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

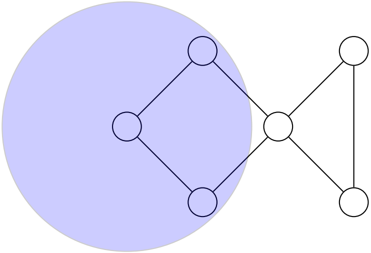
Distributed/concurrent systems are everywhere

Failures can be catastrophic

Ensuring correctness is hard

This thesis: automated verification

Graph Transformation

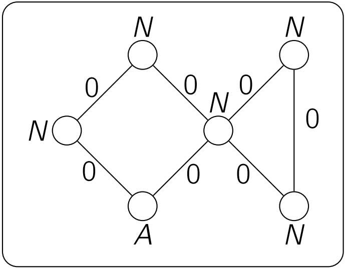


Graph rewriting:

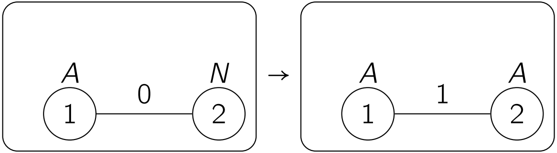
- computational units → nodes
- communication channels → edges
- system states → graphs
- algorithm behaviors → graph transformation rules

Graph Transformation

Graph representing a configuration of a distributed system:

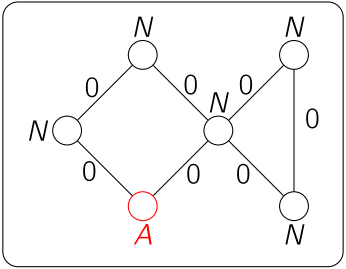


Graph transformation rule:

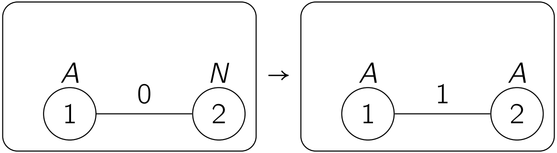


Graph Transformation

Graph representing a configuration of a distributed system:

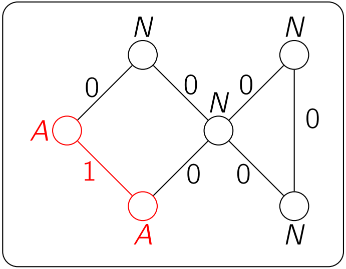


Graph transformation rule:

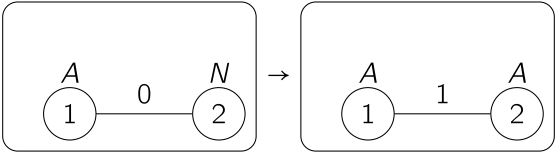


Graph Transformation

Graph representing a configuration of a distributed system:

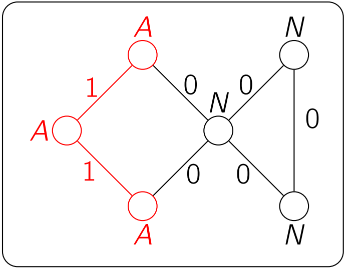


Graph transformation rule:

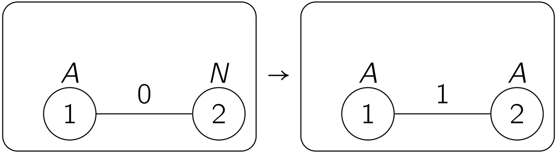


Graph Transformation

Graph representing a configuration of a distributed system:

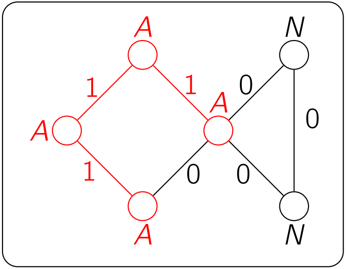


Graph transformation rule:

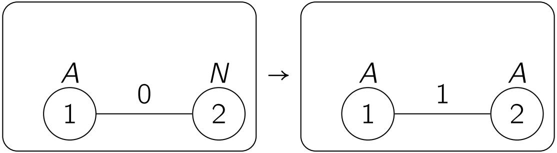


Graph Transformation

Graph representing a configuration of a distributed system:

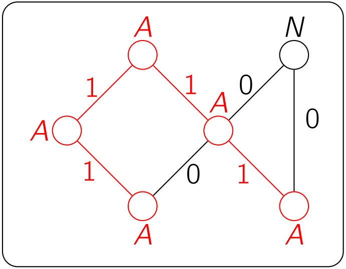


Graph transformation rule:

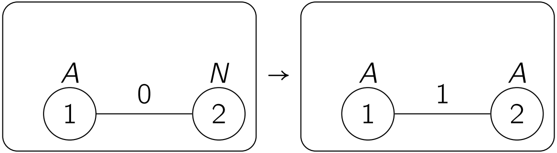


Graph Transformation

Graph representing a configuration of a distributed system:

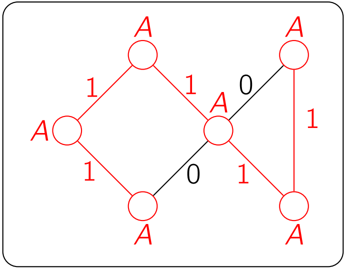


Graph transformation rule:

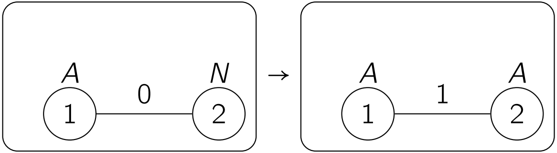


Graph Transformation

Graph representing a configuration of a distributed system:

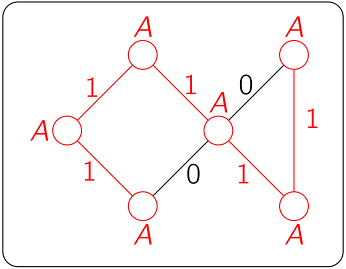


Graph transformation rule:

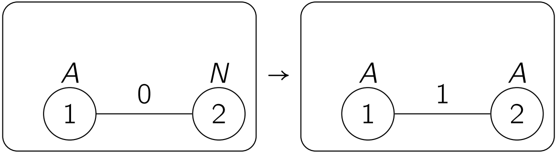


Graph Transformation

Graph representing a configuration of a distributed system:

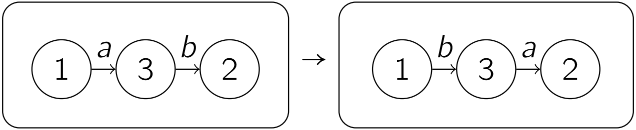


Graph transformation rule:

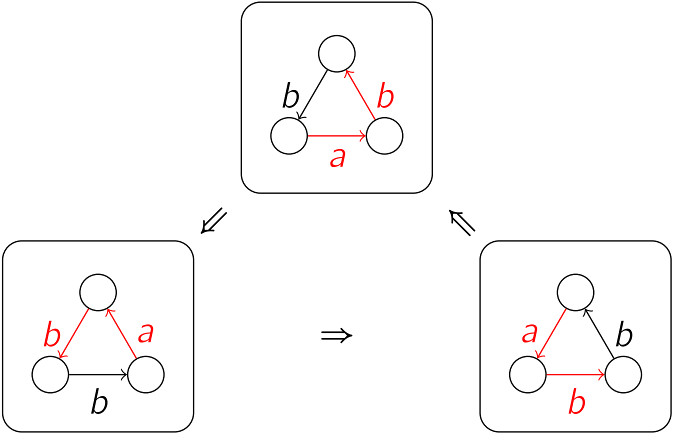


Question: does the transformation process terminate for any initial graph?

A non-termination case



Loop:



Termination of a Graph Transformation System

- 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."

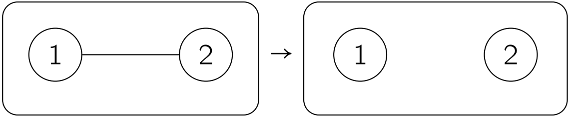
- Undecidable in general
- How to prove termination automatically?

Automated Termination Proofs

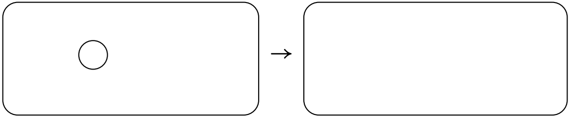
Termination by interpretations:

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

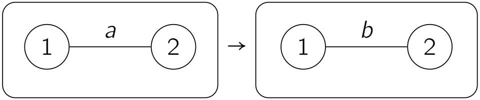
Number of edges:



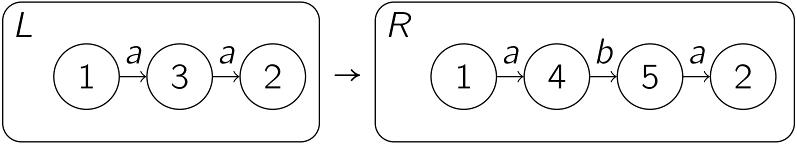
Number of nodes:



Number of edges labeled by a:

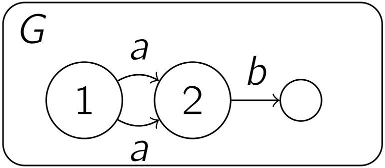


A non-trivial case



Structure of the presentation

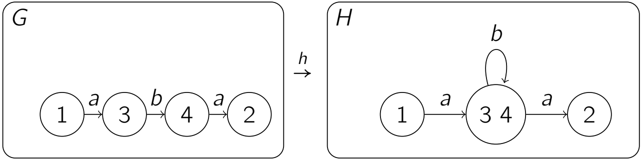
Graphs : finite, directed, edge-labeled multigraphs



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



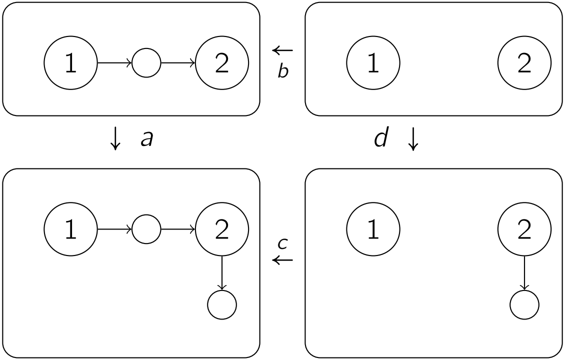
Remarks:

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

Commutative Diagram

$$
\begin{tikzcd}
L & K \arrow[l, "b"'] \arrow[d, "d"'] \\
G & C \arrow[l, "c"]
\arrow[from=1-1, to=2-1, "a"]
\end{tikzcd}
$$

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



Pushouts: gluing graphs along a common part

Definition

The **pushout** of (α, β) is

$$\begin{tikzcd} & A \arrow[dl, "\alpha"'] \arrow[dr, "\beta"] & \\ B & & C \end{tikzcd}$$

Pushouts: gluing graphs along a common part

Definition

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

- ABDC is commutative,

$$\begin{tikzcd}
& A \arrow[dl, "\alpha"'] \arrow[dr, "\beta"] & \\
B \arrow[dr, "\beta'"'] \arrow[rr, phantom, "\mathrm{PO}"] & & C \arrow[dl, "\alpha'"] \\
& D &
\end{tikzcd}$$

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.

$$
\begin{tikzcd}
 & A \arrow[dl, "\alpha"'] \arrow[dr, "\beta"] & \\
B \arrow[dr, "\beta'"'] \arrow[ddr, bend right, "\gamma'"'] & \mathrm{PO} & C \arrow[dl, "\alpha'"] \arrow[ddl, bend left, "\gamma"] \\
 & D \arrow[d, dashed, "\delta"] & \\
 & E &
\end{tikzcd}
$$

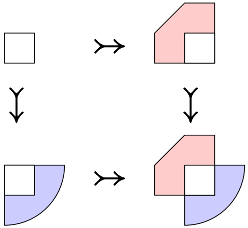
Pushouts: gluing graphs along a common part

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.

$$\begin{tikzcd}
& A \arrow[dl, "\alpha"'] \arrow[dr, "\beta"] & \\
B \arrow[dr, "\beta'"'] \arrow[ddr, bend right, "\gamma'"'] & \mathrm{PO} & C \arrow[dl, "\alpha'"] \arrow[ddl, bend left, "\gamma"] \\
& D \arrow[d, dashed, "\delta"] & \\
& E &
\end{tikzcd}$$

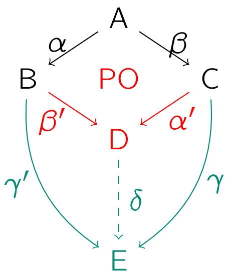


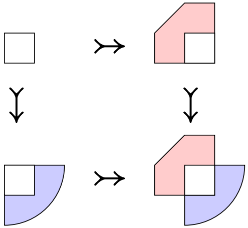
Pushouts: gluing graphs along a common part

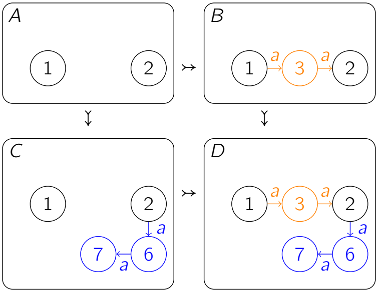
Definition

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.



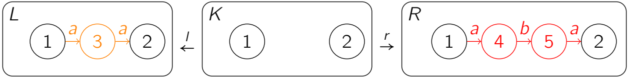




Graph rewriting with double-pushout approach (DPO)

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

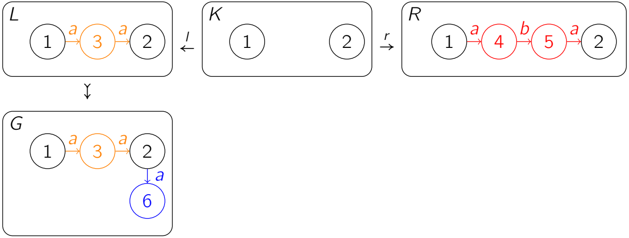
Rewriting rule with interface K



Graph rewriting with double-pushout approach (DPO)

$$\begin{tikzcd} L \arrow[d, tail] & K \arrow[l, "l"'] \arrow[r, "r"] & R \\ G & & \end{tikzcd}$$

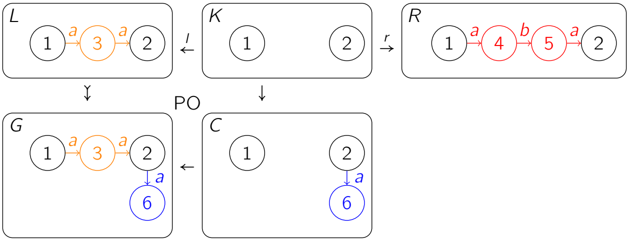
Rewriting rule with interface K



Graph rewriting with double-pushout approach (DPO)

$$\begin{tikzcd} L \arrow[d, tail] \arrow[dr, phantom, "\text{PO}"] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] & R \\ G & C \arrow[l] & \end{tikzcd}$$

Rewriting rule with interface K

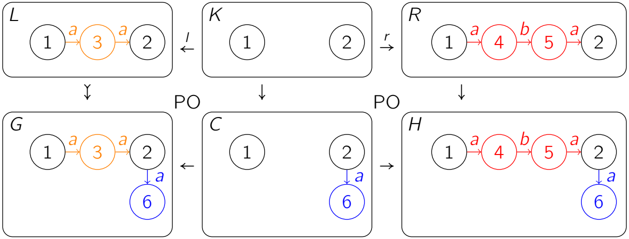


Graph rewriting with double-pushout approach (DPO)

$$\begin{tikzcd} L \arrow[d, tail] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d] \arrow[dl, phantom, "\mathrm{PO}"] \arrow[dr, phantom, "\mathrm{PO}"] & R \arrow[d] \\ G & C \arrow[l] \arrow[r] & H \end{tikzcd}$$

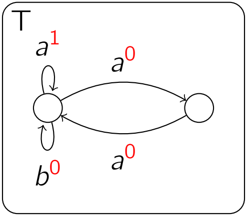
Rewriting rule with interface K

rewriting step $G \Rightarrow H$



Weighted Type Graph

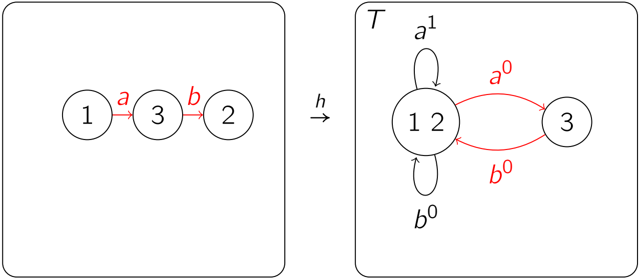
In the context of graph rewriting, a weighted type graph is a graph with weights on its edges.



Morphism weight

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

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

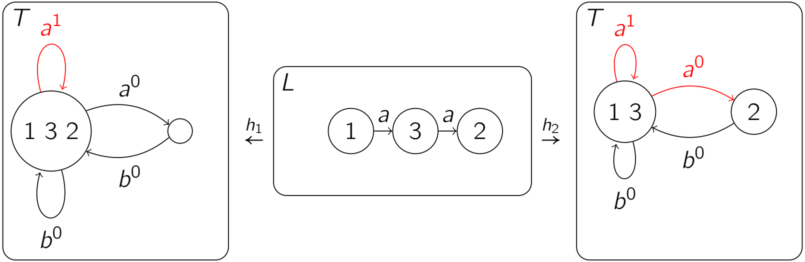


$\text{weight}_T(h) = 0 + 0 = 0$

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

$$\begin{tikzcd}
L \arrow[dr, dashed, "t_L"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "t_K"] & R \arrow[dl, dashed, "t_R"] \\
& T &
\end{tikzcd}$$

Every rewriting step strictly decreases the weight if

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

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

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:

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

Challenge:

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

Termination Criterion

$$
\begin{tikzcd}
L \arrow[dr, dashed, "t_L"'] & K \arrow[l, "l"'] \arrow[r, "r"] \arrow[d, "t_K"] & R \arrow[dl, dashed, "t_R"] \\
 & T &
\end{tikzcd}
$$

Every rewriting step strictly decreases the weight if

- there is $\delta > 0$, for all t_K, if there is t_L making $\triangle KLT$ commute, then

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

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

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
[BKZ14, Example 1]	2.26	1.18			2.24	1.18
[BKZ14, Example 4]	2.25	1.22	2.24	1.18	2.25	1.19
[BKZ14, 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.

Analysis and Implementation choices

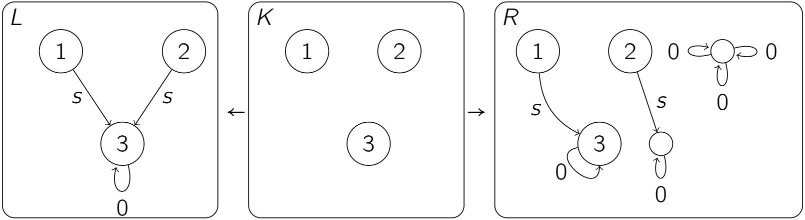
Observations from experiments:

- advantages:
  - less time in average to find a suitable weighted type graph
  - no need to guess maximum edge weight
- disadvantage:
  - impossible to further constrain weight sets to extremely small sets (e.g. with two elements).

Implementation choices:

- search in parallel using all approaches
- Z3 for checking satisfiability

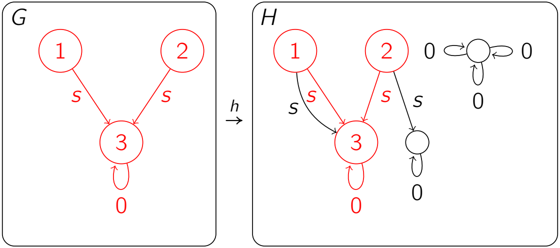
A Limitation of the Weighted Type Graph Method



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.

Graph rewriting rule

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

$$\alpha = \boxed{L \quad 1 \xrightarrow{a} 3 \xrightarrow{a} 2} \overset{l}{\leftarrow} \boxed{K \quad 1 \qquad\qquad 2} \overset{r}{\rightarrow} \boxed{R \quad 1 \xrightarrow{a} 4 \xrightarrow{b} 5 \xrightarrow{a} 2}$$

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

$$\begin{tikzcd}
L' \arrow[d, "a"] & K' \arrow[l, "l'"'] \arrow[r, "r'"] \arrow[d, "b"] \arrow[l, phantom, shift right=6, "="] \arrow[r, phantom, shift right=6, "="] & R' \arrow[d, "c"'] \\
L & K \arrow[l, "l"'] \arrow[r, "r"] & R
\end{tikzcd}$$

$$\alpha' = \boxed{L' \quad 6 \xrightarrow{a} 7 \xrightarrow{a} 8} \overset{l'}{\leftarrow} \boxed{K' \quad 6 \qquad\qquad 7} \overset{r'}{\rightarrow} \boxed{R' \quad 6 \xrightarrow{a} 9 \xrightarrow{b} 10 \xrightarrow{a} 7}$$

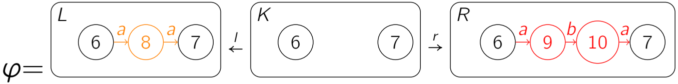
Graph Rewriting

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:

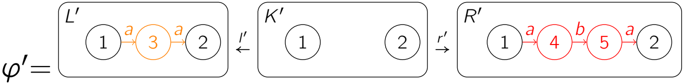
$$\begin{tikzcd} L' \arrow[d] & K' \arrow[l, "l'"'] \arrow[r, "r'"'] \arrow[d] & R' \arrow[d] \\ G & C \arrow[l] \arrow[r] & H \end{tikzcd}$$

A rewriting step with a running example

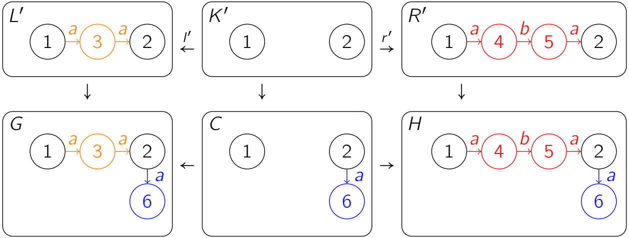
Rewriting rule:



An equivalent rewriting rule:

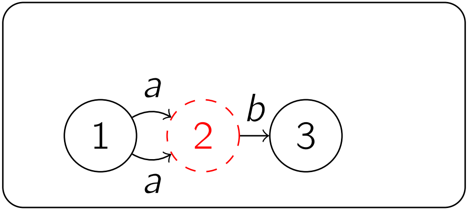


A rewriting step $G \Rightarrow_{\varphi} H$:



Pre-graphs

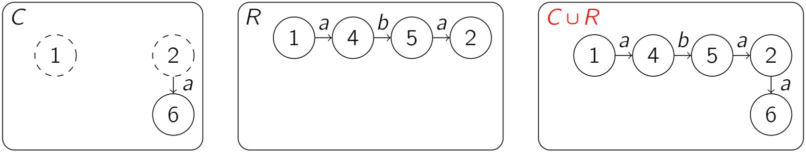
Pre-graphs are graphs with missing nodes and dangling edges.



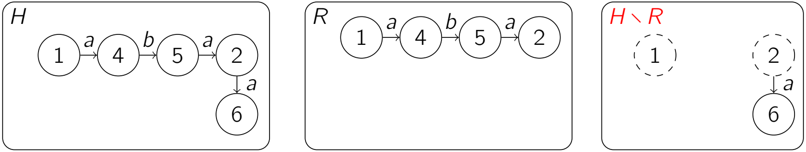
- 1 missing node in red,
- all edges are dangling,
- 2 existing nodes.

Pre-graph operations

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



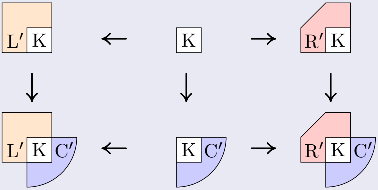
Relative complement of R in H where $R \subseteq H$, denoted $H \setminus R$.



Analysis of rewriting steps

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

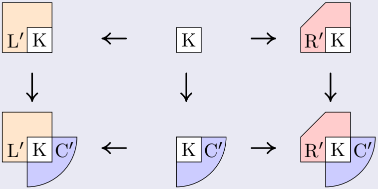
$$\begin{tikzcd}
L' \cup K \arrow[d] & K \arrow[l] \arrow[r] \arrow[d] & R' \cup K \arrow[d] \\
L' \cup K \cup C' & K \cup C' \arrow[l] \arrow[r] & R' \cup K \cup C'
\end{tikzcd}$$

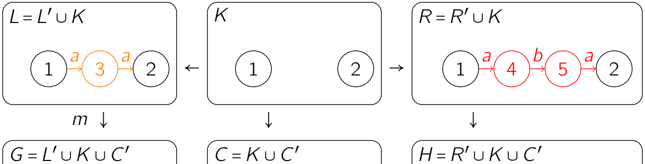


Analysis of rewriting steps

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

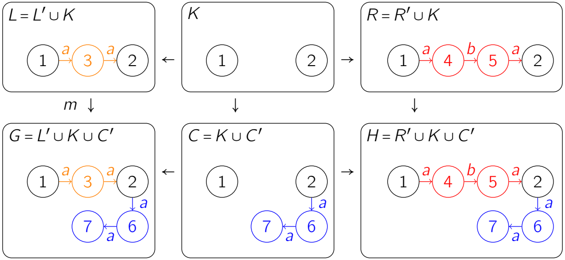
$$\begin{tikzcd}
L' \cup K \arrow[d] & K \arrow[l] \arrow[r] \arrow[d] & R' \cup K \arrow[d] \\
L' \cup K \cup C' & K \cup C' \arrow[l] \arrow[r] & R' \cup K \cup C'
\end{tikzcd}$$





Example

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



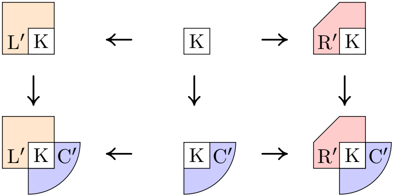
The morphisms from X has their images as follows:

- in G and L: $1 \xrightarrow{a} 3 \xrightarrow{a} 2$
- in H and R: None
- shared by G and H: $2 \xrightarrow{a} 6 \xrightarrow{a} 7$
- formed by subgraphs of L and C: $3 \xrightarrow{a} 2 \xrightarrow{a} 6$
- formed by subgraphs of R and C: $5 \xrightarrow{a} 2 \xrightarrow{a} 6$

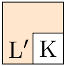
Implicit, Explicit and Shared Occurrences

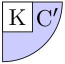
An *X-occurrence* is an injective morphism from X.

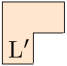
$$\begin{tikzcd} L & K \arrow[l] \arrow[r] & R \\ G \arrow[u, leftarrow] & C \arrow[l] \arrow[r] \arrow[u, leftarrow] & H \arrow[u, leftarrow] \end{tikzcd}$$

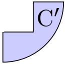


An X-occurrence is

- explicit if **Im**(x) is included in 

- shared if **Im**(x) is included in 

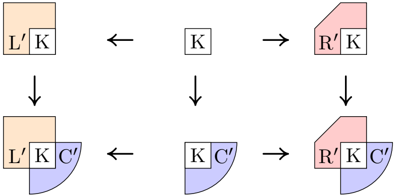
- implicit if **Im**(x) has elements in both 

and 

Similarly, in H.

A sufficient condition for termination

φ terminates if for all rewriting step:



the following holds:

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

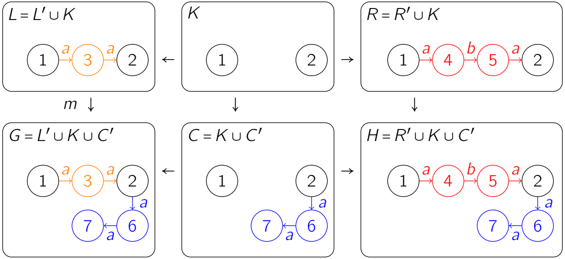
The first condition is straightforward.
Challenge: Establishing the second condition.

Analysis of Implicit Occurrences in G and H

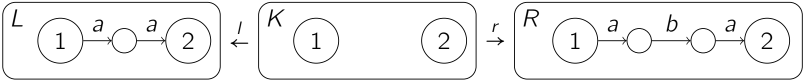
Lemma (More X-occurrences before rewriting)

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.*



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:

$$1 > 0$$

LyonParallel

- Automated tool in Ocaml
- Iterative elimination of graph rewriting rules
- 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 multiple forbidden contexts,
- Extension to other rewriting approaches.

References

[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.

[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.

[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