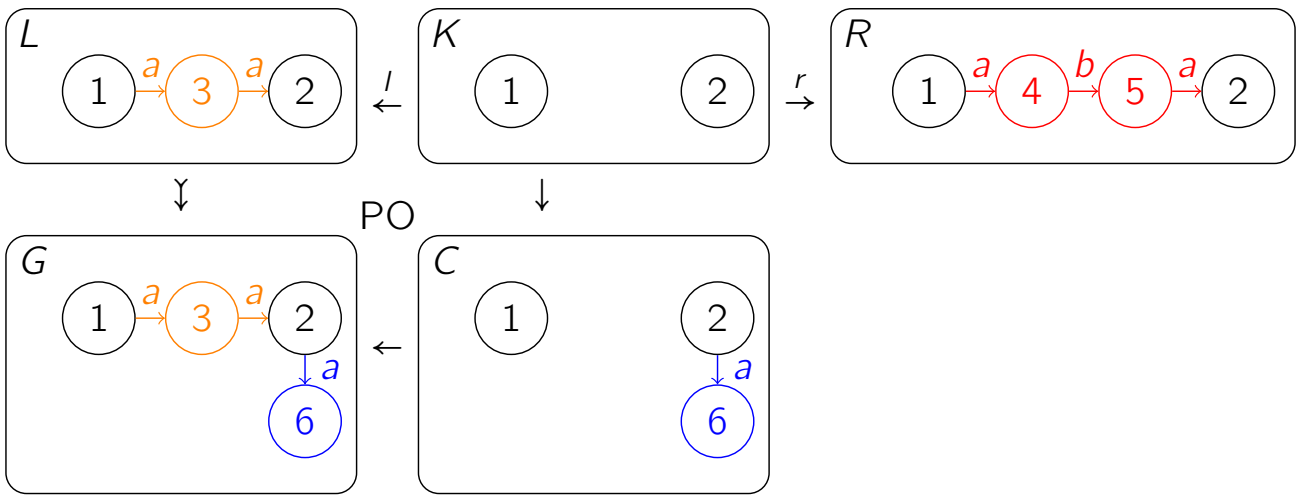
Rewriting rule with interface K



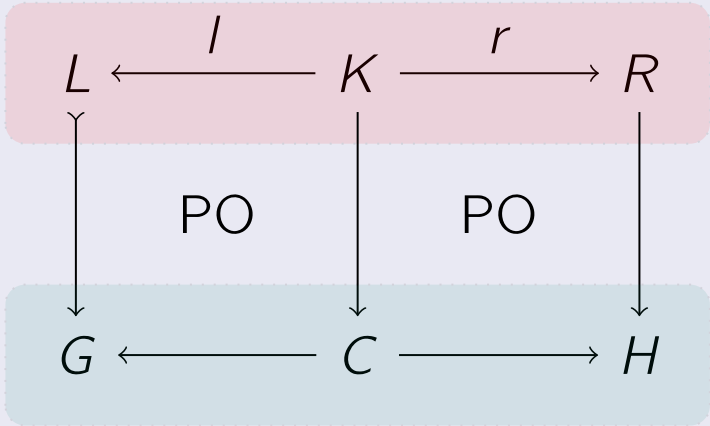
Graph rewriting with double-pushout approach (DPO)



Rewriting rule with interface K

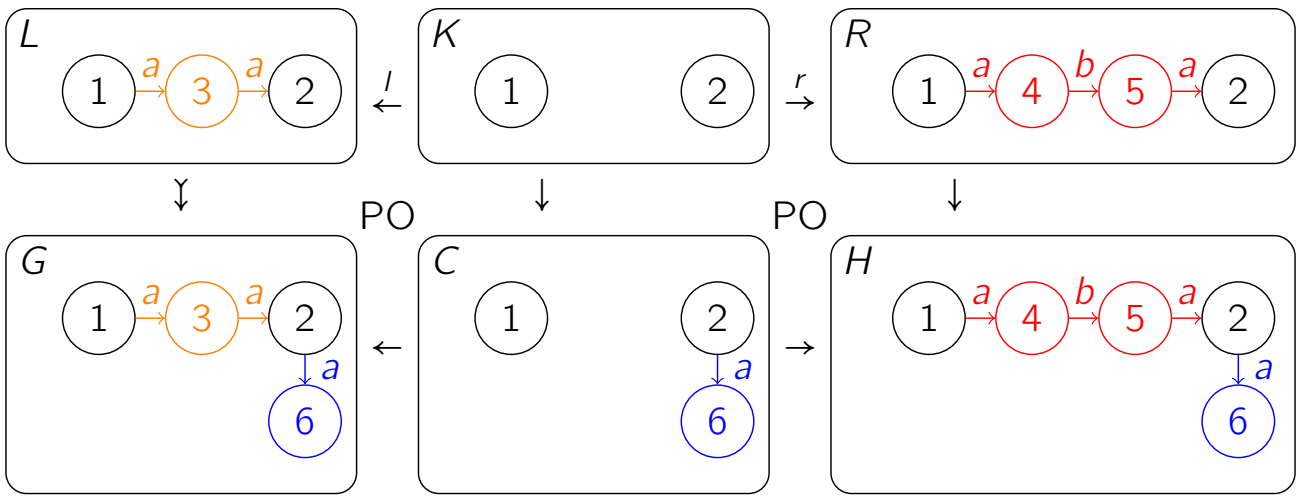


Graph rewriting with double-pushout approach (DPO)



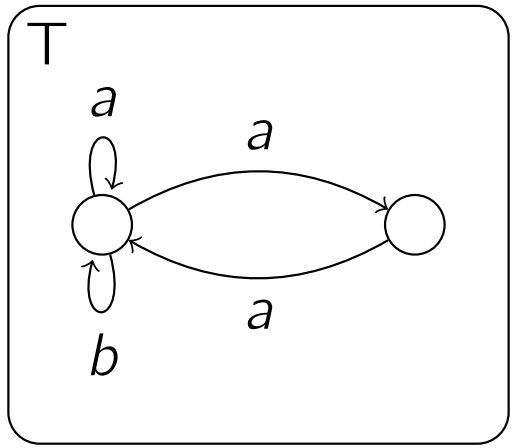
Rewriting rule with interface K

rewriting step $G \Rightarrow H$



Weighted Type Graph

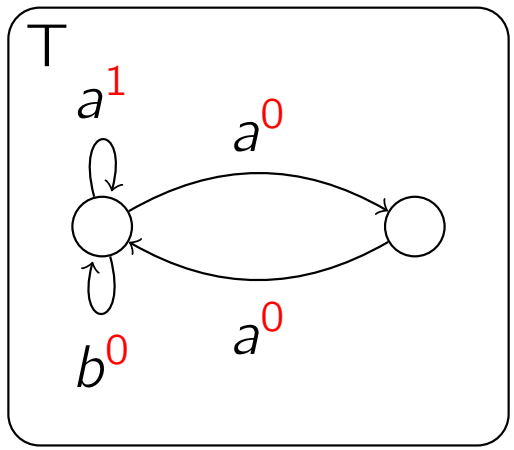
In the context of graph rewriting, a **weighted type graph** is a graph



Weighted Type Graph

In the context of graph rewriting, a **weighted type graph** is a graph

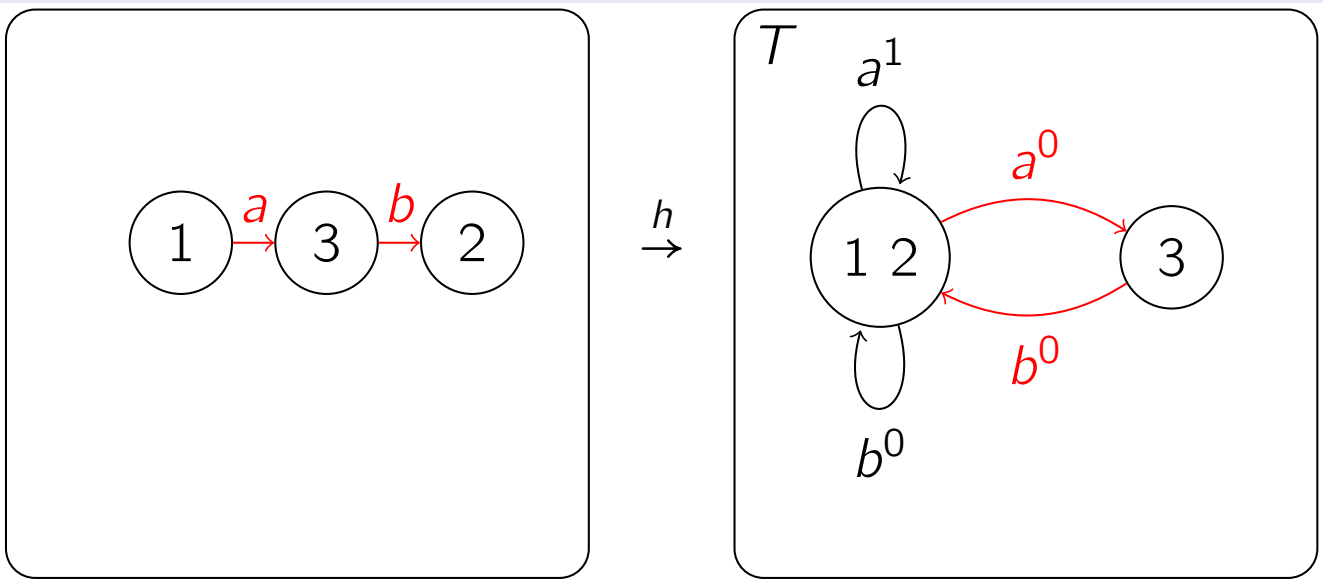
- with weights assigned to its edges.



Morphism weight

The weight of a morphism $h : G \to T$ is

$$\sum_{e \in \mathsf{Edge(G)}} weight(h(e))$$

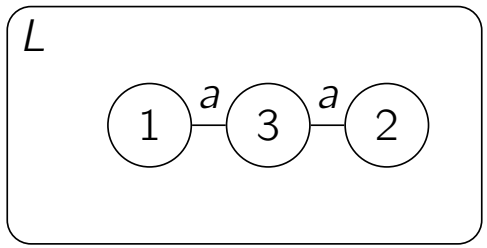


$\mathsf{weight}_T(h) = 0 + 0 = 0$

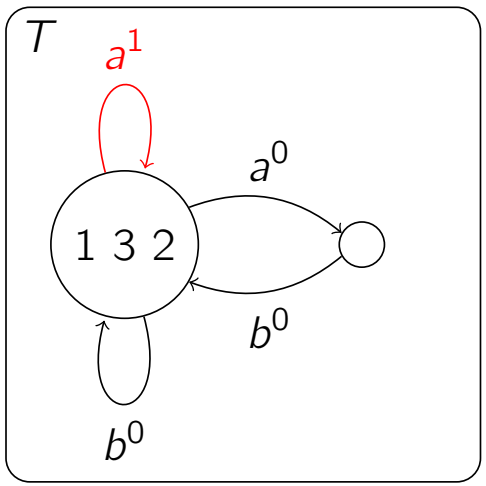
Graph weight

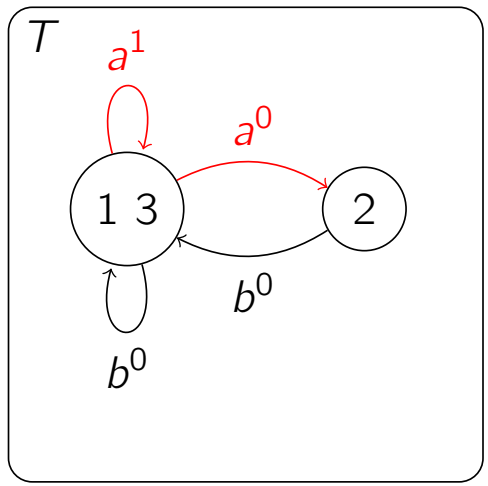
The weight of a graph L is

$$\min_{h:L \to T} \text{weight}_T(h)$$



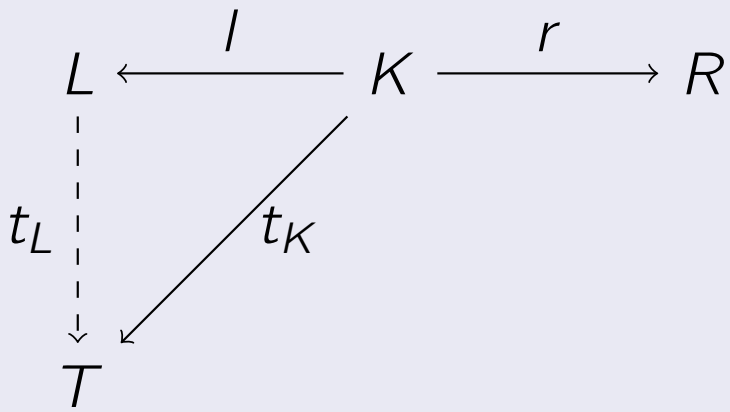
Two morphisms from L to T:





$\text{weight}_T(L) = \min\{1+1, 1+0\} = 1$

Extension of a morphism



A morphism $t_L : L \to T$ extends t_K if $t_K = t_L \circ l$.
For every morphism $t_K : K \to T$, we define

- $S(t_K, L)$: the minimum weight of the morphisms t_L that extend t_K,
- $S(t_K, R)$: similarly.

Termination Condition

Corollary

of Bruggink et al. 2014 [BKZ14] Every rewriting step strictly decreases the weight if

- *for all $t_K : K \to T$, if there is a morphism t_L that extends t_K, then*

$$S(t_K, L) > S(t_K, R)$$

How to find such a suitable weighted type graph ?

Searching for Weighted Type Graphs over Natural Numbers

User-specified parameters:

- k nodes
- maximum edge weight $n \in \mathbb{N}$

Assumption:

- no parallel edges of the same label

The problem amounts to checking the satisfiability of an existential Presburger arithmetic formula with:

- k^2m binary variables where m is the number of labels
- k^2m integer variables

Challenge:

- $2^{k^2m} \cdot n^{k^2m}$ possible assignments of weights
- maximum edge weight hard to determine to guess

Solution: using real numbers instead of natural numbers.

Every rewriting step strictly decreases the weight if

- for all $t_K : K \to T$, if there is a morphism t_L that extends t_K, then
$$S(t_K, L) > S(t_K, R)$$
- there is $\delta > 0$ such that for all $t_K : K \to T$, if there is a morphism t_L that extends t_K, then
$$S(t_K, L) > S(t_K, R) + \delta$$

Searching for Weighted Type Graphs over Real Numbers

User-specified parameters:

- k nodes
- ~~edge weights in $\{0, 1, \ldots, n\}$~~

Assumption:

- no parallel edges of the same label

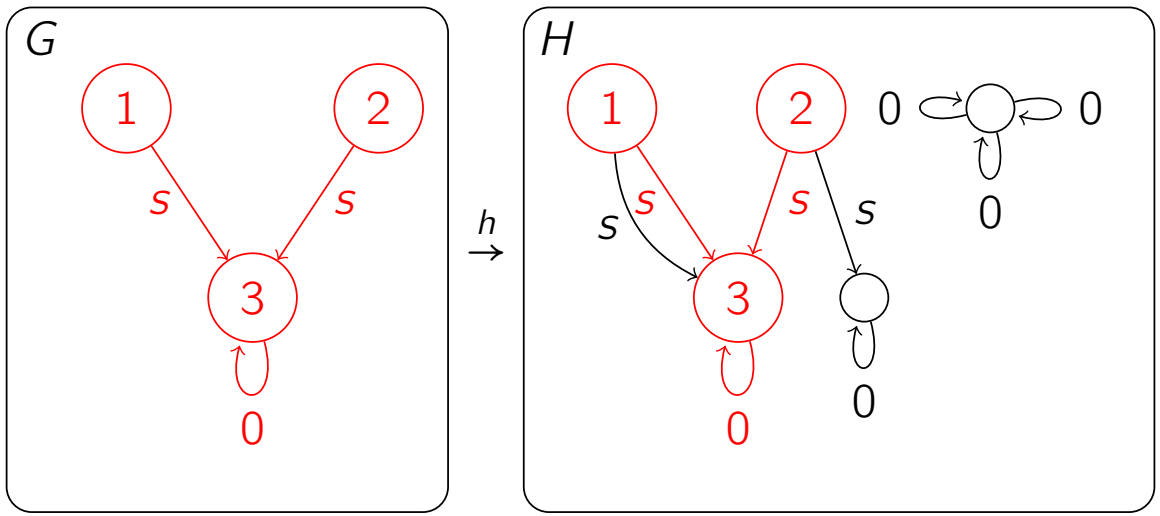
The problem amounts to checking the satisfiability of an existential Presburger arithmetic formula:

- $k^2|\Sigma|$ binary variables
- $k^2|\Sigma|$ ~~integer~~ real variables

Challenge:

- ~~there are $2^{k^2|\Sigma|} \cdot n^{k^2|\Sigma|}$ possible assignments of weights~~
- there are $2^{k^2|\Sigma|}$ linear programs which have polynomial-time average-case complexity

Inclusions : morphisms h with $h(x) = x$ for all x.

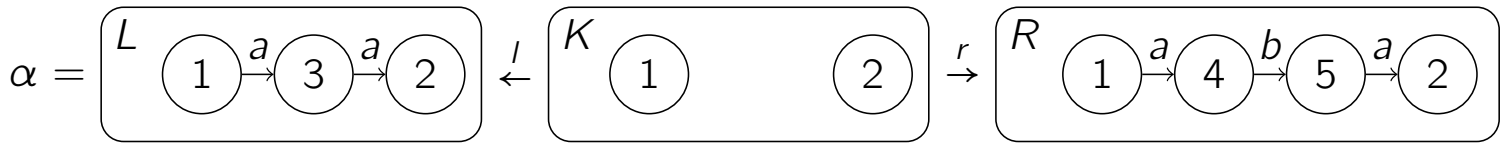


Remarks:

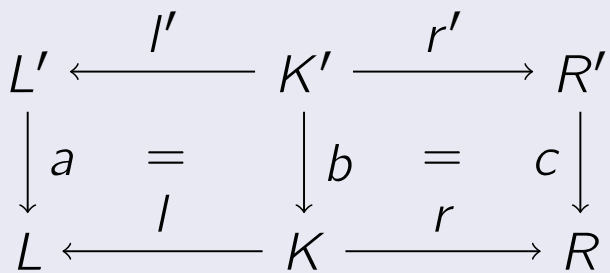
- G is a subgraph of H.

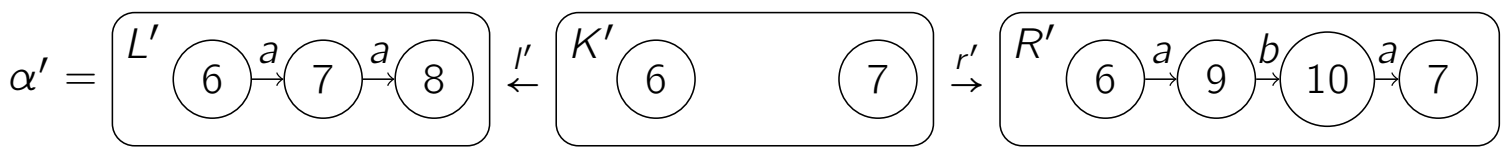
Graph rewriting rule

Rules $\varphi = (L \overset{l}{\leftarrow} K \overset{r}{\rightarrow} R)$ consist of inclusions l and r.



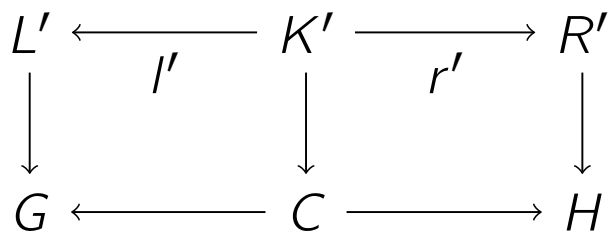
Rule $\varphi' = (L' \overset{l'}{\leftarrow} K' \overset{r'}{\rightarrow} R')$ and φ are equivalent if there are isomorphisms a, b, c such that:





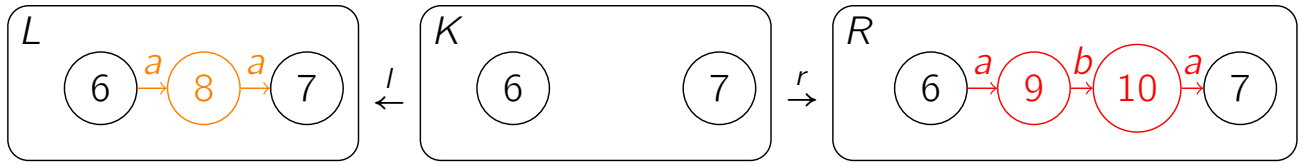
Graph Rewriting

Rewriting steps $G \Rightarrow_\varphi H$ using rule φ are commutative diagrams with an equivalent rule $L' \overset{l'}{\leftarrow} K' \overset{r'}{\rightarrow} R'$ where all morphisms are inclusions:

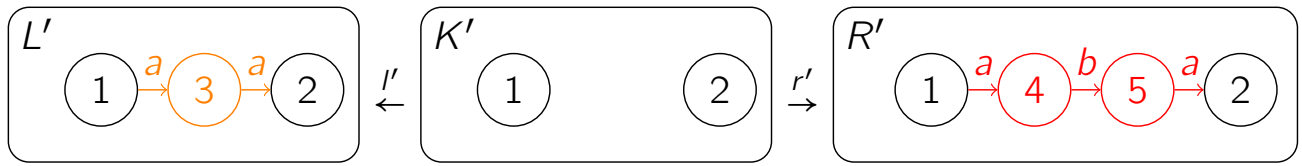


A rewriting step with a running example

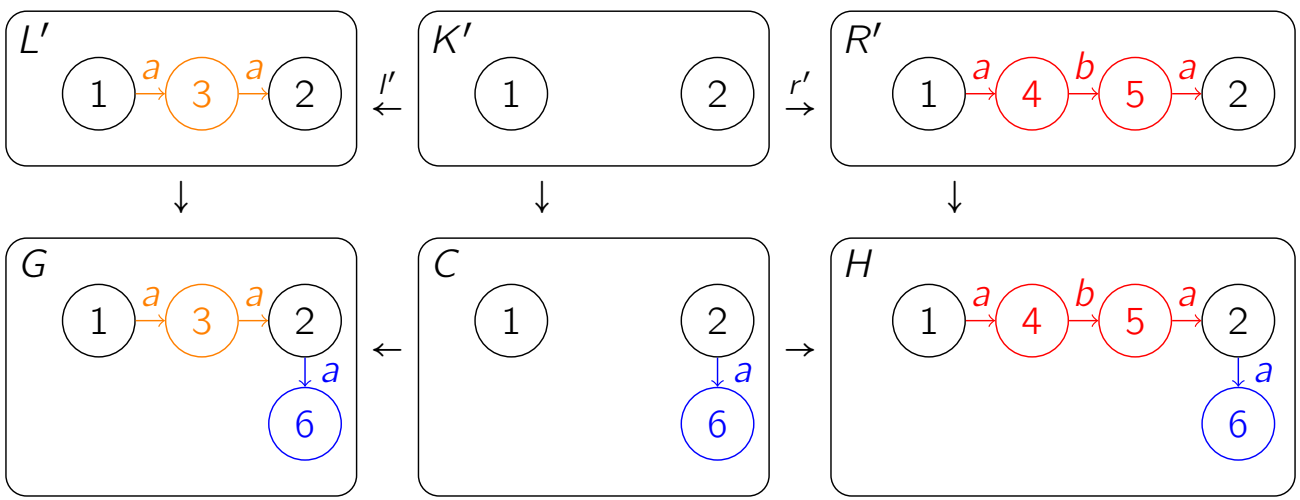
Rewriting rule:

$\varphi =$ 

An equivalent rewriting rule:

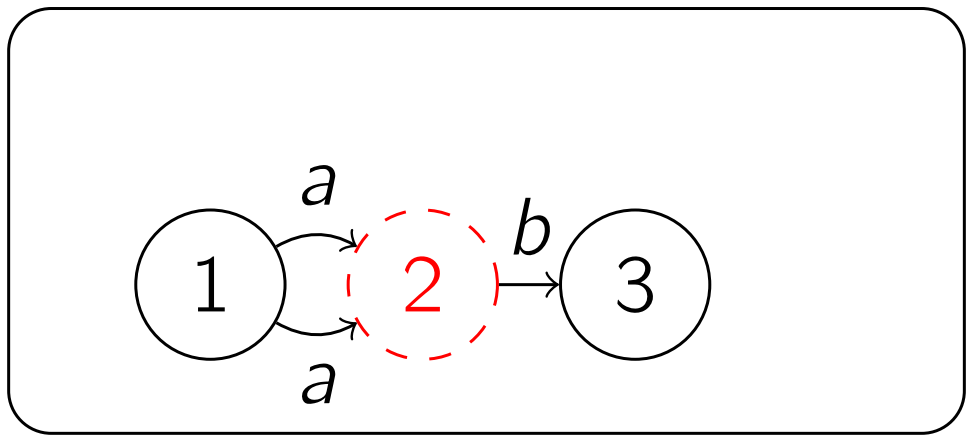
$\varphi' =$ 

A rewriting step $G \Rightarrow_{\varphi} H$:



Pre-graphs

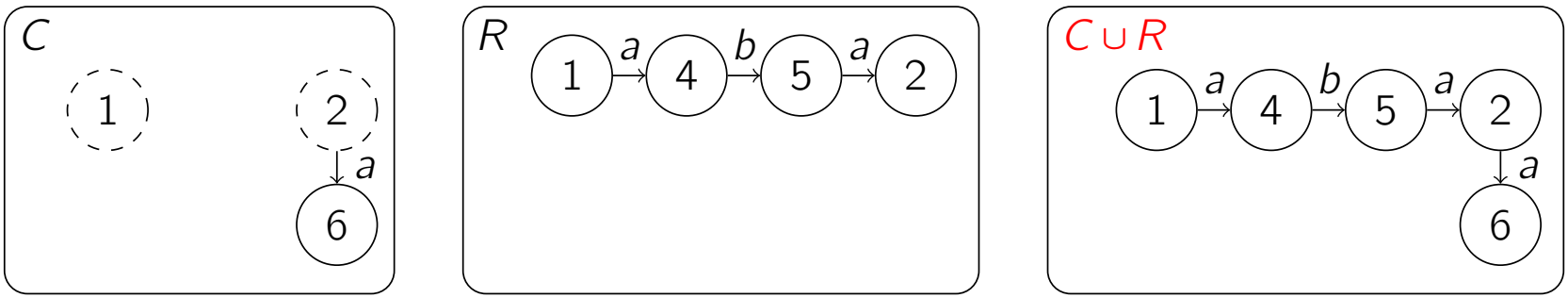
Pre-graphs are graphs with missing nodes and dangling edges.



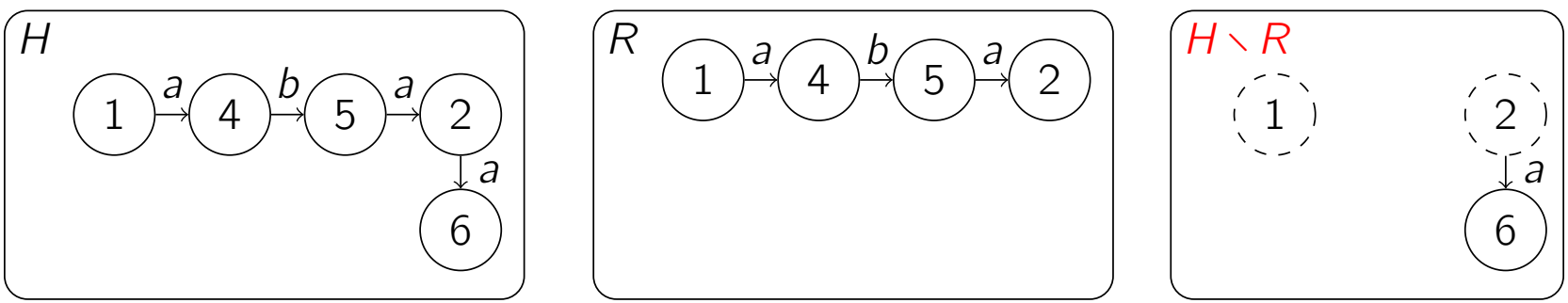
- 1 missing node in red,
- all edges are dangling,
- 2 existing nodes.

Pre-graph operations

Union of two pre-graphs $C \subseteq G$ and $R \subseteq G$, denoted $C \cup R$.

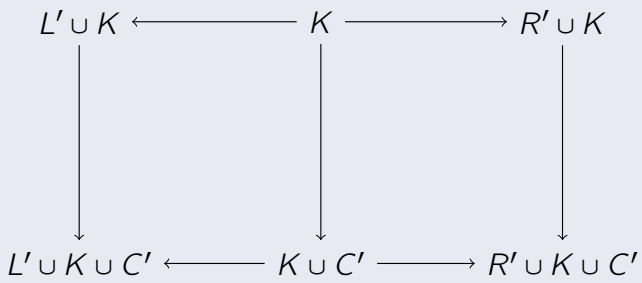


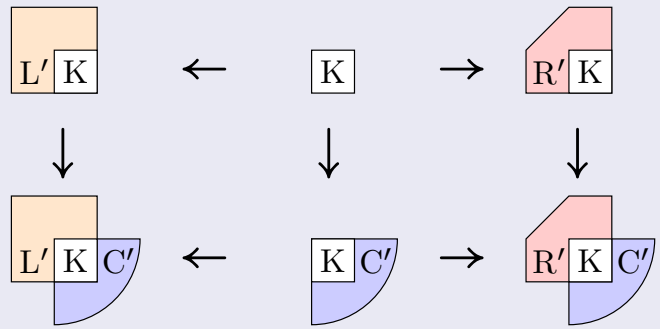
Relative complement of R in H where $R \subseteq H$, denoted $H \setminus R$.



Analysis of rewriting steps

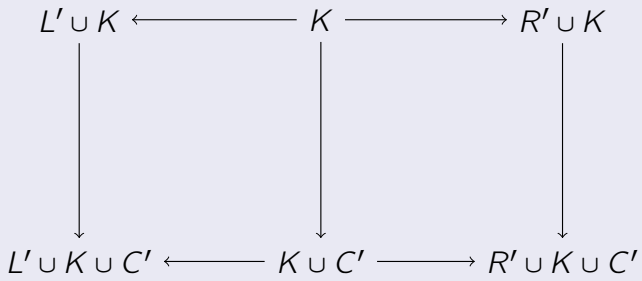
In a rewriting step, graphs can be decomposed as unions of pre-graphs:

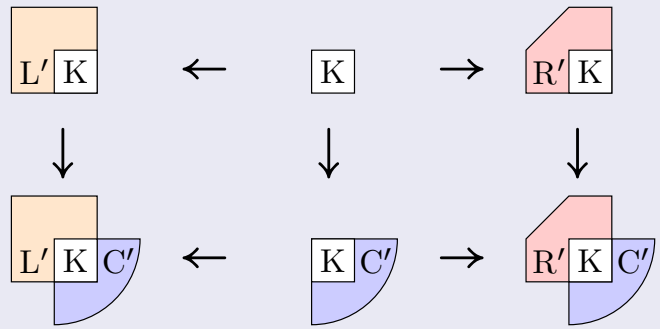


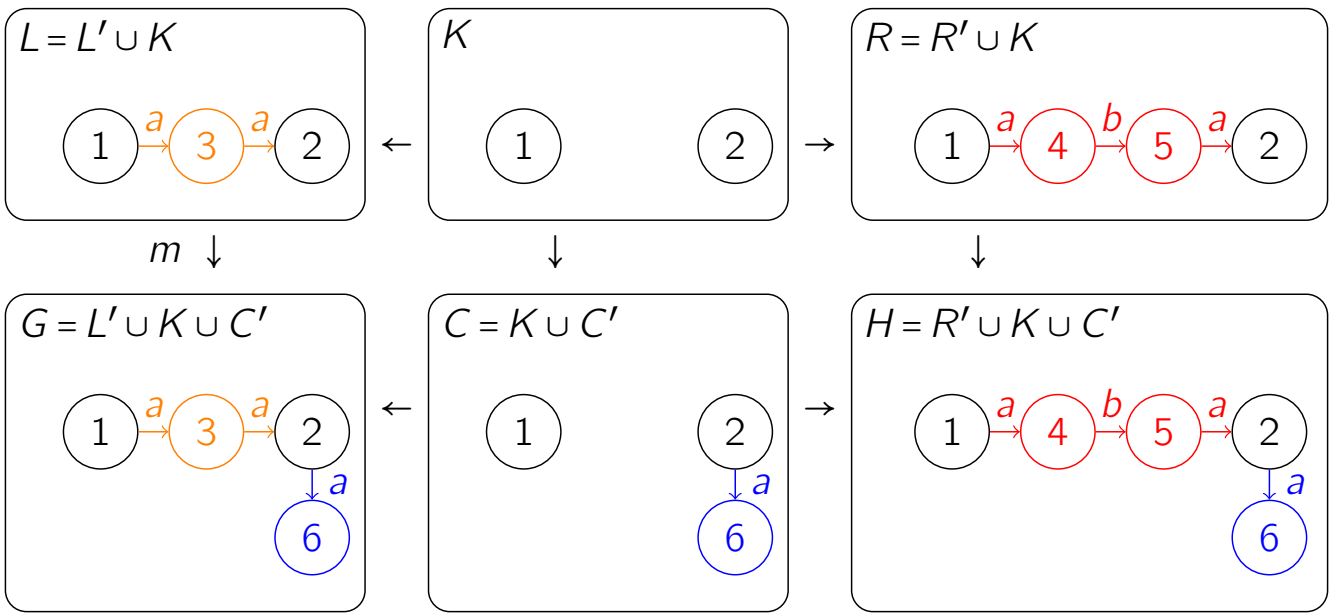


Analysis of rewriting steps

In a rewriting step, graphs can be decomposed as unions of pre-graphs:

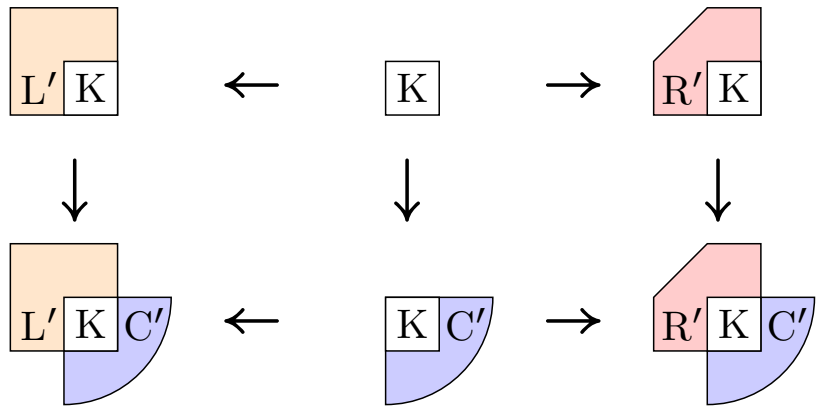




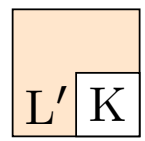
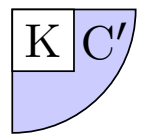
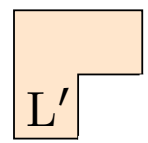
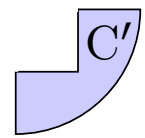


Implicit, Explicit and Shared Occurrences

An X-occurrence in a graph G is an injective morphism $x : X \to G$.



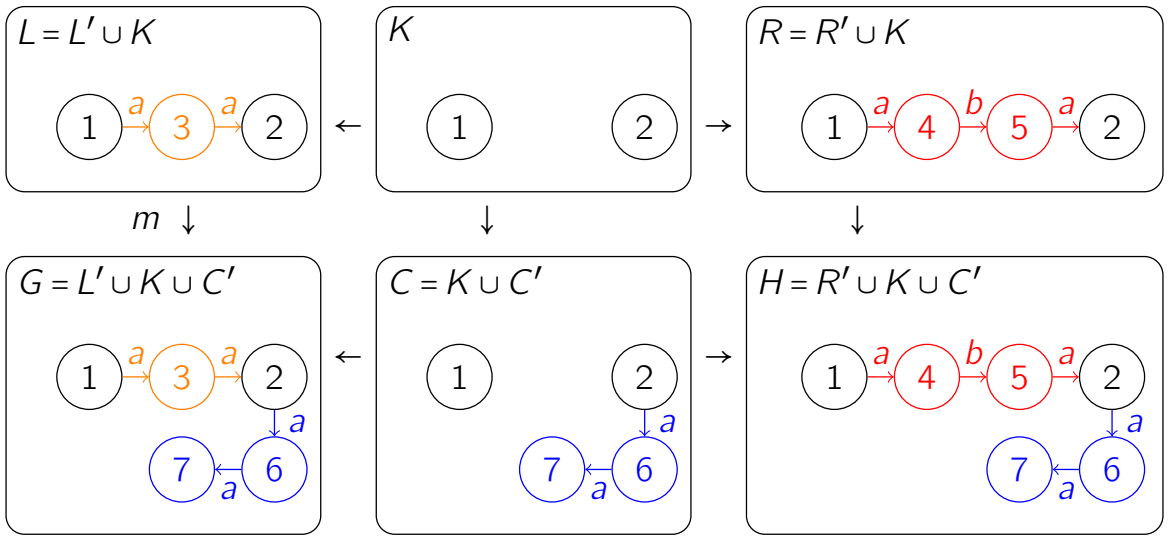
An X-occurrence is

- explicit if $\mathsf{Im}(x)$ is included in 
- shared if $\mathsf{Im}(x)$ is included in 
- implicit if $\mathsf{Im}(x)$ has elements in both  and 

Similarly, in H.

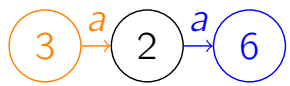
Example

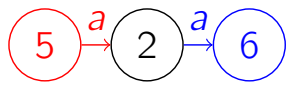
Let X be the graph $\bullet \xrightarrow[a]{} \bullet \xrightarrow[a]{} \bullet$. Consider the rewriting step:

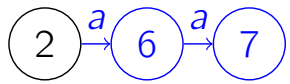


Explicit X-occurrence in G: 

Explicit X-occurrence in H: None.

Implicit X-occurrences in G: 

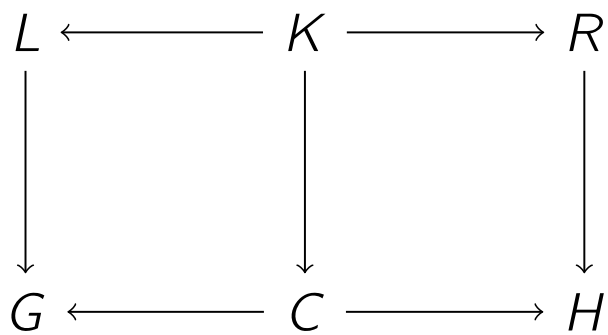
Implicit X-occurrence in H: 

Shared X-occurrence by G and H: 

Obersvation: Shared X-occurrences in G and H are the same.

A sufficient condition for termination

φ terminates if for all rewriting step:



the following holds:

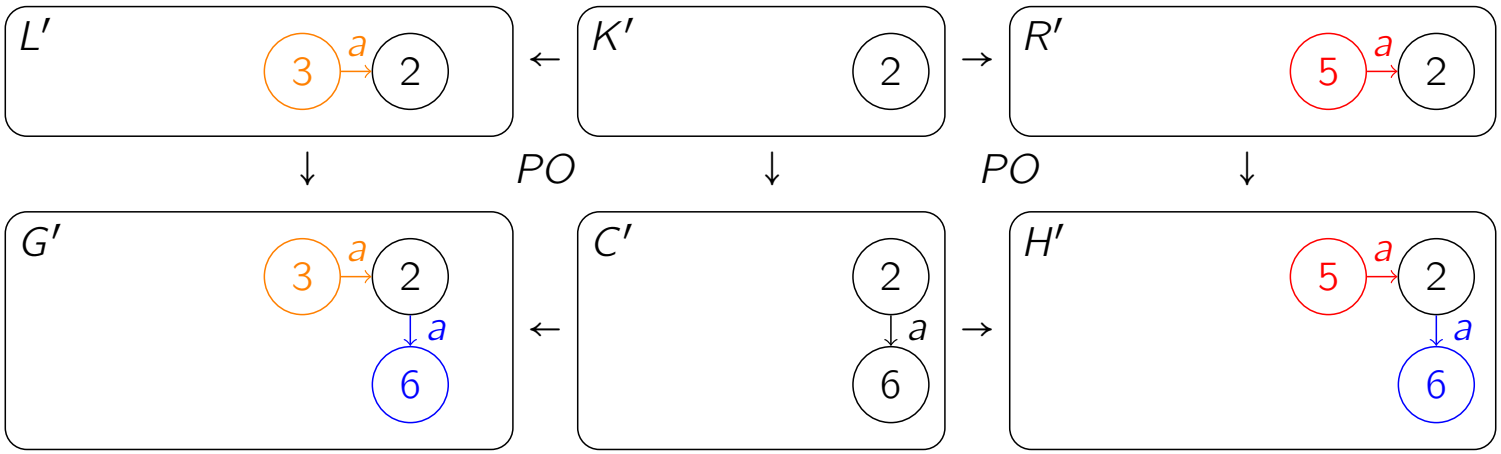
1. $|\text{explicit } X\text{-occurrences in } G| > |\text{explicit } X\text{-occurrences in } H|$;
2. $|\text{implicit } X\text{-occurrences in } G| \geq |\text{implicit } X\text{-occurrences in } H|$.

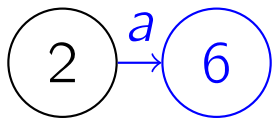
The first condition is straightforward because

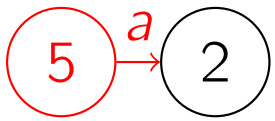
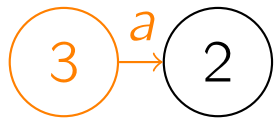
- explicit X-occurrences in G = X-occurrences in L,
- explicit X-occurrences in H = X-occurrences in R,
- X-occurrences in L and R are computable.

Challenge: Establishing the second condition.

Analysis of Implicit Occurrences in G and H



The occurrence is $C' \cup R'$.  is shared by G and H.

 is not in G but there is  in G and $C' \cup L'$ is an implicit occurrence in G.

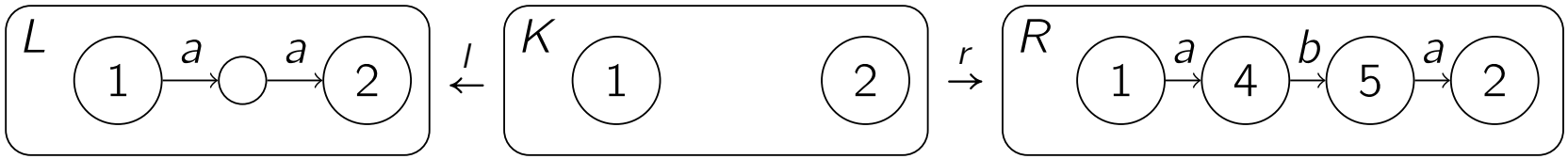
X-non-increasing rule

Let $\varphi : L \leftarrow K \rightarrow R$ be a rule.

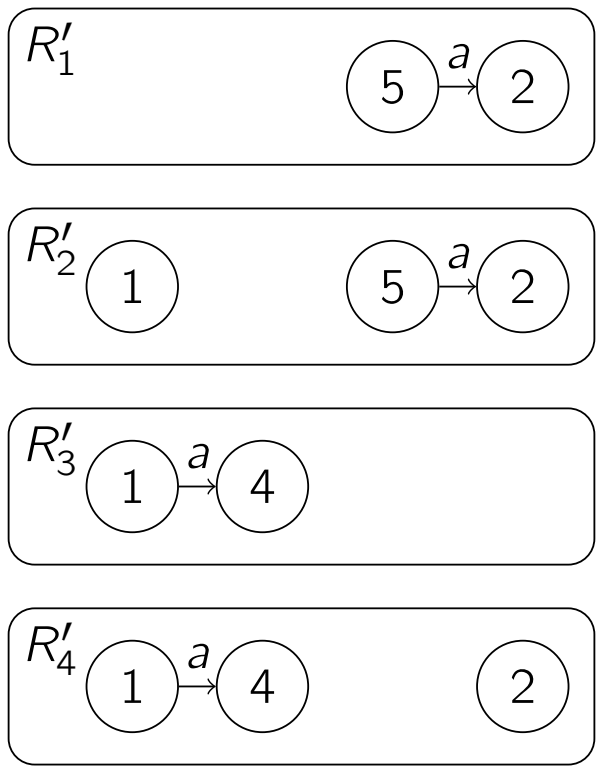
Lemma (More X-occurrences before rewriting)

For all $G \Rightarrow_\varphi H$, there are more implicit X-occurrences in G than in H, if
"subgraphs of R that can form an implicit X-occurrence in some rewriting step can be mapped to subgraphs in L while preserving the interface elements".

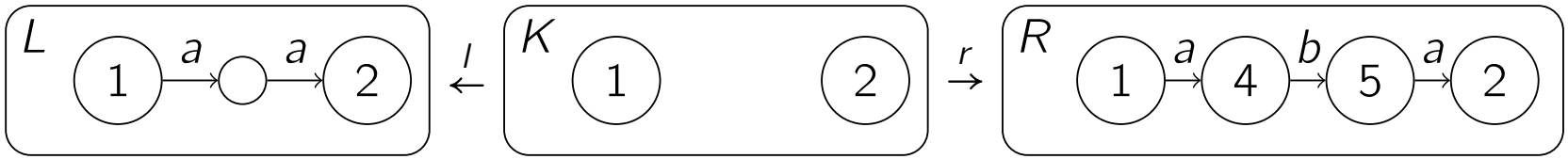
Example



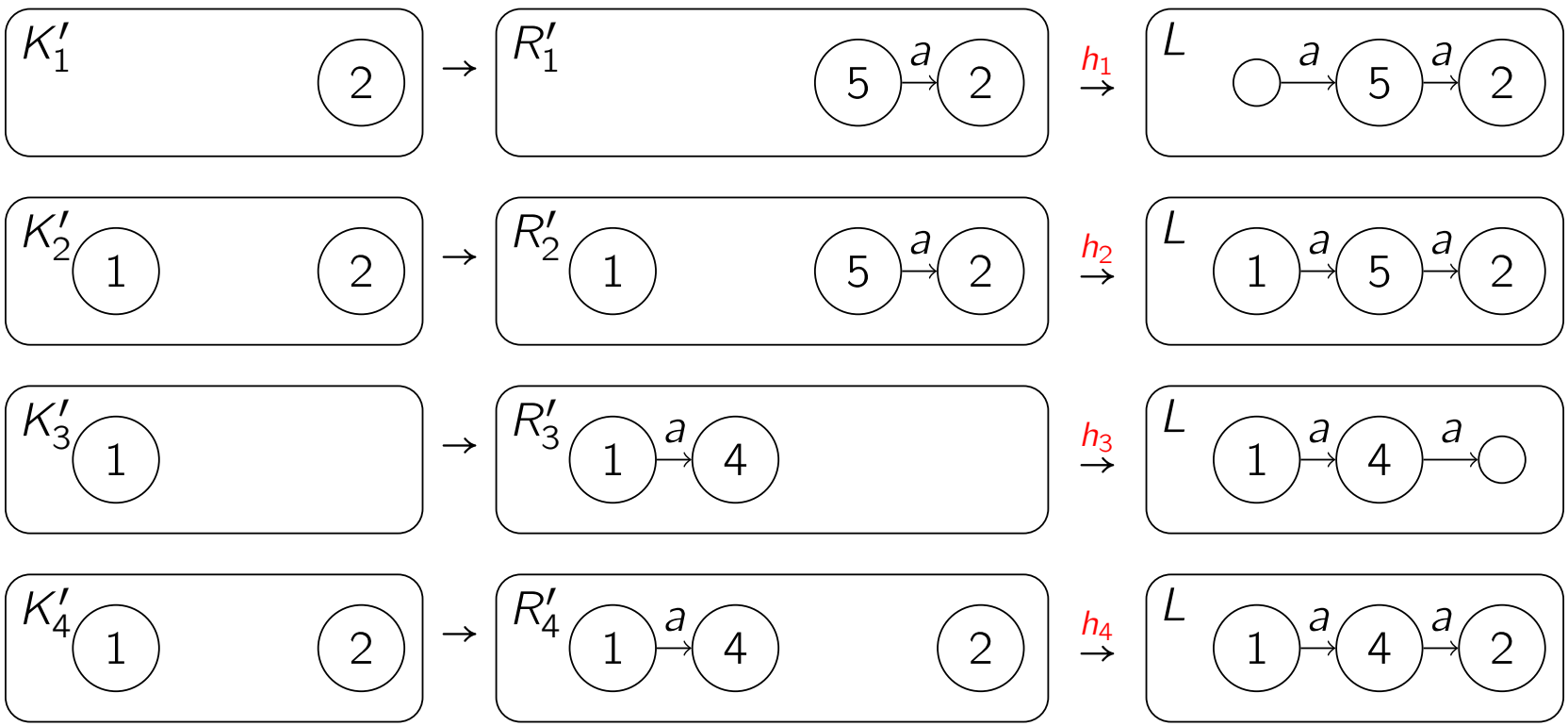
The set $D(R, X)$ of subgraphs of R which can form an implicit X-occurrence in some rewriting step:



Example



Distinct graphs in $D(R, X)$ can be mapped to subgraphs in L while preserving the interface elements.

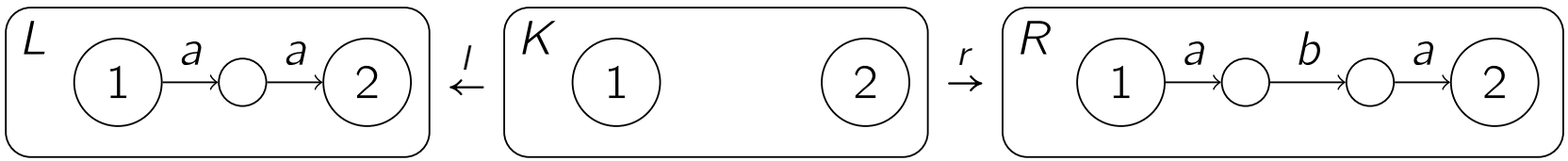


Main Results

Theorem (Sufficient Termination Condition)

Let φ be a X-non-increasing rule. φ is terminating if there are strictly more explicit X-occurrences in L than in R.

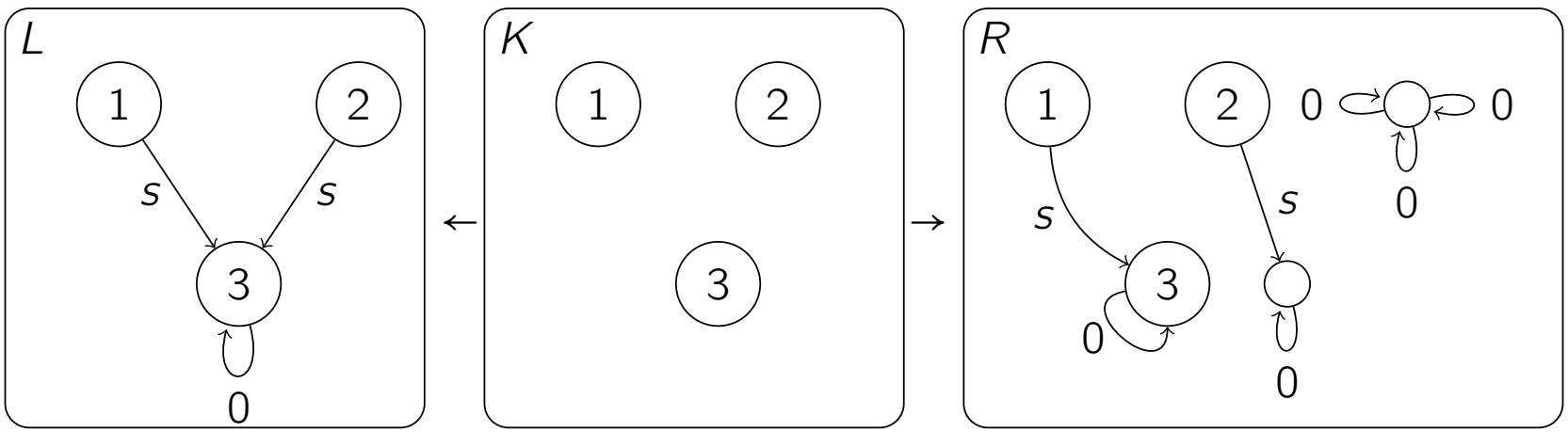
Terminating of Running Example



- X : $\bullet \xrightarrow[a]{} \bullet \xrightarrow[a]{} \bullet$
- X-non-increasing rule
- Strictly more explicit X-occurrences in L than in R:

$$1 > 0$$

Termination of Motivating Rule



Terminating by counting morphisms from $\bullet \xrightarrow{s} \bullet \xleftarrow{s} \bullet$

LyonParallel

- Automated termination tool in Ocaml
- 4 methods implemented:
  - type graph method with weight from non- and well-founded semirings
  - morphism counting method
- Availble : `https://github.com/Qi-tchi/LyonParallel`

Conclusion and Future Work

We have presented

- an extension of an existing method for more efficient and ergonomic implementation,
- a sufficient termination condition based on morphism counting,
- a unified tool in Ocaml for automated iterative termination analysis of graph rewriting systems.

Future work:

- Morphism counting with forbidden contexts,
- Formal proof of correctness using a proof assistant,
- Certificate-generation mechanism,
- Extension to other rewriting approaches.

[BKZ14] H. J. Sander Bruggink, Barbara König, and Hans Zantema. "Termination Analysis for Graph Transformation Systems". In: *Theoretical Computer Science - 8th IFIP TC 1/WG 2.2 International Conference, TCS 2014, Rome, Italy, September 1-3, 2014. Proceedings*. Ed. by Josep Diaz, Ivan Lanese, and Davide Sangiorgi. Vol. 8705. Lecture Notes in Computer Science. Springer, 2014, pp. 179–194. DOI: 10.1007/978-3-662-44602-7_15.