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

Motivation & Goal

- Distributed/concurrent systems are everywhere
- Failures can be catastrophic

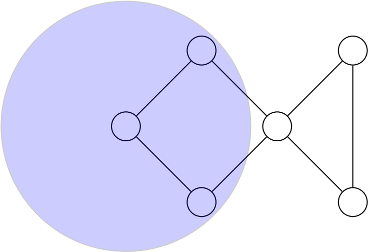
Motivation & Goal

- Distributed/concurrent systems are everywhere
- Failures can be catastrophic
- Ensuring correctness is hard

Motivation & Goal

- Distributed/concurrent systems are everywhere
- Failures can be catastrophic
- Ensuring correctness is hard
- This thesis: automated, rigorous verification

Graph Transformation

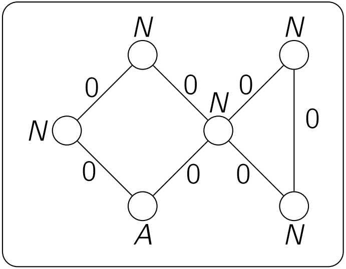


Graph rewriting:

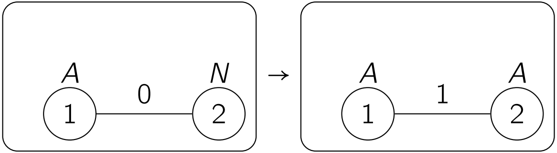
- computational units → nodes
- communication channels → edges
- system states → graphs
- algorithm behaviors → graph transformation rules

Graph Transformation

Graph representing a configuration of a distributed system:

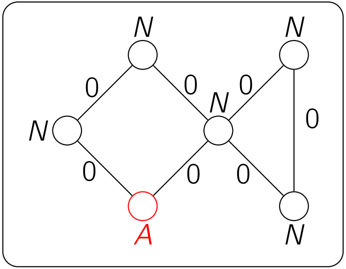


Graph transformation rule:

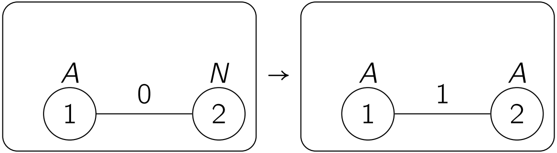


Graph Transformation

Graph representing a configuration of a distributed system:

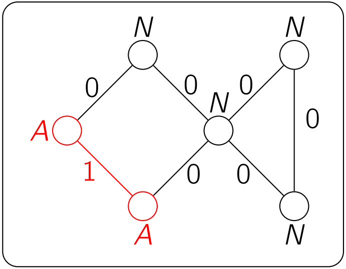


Graph transformation rule:

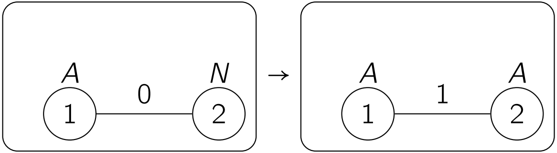


Graph Transformation

Graph representing a configuration of a distributed system:

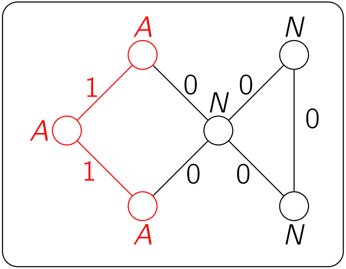


Graph transformation rule:

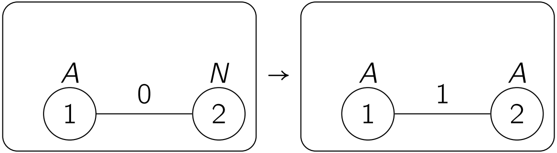


Graph Transformation

Graph representing a configuration of a distributed system:

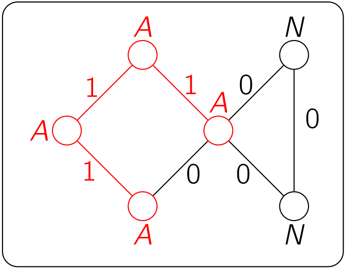


Graph transformation rule:

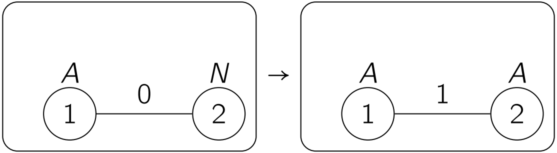


Graph Transformation

Graph representing a configuration of a distributed system:

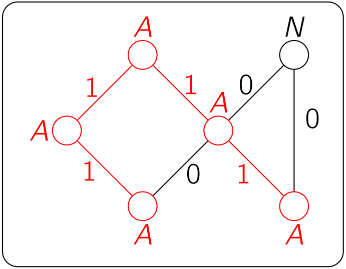


Graph transformation rule:

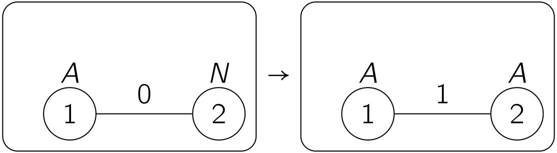


Graph Transformation

Graph representing a configuration of a distributed system:

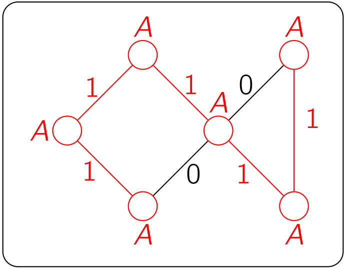


Graph transformation rule:

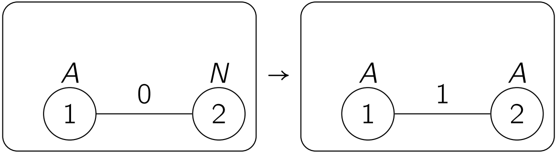


Graph Transformation

Graph representing a configuration of a distributed system:

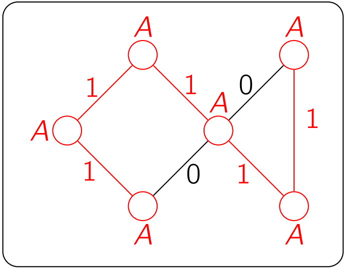


Graph transformation rule:

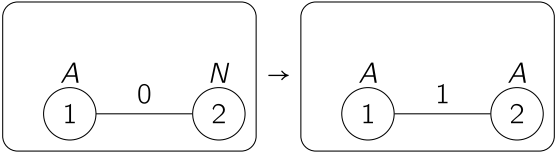


Graph Transformation

Graph representing a configuration of a distributed system:



Graph transformation rule:



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

Termination of graph transformation systems

- $\mathcal{R}$: a set of rules
- No graph G_0 can be transformed forever

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

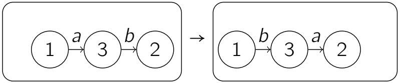
  when using the non-deterministic strategy

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

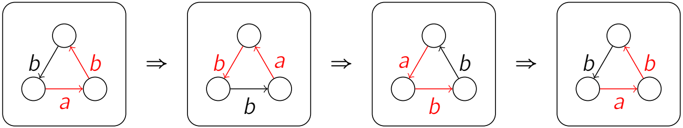
  "every execution (on any input) eventually halts."
- Undecidable in general

One-rule examples of non-termination and termination

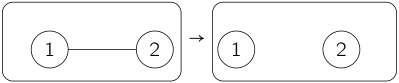
Rule α :



Looping:

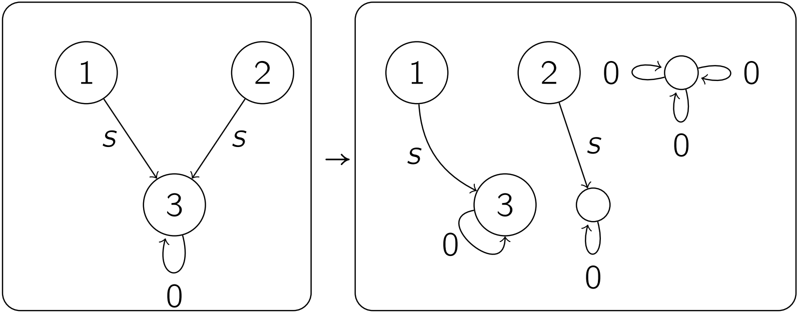


Rule β:



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

Motivating Rule



Question: does this rule terminate?

Structure of the presentation

Introduction

Preliminaries

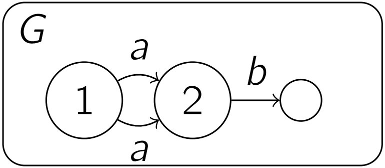
Extending Type Graph Method to Non-well-founded Semirings

Termination using Morphism Counting

LyonParallel—A Tool for Termination of Graph Rewriting

Conclusion and Future Work

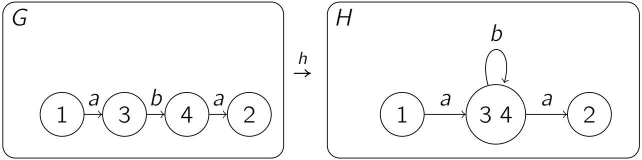
Graphs : finite, directed, edge-labeled multigraphs



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



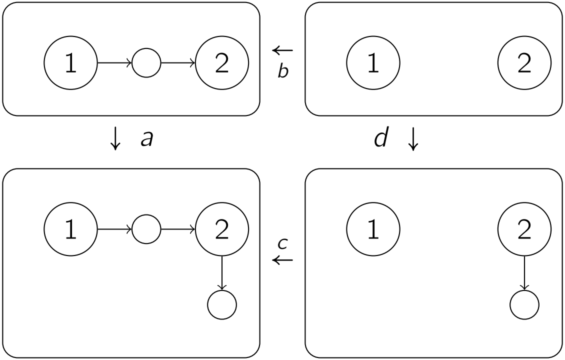
Remarks:

- 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

$$\begin{tikzcd}
L \arrow[d, "a"] & K \arrow[l, "b"] \arrow[d, "d"'] \\
G & C \arrow[l, "c"']
\end{tikzcd}$$

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,

$$\begin{tikzcd}
 & A \arrow[ld, "\alpha"'] \arrow[rd, "\beta"] & \\
B \arrow[rd, "\beta'"'] & \mathrm{PO} & C \arrow[ld, "\alpha'"] \\
 & D &
\end{tikzcd}$$

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.

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

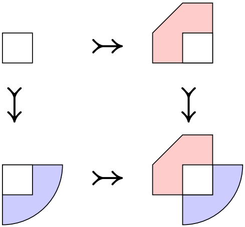
Pushouts: gluing graphs along a common part

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.

$$\begin{tikzcd}
& A \arrow[ld, "\alpha"'] \arrow[rd, "\beta"] & \\
B \arrow[rd, "\beta'"'] \arrow[rdd, bend right, "\gamma'"'] & \text{PO} & C \arrow[ld, "\alpha'"] \arrow[ldd, bend left, "\gamma"] \\
& D \arrow[d, dashed, "\delta"] & \\
& E &
\end{tikzcd}$$



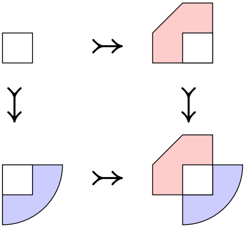
Pushouts: gluing graphs along a common part

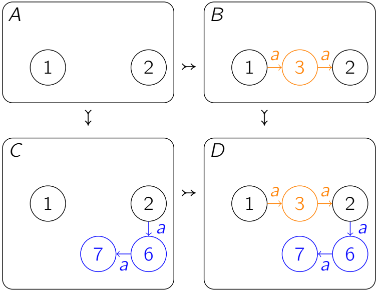
Definition

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.

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

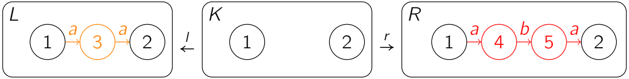




Graph rewriting with double-pushout approach (DPO)

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

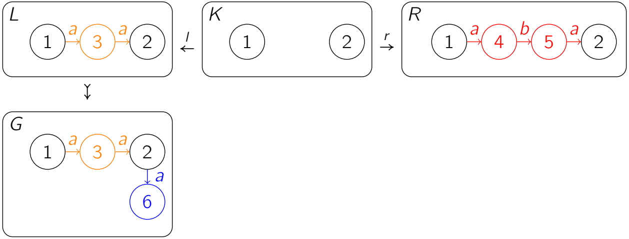
Rewriting rule with interface K



Graph rewriting with double-pushout approach (DPO)

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

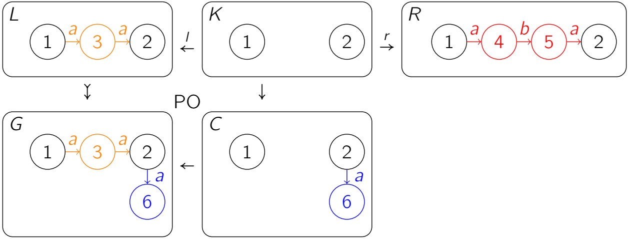
Rewriting rule with interface K



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

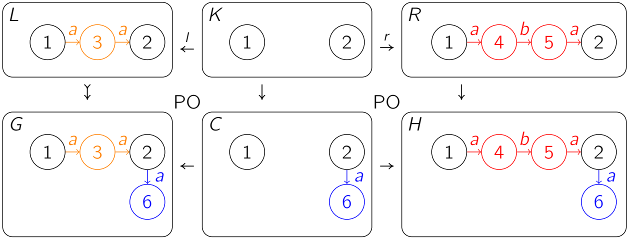


Graph rewriting with double-pushout approach (DPO)

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

Rewriting rule with interface K

rewriting step $G \Rightarrow H$



Introduction

Preliminaries

Extending Type Graph Method to Non-well-founded Semirings

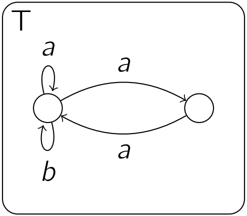
Termination using Morphism Counting

LyonParallel—A Tool for Termination of Graph Rewriting

Conclusion and Future Work

Weighted Type Graph

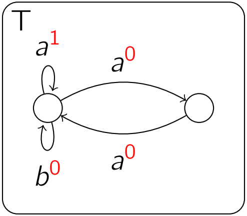
In the context of graph rewriting, a **weighted type graph** is a graph



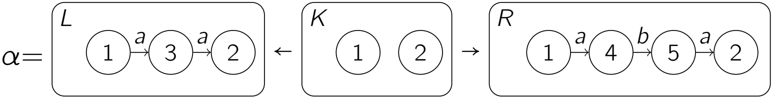
Weighted Type Graph

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

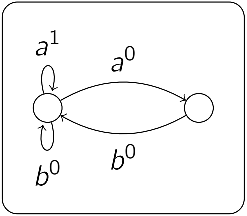
- with weights assigned to its edges.



Type graph method with weighted type graphs over natural numbers with an example



Weighted type graph T over natural numbers:



Intuition

- T : a weighted type graph
-

$$\begin{aligned} G &\Longrightarrow \mathcal{F}(G, T) \\ &\Longrightarrow \{weight(h) \mid h \in \mathcal{F}(G, T)\} \\ &\Longrightarrow \textbf{agregateur}(\{weight(h) \mid h \in \mathcal{F}(G, T)\}) \in \mathbb{N} \end{aligned}$$

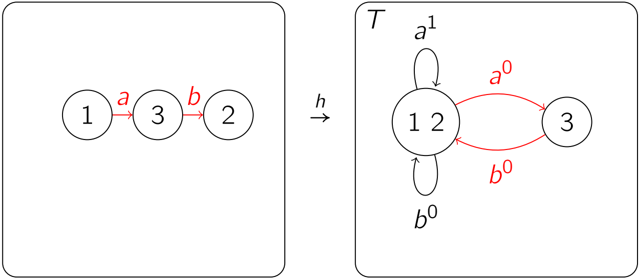
Questions:

- what is the weight of a morphism?
- which agregateur to use?
- How to approximate the weight of G and H in a rewriting step $G \Rightarrow H$?

Morphism weight

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

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

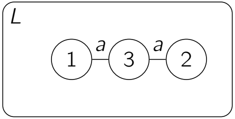


$\mathrm{weight}_T(h) = 0 + 0 = 0$

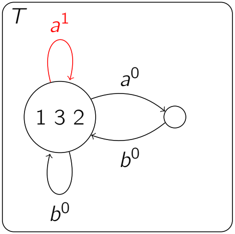
Graph weight

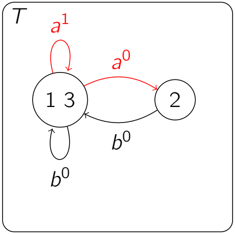
The weight of a graph L is

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



Two morphisms from L to T:





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

Extension of a morphism

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

A morphism $t_L : L \rightarrow T$ extends t_K if $t_K = t_L \circ l$.
For every morphism $t_K : K \rightarrow T$, 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 (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 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

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

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

Challenge:

- ~~there are $2^{k^2m} \cdot n^{k^2m}$ possible assignments of weights~~
- there are 2^{k^2m} 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

Introduction

Preliminaries

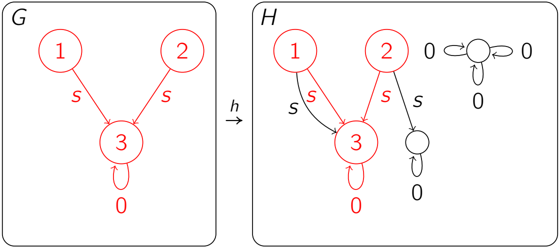
Extending Type Graph Method to Non-well-founded Semirings

Termination using Morphism Counting

LyonParallel—A Tool for Termination of Graph Rewriting

Conclusion and Future Work

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

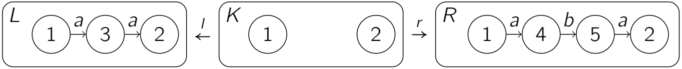


Remarks:

- G is a subgraph of H.

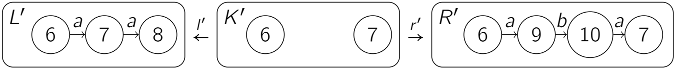
Graph rewriting rule

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

$\alpha =$ 

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

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

$\alpha' =$ 

Graph Rewriting

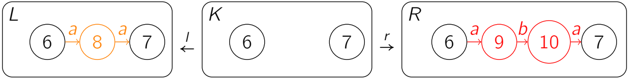
Rewriting steps $G \Rightarrow_\varphi H$ using rule φ are commutative diagrams with an equivalent rule $L' \overset{l'}{\leftarrow} K' \overset{r'}{\rightarrow} R'$ where all morphisms are inclusions:

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

A rewriting step with a running example

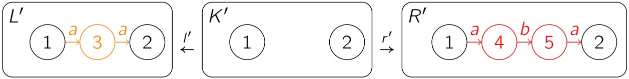
Rewriting rule:

$\varphi=$

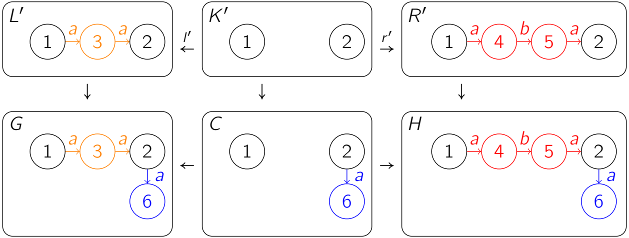


An equivalent rewriting rule:

$\varphi'=$

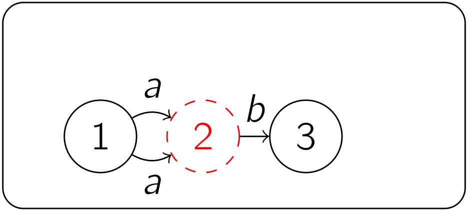


A rewriting step $G \Rightarrow_{\varphi} H$:



Pre-graphs

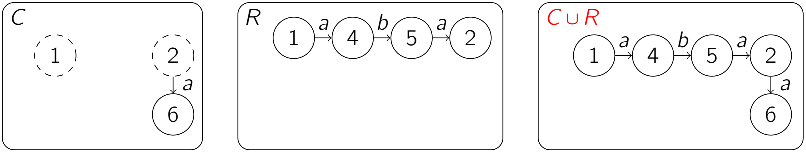
Pre-graphs are graphs with missing nodes and dangling edges.



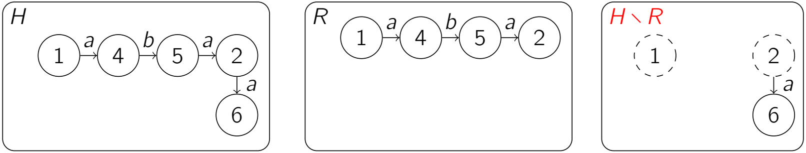
- 1 missing node in red,
- all edges are dangling,
- 2 existing nodes.

Pre-graph operations

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



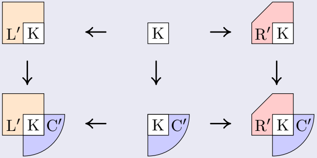
Relative complement of R in H where $R \subseteq H$, denoted $H \smallsetminus R$.



Analysis of rewriting steps

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

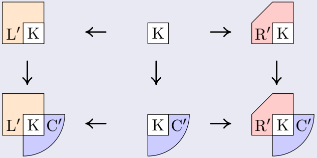
$$\begin{tikzcd}
L' \cup K \arrow[d] & K \arrow[l] \arrow[r] \arrow[d] & R' \cup K \arrow[d] \\
L' \cup K \cup C' & K \cup C' \arrow[l] \arrow[r] & R' \cup K \cup C'
\end{tikzcd}$$

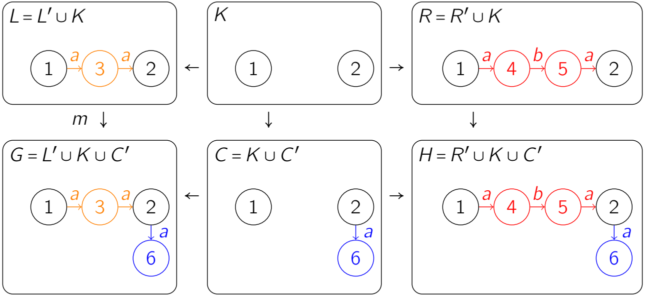


Analysis of rewriting steps

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

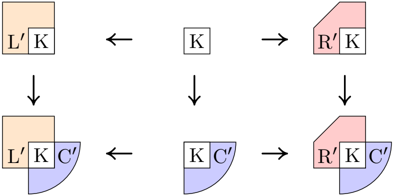
$$\begin{tikzcd}
L' \cup K \arrow[d] & K \arrow[l] \arrow[r] \arrow[d] & R' \cup K \arrow[d] \\
L' \cup K \cup C' & K \cup C' \arrow[l] \arrow[r] & R' \cup K \cup C'
\end{tikzcd}$$



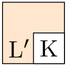


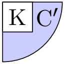
Implicit, Explicit and Shared Occurrences

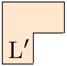
An *X*-occurrence in a graph G is an injective morphism $x : X \to G$.

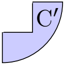


An X-occurrence is

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

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

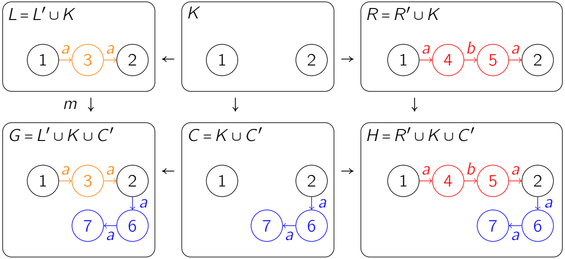
- implicit if **Im**(x) has elements in both 

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: $1 \xrightarrow{a} 3 \xrightarrow{a} 2$

Explicit X-occurrence in H: None.

Implicit X-occurrences in G: $3 \xrightarrow{a} 2 \xrightarrow{a} 6$

Implicit X-occurrence in H: $5 \xrightarrow{a} 2 \xrightarrow{a} 6$

Shared X-occurrence by G and H: $2 \xrightarrow{a} 6 \xrightarrow{a} 7$

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

A sufficient condition for termination

φ terminates if for all rewriting step:

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

the following holds:

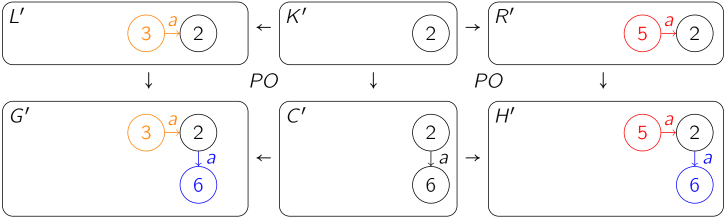
1. $|$explicit X-occurrences in $G| > |$explicit X-occurrences in $H|$;
2. $|$implicit X-occurrences in $G| \geq |$implicit X-occurrences in $H|$.

The first condition is straightforward 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 $C' \cup R'$. $2 \xrightarrow{a} 6$ is shared by G and H.

$5 \xrightarrow{a} 2$ is not in G but there is $3 \xrightarrow{a} 2$ in G and $C' \cup L'$ is an implicit occurrence in G.

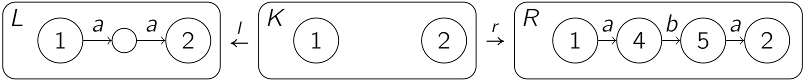
X-non-increasing rule

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

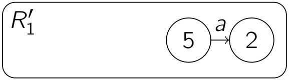
Lemma (More X-occurrences before rewriting)

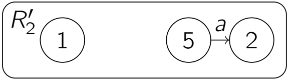
For all $G \Rightarrow_\varphi H$, there are more implicit X-occurrences in G than in H, if
"subgraphs of R that can form an implicit X-occurrence in some rewriting step can be mapped to distinct subgraphs in L while preserving the interface elements".

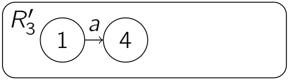
Example

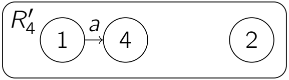


The set $D(R, X)$ of subgraphs of R which can form an implicit X-occurrence in some rewriting step:

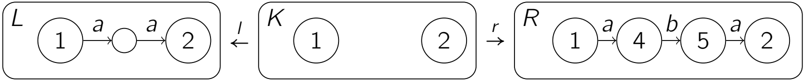




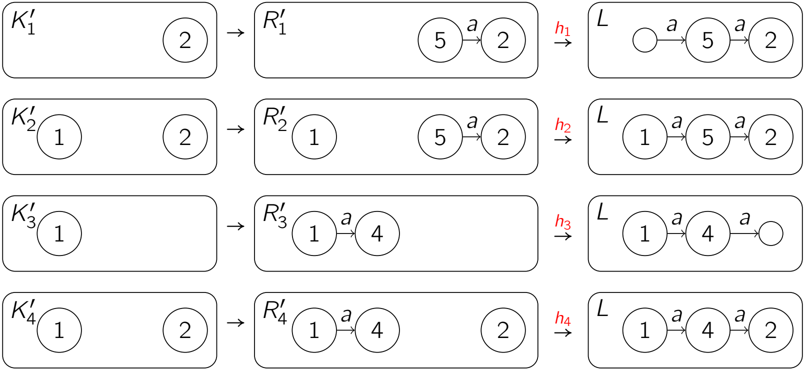




Example



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

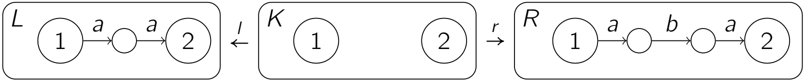


Main Results

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.

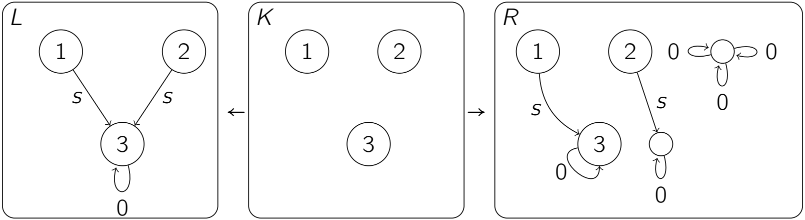
Terminating of Running Example



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

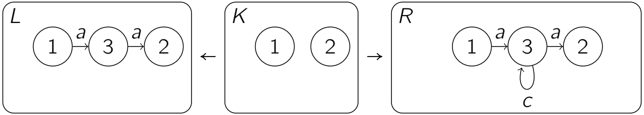
$$1 > 0$$

Termination of Motivating Rule

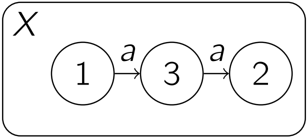


Termination by counting morphisms from $\bullet \xrightarrow{s} \bullet \xleftarrow{s} \bullet$

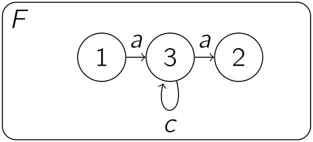
An extension for counting morphisms with a forbidden pattern



Termination by counting morphisms from



whose images are not subgraphs of occurrences of



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.

[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 Transformation". In: *Graph Transformation, Specifications, and Nets - In Memory of Hartmut*