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

Motivation & Goal

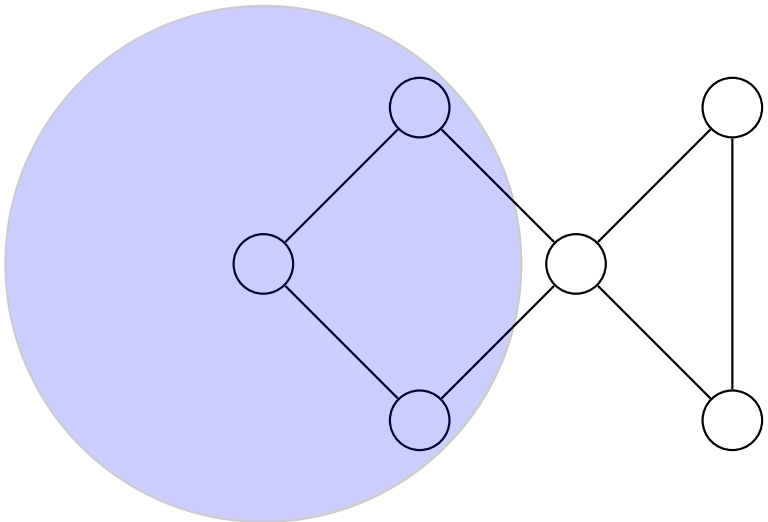
Distributed/concurrent systems are everywhere.

Failures can be catastrophic.

Ensuring correctness is hard.

This thesis: automated verification.

Graph Transformation

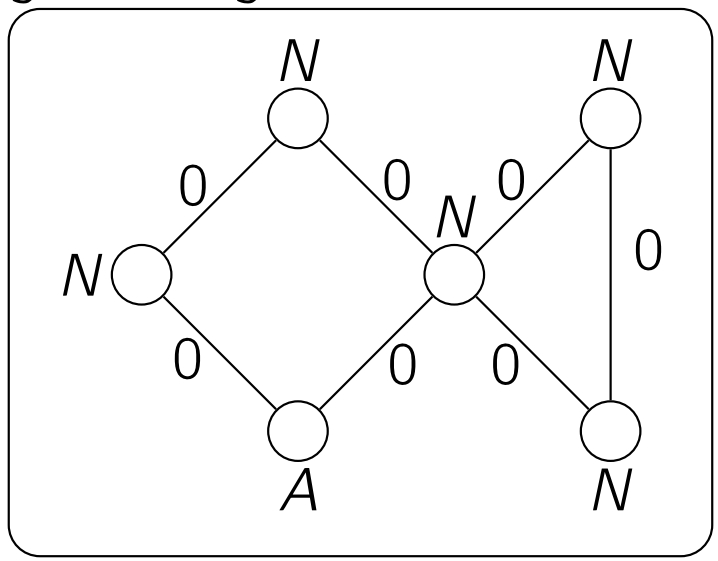


Graph rewriting:

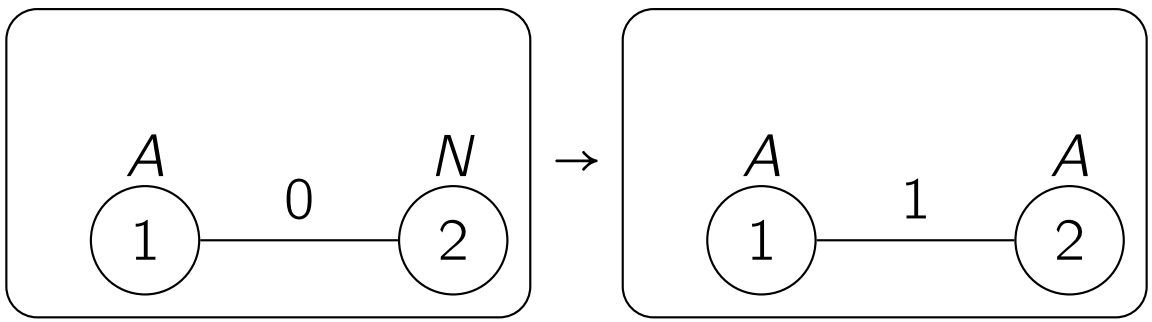
- computational units → nodes
- communication channels → edges
- system states → graphs
- algorithm behaviors → graph transformation rules

Graph Transformation

Graph representing a configuration of a distributed system:

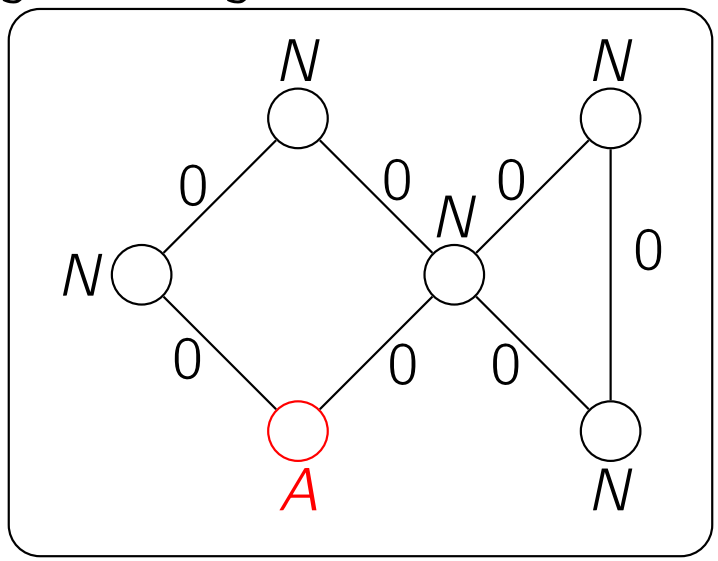


Graph transformation rule:

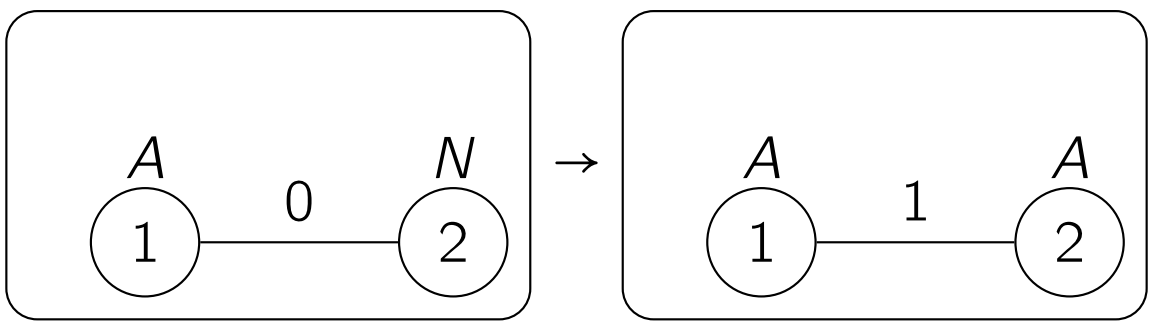


Graph Transformation

Graph representing a configuration of a distributed system:

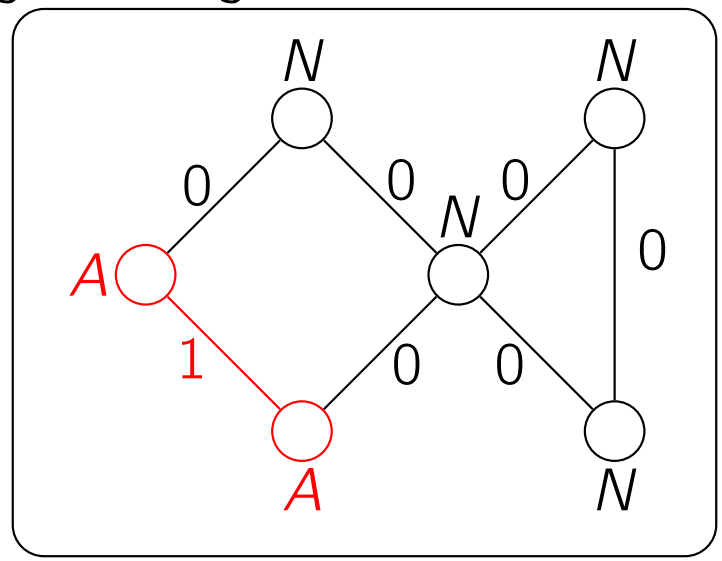


Graph transformation rule:

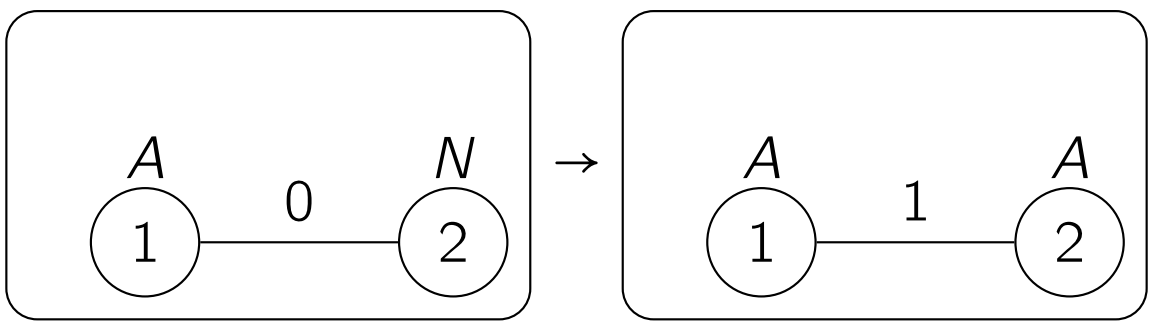


Graph Transformation

Graph representing a configuration of a distributed system:

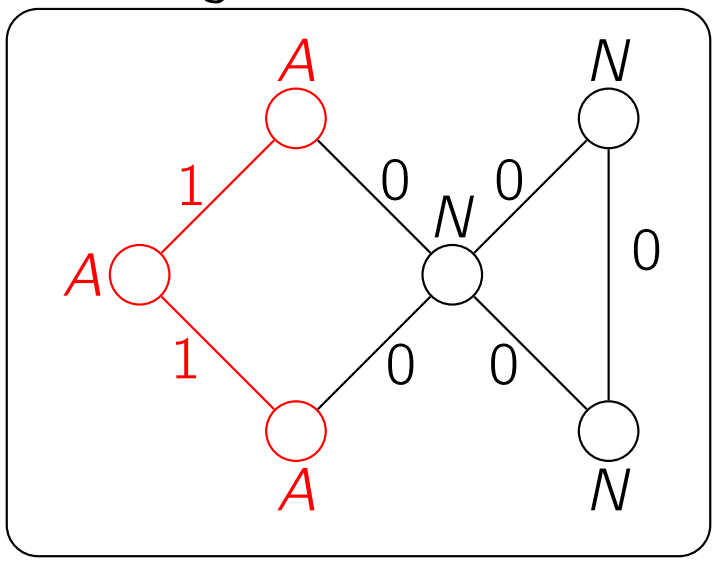


Graph transformation rule:

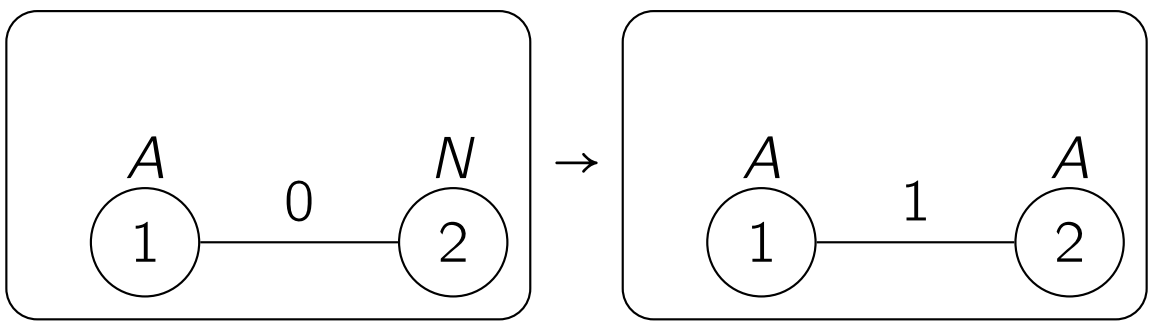


Graph Transformation

Graph representing a configuration of a distributed system:

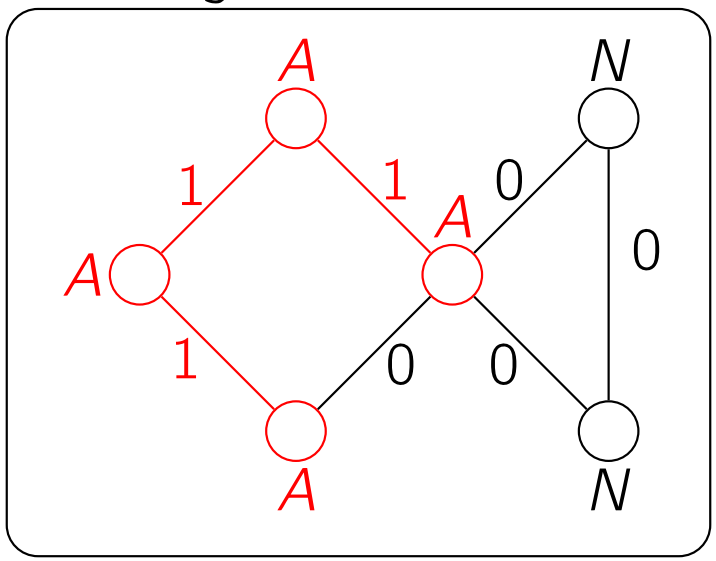


Graph transformation rule:

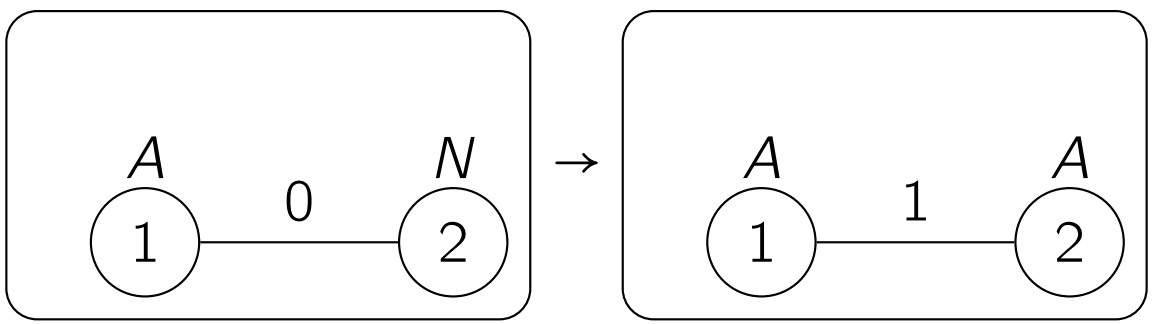


Graph Transformation

Graph representing a configuration of a distributed system:

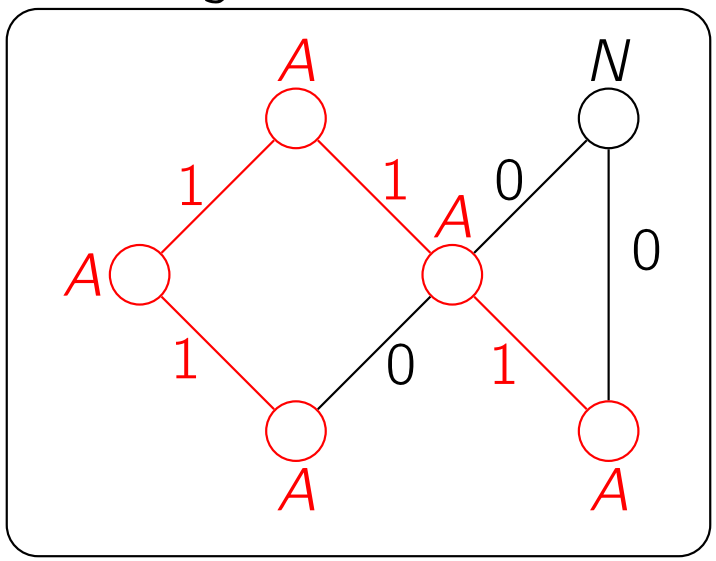


Graph transformation rule:

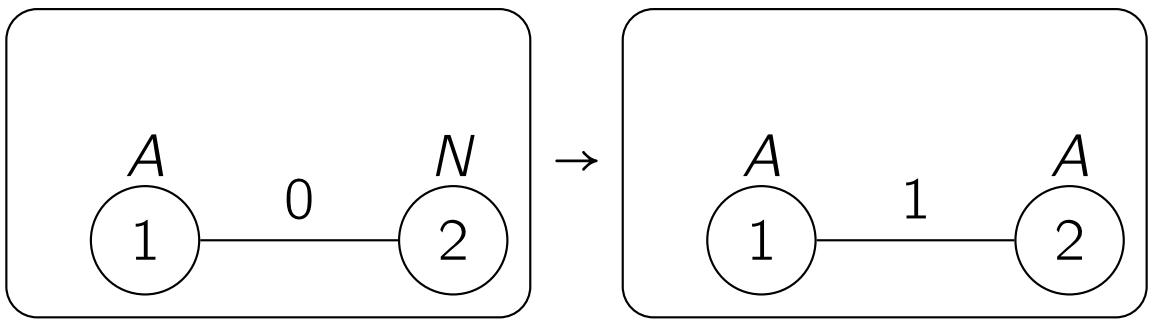


Graph Transformation

Graph representing a configuration of a distributed system:

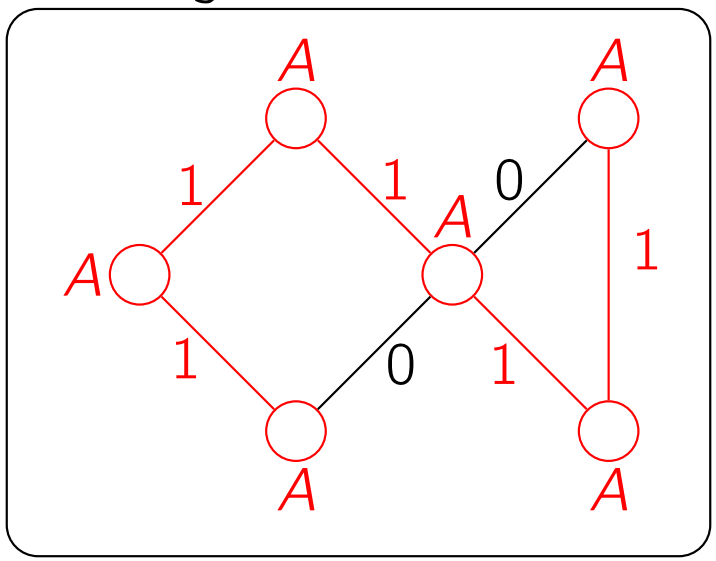


Graph transformation rule:

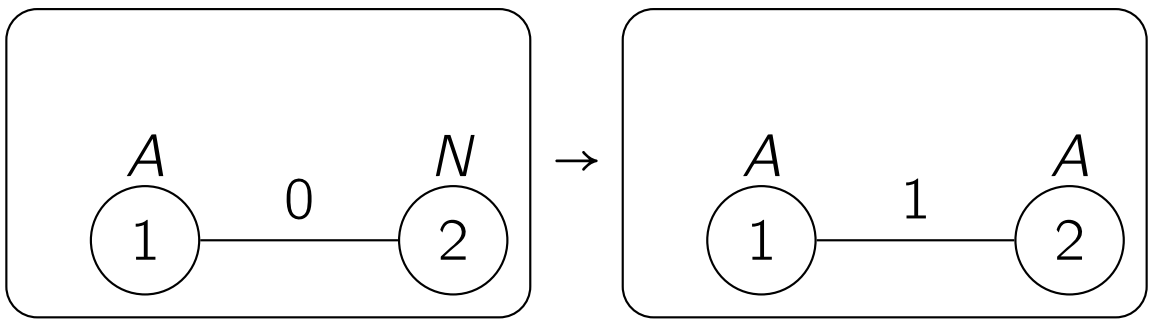


Graph Transformation

Graph representing a configuration of a distributed system:

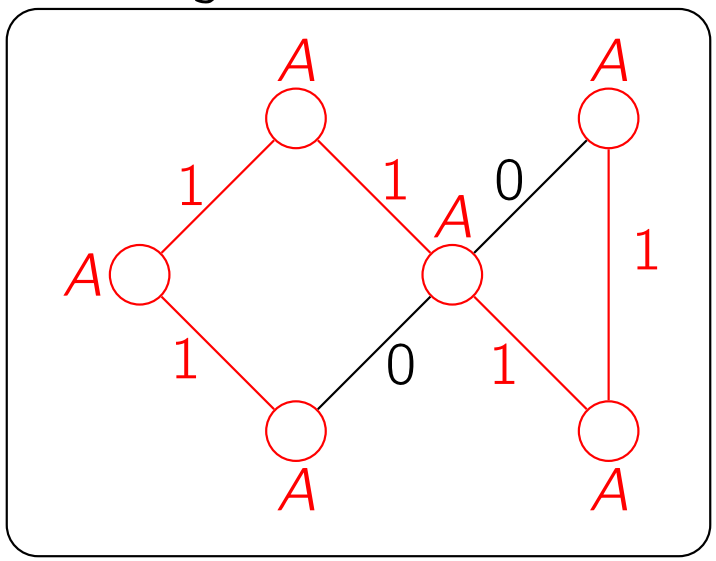


Graph transformation rule:

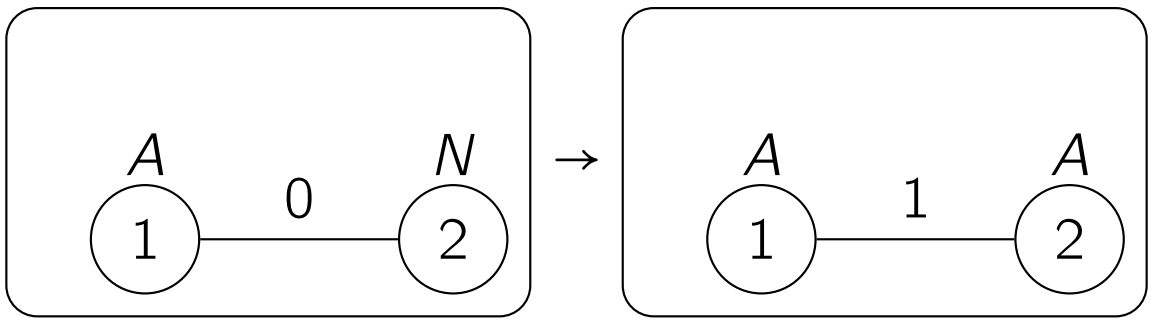


Graph Transformation

Graph representing a configuration of a distributed system:

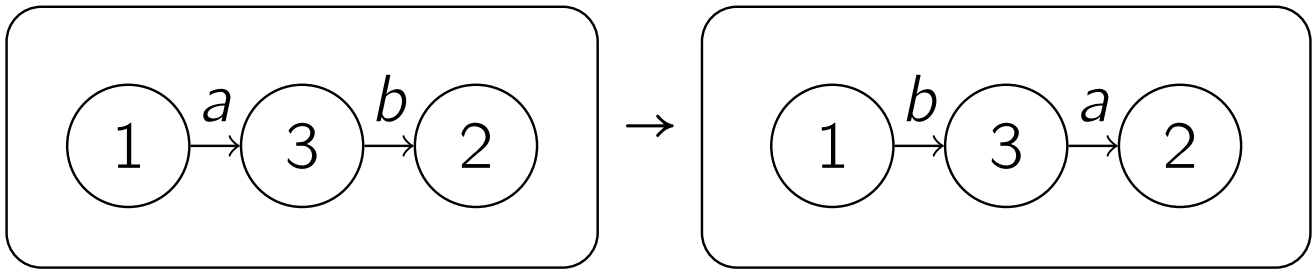


Graph transformation rule:

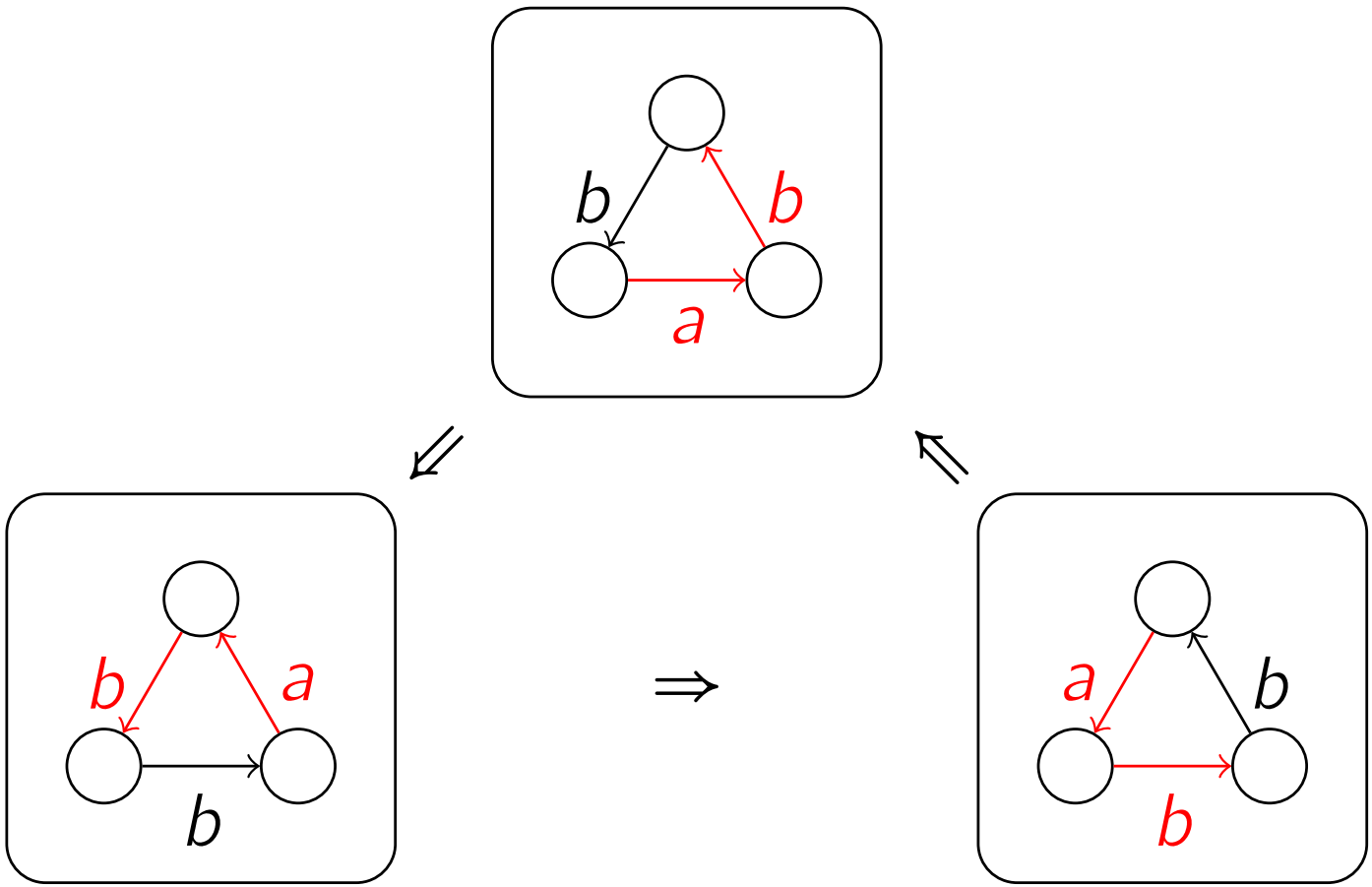


Question: does the transformation process terminate for any initial graph?

A non-termination case



Loop:



Termination

- No graph G_0 can be transformed forever

$$G_0 \Rightarrow G_1 \Rightarrow \cdots$$

  under the strategy

  "apply rules as long as possible"

- Aligns with the notion of program termination:

  "every execution (on any input) halts."

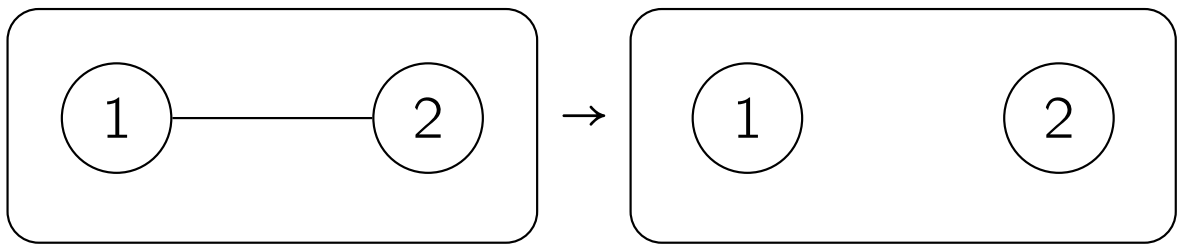
- Undecidable in general
- How to prove termination automatically?

Automated Termination Proofs

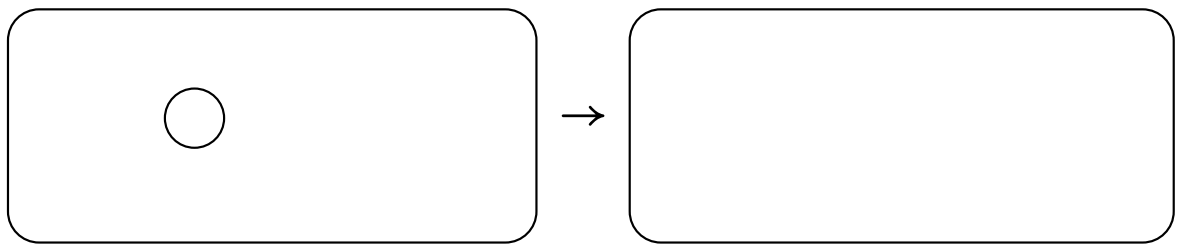
Termination by interpretations:

- interpret graphs as natural numbers;
- show that each transformation step decreases the value.

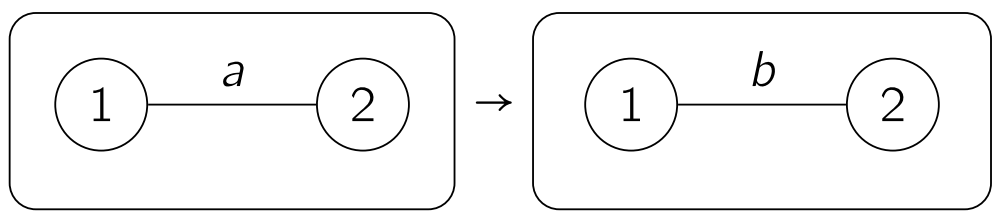
Number of edges:



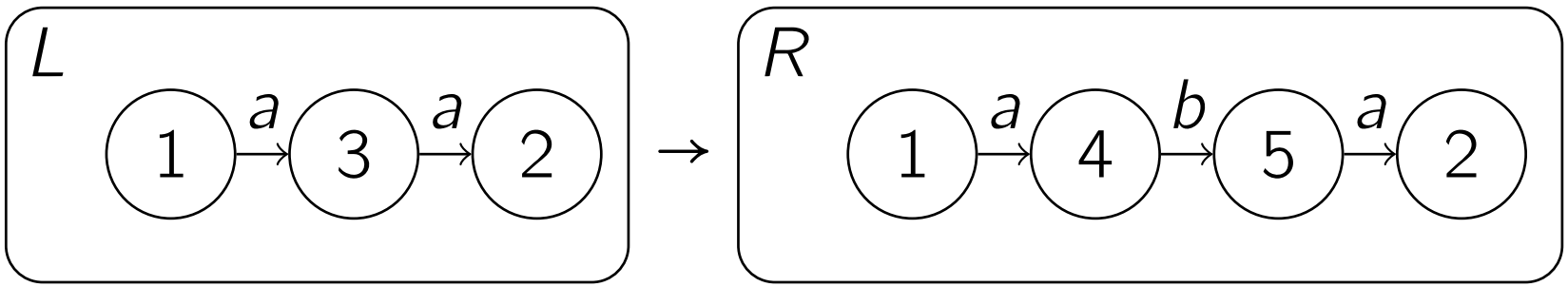
Number of nodes:



Number of edges labeled by a:



A non-trivial example



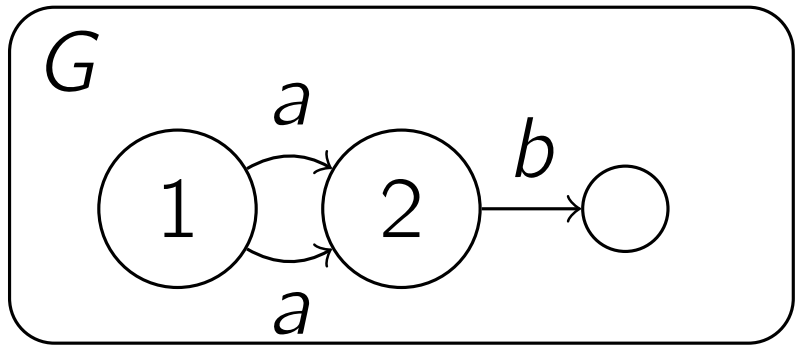
Limitation of simple interpretations

Need a formal definition of graph transformations

- edges incident to deleted nodes

Structure of the presentation

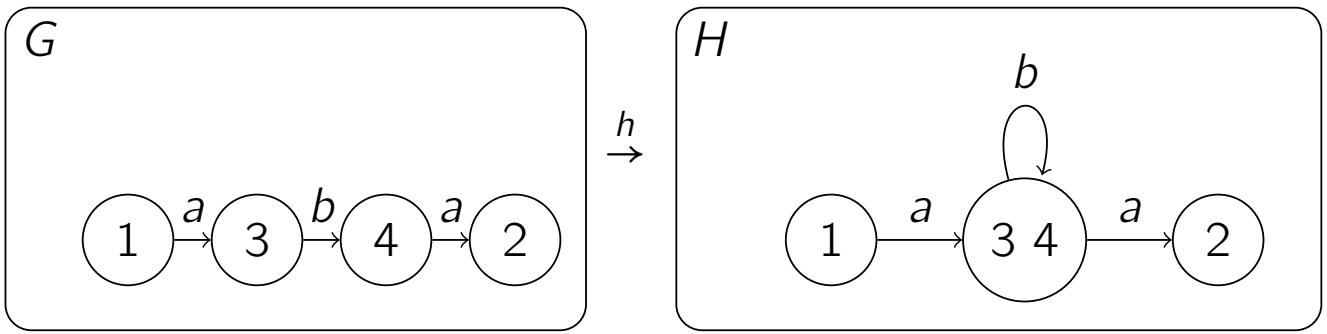
Graphs : finite, directed, edge-labeled multigraphs



Remarks:

- Edges with the same source, target and label are permitted.
- G : graph name
- Numbers inside nodes are identifiers (omitted when not relevant).

Graph morphisms: structure-preserving functions



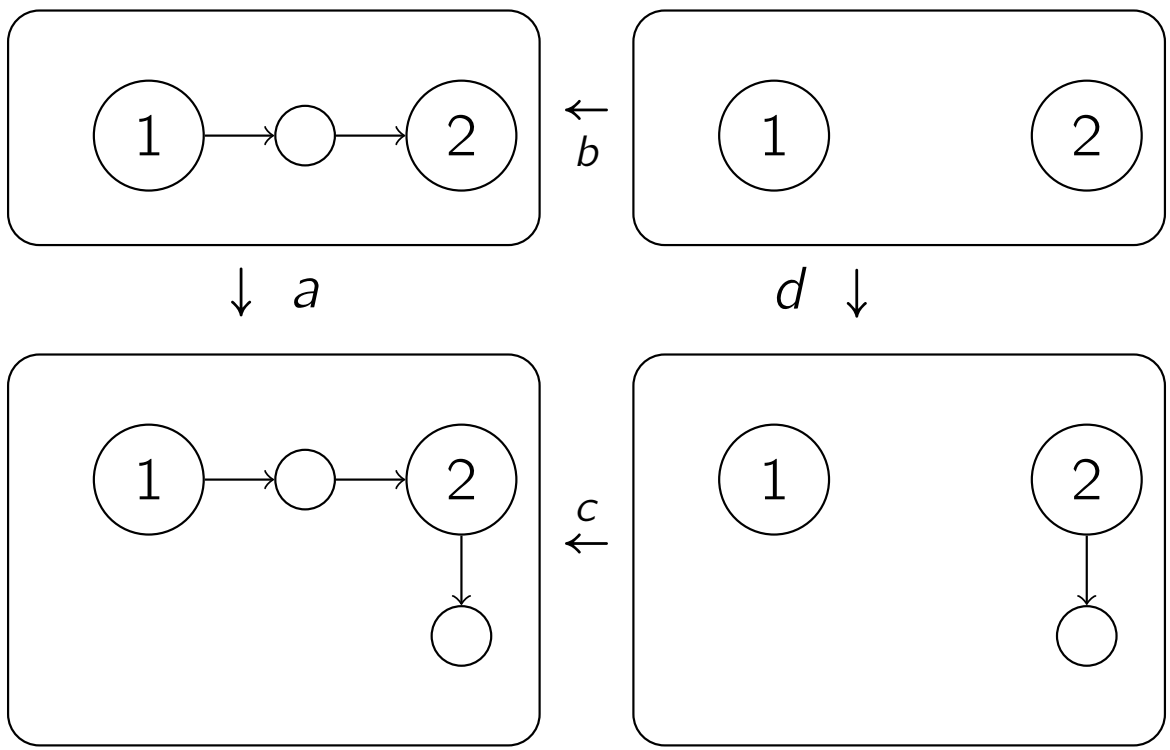
Remarks:

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

Commutative Diagram

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

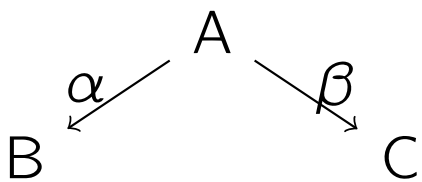
Commutative if $a \circ b = c \circ d$.



Pushouts: gluing graphs along a common part

Definition

The **pushout** of (α, β) is

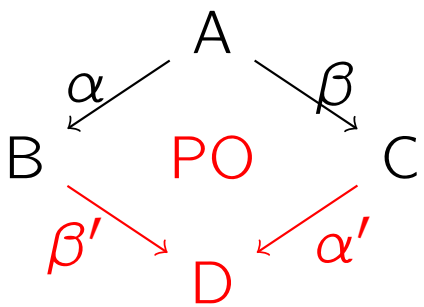


Pushouts: gluing graphs along a common part

Definition

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

- $\square$ABDC is commutative,

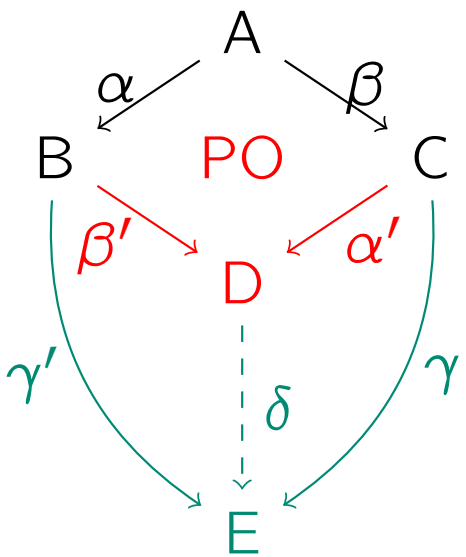


Pushouts: gluing graphs along a common part

Definition

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

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

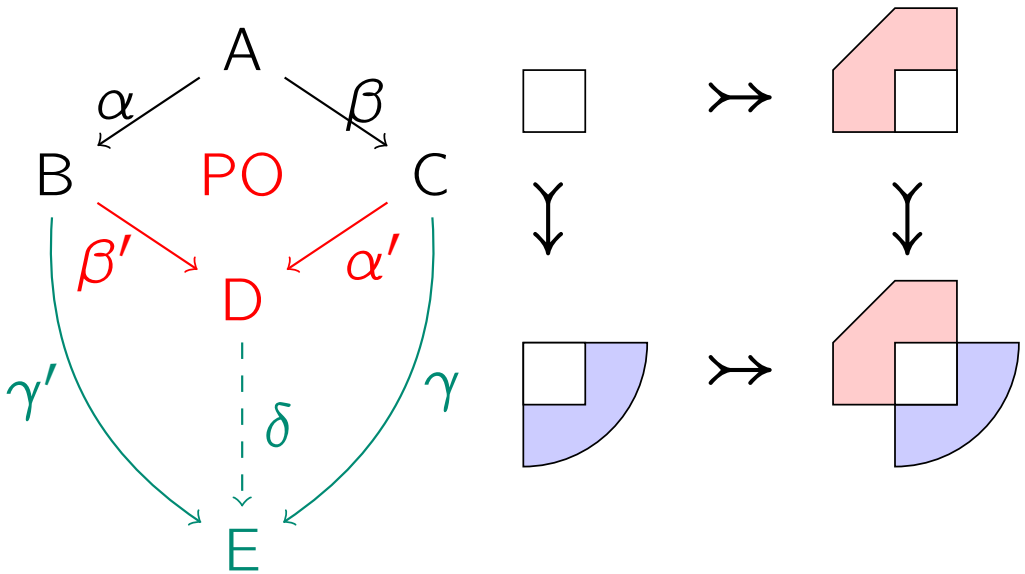


Pushouts: gluing graphs along a common part

Definition

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

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

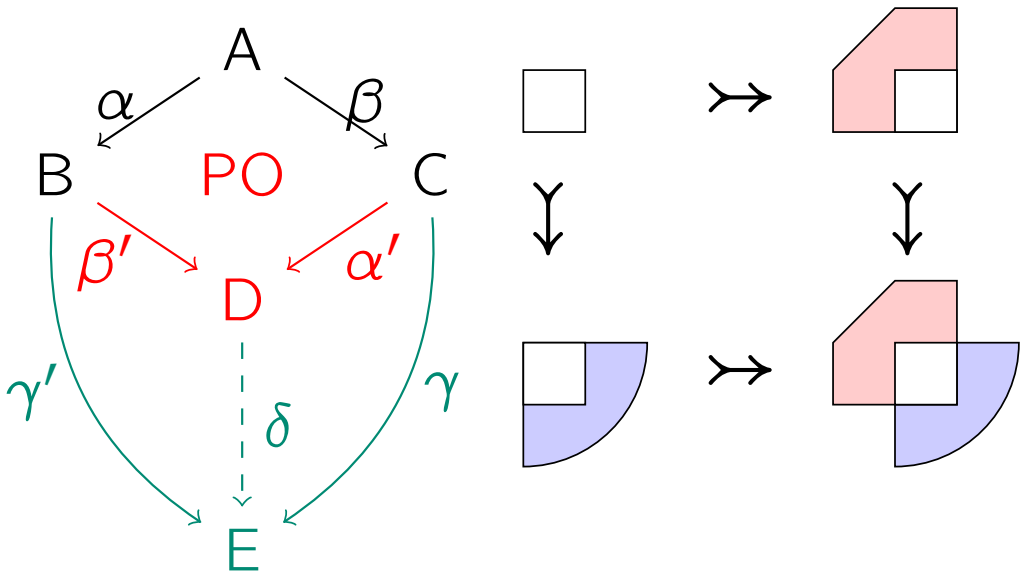


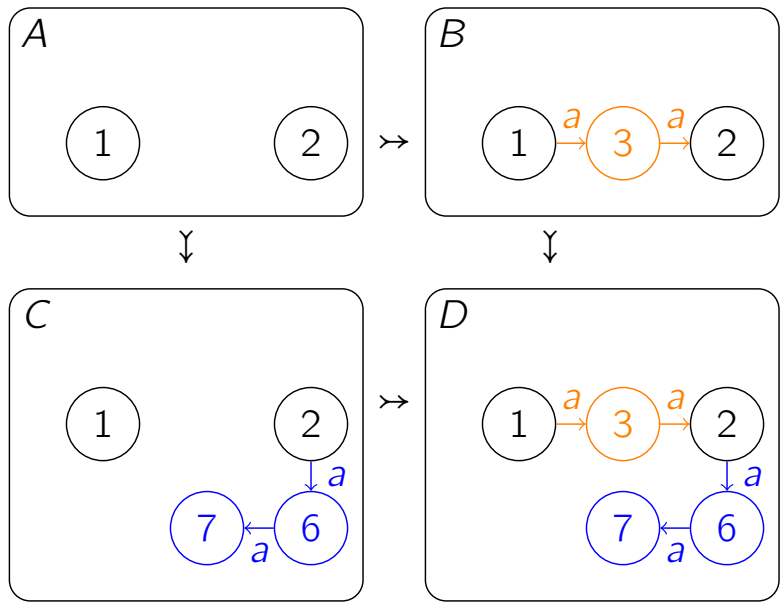
Pushouts: gluing graphs along a common part

Definition

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

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

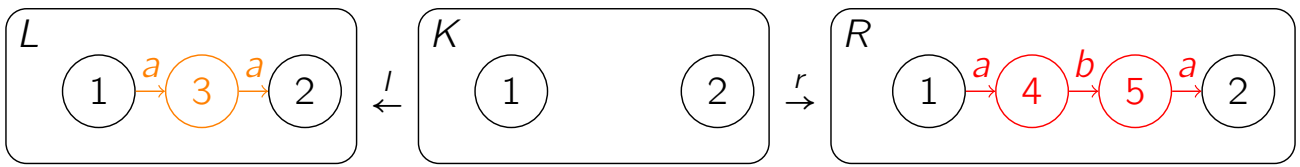




Graph rewriting with double-pushout approach (DPO)

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

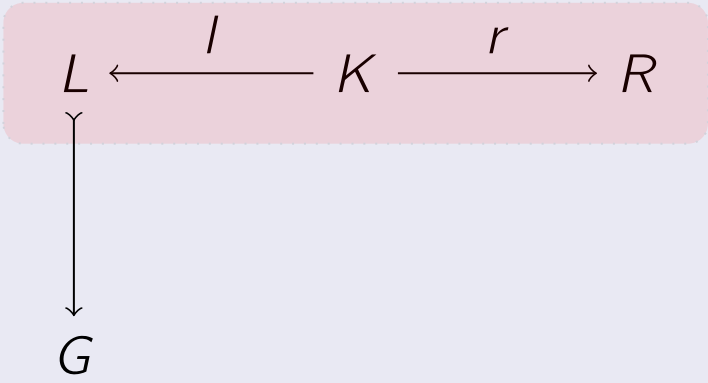
Rewriting rule with interface K



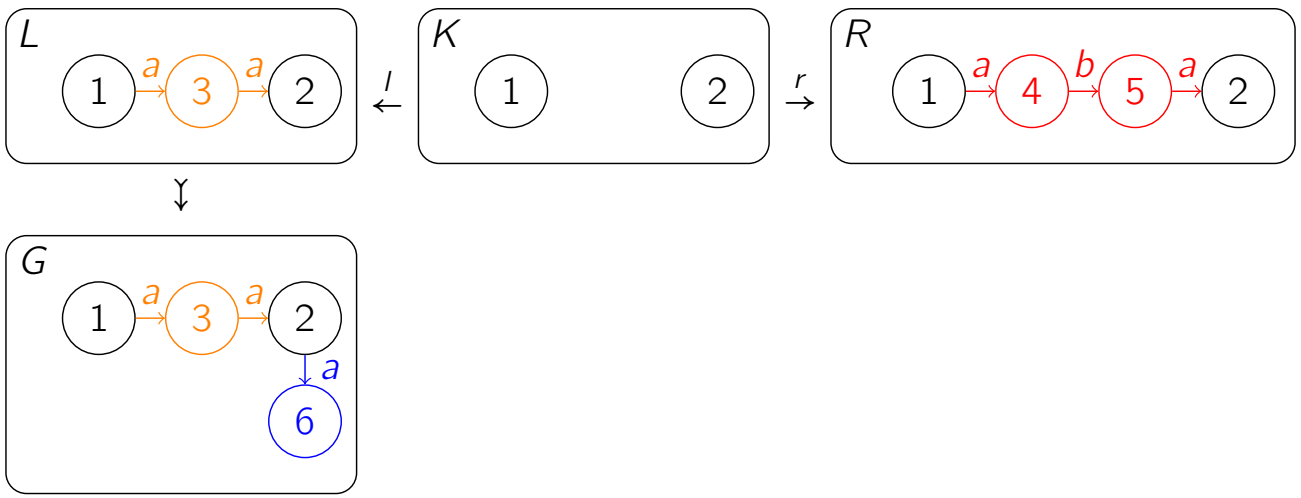
Remark:

- No implicit edge deletion

Graph rewriting with double-pushout approach (DPO)



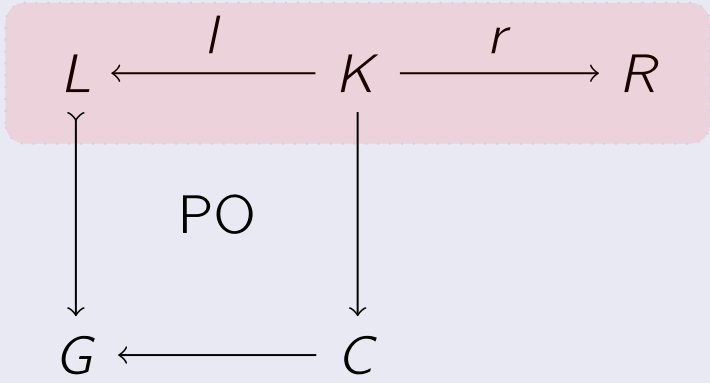
Rewriting rule with interface K



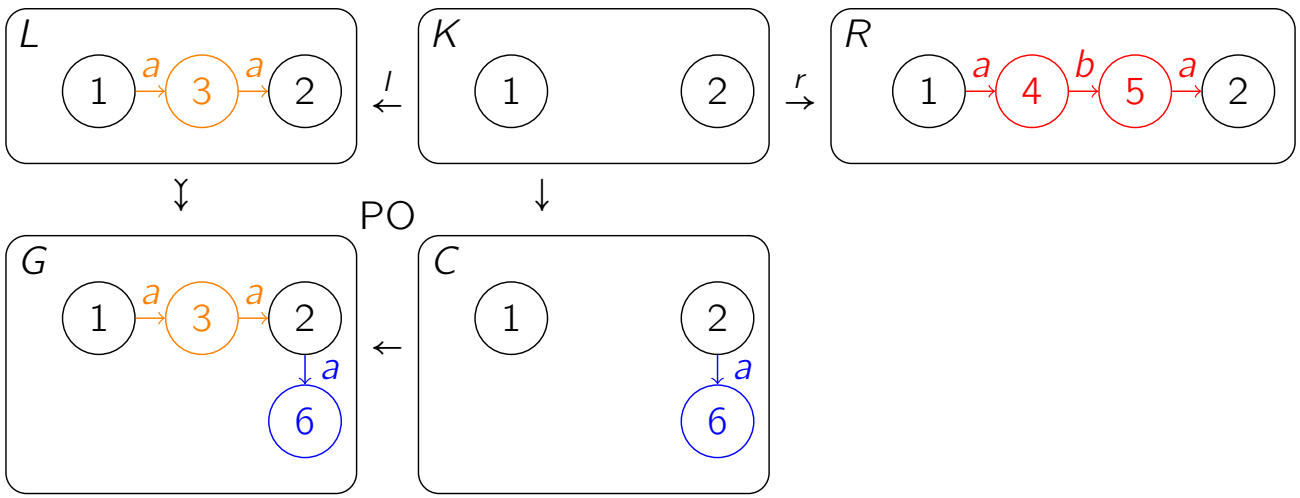
Remark:

- No implicit edge deletion

Graph rewriting with double-pushout approach (DPO)



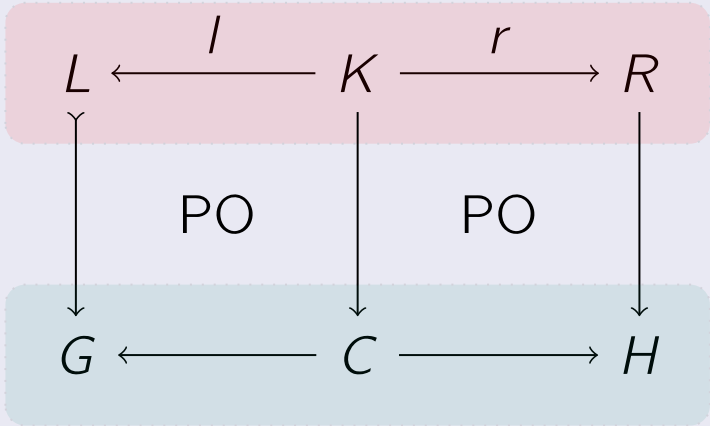
Rewriting rule with interface K



Remark:

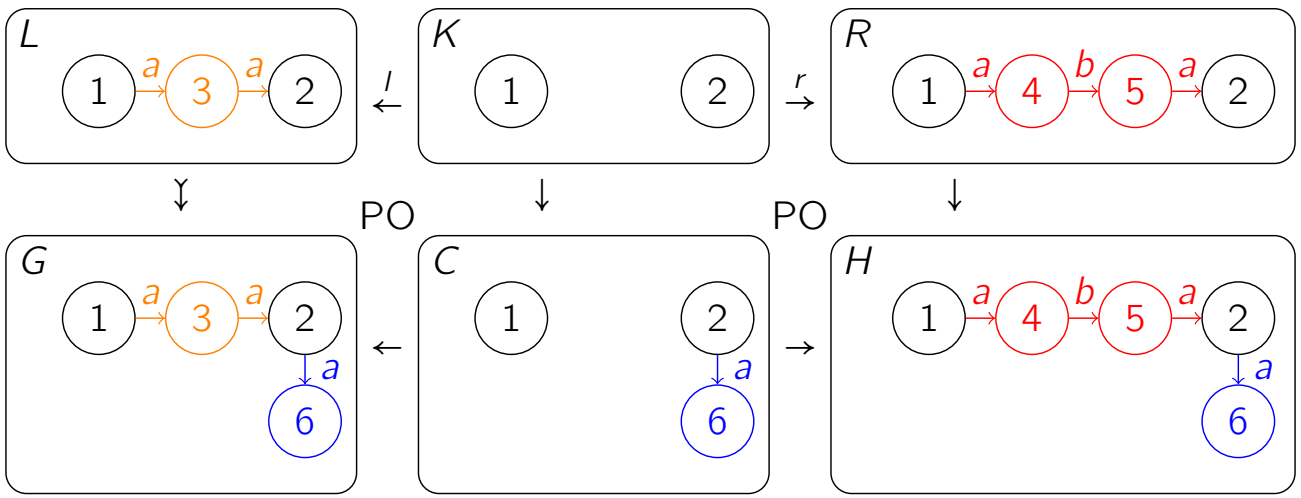
- No implicit edge deletion

Graph rewriting with double-pushout approach (DPO)



Rewriting rule with interface K

rewriting step $G \Rightarrow H$

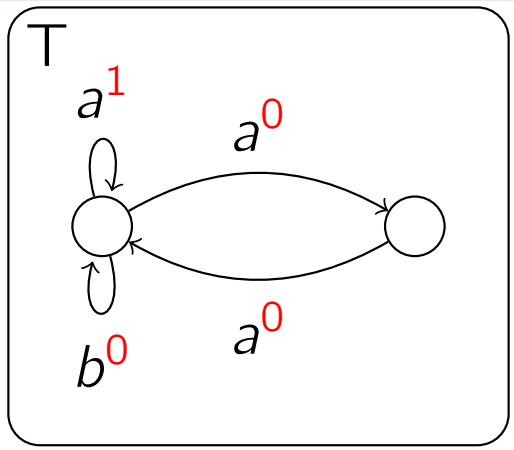


Remark:

- No implicit edge deletion

Weighted Type Graph

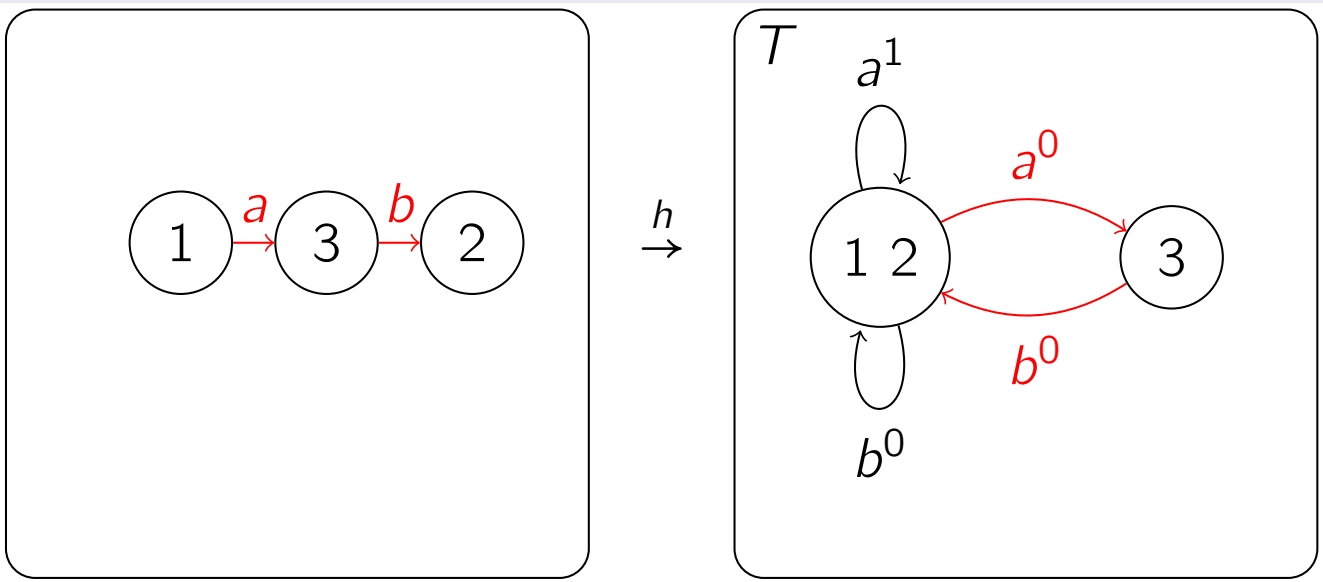
In the context of graph rewriting, a weighted type graph is a graph with weights on its edges.



Morphism weight

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

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

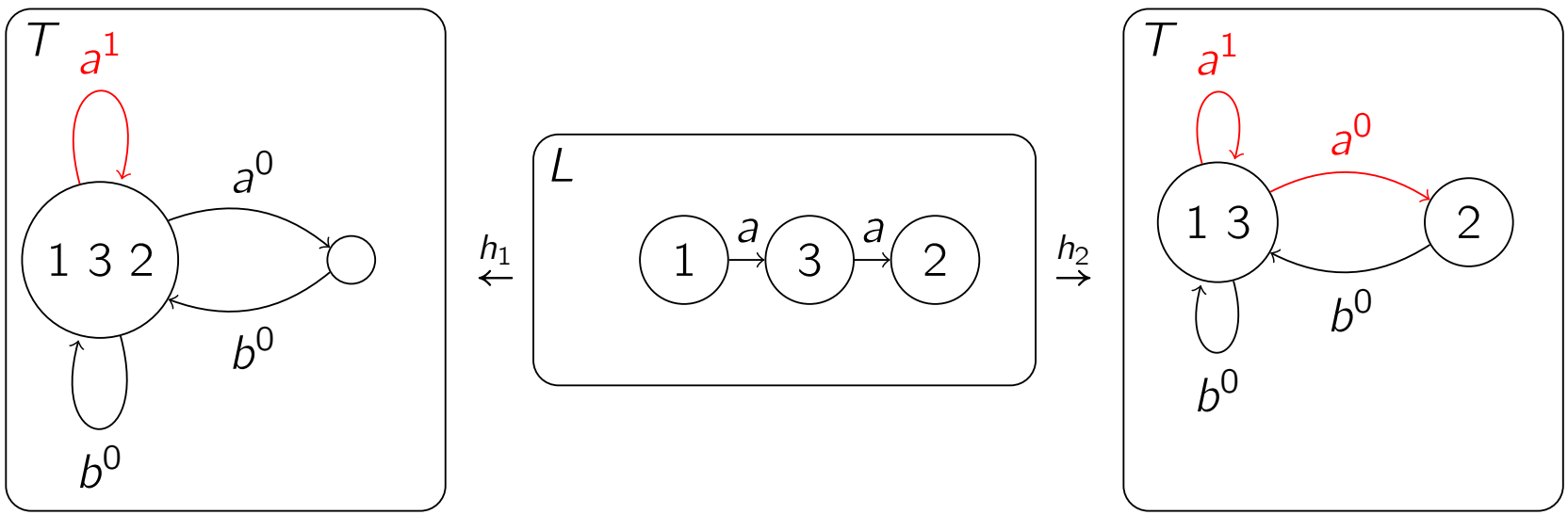


$\mathsf{weight}_T(h) = 0 + 0 = 0$

Graph weight

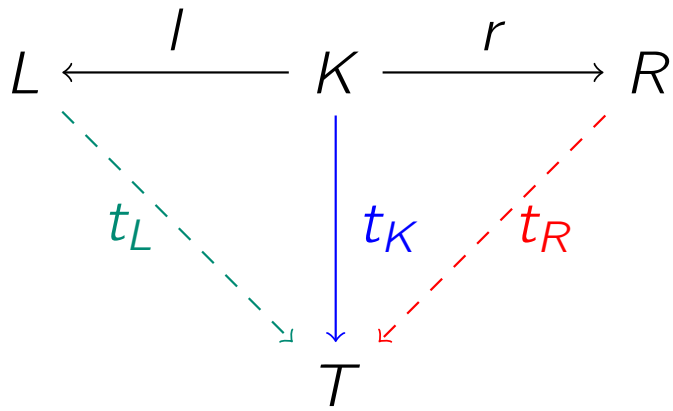
The weight of a graph L is

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



$$\text{weight}_T(L) = \min\{\text{weight}_T(h_1), \text{weight}_T(h_2)\} = \min\{1+1, 1+0\} = 1$$

Termination Criterion



Every rewriting step strictly decreases the weight if

- for all t_K, if there is t_L making $\triangle KLT$ commutative, then

$$\begin{aligned} &\min\{\text{weight}_T(t_L) \mid \triangle KLT \text{ commutative}\} \\ >&\min\{\text{weight}_T(t_R) \mid \triangle KRT \text{ commutative}\} \end{aligned}$$

How to find 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 theory 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

Problem of the size of the search space

With natural numbers as weights:

# nodes (k)	# labels (m)	# weights	# possibilities
2	2	2	$\approx 10^4$
3	3	3	$\approx 10^{21}$
3	3	10	$\approx 10^{45}$
3	3	100	$\approx 10^{87}$
4	4	4	$\approx 10^{57}$
4	4	10	$\approx 10^{95}$
4	4	100	$\approx 10^{181}$

Problems can solved by Z3 in exponential-time with respect to the number of variables $2k^2m$.

Solution

Using positive real numbers as weights

Constraint: there is $\delta > 0$, every rewriting step strictly decreases the weight by at least δ.

What is the impact on complexity ?

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 theory~~ existential theory of the reals with binary variables:

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

Challenge:

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

Complexity comparison

With natural numbers as weights:

# nodes (k)	# labels (m)	# weights	# possibilities
2	2	2	$\approx 10^4$
3	3	3	$\approx 10^{21}$
3	3	10	$\approx 10^{45}$
3	3	100	$\approx 10^{87}$
4	4	4	$\approx 10^{57}$
4	4	10	$\approx 10^{95}$
4	4	100	$\approx 10^{181}$

With real numbers as weights:

# nodes (k)	# labels (m)	# linear programs in $\mathbb{R}$
2	2	$\approx 10^2$
3	3	$\approx 10^8$
4	4	$\approx 10^{19}$

Linear programs can be solved in polynomial-time on average.

Experimental Results

	A	a	T	t	N	n
[EO24, Example 6.3]					2.74	1.16
[EO24, Example D.3]	2.25	1.18			2.24	1.18
[Plu95, Example 3.8]	2.95	1.90	2.94	1.87	3.49	1.87
[Plu18, Example 4]	4.26	3.19	4.24	3.13	5.82	timeout
[Plu18, Example 5]	5.54	5.55	5.53	5.50	9.11	5.62
[Bru+15, Example 4]	2.44	2.46	2.47	2.54	4.58	2.46
[Bru+15, Example 5]					7.80	timeout
[Bru+15, Example 6]					9.75	timeout
[Bru14, Example 1]	2.26	1.18			2.24	1.18
[Bru14, Example 4]	2.25	1.22	2.24	1.18	2.25	1.19
[Bru14, Example 5]	4.23	3.23	4.25	3.28	5.82	timeout

"A", "T", "N" : different configurations with weights over the natural numbers. "a", "t", "n" : corresponding configurations over the real numbers.

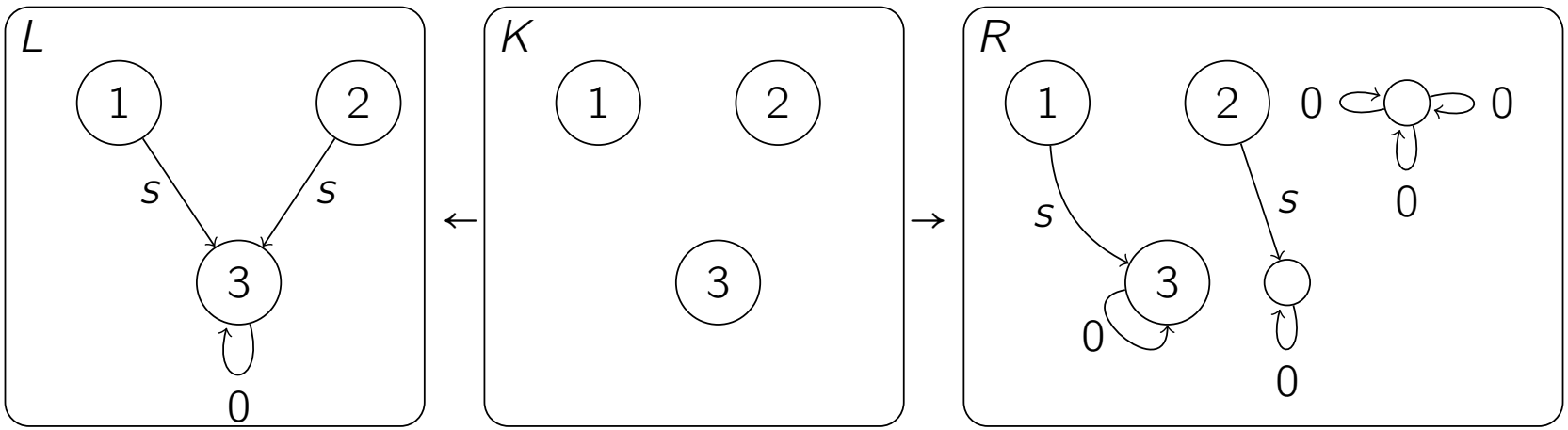
Implementation

LyonParallel

Search parallel with 6 configurations

Z3 for constraint solving

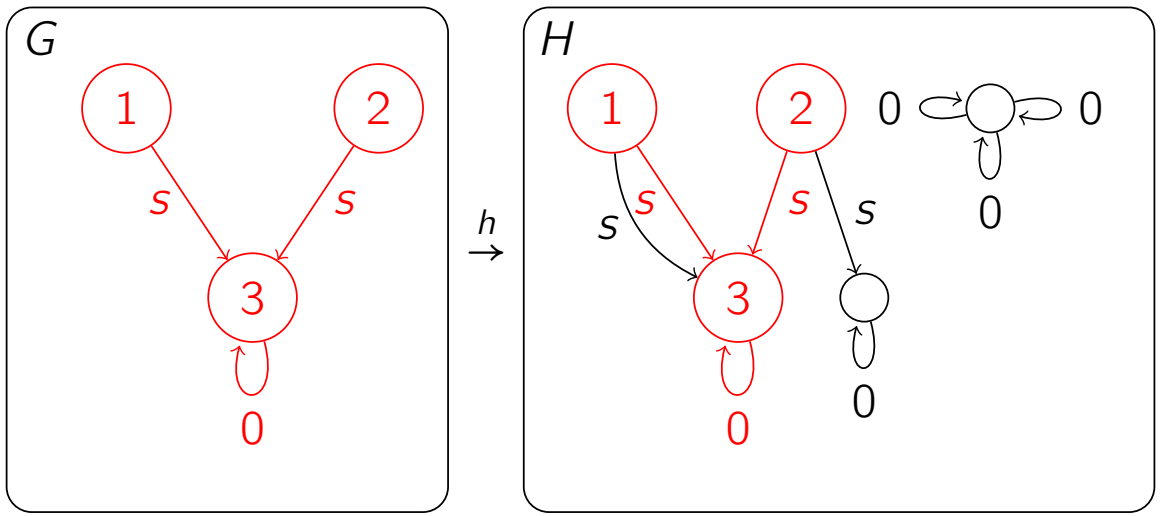
A Limitation of the Weighted Type Graph Method



All existing automated methods fail.

Intuition: the number of morphisms from $\bullet \xrightarrow{s} \bullet \xleftarrow{s} \bullet$ strictly decreases.

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

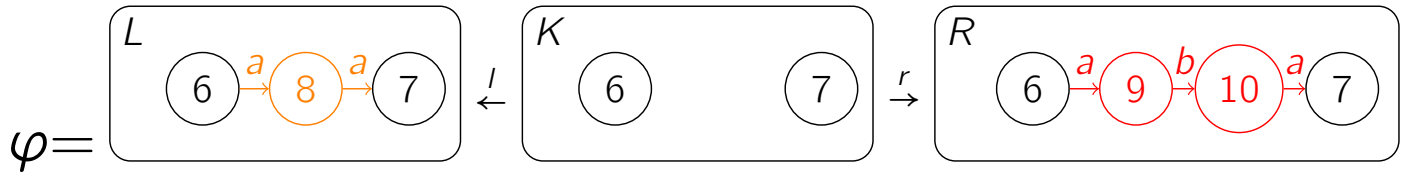


Remarks:

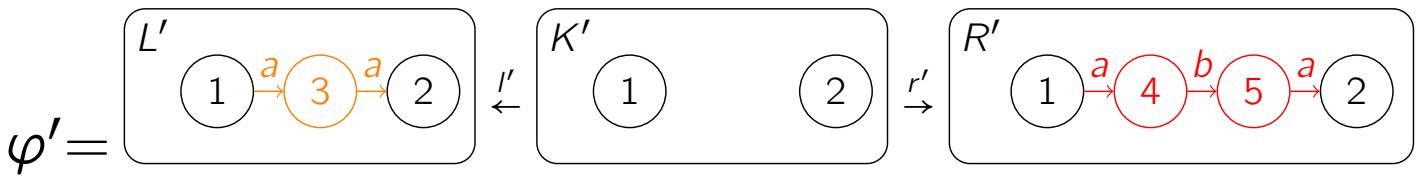
- G is a subgraph of H.

A rewriting step with a running example

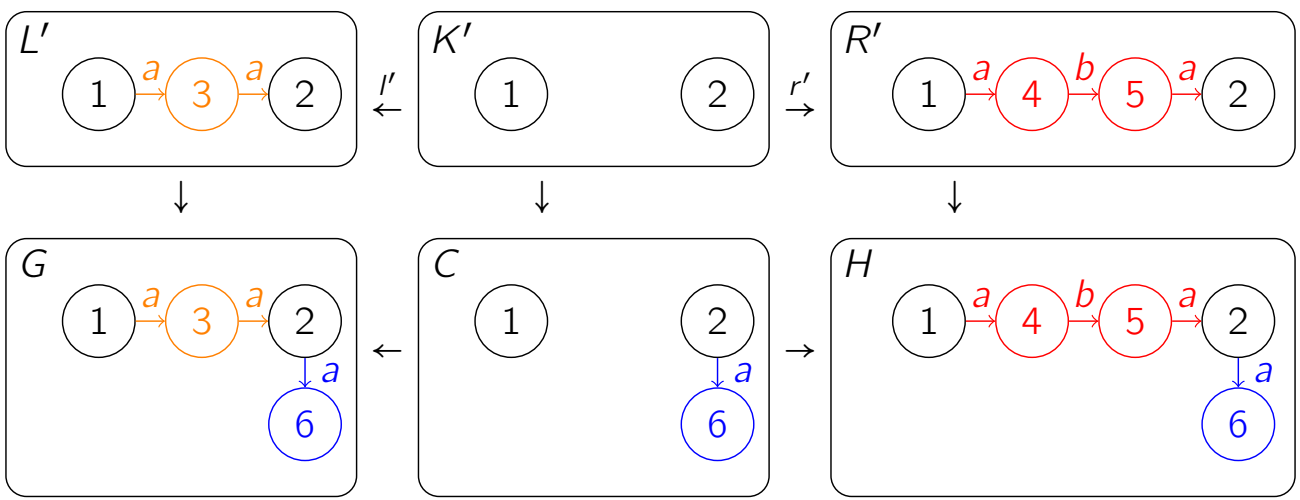
A rewriting rules consists of two inclusions.



An equivalent rewriting rule expresses the same transformation.

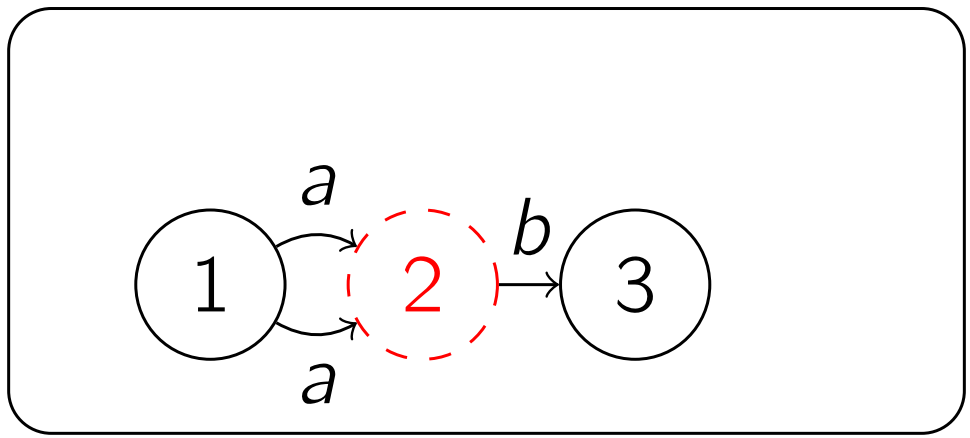


A rewriting step with φ is defined by a DPO diagram with inclusions and φ'.



Pre-graphs

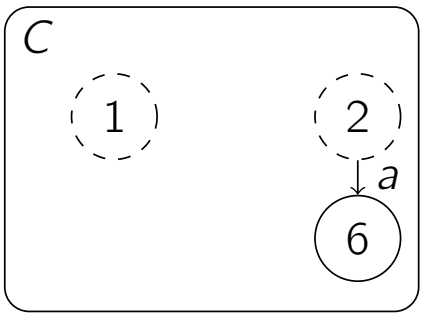
Pre-graphs are graphs with missing nodes and dangling edges.

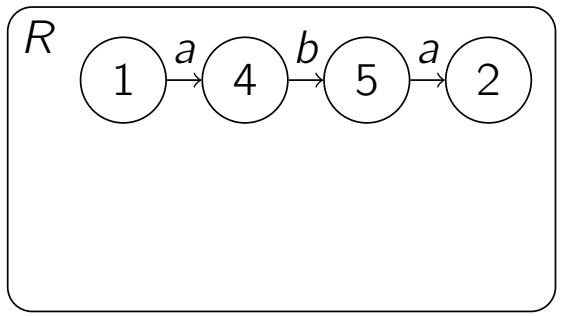


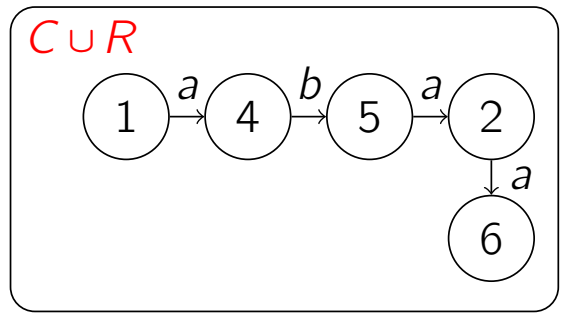
- 1 missing node in red,
- all edges are dangling,
- 2 existing nodes.

Pre-graph operations

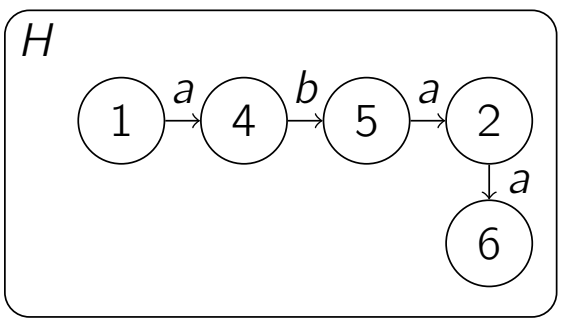
Union of two pre-graphs $C \subseteq G$ and $R \subseteq G$, denoted $C \cup R$.

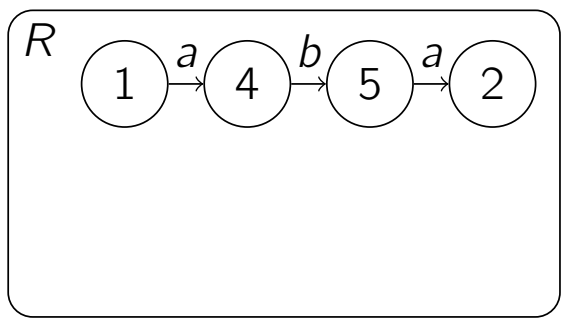


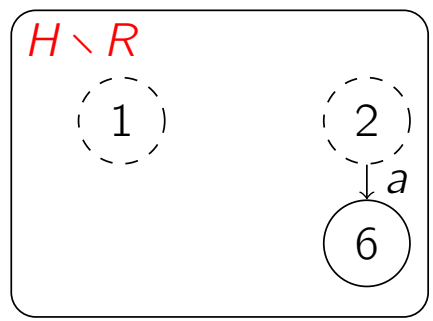




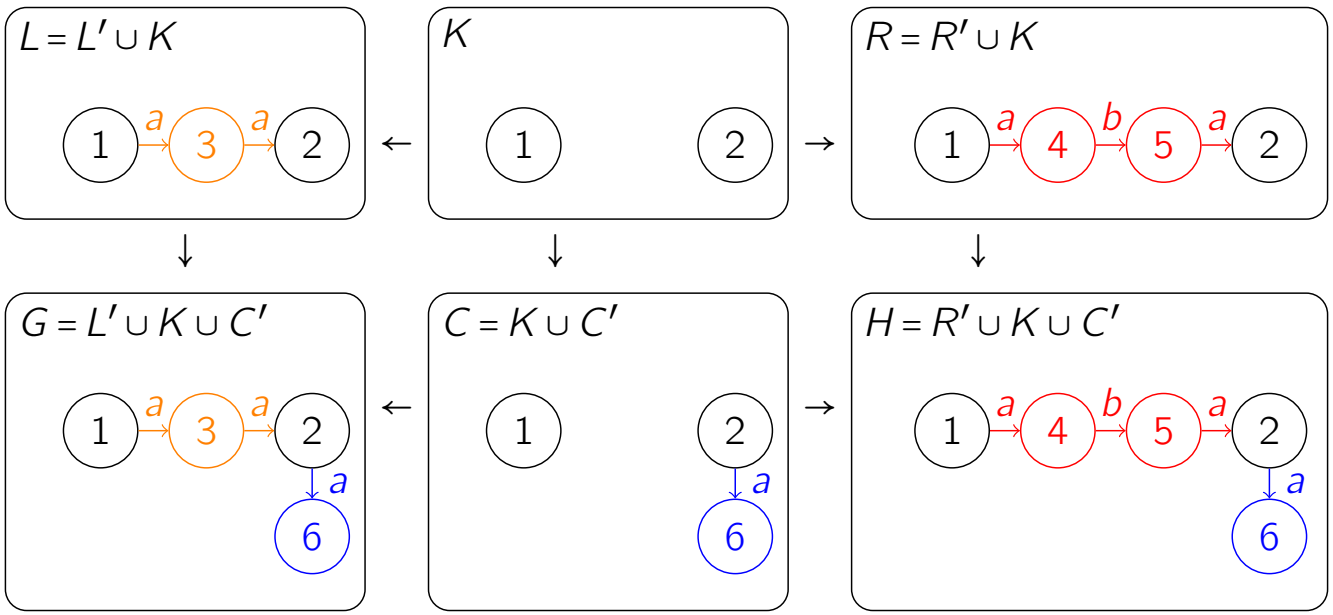
Relative complement of R in H where $R \subseteq H$, denoted $H \setminus R$.



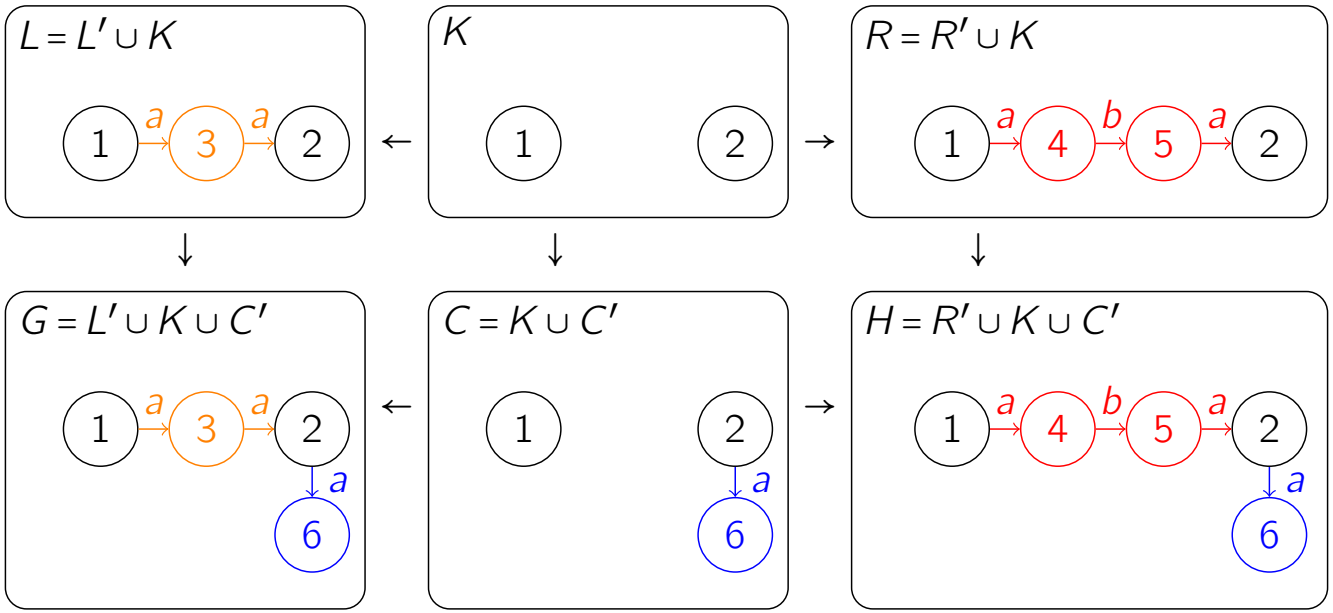


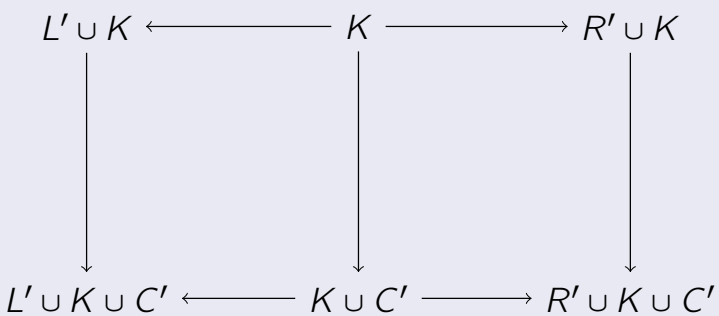


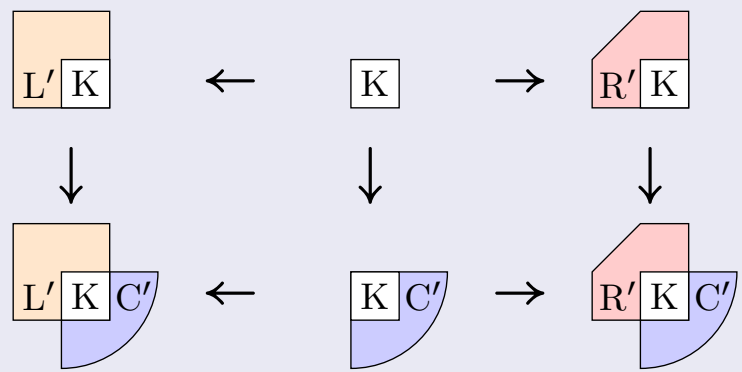
Decomposition of graphs in rewriting steps



Decomposition of graphs in rewriting steps

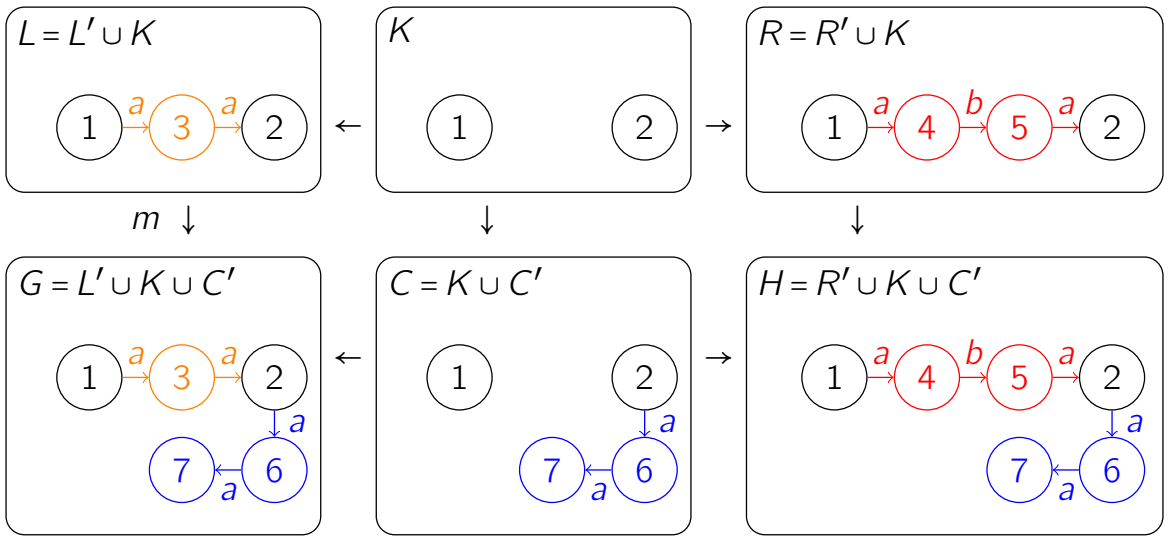




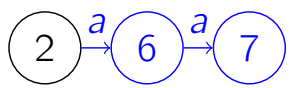
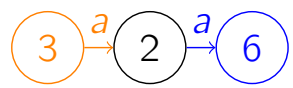
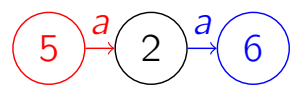


Example

Let X be the graph $\bullet \xrightarrow[a]{} \bullet \xrightarrow[a]{} \bullet$.

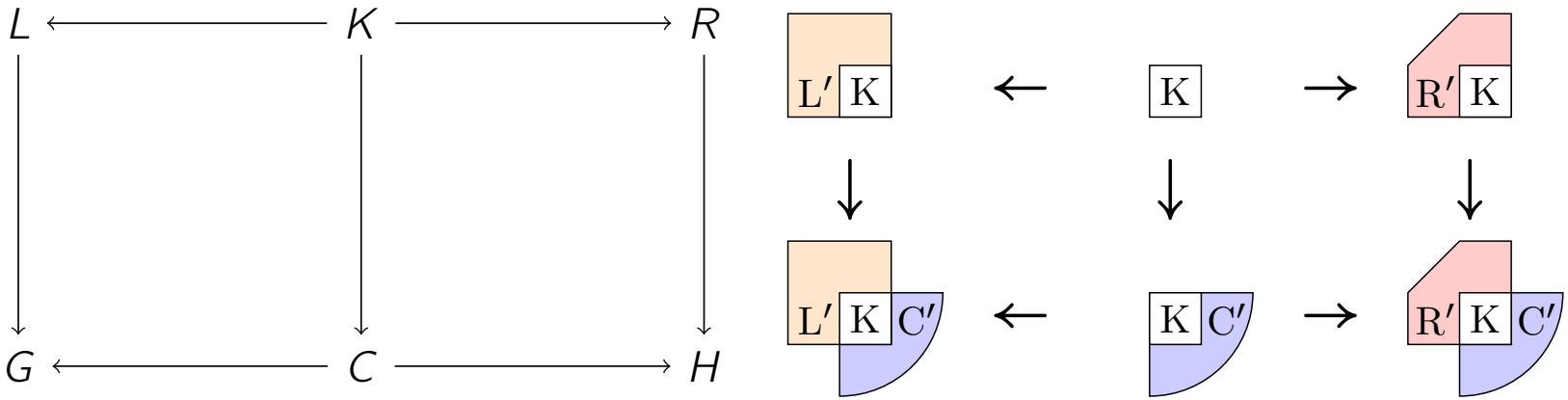


The morphisms from X has their images:

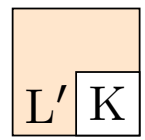
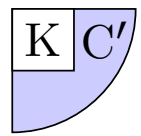
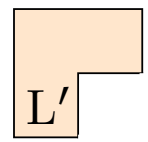
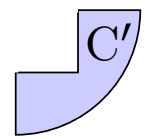
- in G and L: 
- in H and R: None
- shared by G and H: 
- formed by subgraphs of L and C: 
- formed by subgraphs of R and C: 

Implicit, Explicit and Shared Occurrences

An X-occurrence is an injective morphism from X.



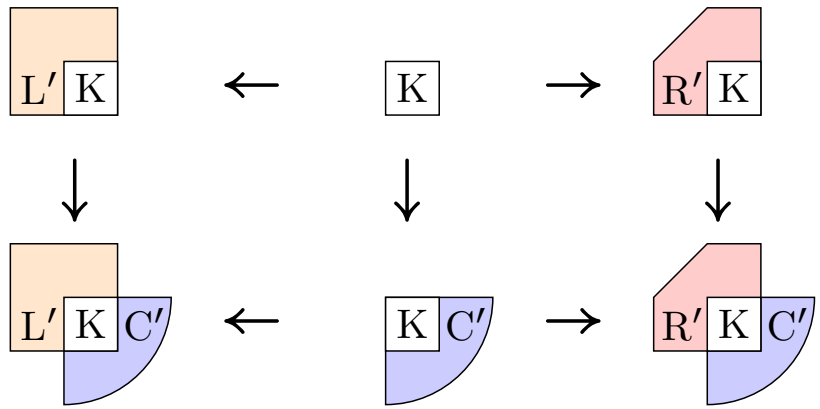
An X-occurrence is

- 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.

A sufficient condition for termination

φ terminates if for all rewriting step:

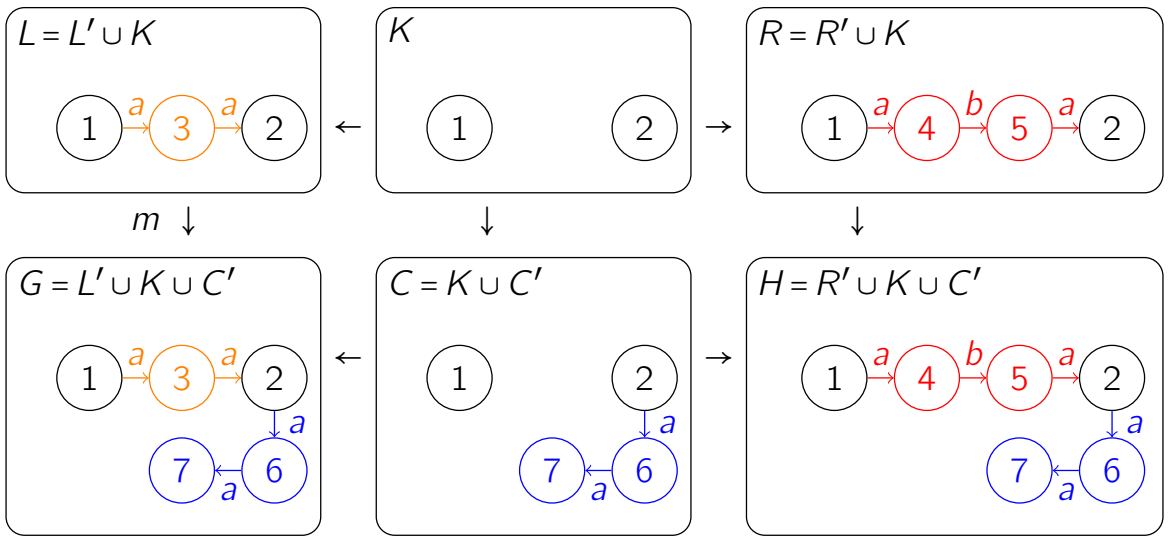


the following holds:

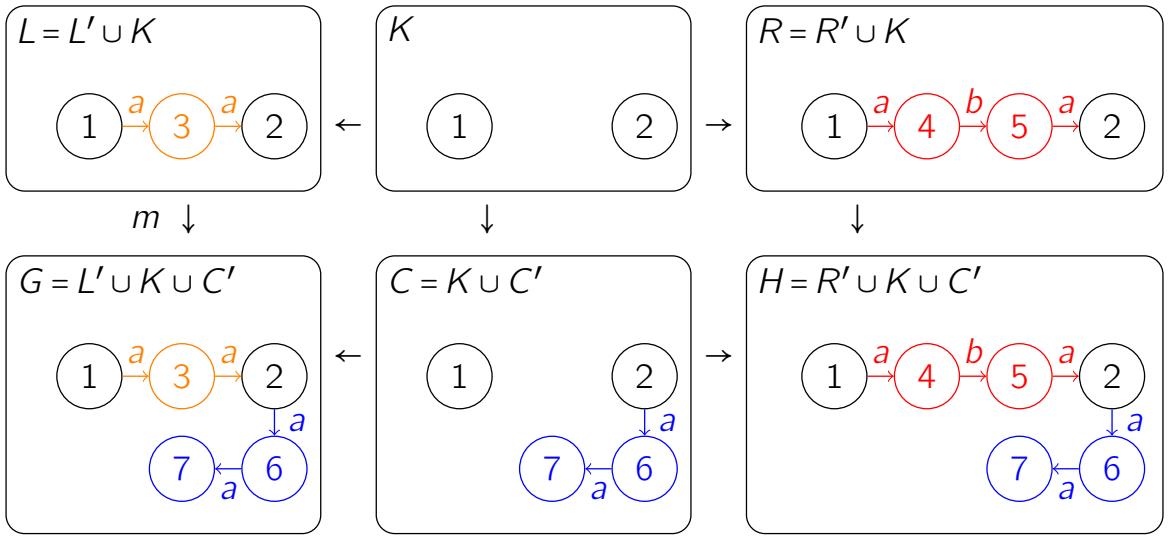
1. $|\text{explicit } X\text{-occurrences in } G| > |\text{explicit } X\text{-occurrences in } H|$;
2. $|\text{implicit } X\text{-occurrences in } G| \geq |\text{implicit } X\text{-occurrences in } H|$.

The first condition is straightforward.
Challenge: the second condition.

Analysis of Implicit Occurrences



Analysis of Implicit Occurrences

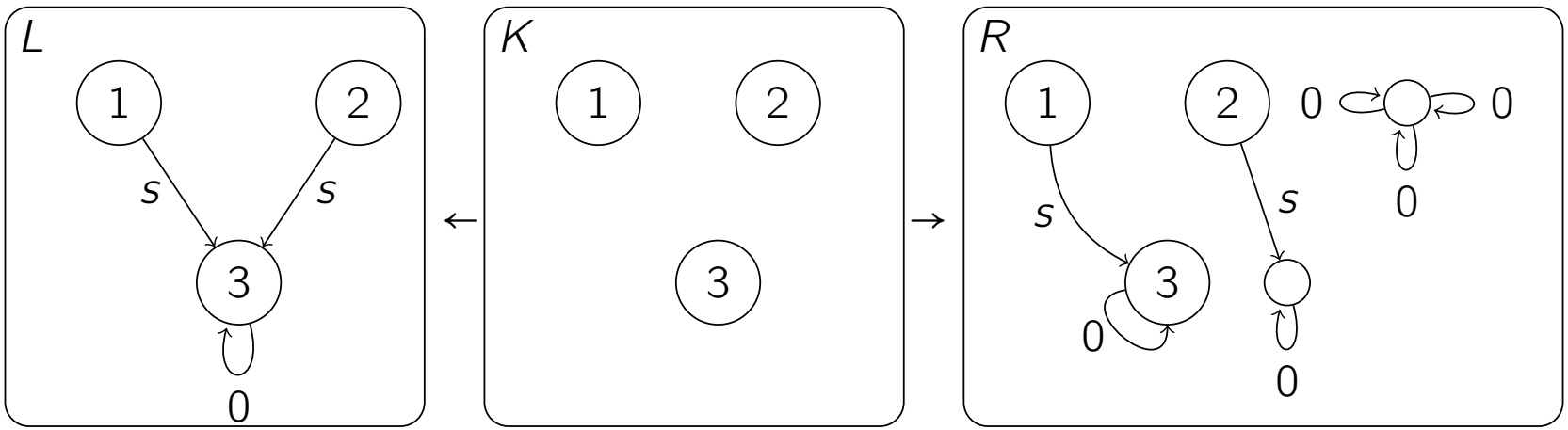


Lemma (More X-occurrences before rewriting)

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 distinct subgraphs in L while preserving the interface elements.*

A Limitation of the Weighted Type Graph Method



Termination proved :

- Number of explicit occurrences from $\bullet \xrightarrow{s} \bullet \xleftarrow{s} \bullet$ strictly decreases
- More implicit occurrences before rewriting

Comparison with the weighted type graph method

Morphism counting method has simpler conditions.

LyonParallel

Automated tool in Ocaml

Iterative elimination of graph rewriting rules

Available : `https://github.com/Qi-tchi/LyonParallel`

Conclusion and Future Work

Contributions:

- an extension of an existing method for greater usability
- a new termination criterion for greater power
- an automated tool for termination analysis

Future work:

- formal verification of the methods
- morphism counting with multiple forbidden contexts
- extension to other rewriting approaches

References

[Bru+15] H. J. Sander Bruggink et al. "Proving Termination of Graph Transformation Systems using Weighted Type Graphs over Semirings". In: *CoRR* abs/1505.01695 (2015). arXiv: 1505.01695.

[Bru14] H. J. Sander Bruggink. "Towards Process Mining with Graph Transformation Systems". In: *Graph Transformation*. Ed. by Holger Giese and Barbara König. Cham: Springer International Publishing, 2014, pp. 253–268. ISBN: 978-3-319-09108-2.

[EO24] J. Endrullis and R. Overbeek. *Generalized Weighted Type Graphs for Termination of Graph Transformation Systems*. 2024. arXiv: 2307.07601v2 [cs.LO].

[Plu18] Detlef Plump. "Modular Termination of Graph Transformation". In: *Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig*. Ed. by Reiko Heckel and Gabriele Taentzer.