

# From Compactifying Lambda-Letrec Terms to Recognizing Regular-Expression Processes

Clemens Grabmayer

Department of Computer Science

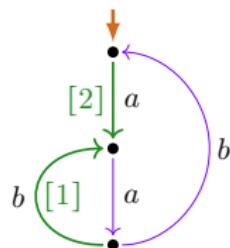
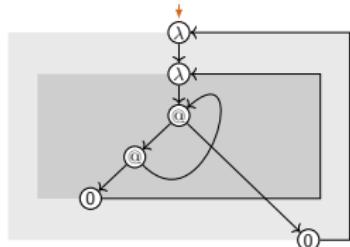


L'Aquila, Italy

DCM-2013

Rome

July 2, 2023



# Overview

## 1. Compactifying $\lambda$ -terms with letrec

- ▶ higher-order  $\lambda$ -term graphs

## 2. Recognizing regular-expression processes

- ▶ **LEE-witnesses:** graph labelings based on a loop-condition LEE

# Overview

## 1. Compactifying $\lambda$ -terms with letrec

- ▶ from terms in the  $\lambda$ -calculus with letrec to:
  - ▶ higher-order  $\lambda$ -term graphs
  - ▶ first-order  $\lambda$ -term graphs
  - ▶  $\lambda$ -NFAs, and  $\lambda$ -DFAs
- ▶ minimization / readback / efficiency / Haskell implementation

## 2. Recognizing regular-expression processes

- ▶ **LEE-witnesses:** graph labelings based on a loop-condition LEE

# Overview

## 1. Compactifying $\lambda$ -terms with letrec

- ▶ from terms in the  $\lambda$ -calculus with letrec to:
  - ▶ higher-order  $\lambda$ -term graphs
  - ▶ first-order  $\lambda$ -term graphs
  - ▶  $\lambda$ -NFAs, and  $\lambda$ -DFAs
- ▶ minimization / readback / efficiency / Haskell implementation

## 2. Recognizing regular-expression processes

- ▶ Milner's questions, known results
- ▶ structure-constrained process graphs:
  - ▶ LEE-witnesses: graph labelings based on a loop-condition LEE
  - ▶ preservation under bisimulation collapse
- ▶ readback: from graph labelings to regular expressions

# Overview

- ▶ Comparison desiderata
- 1. Compactifying  $\lambda$ -terms with letrec
  - ▶ from terms in the  $\lambda$ -calculus with letrec to:
    - ▶ higher-order  $\lambda$ -term graphs
    - ▶ first-order  $\lambda$ -term graphs
    - ▶  $\lambda$ -NFAs, and  $\lambda$ -DFAs
  - ▶ minimization / readback / efficiency / Haskell implementation
- 2. Recognizing regular-expression processes
  - ▶ Milner's questions, known results
  - ▶ structure-constrained process graphs:
    - ▶ LEE-witnesses: graph labelings based on a loop-condition LEE
    - ▶ preservation under bisimulation collapse
  - ▶ readback: from graph labelings to regular expressions
- ▶ Comparison results

# Comparison original desiderata

$\lambda$ -calculus with letrec under the unfolding semantics

*Well-known:* graph representations implemented by compilers

- ▶ but were **not intended** for use under  $\leftrightarrow$

*Not available:* term graph interpretation that is studied under  $\leftrightarrow$

Regular expressions under process semantics (bisimilarity  $\leftrightarrow$ )

# Comparison original desiderata

$\lambda$ -calculus with letrec under the unfolding semantics

*Well-known:* graph representations implemented by compilers

- ▶ but were **not intended** for use under  $\leftrightarrow$

*Not available:* term graph interpretation that is studied under  $\leftrightarrow$

*Desired:* term graph interpretation that:

- ▶ has natural correspondence with terms in  $\lambda_{\text{letrec}}$
- ▶ supports compactification under  $\leftrightarrow$
- ▶ permits efficient translation and readback

Regular expressions under process semantics (bisimilarity  $\leftrightarrow$ )

# Comparison original desiderata

$\lambda$ -calculus with letrec under the unfolding semantics

*Well-known:* graph representations implemented by compilers

- ▶ but were not intended for use under  $\leftrightarrow$

*Not available:* term graph interpretation that is studied under  $\leftrightarrow$

*Desired:* term graph interpretation that:

- ▶ has natural correspondence with terms in  $\lambda_{\text{letrec}}$
- ▶ supports compactification under  $\leftrightarrow$
- ▶ permits efficient translation and readback

Regular expressions under process semantics (bisimilarity  $\leftrightarrow$ )

*Given:* process graph interpretation  $P(\cdot)$ , studied under  $\leftrightarrow$

- ▶ not closed under  $\supseteq$ , and  $\leftrightarrow$ , modulo  $\leftrightarrow$  incomplete

# Comparison original desiderata

$\lambda$ -calculus with letrec under the unfolding semantics

*Well-known:* graph representations implemented by compilers

- ▶ but were not intended for use under  $\leftrightarrow$

*Not available:* term graph interpretation that is studied under  $\leftrightarrow$

*Desired:* term graph interpretation that:

- ▶ has natural correspondence with terms in  $\lambda_{\text{letrec}}$
- ▶ supports compactification under  $\leftrightarrow$
- ▶ permits efficient translation and readback

Regular expressions under process semantics (bisimilarity  $\leftrightarrow$ )

*Given:* process graph interpretation  $P(\cdot)$ , studied under  $\leftrightarrow$

- ▶ not closed under  $\Rightarrow$ , and  $\leftrightarrow$ , modulo  $\leftrightarrow$  incomplete

*Desired:* reason with graphs that are  $P(\cdot)$ -expressible modulo  $\leftrightarrow$   
 (at least with 'sufficiently many')

understand incompleteness by a structural graph property

# structure constraints (L'Aquila)

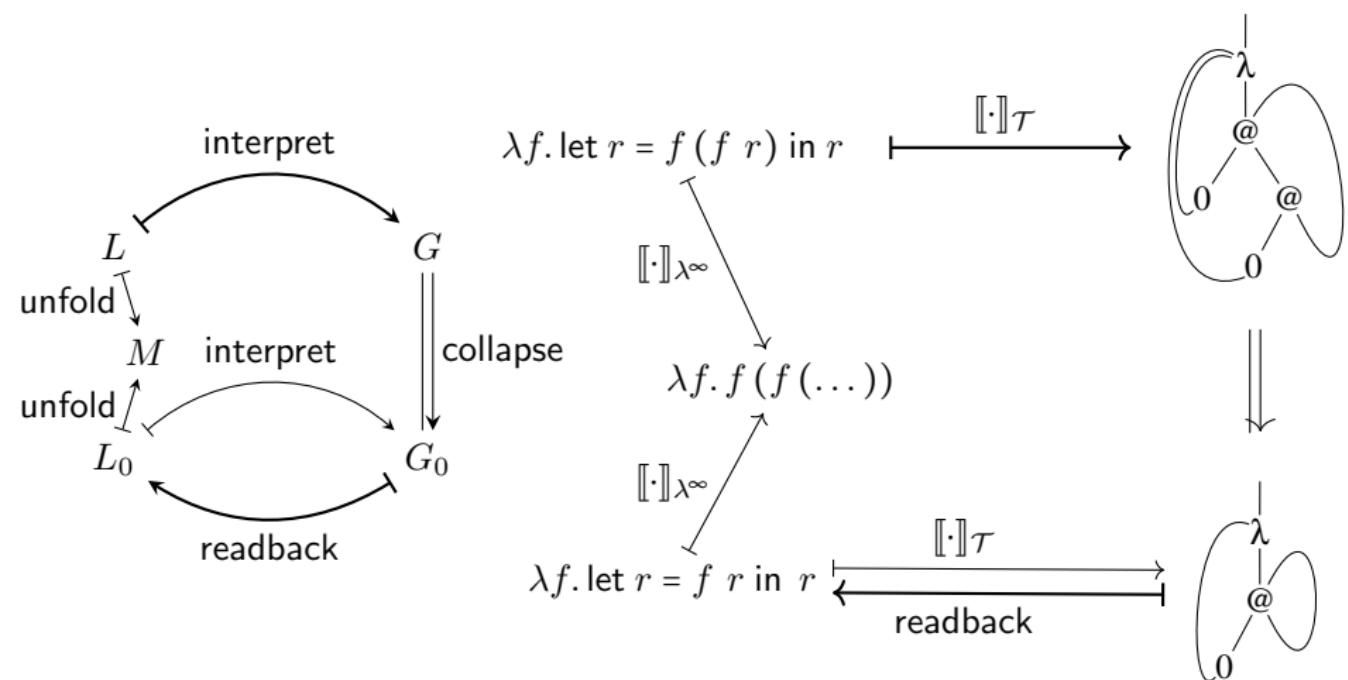


# Maximal sharing of functional programs

(joint work with Jan Rochel)



# Maximal sharing: example (fix)



# Maximal sharing: the method

$$L \xrightarrow{[\cdot]_{\mathcal{H}}} \mathcal{G}$$

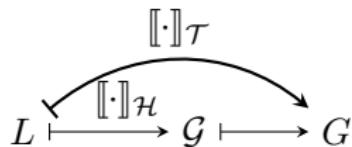
1. term graph interpretation  $[\cdot]$ .  
of  $\lambda_{\text{letrec}}$ -term  $L$  as:
  - a. higher-order term graph  
 $\mathcal{G} = [L]_{\mathcal{H}}$

# Maximal sharing: the method

$$L \xrightarrow{[\cdot]_{\mathcal{H}}} \mathcal{G} \longmapsto G$$

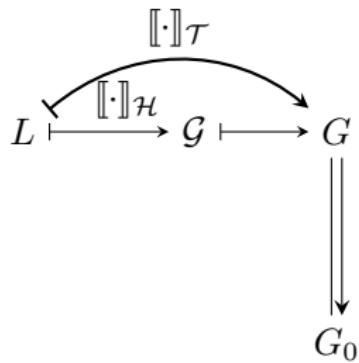
1. term graph interpretation  $[\cdot]$ .  
of  $\lambda_{\text{letrec}}$ -term  $L$  as:
  - a. higher-order term graph  
 $\mathcal{G} = [\cdot]_{\mathcal{H}}$
  - b. first-order term graph  $G = [\cdot]_{\tau}$

# Maximal sharing: the method



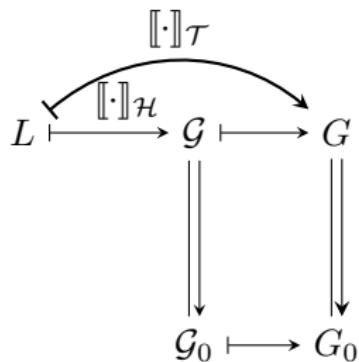
1. term graph interpretation  $\llbracket \cdot \rrbracket$ .  
of  $\lambda_{\text{letrec}}$ -term  $L$  as:
  - a. higher-order term graph  
 $\mathcal{G} = \llbracket L \rrbracket_{\mathcal{H}}$
  - b. first-order term graph  $G = \llbracket L \rrbracket_{\tau}$

# Maximal sharing: the method



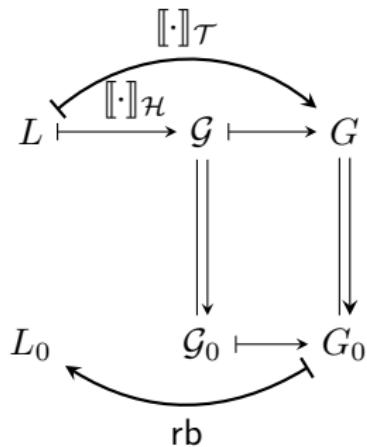
1. term graph interpretation  $\llbracket \cdot \rrbracket$ .  
of  $\lambda_{\text{letrec}}$ -term  $L$  as:
  - a. higher-order term graph  
 $\mathcal{G} = \llbracket L \rrbracket_{\mathcal{H}}$
  - b. first-order term graph  $G = \llbracket L \rrbracket_{\tau}$
2. bisimulation collapse  $\downarrow$   
of f-o term graph  $G$  into  $G_0$

# Maximal sharing: the method



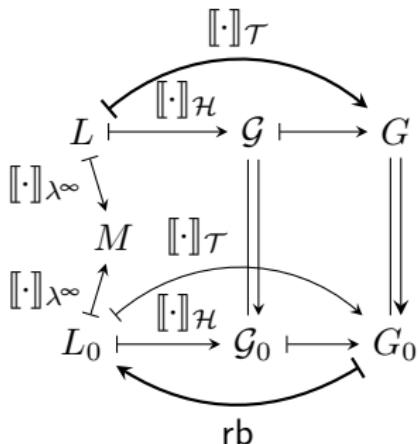
1. term graph interpretation  $\llbracket \cdot \rrbracket$ .  
of  $\lambda_{\text{letrec}}$ -term  $L$  as:
  - a. higher-order term graph  
 $\mathcal{G} = \llbracket L \rrbracket_H$
  - b. first-order term graph  $G = \llbracket L \rrbracket_T$
2. bisimulation collapse  $\downarrow$   
of f-o term graph  $G$  into  $G_0$

# Maximal sharing: the method



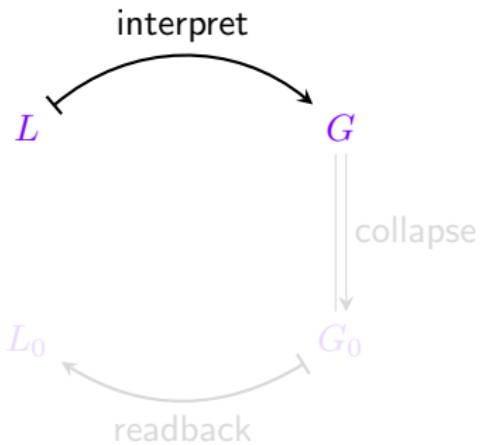
1. term graph interpretation  $[\cdot]$ .  
of  $\lambda_{\text{letrec}}$ -term  $L$  as:
  - a. higher-order term graph  $G = [[L]]_H$
  - b. first-order term graph  $G = [[L]]_T$
2. bisimulation collapse  $\Downarrow$   
of f-o term graph  $G$  into  $G_0$
3. readback rb  
of f-o term graph  $G_0$   
yielding program  $L_0 = \text{rb}(G_0)$ .

# Maximal sharing: the method



1. term graph interpretation  $\llbracket \cdot \rrbracket_{\tau}$  of  $\lambda_{\text{letrec}}$ -term  $L$  as:
  - a. higher-order term graph  $\mathcal{G} = \llbracket L \rrbracket_{\mathcal{H}}$
  - b. first-order term graph  $G = \llbracket L \rrbracket_{\tau}$
2. bisimulation collapse  $\Downarrow$  of f-o term graph  $G$  into  $G_0$
3. readback rb of f-o term graph  $G_0$  yielding program  $L_0 = \text{rb}(G_0)$ .

# Interpretation



# Running example

instead of:

$$\lambda f. \text{let } r = f(f r) \text{ in } r \xrightarrow{\text{max-sharing}} \lambda f. \text{let } r = f r \text{ in } r$$

we use:

$$\lambda x. \lambda f. \text{let } r = f(f r x) x \text{ in } r \xrightarrow{\text{max-sharing}} \lambda x. \lambda f. \text{let } r = f r x \text{ in } r$$

*L*

$\xrightarrow{\text{max-sharing}}$

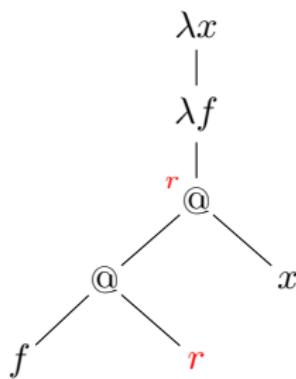
*L*<sub>0</sub>

# Graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$

# Graph interpretation (example 1)

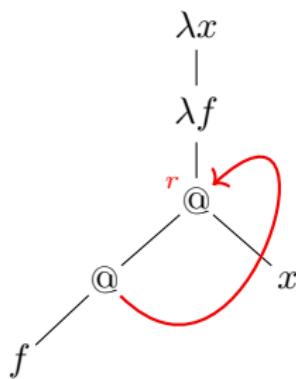
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree

# Graph interpretation (example 1)

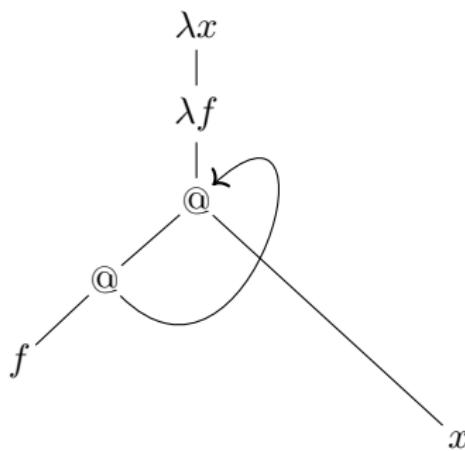
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree (+ recursive backlink)

# Graph interpretation (example 1)

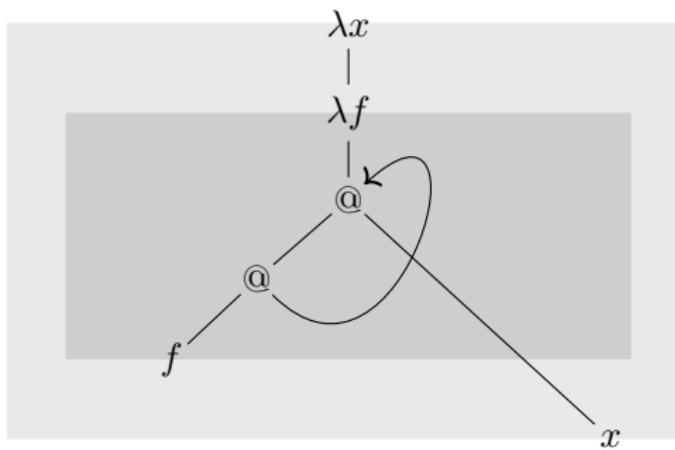
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree (+ recursive backlink)

# Graph interpretation (example 1)

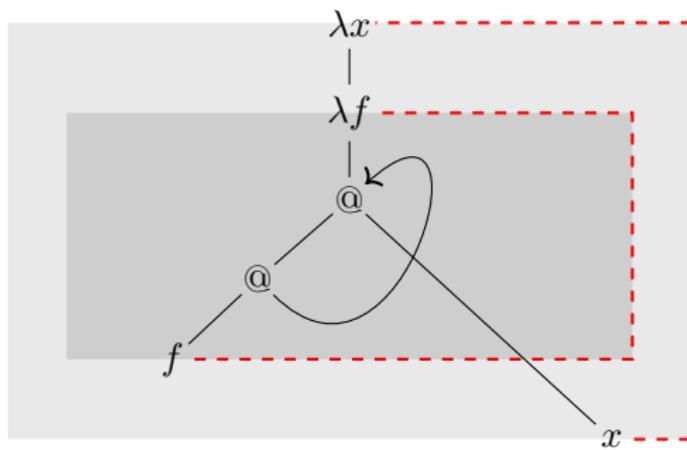
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

# Graph interpretation (example 1)

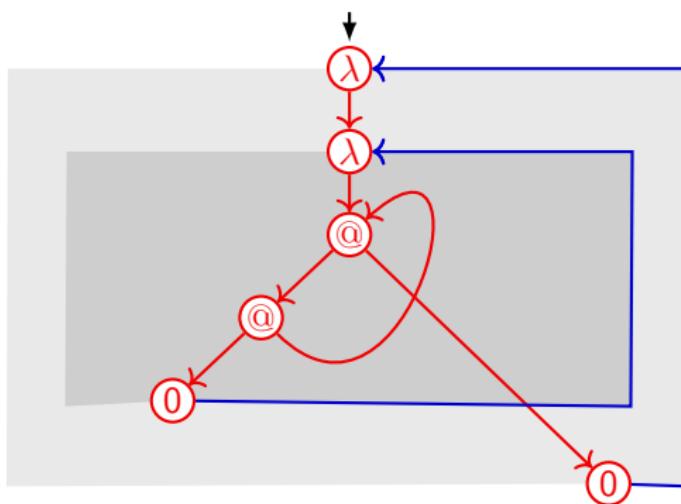
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



syntax tree (+ recursive backlink, + scopes, + binding links)

# Graph interpretation (example 1)

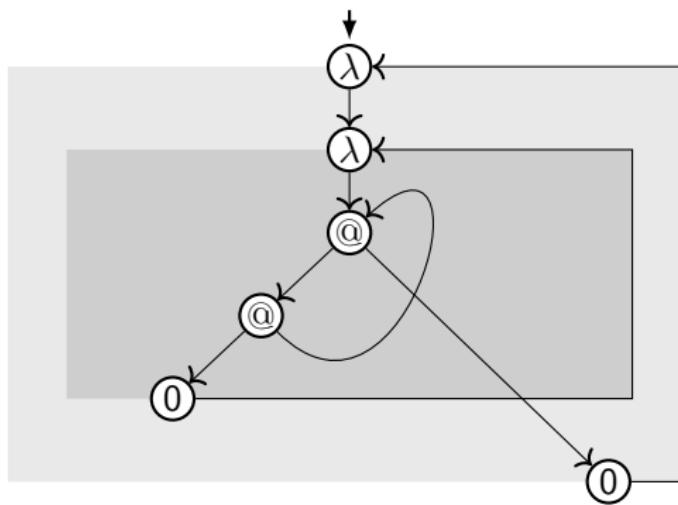
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

# Graph interpretation (example 1)

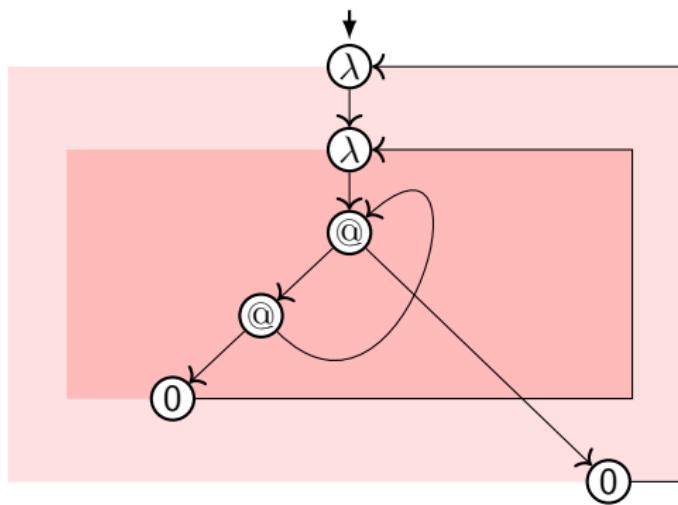
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

# Graph interpretation (example 1)

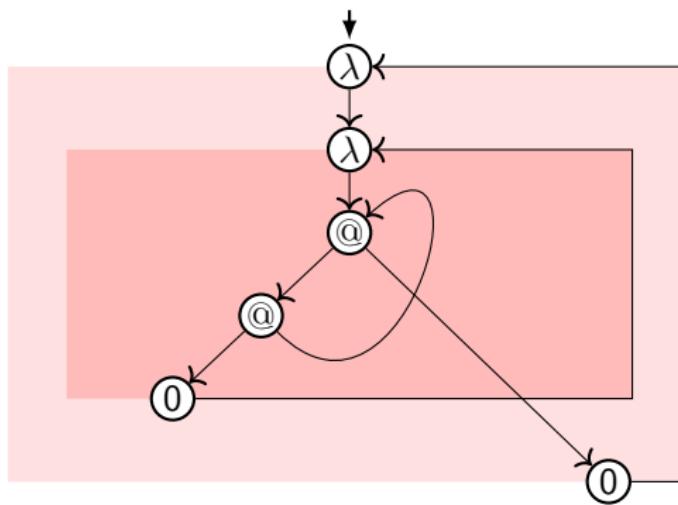
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



first-order term graph (+ scope sets)

# Graph interpretation (example 1)

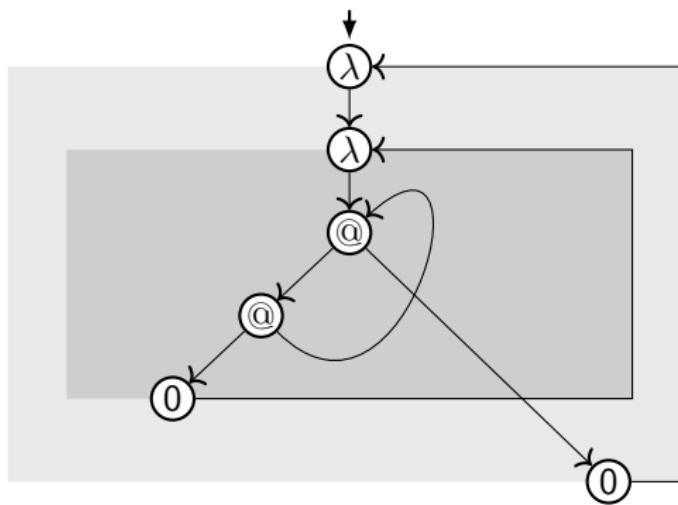
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



higher-order term graph (with scope sets, Blom [2003])

# Graph interpretation (example 1)

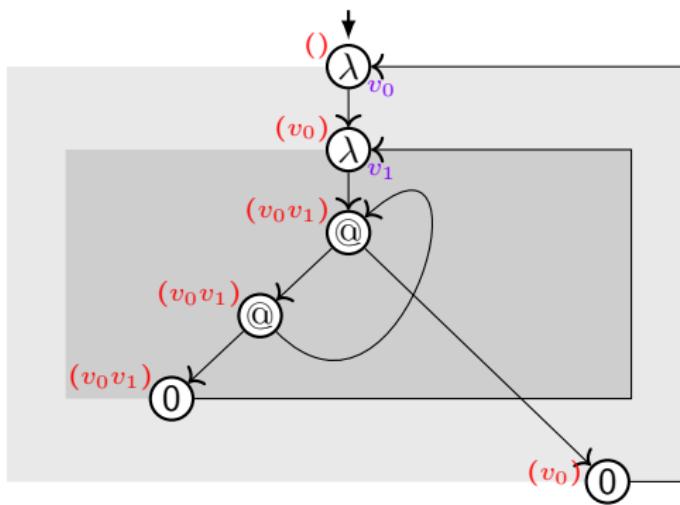
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



higher-order term graph (with scope sets, Blom [2003])

# Graph interpretation (example 1)

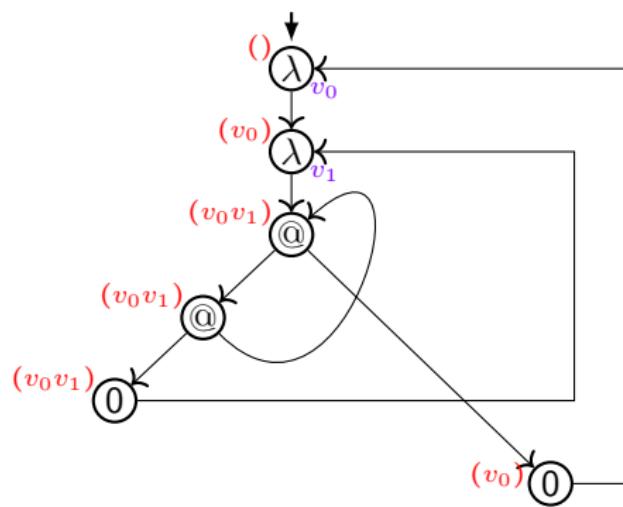
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



higher-order term graph (with scope sets, + abstraction-prefix function)

# Graph interpretation (example 1)

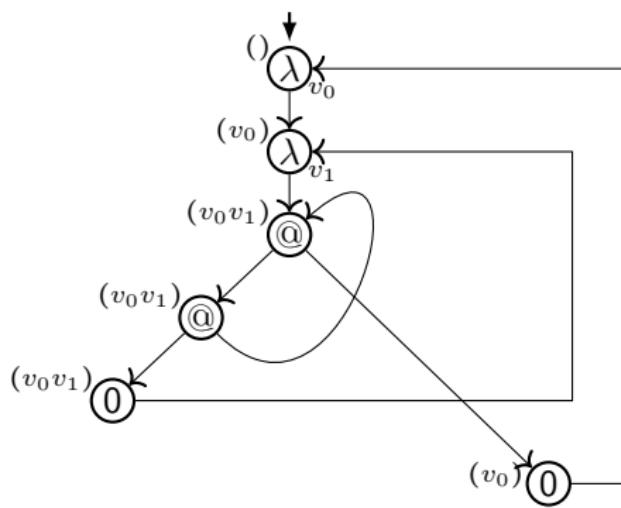
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



higher-order term graph (with abstraction-prefix function)

## Graph interpretation (example 1)

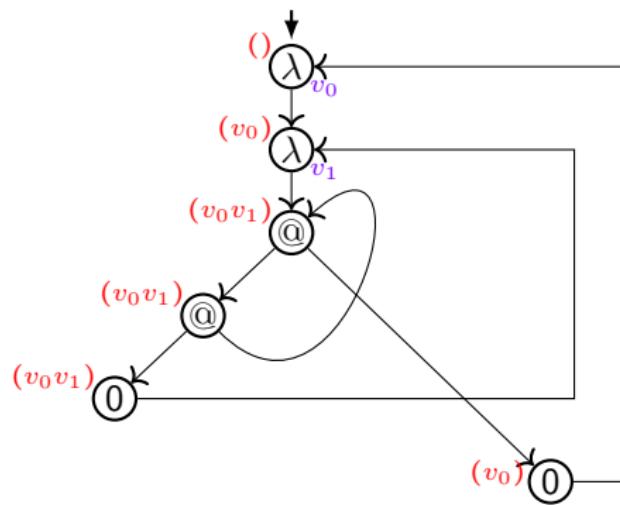
$L_0 = \lambda x. \lambda f. \text{let } r = f\,r\,x \text{ in } r$



## $\lambda$ -higher-order-term-graph $\llbracket L_0 \rrbracket_{\mathcal{H}}$

# Graph interpretation (example 1)

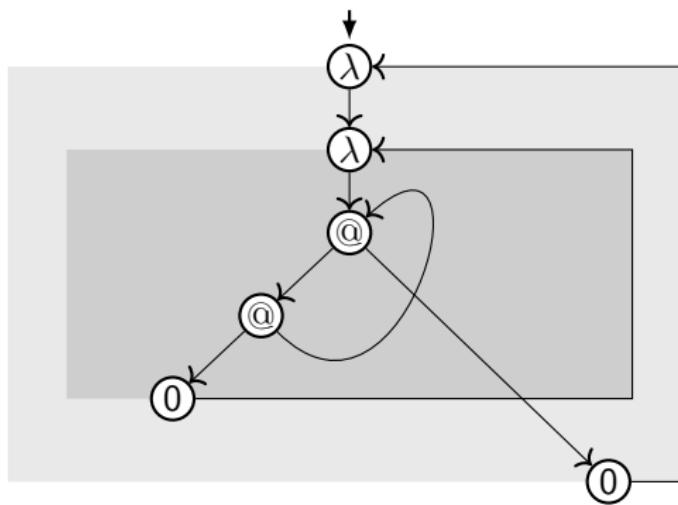
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



first-order term graph (+ abstraction-prefix function)

# Graph interpretation (example 1)

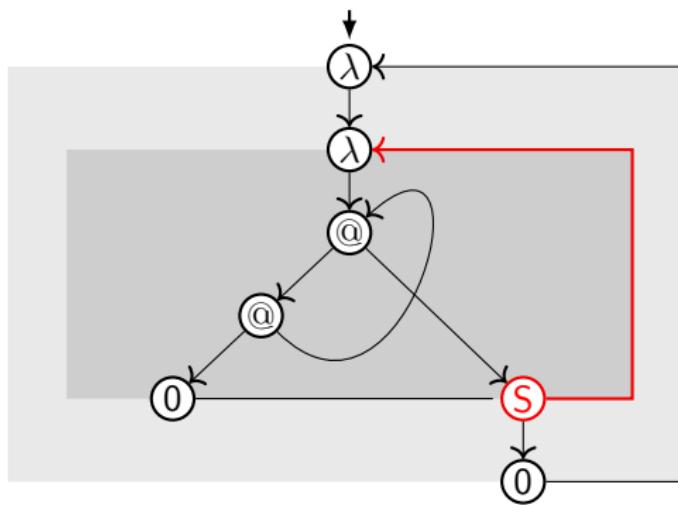
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

# Graph interpretation (example 1)

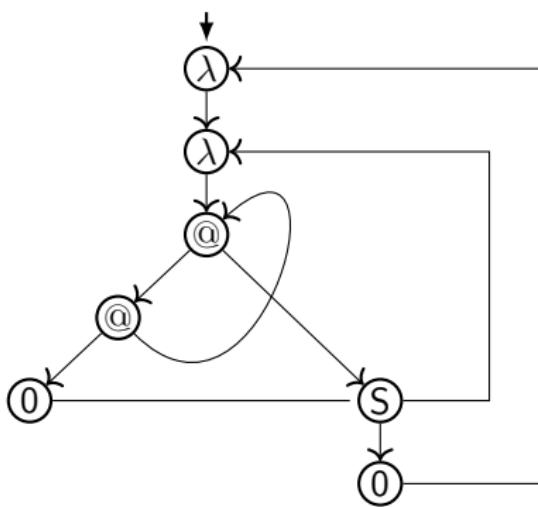
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



first-order term graph with scope vertices with backlinks (+ scope sets)

# Graph interpretation (example 1)

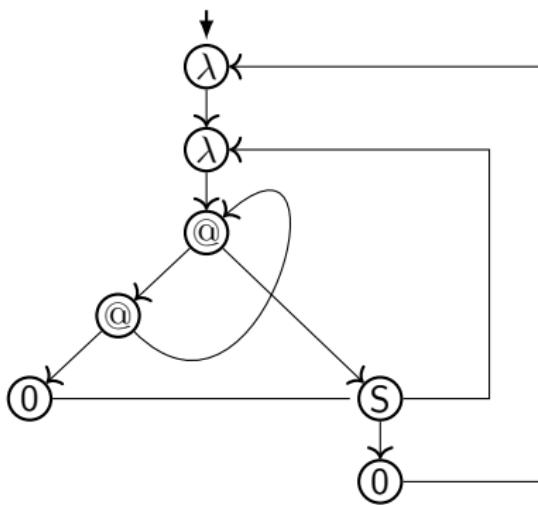
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



first-order term graph with scope vertices with backlinks

# Graph interpretation (example 1)

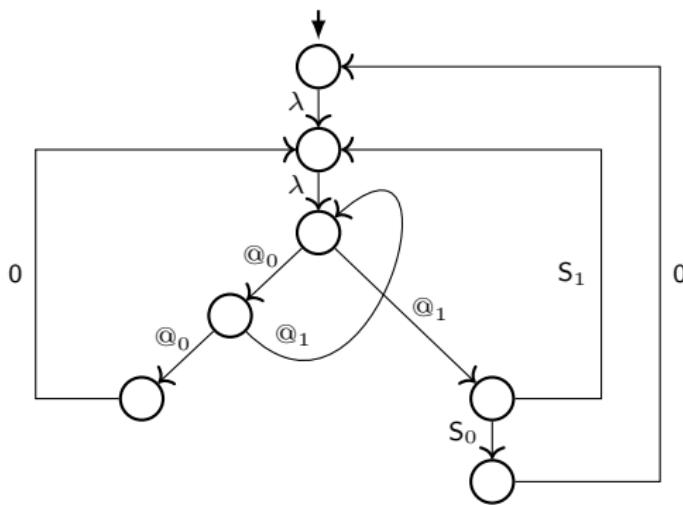
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



$\lambda$ -term-graph  $\llbracket L_0 \rrbracket_T$

# Graph interpretation (example 1)

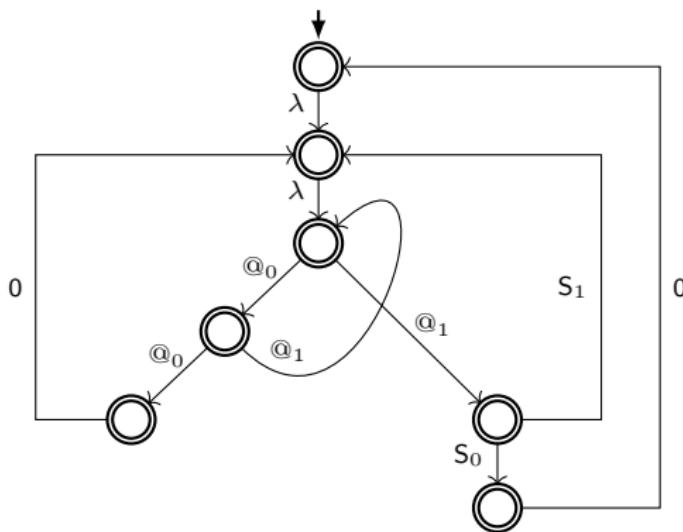
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



incomplete DFA

# Graph interpretation (example 1)

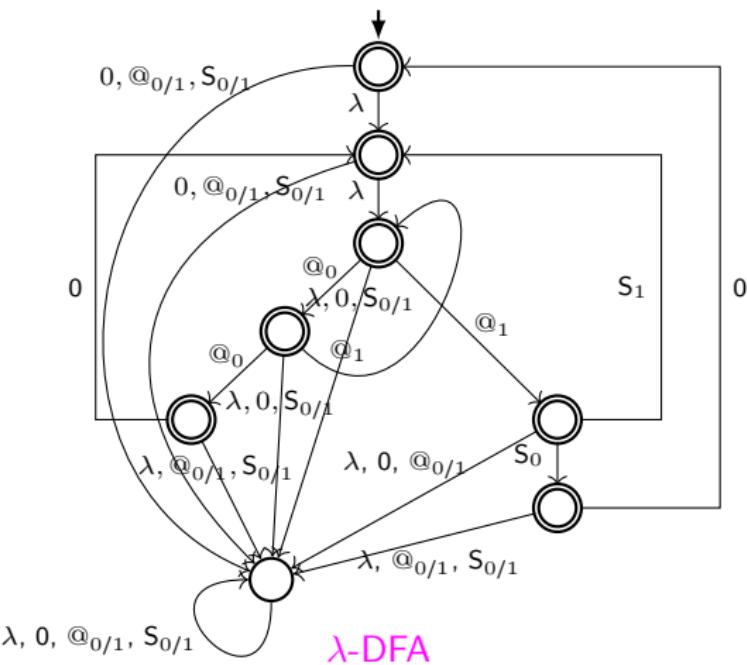
$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$



incomplete  $\lambda$ -DFA

# Graph interpretation (example 1)

$L_0 = \lambda x. \lambda f. \text{let } r = f\ r\ x \text{ in } r$

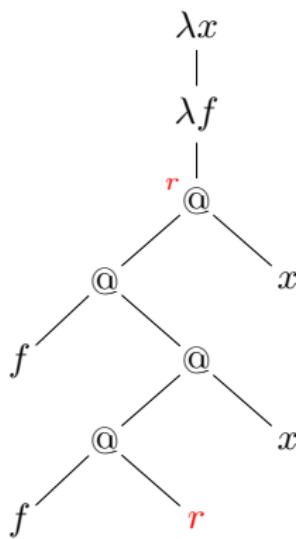


# Graph interpretation (example 2)

$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$

# Graph interpretation (example 2)

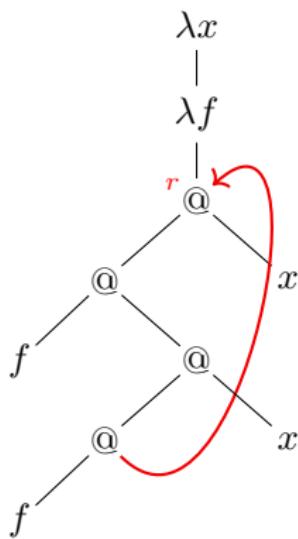
$L = \lambda x. \lambda f. \text{let } r = f(f\ r\ x) \text{ in } r$



syntax tree

# Graph interpretation (example 2)

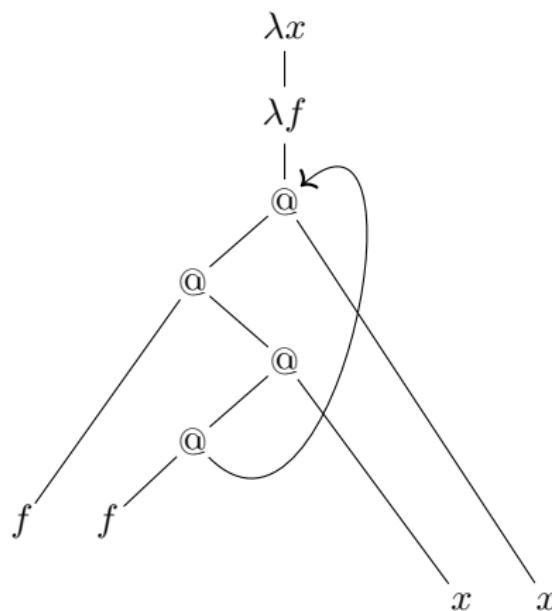
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



syntax tree (+ recursive backlink)

# Graph interpretation (example 2)

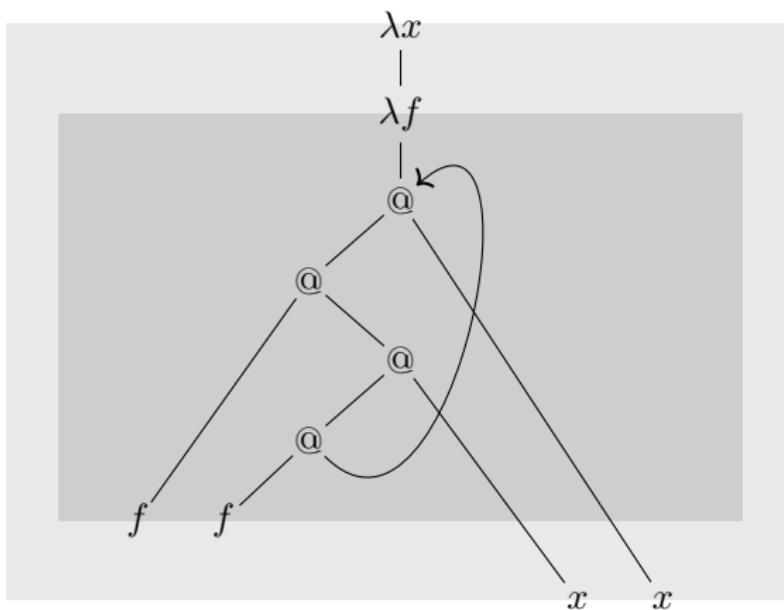
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



syntax tree (+ recursive backlink)

# Graph interpretation (example 2)

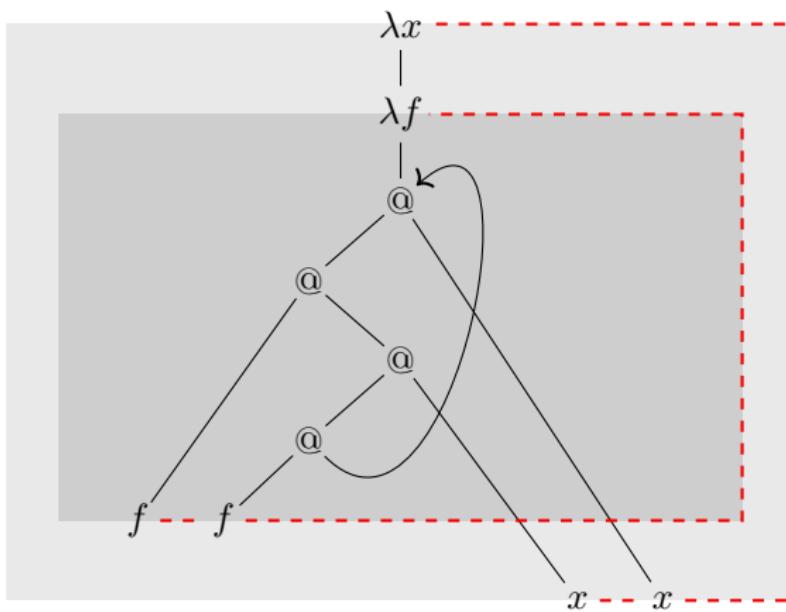
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



syntax tree (+ recursive backlink, + scopes)

## Graph interpretation (example 2)

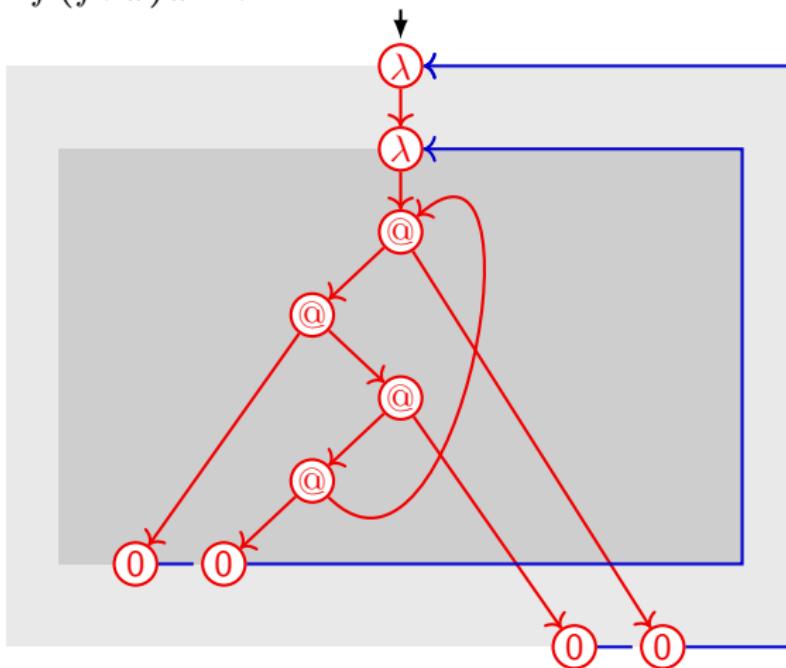
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



syntax tree (+ recursive backlink, + scopes, + binding links)

## Graph interpretation (example 2)

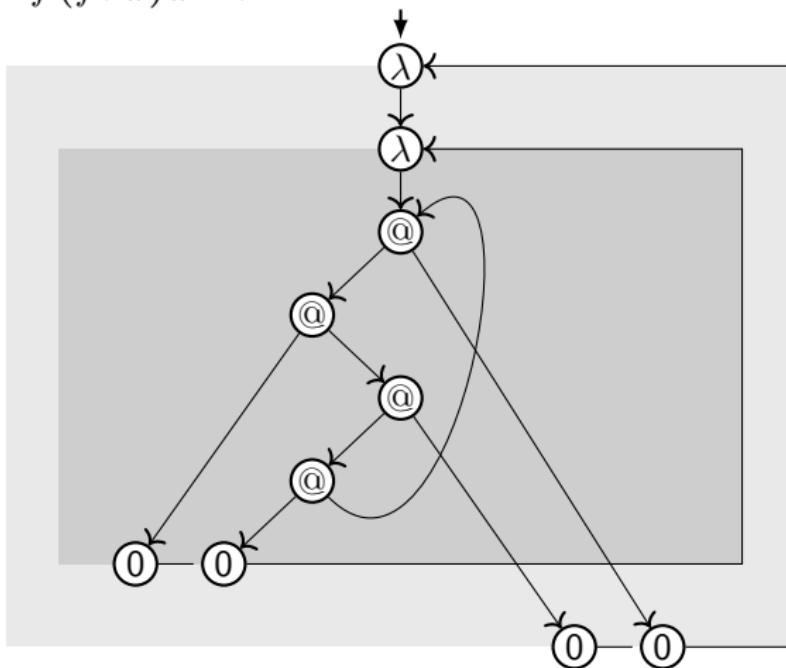
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

## Graph interpretation (example 2)

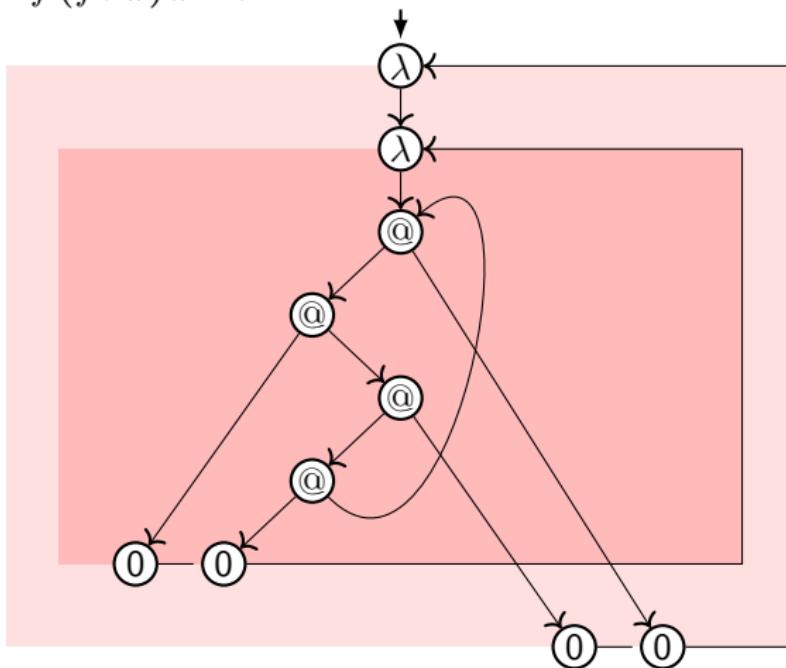
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

## Graph interpretation (example 2)

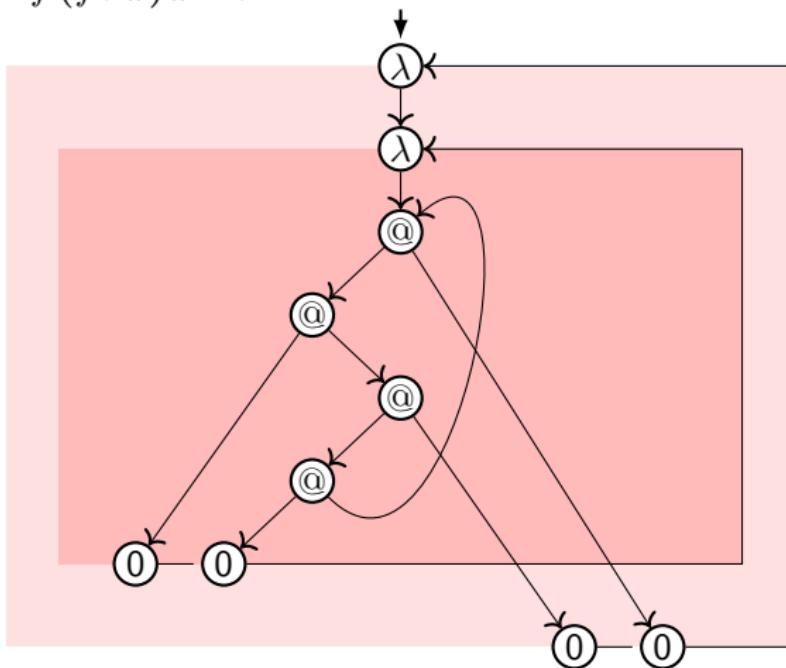
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



first-order term graph (+ scope sets)

## Graph interpretation (example 2)

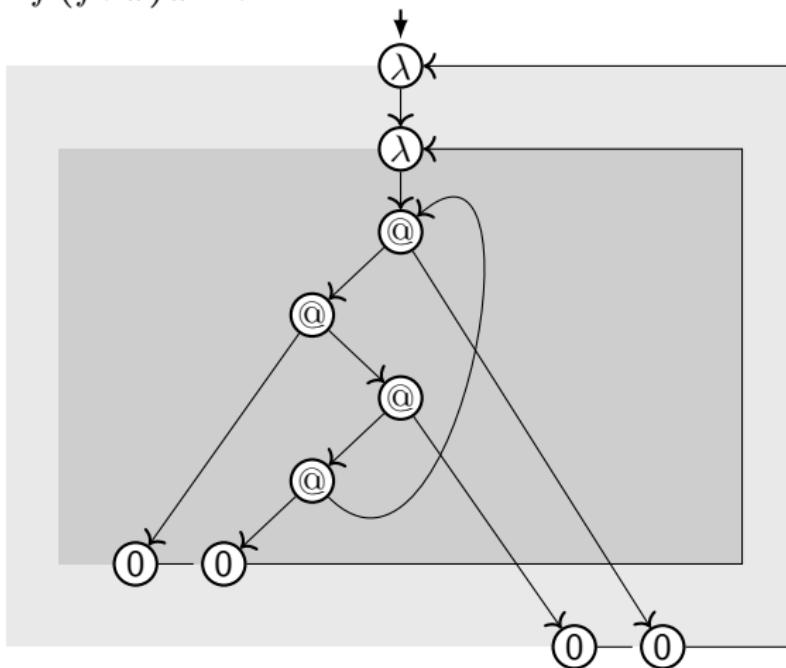
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



higher-order term graph (with scope sets, Blom [2003])

## Graph interpretation (example 2)

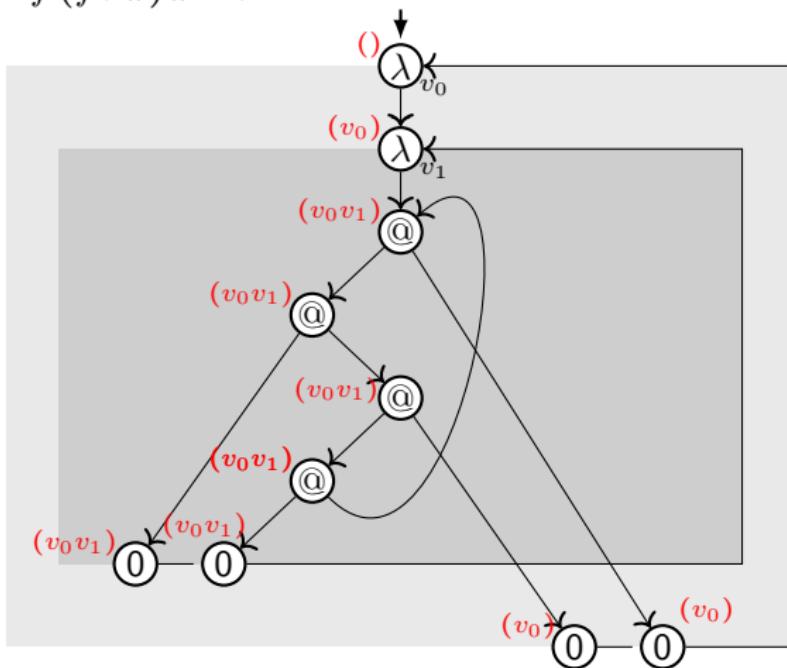
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



higher-order term graph (with scope sets, Blom [2003])

# Graph interpretation (example 2)

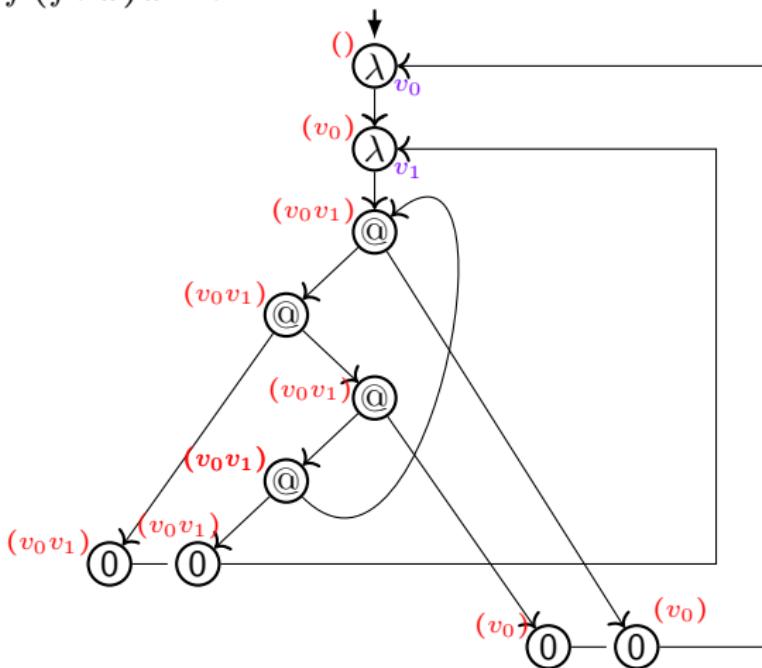
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



higher-order term graph (with scope sets, + abstraction-prefix function)

# Graph interpretation (example 2)

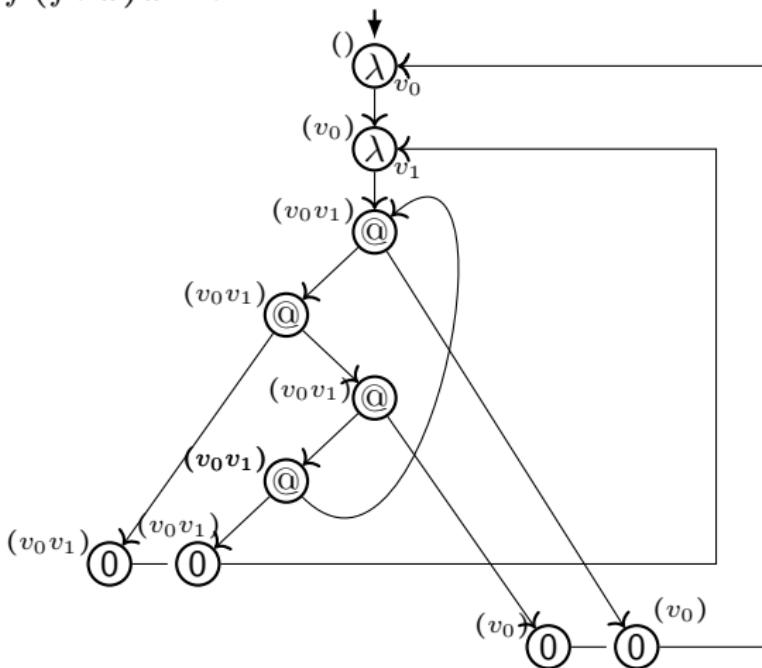
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



higher-order term graph (with abstraction-prefix function)

# Graph interpretation (example 2)

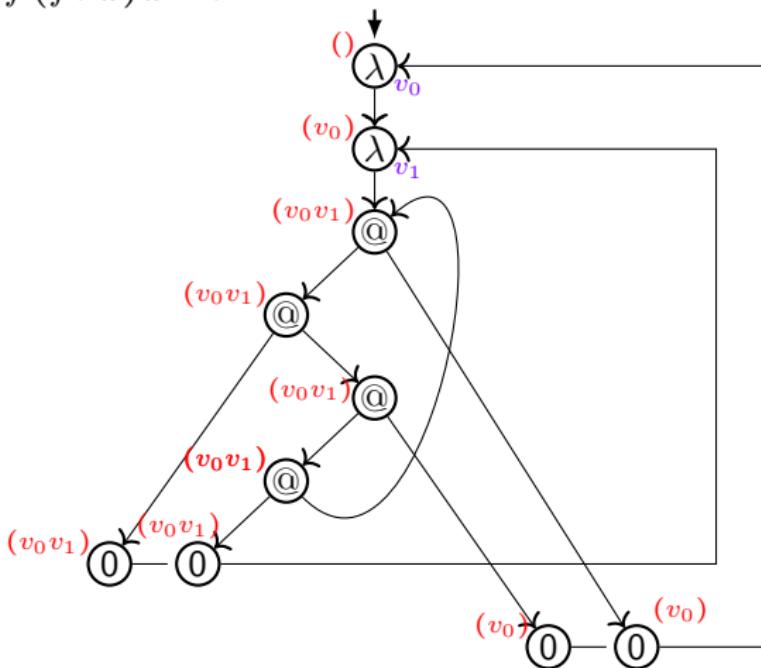
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



$\lambda$ -higher-order-term-graph  $\llbracket L \rrbracket_{\mathcal{H}}$

## Graph interpretation (example 2)

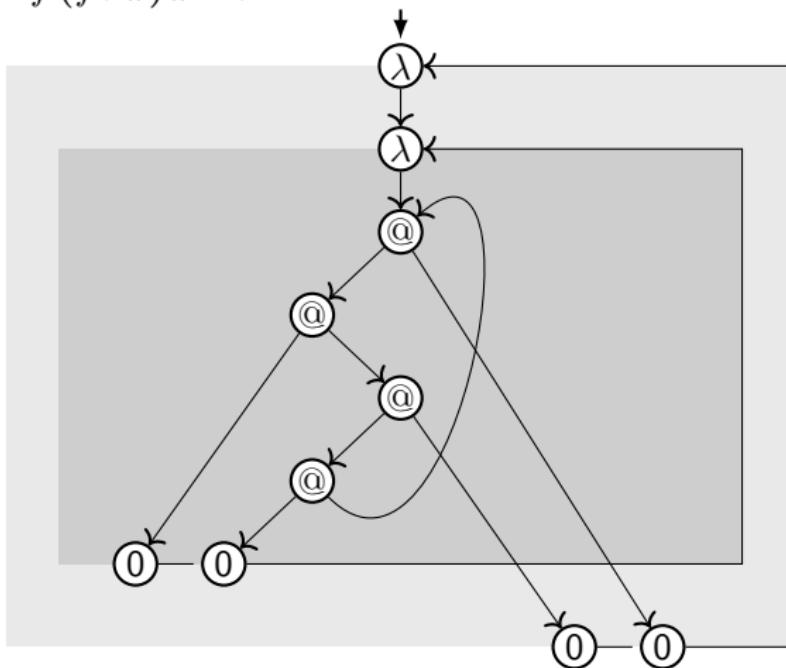
$L = \lambda x. \lambda f. \text{let } r = f(frx) \text{ in } r$



first-order term graph (+ abstraction-prefix function)

## Graph interpretation (example 2)

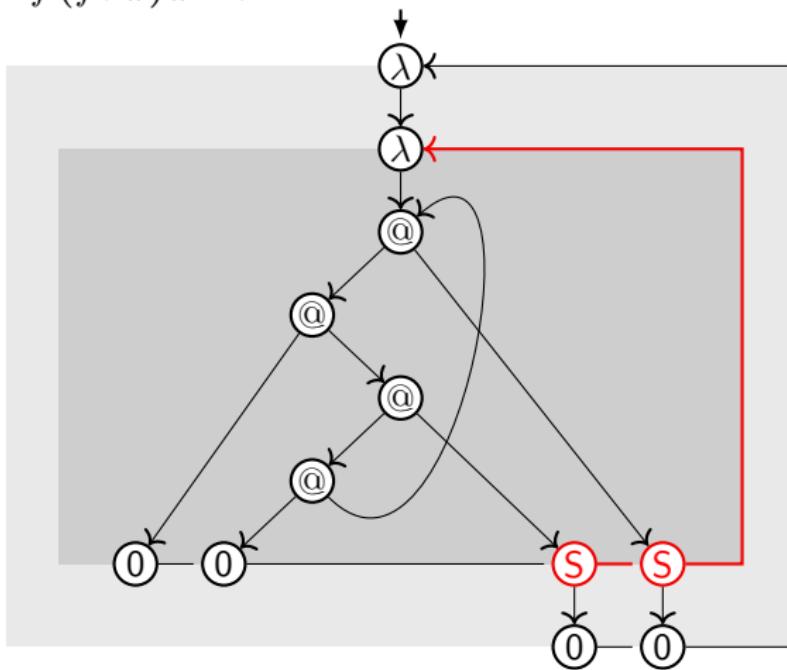
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



first-order term graph with binding backlinks (+ scope sets)

## Graph interpretation (example 2)

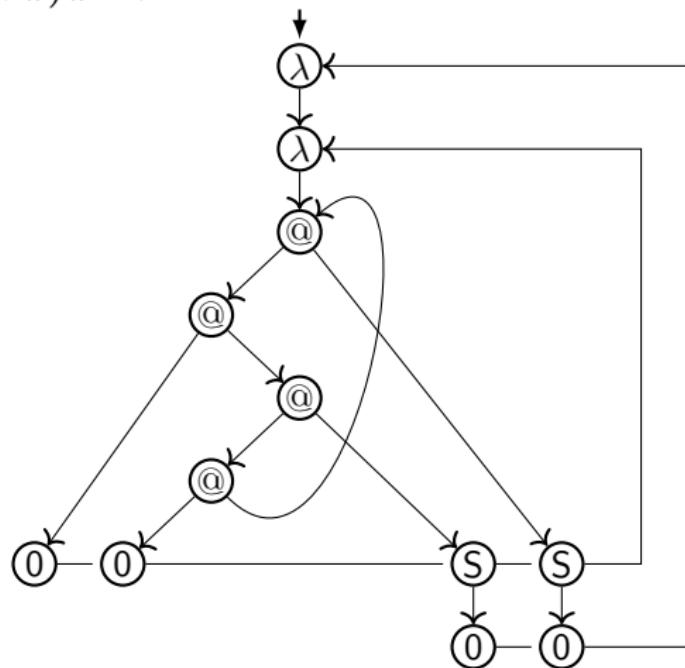
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



first-order term graph with scope vertices with backlinks (+ scope sets)

## Graph interpretation (example 2)

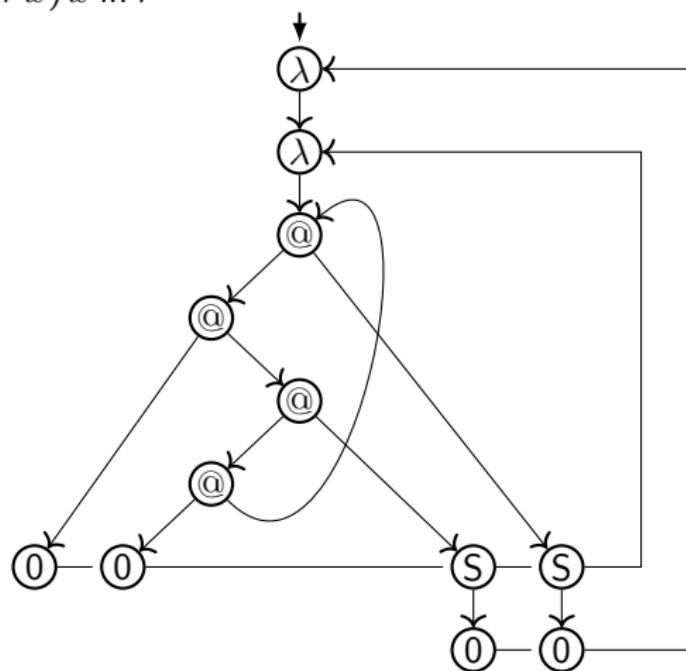
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



first-order term graph with scope vertices with backlinks

# Graph interpretation (example 2)

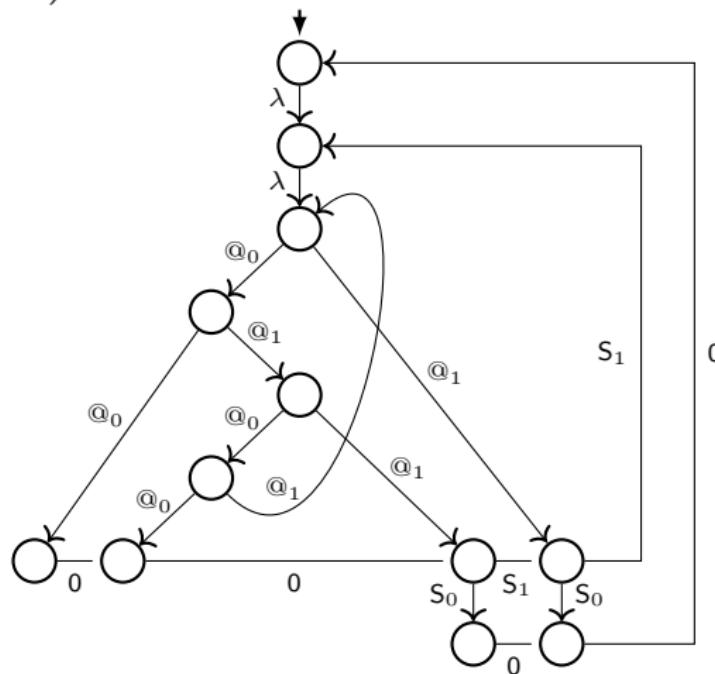
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



$\lambda$ -term-graph  $\llbracket L \rrbracket_{\mathcal{T}}$

# Graph interpretation (example 2)

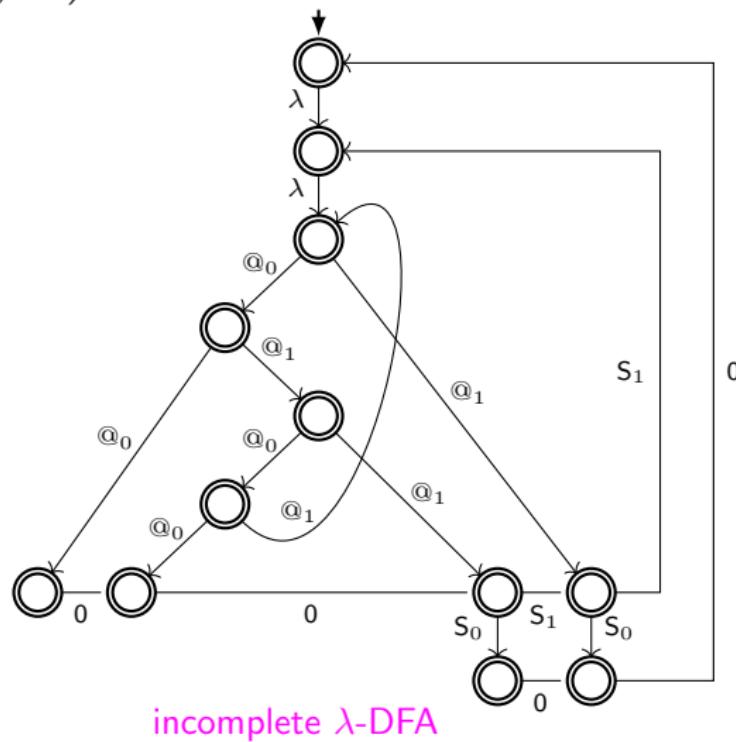
$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



incomplete DFA

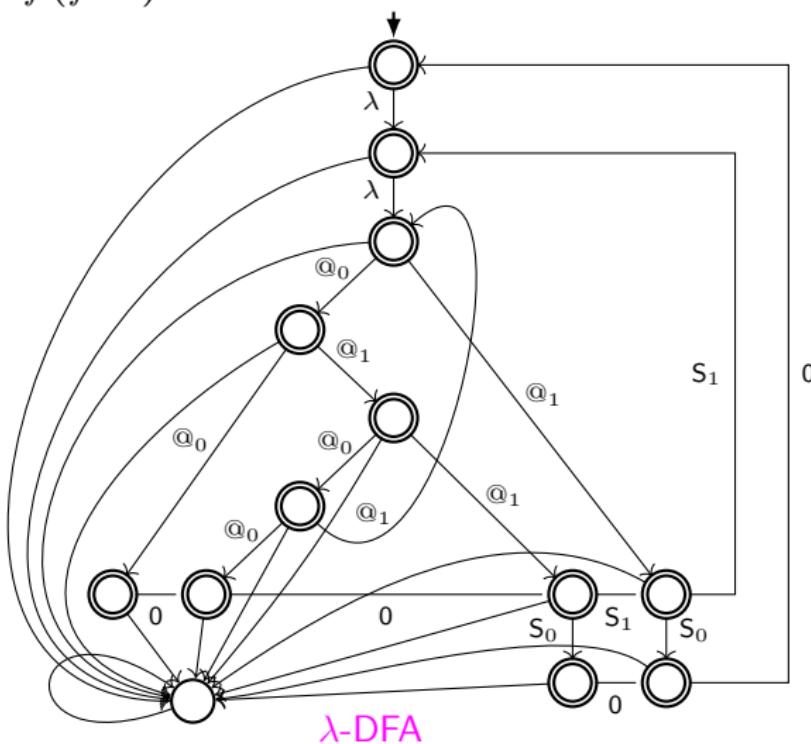
# Graph interpretation (example 2)

$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$

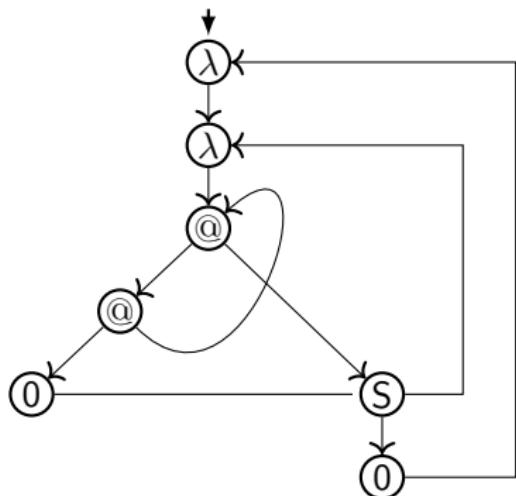
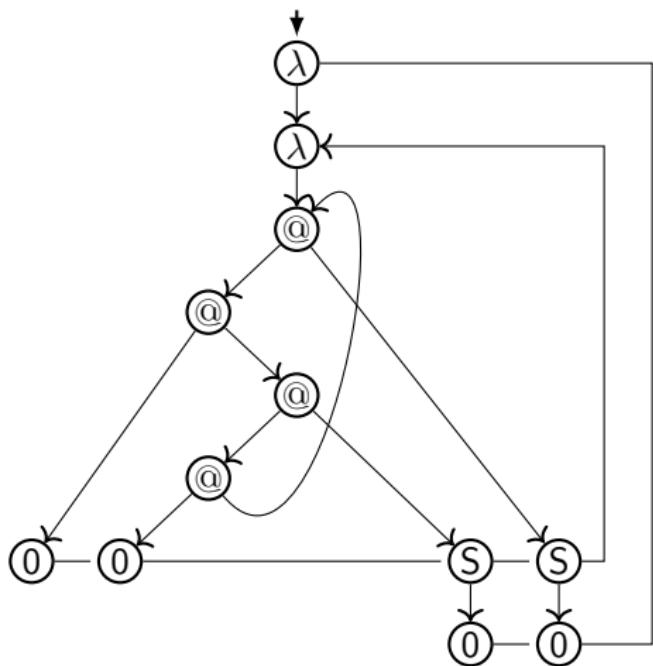


# Graph interpretation (example 2)

$L = \lambda x. \lambda f. \text{let } r = f(f\,r\,x) \text{ in } r$



# Graph interpretation (examples 1 and 2)


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 

 $\llbracket L \rrbracket_{\mathcal{T}}$

# Interpretation $\llbracket \cdot \rrbracket_{\mathcal{T}}$ : properties (cont.)

interpretation  $\lambda_{\text{letrec}}$ -term  $L \mapsto \lambda\text{-term-graph } \llbracket L \rrbracket_{\mathcal{T}}$

- ▶ defined by induction on structure of  $L$
- ▶ similar analysis as fully-lazy lambda-lifting
- ▶ yields **eager-scope  $\lambda$ -term-graphs**:  $\sim$  minimal scopes

## Theorem

For  $\lambda_{\text{letrec}}$ -terms  $L_1$  and  $L_2$  it holds: Equality of infinite unfolding coincides with bisimilarity of  $\lambda$ -term-graph interpretations:

$$\llbracket L_1 \rrbracket_{\lambda^\infty} = \llbracket L_2 \rrbracket_{\lambda^\infty} \iff \llbracket L_1 \rrbracket_{\mathcal{T}} \simeq \llbracket L_2 \rrbracket_{\mathcal{T}}$$

# Interpretation $\llbracket \cdot \rrbracket_{\mathcal{T}}$ : properties (cont.)

interpretation  $\lambda_{\text{letrec}}$ -term  $L \mapsto \lambda\text{-term-graph } \llbracket L \rrbracket_{\mathcal{T}}$

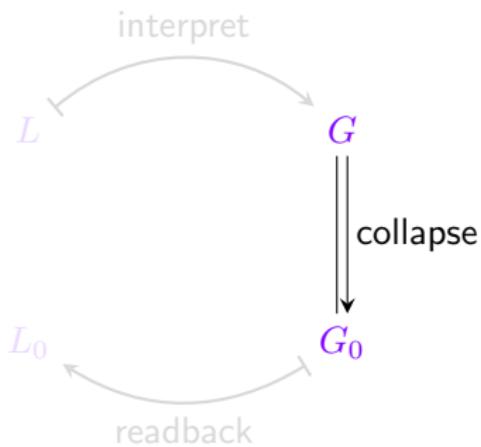
- ▶ defined by induction on structure of  $L$
- ▶ similar analysis as fully-lazy lambda-lifting
- ▶ yields **eager-scope  $\lambda$ -term-graphs**:  $\sim$  minimal scopes

## Theorem

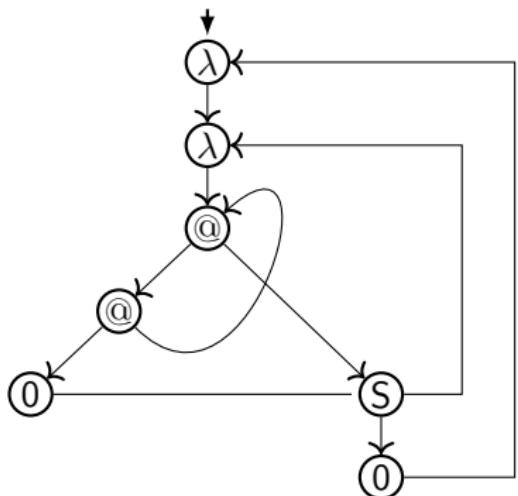
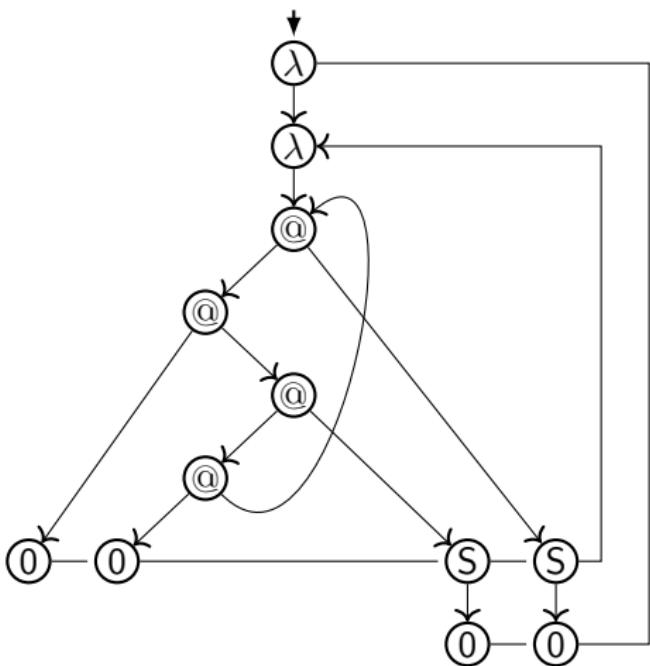
For  $\lambda_{\text{letrec}}$ -terms  $L_1$  and  $L_2$  it holds: Equality of infinite unfolding coincides with **bisimilarity** of  $\lambda$ -term-graph interpretations:

$$\llbracket L_1 \rrbracket_{\lambda^\infty} = \llbracket L_2 \rrbracket_{\lambda^\infty} \iff \llbracket L_1 \rrbracket_{\mathcal{T}} \simeq \llbracket L_2 \rrbracket_{\mathcal{T}}$$

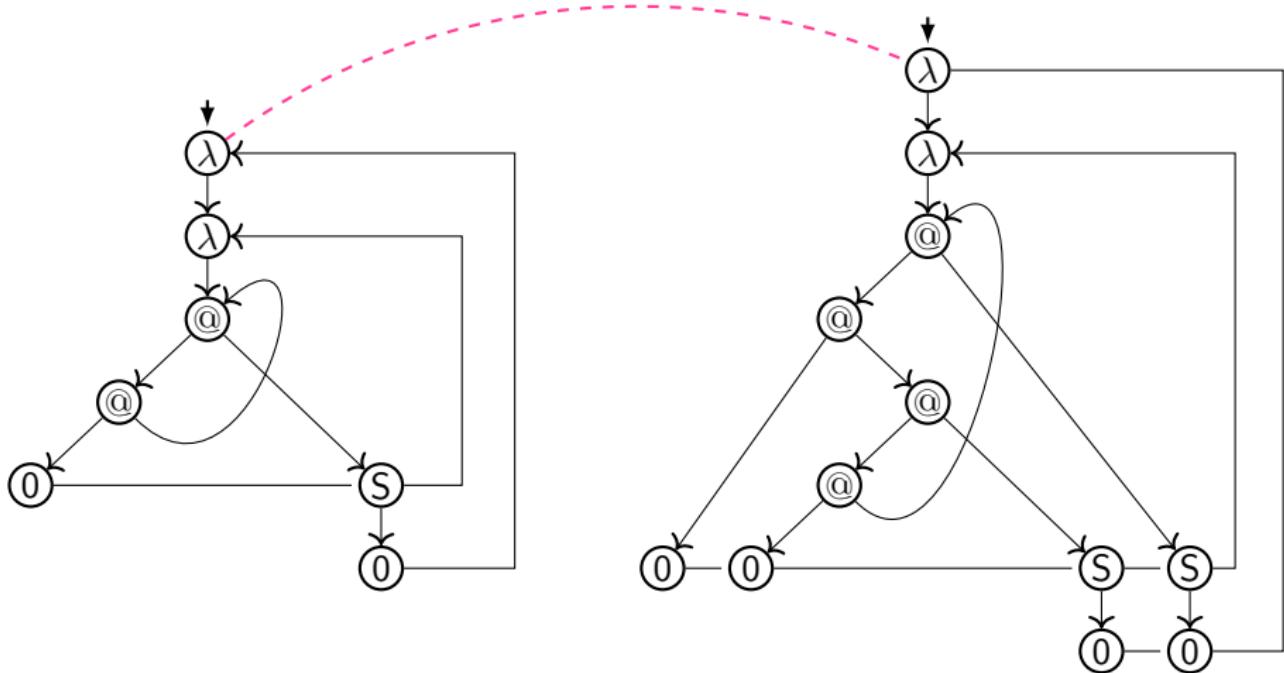
# Collapse



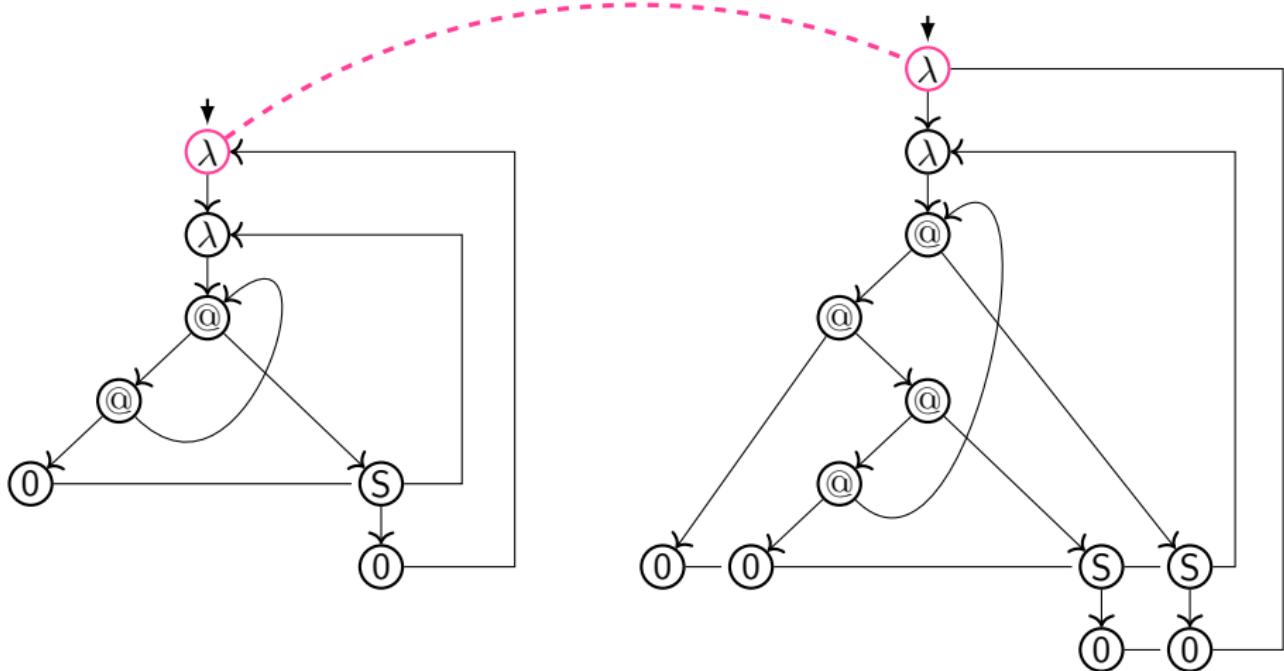
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 

 $\llbracket L \rrbracket_{\mathcal{T}}$

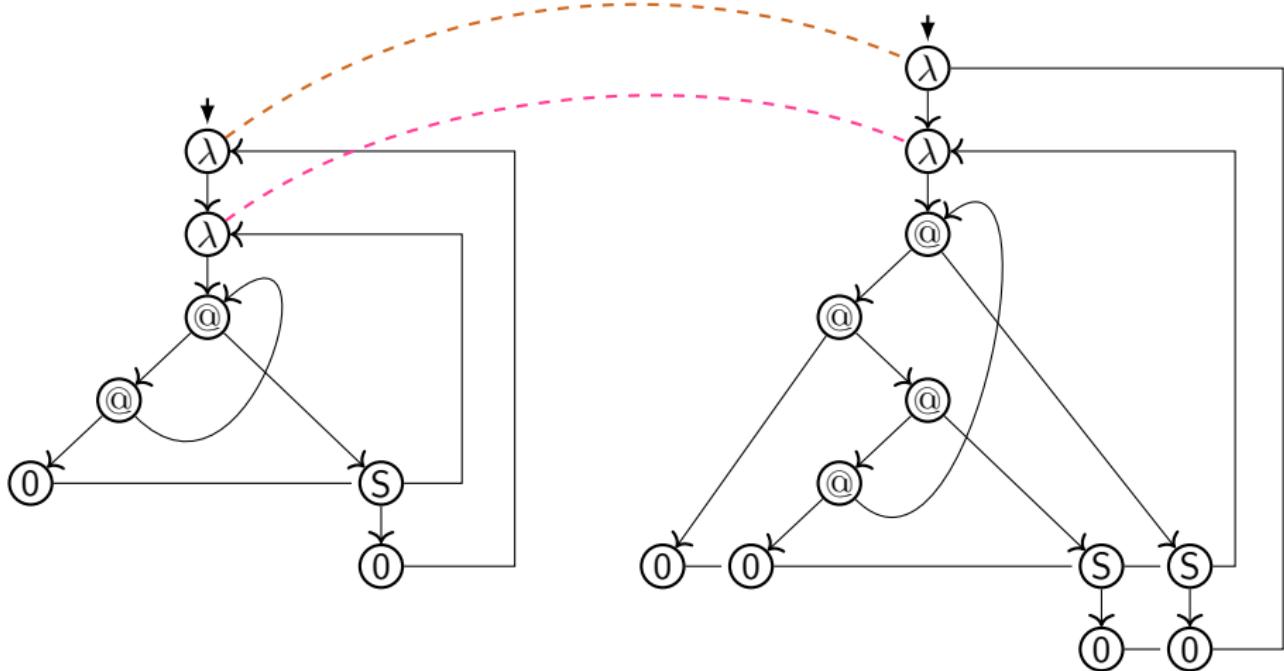
# Bisimulation check between $\lambda$ -term-graphs


 $[[L_0]]_\tau$ 
 $[[L]]_\tau$

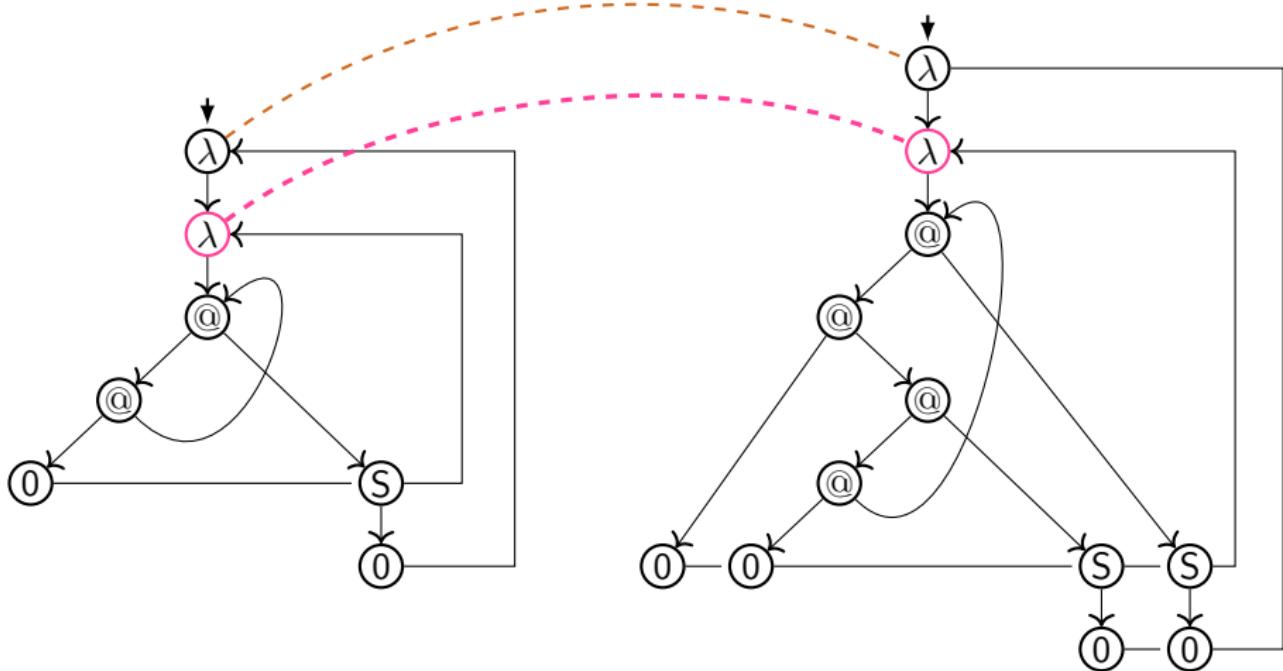
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

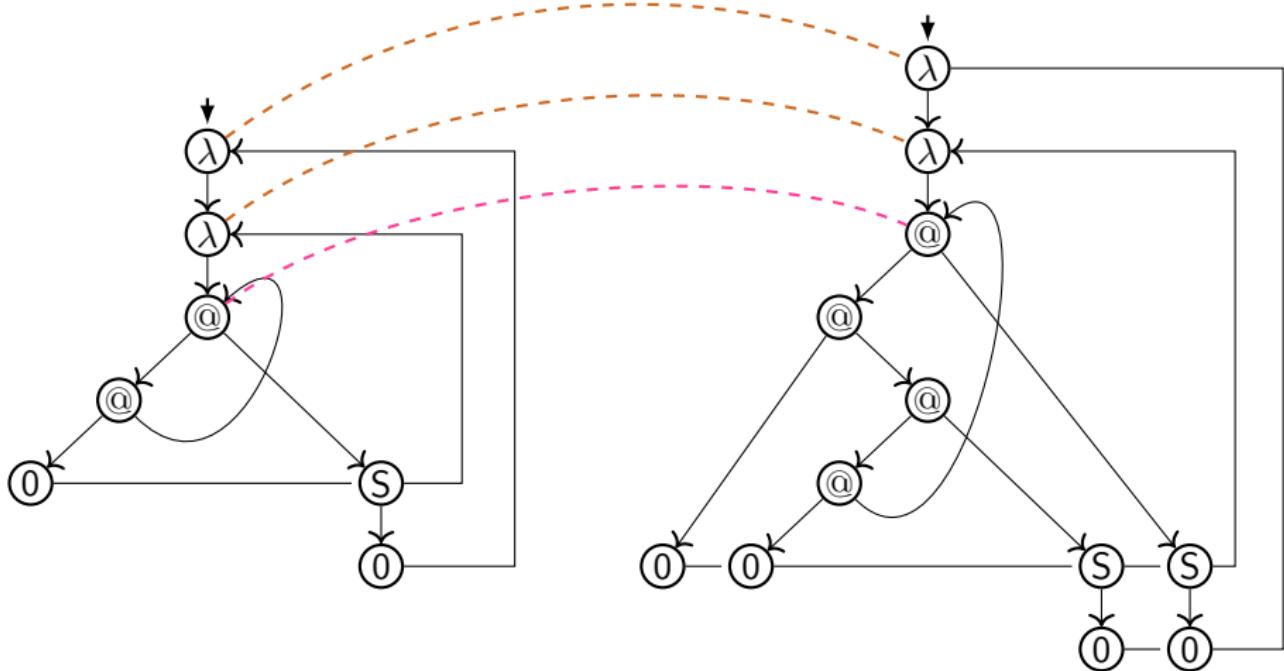
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_T$ 
 $\llbracket L \rrbracket_T$

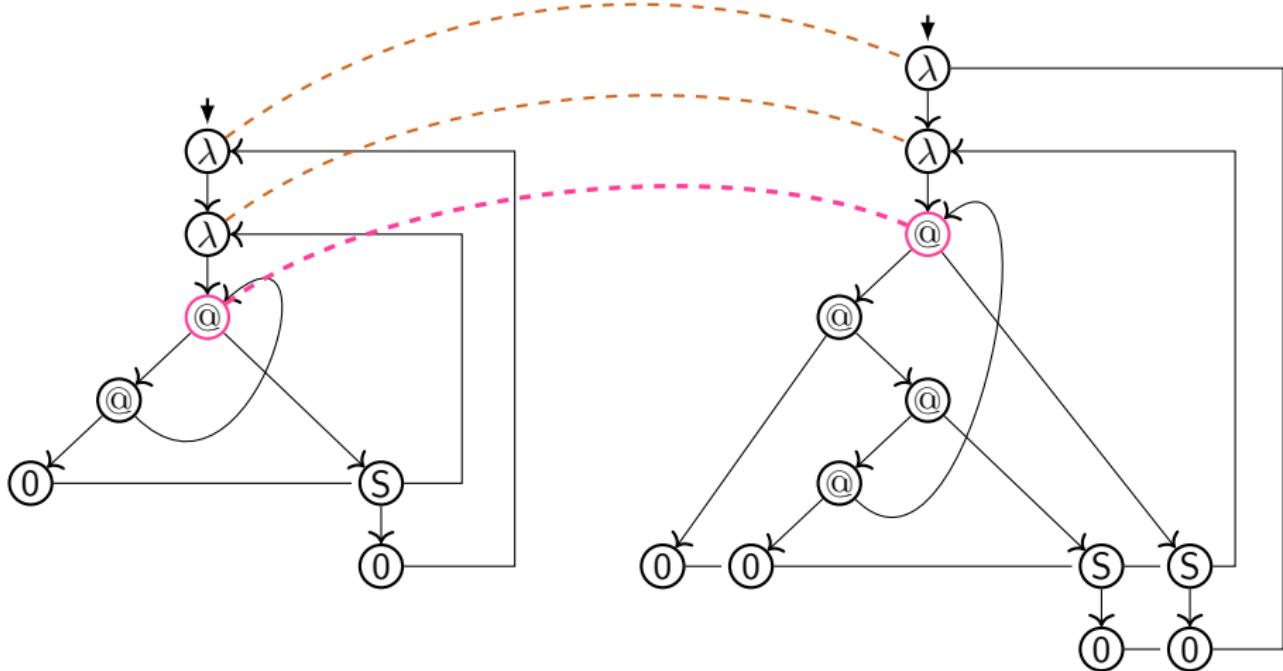
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

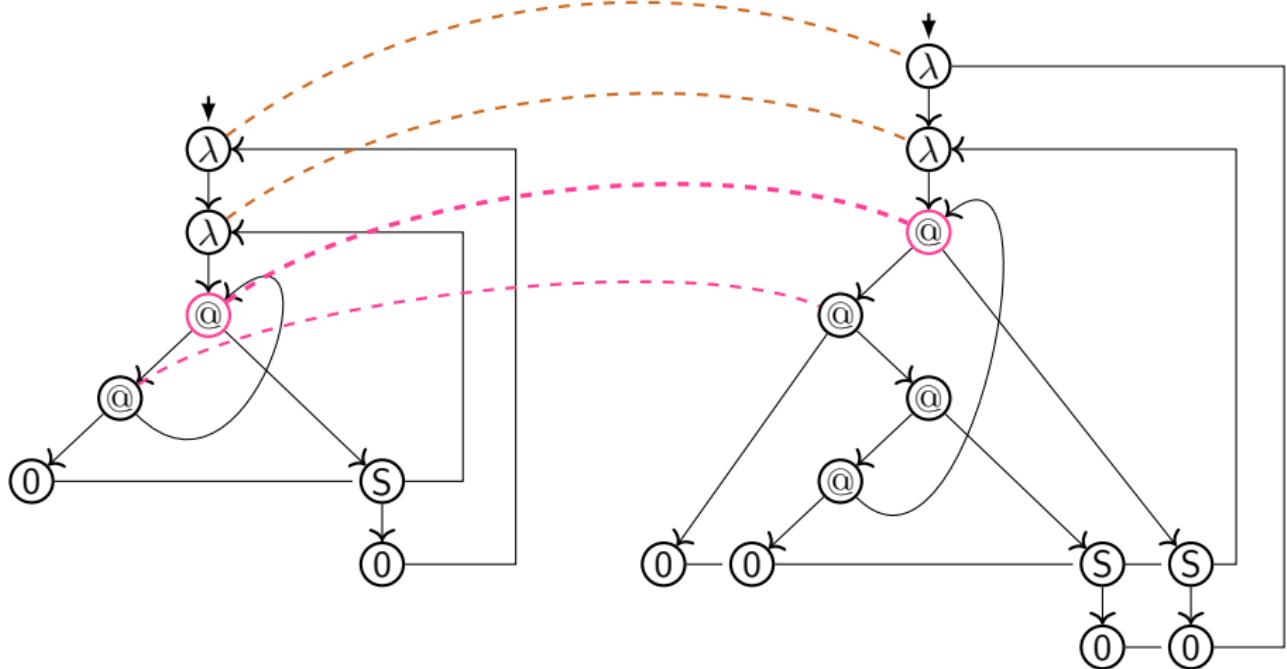
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_T$ 
 $\llbracket L \rrbracket_T$

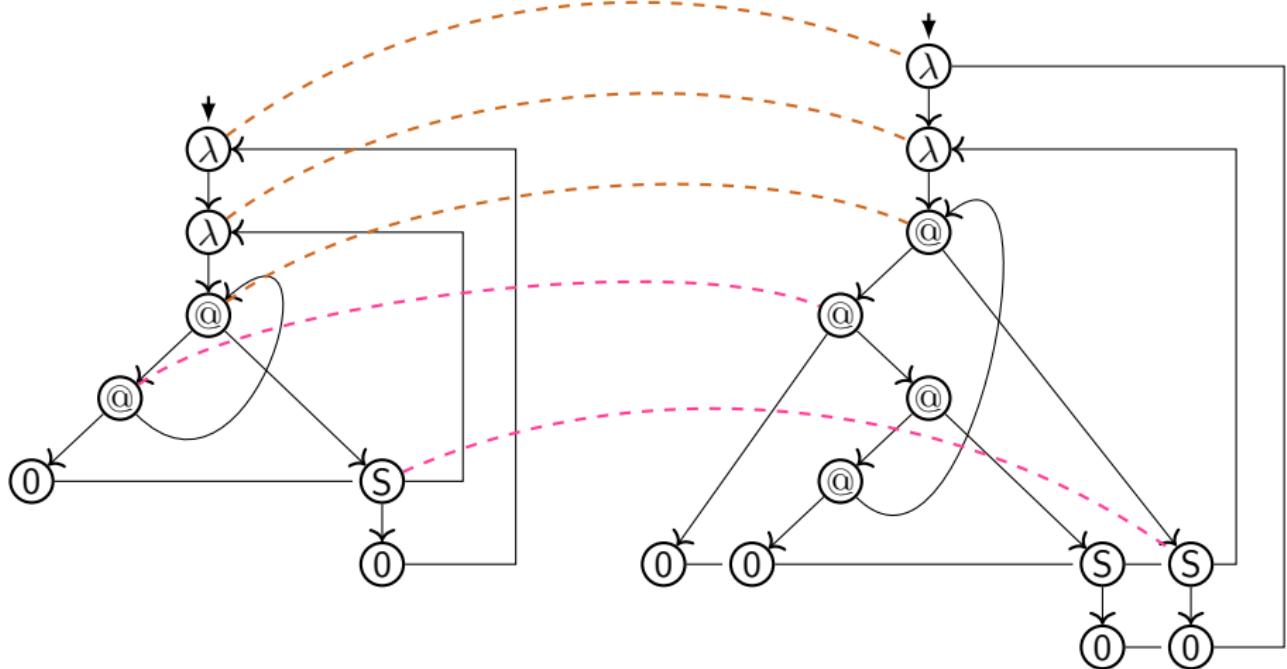
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

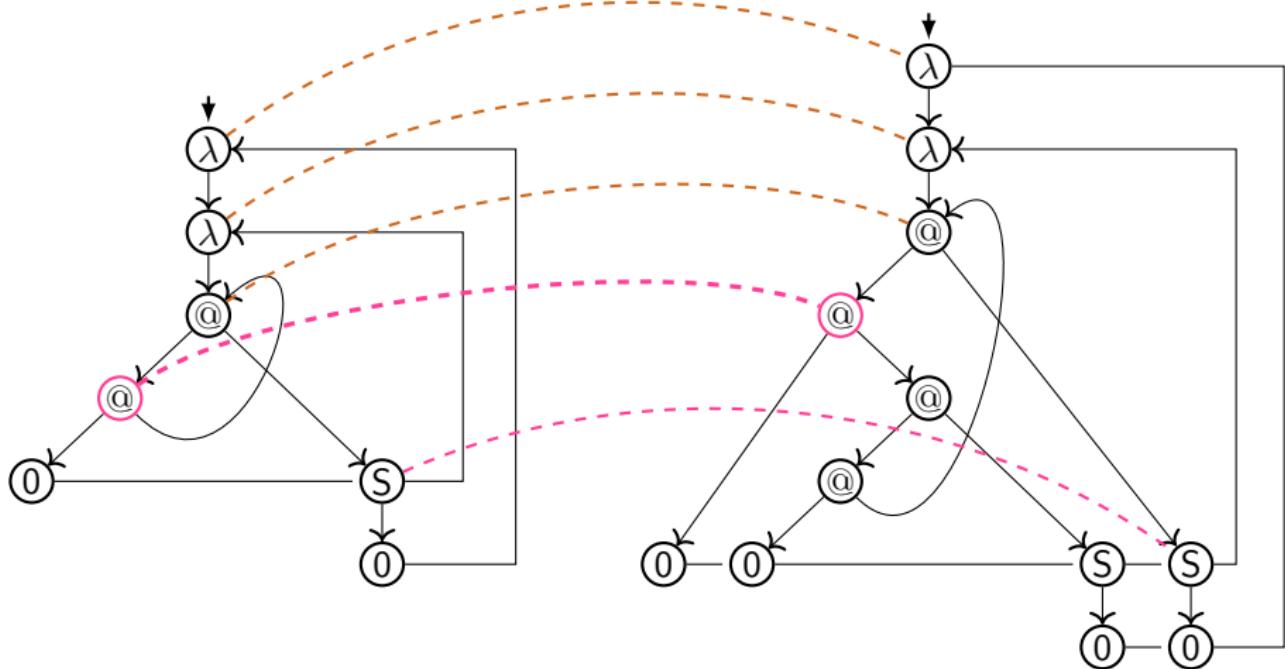
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

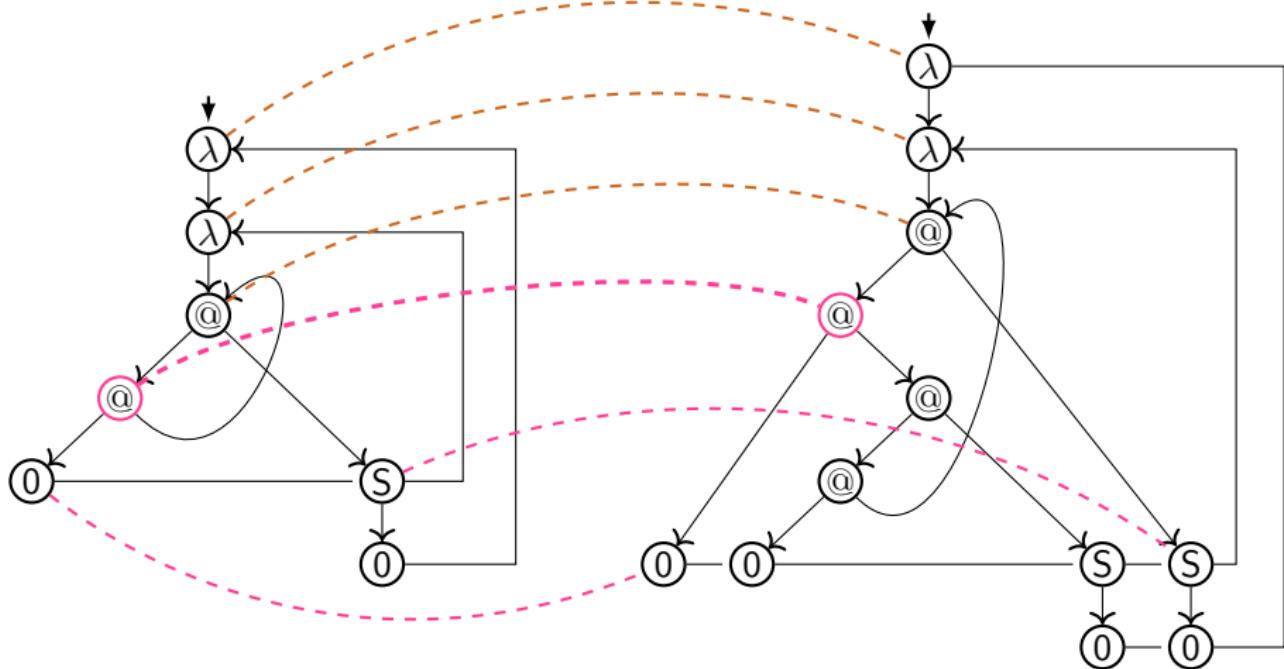
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

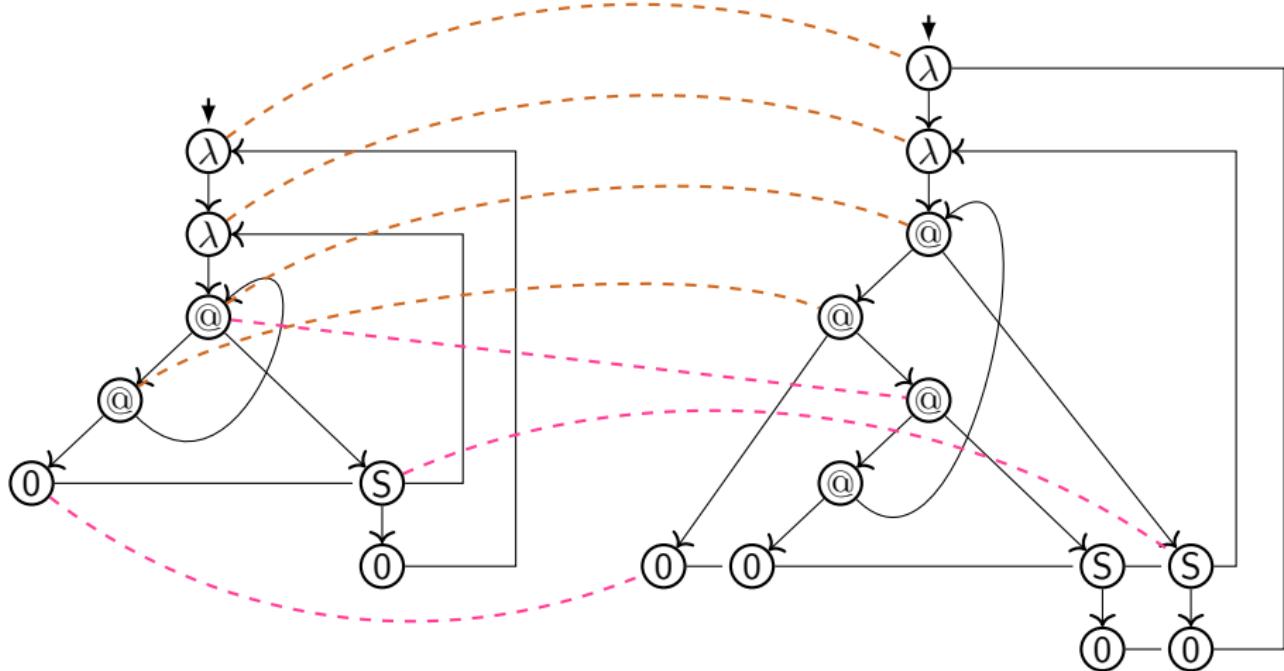
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

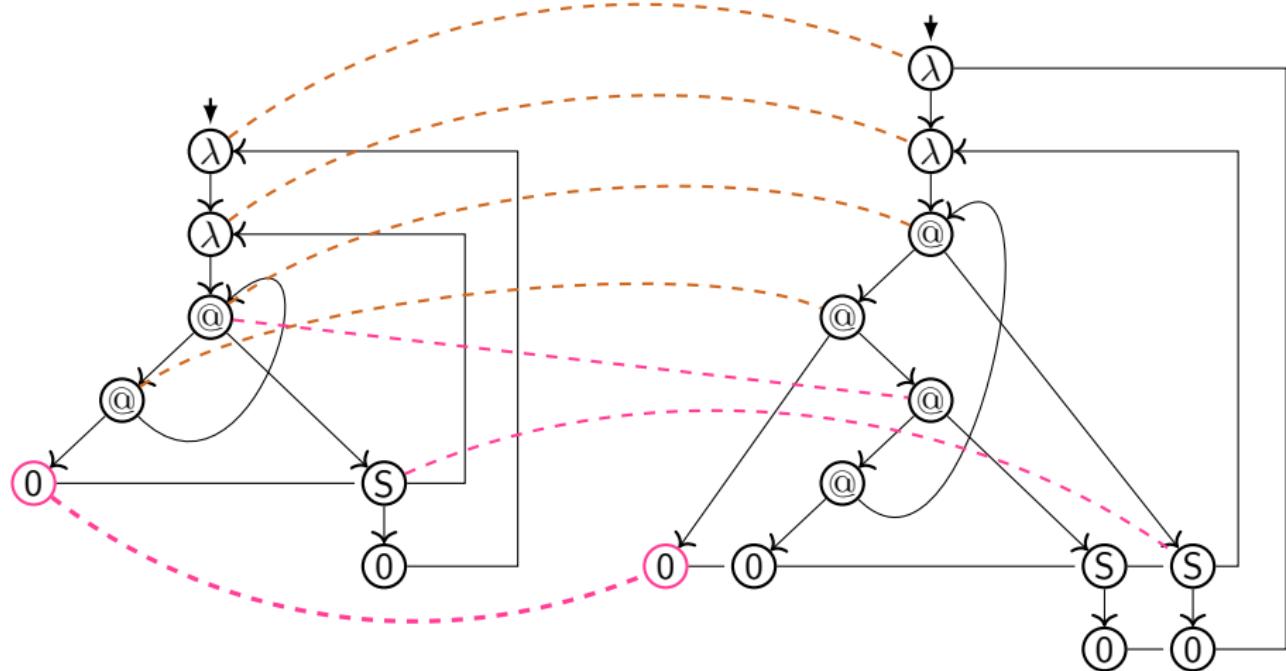
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

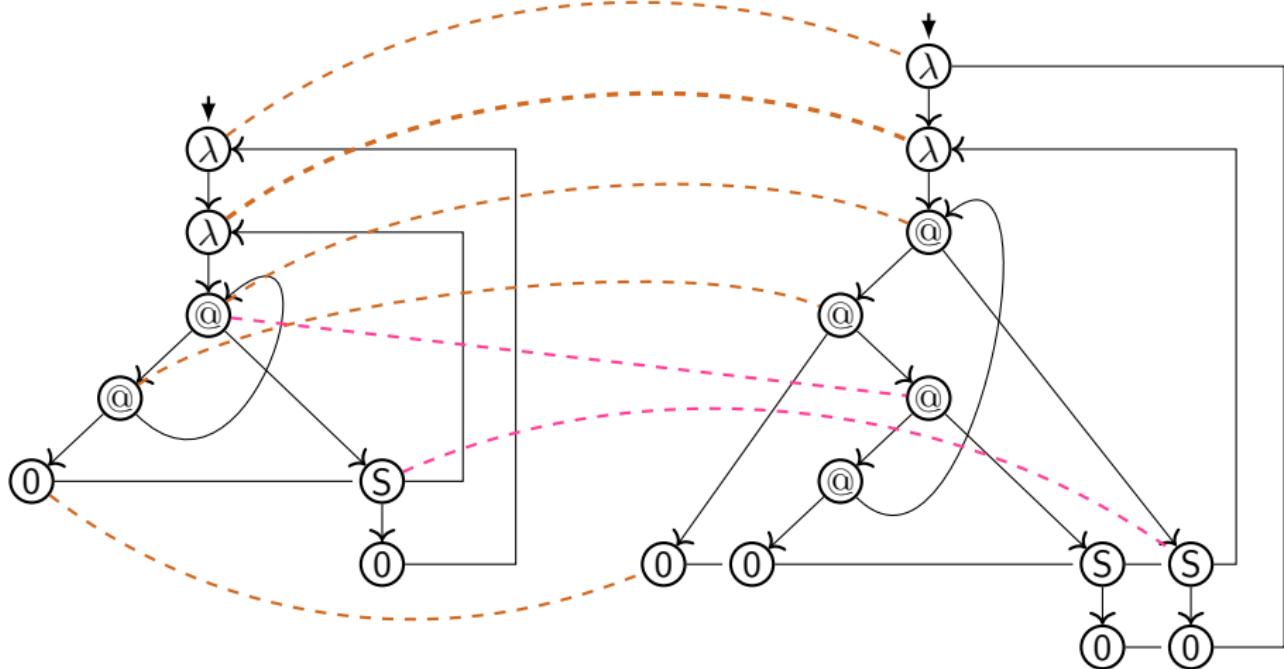
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

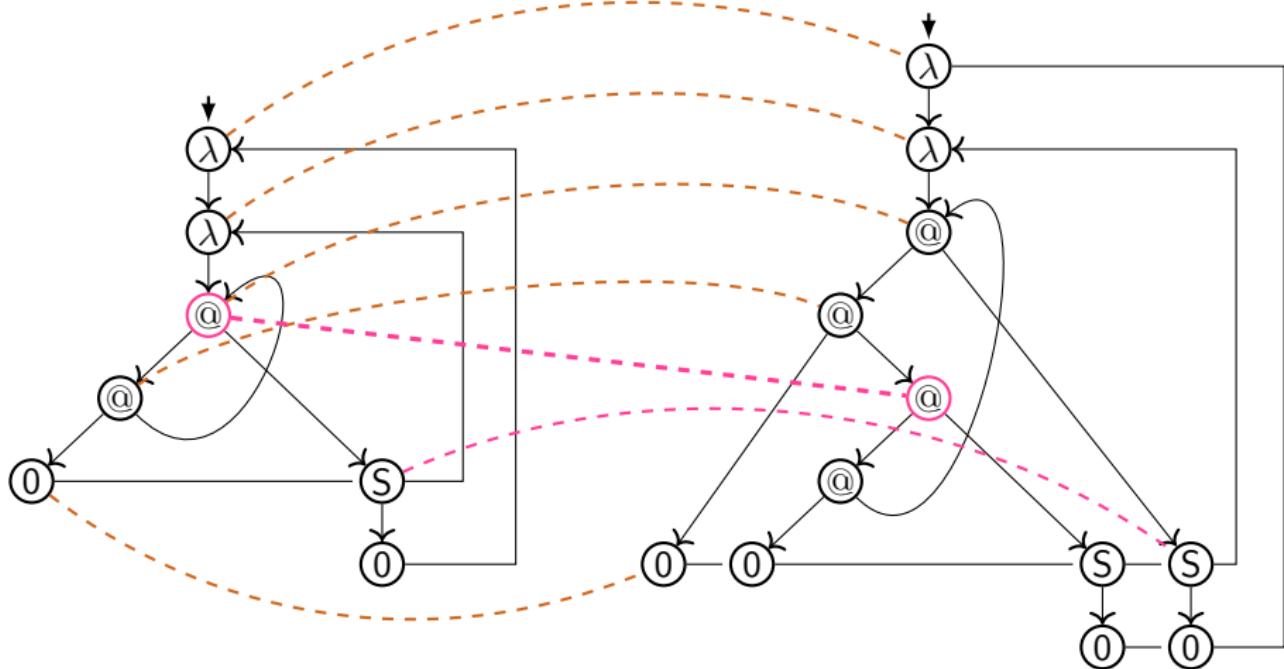
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

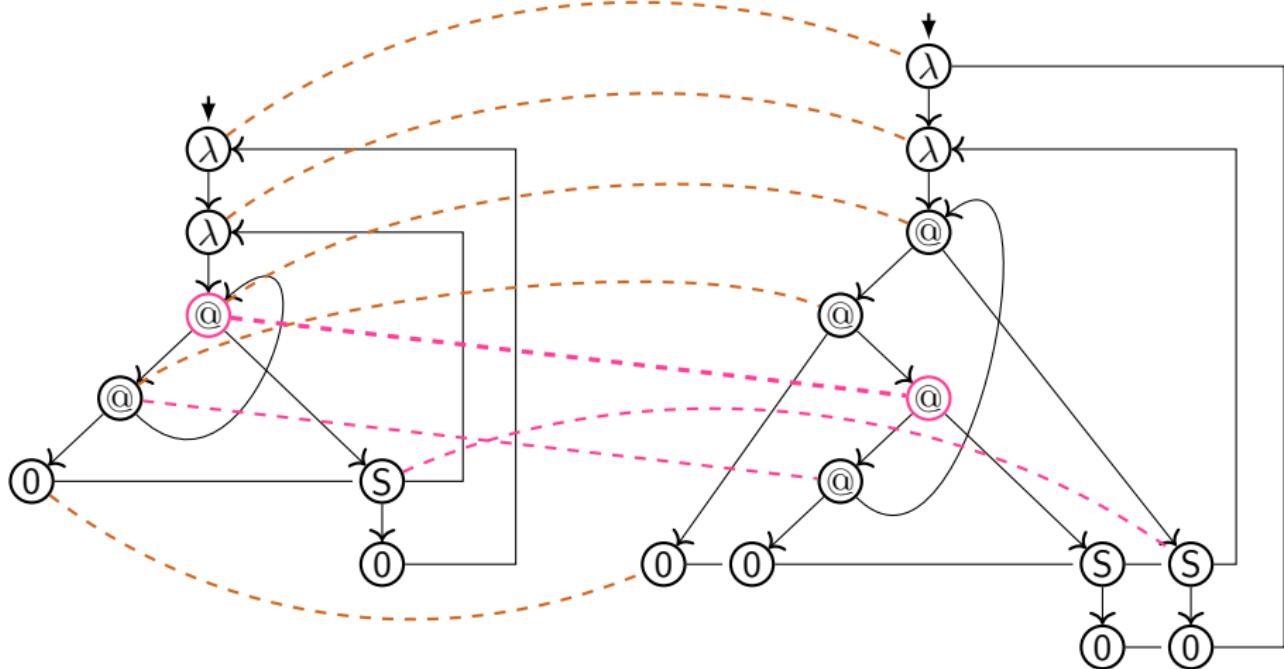
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

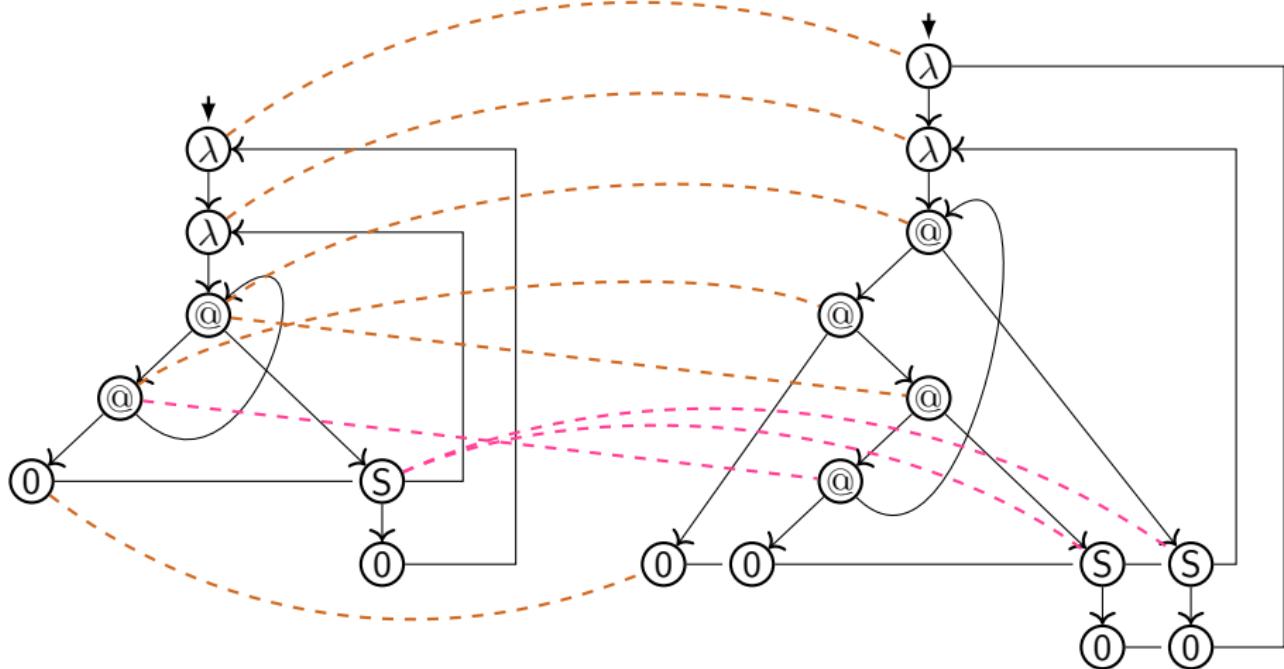
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

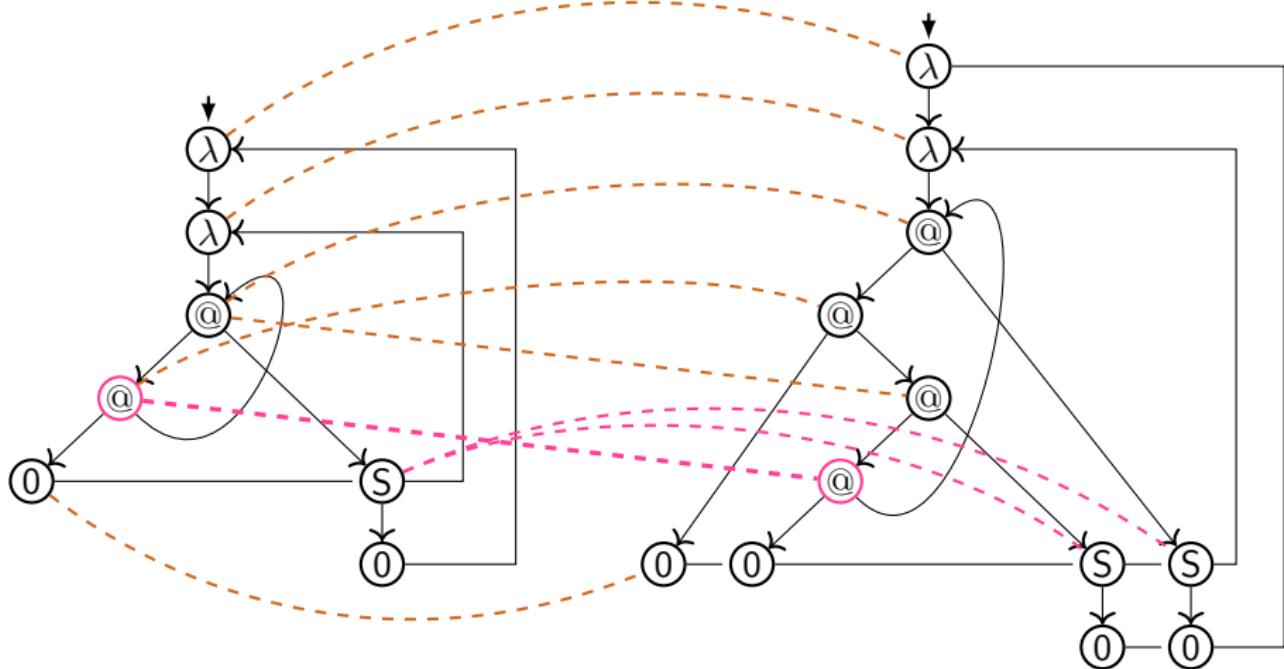
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

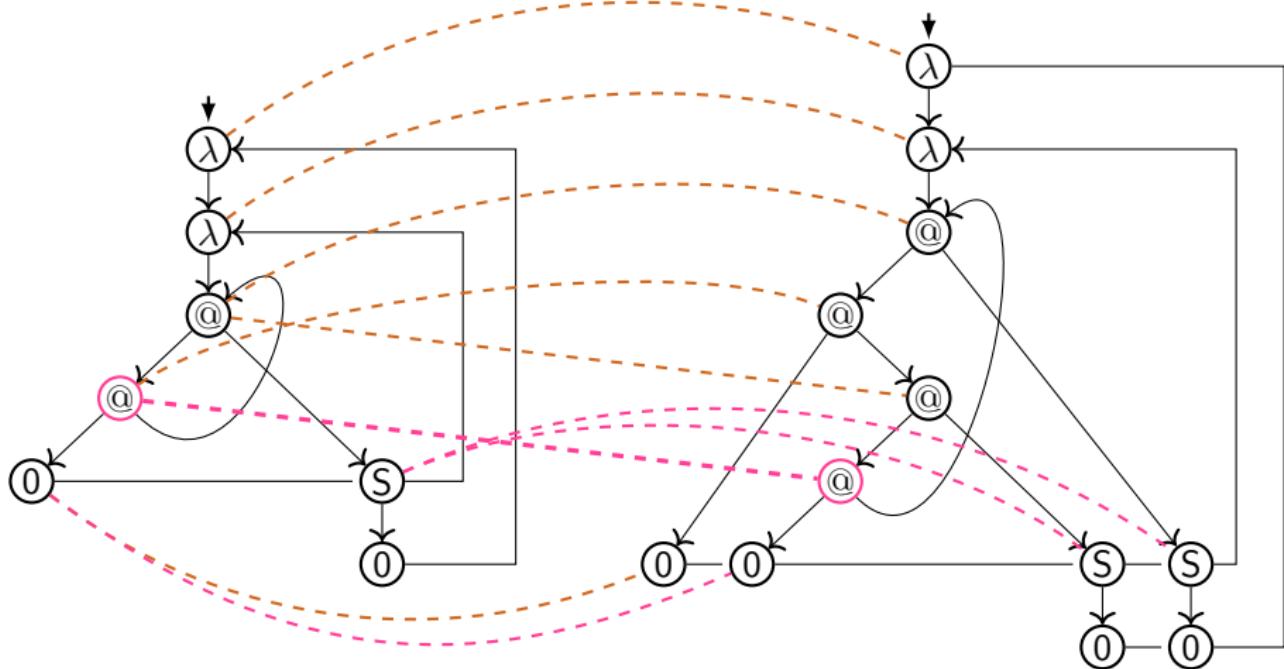
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

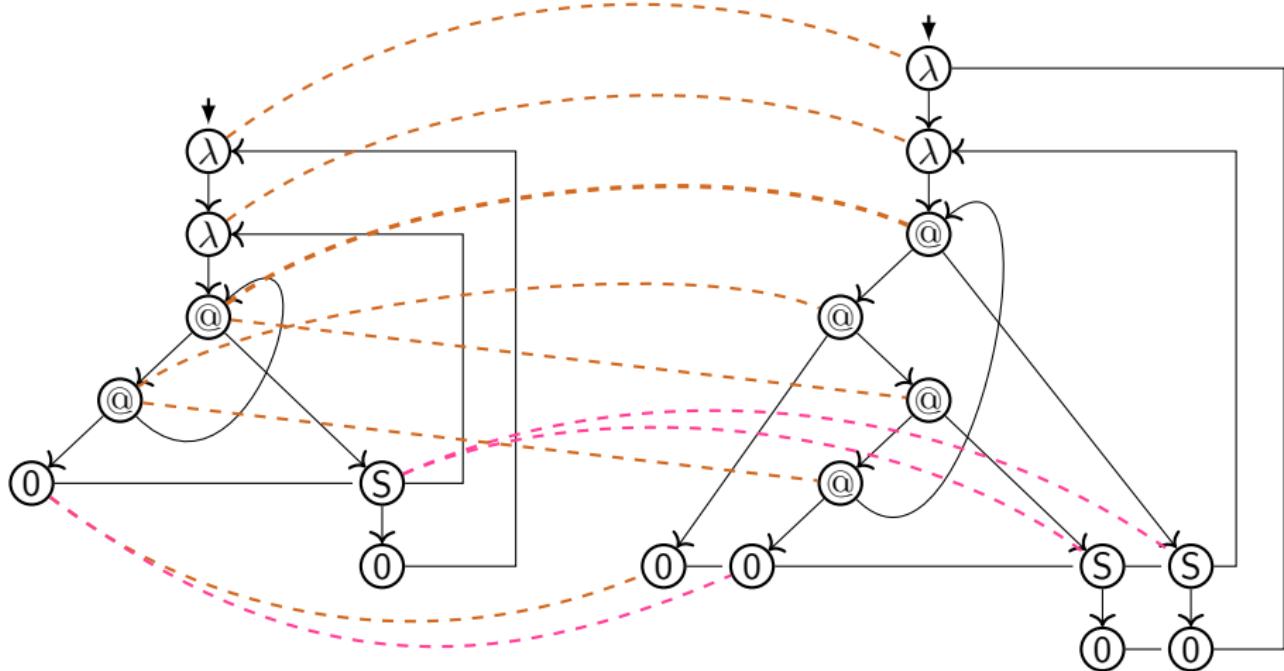
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

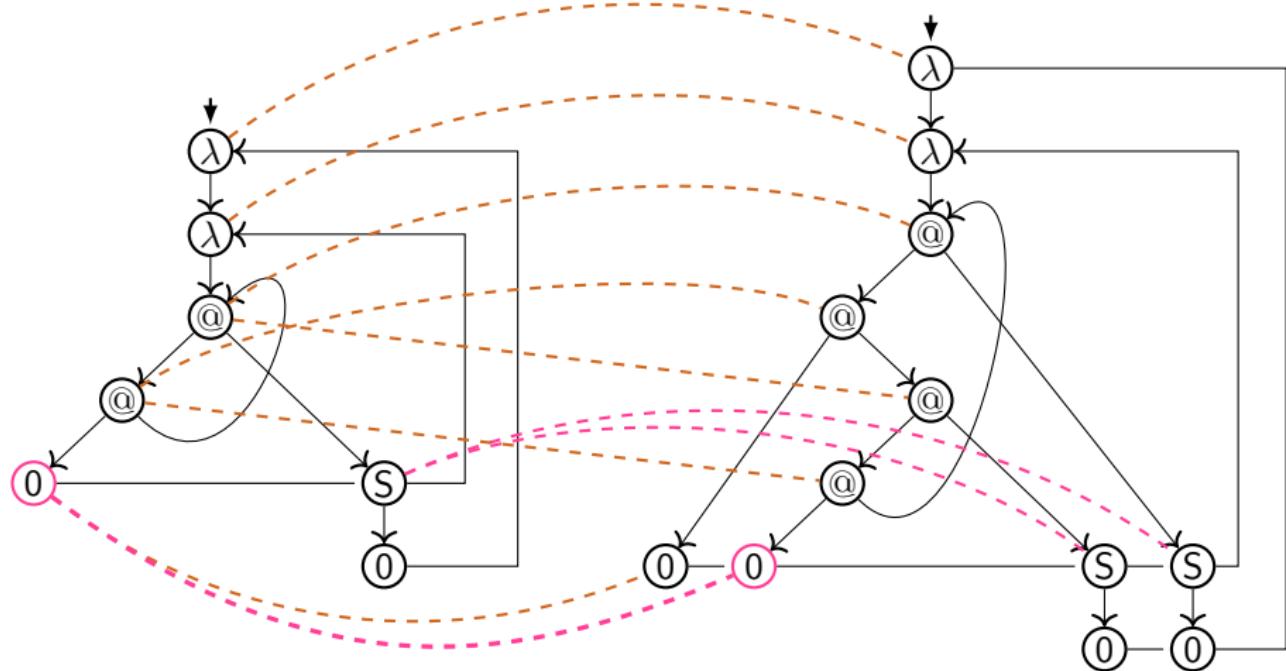
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

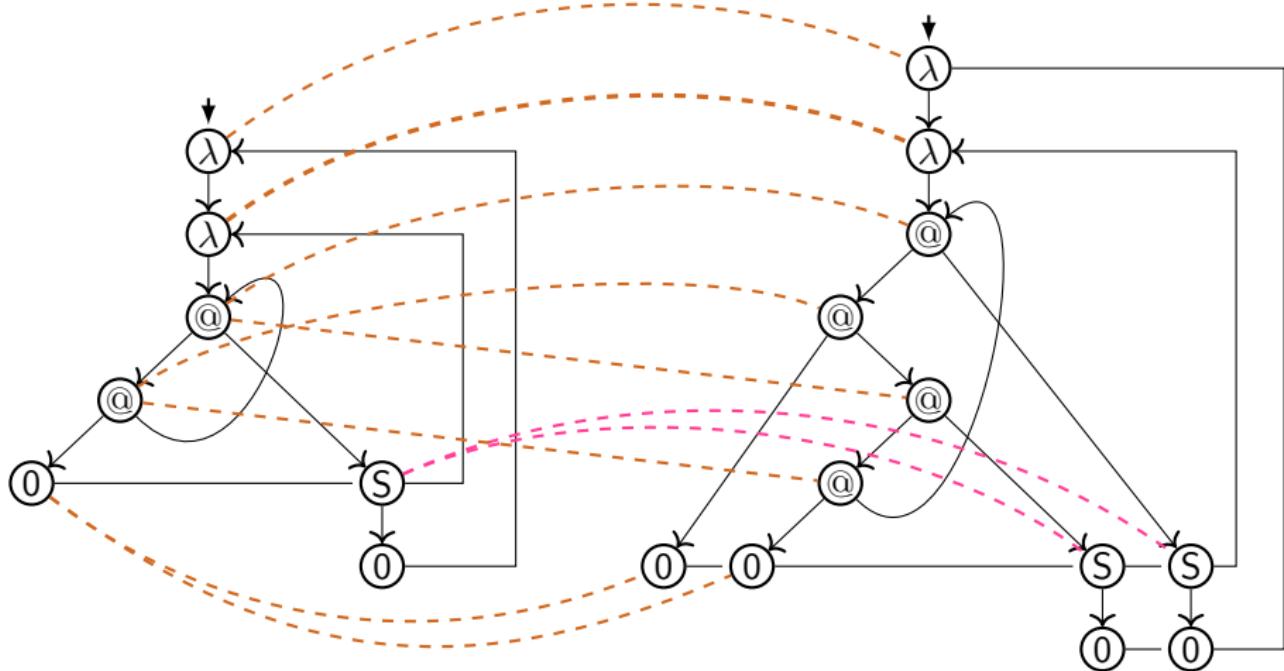
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

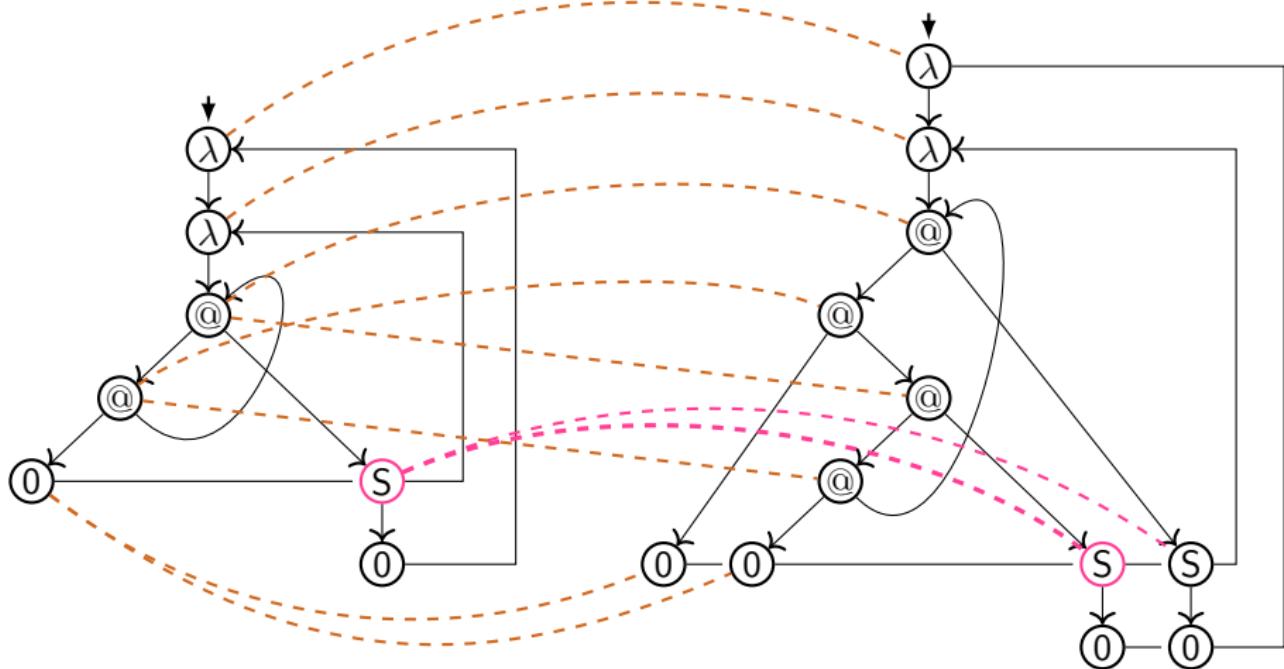
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

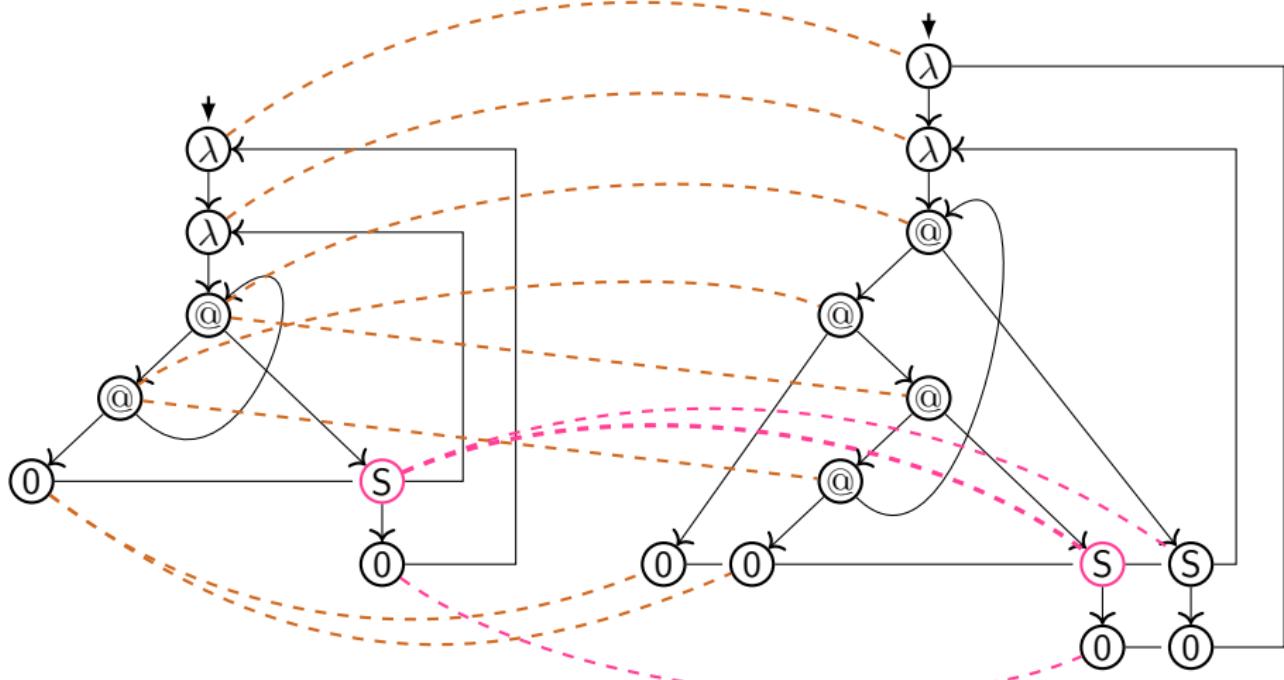
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

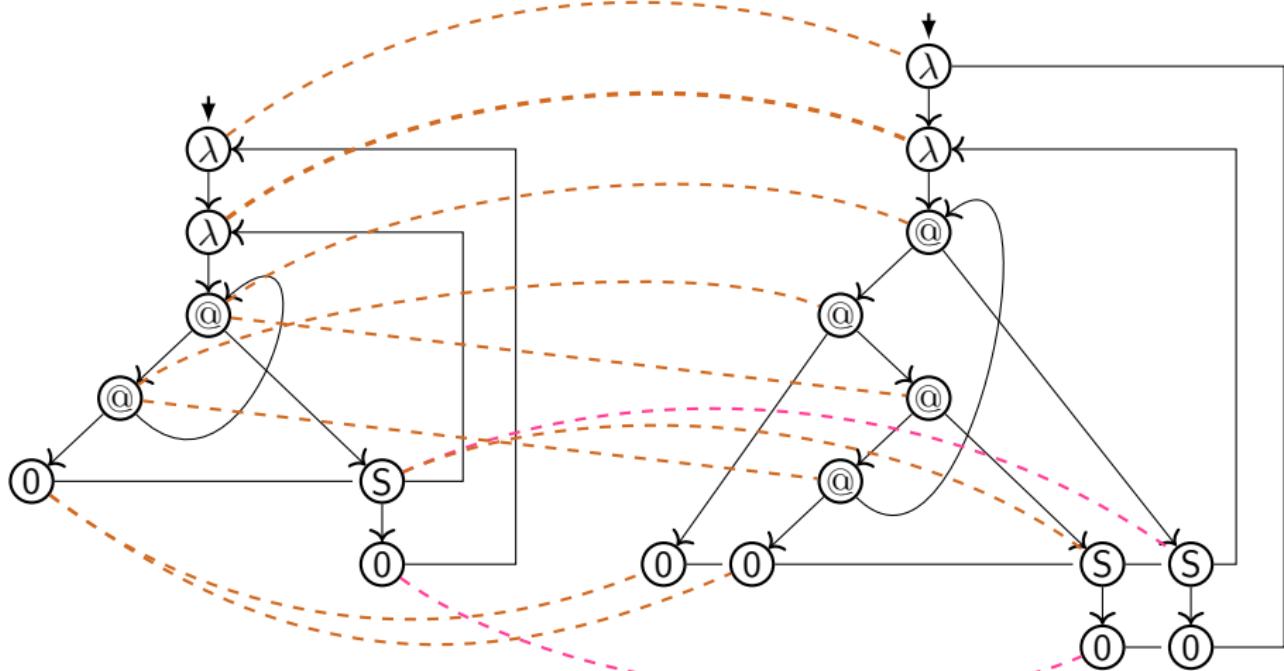
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

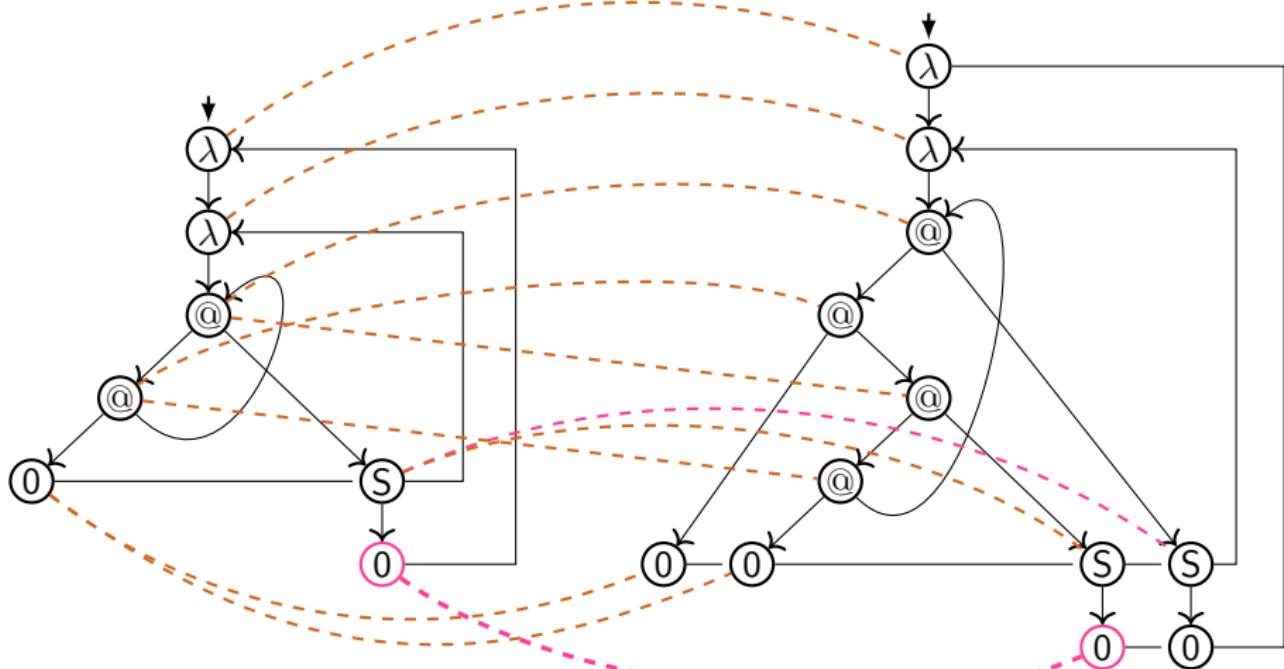
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

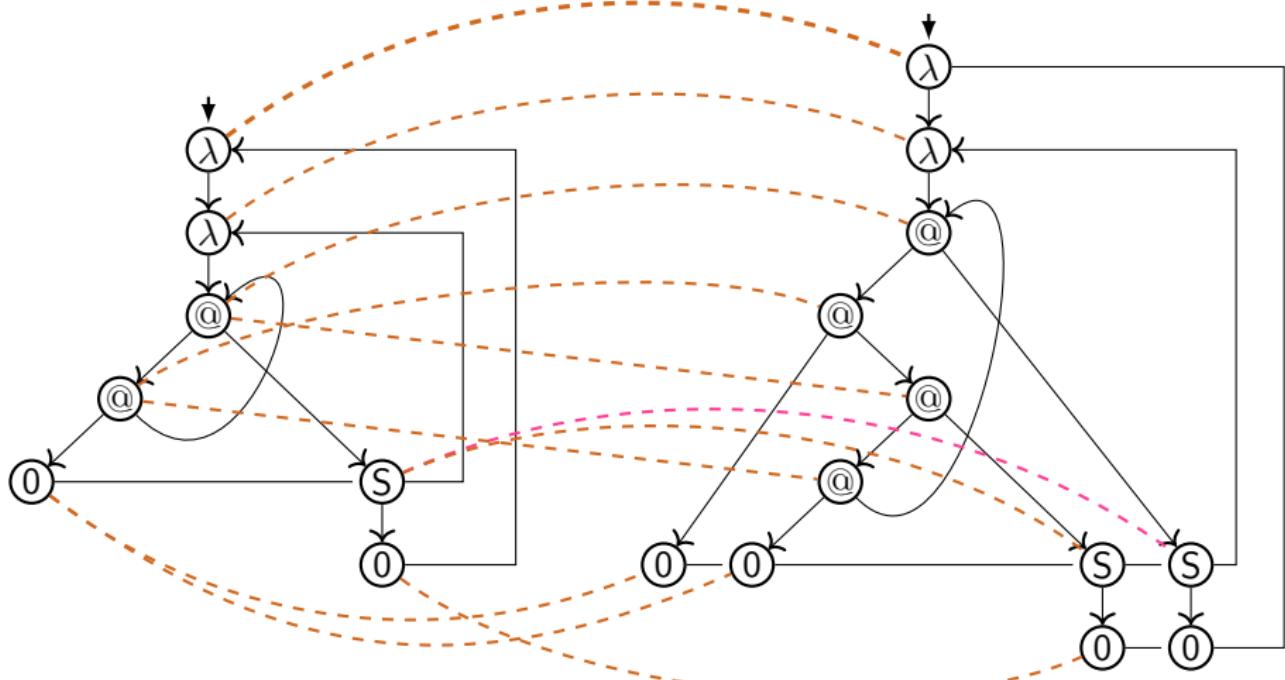
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

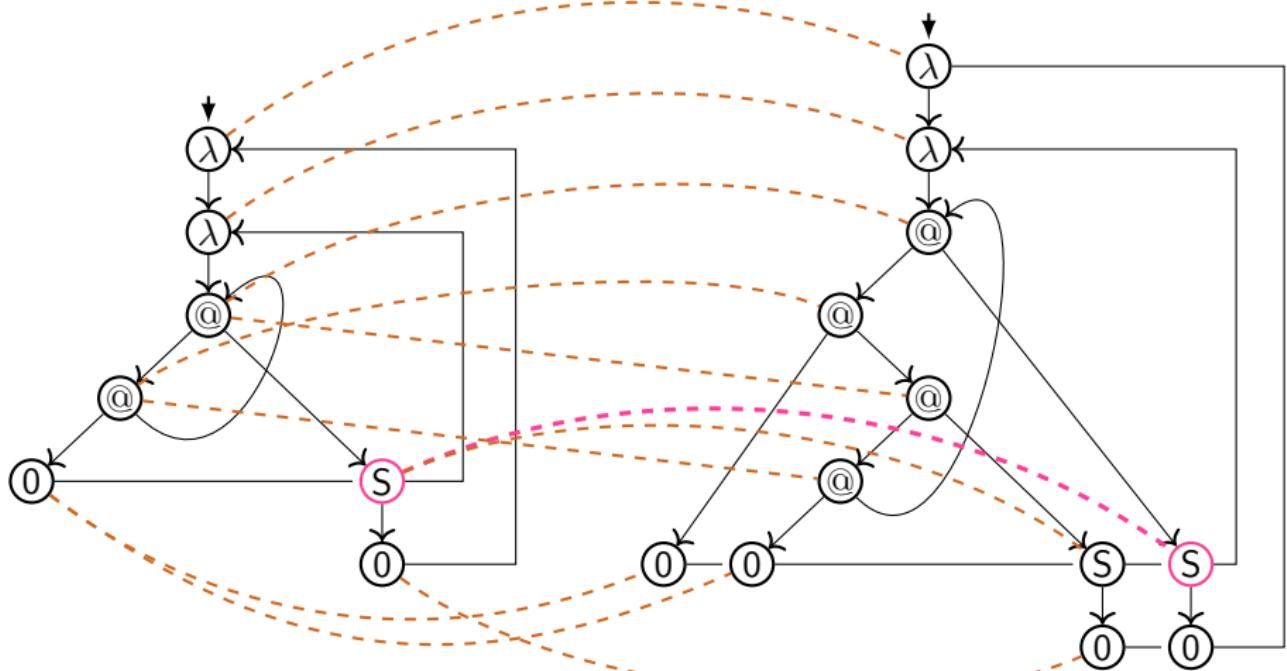
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

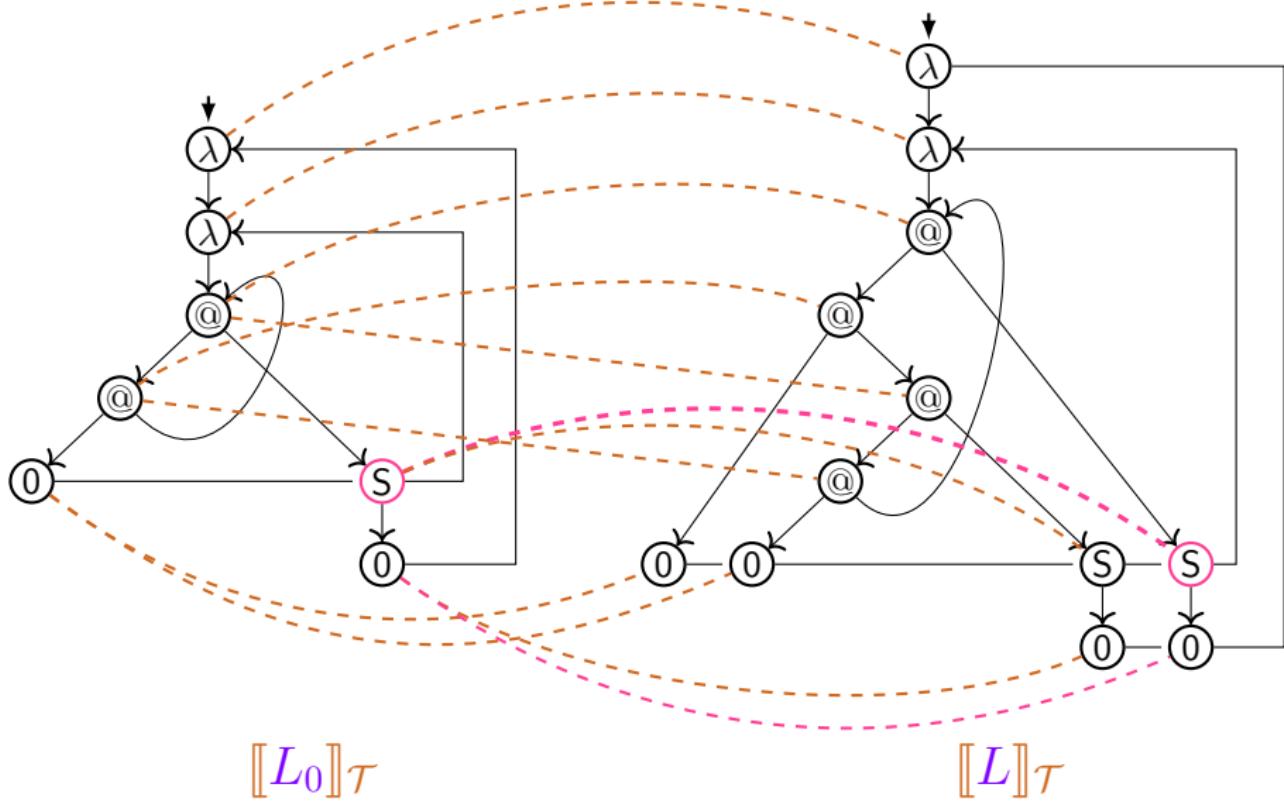
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_\tau$ 
 $\llbracket L \rrbracket_\tau$

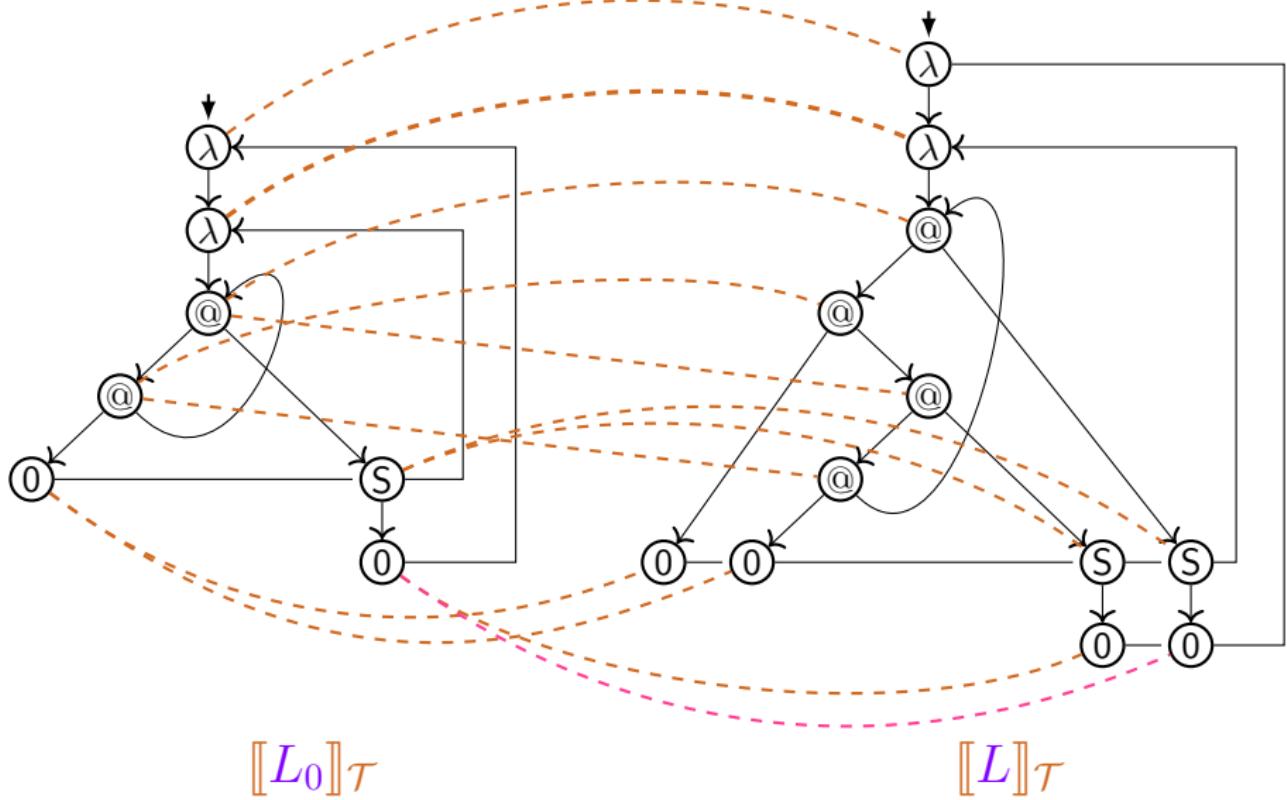
# Bisimulation check between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

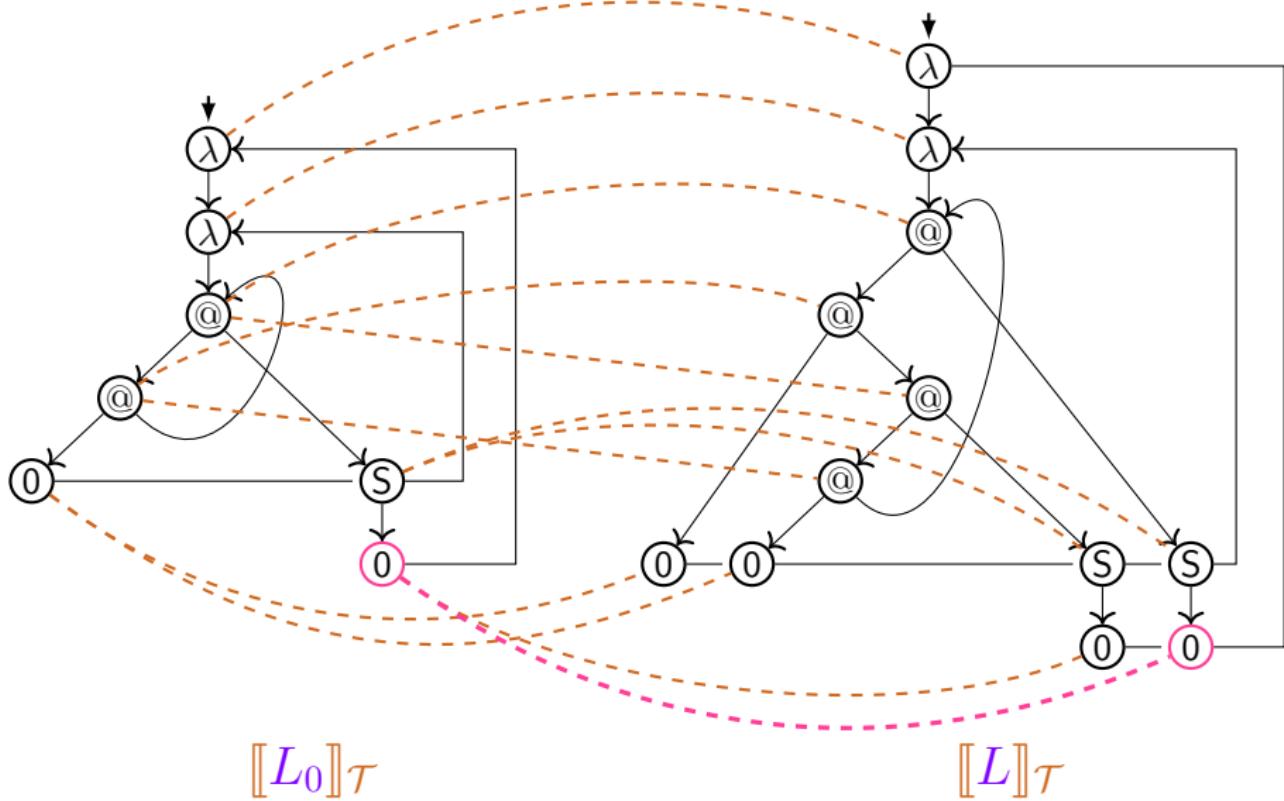
# Bisimulation check between $\lambda$ -term-graphs



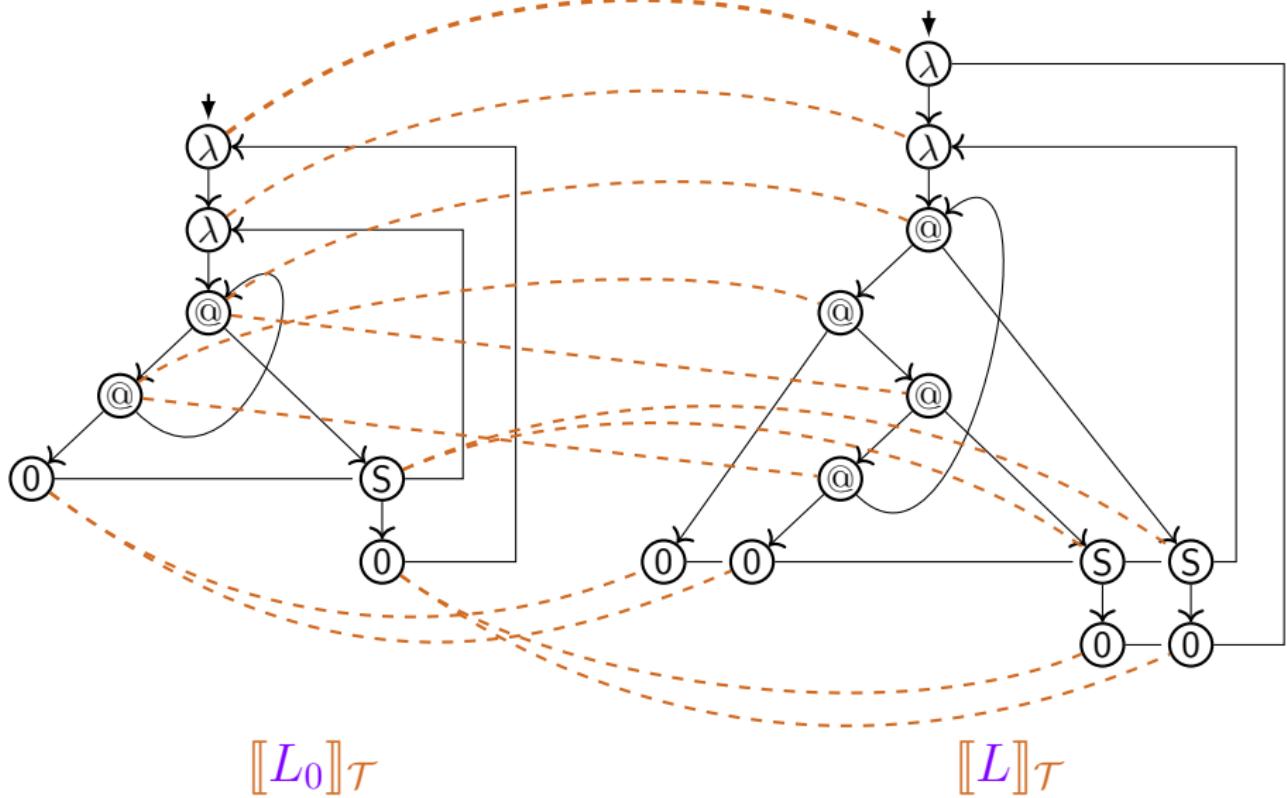
# Bisimulation check between $\lambda$ -term-graphs



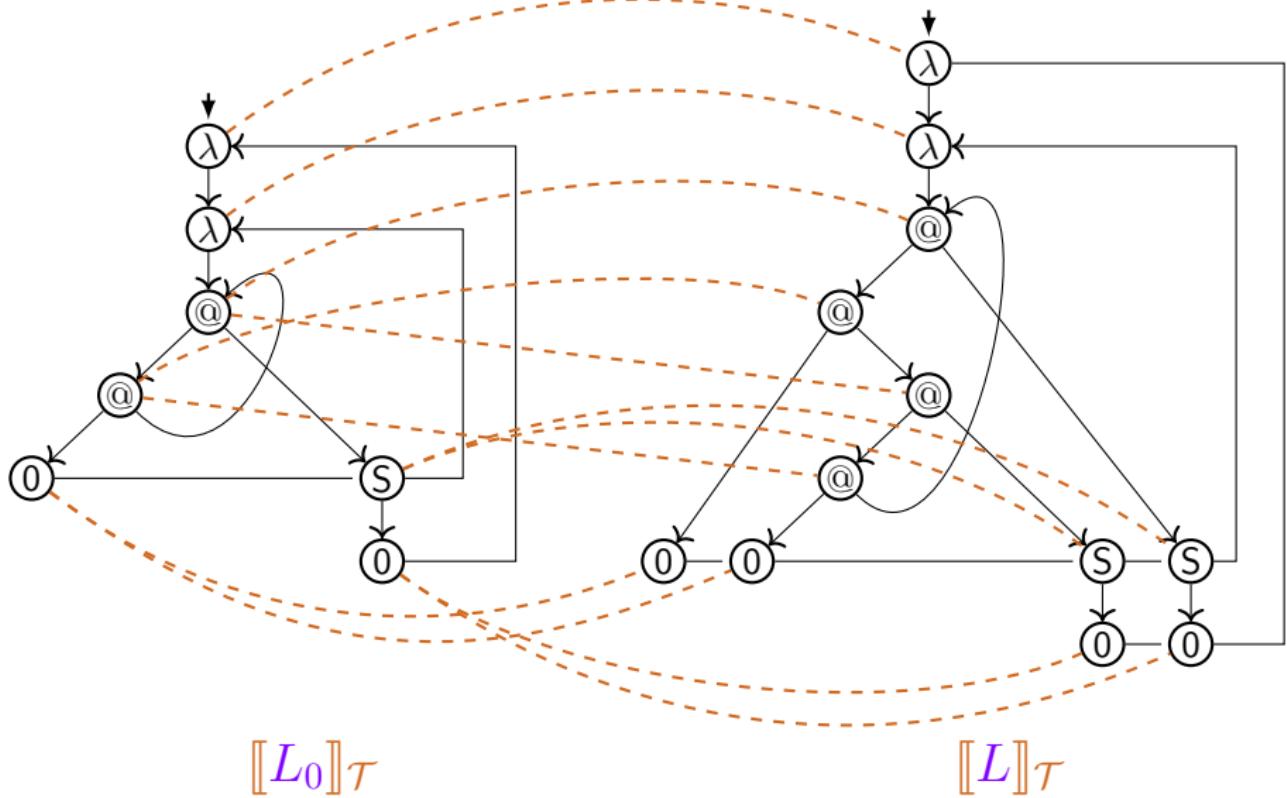
# Bisimulation check between $\lambda$ -term-graphs



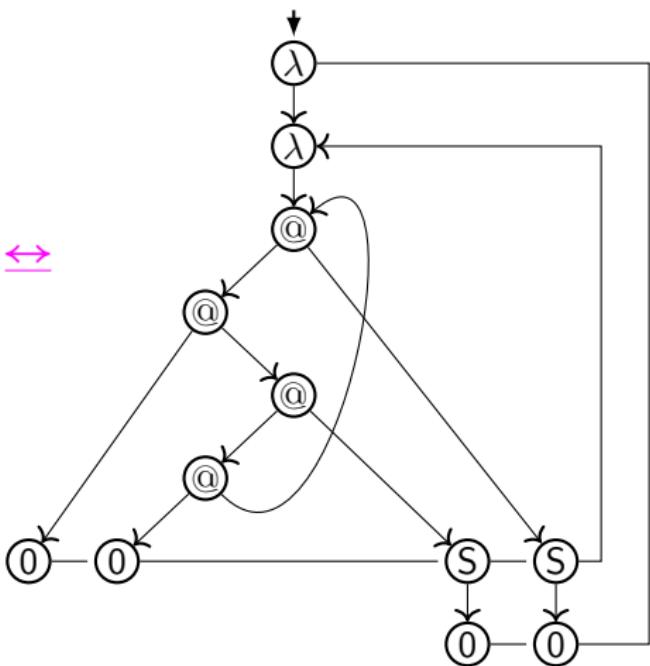
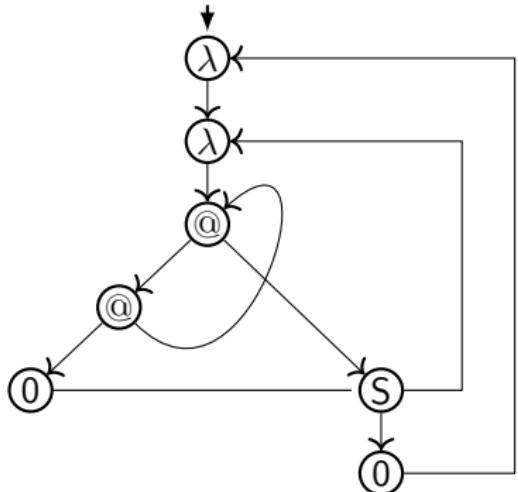
# Bisimulation check between $\lambda$ -term-graphs



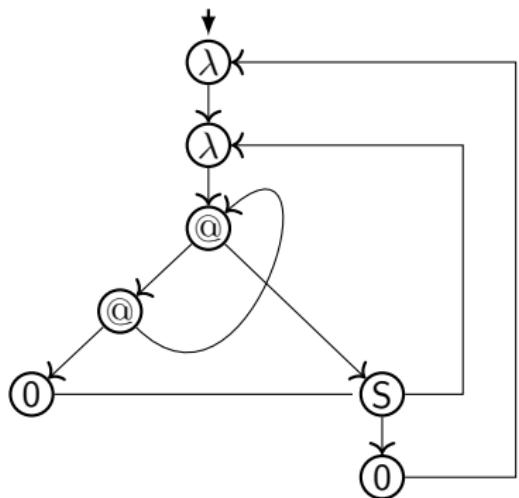
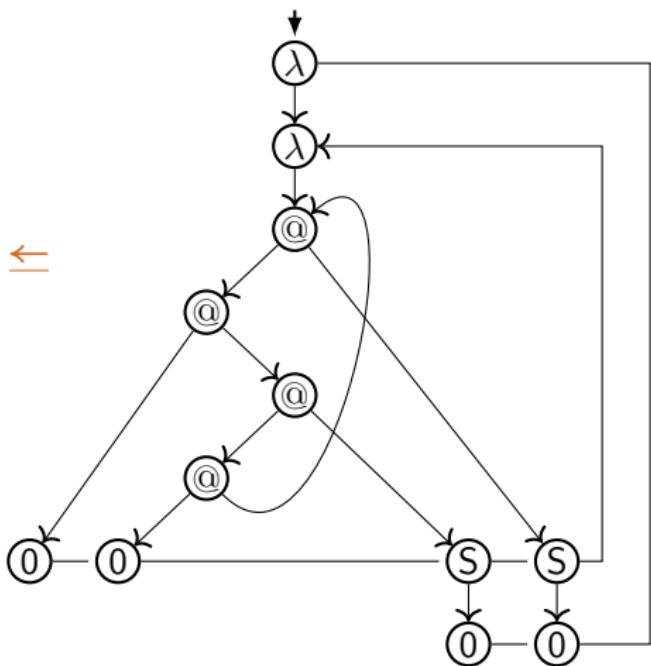
# bisimulation between $\lambda$ -term-graphs



# bisimilarity between $\lambda$ -term-graphs


 $\llbracket L_0 \rrbracket_{\mathcal{T}}$ 
 $\Leftrightarrow$ 
 $\llbracket L \rrbracket_{\mathcal{T}}$

# functional bisimilarity and bisimulation collapse


 $\llbracket L_0 \rrbracket_T$ 
 $\Leftarrow$ 

 $\llbracket L \rrbracket_T$

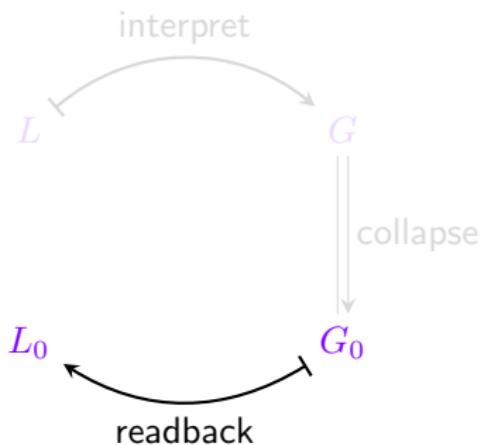
# Bisimulation collapse: property

## Theorem

*The class of eager-scope  $\lambda$ -term-graphs  
is closed under functional bisimilarity  $\Xi$ .*

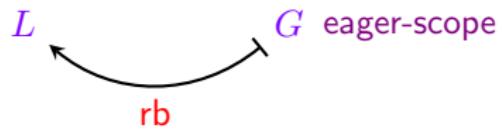
→ For a  $\lambda_{\text{letrec}}$ -term  $L$   
the bisimulation collapse of  $\llbracket L \rrbracket_{\mathcal{T}}$  is again an eager-scope  $\lambda$ -term-graph.

# Readback



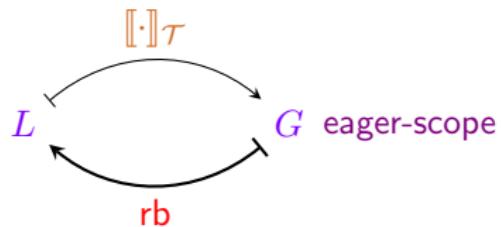
# Readback

defined with property:



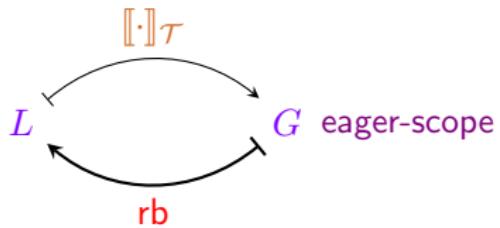
# Readback

defined with property:



# Readback

defined with property:



## Theorem

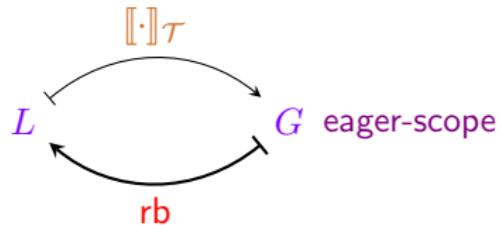
For all eager-scope  $\lambda$ -term-graphs  $G$ :

$$([::]\tau \circ rb)(G) \simeq G$$

The readback  $rb$  is a right-inverse of  $[::]\tau$  modulo isomorphism  $\simeq$ .

# Readback

defined with property:



## Theorem

For all eager-scope  $\lambda$ -term-graphs  $G$ :

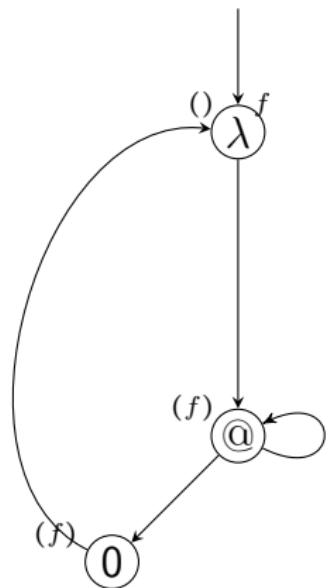
$$([·]\tau \circ rb)(G) \simeq G$$

The readback  $rb$  is a right-inverse of  $[·]\tau$  modulo isomorphism  $\simeq$ .

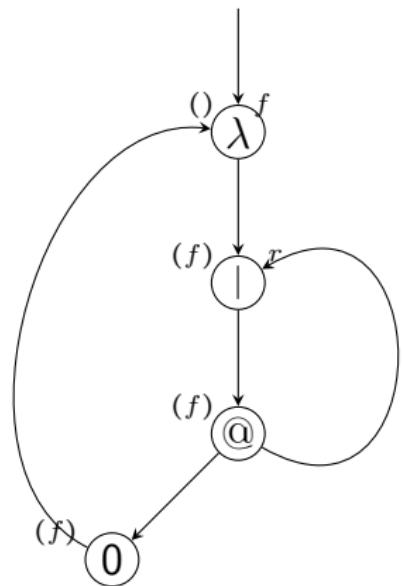
idea:

1. construct a spanning tree  $T$  of  $G$
2. using local rules, in a bottom-up traversal of  $T$  synthesize  $L = rb(G)$

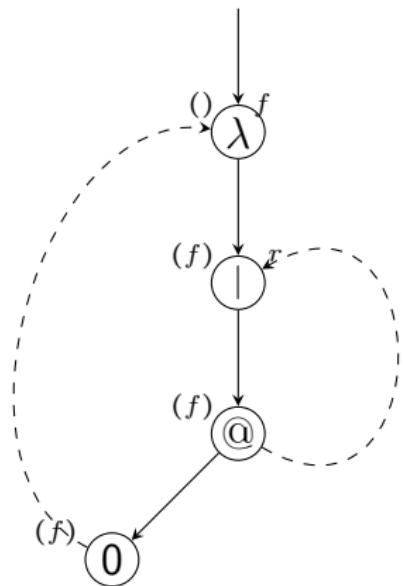
# Readback: example (fix)



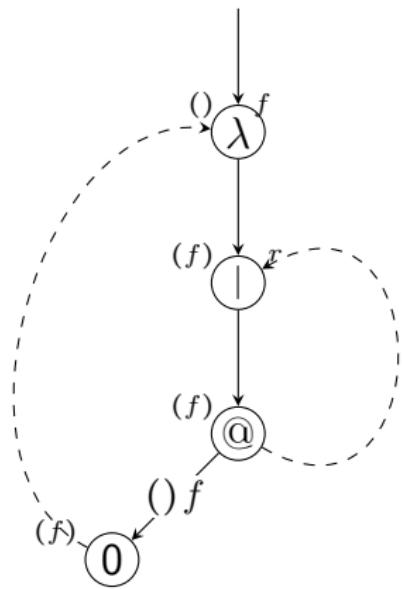
# Readback: example (fix)



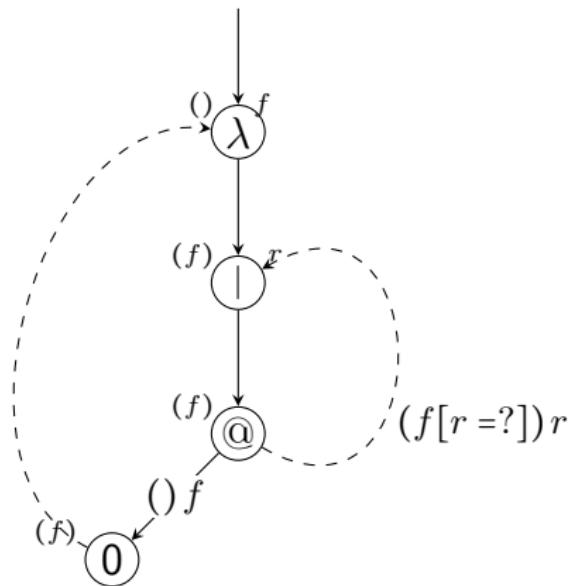
## Readback: example (fix)



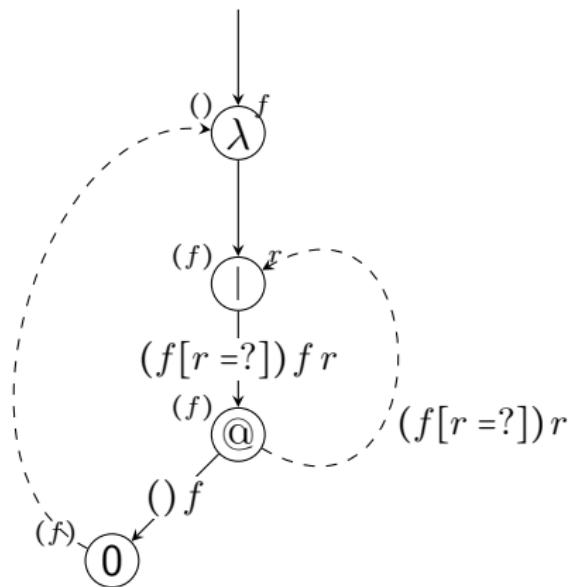
# Readback: example (fix)



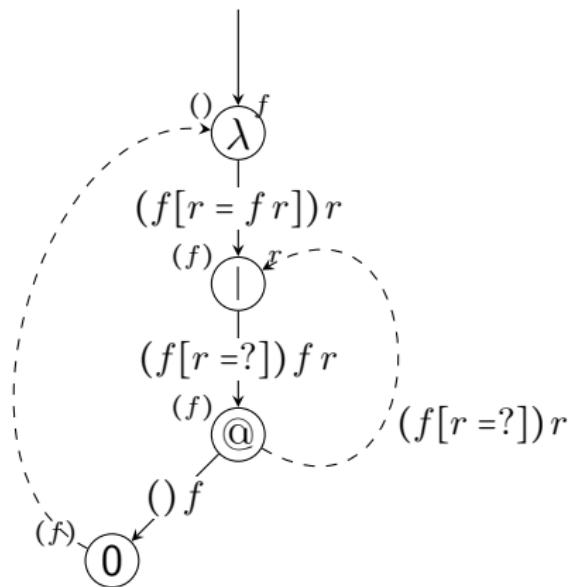
# Readback: example (fix)



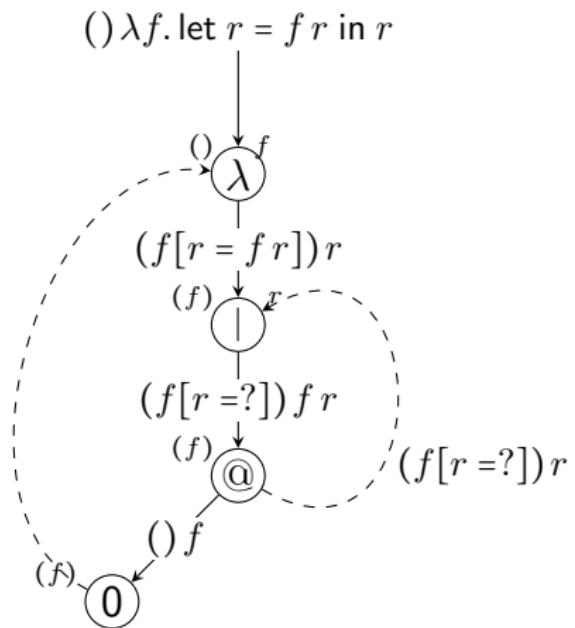
# Readback: example (fix)



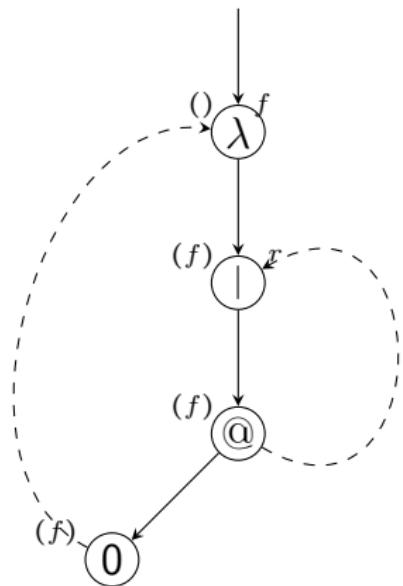
# Readback: example (fix)



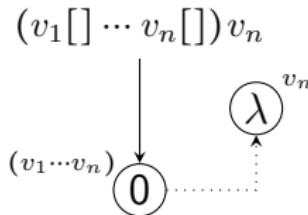
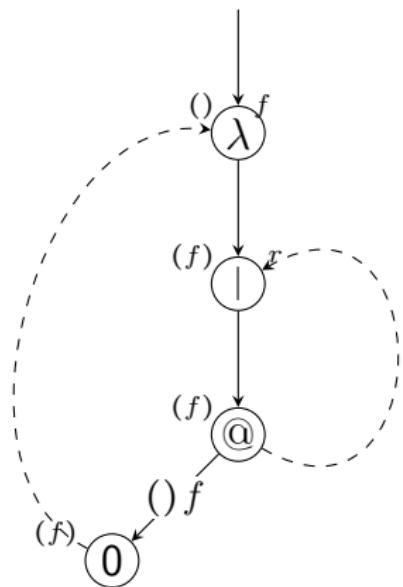
# Readback: example (fix)



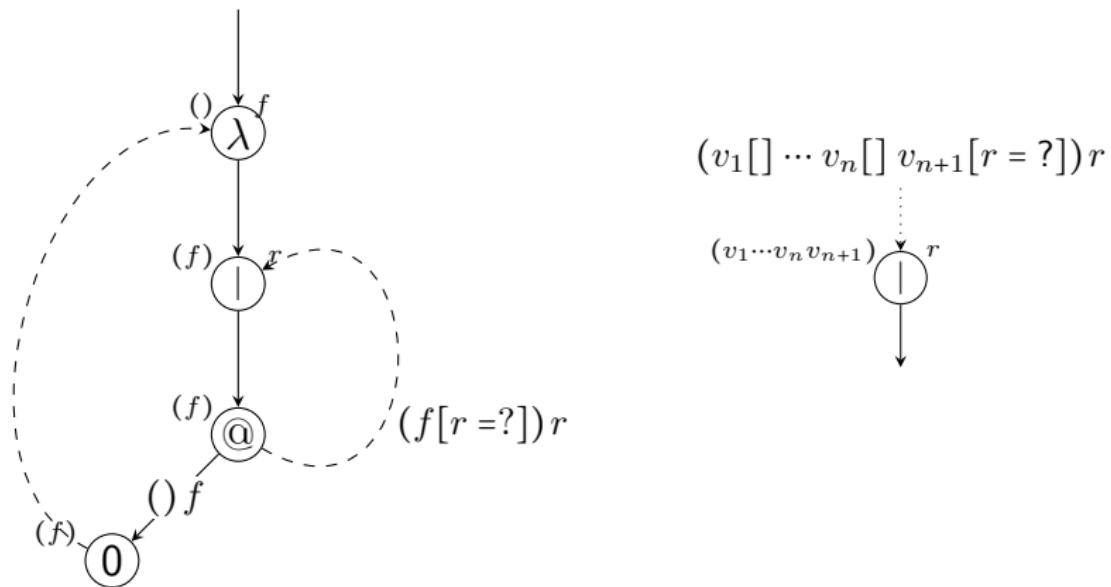
# Readback: example (fix)



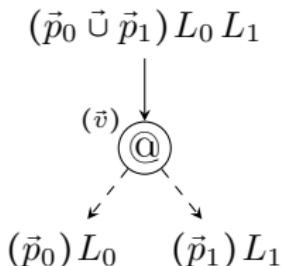
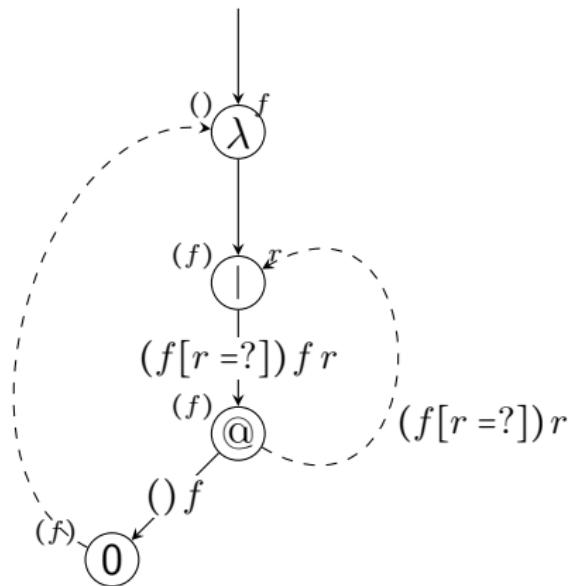
# Readback: example (fix)



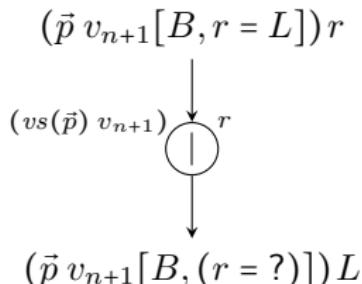
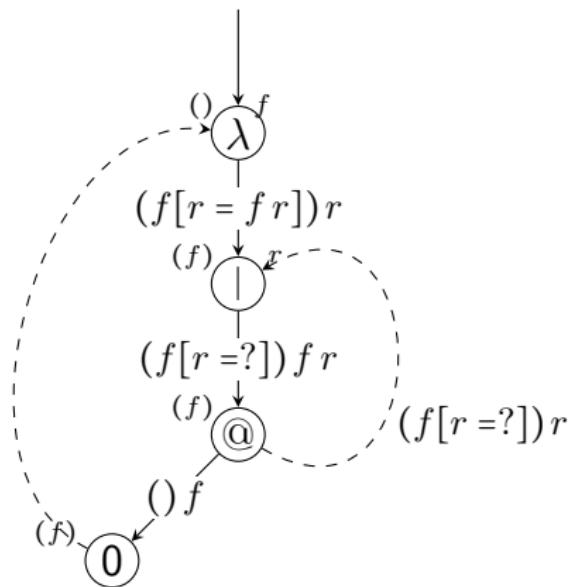
# Readback: example (fix)



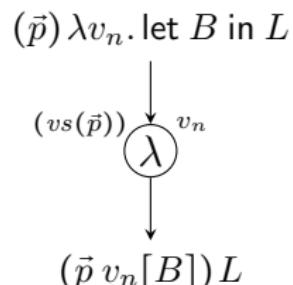
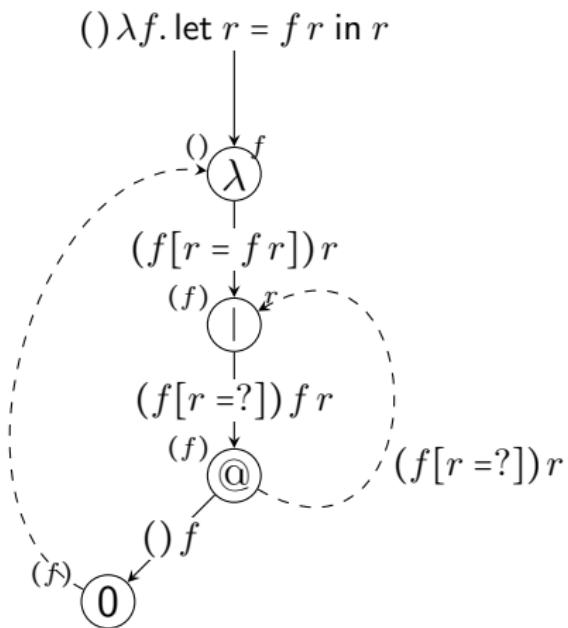
# Readback: example (fix)



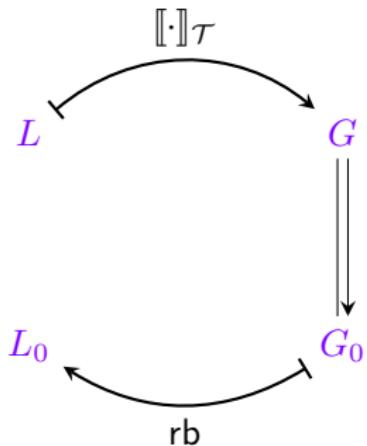
# Readback: example (fix)



# Readback: example (fix)



# Maximal sharing: complexity



## 1. interpretation

of  $\lambda_{\text{letrec}}$ -term  $L$  with  $|L| = n$

as  $\lambda$ -term-graph  $G = \llbracket L \rrbracket \tau$

- ▶ in time  $O(n^2)$ , size  $|G| \in O(n^2)$ .

## 2. bisimulation collapse ↓

of f-o term graph  $G$  into  $G_0$

- ▶ in time  $O(|G| \log |G|) = O(n^2 \log n)$

## 3. readback rb

of f-o term graph  $G_0$

yielding  $\lambda_{\text{letrec}}$ -term  $L_0 = \text{rb}(G_0)$ .

- ▶ in time  $O(|G| \log |G|) = O(n^2 \log n)$

## Theorem

Computing a maximally compact form  $L_0 = (\text{rb} \circ \downarrow \circ \llbracket \cdot \rrbracket \tau)(L)$  of  $L$  for a  $\lambda_{\text{letrec}}$ -term  $L$  requires time  $O(n^2 \log n)$ , where  $|L| = n$ .

# Demo: console output

jan:~/papers/maxsharing-ICFP/talks/ICFP-2014> maxsharing running.l

$\lambda\text{-letrec-term}$ :

$\lambda x. \lambda f. \text{ let } r = f (f\ r\ x) \ x \text{ in } r$

derivation:

```

----- 0          ----- 0
(x f[r]) f      (x f[r]) r      (x) x
----- @ ----- S
(x f[r]) f r    (x f[r]) x
----- 0          ----- 0
(x f[r]) f      (x f[r]) f r x
----- @ ----- S
(x f[r]) f (f r x)      (x f[r]) x
----- @ ----- S
(x f[r]) f (f r x) x      (x f[r]) r
----- @ ----- let
(x f) let r = f (f r x) x in r
----- @ ----- λ
(x) λf. let r = f (f r x) x in r
----- @ ----- λ
() λx. λf. let r = f (f r x) x in r

```

writing DFA to file: running-dfa.pdf

readback of DFA:

$\lambda x. \lambda y. \text{ let } F = y (y\ F\ x) \ x \text{ in } F$

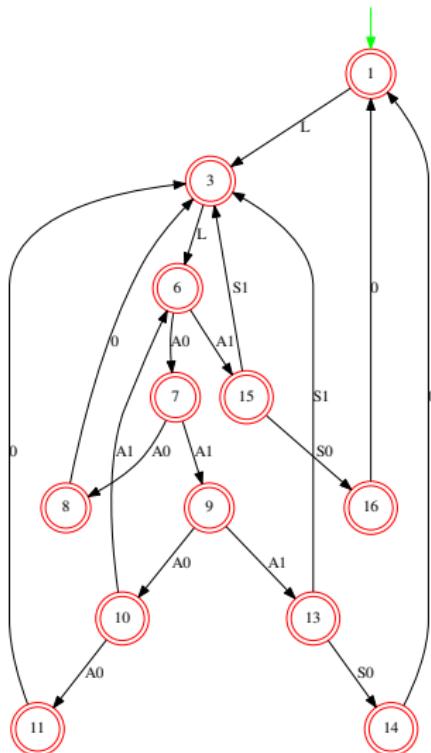
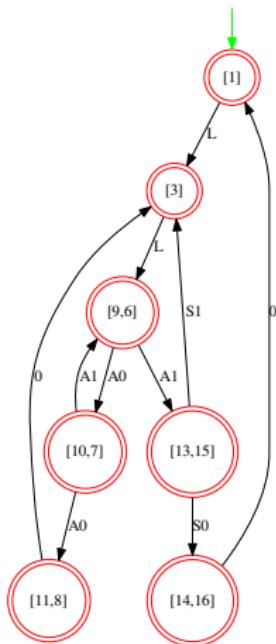
writing minimised DFA to file: running-mindfa.pdf

readback of minimised DFA:

$\lambda x. \lambda y. \text{ let } F = y\ F\ x \text{ in } F$

jan:~/papers/maxsharing-ICFP/talks/ICFP-2014> █

# Demo: generated $\lambda$ -NFAs

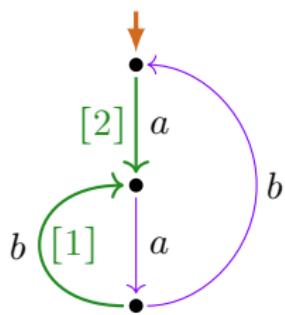


# Resources (maximal sharing)

- ▶ tool **maxsharing** on [hackage.haskell.org](https://hackage.haskell.org)
- ▶ papers and reports
  - ▶ Maximal Sharing in the Lambda Calculus with Letrec
    - ▶ ICFP 2014 paper
    - ▶ accompanying report [arXiv:1401.1460](https://arxiv.org/abs/1401.1460)
  - ▶ Term Graph Representations for Cyclic Lambda Terms
    - ▶ TERMGRAPH 2013 proceedings
    - ▶ extended report [arXiv:1308.1034](https://arxiv.org/abs/1308.1034)
  - ▶ Vincent van Oostrom, CG: Nested Term Graphs
    - ▶ TERMGRAPH 2014 post-proceedings in [EPTCS 183](https://eptcs.net/eptcs/183)
- ▶ thesis Jan Rochel
  - ▶ Unfolding Semantics of the Untyped  $\lambda$ -Calculus with letrec
    - ▶ Ph.D. Thesis, Utrecht University, 2016

# Process interpretation of regular expressions

(based on joint work with Wan Fokkink)



# Regular expressions *(S.C. Kleene, 1951)*

## Definition

The set  $\text{Reg}(A)$  of **regular expressions** over alphabet  $A$  is defined by the grammar:

$$e, f ::= 0 \mid 1 \mid a \mid (e + f) \mid (e \cdot f) \mid (e^*) \quad (\text{for } a \in A).$$

# Regular expressions (S.C. Kleene, 1951)

## Definition

The set  $\text{Reg}(A)$  of **regular expressions** over alphabet  $A$  is defined by the grammar:

$$e, f ::= 0 \mid 1 \mid a \mid (e + f) \mid (e \cdot f) \mid (e^*) \quad (\text{for } a \in A).$$

Note, here:

- ▶ symbol 0 instead of  $\emptyset$
- ▶ symbol 1 used (often dropped, definable as  $0^*$ )
- ▶ no complementation operation  $\bar{e}$ 
  - ▶ which is not expressible under language interpretation

# Language semantics $\llbracket \cdot \rrbracket_L$ of reg. expr's (Copi–Elgot–Wright, 1958)

**0**  $\xrightarrow{L}$  empty language  $\emptyset$

**1**  $\xrightarrow{L}$   $\{\epsilon\}$  ( $\epsilon$  the empty word)

**a**  $\xrightarrow{L}$   $\{a\}$

# Language semantics $\llbracket \cdot \rrbracket_L$ of reg. expr's (Copi–Elgot–Wright, 1958)

**0**  $\xrightarrow{L}$  empty language  $\emptyset$

**1**  $\xrightarrow{L}$   $\{\epsilon\}$  ( $\epsilon$  the empty word)

**a**  $\xrightarrow{L}$   $\{a\}$

$e + f$   $\xrightarrow{L}$  union of  $L(e)$  and  $L(f)$

$e \cdot f$   $\xrightarrow{L}$  element-wise concatenation of  $L(e)$  and  $L(f)$

$e^*$   $\xrightarrow{L}$  set of words formed by concatenating words in  $L(e)$ ,  
and adding the empty word  $\epsilon$

# Language semantics $\llbracket \cdot \rrbracket_L$ of reg. expr's (Copi–Elgot–Wright, 1958)

**0**  $\xrightarrow{L}$  empty language  $\emptyset$

**1**  $\xrightarrow{L}$   $\{\epsilon\}$  ( $\epsilon$  the empty word)

**a**  $\xrightarrow{L}$   $\{a\}$

$e + f$   $\xrightarrow{L}$  union of  $L(e)$  and  $L(f)$

$e \cdot f$   $\xrightarrow{L}$  element-wise concatenation of  $L(e)$  and  $L(f)$

$e^*$   $\xrightarrow{L}$  set of words formed by concatenating words in  $L(e)$ ,  
and adding the empty word  $\epsilon$

$\llbracket e \rrbracket_L := L(e)$  (language defined by  $e$ )

# Process semantics of regular expressions $\llbracket \cdot \rrbracket_P$ (Milner, 1984)

0  $\xrightarrow{P}$  deadlock  $\delta$ , no termination

1  $\xrightarrow{P}$  empty-step process  $\epsilon$ , then terminate

$a$   $\xrightarrow{P}$  atomic action  $a$ , then terminate

# Process semantics of regular expressions $\llbracket \cdot \rrbracket_P$ (Milner, 1984)

$0 \xrightarrow{P}$  deadlock  $\delta$ , no termination

$1 \xrightarrow{P}$  empty-step process  $\epsilon$ , then terminate

$a \xrightarrow{P}$  atomic action  $a$ , then terminate

$e + f \xrightarrow{P}$  (choice) execute  $\llbracket e \rrbracket_P$  or  $\llbracket f \rrbracket_P$

$e \cdot f \xrightarrow{P}$  (sequentialization) execute  $\llbracket e \rrbracket_P$ , then  $\llbracket f \rrbracket_P$

$e^* \xrightarrow{P}$  (iteration) repeat (terminate or execute  $\llbracket e \rrbracket_P$ )

# Process semantics of regular expressions $\llbracket \cdot \rrbracket_P$ (Milner, 1984)

$0 \xrightarrow{P}$  deadlock  $\delta$ , no termination

$1 \xrightarrow{P}$  empty-step process  $\epsilon$ , then terminate

$a \xrightarrow{P}$  atomic action  $a$ , then terminate

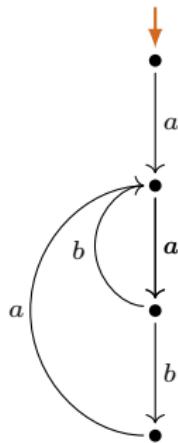
$e + f \xrightarrow{P}$  (choice) execute  $\llbracket e \rrbracket_P$  or  $\llbracket f \rrbracket_P$

$e \cdot f \xrightarrow{P}$  (sequentialization) execute  $\llbracket e \rrbracket_P$ , then  $\llbracket f \rrbracket_P$

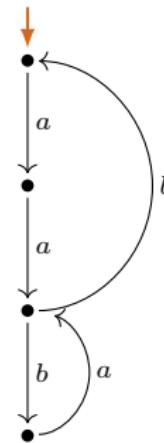
$e^* \xrightarrow{P}$  (iteration) repeat (terminate or execute  $\llbracket e \rrbracket_P$ )

$\llbracket e \rrbracket_P := [P(e)]_{\Leftarrow}$  (bisimilarity equivalence class of process  $P(e)$ )

# Process interpretation of regular expressions

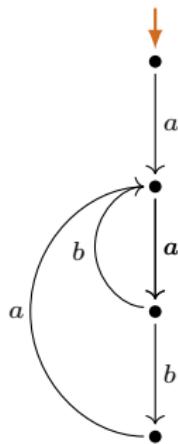


$P(a(a(b+ba))^*0)$

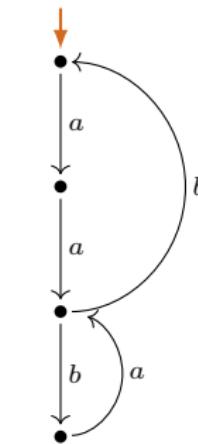


$P((aa(ba))^*b)^*0)$

# Process interpretation of regular expressions

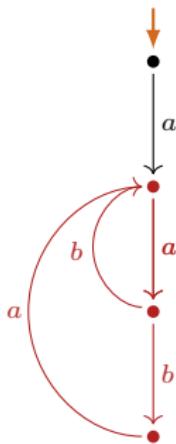
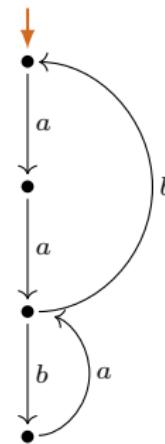


$P(a \cdot (a \cdot (b + b \cdot a))^* \cdot 0)$

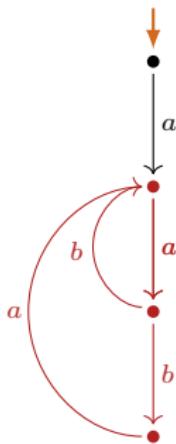
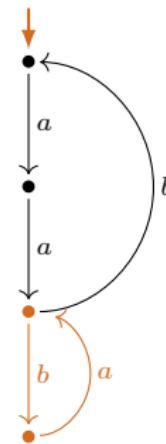


$P((a \cdot a \cdot (b \cdot a))^* \cdot b)^* \cdot 0)$

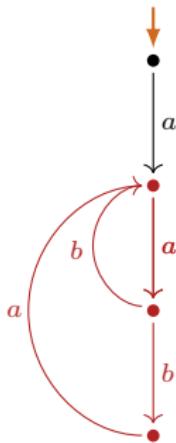
# Process interpretation of regular expressions

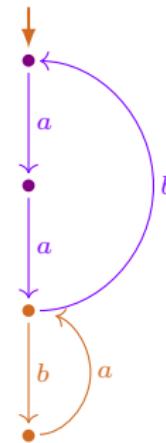

 $P(a \cdot (a \cdot (b + b \cdot a))^* \cdot 0)$ 

 $P((a \cdot a \cdot (b \cdot a))^* \cdot b)^* \cdot 0)$

# Process interpretation of regular expressions


 $P(a \cdot (a \cdot (b + b \cdot a))^* \cdot 0)$ 

 $P((a \cdot a \cdot (b \cdot a))^* \cdot b)^* \cdot 0)$

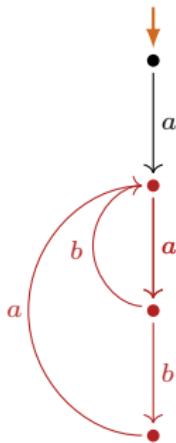
# Process interpretation of regular expressions

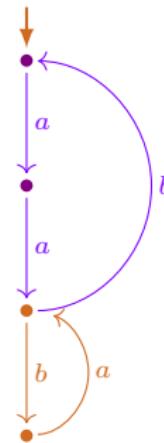


$$P(a \cdot (a \cdot (b + b \cdot a))^* \cdot 0)$$


$$P((a \cdot a \cdot (b \cdot a))^* \cdot b)^* \cdot 0)$$

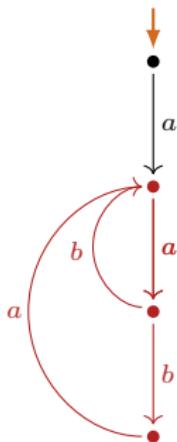
# Process interpretation of regular expressions

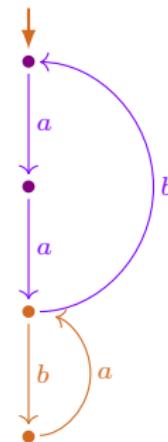


$$P(a \cdot (a \cdot (b + b \cdot a))^* \cdot 0)$$


$$P((a \cdot a \cdot (b \cdot a))^* \cdot b)^* \cdot 0)$$

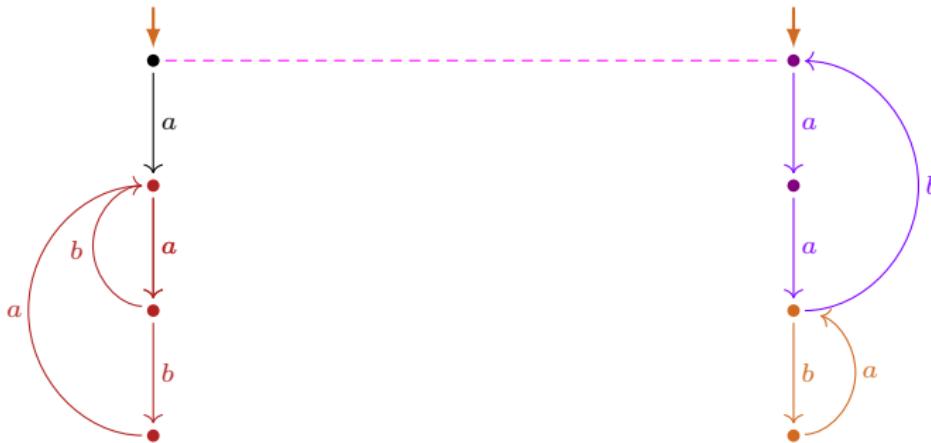
# Process interpretation of regular expressions



$$P(a(a(b+ba))^*0)$$


$$P((aa(ba))^*b)^*0$$

# Process interpretation of regular expressions



$$P(a(a(b+ba))^*0)$$

$$P((aa(ba)^*b)^*0)$$

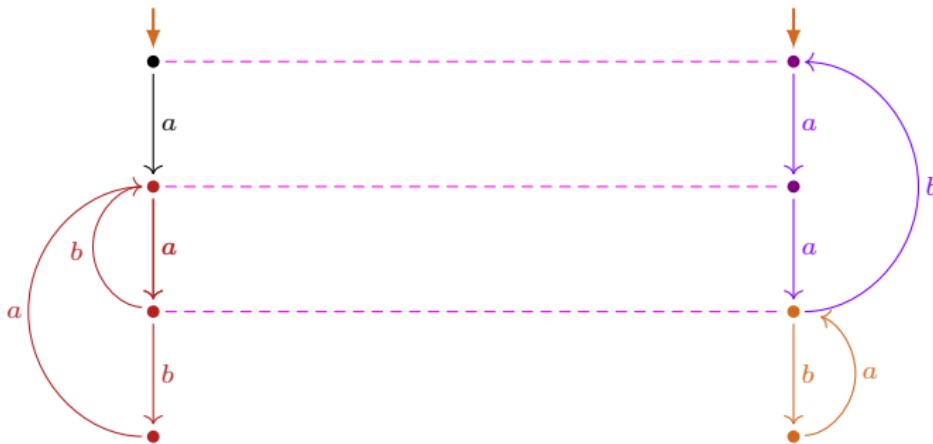
# Process interpretation of regular expressions



$$P(a(a(b+ba))^*0)$$

$$P((aa(ba)^*b)^*0)$$

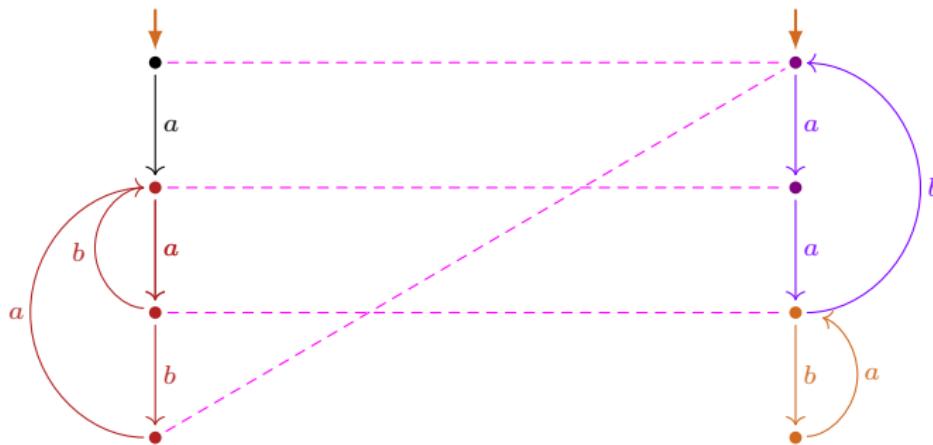
# Process interpretation of regular expressions



$$P(a(a(b+ba))^*0)$$

$$P((aa(ba)^*b)^*0)$$

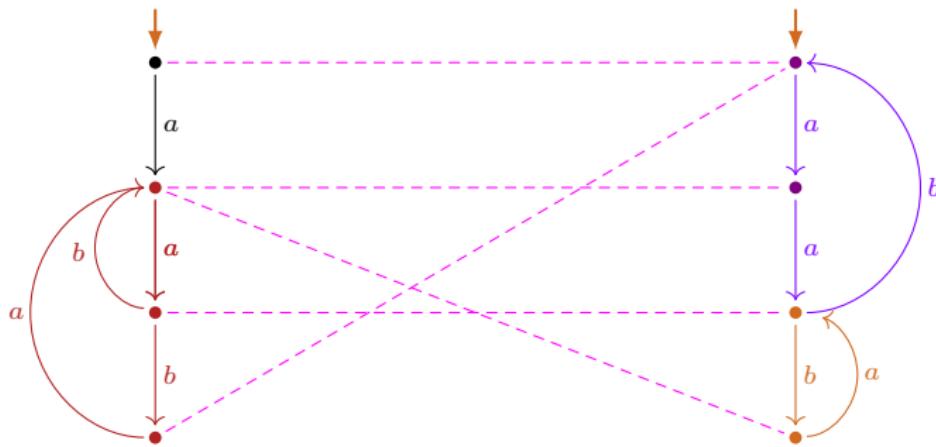
# Process interpretation of regular expressions



$$P(a(a(b+ba))^*0)$$

$$P((aa(ba)^*b)^*0)$$

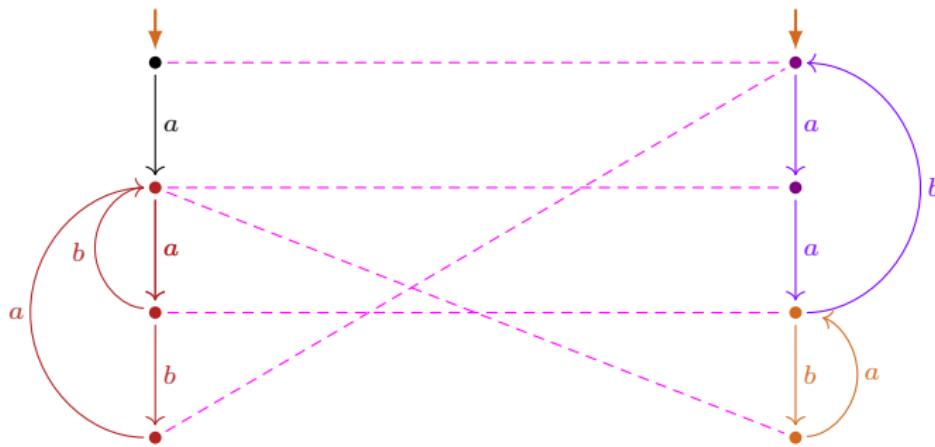
# Process interpretation of regular expressions



$$P(a(a(b+ba))^*0)$$

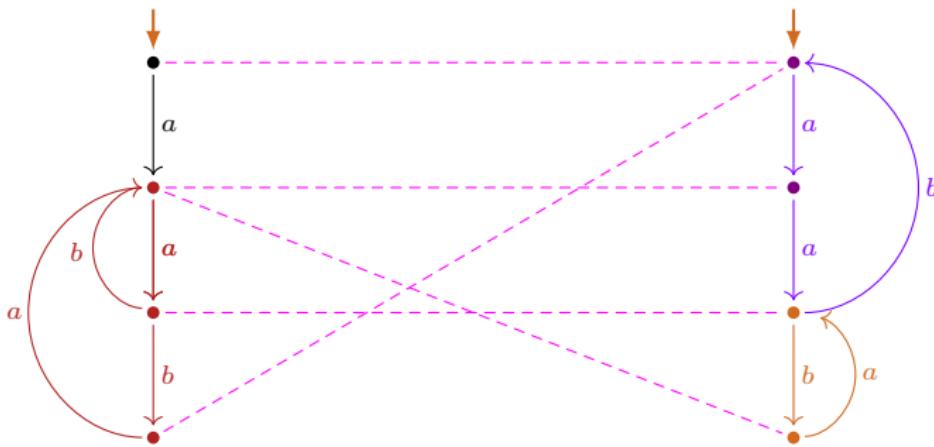
$$P((aa(ba)^*b)^*0)$$

# Process interpretation of regular expressions



$$P(a(a(b+ba))^*0) \quad \leftrightarrow \quad P((aa(ba)^*b)^*0)$$

# Process interpretation of regular expressions


 $a(a(b+ba))^* 0$ 
 $\xleftrightarrow{P}$ 
 $(aa(ba)^*b)^* 0$

# Process graphs and NFAs

## Definition

A **process graph** over actions in  $A$  is a tuple  $G = \langle V, v_s, T, E \rangle$  where:

- ▶  $V$  is a set of *vertices*,
- ▶  $v_s \in V$  is the *start vertex*,
- ▶  $T \subseteq V \times A \times V$  the set of *transitions*,
- ▶  $E \subseteq V \times \{\downarrow\}$  the set of *termination extensions*.

# Process graphs and NFAs

## Definition

A **process graph** over actions in  $A$  is a tuple  $G = \langle V, v_s, T, E \rangle$  where:

- ▶  $V$  is a set of *vertices*,
- ▶  $v_s \in V$  is the *start vertex*,
- ▶  $T \subseteq V \times A \times V$  the set of *transitions*,
- ▶  $E \subseteq V \times \{\downarrow\}$  the set of *termination extensions*.

## Restriction

Here we only consider finite and start-vertex connected process graphs.

# Process graphs and NFAs

## Definition

A **process graph** over actions in  $A$  is a tuple  $G = \langle V, v_s, T, E \rangle$  where:

- ▶  $V$  is a set of *vertices*,
- ▶  $v_s \in V$  is the *start vertex*,
- ▶  $T \subseteq V \times A \times V$  the set of *transitions*,
- ▶  $E \subseteq V \times \{\downarrow\}$  the set of *termination extensions*.

## Restriction

Here we only consider finite and start-vertex connected process graphs.

## Correspondence with NFAs

With the finiteness restriction, process graphs can be viewed as:

- ▶ nondeterministic finite-state automata (**NFAs**),

that are studied under bisimulation, not under language equivalence.

# Process graphs and NFAs

## Definition

A **process graph** over actions in  $A$  is a tuple  $G = \langle V, v_s, T, E \rangle$  where:

- ▶  $V$  is a set of *vertices*,
- ▶  $v_s \in V$  is the *start vertex*,
- ▶  $T \subseteq V \times A \times V$  the set of *transitions*,
- ▶  $E \subseteq V \times \{\downarrow\}$  the set of *termination extensions*.

## Restriction

Here we only consider **finite** and **start-vertex connected** process graphs.

## Correspondence with NFAs

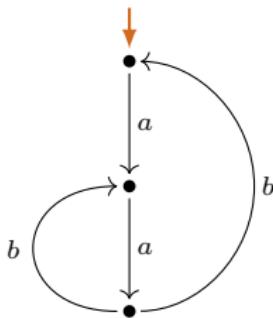
With the finiteness restriction, process graphs can be viewed as:

- ▶ nondeterministic finite-state automata (**NFAs**),

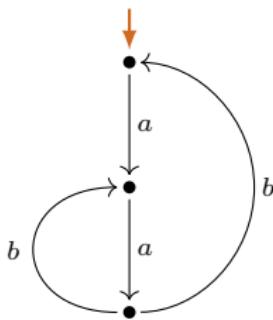
that are studied under bisimulation, not under language equivalence.

*Antimirov (1996): NFA-definition of  $P(\cdot)$  via partial derivatives.*

# Expressible process graphs (under bisimulation $\leftrightarrow$ )

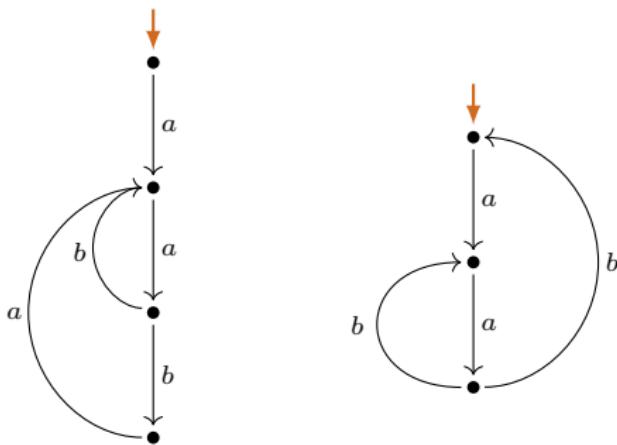


# Expressible process graphs (under bisimulation $\leftrightarrow$ )



?  $\in im(\textcolor{green}{P}(\cdot))$  ?

# Expressible process graphs (under bisimulation $\leftrightarrow$ )

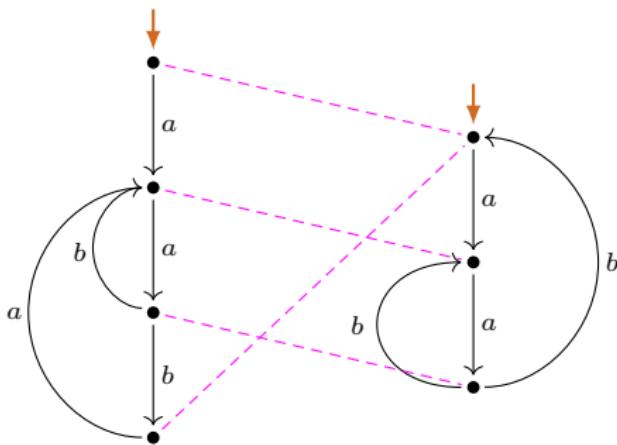


$\in im(\textcolor{green}{P}(\cdot))$

$? \in im(\textcolor{green}{P}(\cdot)) ?$

$\textcolor{blue}{P}(\cdot)$ -expressible

# Expressible process graphs (under bisimulation $\leftrightarrow$ )

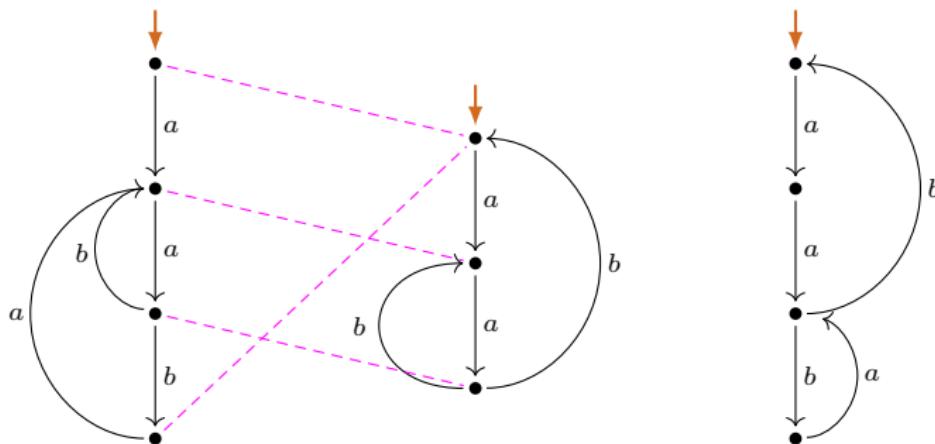


$\in im(\textcolor{green}{P}(\cdot))$

?  $\in im(\textcolor{green}{P}(\cdot))$  ?

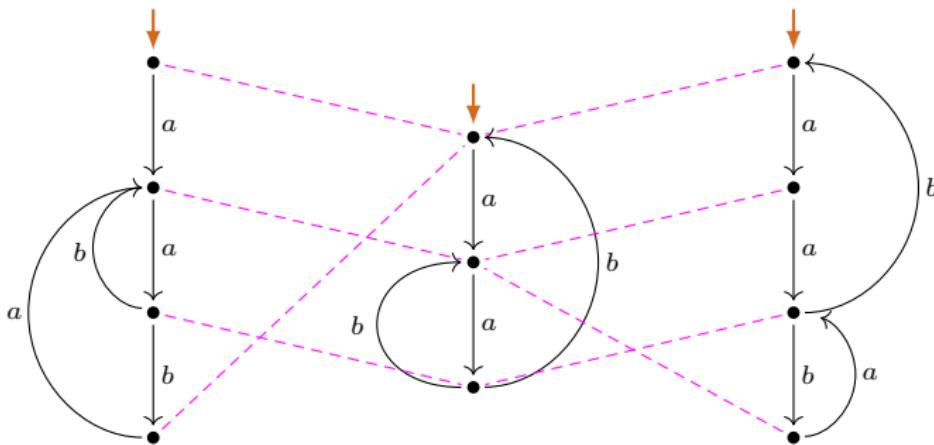
$\textcolor{blue}{P}(\cdot)$ -expressible

# Expressible process graphs (under bisimulation $\leftrightarrow$ )


 $\in im(\textcolor{green}{P}(\cdot))$ 
 $P(\cdot)$ -expressible

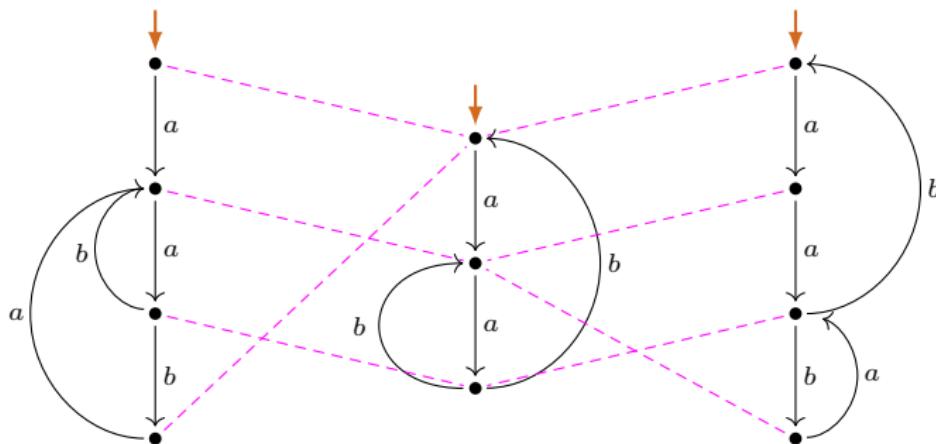
 $? \in im(\textcolor{green}{P}(\cdot)) ?$ 
 $P(\cdot)$ -expressible

# Expressible process graphs (under bisimulation $\leftrightarrow$ )

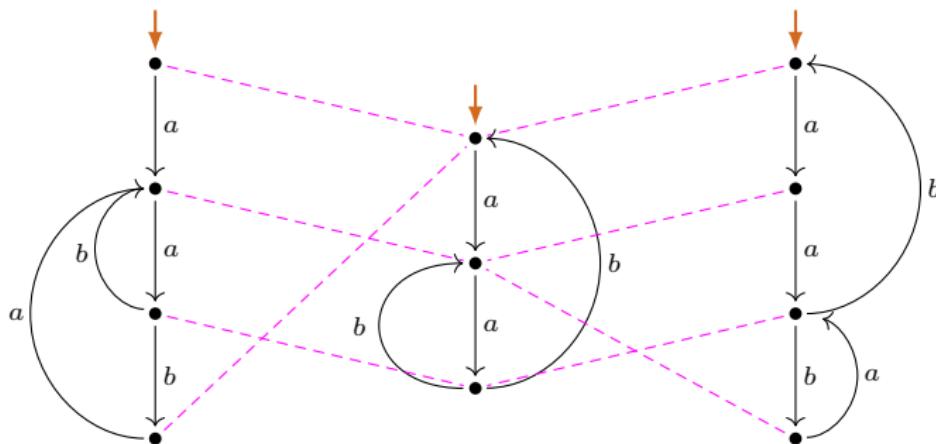

 $\in im(\textcolor{green}{P}(\cdot))$ 
 $P(\cdot)$ -expressible

 $? \in im(\textcolor{green}{P}(\cdot)) ?$ 
 $\in im(\textcolor{green}{P}(\cdot))$ 
 $P(\cdot)$ -expressible

# Expressible process graphs (under bisimulation $\Leftrightarrow$ )


 $\in im(\textcolor{violet}{P}(\cdot))$ 
 $P(\cdot)$ -expressible
 
 $? \in im(\textcolor{violet}{P}(\cdot)) ?$ 
 $P(\cdot)$ -expressible  
modulo  $\Leftrightarrow$ 
 $\in im(\textcolor{violet}{P}(\cdot))$ 
 $P(\cdot)$ -expressible

# Expressible process graphs (under bisimulation $\Leftrightarrow$ )


 $\in im(P(\cdot))$ 
 $P(\cdot)$ -expressible

 $[\cdot]_P$ -expressible

 $? \in im(P(\cdot)) ?$ 
 $P(\cdot)$ -expressible

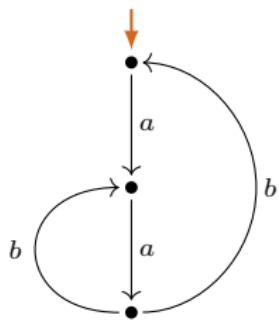
modulo  $\Leftrightarrow$ 
 $[\cdot]_P$ -expressible

 $\in im(P(\cdot))$ 
 $P(\cdot)$ -expressible

 $[\cdot]_P$ -expressible

# Properties of $P$ and $\llbracket \cdot \rrbracket_P$

- ▶ Not every finite-state process is  $P(\cdot)$ -expressible.

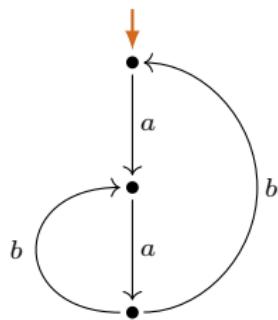


?  $P(\cdot)$ -expressible ?

$\llbracket \cdot \rrbracket_P$ -expressible

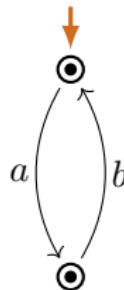
# Properties of $P$ and $\llbracket \cdot \rrbracket_P$

- ▶ Not every finite-state process is  $P(\cdot)$ -expressible.

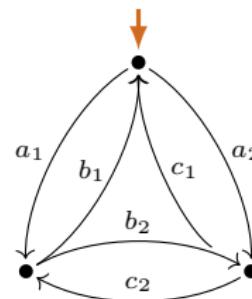


?  $P(\cdot)$ -expressible ?

$\llbracket \cdot \rrbracket_P$ -expressible

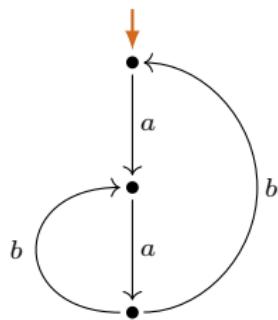


not  $P(\cdot)$ -expressible



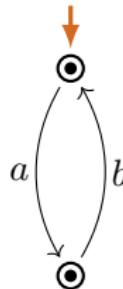
# Properties of $P$ and $\llbracket \cdot \rrbracket_P$

- ▶ Not every finite-state process is  $P(\cdot)$ -expressible.
- ▶ Not every finite-state process is  $\llbracket \cdot \rrbracket_P$ -expressible.



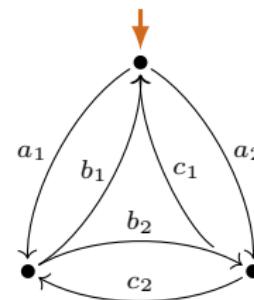
?  $P(\cdot)$ -expressible ?

$\llbracket \cdot \rrbracket_P$ -expressible



not  $P(\cdot)$ -expressible

not  $\llbracket \cdot \rrbracket_P$ -expressible

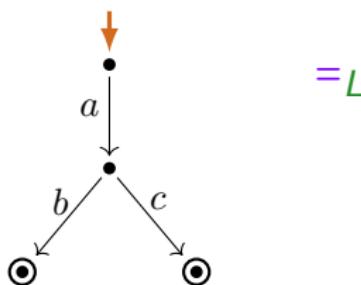


# Properties of $P$ and $\llbracket \cdot \rrbracket_P$

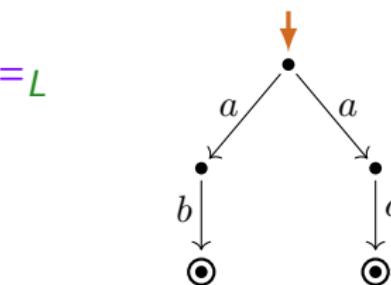
- ▶ Not every finite-state process is  $P(\cdot)$ -expressible.
- ▶ Not every finite-state process is  $\llbracket \cdot \rrbracket_P$ -expressible.
- ▶ Fewer identities hold for  $\leftrightarrow_P$  than for  $=_L$ :  $\leftrightarrow_P \subsetneq =_L$ .

# Properties of $P$ and $\llbracket \cdot \rrbracket_P$

- ▶ Not every finite-state process is  $P(\cdot)$ -expressible.
- ▶ Not every finite-state process is  $\llbracket \cdot \rrbracket_P$ -expressible.
- ▶ Fewer identities hold for  $\Leftrightarrow_P$  than for  $=_L$ :  $\Leftrightarrow_P \subsetneq =_L$ .



$$a \cdot (b + c)$$

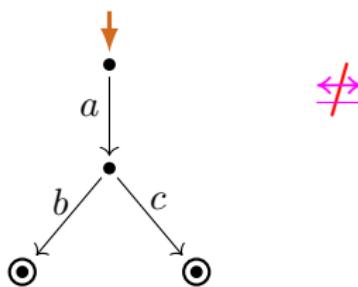


$$= L$$

$$a \cdot b + a \cdot c$$

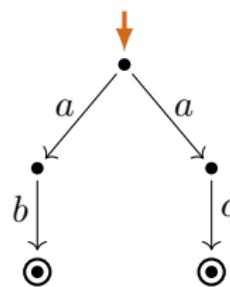
# Properties of $P$ and $\llbracket \cdot \rrbracket_P$

- ▶ Not every finite-state process is  $P(\cdot)$ -expressible.
- ▶ Not every finite-state process is  $\llbracket \cdot \rrbracket_P$ -expressible.
- ▶ Fewer identities hold for  $\llbracket \cdot \rrbracket_P$  than for  $=_L$ :  $\llbracket \cdot \rrbracket_P \subsetneq =_L$ .



$$a \cdot (b + c)$$

$$\not\equiv_P$$



$$a \cdot b + a \cdot c$$

# Complete axiomatization of $=_L$ (Aanderaa/Salomaa, 1965/66)

*Axioms:*

$$(B1) \quad e + (f + g) = (e + f) + g$$

$$(B2) \quad (e \cdot f) \cdot g = e \cdot (f \cdot g)$$

$$(B3) \quad e + f = f + e$$

$$(B4) \quad (e + f) \cdot g = e \cdot g + f \cdot g$$

$$(B5) \quad e \cdot (f + g) = e \cdot f + e \cdot g$$

$$(B6) \quad e + e = e$$

$$(B7) \quad e \cdot 1 = e$$

$$(B8) \quad e \cdot 0 = 0$$

$$(B9) \quad e + 0 = e$$

$$(B10) \quad e^* = 1 + e \cdot e^*$$

$$(B11) \quad e^* = (1 + e)^*$$

*Inference rules:* equational logic plus

$$\frac{e = f \cdot e + g}{e = f^* \cdot g} \text{ FIX } (\text{if } \underbrace{\{e\}}_{\text{non-empty-word}} \notin \textcolor{blue}{L}(\textcolor{brown}{f}))$$

# Sound and unsound axioms with respect to $\leftrightarrow_P$

*Axioms:*

$$(B1) \quad e + (f + g) = (e + f) + g$$

$$(B2) \quad (e \cdot f) \cdot g = e \cdot (f \cdot g)$$

$$(B3) \quad e + f = f + e$$

$$(B4) \quad (e + f) \cdot g = e \cdot g + f \cdot g$$

$$(B5) \quad e \cdot (f + g) = e \cdot f + e \cdot g$$

$$(B6) \quad e + e = e$$

$$(B7) \quad e \cdot 1 = e$$

$$(B8) \quad e \cdot 0 = 0$$

$$(B9) \quad e + 0 = e$$

$$(B10) \quad e^* = 1 + e \cdot e^*$$

$$(B11) \quad e^* = (1 + e)^*$$

*Inference rules:* equational logic plus

$$\frac{e = f \cdot e + g}{e = f^* \cdot g} \text{ FIX} \quad (\text{if } \underbrace{\{e\}}_{\text{non-empty-word}} \notin L(f))$$

# Sound and unsound axioms with respect to $\leftrightarrow_P$

*Axioms:*

$$(B1) \quad e + (f + g) = (e + f) + g$$

$$(B2) \quad (e \cdot f) \cdot g = e \cdot (f \cdot g)$$

$$(B3) \quad e + f = f + e$$

$$(B4) \quad (e + f) \cdot g = e \cdot g + f \cdot g$$

$$(B5) \quad e \cdot (f + g) = e \cdot f + e \cdot g$$

$$(B6) \quad e + e = e$$

$$(B7) \quad e \cdot 1 = e$$

$$(B8) \quad e \cdot 0 = 0$$

$$(B9) \quad e + 0 = e$$

$$(B10) \quad e^* = 1 + e \cdot e^*$$

$$(B11) \quad e^* = (1 + e)^*$$

$$(B8)' \quad 0 \cdot e = 0$$

*Inference rules:* equational logic plus

$$\frac{e = f \cdot e + g}{e = f^* \cdot g} \text{ FIX} \quad (\text{if } \underbrace{\{e\}}_{\text{non-empty-word}} \notin \mathcal{L}(f))$$

# Adaptation for $\leftrightarrow_P$ (Milner, 1984) ( $\text{Mil} = \text{Mil}^- + \text{RSP}^*$ )

*Axioms:*

$$(B1) \quad e + (f + g) = (e + f) + g$$

$$(B2) \quad (e \cdot f) \cdot g = e \cdot (f \cdot g)$$

$$(B3) \quad e + f = f + e$$

$$(B4) \quad (e + f) \cdot g = e \cdot g + f \cdot g$$

$$(B6) \quad e + e = e$$

$$(B7) \quad e \cdot 1 = e$$

$$(B9) \quad e + 0 = e$$

$$(B10) \quad e^* = 1 + e \cdot e^*$$

$$(B11) \quad e^* = (1 + e)^*$$

$$(B8)' \quad 0 \cdot e = 0$$

*Inference rules:* equational logic plus

$$\frac{e = \underline{f} \cdot e + g}{e = \underline{f}^* \cdot g} \text{ RSP}^* \left( \text{if } \underbrace{\{\epsilon\}}_{\text{non-empty-word}} \notin \text{L}(\underline{f}) \right)$$

# Adaptation for $\leftrightarrow_P$ (Milner, 1984) ( $\text{Mil} = \text{Mil}^- + \text{RSP}^*$ )

*Axioms:*

$$(B1) \quad e + (f + g) = (e + f) + g$$

$$(B2) \quad (e \cdot f) \cdot g = e \cdot (f \cdot g)$$

$$(B3) \quad e + f = f + e$$

$$(B4) \quad (e + f) \cdot g = e \cdot g + f \cdot g$$

$$(B6) \quad e + e = e$$

$$(B7) \quad e \cdot 1 = e$$

$$(B8)' \quad 0 \cdot e = 0$$

$$(B9) \quad e + 0 = e$$

$$(B10) \quad e^* = 1 + e \cdot e^*$$

$$(B11) \quad e^* = (1 + e)^*$$

*Inference rules:* equational logic plus

$$\frac{e = \underline{f} \cdot e + g}{e = \underline{f}^* \cdot g} \text{ RSP}^* \left( \text{if } \underbrace{\{\epsilon\}}_{\text{non-empty-word}} \notin \text{L}(\underline{f}) \right)$$

# Adaptation for $\leftrightarrow_P$ (Milner, 1984) ( $\text{Mil} = \text{Mil}^- + \text{RSP}^*$ )

*Axioms:*

$$(B1) \quad e + (f + g) = (e + f) + g$$

$$(B2) \quad (e \cdot f) \cdot g = e \cdot (f \cdot g)$$

$$(B3) \quad e + f = f + e$$

$$(B4) \quad (e + f) \cdot g = e \cdot g + f \cdot g$$

$$(B6) \quad e + e = e$$

$$(B7) \quad e \cdot 1 = e$$

$$(B8)' \quad 0 \cdot e = 0$$

$$(B9) \quad e + 0 = e$$

$$(B10) \quad e^* = 1 + e \cdot e^*$$

$$(B11) \quad e^* = (1 + e)^*$$

*Inference rules:* equational logic plus

$$\frac{e = \underline{f} \cdot e + g}{e = \underline{f}^* \cdot g} \text{ RSP}^* \left( \text{if } \underbrace{\{\epsilon\}}_{\text{non-empty-word}} \notin \text{L}(\underline{f}) \right)$$

# Milner's questions

Q2. Complete axiomatization: Is Mil complete for  $\Leftarrow_P$  ?

# Milner's questions

Q1. **Recognition:** Which structural property of finite process graphs characterizes  $P(\cdot)$ -expressibility modulo  $\leftrightarrow$  ?

Q2. **Complete axiomatization:** Is Mil complete for  $\leftrightarrow_P$  ?

# Milner's questions, and partial results

Q1. **Recognition:** Which structural property of finite process graphs characterizes  $P(\cdot)$ -expressibility modulo  $\leftrightarrow$  ?

Q2. **Complete axiomatization:** Is Mil complete for  $\leftrightarrow_P$  ?

# Milner's questions, and partial results

Q1. **Recognition:** Which structural property of finite process graphs characterizes  $P(\cdot)$ -expressibility modulo  $\leftrightarrow$  ?

- ▶ definability by well-behaved specifications (*Baeten/Corradini, 2005*)

Q2. **Complete axiomatization:** Is Mil complete for  $\leftrightarrow_P$  ?

# Milner's questions, and partial results

Q1. **Recognition:** Which structural property of finite process graphs characterizes  $P(\cdot)$ -expressibility modulo  $\leftrightarrow$  ?

- ▶ definability by well-behaved specifications (*Baeten/Corradini, 2005*)
- ▶ that is decidable (super-exponentially) (*Baeten/Corradini/G, 2007*)

Q2. **Complete axiomatization:** Is Mil complete for  $\leftrightarrow_P$  ?

# Milner's questions, and partial results

Q1. **Recognition:** Which structural property of finite process graphs characterizes  $P(\cdot)$ -expressibility modulo  $\leftrightarrow$ ?

- ▶ definability by well-behaved specifications (*Baeten/Corradini, 2005*)
- ▶ that is decidable (super-exponentially) (*Baeten/Corradini/G, 2007*)

Q2. **Complete axiomatization:** Is Mil complete for  $\leftrightarrow_P$ ?

- ▶ Mil is complete for perpetual-loop expressions (*Fokkink, 1996*)
  - ▶ every iteration  $e^*$  occurs as part of a '*no-exit*' subexpression  $e^* \cdot 0$

# Milner's questions, and partial results

Q1. **Recognition:** Which structural property of finite process graphs characterizes  $P(\cdot)$ -expressibility modulo  $\leftrightarrow$ ?

- ▶ definability by well-behaved specifications (*Baeten/Corradini, 2005*)
- ▶ that is decidable (super-exponentially) (*Baeten/Corradini/G, 2007*)

Q2. **Complete axiomatization:** Is Mil complete for  $\leftrightarrow_P$ ?

- ▶ Mil is complete for perpetual-loop expressions (*Fokkink, 1996*)
  - ▶ every iteration  $e^*$  occurs as part of a '*no-exit*' subexpression  $e^* \cdot 0$
- ▶ Mil is complete when restricted to 1-return-less expressions  
(*Corradini, De Nicola, Labella, 2002*)

# Milner's questions, and partial results

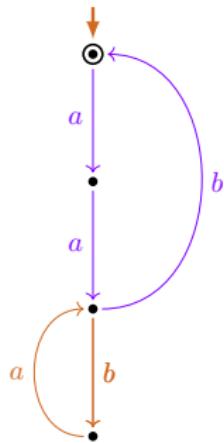
Q1. **Recognition:** Which structural property of finite process graphs characterizes  $P(\cdot)$ -expressibility modulo  $\leftrightarrow$ ?

- ▶ definability by well-behaved specifications (*Baeten/Corradini, 2005*)
- ▶ that is decidable (super-exponentially) (*Baeten/Corradini/G, 2007*)

Q2. **Complete axiomatization:** Is Mil complete for  $\leftrightarrow_P$ ?

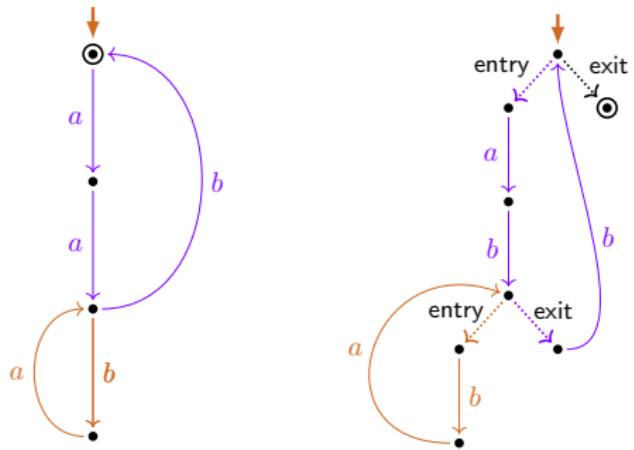
- ▶ Mil is complete for perpetual-loop expressions (*Fokkink, 1996*)
  - ▶ every iteration  $e^*$  occurs as part of a '*no-exit*' subexpression  $e^* \cdot 0$
- ▶ Mil is complete when restricted to 1-return-less expressions (*Corradini, De Nicola, Labella, 2002*)
- ▶ Mil<sup>-</sup> + one of two stronger rules (than RSP\*) is complete (*G, 2006*)

# Well-behaved form, looping palm trees



$P((aa(ba)^*b)^*)$

# Well-behaved form, looping palm trees

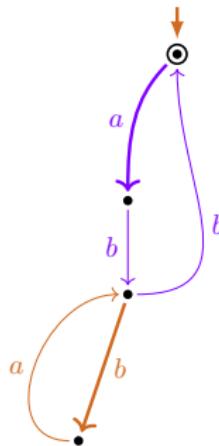
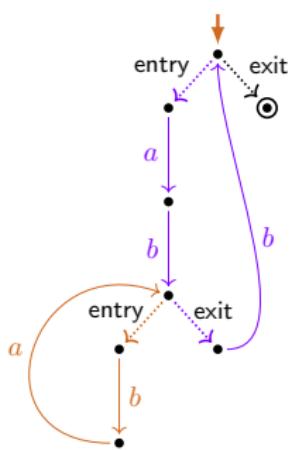
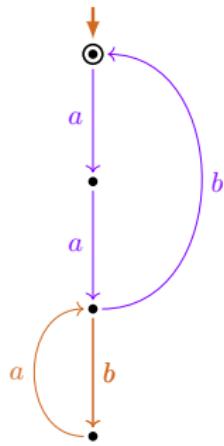


well-behaved form  
(Corradini, Baeten)

$$P((aa(ba)^*b)^*)$$

$$P((1 \cdot aa(1 \cdot ba)^* \cdot 1 \cdot b)^*(1 \cdot 1))$$

# Well-behaved form, looping palm trees



well-behaved form  
(Corradini, Baeten)

$$P((aa(ba)^*b)^*)$$

$$P((1 \cdot aa(1 \cdot ba)^* \cdot 1 \cdot b)^*(1 \cdot 1))$$

looping palm tree

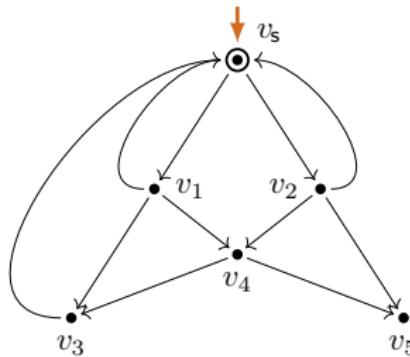
$$P((aa(ba)^*b)^*)$$

# Loop charts (interpretations of innermost iterations)

## Definition

A process graph is a **loop chart** if:

- L-1.
- L-2.
- L-3.

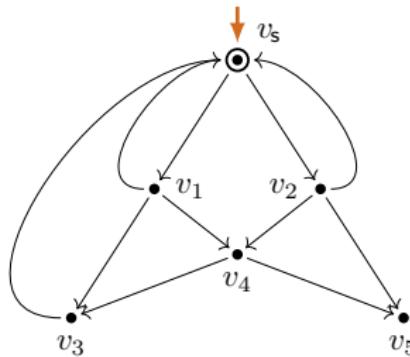


# Loop charts (interpretations of innermost iterations)

## Definition

A process graph is a **loop chart** if:

- L-1. There is an infinite path from the **start vertex**.
- L-2.
- L-3.

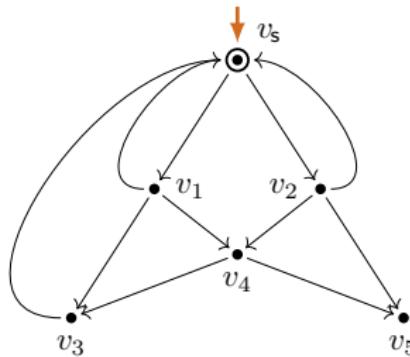


# Loop charts (interpretations of innermost iterations)

## Definition

A process graph is a **loop chart** if:

- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3.

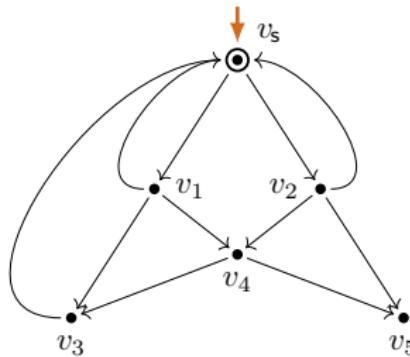


# Loop charts (interpretations of innermost iterations)

## Definition

A process graph is a **loop chart** if:

- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.

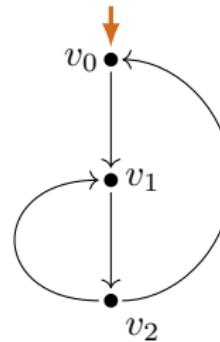
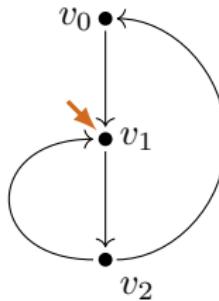
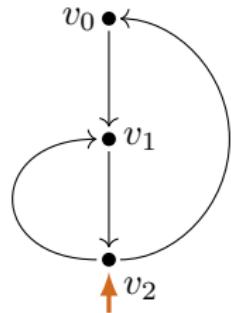


# Loop charts (interpretations of innermost iterations)

## Definition

A process graph is a **loop chart** if:

- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.

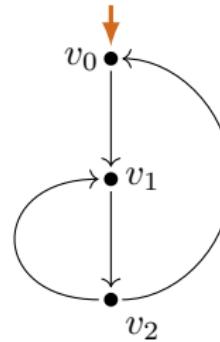
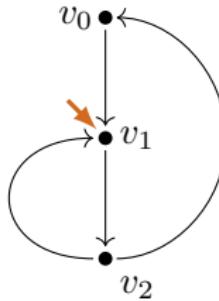
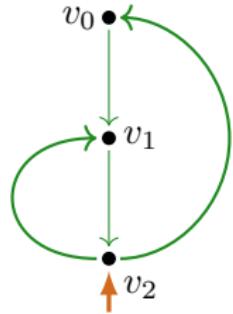


# Loop charts (interpretations of innermost iterations)

## Definition

A process graph is a **loop chart** if:

- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.

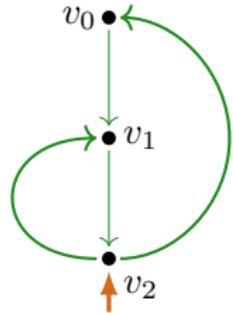


# Loop charts (interpretations of innermost iterations)

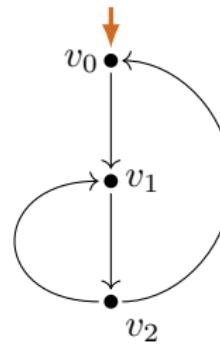
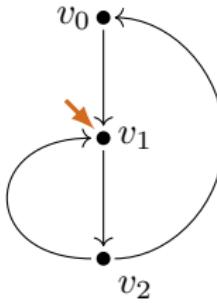
## Definition

A process graph is a **loop chart** if:

- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.



loop chart

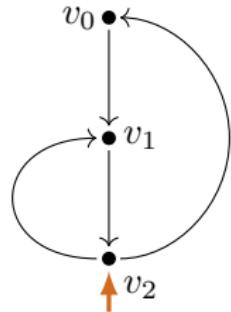


# Loop charts (interpretations of innermost iterations)

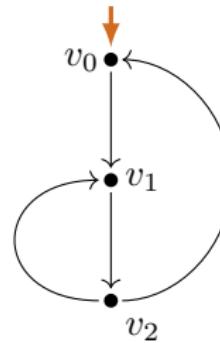
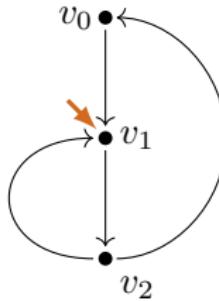
## Definition

A process graph is a **loop chart** if:

- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.



loop chart

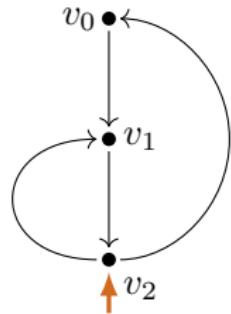


# Loop charts (interpretations of innermost iterations)

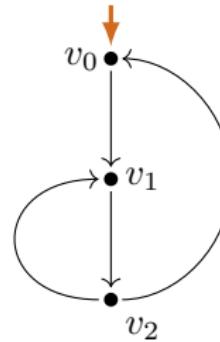
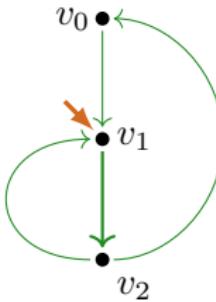
## Definition

A process graph is a **loop chart** if:

- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.



loop chart

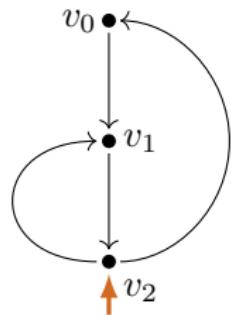


# Loop charts (interpretations of innermost iterations)

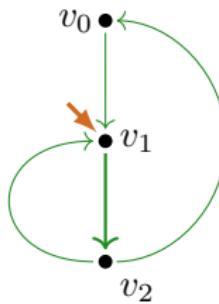
## Definition

A process graph is a **loop chart** if:

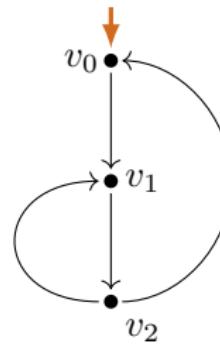
- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.



loop chart



loop chart

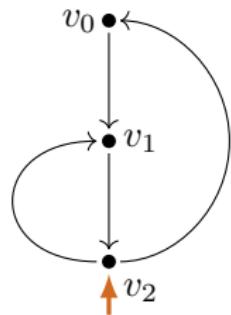


# Loop charts (interpretations of innermost iterations)

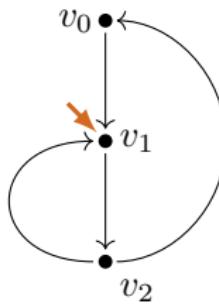
## Definition

A process graph is a **loop chart** if:

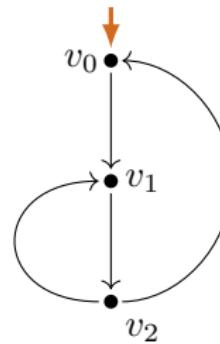
- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.



loop chart



loop chart

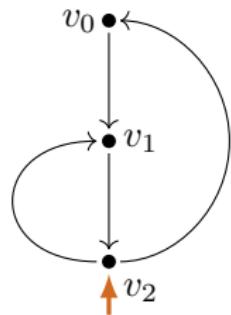


# Loop charts (interpretations of innermost iterations)

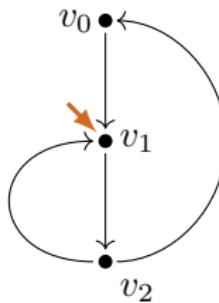
## Definition

A process graph is a **loop chart** if:

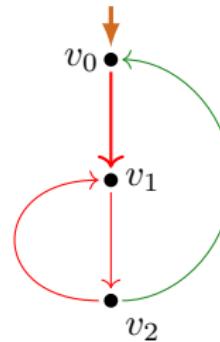
- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.



loop chart



loop chart

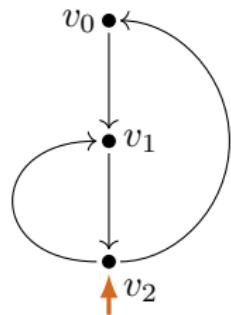


# Loop charts (interpretations of innermost iterations)

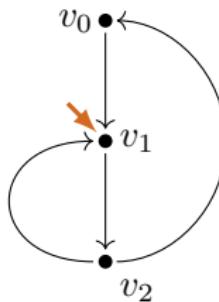
## Definition

A process graph is a **loop chart** if:

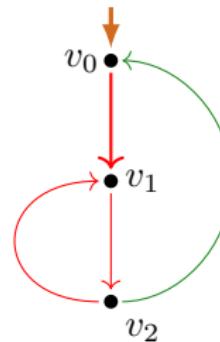
- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.



loop chart



loop chart



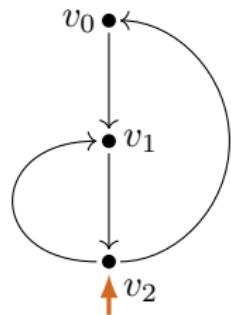
no loop chart

# Loop charts (interpretations of innermost iterations)

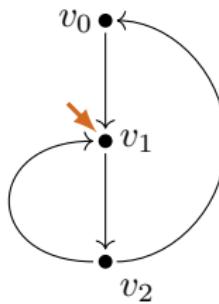
## Definition

A process graph is a **loop chart** if:

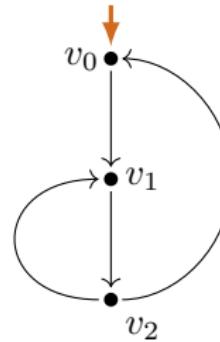
- L-1. There is an infinite path from the **start vertex**.
- L-2. Every infinite path from the **start vertex** returns to **it**.
- L-3. Termination is only possible at the **start vertex**.



loop chart

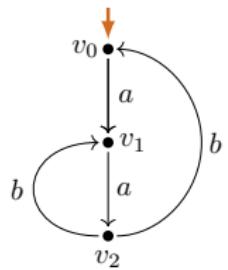


loop chart

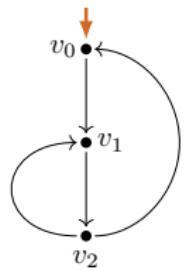


no loop chart

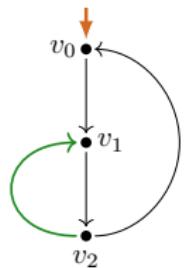
# Loop elimination



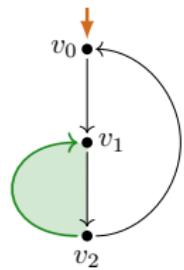
# Loop elimination



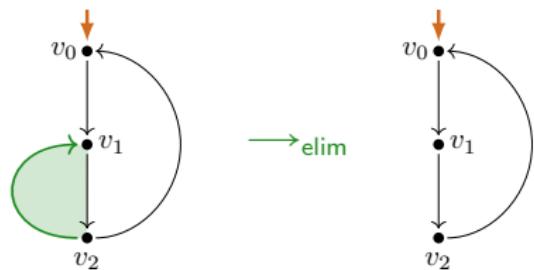
# Loop elimination



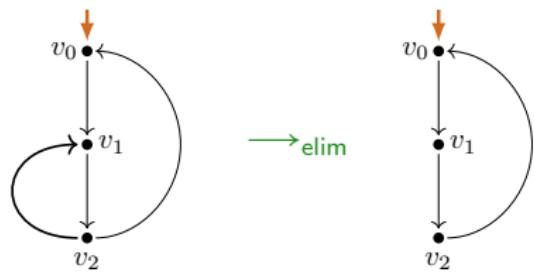
# Loop elimination



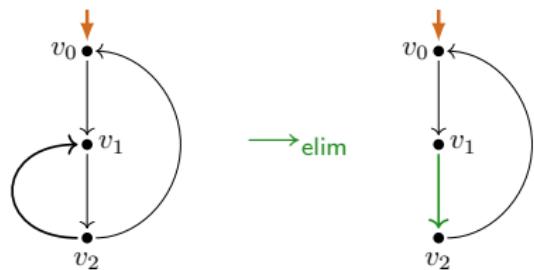
# Loop elimination



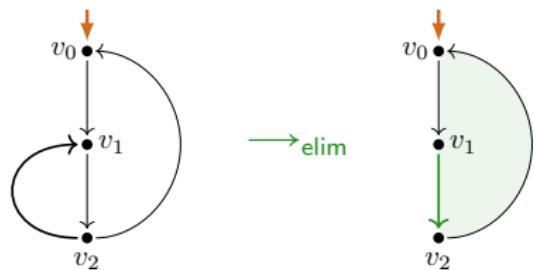
# Loop elimination



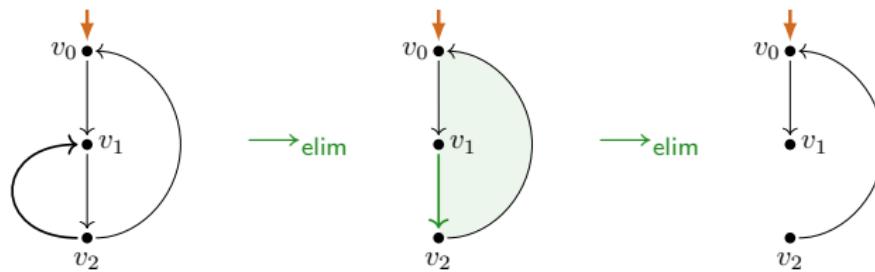
# Loop elimination



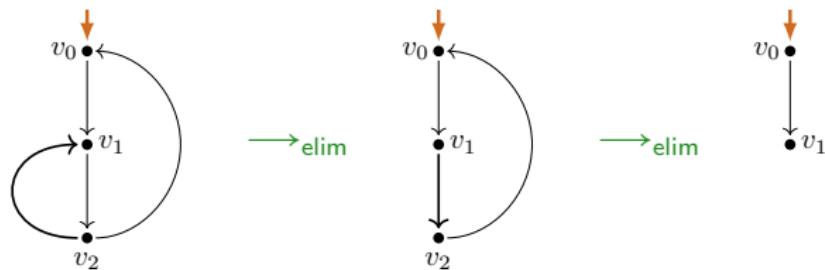
# Loop elimination



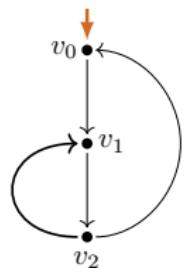
# Loop elimination



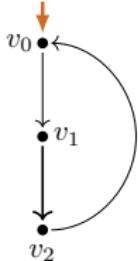
# Loop elimination



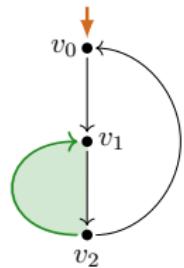
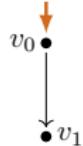
# Loop elimination



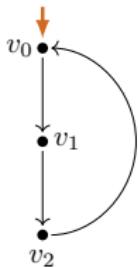
→ elim



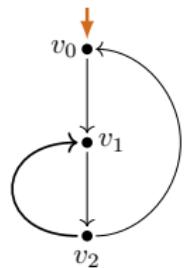
→ elim



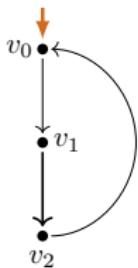
→ elim



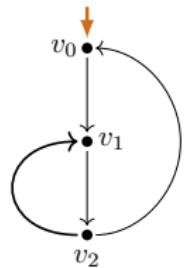
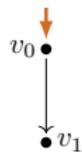
# Loop elimination



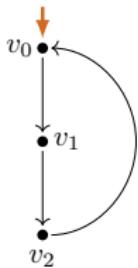
→ elim



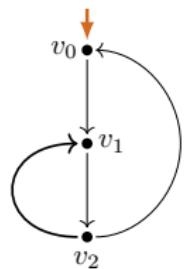
→ elim



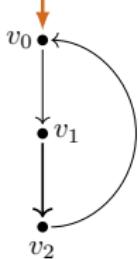
→ elim



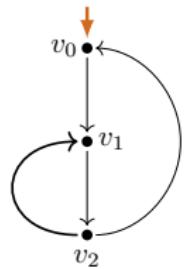
# Loop elimination



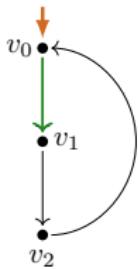
→ elim



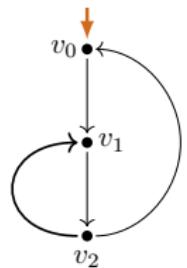
→ elim



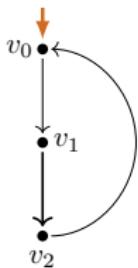
→ elim



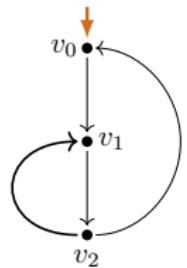
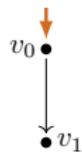
# Loop elimination



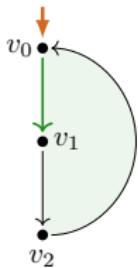
→ elim



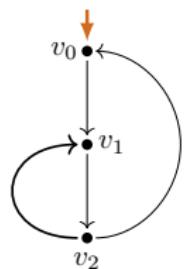
→ elim



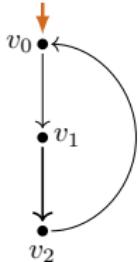
→ elim



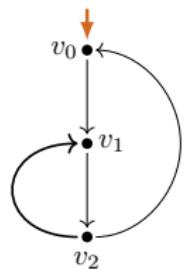
# Loop elimination



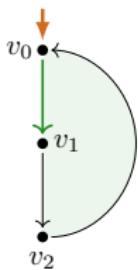
→ elim



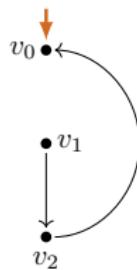
→ elim



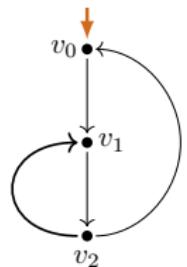
→ elim



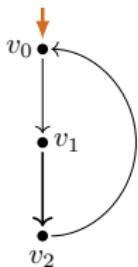
→ elim



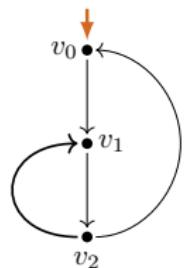
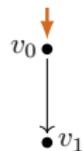
# Loop elimination



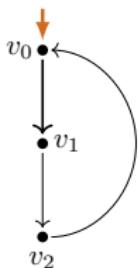
→ elim



→ elim



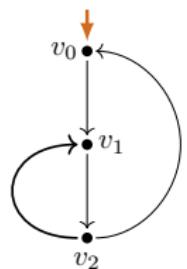
→ elim



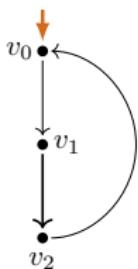
→ elim



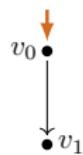
# Loop elimination



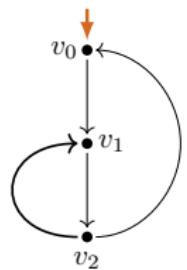
→ elim



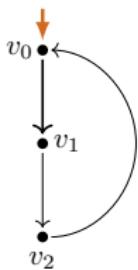
→ elim



→ prune



→ elim



→ elim



# Loop elimination, and properties

$\longrightarrow_{\text{elim}}$  : eliminate a transition-induced loop by:

- ▶ removing the loop-entry transition(s)
- ▶ garbage collection

$\longrightarrow_{\text{prune}}$  : remove a transition to a deadlocking state

## Lemma

(i)  $\longrightarrow_{\text{elim}} \cup \longrightarrow_{\text{prune}}$  is terminating.

# Loop elimination, and properties

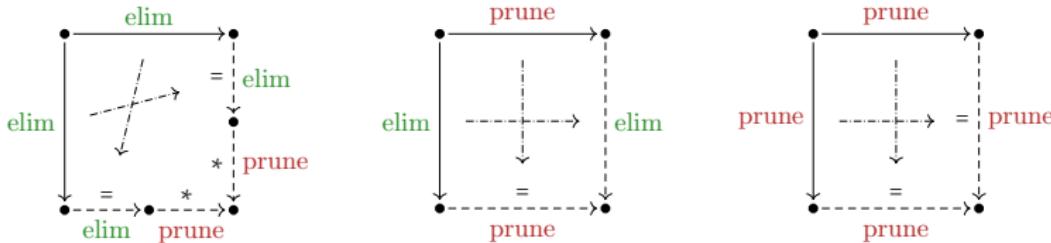
$\rightarrow_{\text{elim}}$  : eliminate a transition-induced loop by:

- ▶ removing the loop-entry transition(s)
- ▶ garbage collection

$\rightarrow_{\text{prune}}$  : remove a transition to a deadlocking state

## Lemma

- (i)  $\rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$  is terminating.
- (ii)  $\rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$  is decreasing, and hence locally confluent.



# Loop elimination, and properties

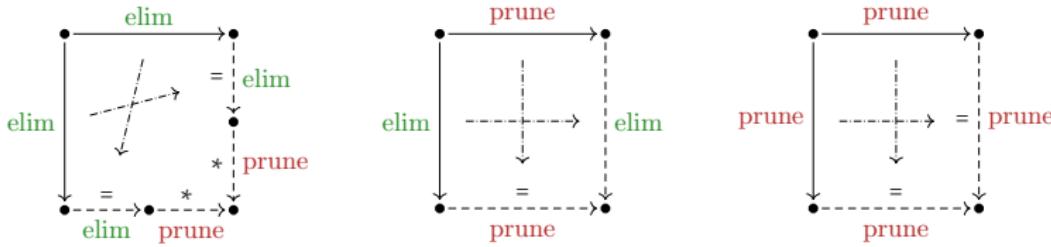
$\rightarrow_{\text{elim}}$  : eliminate a transition-induced loop by:

- ▶ removing the loop-entry transition(s)
- ▶ garbage collection

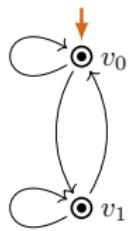
$\rightarrow_{\text{prune}}$  : remove a transition to a deadlocking state

## Lemma

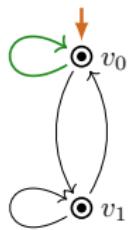
- (i)  $\rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$  is terminating.
- (ii)  $\rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$  is decreasing, and hence locally confluent.
- (iii)  $\rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$  is confluent.



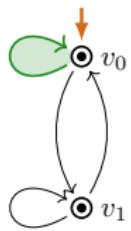
# Loop elimination



# Loop elimination



# Loop elimination



# Loop elimination



# Loop elimination



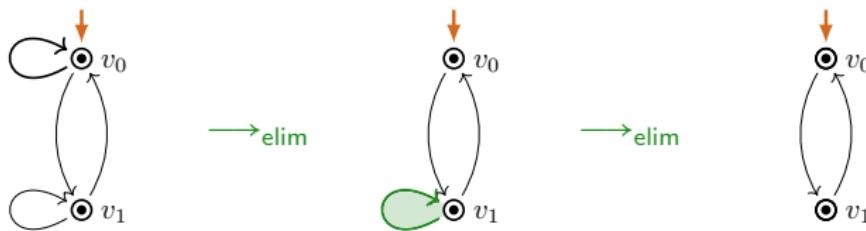
# Loop elimination



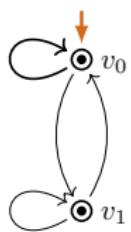
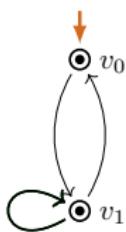
# Loop elimination



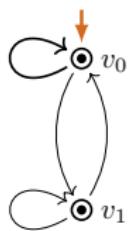
# Loop elimination



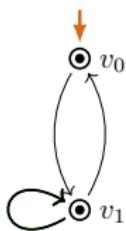
# Loop elimination

 $\rightarrow_{\text{elim}}$  $\rightarrow_{\text{elim}}$ 

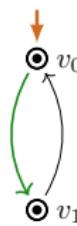
# Loop elimination



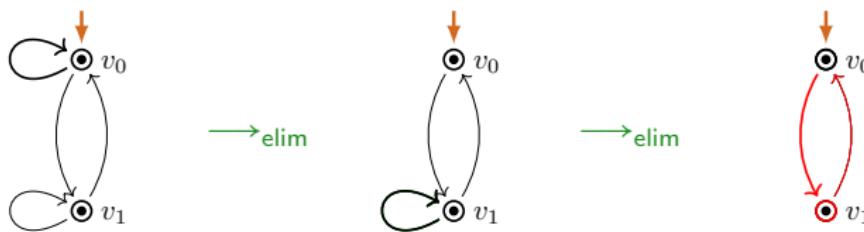
→<sub>elim</sub>



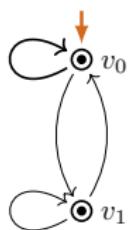
→<sub>elim</sub>



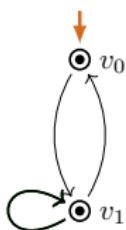
# Loop elimination



# Loop elimination



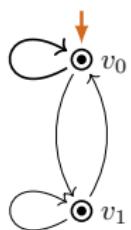
→<sub>elim</sub>



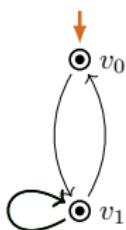
→<sub>elim</sub>



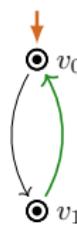
# Loop elimination



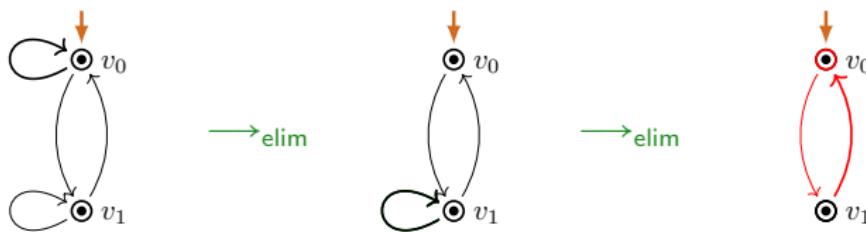
→<sub>elim</sub>



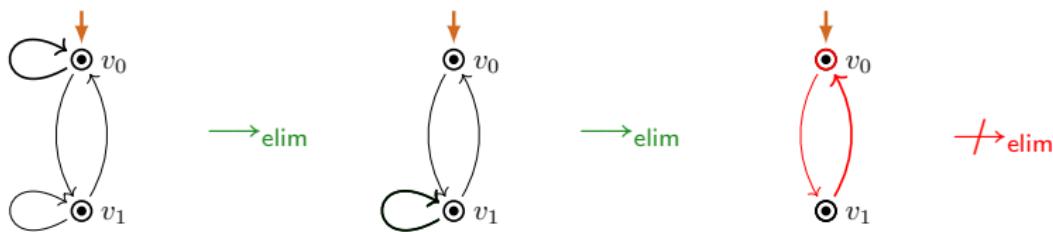
→<sub>elim</sub>



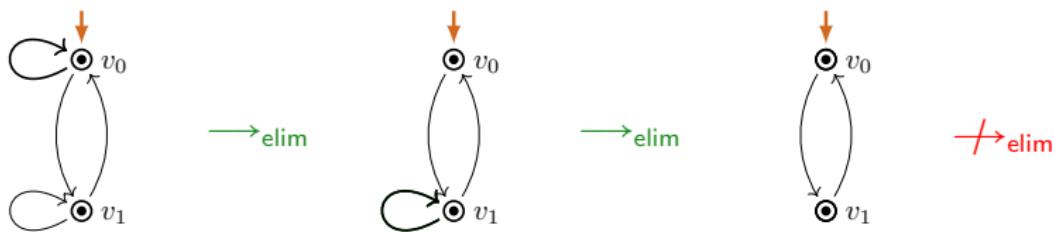
# Loop elimination



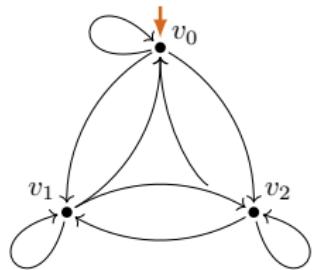
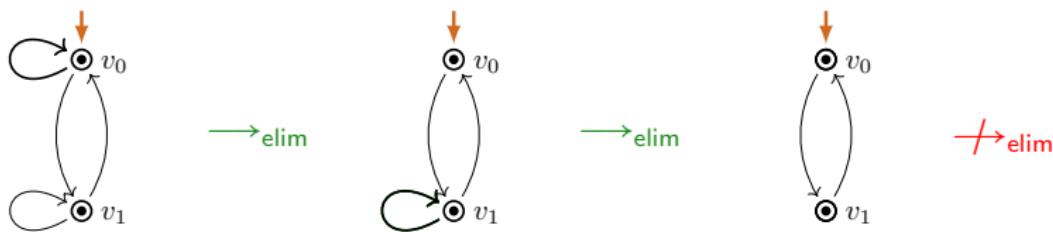
# Loop elimination



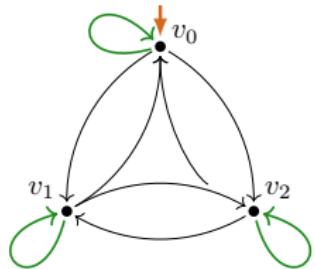
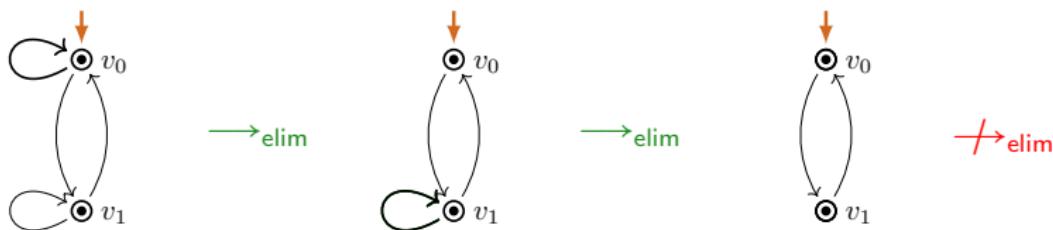
# Loop elimination



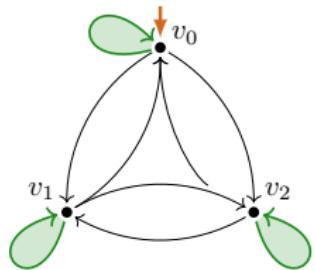
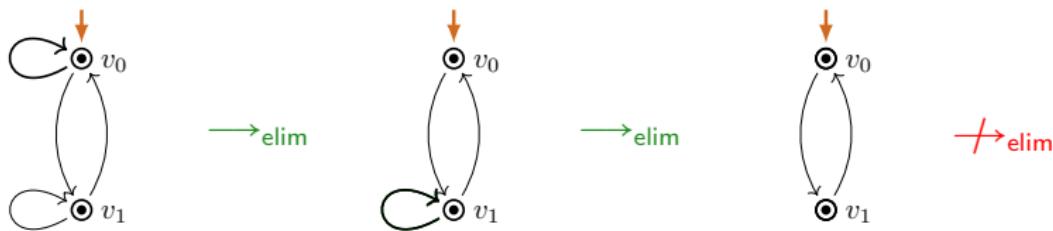
# Loop elimination



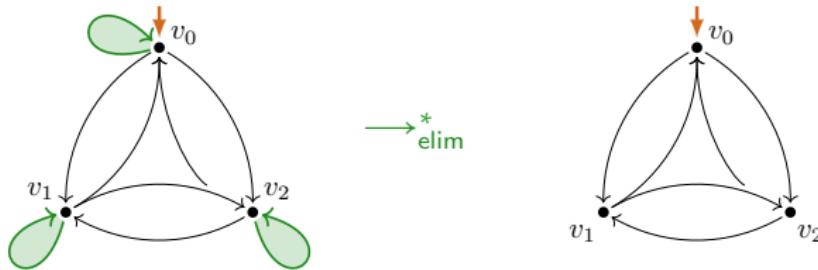
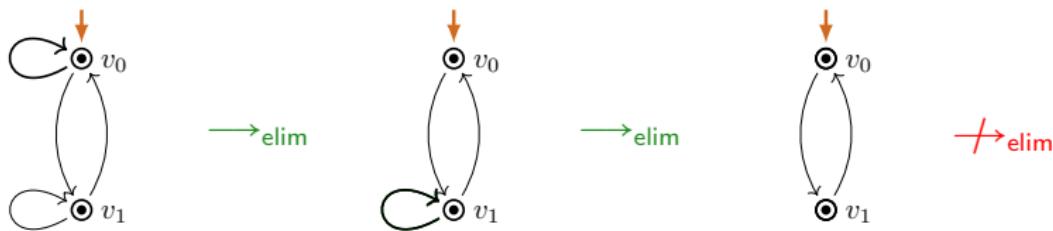
# Loop elimination



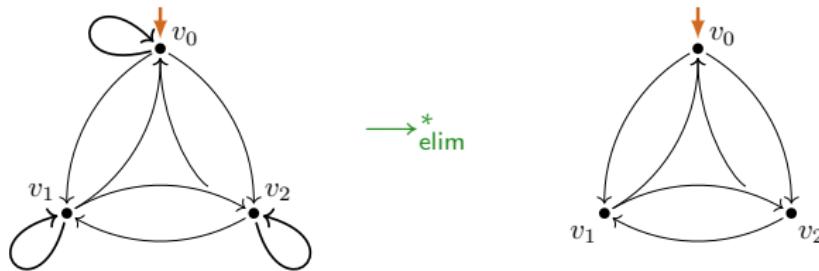
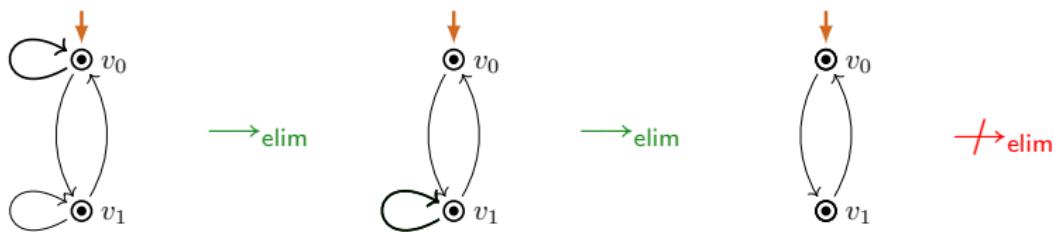
# Loop elimination



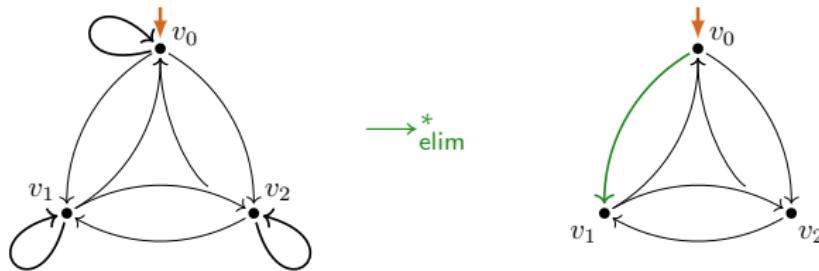
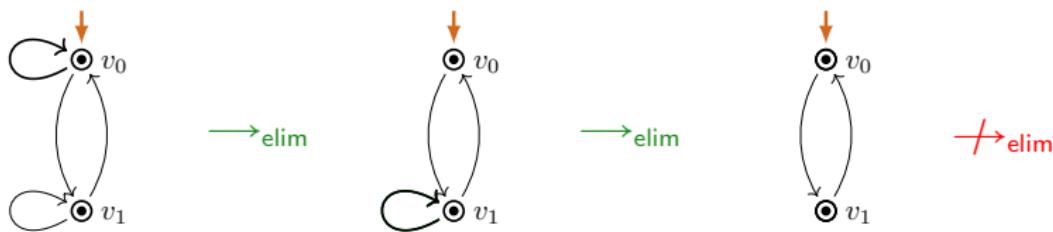
# Loop elimination



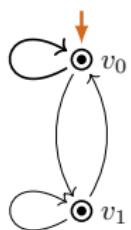
# Loop elimination



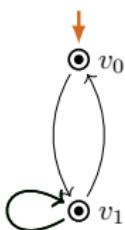
# Loop elimination



# Loop elimination



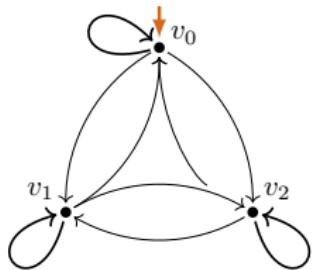
$\rightarrow_{\text{elim}}$



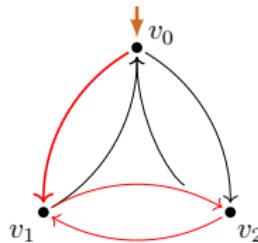
$\rightarrow_{\text{elim}}$



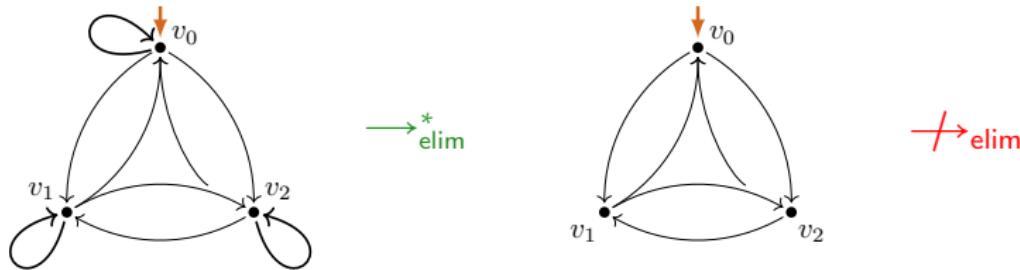
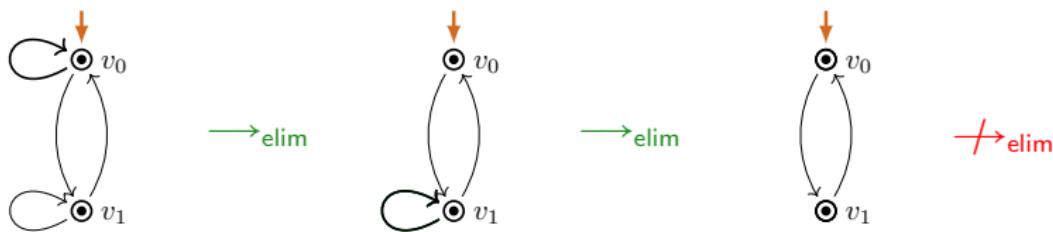
$\cancel{\rightarrow}_{\text{elim}}$



$\rightarrow^*_{\text{elim}}$



# Loop elimination



# Structure property LEE

## Definition

A process graph  $G$  satisfies **LEE** (*loop existence and elimination*) if:

$$\exists G_0 \left( G \xrightarrow{\text{elim}}^* G_0 \not\rightarrow_{\text{elim}} \wedge G_0 \text{ has no infinite trace} \right).$$

# Structure property LEE

## Definition

A process graph  $G$  satisfies LEE (*loop existence and elimination*) if:

$$\exists G_0 \left( G \xrightarrow{\text{elim}}^* G_0 \not\rightarrow_{\text{elim}} \wedge G_0 \text{ has no infinite trace} \right).$$

## Lemma (by using termination and confluence)

For every process graph  $G$  the following are equivalent:

- (i) LEE( $G$ ).
- (ii) *There is an  $\xrightarrow{\text{elim}}$  normal form without an infinite trace.*

# Structure property LEE

## Definition

A process graph  $G$  satisfies LEE (*loop existence and elimination*) if:

$$\exists G_0 \left( G \xrightarrow{\text{elim}}^* G_0 \not\rightarrow_{\text{elim}} \wedge G_0 \text{ has no infinite trace} \right).$$

## Lemma (by using termination and confluence)

For every process graph  $G$  the following are equivalent:

- (i) LEE( $G$ ).
- (ii) *There is an  $\xrightarrow{\text{elim}}$  normal form without an infinite trace.*
- (iii) *There is an  $\xrightarrow{\text{elim,prune}}$  normal form without an infinite trace.*

# Structure property LEE

## Definition

A process graph  $G$  satisfies LEE (*loop existence and elimination*) if:

$$\exists G_0 \left( G \xrightarrow{\text{elim}}^* G_0 \not\rightarrow_{\text{elim}} \wedge G_0 \text{ has no infinite trace} \right).$$

## Lemma (by using termination and confluence)

For every process graph  $G$  the following are equivalent:

- (i) LEE( $G$ ).
- (ii) *There is an*  $\xrightarrow{\text{elim}}$  *normal form without* an infinite trace.
- (iii) *There is an*  $\xrightarrow{\text{elim,prune}}$  *normal form without* an infinite trace.
- (iv) *Every*  $\xrightarrow{\text{elim}}$  *normal form is without* an infinite trace.
- (v) *Every*  $\xrightarrow{\text{elim,prune}}$  *normal form is without* an infinite trace.

# Structure property LEE

## Definition

A process graph  $G$  satisfies LEE (*loop existence and elimination*) if:

$$\exists G_0 \left( G \xrightarrow{\text{elim}}^* G_0 \not\rightarrow_{\text{elim}} \wedge G_0 \text{ has no infinite trace} \right).$$

## Lemma (by using termination and confluence)

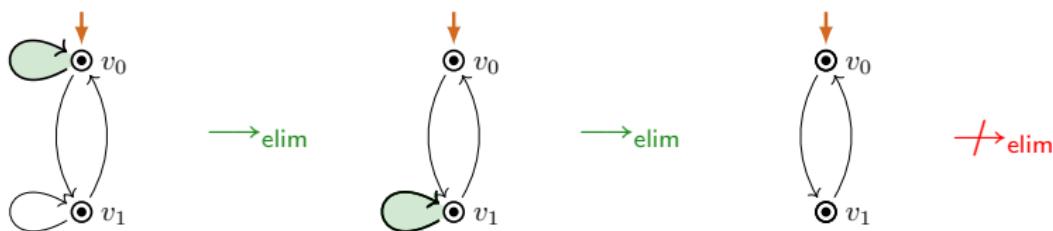
For every process graph  $G$  the following are equivalent:

- (i) LEE( $G$ ).
- (ii) *There is an*  $\xrightarrow{\text{elim}}$  *normal form without* an infinite trace.
- (iii) *There is an*  $\xrightarrow{\text{elim,prune}}$  *normal form without* an infinite trace.
- (iv) *Every*  $\xrightarrow{\text{elim}}$  *normal form is without* an infinite trace.
- (v) *Every*  $\xrightarrow{\text{elim,prune}}$  *normal form is without* an infinite trace.

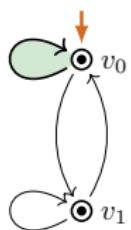
## Theorem (efficient decidability)

The problem of deciding LEE( $G$ ) for process graphs  $G$  is in P.

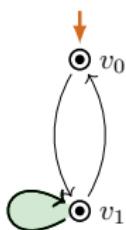
# LEE fails



# LEE fails



$\rightarrow_{\text{elim}}$



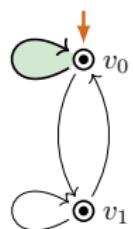
$\rightarrow_{\text{elim}}$



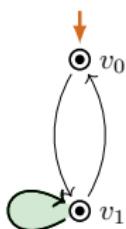
$\cancel{\rightarrow}_{\text{elim}}$

$\neg \text{LEE}$

# LEE fails



$\rightarrow_{\text{elim}}$

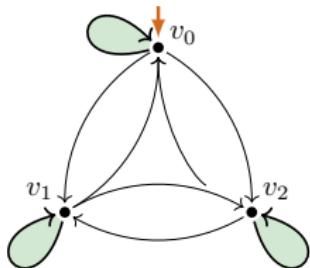


$\rightarrow_{\text{elim}}$

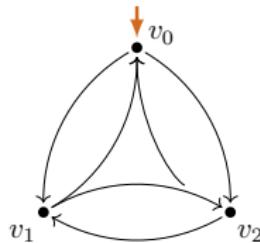


$\cancel{\rightarrow}_{\text{elim}}$

$\neg \text{LEE}$

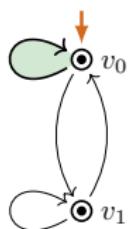


$\rightarrow^*_{\text{elim}}$

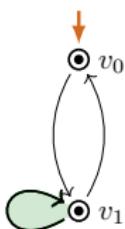


$\cancel{\rightarrow}_{\text{elim}}$

# LEE fails



$\rightarrow_{\text{elim}}$

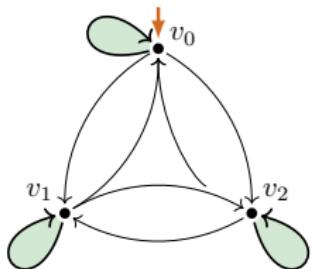


$\rightarrow_{\text{elim}}$

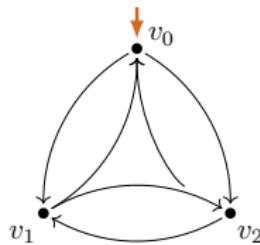


$\cancel{\rightarrow}_{\text{elim}}$

$\neg \text{LEE}$



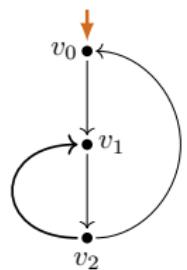
$\rightarrow^*_{\text{elim}}$



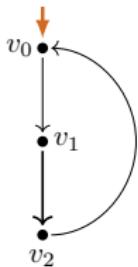
$\cancel{\rightarrow}_{\text{elim}}$

$\neg \text{LEE}$

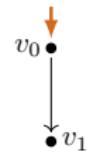
# LEE holds



→ elim



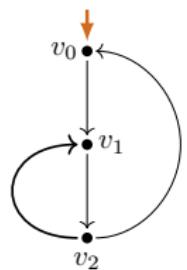
→ elim



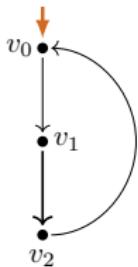
→ prune



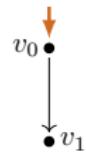
# LEE holds



→ elim



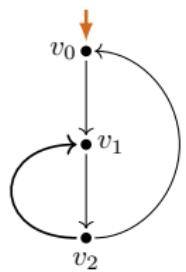
→ elim



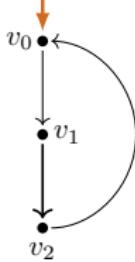
→ prune

LEE

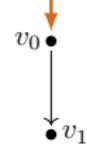
# LEE holds / Recording loop elimination



→ elim



→ elim

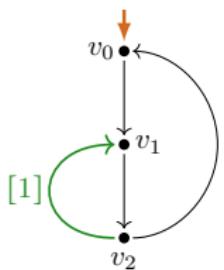


→ prune

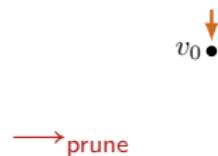
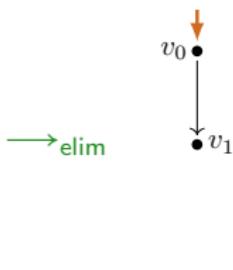
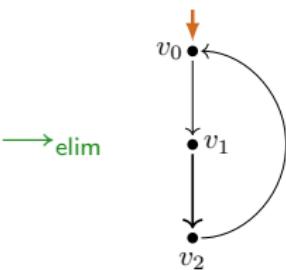


LEE

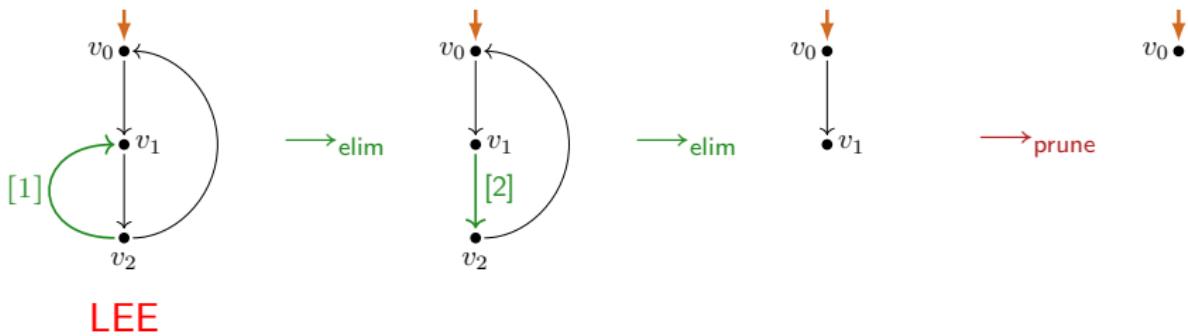
# LEE holds / Recording loop elimination



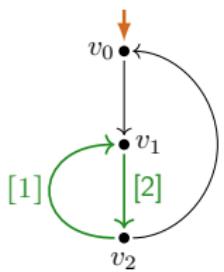
LEE



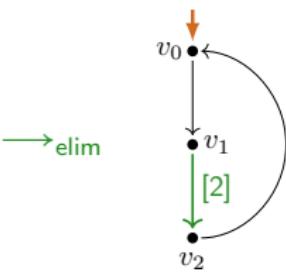
# LEE holds / Recording loop elimination



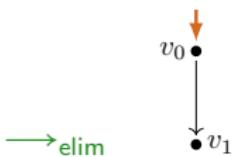
# LEE holds / Recording loop elimination



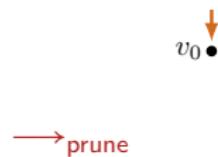
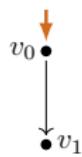
LEE



→ elim

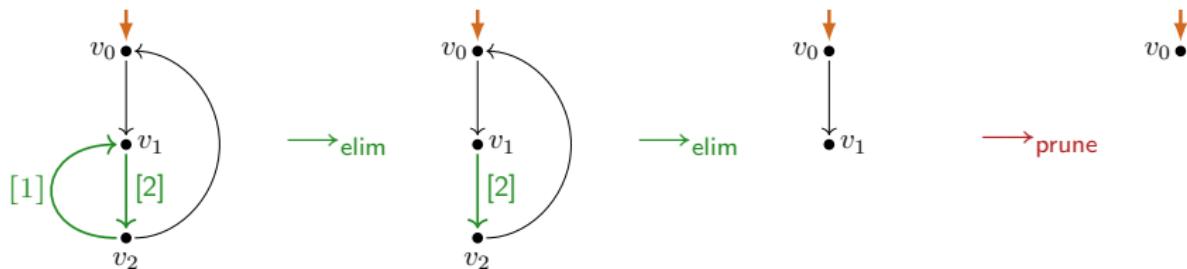


→ elim

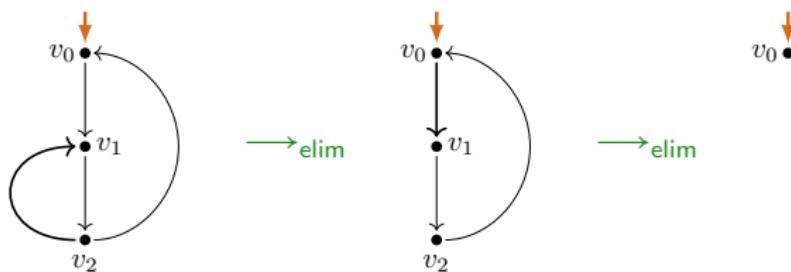


→ prune

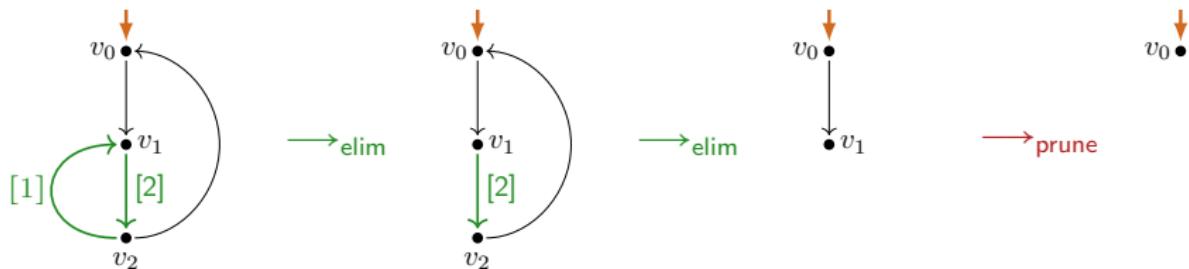
# LEE holds / Recording loop elimination



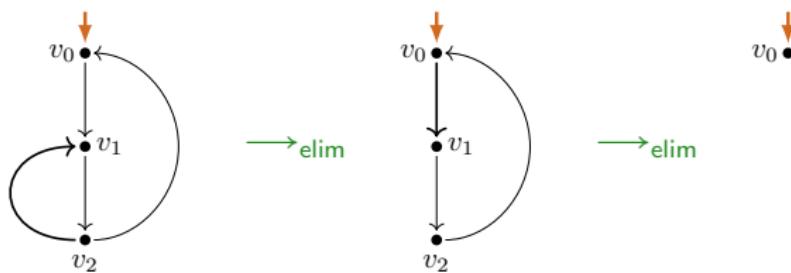
**LEE**



# LEE holds / Recording loop elimination

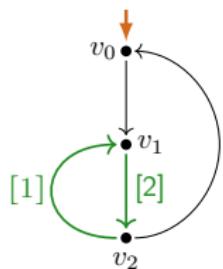
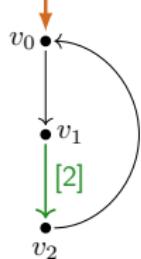
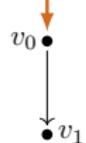


**LEE**

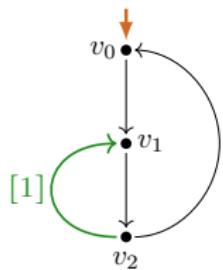
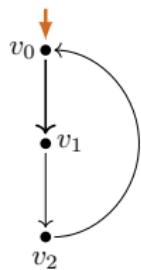


**LEE**

# LEE holds / Recording loop elimination

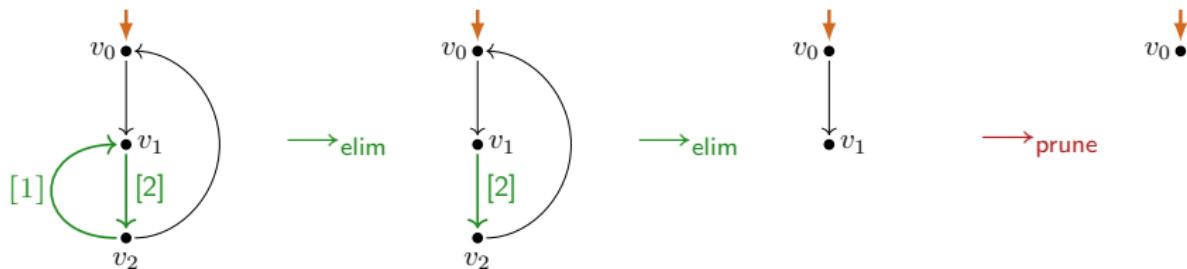
 $\rightarrow$  elim $\rightarrow$  elim $\rightarrow$  prune

LEE

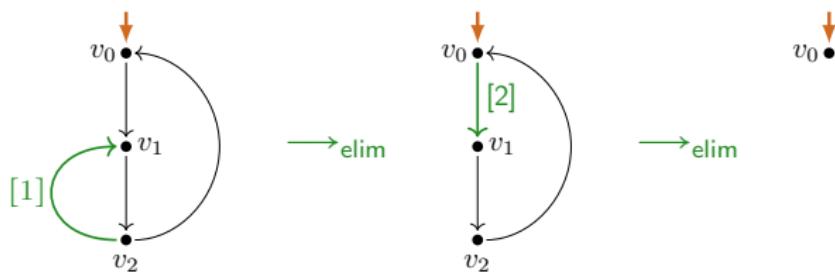
 $\rightarrow$  elim $\rightarrow$  elim

LEE

# LEE holds / Recording loop elimination

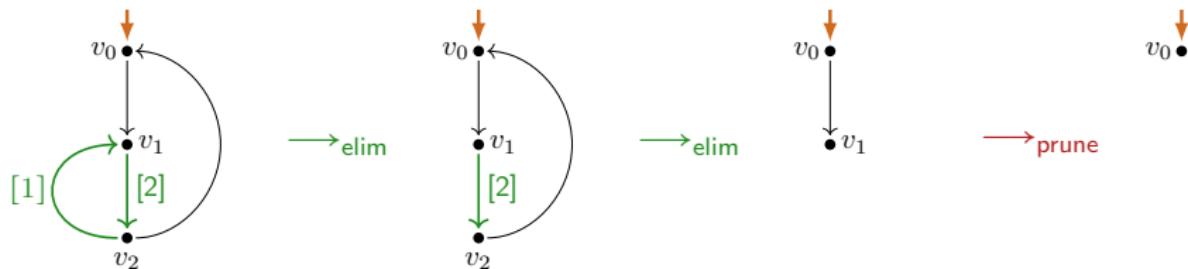


LEE

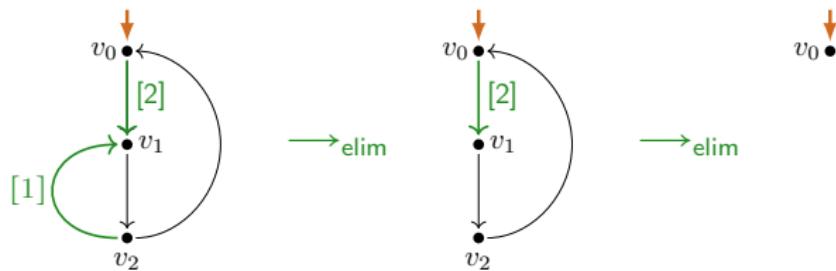


LEE

# LEE holds / Recording loop elimination

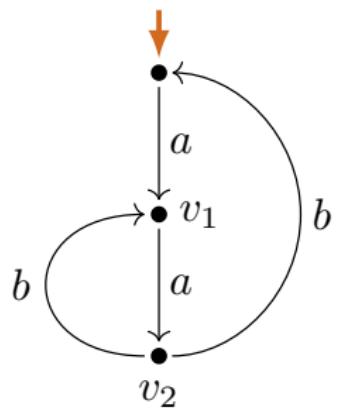


**LEE**



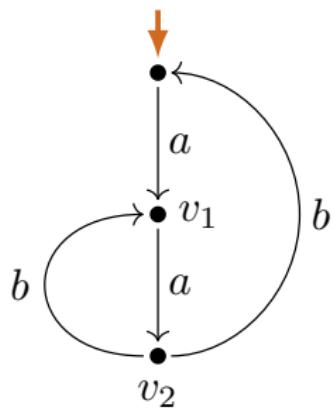
**LEE**

# LEE-witness



# LEE-witness

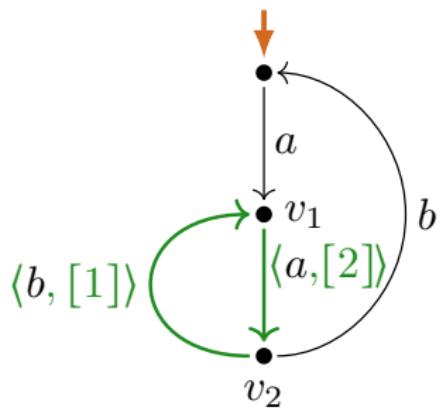
loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:



# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

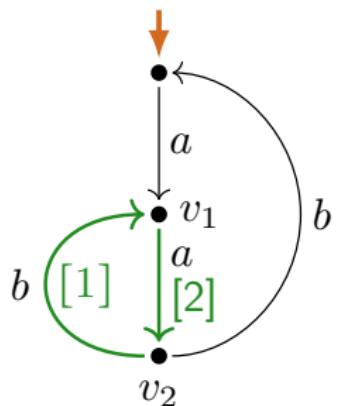
► entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ ,



# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

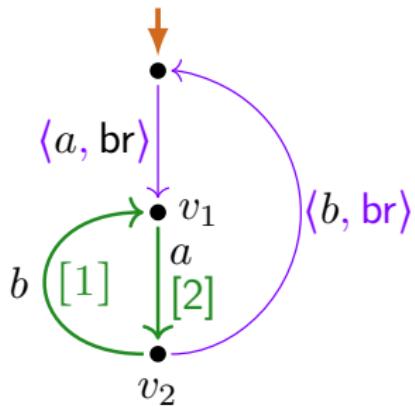
► entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a} [n]$ ,



# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

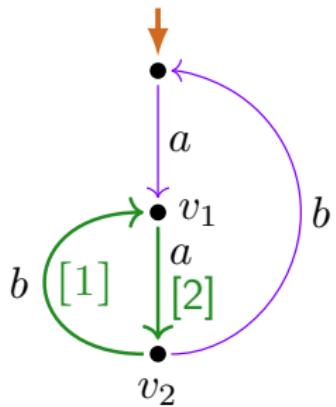
- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a} [n]$ ,
- ▶ branch steps  $\xrightarrow{(a,br)}$ ,



# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

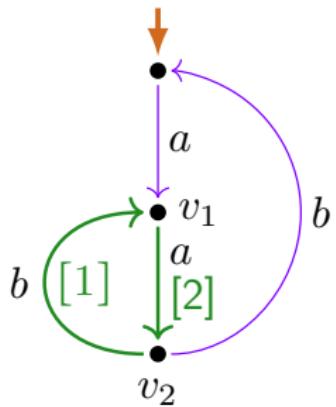
- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a} [n]$ ,
- ▶ branch steps  $\xrightarrow{(a,br)}$ , written  $\xrightarrow{a}_{br}$  or  $\xrightarrow{a}$ .



# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a} [n]$ ,
- ▶ branch steps  $\xrightarrow{(a,br)}$ , written  $\xrightarrow{a}_{br}$  or  $\xrightarrow{a}$ .



## Definition

A loop–branch labeling is a LEE-witness, if:

L1.

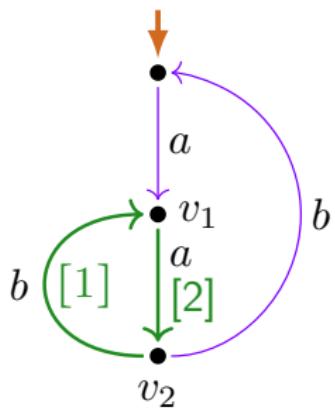
L2.

L3.

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a}_{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,br)}$ , written  $\xrightarrow{a}_{br}$  or  $\xrightarrow{a}$ .



## Definition

A loop–branch labeling is a LEE-witness, if:

L1.

L2.

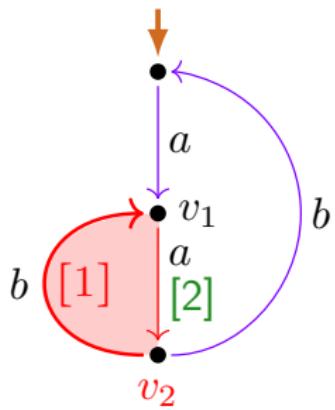
L3.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [>n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{br}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a}_{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,br)}$ , written  $\xrightarrow{a}_{br}$  or  $\xrightarrow{a}$ .



$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{br, [>1]})$$

## Definition

A loop–branch labeling is a LEE-witness, if:

L1.

L2.

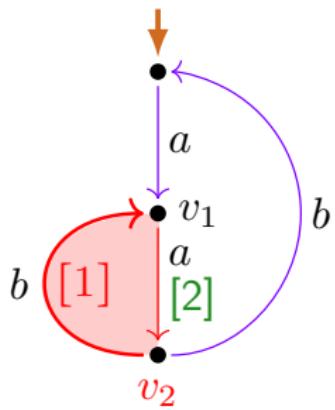
L3.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [>n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{br}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,br)}$ , written  $\xrightarrow[a]{br}$  or  $\xrightarrow[a]$ .



$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{br, [>1]})$   
is loop subchart

## Definition

A loop–branch labeling is a LEE-witness, if:

L1.

L2.

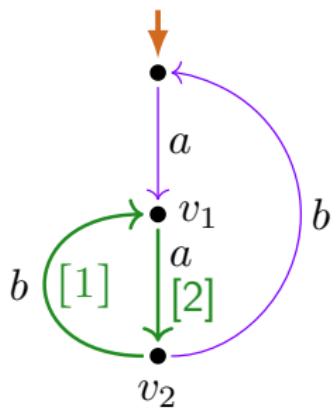
L3.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [>n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{br}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a}_{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,br)}$ , written  $\xrightarrow{a}_{br}$  or  $\xrightarrow{a}$ .



## Definition

A loop–branch labeling is a LEE-witness, if:

L1.

L2.

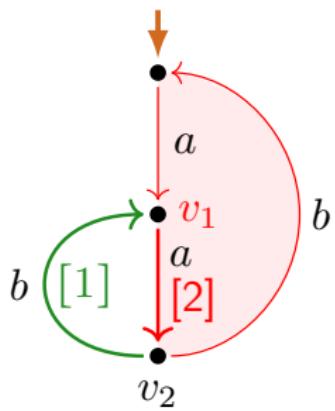
L3.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [m]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{br}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,br)}$ , written  $\xrightarrow[a]{br}$  or  $\xrightarrow[a]$ .



$$\mathcal{L}(v_1, \xrightarrow{[2]}, \xrightarrow{br, [>2]})$$

## Definition

A loop–branch labeling is a LEE-witness, if:

L1.

L2.

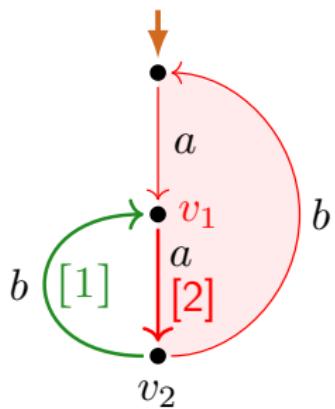
L3.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [>n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{br}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,\text{br})}$ , written  $\xrightarrow[a]{\text{br}}$  or  $\xrightarrow[a]$ .



$\mathcal{L}(v_1, \xrightarrow{[2]}, \xrightarrow{\text{br},[>2]})$   
is loop subchart

## Definition

A loop–branch labeling is a LEE-witness, if:

L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) \text{ is a loop subchart} \right)$ .

L2.

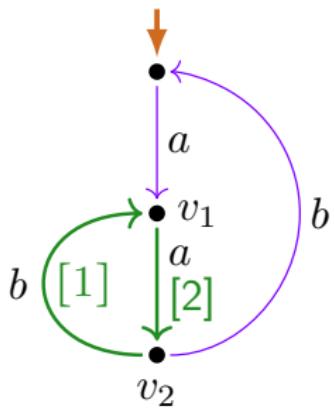
L3.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{\text{br}}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,\text{br})}$ , written  $\xrightarrow[a]{\text{br}}$  or  $\xrightarrow[a]$ .



## Definition

A loop–branch labeling is a LEE-witness, if:

L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) \text{ is a loop subchart} \right)$ .

L2.

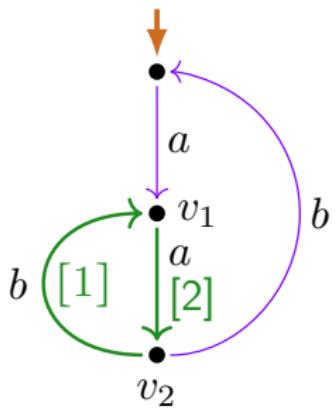
L3.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{\text{br}}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,\text{br})}$ , written  $\xrightarrow[a]{\text{br}}$  or  $\xrightarrow[a]$ .



## Definition

A loop–branch labeling is a LEE-witness, if:

L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) \text{ is a loop subchart} \right)$ .

L2. No infinite  $\xrightarrow{\text{br}}$  path from start vertex.

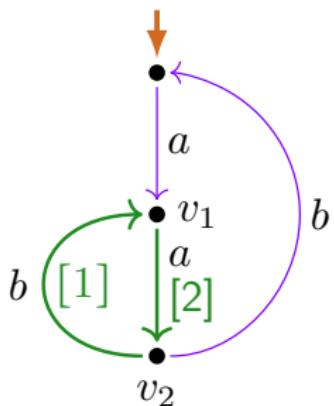
L3.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{\text{br}}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,\text{br})}$ , written  $\xrightarrow[a]{\text{br}}$  or  $\xrightarrow[a]$ .



## Definition

A loop–branch labeling is a LEE-witness, if:

L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br}, [n]}) \text{ is a loop subchart} \right)$ .

L2. No infinite  $\xrightarrow{\text{br}}$  path from start vertex.

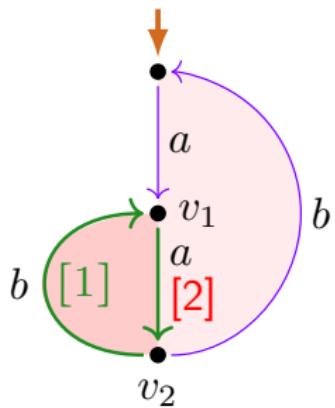
L3. Loop subcharts contained in other loop subcharts have different entry-step levels.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br}, [n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{\text{br}}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a,[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,\text{br})}$ , written  $\xrightarrow{a,\text{br}}$  or  $\xrightarrow{a}$ .



## Definition

A loop–branch labeling is a **LEE-witness**, if:

L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) \text{ is a loop subchart} \right)$ .

L2. No infinite  $\xrightarrow{\text{br}}$  path from **start vertex**.

L3. Loop subcharts contained in other loop subcharts have **different entry-step levels**.

$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{\text{br},[>1]})$$

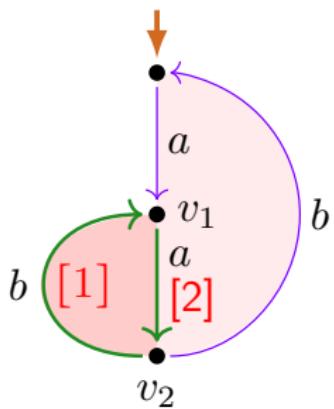
$$\mathcal{L}(v_1, \xrightarrow{[2]}, \xrightarrow{\text{br},[>2]})$$

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) :=$  subchart induced by entry steps  $\xrightarrow{[n]}$  from  $v$  followed by branch steps  $\xrightarrow{\text{br}}$  or entry steps  $\xrightarrow{[m]}$  with  $m > n$ , until  $v$  is reached again

# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a}_{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,\text{br})}$ , written  $\xrightarrow{a}_{\text{br}}$  or  $\xrightarrow{a}$ .



$$\begin{aligned}\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{\text{br},[>1]}) \\ \mathcal{L}(v_1, \xrightarrow{[2]}, \xrightarrow{\text{br},[>2]})\end{aligned}$$

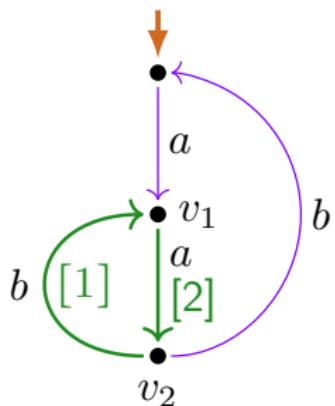
## Definition

A loop–branch labeling is a LEE-witness, if:

- L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) \text{ is a loop subchart} \right)$ .
- L2. No infinite  $\xrightarrow{\text{br}}$  path from start vertex.
- L3.  $\mathcal{L}(w_i, \xrightarrow{[n_i]}, \xrightarrow{\text{br},[>n_i]})$  for  $i \in \{1, 2\}$  loop charts  
 $\wedge w_1 \neq w_2 \wedge w_1 \in \mathcal{L}(w_2, \dots, \dots) \implies n_1 \neq n_2$ .

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) :=$  subchart induced  
 by entry steps  $\xrightarrow{[n]}$  from  $v$   
 followed by branch steps  $\xrightarrow{\text{br}}$   
 or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
 until  $v$  is reached again

# LEE-witness



LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a,[n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow{a,[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a,\text{br})}$ , written  $\xrightarrow{a,\text{br}}$  or  $\xrightarrow{a}$ .

## Definition

A loop–branch labeling is a LEE-witness, if:

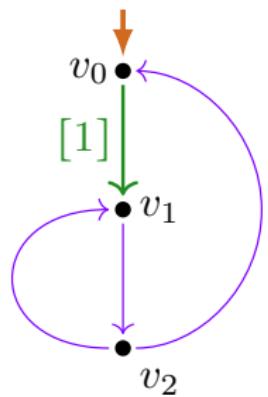
L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) \text{ is a loop subchart} \right)$ .

L2. No infinite  $\xrightarrow{\text{br}}$  path from start vertex.

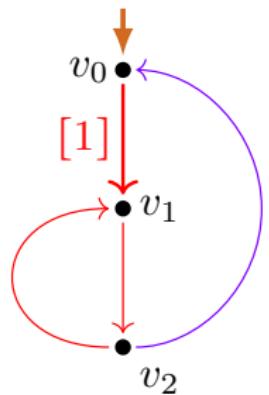
L3.  $\mathcal{L}(w_i, \xrightarrow{[n_i]}, \xrightarrow{\text{br},[>n_i]})$  for  $i \in \{1, 2\}$  loop charts  
 $\wedge w_1 \neq w_2 \wedge w_1 \in \mathcal{L}(w_2, \dots, \dots) \implies n_1 \neq n_2$ .

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br},[>n]}) :=$  subchart induced  
 by entry steps  $\xrightarrow{[n]}$  from  $v$   
 followed by branch steps  $\xrightarrow{\text{br}}$   
 or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
 until  $v$  is reached again

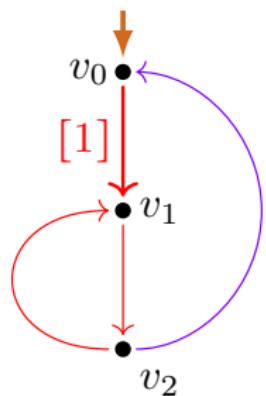
# LEE-witness ?



# LEE-witness ?



# LEE-witness ?



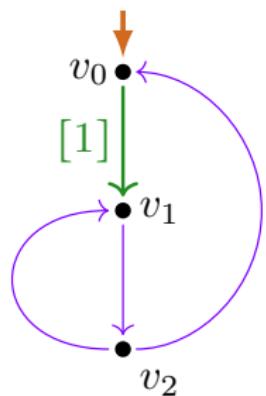
no!

(L1.) violated:

$$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br}, [>1]})$$

not a loop chart

# LEE-witness ?



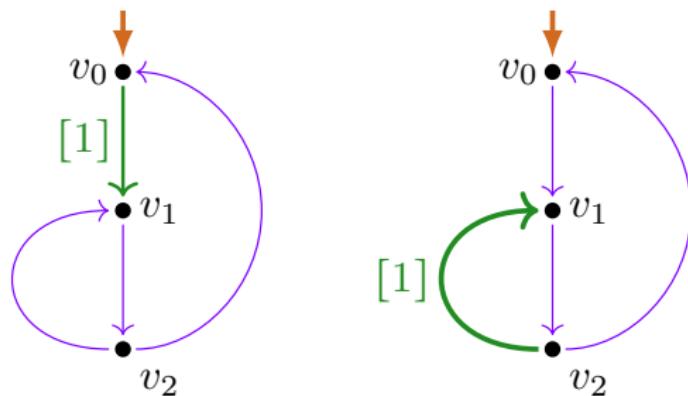
no!

(L1.) violated:

$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br}, [>1]})$

not a loop chart

# LEE-witness ?



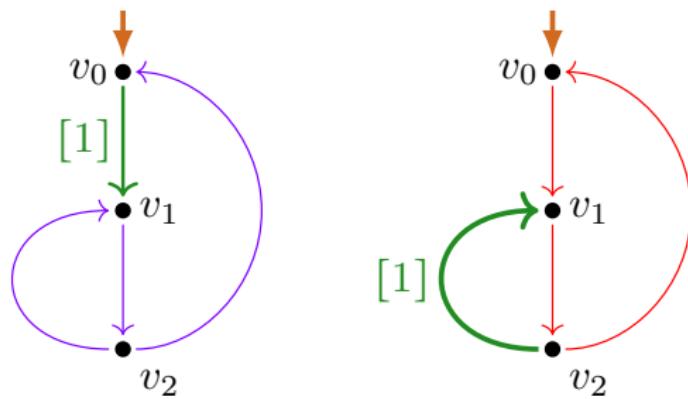
no!

(L1.) violated:

$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br}, [>1]})$

not a loop chart

# LEE-witness ?



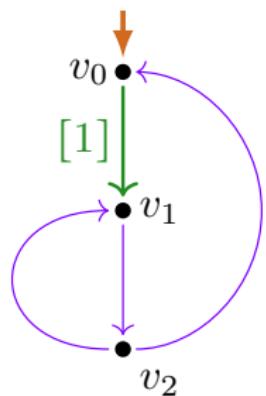
no!

(L1.) violated:

$$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br}, [>1]})$$

not a loop chart

# LEE-witness ?

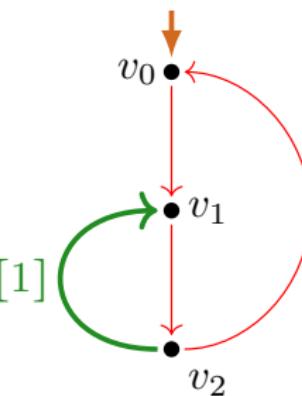


no!

(L1.) violated:

$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br}, [>1]})$

not a loop chart



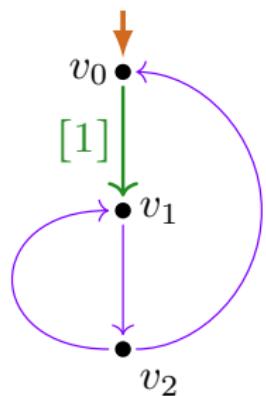
no!

(L2.) violated:

infinite  $\rightarrow_{\text{br}}$  path

from start vertex

# LEE-witness ?

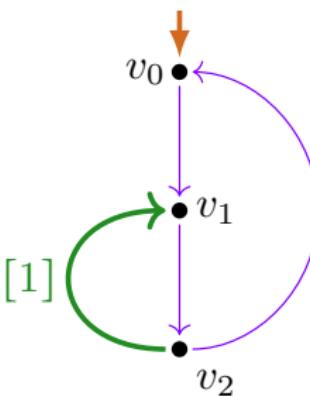


no!

(L1.) violated:

$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br},[>1]})$

not a loop chart



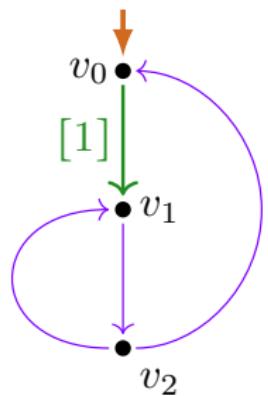
no!

(L2.) violated:

infinite  $\rightarrow_{\text{br}}$  path

from start vertex

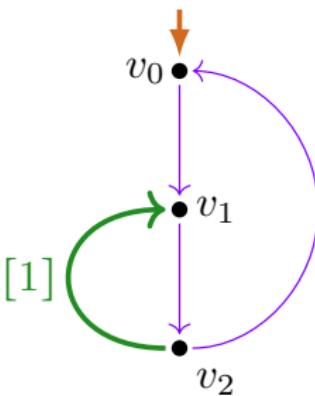
# LEE-witness ?



no!

(L1.) violated:

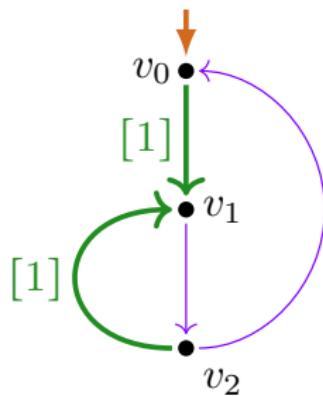
$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br}, [>1]})$   
not a loop chart



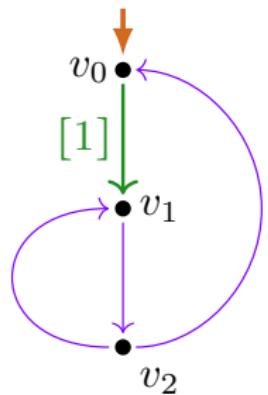
no!

(L2.) violated:

infinite  $\rightarrow_{\text{br}}$  path  
from start vertex



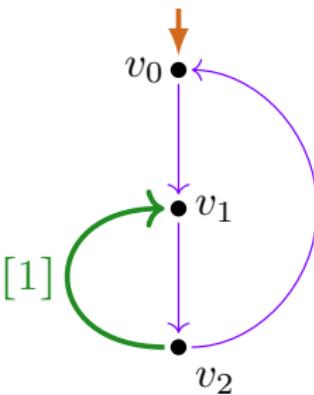
# LEE-witness ?



no!

(L1.) violated:

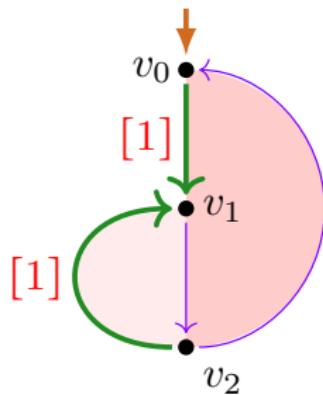
$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br},[>1]})$   
not a loop chart



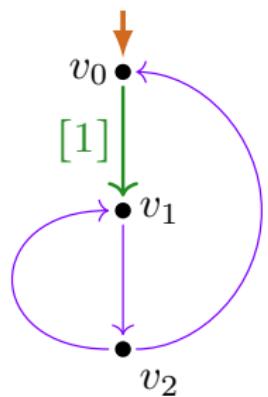
no!

(L2.) violated:

infinite  $\rightarrow_{\text{br}}$  path  
from start vertex



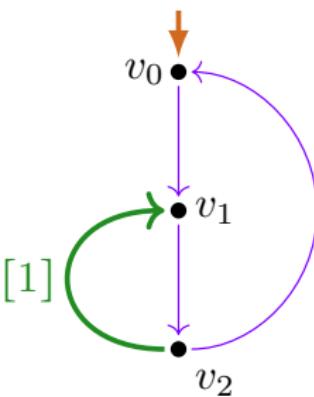
# LEE-witness ?



no!

(L1.) violated:

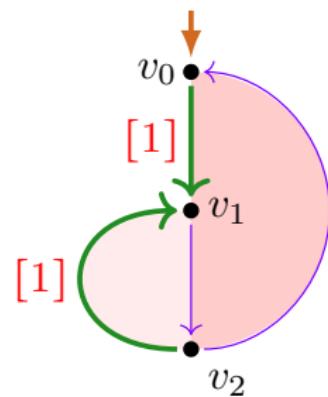
$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br},[>1]})$   
not a loop chart



no!

(L2.) violated:

infinite  $\rightarrow_{\text{br}}$  path  
from start vertex

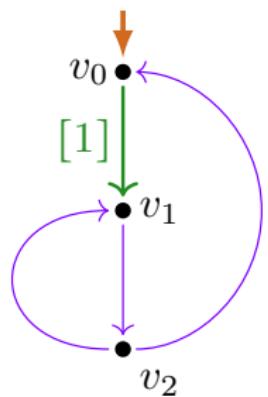


no!

(L3.) violated:

overlapping loop charts  
have same level

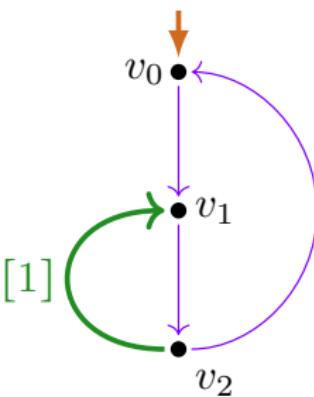
# LEE-witness ?



no!

(L1.) violated:

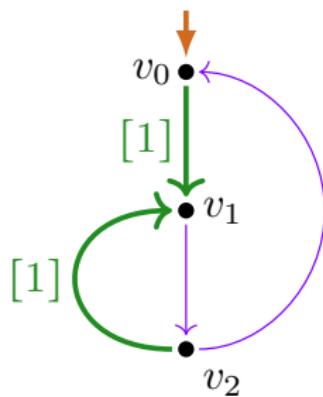
$\mathcal{L}(v_0, \rightarrow_{[1]}, \rightarrow_{\text{br}, [>1]})$   
not a loop chart



no!

(L2.) violated:

infinite  $\rightarrow_{\text{br}}$  path  
from start vertex

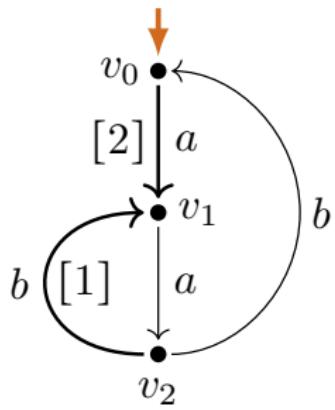


no!

(L3.) violated:

overlapping loop charts  
have same level

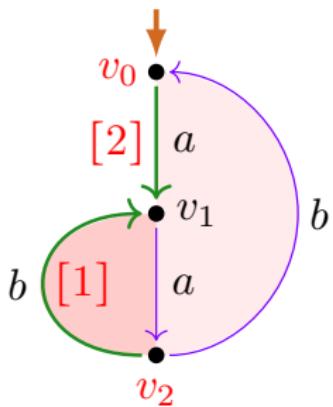
# LEE-witness ?



# LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a, [n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a, br)}$ , written  $\xrightarrow[a]{br}$  or  $\xrightarrow[a]$ .



$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{br, [>1]})$$

$$\mathcal{L}(v_0, \xrightarrow{[2]}, \xrightarrow{br, [>2]})$$

LEE-witness

## Definition

A loop–branch labeling is a **LEE-witness**, if:

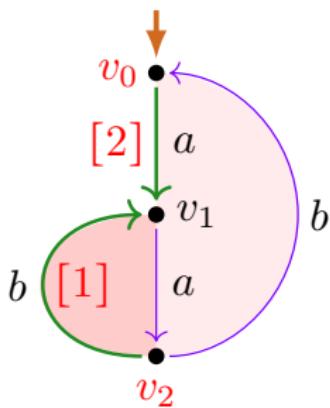
- L1.  $\forall n \in \mathbb{N} \forall v \in V \left( \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [>n]}) \text{ is a loop subchart, or trivial} \right)$ .
- L2. No infinite  $\xrightarrow{br}$  path from **start vertex**.
- L3.  $\mathcal{L}(w_i, \xrightarrow{[n_i]}, \xrightarrow{br, [>n_i]})$  for  $i \in \{1, 2\}$  loop charts  
 $\wedge w_1 \neq w_2 \wedge w_1 \in \mathcal{L}(w_2, \dots, \dots) \implies n_1 \neq n_2$ .

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [>n]}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{br}$   
or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
until  $v$  is reached again

# Layered LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a, [n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a, br)}$ , written  $\xrightarrow[a]{br}$  or  $\xrightarrow[a]$ .



$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{br, [>1]})$$

$$\mathcal{L}(v_0, \xrightarrow{[2]}, \xrightarrow{br, [>2]})$$

## Definition

A loop–branch labeling is a **layered LEE-witness**, if:

I-L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [>n]}) \text{ is a loop subchart} \right)$ .

I-L2. No infinite  $\xrightarrow{br}$  path from **start vertex**.

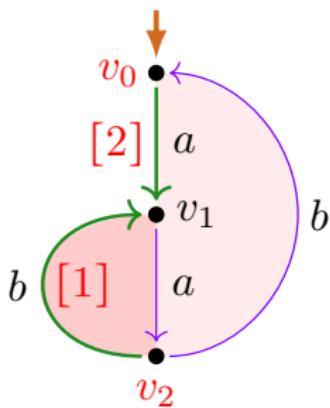
I-L3.  $\mathcal{L}(w_i, \xrightarrow{[n_i]}, \xrightarrow{br, [>n_i]})$  for  $i \in \{1, 2\}$  loop charts  
 $\wedge w_1 \neq w_2 \wedge w_1 \in \mathcal{L}(w_2, \dots, \dots) \implies n_1 < n_2$ .

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br, [>n]}) :=$  subchart induced  
 by entry steps  $\xrightarrow{[n]}$  from  $v$   
 followed by branch steps  $\xrightarrow{br}$   
 or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
 until  $v$  is reached again

# Layered LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a, [n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a, br)}$ , written  $\xrightarrow[a]{br}$  or  $\xrightarrow[a]$ .



$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{br, [>1]})$$

$$\mathcal{L}(v_0, \xrightarrow{[2]}, \xrightarrow{br, [>2]})$$

## Definition

A loop–branch labeling is a **layered LEE-witness**, if:

I-L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br}) \text{ is a loop subchart} \right)$ .

I-L2. No infinite  $\xrightarrow{br}$  path from **start vertex**.

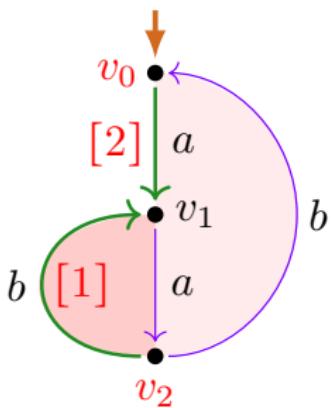
I-L3.  $\mathcal{L}(w_i, \xrightarrow{[n_i]}, \xrightarrow{br})$  for  $i \in \{1, 2\}$  loop charts  
 $\wedge w_1 \neq w_2 \wedge w_1 \in \mathcal{L}(w_2, \dots, \dots) \implies n_1 < n_2$ .

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br}) :=$  subchart induced  
 by entry steps  $\xrightarrow{[n]}$  from  $v$   
 followed by branch steps  $\xrightarrow{br}$   
 or entry steps  $\xrightarrow{[m]}$  with  $m > n$ ,  
 until  $v$  is reached again

# Layered LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{\langle a, [n] \rangle}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{\langle a, \text{br} \rangle}$ , written  $\xrightarrow[a]{\text{br}}$  or  $\xrightarrow[a]$ .



$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{\text{br}})$$

$$\mathcal{L}(v_0, \xrightarrow{[2]}, \xrightarrow{\text{br}})$$

## Definition

A loop–branch labeling is a **layered LEE-witness**, if:

I-L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br}}) \text{ is a loop subchart} \right).$

I-L2. No infinite  $\xrightarrow{\text{br}}$  path from **start vertex**.

I-L3.  $\mathcal{L}(w_i, \xrightarrow{[n_i]}, \xrightarrow{\text{br}})$  for  $i \in \{1, 2\}$  loop charts  
 $\wedge w_1 \neq w_2 \wedge w_1 \in \mathcal{L}(w_2, \dots, \dots) \implies n_1 < n_2.$

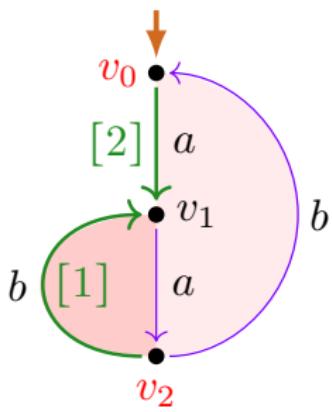
$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br}}) :=$  subchart induced  
 by entry steps  $\xrightarrow{[n]}$  from  $v$   
 followed by branch steps  $\xrightarrow{\text{br}}$

until  $v$  is reached again

# Layered LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{\langle a, [n] \rangle}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{\langle a, \text{br} \rangle}$ , written  $\xrightarrow[a]{\text{br}}$  or  $\xrightarrow[a]$ .



$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{\text{br}})$$

$$\mathcal{L}(v_0, \xrightarrow{[2]}, \xrightarrow{\text{br}})$$

## Definition

A loop–branch labeling is a **layered LEE-witness**, if:

I-L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br}}) \text{ is a loop subchart} \right)$ .

I-L2. No infinite  $\xrightarrow{\text{br}}$  path from **start vertex**.

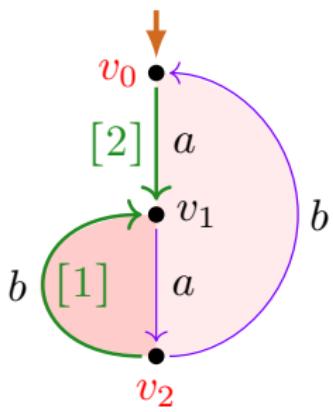
I-L3. A loop subchart generated by a vertex contained in another generated loop subchart has lower level.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br}}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{\text{br}}$

# Layered LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{\langle a, [n] \rangle}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{\langle a, \text{br} \rangle}$ , written  $\xrightarrow[a]{\text{br}}$  or  $\xrightarrow[a]$ .



$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{\text{br}})$$

$$\mathcal{L}(v_0, \xrightarrow{[2]}, \xrightarrow{\text{br}})$$

layered  
LEE-witness

## Definition

A loop–branch labeling is a **layered LEE-witness**, if:

I-L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br}}) \text{ is a loop subchart} \right)$ .

I-L2. No infinite  $\xrightarrow{\text{br}}$  path from **start vertex**.

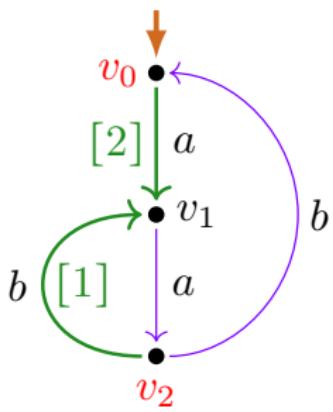
I-L3. A loop subchart generated by a vertex contained in another generated loop subchart has lower level.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{\text{br}}) :=$  subchart induced by entry steps  $\xrightarrow{[n]}$  from  $v$  followed by branch steps  $\xrightarrow{\text{br}}$

# Layered LEE-witness

loop–branch labeling: marking transitions  $\xrightarrow{a}$  as:

- ▶ entry steps  $\xrightarrow{(a, [n])}$  for  $n \in \mathbb{N}$ , written  $\xrightarrow[a]{[n]}$ ,
- ▶ branch steps  $\xrightarrow{(a, br)}$ , written  $\xrightarrow[a]{br}$  or  $\xrightarrow[a]$ .



$$\mathcal{L}(v_2, \xrightarrow{[1]}, \xrightarrow{br})$$

$$\mathcal{L}(v_0, \xrightarrow{[2]}, \xrightarrow{br})$$

layered  
LEE-witness

## Definition

A loop–branch labeling is a **layered LEE-witness**, if:

I-L1.  $\forall n \in \mathbb{N} \forall v \in V \left( v \xrightarrow{[n]} \Rightarrow \mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br}) \text{ is a loop subchart} \right)$ .

I-L2. No infinite  $\xrightarrow{br}$  path from **start vertex**.

I-L3. A loop subchart generated by a vertex contained in another generated loop subchart has lower level.

$\mathcal{L}(v, \xrightarrow{[n]}, \xrightarrow{br}) :=$  subchart induced  
by entry steps  $\xrightarrow{[n]}$  from  $v$   
followed by branch steps  $\xrightarrow{br}$

# LEE versus LEE-witness

## Theorem

For every process graph  $G$ :

$$\text{LEE}(G) \iff G \text{ has a LEE-witness.}$$

# LEE versus LEE-witness

## Theorem

For every process graph  $G$ :

$$\text{LEE}(G) \iff G \text{ has a LEE-witness.}$$

## Proof.

$\Rightarrow$ : record loop elimination

# LEE versus LEE-witness

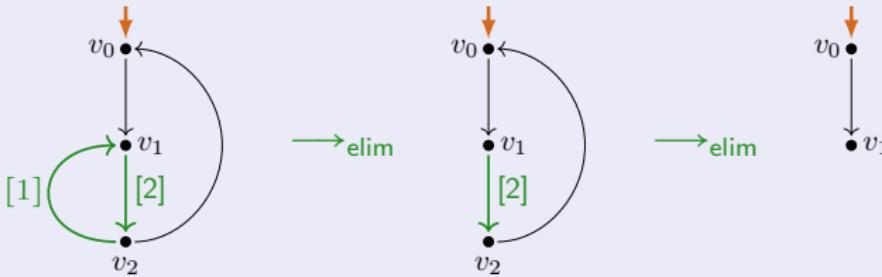
## Theorem

For every process graph  $G$ :

$$\text{LEE}(G) \iff G \text{ has a LEE-witness.}$$

## Proof.

- : record loop elimination
- ← : carry out loop-elimination as indicated in the LEE-witness,  
in *inside-out* direction, e.g.:



# LEE and (layered) LEE-witness

## Lemma

Every layered LEE-witness is a LEE-witness.

## Lemma

Every LEE-witness  $\widehat{G}$  of a process graph  $G$

can be transformed by an effective procedure (cut-elimination-like)  
into a layered LEE-witness  $\widehat{G}'$  of  $G$ .

# LEE and (layered) LEE-witness

## Lemma

Every layered LEE-witness is a LEE-witness.

## Lemma

Every LEE-witness  $\widehat{G}$  of a process graph  $G$

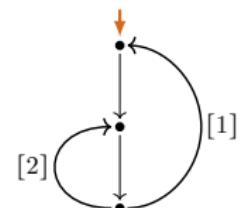
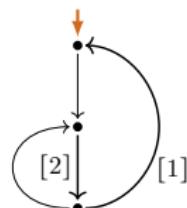
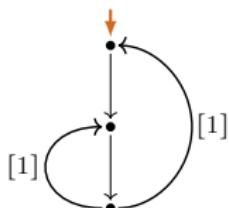
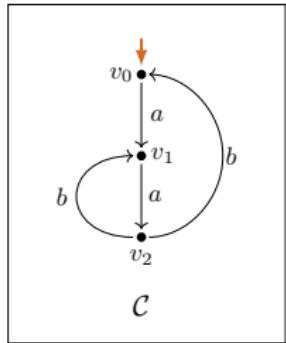
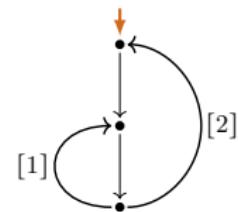
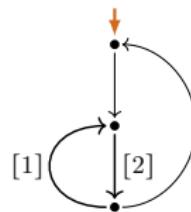
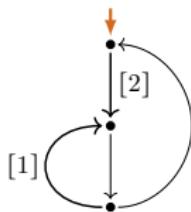
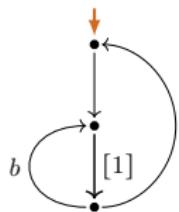
can be transformed by an effective procedure (cut-elimination-like)  
into a layered LEE-witness  $\widehat{G}'$  of  $G$ .

## Lemma

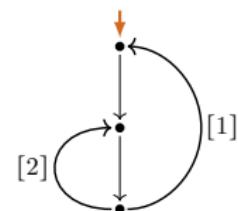
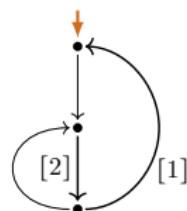
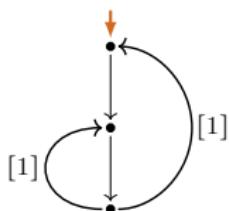
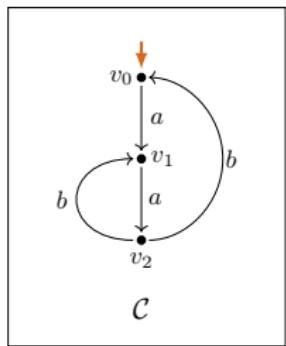
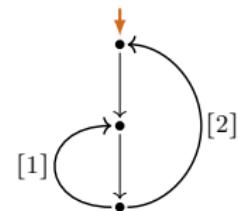
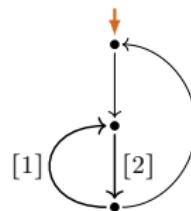
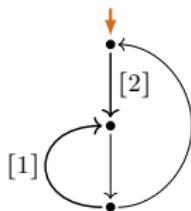
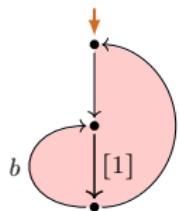
For every process graph  $G$  the following are equivalent:

- (i)  $\text{LEE}(G)$ .
- (ii)  $G$  has a LEE-witness.
- (iii)  $G$  has a layered LEE-witness.

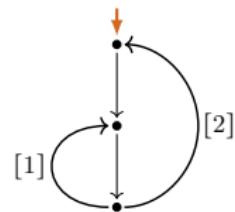
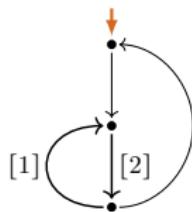
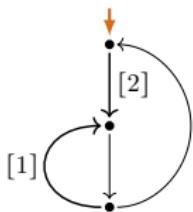
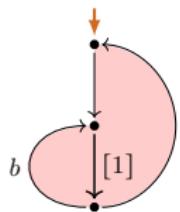
# 7 LEE-witnesses



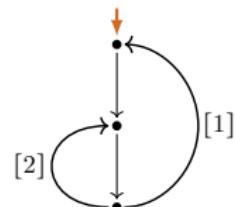
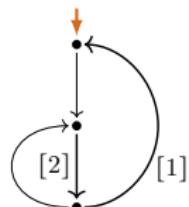
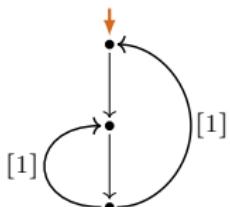
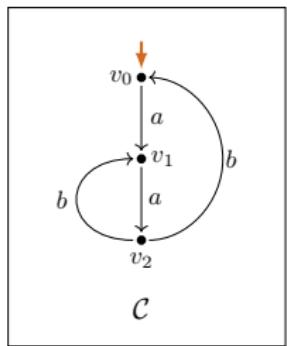
# 7 LEE-witnesses



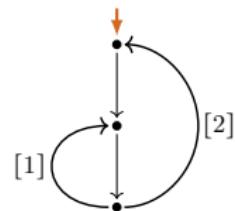
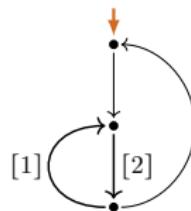
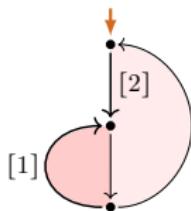
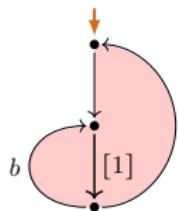
# 7 LEE-witnesses



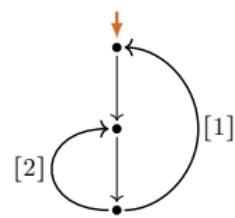
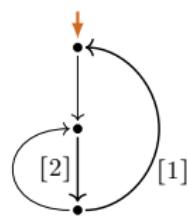
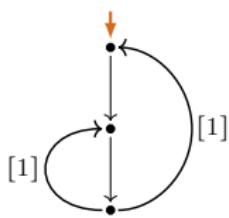
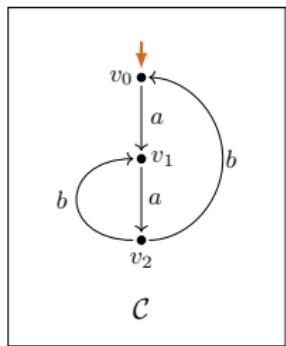
layered



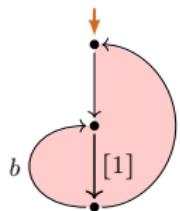
# 7 LEE-witnesses



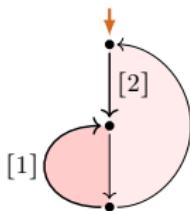
layered



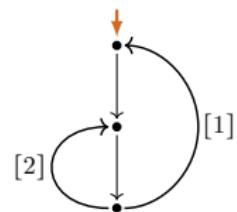
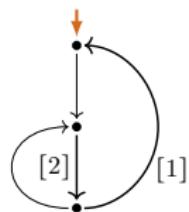
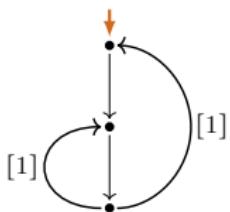
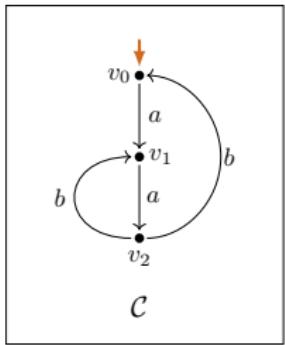
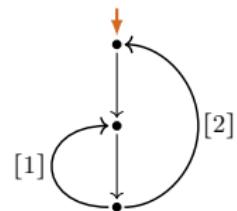
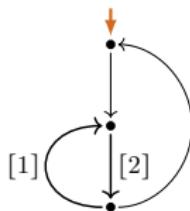
# 7 LEE-witnesses



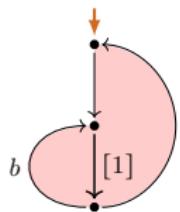
layered



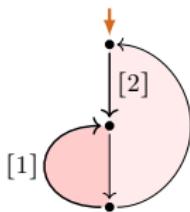
layered



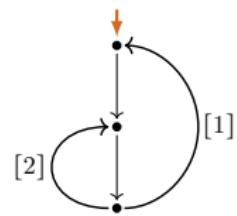
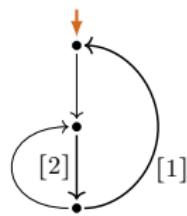
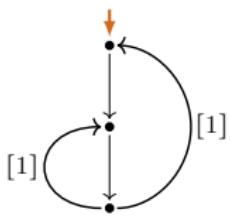
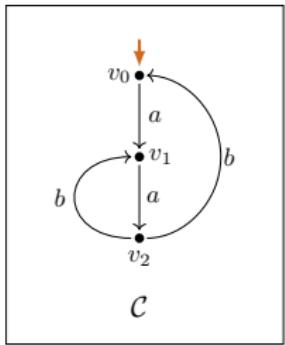
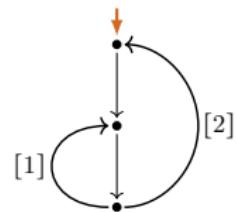
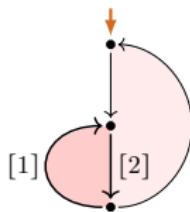
# 7 LEE-witnesses



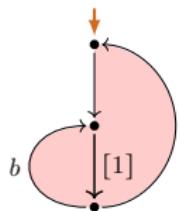
layered



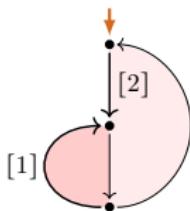
layered



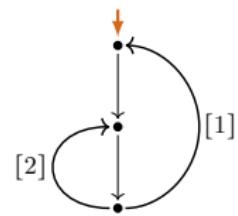
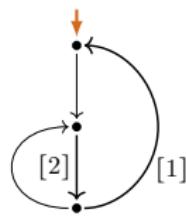
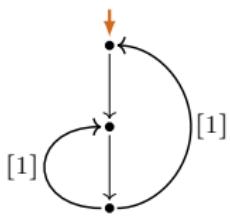
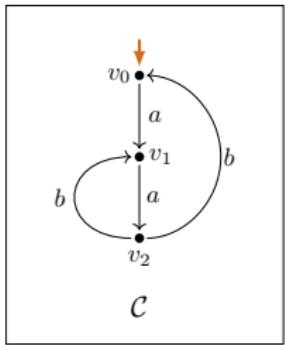
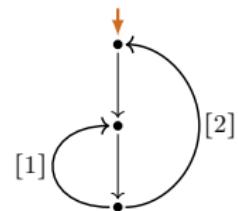
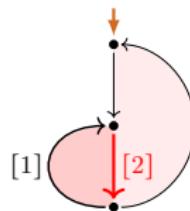
# 7 LEE-witnesses



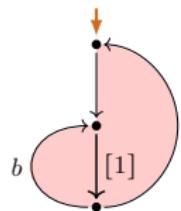
layered



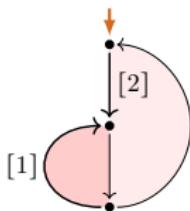
layered



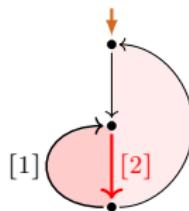
# 7 LEE-witnesses



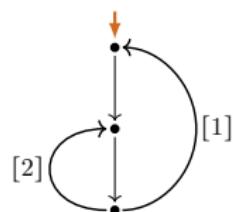
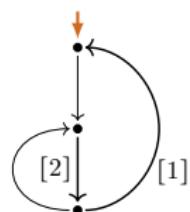
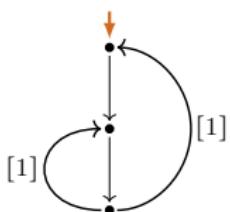
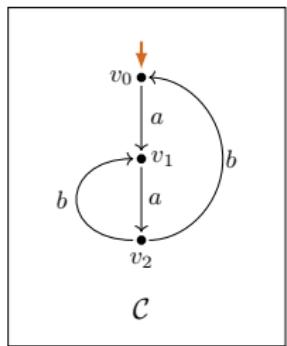
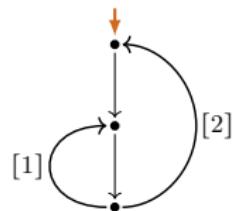
layered



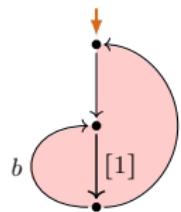
layered



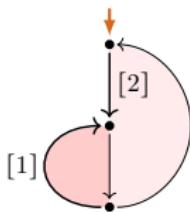
not layered



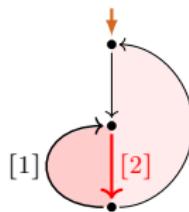
# 7 LEE-witnesses



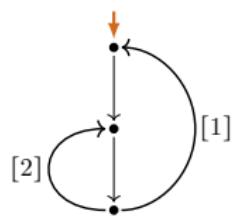
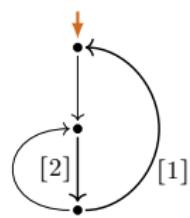
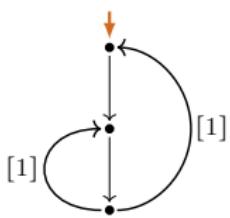
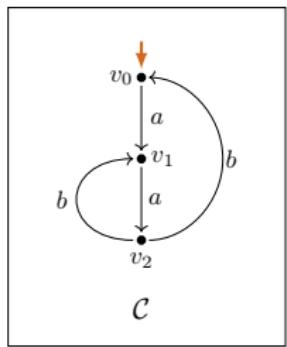
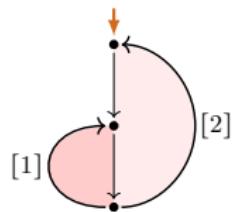
layered



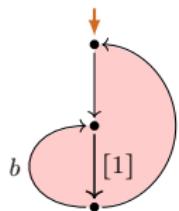
layered



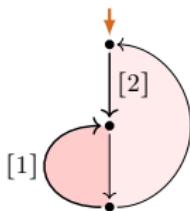
not layered



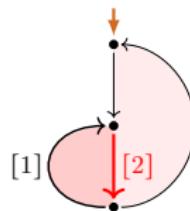
# 7 LEE-witnesses



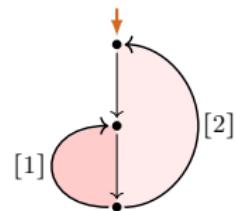
layered



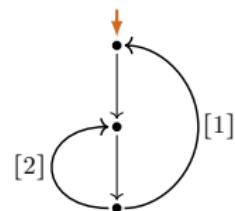
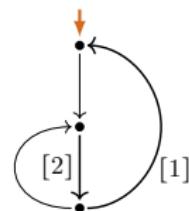
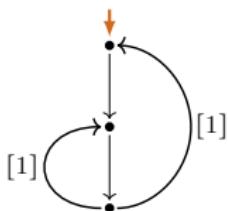
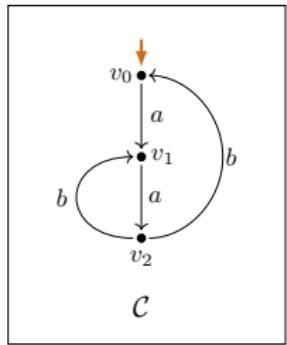
layered



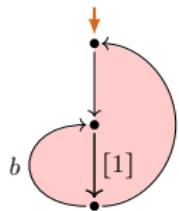
not layered



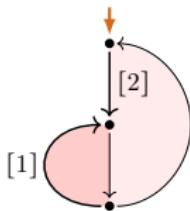
layered



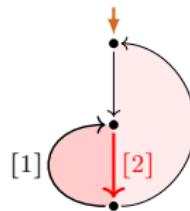
# 7 LEE-witnesses



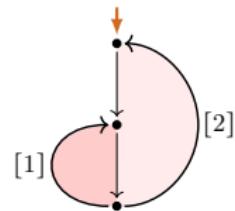
layered



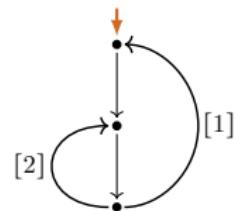
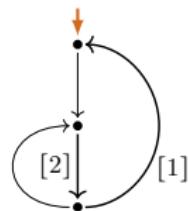
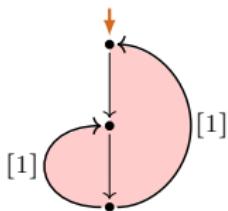
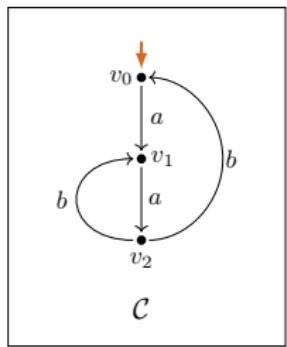
layered



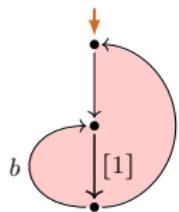
not layered



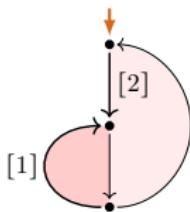
layered



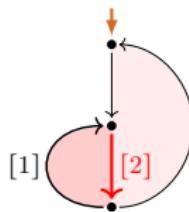
# 7 LEE-witnesses



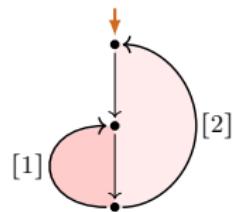
layered



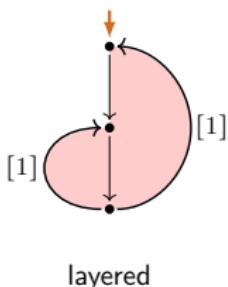
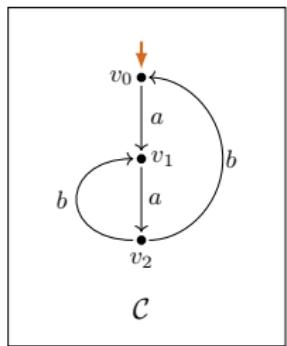
layered



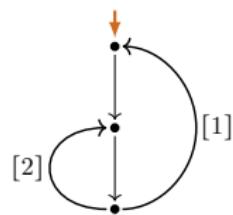
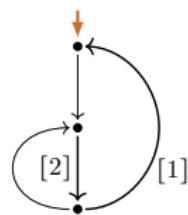
not layered



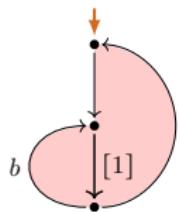
layered



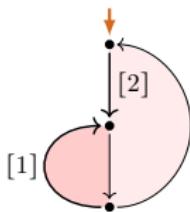
layered



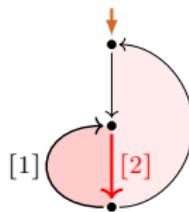
# 7 LEE-witnesses



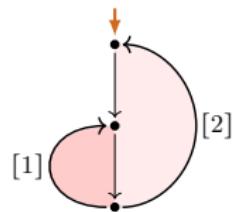
layered



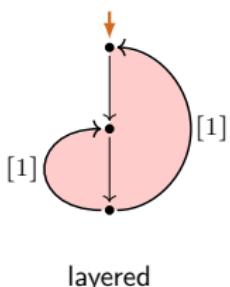
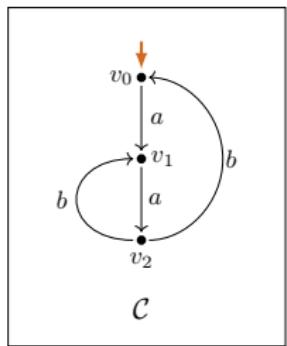
layered



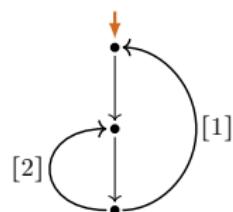
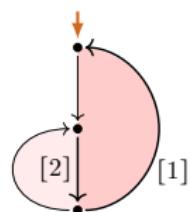
not layered



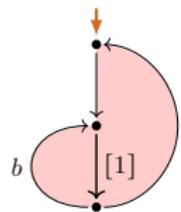
layered



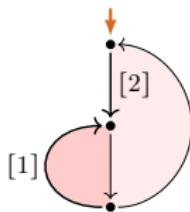
layered



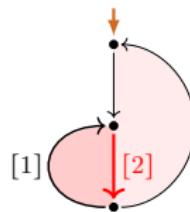
# 7 LEE-witnesses



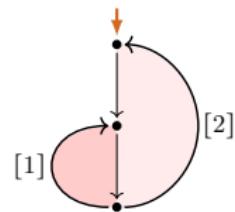
layered



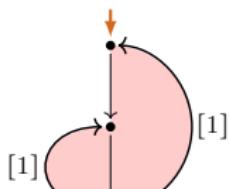
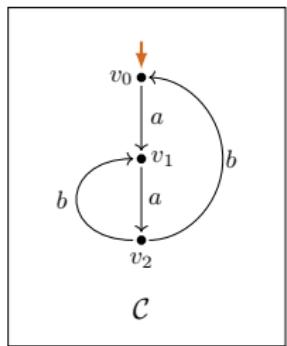
layered



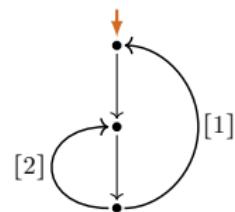
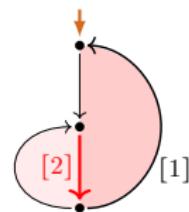
not layered



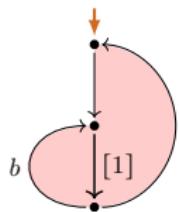
layered



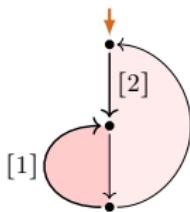
layered



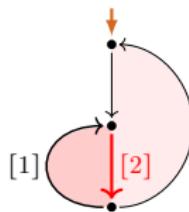
# 7 LEE-witnesses



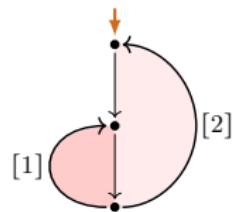
layered



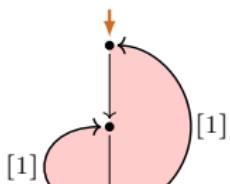
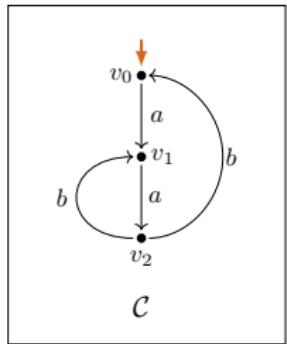
layered



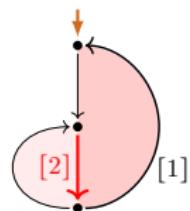
not layered



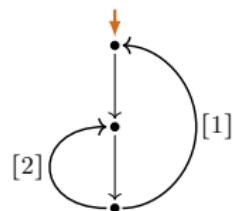
layered



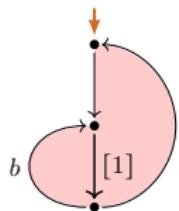
layered



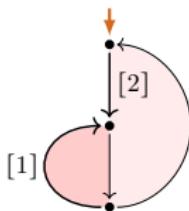
not layered



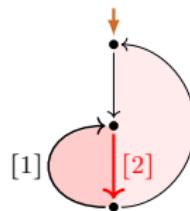
# 7 LEE-witnesses



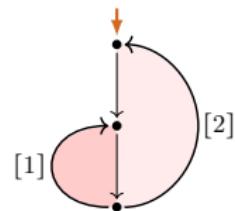
layered



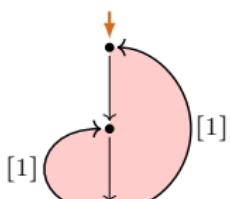
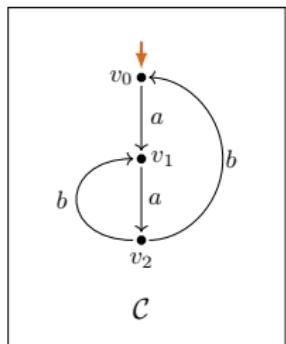
layered



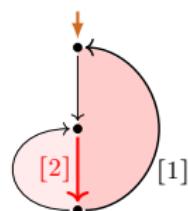
not layered



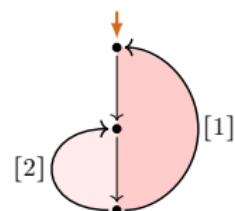
layered



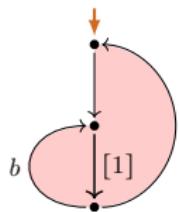
layered



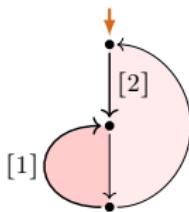
not layered



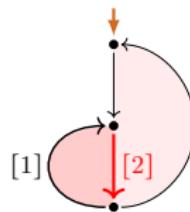
# 7 LEE-witnesses



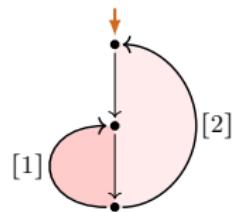
layered



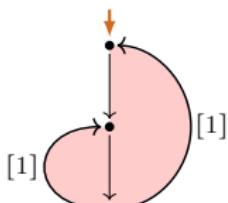
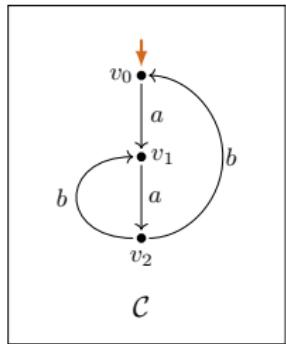
layered



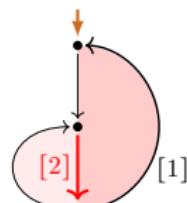
not layered



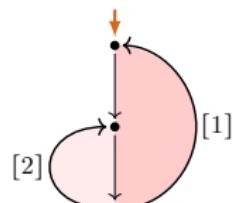
layered



layered

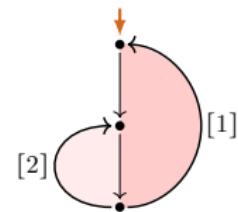
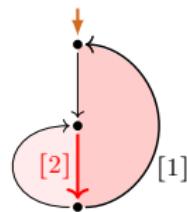
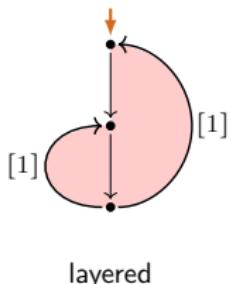
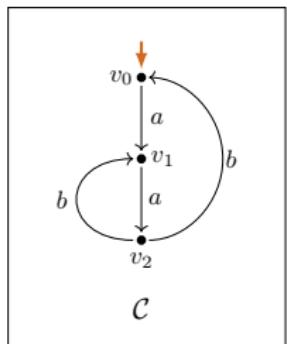
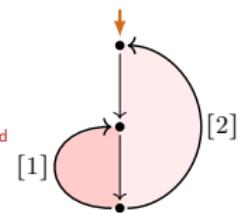
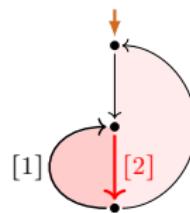
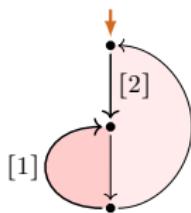
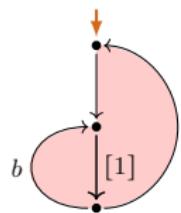


not layered

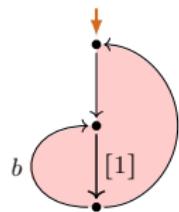


layered

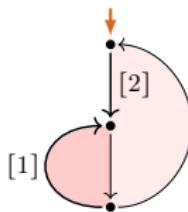
# 7 LEE-witnesses



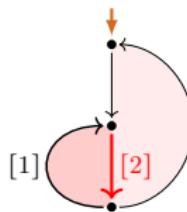
# 7 LEE-witnesses



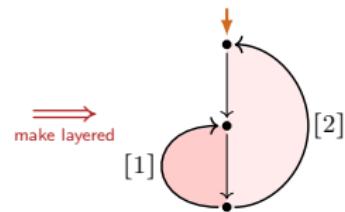
layered



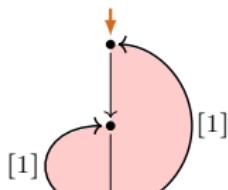
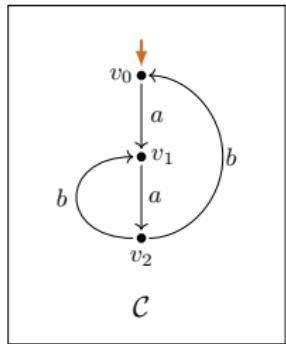
layered



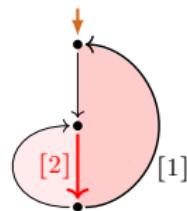
not layered



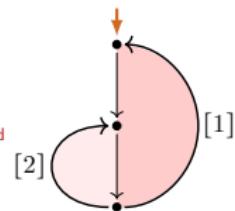
layered



layered

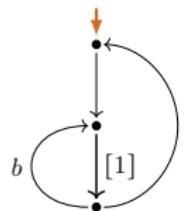


not layered

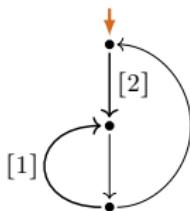


layered

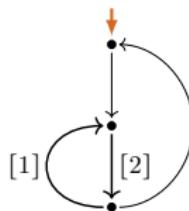
# 7 LEE-witnesses



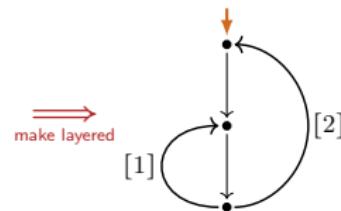
layered



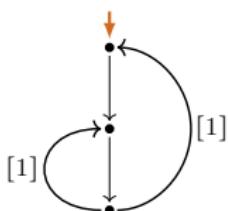
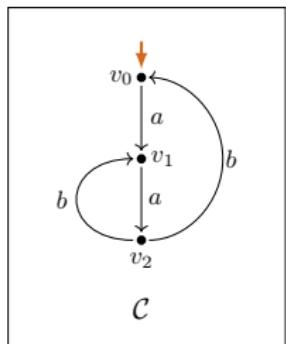
layered



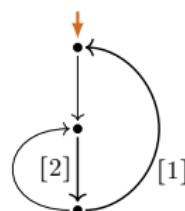
not layered



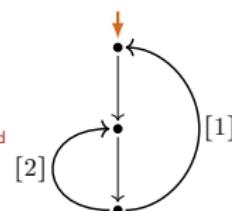
layered



layered



not layered



layered

# LEE under bisimulation?

# LEE under bisimulation

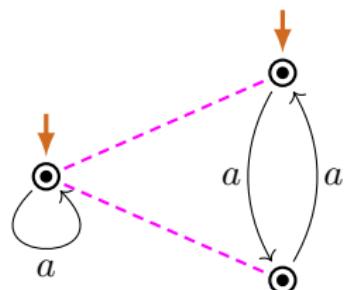
## Observation

- ▶ LEE is **not** invariant under bisimulation.

# LEE under bisimulation

## Observation

- ▶ LEE is **not** invariant under bisimulation.



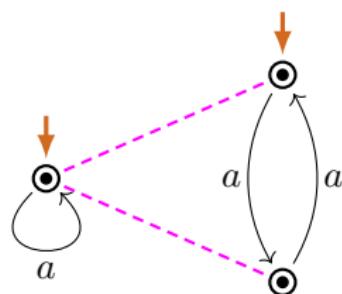
LEE

$\neg$ LEE

# LEE under bisimulation

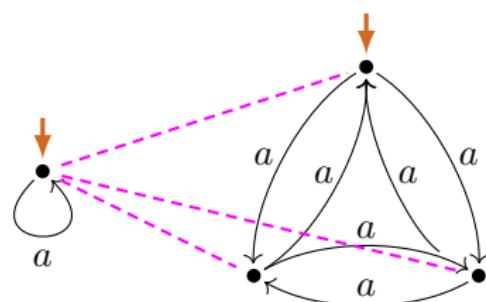
## Observation

- ▶ LEE is **not** invariant under bisimulation.



LEE

$\neg$ LEE



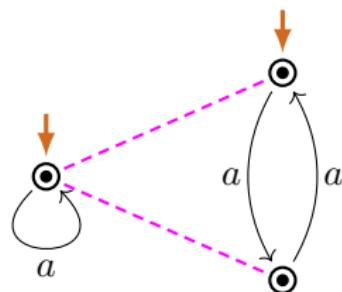
LEE

$\neg$ LEE

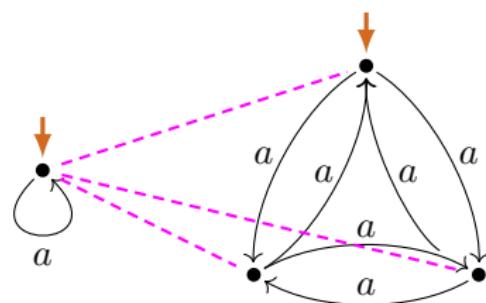
# LEE under bisimulation

## Observation

- ▶ LEE is **not** invariant under bisimulation.
- ▶ LEE is **not** preserved by converse functional bisimulation.



LEE



LEE

$\neg$ LEE

# LEE under functional bisimulation

## Lemma

(i) LEE is preserved by *functional bisimulations*:

$$\text{LEE}(G_1) \wedge G_1 \mathrel{\sqsupseteq} G_2 \implies \text{LEE}(G_2).$$

# LEE under functional bisimulation

## Lemma

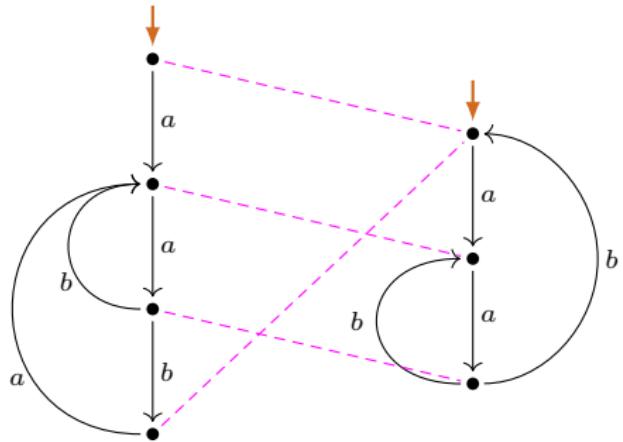
(i) LEE is preserved by *functional bisimulations*:

$$\text{LEE}(\textcolor{violet}{G}_1) \wedge \textcolor{violet}{G}_1 \mathrel{\sqsupseteq} \textcolor{violet}{G}_2 \implies \text{LEE}(\textcolor{violet}{G}_2).$$

## Proof (Idea).

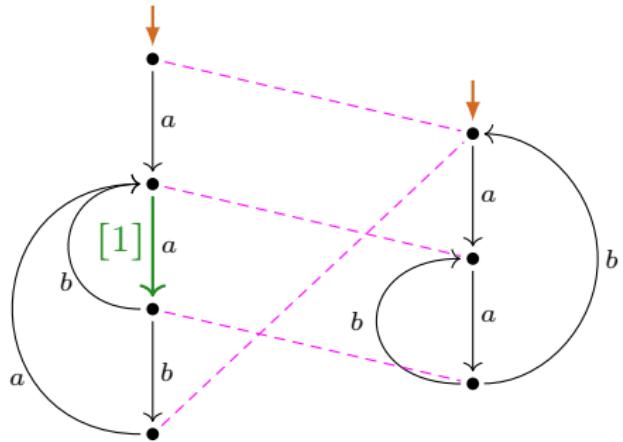
Use loop elimination in  $\textcolor{violet}{G}_1$  to carry out loop elimination in  $\textcolor{violet}{G}_2$ .

# Collapsing LEE-witnesses



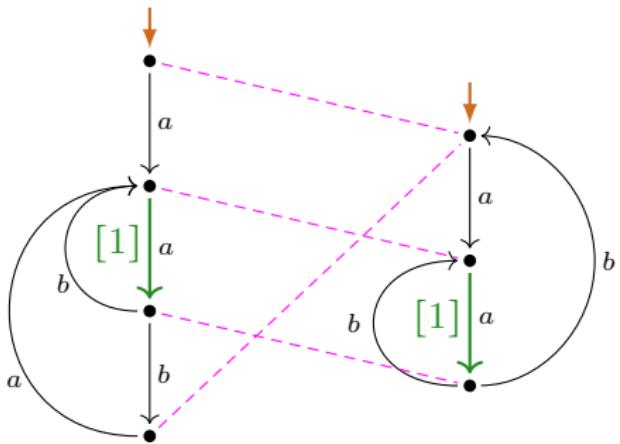
$$P(a(a(b+ba))^*0)$$

# Collapsing LEE-witnesses



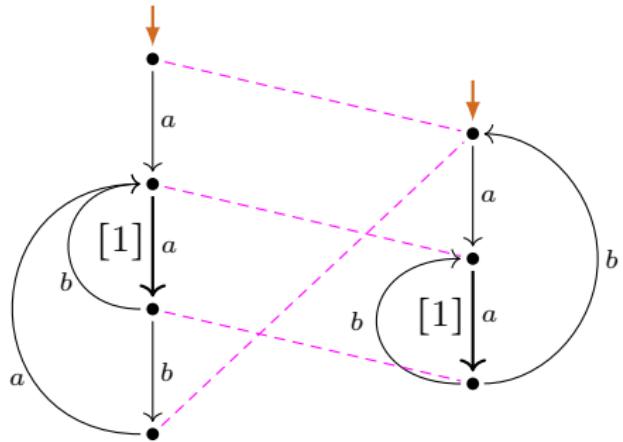
$$P(a(a(b+ba))^*0)$$

# Collapsing LEE-witnesses



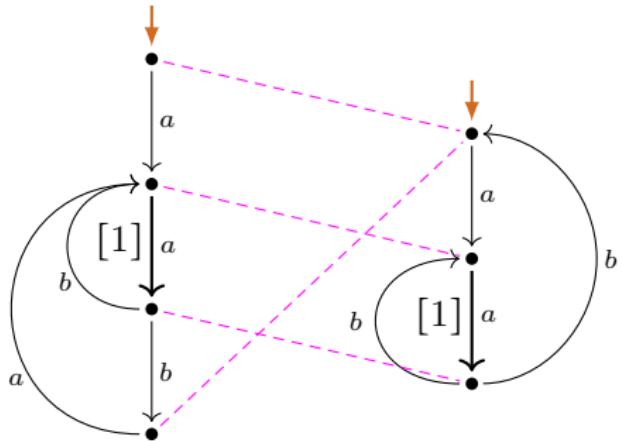
$$P(a(a(b + ba))^*0)$$

# Collapsing LEE-witnesses

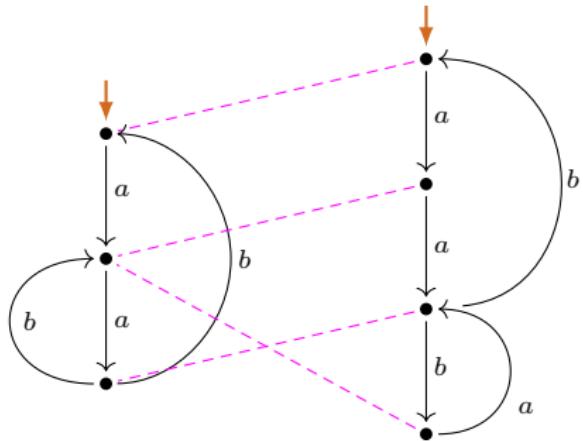


$$P(a(a(b+ba))^*0)$$

# Collapsing LEE-witnesses

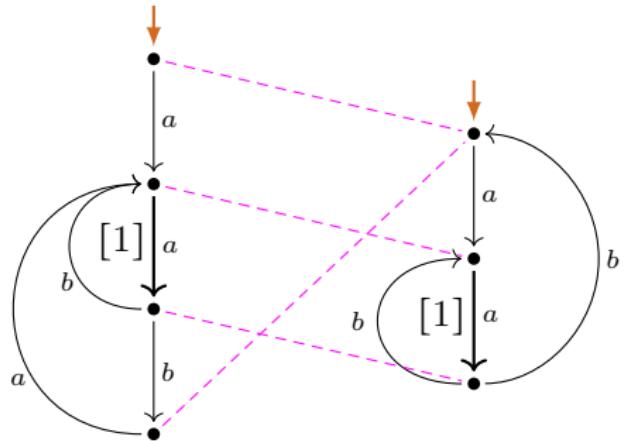


$P(a(a(b+ba))^*0)$

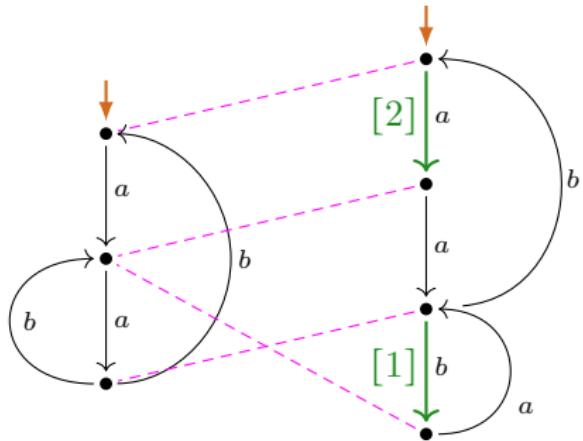


$P((aa(ba)^*b)^*0)$

# Collapsing LEE-witnesses

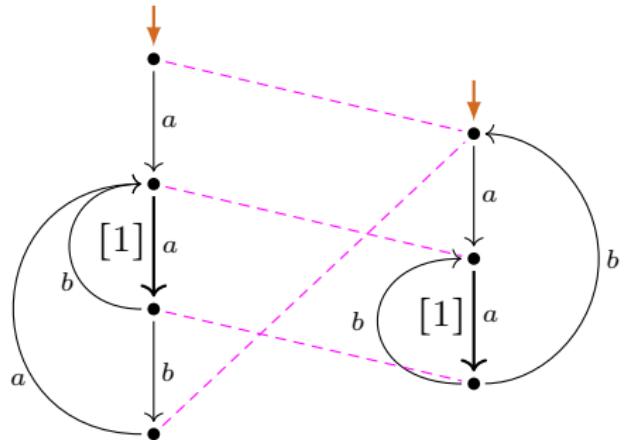


$P(a(a(b+ba))^*0)$

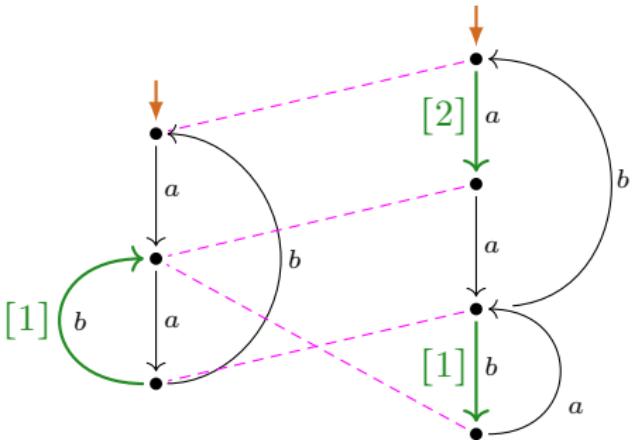


$P((aa(ba)^*b)^*0)$

# Collapsing LEE-witnesses

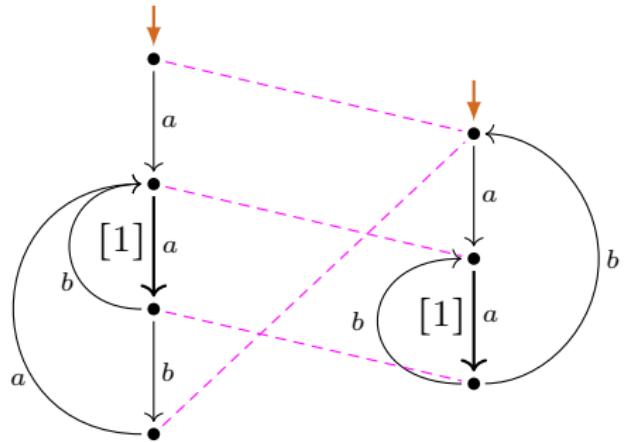


$P(a(a(b+ba))^*0)$

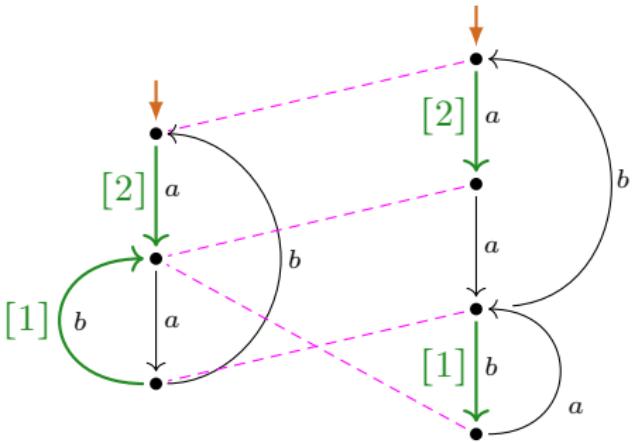


$P((aa(ba)^*b)^*0)$

# Collapsing LEE-witnesses

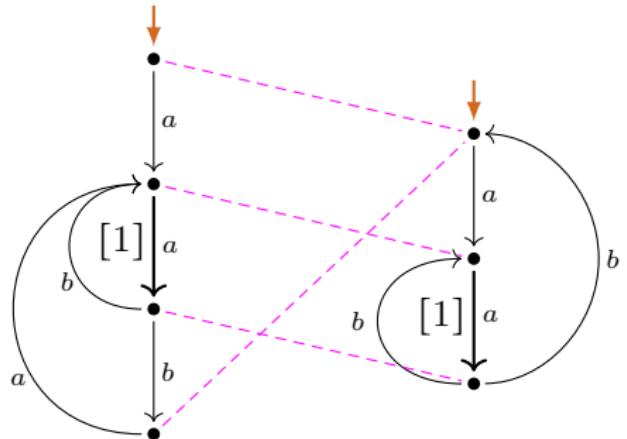


$P(a(a(b+ba))^*0)$

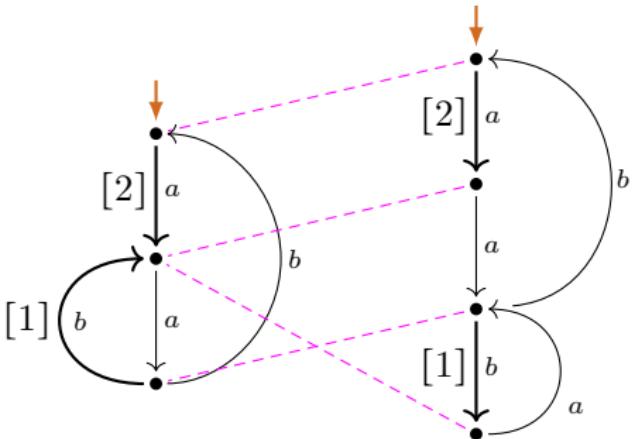


$P((aa(ba)^*b)^*0)$

# Collapsing LEE-witnesses



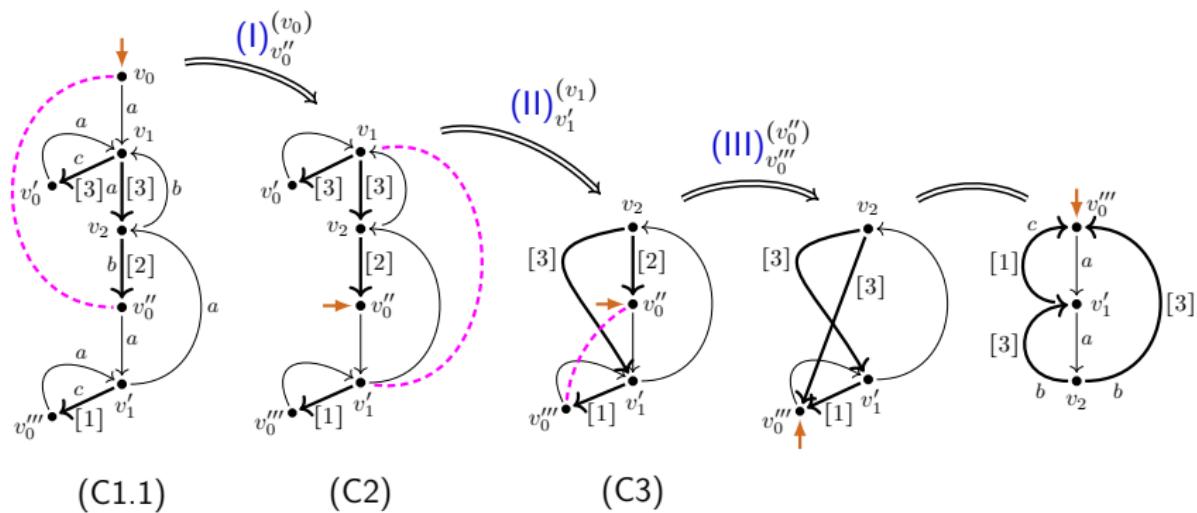
$P(a(a(b+ba))^*0)$



$P((aa(ba)^*b)^*0)$

# LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20)

(no 1-transitions!)



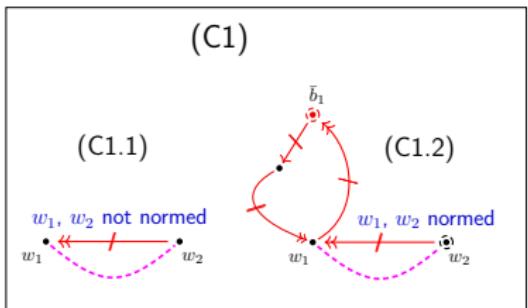
## Lemma

*The bisimulation collapse of a LLEE-chart is again a LLEE-chart.*

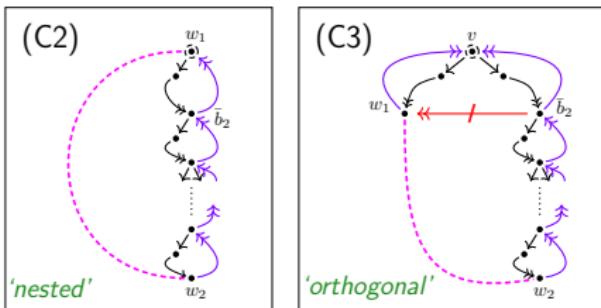
# Reduced bisimilarity redundancies in LLEE-charts (no 1-trans.!)

(G/Fokkink, LICS'20)

$w_1, w_2$  in different scc's



$w_1, w_2$  in the same scc



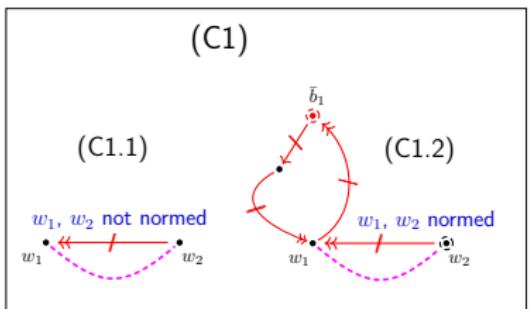
## Lemma

Every *not collapsed* LLEE-chart contains bisimilar vertices  $w_1 \neq w_2$  of kind (C1), (C2), or (C3) (a *reduced bisimilarity redundancy*  $\langle w_1, w_2 \rangle$ ):

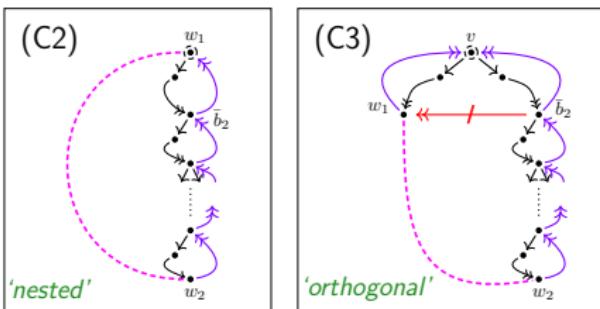
# Reduced bisimilarity redundancies in LLEE-charts (no 1-trans.!)

(G/Fokkink, LICS'20)

$w_1, w_2$  in different scc's



$w_1, w_2$  in the same scc



## Lemma

Every *not collapsed LLEE-chart* contains bisimilar vertices  $w_1 \neq w_2$  of kind (C1), (C2), or (C3) (a *reduced bisimilarity redundancy*  $\langle w_1, w_2 \rangle$ ):

## Lemma

Every *reduced bisimilarity redundancy* in a LLEE-chart can be eliminated LLEE-preservingly.

# LEE under functional bisimulation

## Lemma

(i) LEE is preserved by *functional bisimulations*:

$$\text{LEE}(\textcolor{violet}{G}_1) \wedge \textcolor{violet}{G}_1 \mathrel{\textcolor{brown}{\rightarrow}} \textcolor{violet}{G}_2 \implies \text{LEE}(\textcolor{violet}{G}_2).$$

## Idea of Proof for (i)

Use loop elimination in  $\textcolor{violet}{G}_1$  to carry out loop elimination in  $\textcolor{violet}{G}_2$ .

# LEE under functional bisimulation / bisimulation collapse

## Lemma

(i) LEE is preserved by *functional bisimulations*:

$$\text{LEE}(\textcolor{violet}{G}_1) \wedge G_1 \mathrel{\sqsupseteq} G_2 \implies \text{LEE}(\textcolor{violet}{G}_2).$$

(ii) LEE is preserved from a process graph to its *bisimulation collapse*:

$$\text{LEE}(\textcolor{violet}{G}) \wedge \textcolor{brown}{C} \text{ is bisimulation collapse of } \textcolor{violet}{G} \implies \text{LEE}(\textcolor{brown}{C}).$$

## Idea of Proof for (i)

Use loop elimination in  $\textcolor{violet}{G}_1$  to carry out loop elimination in  $\textcolor{violet}{G}_2$ .

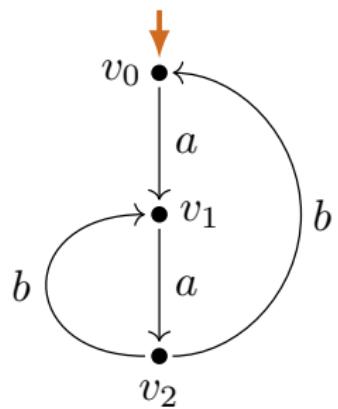
# Readback

## Lemma

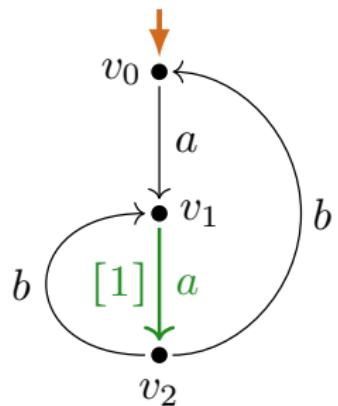
Process graphs with LEE are  $P(\cdot)$ -expressible:

$$\text{LEE}(\textcolor{violet}{G}) \implies \exists e \in \text{Reg}(A) (\textcolor{violet}{G} \sqsubseteq P(e)).$$

# Readback from layered LEE-witness (example)

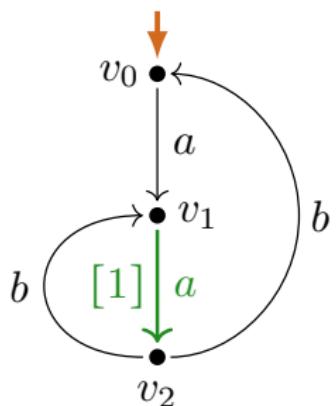


# Readback from layered LEE-witness (example)



layered  
LEE-witness

# Readback from layered LEE-witness (example)



layered  
LEE-witness

$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

$$=_{\text{Mil}} a \cdot s(v_1)$$

$$=_{\text{Mil}} a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$$

$$s(v_1) = (a \cdot s(v_2, v_1))^* \cdot 0$$

$$=_{\text{Mil}} (a \cdot (b + b \cdot a))^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1))$$

$$=_{\text{Mil}} 0^* \cdot (b \cdot 1 + b \cdot a)$$

$$=_{\text{Mil}} b + b \cdot a$$

$$s(v_1, v_1) = 1$$

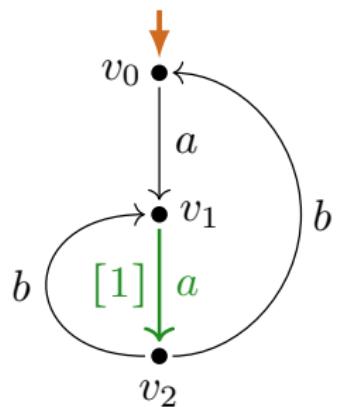
$$s(v_0, v_1) = 0^* \cdot a \cdot s(v_1, v_1)$$

$$= 0^* \cdot a \cdot 1$$

$$=_{\text{Mil}} a$$

# Readback from layered LEE-witness (example)

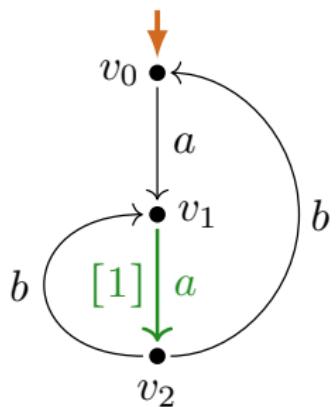
$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$



layered  
LEE-witness

## Readback from layered LEE-witness (example)

$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

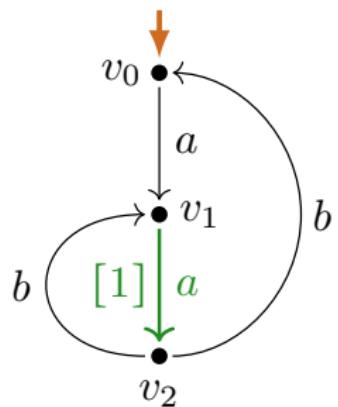


$$s(v_1) = \left( \textcolor{teal}{a} \cdot s(v_2, v_1) \right)^* \cdot 0$$

layered  
LEE-witness

# Readback from layered LEE-witness (example)

$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

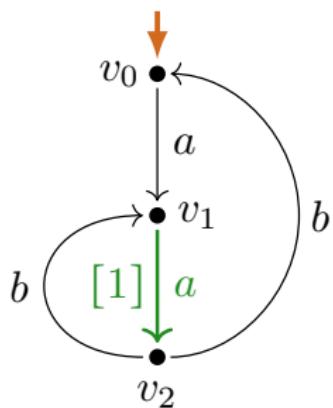


$$s(v_1) = (\textcolor{green}{a} \cdot s(v_2, v_1))^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1))$$

layered  
LEE-witness

# Readback from layered LEE-witness (example)



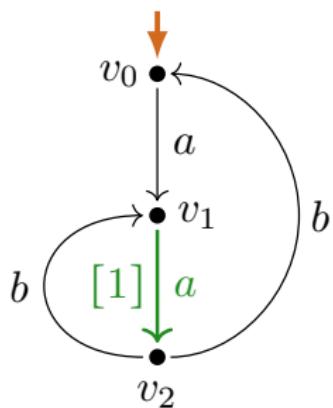
$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

$$s(v_1) = (a \cdot s(v_2, v_1))^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1))$$

$$s(v_1, v_1) = 1$$

# Readback from layered LEE-witness (example)



$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

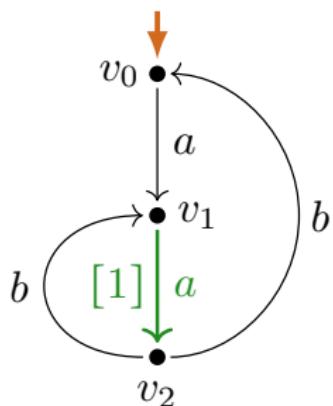
$$s(v_1) = (\textcolor{green}{a} \cdot s(v_2, v_1))^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1))$$

$$s(v_1, v_1) = 1$$

$$s(v_0, v_1) = 0^* \cdot a \cdot s(v_1, v_1)$$

# Readback from layered LEE-witness (example)



$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

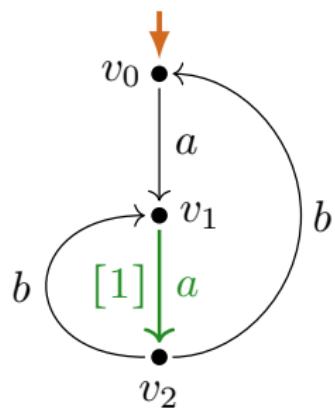
$$s(v_1) = (\textcolor{green}{a} \cdot s(v_2, v_1))^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1))$$

$$s(v_1, v_1) = 1$$

$$\begin{aligned} s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \end{aligned}$$

# Readback from layered LEE-witness (example)



$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

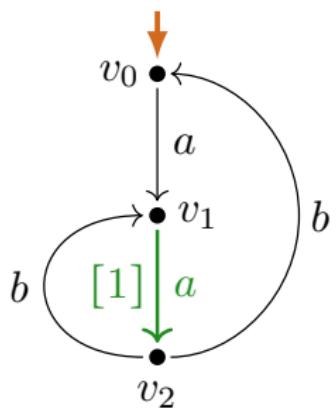
$$s(v_1) = (\textcolor{green}{a} \cdot s(v_2, v_1))^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1))$$

$$s(v_1, v_1) = 1$$

$$\begin{aligned} s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \\ &= \textcolor{blue}{\text{Mil}}^- a \end{aligned}$$

# Readback from layered LEE-witness (example)



$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

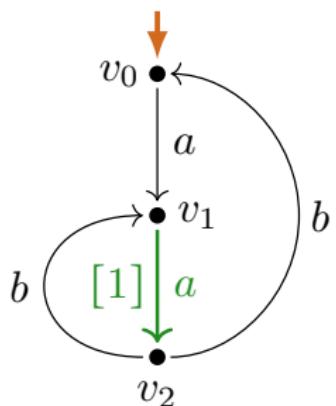
$$s(v_1) = (\textcolor{green}{a} \cdot s(v_2, v_1))^* \cdot 0$$

$$\begin{aligned} s(v_2, v_1) &= 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)) \\ &= \textcolor{purple}{\text{Mil}} \cdot 0^* \cdot (b \cdot 1 + b \cdot a) \end{aligned}$$

$$s(v_1, v_1) = 1$$

$$\begin{aligned} s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \\ &= \textcolor{purple}{\text{Mil}} \cdot a \end{aligned}$$

# Readback from layered LEE-witness (example)



$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

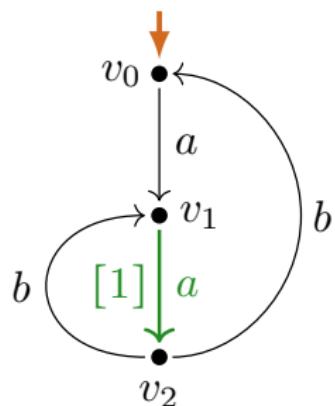
$$s(v_1) = (\textcolor{green}{a} \cdot s(v_2, v_1))^* \cdot 0$$

$$\begin{aligned} s(v_2, v_1) &= 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)) \\ &= \textcolor{purple}{\text{Mil}}^- 0^* \cdot (b \cdot 1 + b \cdot a) \\ &= \textcolor{purple}{\text{Mil}}^- b + b \cdot a \end{aligned}$$

$$s(v_1, v_1) = 1$$

$$\begin{aligned} s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \\ &= \textcolor{purple}{\text{Mil}}^- a \end{aligned}$$

# Readback from layered LEE-witness (example)



$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

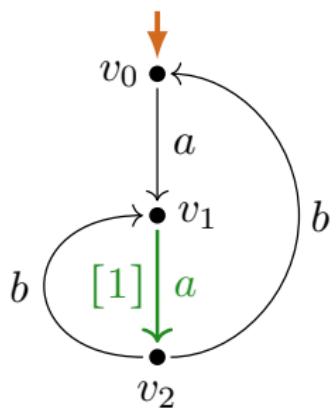
$$\begin{aligned} s(v_1) &= (\textcolor{green}{a} \cdot s(v_2, v_1))^* \cdot 0 \\ &= \textcolor{purple}{\text{Mil}}^- (\textcolor{green}{a} \cdot (b + b \cdot a))^* \cdot 0 \end{aligned}$$

$$\begin{aligned} s(v_2, v_1) &= 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)) \\ &= \textcolor{purple}{\text{Mil}}^- 0^* \cdot (b \cdot 1 + b \cdot a) \\ &= \textcolor{purple}{\text{Mil}}^- b + b \cdot a \end{aligned}$$

$$s(v_1, v_1) = 1$$

$$\begin{aligned} s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \\ &= \textcolor{purple}{\text{Mil}}^- a \end{aligned}$$

# Readback from layered LEE-witness (example)



$$\begin{aligned}s(v_0) &= 0^* \cdot a \cdot s(v_1) \\ &= \text{Mil}^- a \cdot s(v_1)\end{aligned}$$

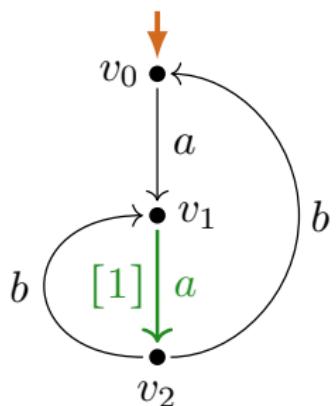
$$\begin{aligned}s(v_1) &= (a \cdot s(v_2, v_1))^* \cdot 0 \\ &= \text{Mil}^- (a \cdot (b + b \cdot a))^* \cdot 0\end{aligned}$$

$$\begin{aligned}s(v_2, v_1) &= 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1)) \\ &= \text{Mil}^- 0^* \cdot (b \cdot 1 + b \cdot a) \\ &= \text{Mil}^- b + b \cdot a\end{aligned}$$

$$s(v_1, v_1) = 1$$

$$\begin{aligned}s(v_0, v_1) &= 0^* \cdot a \cdot s(v_1, v_1) \\ &= 0^* \cdot a \cdot 1 \\ &= \text{Mil}^- a\end{aligned}$$

# Readback from layered LEE-witness (example)



$$s(v_0) = 0^* \cdot a \cdot s(v_1)$$

$$=_{\text{Mil}} a \cdot s(v_1)$$

$$=_{\text{Mil}} a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$$

$$s(v_1) = (a \cdot s(v_2, v_1))^* \cdot 0$$

$$=_{\text{Mil}} (a \cdot (b + b \cdot a))^* \cdot 0$$

$$s(v_2, v_1) = 0^* \cdot (b \cdot s(v_1, v_1) + b \cdot s(v_0, v_1))$$

$$=_{\text{Mil}} 0^* \cdot (b \cdot 1 + b \cdot a)$$

$$=_{\text{Mil}} b + b \cdot a$$

$$s(v_1, v_1) = 1$$

$$s(v_0, v_1) = 0^* \cdot a \cdot s(v_1, v_1)$$

$$= 0^* \cdot a \cdot 1$$

$$=_{\text{Mil}} a$$

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $P(\cdot)$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}(A) ( G \xrightarrow{\cdot} P(e) ).$$

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(\textcolor{violet}{G}) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) \left( \textcolor{violet}{G} \xrightarrow{\quad} \textcolor{green}{P}(e) \right).$$

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- ▶  $(a \cdot (1 + b))^*$

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- ▶  $(a \cdot (1 + b))^*$

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- ▶  $(a \cdot (1 + b))^*$  ✗

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- ▶  $(a \cdot (1 + b))^*$  X
- ▶  $(a \cdot (0^* + b))^*$

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- ▶  $(a \cdot (1 + b))^*$  X
- ▶  $(a \cdot (0^* + b))^*$  X

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- ▶  $(a \cdot (1 + b))^*$  X
- ▶  $(a \cdot (0^* + b))^*$  X
- ▶  $a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- ▶  $(a \cdot (1 + b))^*$  ✗
- ▶  $(a \cdot (0^* + b))^*$  ✗
- ▶  $a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$  ✓

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- |                                                 |   |                                 |
|-------------------------------------------------|---|---------------------------------|
| ▶ $(a \cdot (1 + b))^*$                         | ✗ | ▶ $(a^* (b^* + c \cdot 0)^*)^*$ |
| ▶ $(a \cdot (0^* + b))^*$                       | ✗ |                                 |
| ▶ $a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$ | ✓ |                                 |

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- |                                                 |   |                                 |   |
|-------------------------------------------------|---|---------------------------------|---|
| ▶ $(a \cdot (1 + b))^*$                         | ✗ | ▶ $(a^* (b^* + c \cdot 0)^*)^*$ | ✗ |
| ▶ $(a \cdot (0^* + b))^*$                       | ✗ |                                 |   |
| ▶ $a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$ | ✓ |                                 |   |

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- |                                                 |   |                                 |   |
|-------------------------------------------------|---|---------------------------------|---|
| ▶ $(a \cdot (1 + b))^*$                         | ✗ | ▶ $(a^* (b^* + c \cdot 0)^*)^*$ | ✗ |
| ▶ $(a \cdot (0^* + b))^*$                       | ✗ | ▶ $(a^* (b^* + c \cdot 0))^*$   |   |
| ▶ $a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$ | ✓ |                                 |   |

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- |                                                 |   |                                 |   |
|-------------------------------------------------|---|---------------------------------|---|
| ▶ $(a \cdot (1 + b))^*$                         | ✗ | ▶ $(a^* (b^* + c \cdot 0)^*)^*$ | ✗ |
| ▶ $(a \cdot (0^* + b))^*$                       | ✗ | ▶ $(a^* (b^* + c \cdot 0))^*$   | ✗ |
| ▶ $a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$ | ✓ |                                 |   |

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- |                                                 |   |                                 |   |
|-------------------------------------------------|---|---------------------------------|---|
| ▶ $(a \cdot (1 + b))^*$                         | ✗ | ▶ $(a^* (b^* + c \cdot 0)^*)^*$ | ✗ |
| ▶ $(a \cdot (0^* + b))^*$                       | ✗ | ▶ $(a^* (b^* + c \cdot 0))^*$   | ✗ |
| ▶ $a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$ | ✓ | ▶ $(a^* (b + c \cdot 0))^*$     |   |

# 1-return-less regular expressions

## Lemma

Process graphs with LEE are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible:

$$\text{LEE}(G) \implies \exists e \in \text{Reg}^{1\text{-}\star}(A) (G \sqsubseteq P(e)).$$

## Definition (Corradini, De Nicola, Labella (here intuitive version))

A regular expression  $e$  is 1-return-less(-under- $\star$ ) ( $e \in \text{Reg}^{1\text{-}\star}(A)$ ) if:

- ▶ for no iteration subexpression  $f^*$  of  $e$  does  $P(f)$  proceed to a process  $p$  such that:
  - ▶  $p$  has the option to immediately terminate, and
  - ▶  $p$  has the option to do a proper step, and terminate later.

## Non-/Examples of 1-return-less regular expressions

- |                                                 |   |                                 |   |
|-------------------------------------------------|---|---------------------------------|---|
| ▶ $(a \cdot (1 + b))^*$                         | ✗ | ▶ $(a^* (b^* + c \cdot 0)^*)^*$ | ✗ |
| ▶ $(a \cdot (0^* + b))^*$                       | ✗ | ▶ $(a^* (b^* + c \cdot 0))^*$   | ✗ |
| ▶ $a \cdot (a \cdot (b + b \cdot a))^* \cdot 0$ | ✓ | ▶ $(a^* (b + c \cdot 0))^*$     | ✓ |

# Characterization of expressibility<sup>1r\star</sup>

## Theorem

For every process graph  $G$  with bisimulation collapse  $C$  the following are equivalent:

- (i)  $G$  is  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressible.
- (ii) LEE( $C$ ).
- (iii)  $C$  has a LEE-witness.
- (iv)  $C$  has a layered LEE-witness.

# Characterization of expressibility<sup>1r\star</sup>

## Theorem

For every process graph  $G$  with bisimulation collapse  $C$  the following are equivalent:

- (i)  $G$  is  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressible.
- (ii) LEE( $C$ ).
- (iii)  $C$  has a LEE-witness.
- (iv)  $C$  has a layered LEE-witness.

Milners characterization question:

- Q1. Which structural property of finite process graphs characterizes  $P(\cdot)$ -expressibility?

# Characterization of expressibility<sup>1r\star</sup>

## Theorem

For every process graph  $G$  with bisimulation collapse  $C$  the following are equivalent:

- (i)  $G$  is  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressible.
- (ii) LEE( $C$ ).
- (iii)  $C$  has a LEE-witness.
- (iv)  $C$  has a layered LEE-witness.

Milners characterization question restricted:

Q1'. Which structural property of finite process graphs characterizes  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressibility?

# Characterization of expressibility<sup>1r\star</sup>

## Theorem

For every process graph  $G$  with bisimulation collapse  $C$  the following are equivalent:

- (i)  $G$  is  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressible.
- (ii) LEE( $C$ ).
- (iii)  $C$  has a LEE-witness.
- (iv)  $C$  has a layered LEE-witness.

Milners characterization question restricted, and adapted:

Q1''. Which structural property of collapsed finite process graphs characterizes  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressibility?

# Characterization of expressibility<sup>1r\star</sup>

## Theorem

For every process graph  $G$  with bisimulation collapse  $C$  the following are equivalent:

- (i)  $G$  is  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressible.
- (ii) LEE( $C$ ).
- (iii)  $C$  has a LEE-witness.
- (iv)  $C$  has a layered LEE-witness.

Answering Milner's characterization question restricted, and adapted:

Q1''. Which structural property of collapsed finite process graphs characterizes  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressibility?

- The loop-existence and elimination property LEE.

# Characterization of expressibility<sup>1r\star</sup>

## Theorem

For every process graph  $G$  with bisimulation collapse  $C$  the following are equivalent:

- (i)  $G$  is  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressible.
- (ii) LEE( $C$ ).
- (iii)  $C$  has a LEE-witness.
- (iv)  $C$  has a layered LEE-witness.

Answering Milner's characterization question restricted, and adapted:

Q1''. Which structural property of collapsed finite process graphs characterizes  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressibility?

- ▶ The loop-existence and elimination property LEE.

Also yields: efficient decision method of  $\llbracket \cdot \rrbracket_P^{1r\star}$ -expressibility?

# Structure constrained finite process graphs

graphs with LEE / a (layered) LEE-witness

*Benefits* of the class of process graphs with LEE:

- ▶ is closed under  $\preceq$
- ▶ forth-/back-correspondence with 1-return-less regular expressions

# Structure constrained finite process graphs

graphs with LEE / a (layered) LEE-witness  
‡ graphs whose collapse satisfies LEE  
= graphs that are  $\llbracket \cdot \rrbracket_P^{1\text{-}\star}$ -expressible

*Benefits* of the class of process graphs with LEE:

- ▶ is closed under  $\preceq$
- ▶ forth-/back-correspondence with 1-return-less regular expressions

# Structure constrained finite process graphs

$\llbracket \cdot \rrbracket_P^{1r\backslash *}$ -expressible graphs

- ⊓ graphs with LEE / a (layered) LEE-witness
- ⊓ graphs whose collapse satisfies LEE
- = graphs that are  $\llbracket \cdot \rrbracket_P^{1r\backslash *}$ -expressible

*Benefits* of the class of process graphs with LEE:

- ▶ is closed under  $\preceq$
- ▶ forth-/back-correspondence with 1-return-less regular expressions

# Structure constrained finite process graphs

$\llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressible graphs

- ⊓ graphs with LEE / a (layered) LEE-witness
- ⊓ graphs whose collapse satisfies LEE
- = graphs that are  $\llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressible
- ⊓ graphs that are  $P(\cdot)$ -expressible

*Benefits* of the class of process graphs with LEE:

- ▶ is closed under  $\preceq$
- ▶ forth-/back-correspondence with 1-return-less regular expressions

# Structure constrained finite process graphs

$\llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressible graphs

- ⊓ graphs with LEE / a (layered) LEE-witness
- ⊓ graphs whose collapse satisfies LEE
- = graphs that are  $\llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressible
- ⊓ graphs that are  $P(\cdot)$ -expressible
- ⊓ finite process graphs

*Benefits* of the class of process graphs with LEE:

- ▶ is closed under  $\preceq$
- ▶ forth-/back-correspondence with 1-return-less regular expressions

# Structure constrained finite process graphs

- loop-exit palm trees  $\subseteq \llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressible graphs
- $\subseteq$  graphs with LEE / a (layered) LEE-witness
- $\subseteq$  graphs whose collapse satisfies LEE
- = graphs that are  $\llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressible
- $\subseteq$  graphs that are  $P(\cdot)$ -expressible
- $\subseteq$  finite process graphs

*Benefits* of the class of process graphs with LEE:

- ▶ is closed under  $\preceq$
- ▶ forth-/back-correspondence with 1-return-less regular expressions

# Structure constrained finite process graphs

- loop-exit palm trees  $\subseteq \llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressible graphs
- $\subseteq$  graphs with LEE / a (layered) LEE-witness
- $\subseteq$  graphs whose collapse satisfies LEE
- = graphs that are  $\llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressible
- $\subseteq$  graphs that are  $P(\cdot)$ -expressible
- $\subseteq$  finite process graphs

*Benefits* of the class of process graphs with LEE:

- ▶ is closed under  $\preceq$
- ▶ forth-/back-correspondence with **1-return-less** regular expressions

*Application to Milner's questions* yields partial results:

Q1: characterization/efficient decision of  $\llbracket \cdot \rrbracket_P^{1r\backslash\star}$ -expressibility

Q2: alternative compl. proof of Mil on **1-return-less** expressions (C/DN/L)

# Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete

## Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions

Clemens Grabmayer (Department of Computer Science, Gran Sasso Science Institute, Viale F. Crispi, 7, 67100 L'Aquila AQ, Italy)

### Abstract

We report on a lengthy completeness proof for Robin Milner's proof system **Mil** (1984) for bisimilarity of regular expressions in the process semantics. Central for our proof are the recognitions:

1. Process graphs with 1-transitions (1-charts) and the loop existence/elimination property **LLEE** are **not** closed under bisimulation collapse,
2. Such process graphs can be **'crystallized'** to 'near-collapsed' 1-charts with some strongly connected components of 'twin-crystal' form.

### The Process Semantics of Regular Expressions

Milner (1984) introduced a process semantics for regular expressions: the interpretation of 0 is *deadlock*, of 1 is an *empty step to termination*, letters *a* are *atomic actions*, the operators + and · stand for *choice* and *concatenation* of processes, and unary Kleene star ( $\cdot^*$ ) represents *(unbounded) iteration*. Formally, Milner defined chart (finite process graph) interpretations  $\mathcal{C}(e)$  of regular expressions  $e$ .

### Milner's Proof System

As axiomatization of the relation  $e_1 =_{\text{P}} e_2$  on regular expressions  $e_1$  and  $e_2$  defined by  $\mathcal{C}(e_1) \sqsubseteq \mathcal{C}(e_2)$  (as bisimilarity  $\sqsubseteq$  of chart interpretations), Milner asked whether the following system **Mil** is complete:

- (A1)  $e + (f + g) = (e + f) + g$  (A7)  $e = 1 \cdot e$
  - (A2)  $e + 0 = e$  (A8)  $e = e \cdot 1$
  - (A3)  $e \cdot f = f \cdot e$  (A9)  $0 = 0 \cdot e$
  - (A4)  $e + e = e$  (A10)  $e^* = 1 + e \cdot e^*$
  - (A5)  $e \cdot (f \cdot g) = (e \cdot f) \cdot g$  (A11)  $e^* = (1 + e)^*$
  - (A6)  $(e + f) \cdot g = e \cdot g + f \cdot g$
- $e = f \cdot g$  RSP\* if  $f$  does not terminate immediately  
 $e = f \cdot g$

This system is a variation of Salomaa's complete axiom system (1966) for language equality of regular expressions, missing left-distributivity  $e \cdot (f + g) = e \cdot f + e \cdot g$  and  $e \cdot 0 = 0$ , which are unsound here.

### Loop Existence and Elimination

The process semantics is incomplete: not every finite process graph is *expressible* by (=bisimilar to the interpretation of) a regular expression. A sufficient condition for expressibility is the (*layered*) **loop existence and elimination property LLEE**. It is defined via elimination of 'loops' (loop subcharts):

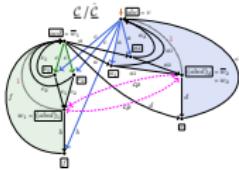


LLEE holds if a graph without infinite behavior can be obtained. Important features of LLEE:

- (US) Every guarded LLEE-1-chart (chart, maybe 1-transitions, with LLEE) is uniquely Mil-provably solvable modulo provability in **Mil** (CALCO 2021).
- (IV) The chart interpretation  $\mathcal{C}(e)$  of a regular expression  $e$  always can be expanded under bisimilarity to a LLEE-1-chart  $\mathcal{C}(e)$  (TERMGRAPH 2020).
- (C<sub>1</sub>) LLEE-charts (without 1-transitions) are preserved by bisimulation collapse (G/Fokkink, LICS'20).

### LLEE-preserving Collapse Fails

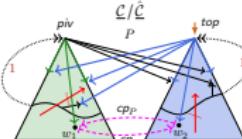
LLEE-1-charts with 1-transitions, however, are **not** preserved under bisimulation collapse. A counterexample is provided by the following LLEE-1-chart  $\mathcal{L}$ :



Identifying the bisimilar vertices  $w_1$  and  $w_2$  yields a chart for which LLEE fails. Also, the subcharts of  $\mathcal{L}$  that are rooted at  $w_1$  and  $w_2$  are **not** LLEE-preservingly jointly minimizable under bisimilarity.

### Twin-Crystals

The counterexample to LLEE-preserving collapse is symmetric, and its structure can be abstracted as:



It is a LLEE-1-chart with a single scc (strongly connected component)  $P$  that consists of a *pivot part*  $P_1$  below *pivot vertex*  $piv$ , and a *top part*  $P_2$  below *top vertex*  $top$ .  $P_1$  and  $P_2$  are connected only via transitions from  $piv$  and from  $top$ . While both  $P_1$  and  $P_2$  are collapsed,  $P$  contains *bisimilarity redundancies* ( $=$  distinct bisimilar vertices such as  $w_1, w_2$ ) that are linked by a self-inverse counterpart function  $cp_P$ . We call such an scc a *twin-crystal*. We have:

- (CC) Every Mil-provable solution of a twin-crystal gives rise to a Mil-provable solution of its bisimulation collapse (which often is not a LLEE-1-chart).

### Crystallization of LLEE-1-charts

By *crystallization* of a LLEE-1-chart  $\mathcal{L}$  we mean:

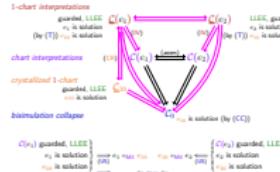
- ▷ a process of minimization of  $\mathcal{L}$  under bisimilarity by steps that **eliminate most** (all but crystalline) bisimilarity redundancies  $\{w_1, w_2\}$ , roughly by redirecting transitions that target  $w_1$  over to  $w_2$ ;
- ▷ hereby only **reduced** bisimilarity redundancies can be eliminated LLEE-preservingly, which exist whenever a LLEE-1-chart is **not collapsed**;
- ▷ the result is a **crystallized** LLEE-1-chart that is bisimilar to  $\mathcal{L}$ , and collapsed **apart from** within some its scc's that are twin-crystals.

The *crystallization process* facilitates to show:

- (CR) From every LLEE-1-chart a bisimilar crystallized LLEE-1-chart can be obtained.

### Completeness Proof

Let  $\mathcal{C}(e_1) \sqsubseteq \mathcal{C}(e_2)$  be bisimilar chart interpretations of regular expressions  $e_1$  and  $e_2$ . To secure LLEE,  $\mathcal{C}(e_1)$  and  $\mathcal{C}(e_2)$  are expanded to their 1-chart interpretations  $\mathcal{Q}(e_1)$  and  $\mathcal{Q}(e_2)$ . One of them, say  $\mathcal{Q}(e_1)$ , is crystallized to  $\mathcal{C}_0$ . All (1-)charts are linked by (1-)bisimulations to their bisimulation collapse  $\mathcal{C}_0$ .



From  $\mathcal{C}_0$  a provable solution  $e_{00}$  can be extracted due to LLEE, transferred (T) to the collapse  $\mathcal{C}_0$ , and then to  $\mathcal{Q}(e_1)$  and  $\mathcal{Q}(e_2)$ . On the LLEE-1-charts  $\mathcal{Q}(e_1)$  and  $\mathcal{Q}(e_2)$ ,  $e_{00}$  can be proved equal to the solutions  $e_1$  and  $e_2$  there, respectively. By transitivity,  $e_1 =_{\text{Mil}} e_2$  (provability of  $e_1 = e_2$  in Mil) follows.

**Theorem.** *Milner's system Mil is complete:  $e_1 =_{\text{P}} e_2$  implies  $e_1 =_{\text{Mil}} e_2$ , for reg. expr.'s  $e_1, e_2$ .*

### Next Steps and Projects

- ▷ Monograph project: proof in fine-grained details.
- ▷ Build an animation tool for crystallization.
- ▷ Apply crystallization to find an efficient algorithm for expressivity of finite process graphs by a regular expression modulo bisimilarity.

### Contact

clemens.grabmayer@gssi.it



# Comparison results: structure-constrained graphs

$\lambda$ -calculus with letrec under  $=_{\lambda^\infty}$

*Not available:* graph interpretation that is studied under  $\leftrightarrow$

Regular expressions under  $\leftrightarrow_P$

*Given:* graph interpretation  $P(\cdot)$ , studied under bisimulation  $\leftrightarrow$

- ▶ not closed under  $\Rightarrow$ , and  $\leftrightarrow$ , incomplete under  $\leftrightarrow$

# Comparison results: structure-constrained graphs

$\lambda$ -calculus with letrec under  $=_{\lambda^\infty}$

*Not available:* graph interpretation that is studied under  $\leftrightarrow$

*Defined:* int's  $\llbracket \cdot \rrbracket_{\mathcal{H}} / \llbracket \cdot \rrbracket_{\mathcal{T}}$  as higher-order/first-order  $\lambda$ -term graphs

- ▶ closed under  $\succeq$  (hence under collapse)
- ▶ back-/forth correspondence with  $\lambda$ -calculus with letrec
  - ▶ efficient translation and readback
  - ▶ translation is inverse of readback

Regular expressions under  $\leftrightarrow_P$

*Given:* graph interpretation  $P(\cdot)$ , studied under bisimulation  $\leftrightarrow$

- ▶ not closed under  $\succeq$ , and  $\leftrightarrow$ , incomplete under  $\leftrightarrow$

# Comparison results: structure-constrained graphs

$\lambda$ -calculus with letrec under  $=_{\lambda^\infty}$

*Not available:* graph interpretation that is studied under  $\leftrightarrow$

*Defined:* int's  $\llbracket \cdot \rrbracket_{\mathcal{H}} / \llbracket \cdot \rrbracket_{\mathcal{T}}$  as higher-order/first-order  $\lambda$ -term graphs

- ▶ closed under  $\succeq$  (hence under collapse)
- ▶ back-/forth correspondence with  $\lambda$ -calculus with letrec
  - ▶ efficient translation and readback
  - ▶ translation is inverse of readback

Regular expressions under  $\leftrightarrow_P$

*Given:* graph interpretation  $P(\cdot)$ , studied under bisimulation  $\leftrightarrow$

- ▶ not closed under  $\succeq$ , and  $\leftrightarrow$ , incomplete under  $\leftrightarrow$

*Defined:* class of process graphs with LEE / (layered) LEE-witness

- ▶ closed under  $\succeq$  (hence under collapse)
- ▶ back-/forth correspondence with 1-return-less expr's
- ▶ contains the collapse of a process graph  $G$ 
  - $\iff G$  is  $\llbracket \cdot \rrbracket_P^{1\text{-R}^*}$ -expressible