

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

Qi QIU

Supervisor: Xavier URBAIN  
Université Claude Bernard Lyon 1, France  
Université de Lorraine, France

## Introduction

Graphs and Graph Morphisms

Graph Rewriting

Termination of graph rewriting systems

## Termination using Morphism Counting

Implicit, Explicit and Shared Occurrences

Challenges in establishing a sufficient condition for termination

A solution

## Morphism Counting with Antipattern

### Extending Type Graph Method to Non-well-founded Semirings

Type graph method

Searching for suitable weighted type graphs in practice

Accelerating the Search

## LyonParallel—A Tool for Termination of Graph Rewriting

# Rule-based Graph Rewriting and Distributed algorithms

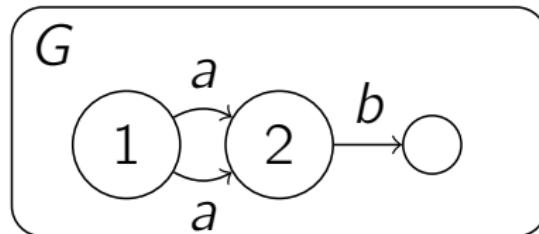
- ▶ Correctness of distributed algorithms is hard to ensure.
- ▶ Rule-based graph rewriting for modeling distributed algorithms:
  - ▶ Configurations : graphs
  - ▶ Operations : graph rewriting rules
- ▶ Automated reasoning for verification

# Graphs

Graphs are **finite** and **directed** with:

- ▶ Labeled edges,
- ▶ Finite labels,
- ▶ Distinct edges with the same source, target, and label

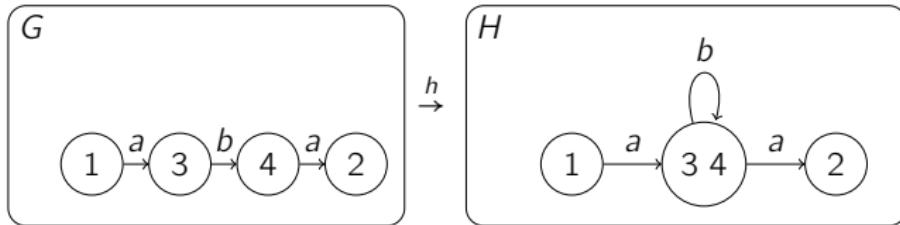
Example:



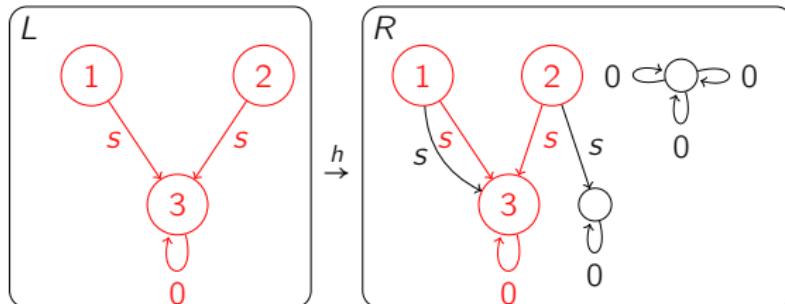
- ▶ Enclosed in a box
- ▶ Graph name  $G$  in the top left corner
- ▶ Numbers in nodes are identifiers, omitted when not relevant.

# Graph Morphisms

- ▶ **Graph morphisms** are structure-preserving mappings.
- ▶ Example of a morphism  $h: G \rightarrow H$ :



- ▶ Isomorphisms: bijective morphisms.
- ▶ Inclusions : morphisms with  $f(x) = x$  for all  $x$  from the domain.
- ▶ Example of an inclusion:



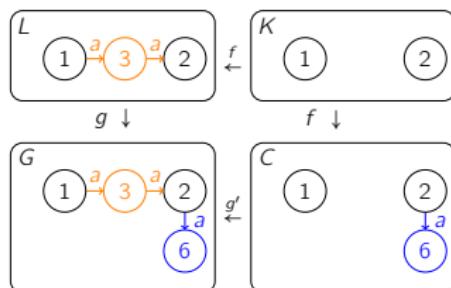
# Commutative Diagram

A diagram:

$$\begin{array}{ccc} L & \xleftarrow{f} & K \\ \downarrow g & & \downarrow f' \\ G & \xleftarrow{g'} & C \end{array}$$

is **commutes** if  $g \circ f = g' \circ f'$ .

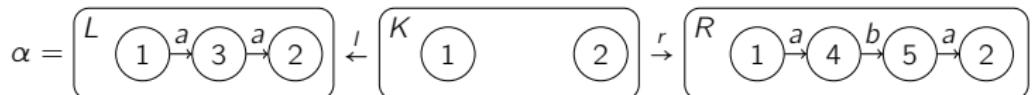
Example:



# Graph rewriting rule

Rules  $\varphi = (L \xleftarrow{I} K \xrightarrow{r} R)$  consist of inclusions  $I$  and  $r$ .

Example:



Rule  $\varphi' = (L' \xleftarrow{I'} K' \xrightarrow{r'} R')$  and  $\varphi$  are equivalent if the following commutative diagram can be constructed:

$$\begin{array}{ccccc} L' & \xleftarrow{I'} & K' & \xrightarrow{r'} & R' \\ \downarrow & & \downarrow & & \downarrow \\ L & \xleftarrow{I} & K & \xrightarrow{r} & R \end{array}$$

An equivalent rule of  $\alpha$ :



# Graph Rewriting

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

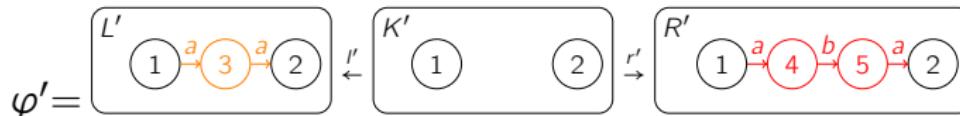
$$\begin{array}{ccccc} L' & \xleftarrow{l'} & K' & \xrightarrow{r'} & R' \\ \downarrow & & \downarrow & & \downarrow \\ G & \xleftarrow{\quad} & C & \xrightarrow{\quad} & H \end{array}$$

# A rewriting step with a running example

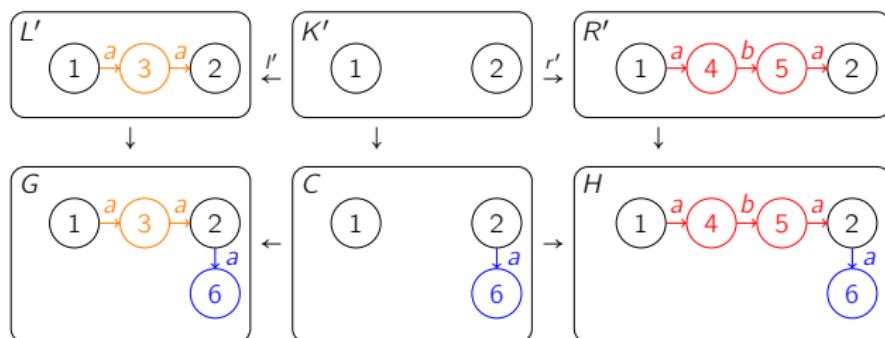
Rewriting rule:



An equivalent rewriting rule:



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



It replaces chain  $\bullet \xrightarrow{a} \bullet \xrightarrow{a} \bullet$  with  $\bullet \xrightarrow{a} \bullet \xrightarrow{b} \bullet \xrightarrow{a} \bullet$ .

# Termination of graph rewriting systems

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

$$G_0 \xrightarrow{\mathcal{R}} G_1 \xrightarrow{\mathcal{R}} G_2 \xrightarrow{\mathcal{R}} \dots$$

when using the non-deterministic strategy

“apply rules as long as possible”

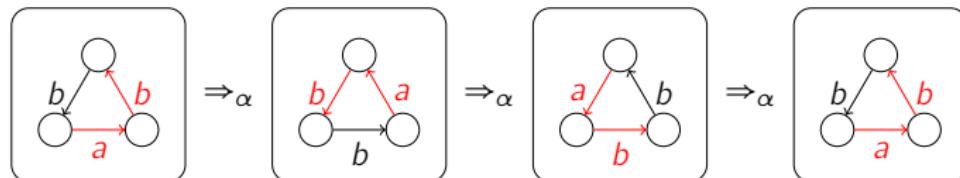
- ▶ Aligns with the standard notion of program termination:  
“every execution (on any input) eventually halts.”
- ▶ Undecidable in general

## One-rule examples of non-termination and termination

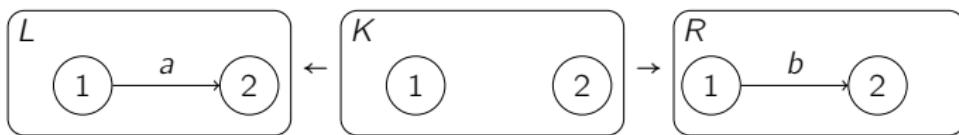
Rule  $\alpha$  :



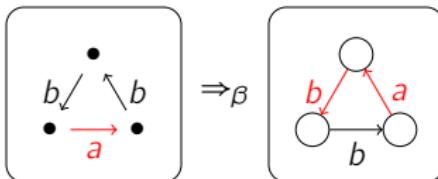
Looping:



Rule  $b$ :



Termination by the number of edges labeled by “a”:



## Introduction

### Termination using Morphism Counting

Implicit, Explicit and Shared Occurrences

Challenges in establishing a sufficient condition for termination

A solution

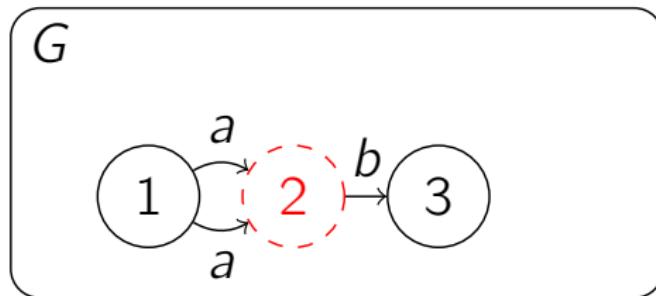
### Morphism Counting with Antipattern

### Extending Type Graph Method to Non-well-founded Semirings

### LyonParallel—A Tool for Termination of Graph Rewriting

## Pre-graphs

Pre-graphs are **graphs with missing nodes** and dangling edges.  
Example:

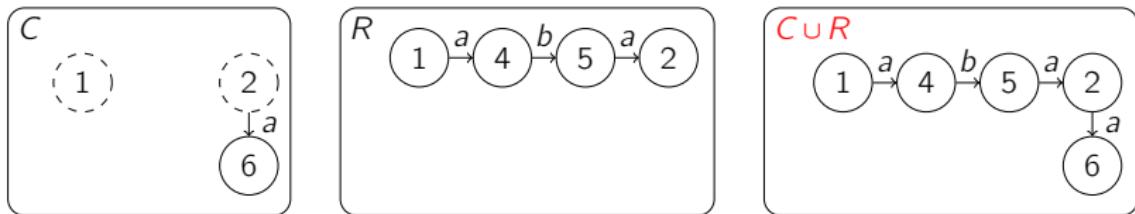


$G$  has

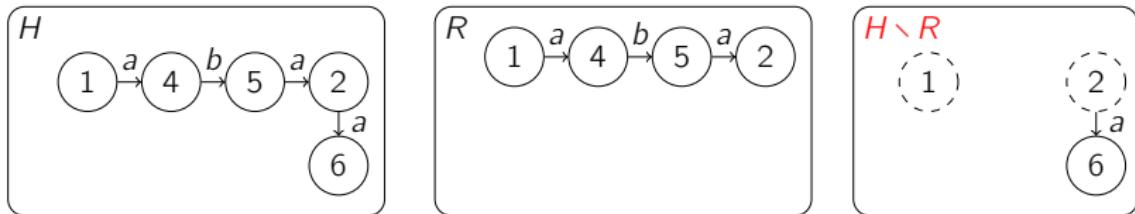
- ▶ 2 existing nodes,
- ▶ 1 missing node in red,
- ▶ 3 dangling edges.

## Pre-graph operations

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

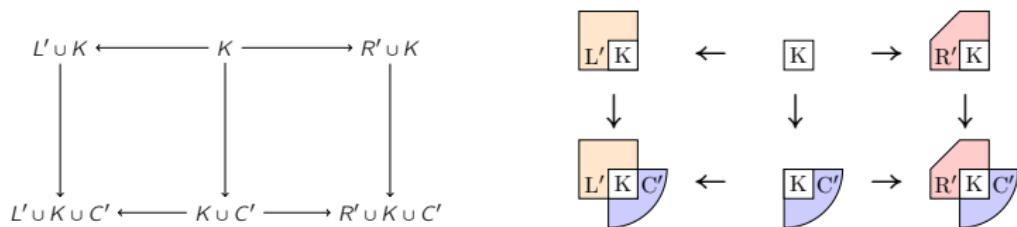


Relative complement of  $R$  in  $H$  where  $R \subseteq H$ , denoted  $H \setminus R$ :

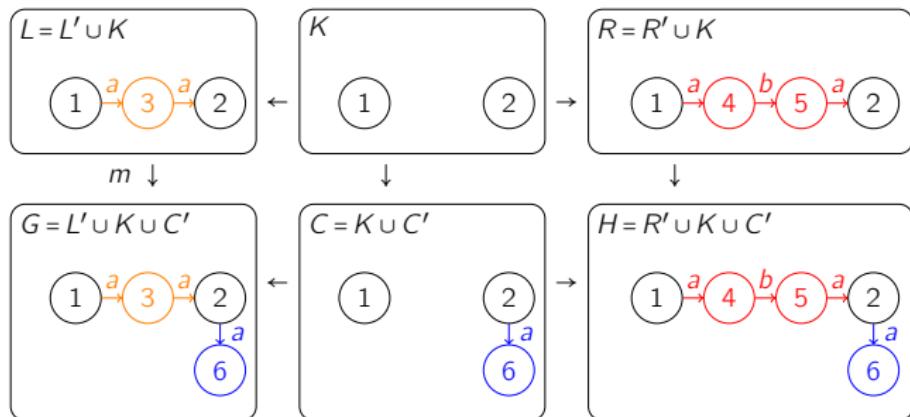


# Analysis of rewriting steps

In a rewriting step, since all arrows are inclusions by definition, graphs can be decomposed as unions of pre-graphs:

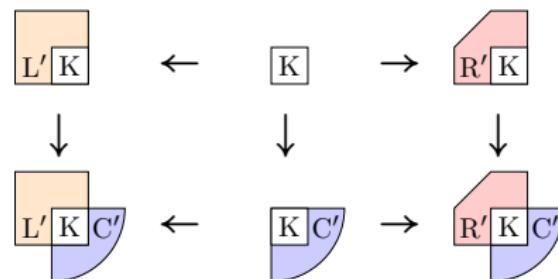


Example with running rule:



# Implicit, Explicit and Shared Occurrences

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



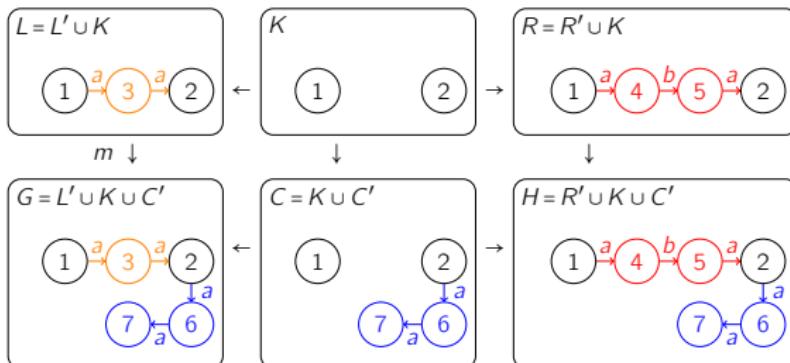
An  $X$ -occurrence  $x : X \rightarrow G$  is

- ▶ **explicit** if  $\text{Im}(x)$  is included in
- ▶ **shared** if  $\text{Im}(x)$  is included in
- ▶ **implicit** if  $\text{Im}(x)$  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  $G$ :  $\bullet \xrightarrow{a} \bullet \xrightarrow{a} \bullet$

Explicit  $X$ -occurrence in  $H$ : None.

Implicit  $X$ -occurrences in  $G$ :  $\bullet \xrightarrow{a} \bullet \xrightarrow{a} \bullet$

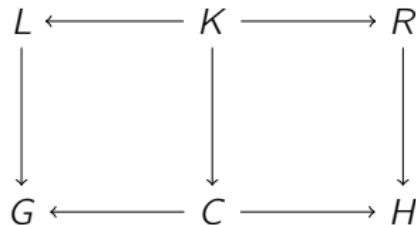
Implicit  $X$ -occurrence in  $H$ :  $\bullet \xrightarrow{a} \bullet \xrightarrow{a} \bullet$

Shared  $X$ -occurrence by  $G$  and  $H$ :  $\bullet \xrightarrow{a} \bullet \xrightarrow{a} \bullet$

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

## A sufficient condition for termination

$\varphi$  terminates if for all rewriting step:



the following holds:

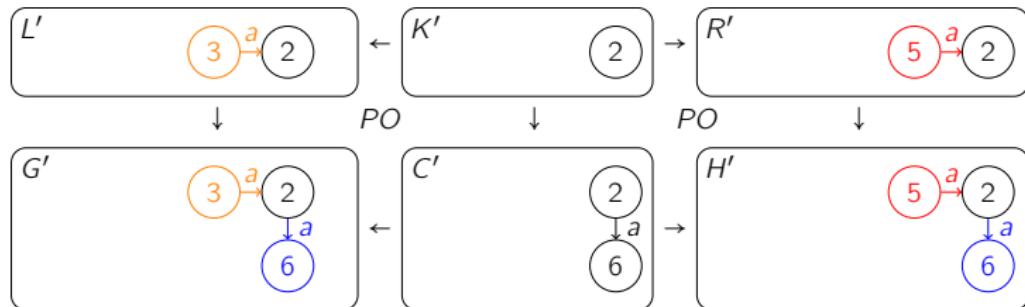
1.  $|\text{explicit } X\text{-occurrences in } G| > |\text{explicit } X\text{-occurrences in } H|$ ;
2.  $|\text{implicit } X\text{-occurrences in } G| \geq |\text{implicit } X\text{-occurrences in } H|$ .

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  $|X$ -occurrences in  $R$  are computable.

Challenge: Establishing the **second condition**.

# Analysis of Implicit Occurrences in $G$ and $H$



The occurrence is  $C' \cup R'$ .  $\circlearrowleft 2 \xrightarrow{a} 6 \circlearrowright$  is shared by  $G$  and  $H$ .

$\circlearrowleft 5 \xrightarrow{a} 2 \circlearrowright$  is not in  $G$  but there is  $\circlearrowleft 3 \xrightarrow{a} 2 \circlearrowright$  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*

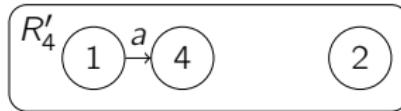
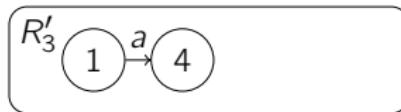
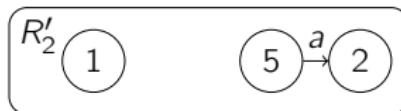
*"subgraphs of  $R$  that can form an implicit  $X$ -occurrence in some rewriting step can be mapped to subgraphs in  $L$  while preserving the interface elements".*

## Example

Rewriting rule:



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

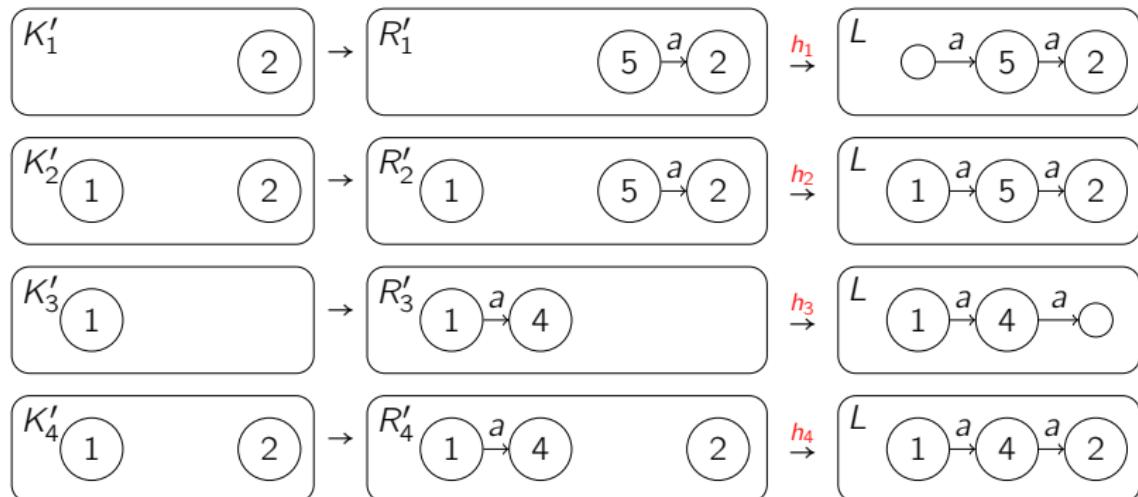


## Example

Rewriting rule:



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  $\varphi$  be a  $X$ -non-increasing rule.  $\varphi$  is terminating if there are strictly more explicit  $X$ -occurrences in  $L$  than in  $R$ .*

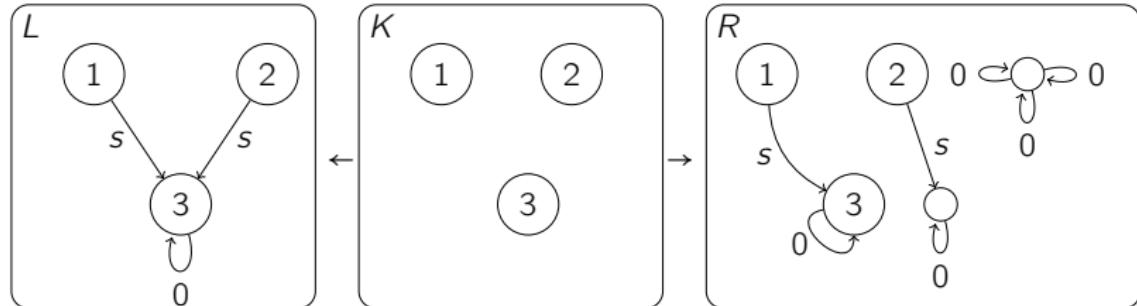
## Terminating of Running Example



- ▶ 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



- ▶  $X : \bullet \xrightarrow{s} \bullet \xleftarrow{s} \bullet$
- ▶  $D(R, X)$  consists of  $R_1: \textcircled{1} \xrightarrow{s} \textcircled{3}$  and  $R_2: \textcircled{1} \xrightarrow{s} \textcircled{3} \quad \textcircled{2}$
- ▶  $X$ -non-increasing rule
- ▶ Strictly more explicit  $X$ -occurrences in  $L$  than in  $R$  :  $1 > 0$
- ▶ Terminating
- ▶ Its termination cannot be shown by existing techniques.

## Introduction

Termination using Morphism Counting

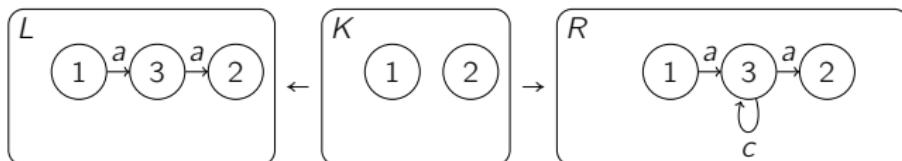
Morphism Counting with Antipattern

Extending Type Graph Method to Non-well-founded Semirings

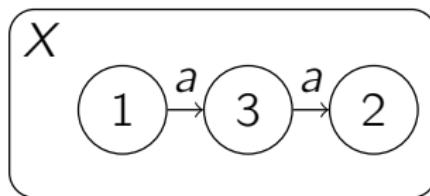
LyonParallel—A Tool for Termination of Graph Rewriting

## An extension for counting subgraphs with antipattern

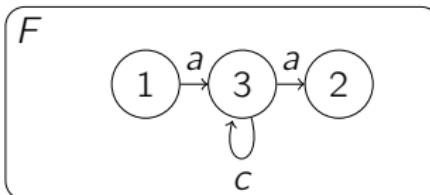
The morphism counting method can be extended for termination of the following rule:



by counting the number of injective morphisms from



whose images are not included in the following graph  
(antipattern):



Introduction

Termination using Morphism Counting

Morphism Counting with Antipattern

## Extending Type Graph Method to Non-well-founded Semirings

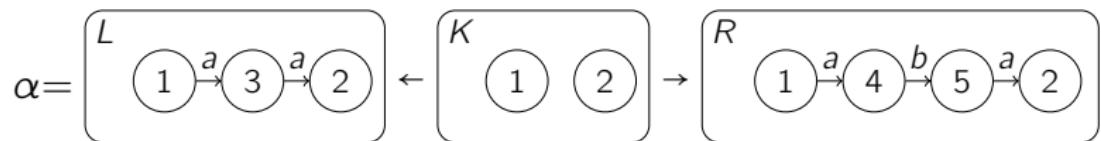
Type graph method

Searching for suitable weighted type graphs in practice

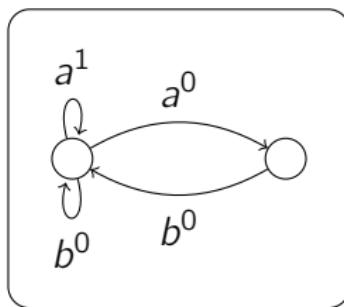
Accelerating the Search

LyonParallel—A Tool for Termination of Graph Rewriting

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

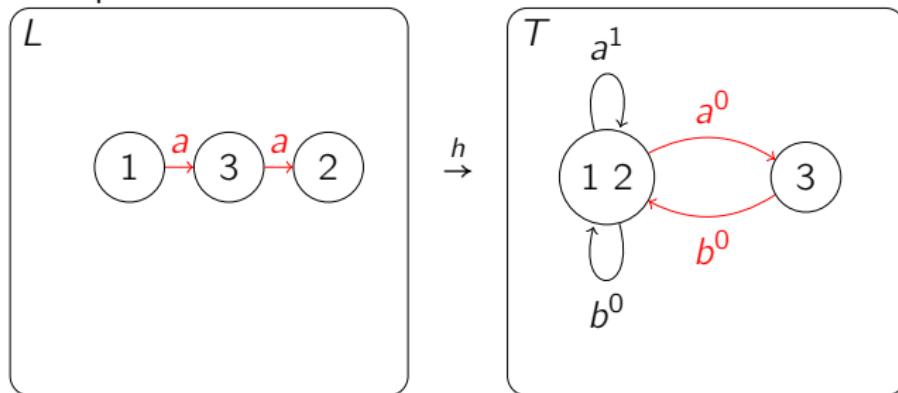


Weighted type graph  $T$  over natural numbers:



Weight of a morphism  $h: L \rightarrow T$ : the sum of weights of all edges in  $\text{Im}(h)$

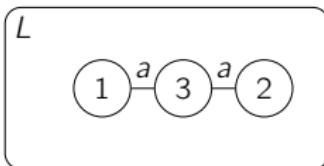
Example:



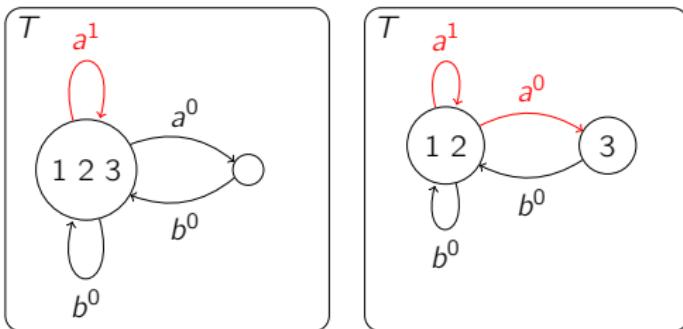
$$w_T(h) = 0 + 0 = 0$$

The weight of a graph  $G$  is defined as the minimum weight  $w_T(h)$  of all morphisms  $h: G \rightarrow T$

Example: The following graph

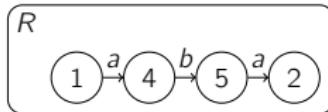


has two morphisms to  $T$ :

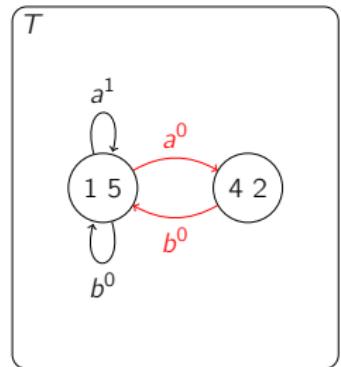
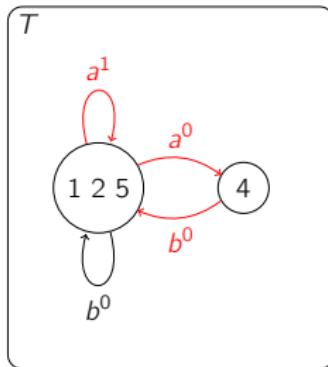
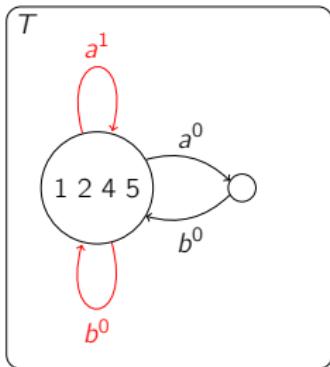


$$w_T(L) = \min\{1 + 1, 1 + 0\} = 1$$

The following graph

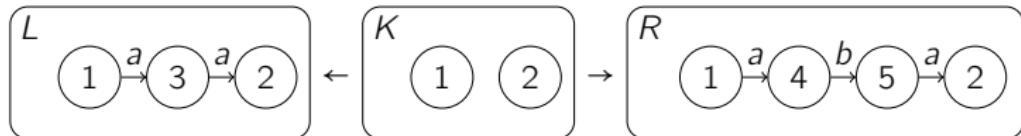


has three morphisms to  $T$ :



Its weight is  $\min\{1 + 0 + 1, 0 + 0 + 1, 0 + 0 + 0\} = 0$ .

## Termination Condition



For every morphism  $t_K : K \rightarrow T$ , we define

- ▶  $S(t_k, L)$  : the min of the weights of the morphisms  $t_L$  that extend  $t_K$
- ▶  $S(t_k, R)$  : the min of the weights of the morphisms  $t_R$  that extend  $t_K$

Every rewriting step strictly decreases the weight if

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

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

,

by Bruggink et al. 2014 [BKZ14].

Problem: existence of such a weighted type graph is undecidable in general.

# 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:

- ▶  $k^2|\Sigma|$  binary variables
- ▶  $k^2|\Sigma|$  integer variables

Challenge:

- ▶  $2^{k^2|\Sigma|} \cdot n^{k^2|\Sigma|}$  possible assignments of weights
- ▶ maximum edge weight impossible to determine in advance

## Solution: using real numbers instead of natural numbers.

Every rewriting step strictly decreases the weight if

- ▶ for all  $t_K : K \rightarrow T$ , 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  $t_K : K \rightarrow T$ , 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  $\{0, 1, \dots, n\}$~~

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

# LyonParallel

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

- [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](https://doi.org/10.1007/978-3-662-44602-7_15).