

# The Graph Structure of Process Interpretations of Regular Expressions

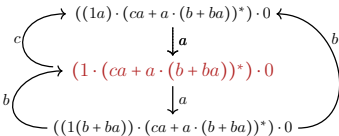
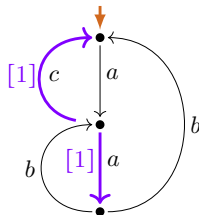
Clemens Grabmayer

<https://clegra.github.io>

Department of Computer Science



L'Aquila, Italy



IFIP 1.6 Working Group Meeting

Nancy

July 1, 2024

# Overview

- ▶ regular expressions (with unary/binary star, 1-free-under-star  $(*/\perp)$ )
- ▶ Milner's process interpretation  $P$ /semantics  $\llbracket \cdot \rrbracket_P$ 
  - ▶  $P$ -/ $\llbracket \cdot \rrbracket_P$ -expressible graphs ( $\leadsto$  expressibility question)
  - ▶ axioms for  $\llbracket \cdot \rrbracket_P$ -identity ( $\leadsto$  completeness question)
- ▶ loop existence and elimination (LEE)
  - ▶ defined by loop elimination rewrite system, its completion
  - ▶ describes interpretations of  $(*/\perp)$  reg. expr.s (extraction possible)
  - ▶ LEE-witnesses: labelings of process graphs with LEE
  - ▶ LEE is preserved under bisimulation collapse (stepwise collapse)
- ▶ 1-LEE = sharing via 1-transitions facilitates LEE
- ▶ LEE/1-LEE characterize image of  $P^\bullet$  (restricted/unrestricted)
  - ▶ where  $P^\bullet$  a compact (sharing-increased) refinement of  $P$
- ▶ outlook on work-to-do

# Overview

- ▶ regular expressions (with unary/binary star, 1-free-under-star  $(*/\perp)$ )
- ▶ Milner's process interpretation  $P$ /semantics  $\llbracket \cdot \rrbracket_P$ 
  - ▶  $P$ -/ $\llbracket \cdot \rrbracket_P$ -expressible graphs ( $\leadsto$  expressibility question)
  - ▶ axioms for  $\llbracket \cdot \rrbracket_P$ -identity ( $\leadsto$  completeness question)
- ▶ loop existence and elimination (LEE)
  - ▶ defined by loop elimination rewrite system, its completion
  - ▶ describes interpretations of  $(*/\perp)$  reg. expr.s (extraction possible)
  - ▶ LEE-witnesses: labelings of process graphs with LEE
  - ▶ LEE is preserved under bisimulation collapse (stepwise collapse)
- ▶ 1-LEE = sharing via 1-transitions facilitates LEE
  - ▶ describes interpretations of all reg. expr.s (extraction possible)
  - ▶ not preserved under bisimulation collapse (approximation possible)
- ▶ LEE/1-LEE characterize image of  $P^\bullet$  (restricted/unrestricted)
  - ▶ where  $P^\bullet$  a compact (sharing-increased) refinement of  $P$
  - ▶ via refined extraction using LEE/1-LEE
- ▶ outlook on work-to-do

# Regular Expressions

Definition ( *~ Copi-Elgot-Wright, 1958* )

Regular expressions over alphabet  $A$  with unary Kleene star:

$e, e_1, e_2 ::= 0 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^*$  (for  $a \in A$ ).

- ▶ symbol **0** instead of  $\emptyset$ , symbol **1** instead of  $\{\epsilon\}$

# Regular Expressions

Definition ( *~ Copi-Elgot-Wright, 1958* )

Regular expressions over alphabet  $A$  with unary Kleene star:

$e, e_1, e_2 ::= 0 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^*$  (for  $a \in A$ ).

- ▶ symbol **0** instead of  $\emptyset$ , symbol **1** instead of  $\{\epsilon\}$
- ▶ with unary star **\***: **1** is definable as **0\***

# Regular Expressions

Definition (*~ Kleene, 1951, ~ Copi-Elgot-Wright, 1958*)

Regular expressions over alphabet  $A$  with unary / binary Kleene star:

$$e, e_1, e_2 ::= 0 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^* \quad (\text{for } a \in A).$$

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e_1 \oplus e_2 \quad (\text{for } a \in A).$$

- ▶ symbol **0** instead of  $\emptyset$ , symbol **1** instead of  $\{\epsilon\}$
- ▶ with unary star  $^*$ : **1** is definable as **0<sup>\*</sup>**
- ▶ with binary star  $\oplus$ : **1** is **not** definable (in its absence)

# Regular Expressions

Definition (*~ Kleene, 1951, ~ Copi-Elgot-Wright, 1958*)

Regular expressions over alphabet  $A$  with unary / binary Kleene star:

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^* \quad (\text{for } a \in A).$$

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e_1 \oplus e_2 \quad (\text{for } a \in A).$$

- ▶ symbol **0** instead of  $\emptyset$ , symbol **1** instead of  $\{\epsilon\}$
- ▶ with unary star  $^*$ : **1** is definable as **0<sup>\*</sup>**
- ▶ with binary star  $\oplus$ : **1** is **not** definable (in its absence)

# Regular Expressions (1-free)

Definition (*~ Kleene, 1951, ~ Copi-Elgot-Wright, 1958*)

Regular expressions over alphabet  $A$  with unary / binary Kleene star:

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^* \quad (\text{for } a \in A).$$

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e_1^{\oplus} e_2 \quad (\text{for } a \in A).$$

- ▶ symbol **0** instead of  $\emptyset$ , symbol **1** instead of  $\{\epsilon\}$
- ▶ with unary star  $^*$ : **1** is definable as **0<sup>\*</sup>**
- ▶ with binary star  $^{\oplus}$ : **1** is **not** definable (in its absence)

Definition (for process interpretation)

1-free regular expressions over alphabet  $A$  with binary Kleene star:

$$f, f_1, f_2 ::= 0 \mid a \mid f_1 + f_2 \mid f_1 \cdot f_2 \mid f_1^{\oplus} f_2 \quad (\text{for } a \in A).$$



# Regular Expressions (1-free)

Definition (~ Kleene, 1951, ~ Copi-Elgot-Wright, 1958)

Regular expressions over alphabet  $A$  with unary / binary Kleene star:

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^* \quad (\text{for } a \in A).$$

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e_1^{\otimes} e_2 \quad (\text{for } a \in A).$$

- ▶ symbol **0** instead of  $\emptyset$ , symbol **1** instead of  $\{\epsilon\}$
- ▶ with unary star  $^*$ : **1** is definable as **0<sup>\*</sup>**
- ▶ with binary star  $^{\otimes}$ : **1** is **not** definable (in its absence)

Definition (for process interpretation)

1-free regular expressions over alphabet  $A$  with unary / binary Kleene star:

$$f, f_1, f_2 ::= 0 \mid a \mid f_1 + f_2 \mid f_1 \cdot f_2 \mid (f_1^*) \cdot f_2 \quad (\text{for } a \in A),$$

$$f, f_1, f_2 ::= 0 \mid a \mid f_1 + f_2 \mid f_1 \cdot f_2 \mid f_1^{\otimes} f_2 \quad (\text{for } a \in A).$$

# Regular Expressions (under-star-/1-free)

Definition ( $\sim$  Kleene, 1951,  $\sim$  Copi-Elgot-Wright, 1958)

Regular expressions over alphabet  $A$  with unary / binary Kleene star:

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e^* \quad (\text{for } a \in A).$$

$$e, e_1, e_2 ::= 0 \mid 1 \mid a \mid e_1 + e_2 \mid e_1 \cdot e_2 \mid e_1^{\oplus} e_2 \quad (\text{for } a \in A).$$

- ▶ symbol **0** instead of  $\emptyset$ , symbol **1** instead of  $\{\epsilon\}$
- ▶ with unary star  $^*$ : **1** is definable as **0** $^*$
- ▶ with binary star  $^{\oplus}$ : **1** is **not** definable (in its absence)

Definition (for process interpretation)

The set  $RExp^{(+)}(A)$  of **1-free regular expressions** over  $A$  is defined by:

$$f, f_1, f_2 ::= 0 \mid a \mid f_1 + f_2 \mid f_1 \cdot f_2 \mid f_1^* \cdot f_2 \quad (\text{for } a \in A),$$

the set  $RExp^{(*/+)}(A)$  of **under-star-1-free regular expressions** over  $A$  by:

$$uf, uf_1, uf_2 ::= 0 \mid 1 \mid a \mid uf_1 + uf_2 \mid uf_1 \cdot uf_2 \mid f^* \quad (\text{for } a \in A).$$

# Process interpretation $P$ of regular expressions *(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 interpretation $P$ of regular expressions (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

$e_1 + e_2 \xrightarrow{P}$  (*choice*) execute  $P(e_1)$  or  $P(e_2)$

$e_1 \cdot e_2 \xrightarrow{P}$  (*sequentialization*) execute  $P(e_1)$ , then  $P(e_2)$

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

# Process interpretation $P$ of regular expressions (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

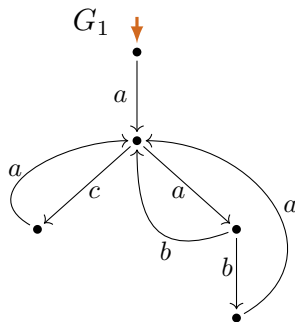
$e_1 + e_2 \xrightarrow{P}$  (*choice*) execute  $P(e_1)$  or  $P(e_2)$

$e_1 \cdot e_2 \xrightarrow{P}$  (*sequentialization*) execute  $P(e_1)$ , then  $P(e_2)$

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

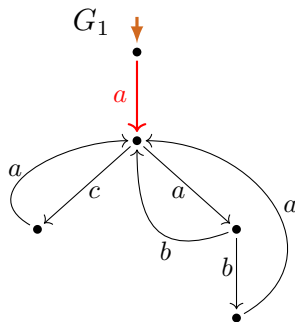
$\llbracket e \rrbracket_P := \llbracket P(e) \rrbracket_{\leftrightarrow}$  (*bisimilarity* equivalence class of process  $P(e)$ )

# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



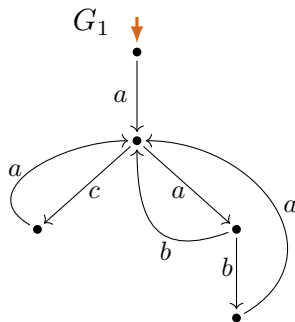
$$P\left(\overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^*}^f \cdot 0\right)$$

# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



$$P\left(\overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^*}^f \cdot 0\right)$$

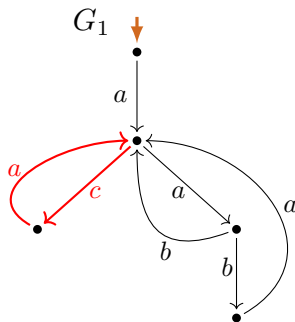
# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



$$P\left(\overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^*}^f \cdot 0\right)$$

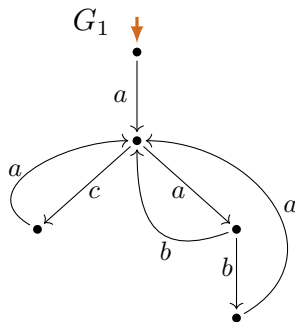


# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



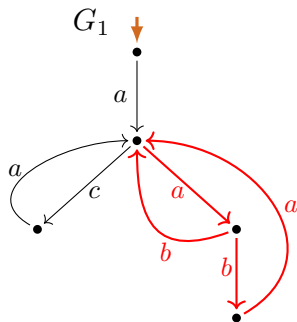
$$P\left(\overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^*}^f \cdot 0\right)$$

# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



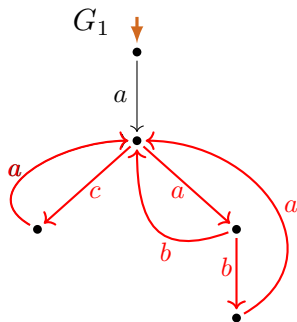
$$P\left(\overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^*}^f \cdot 0\right)$$

# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



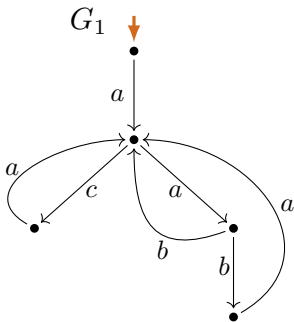
$$P\left(\overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^*}^f \cdot 0\right)$$

# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



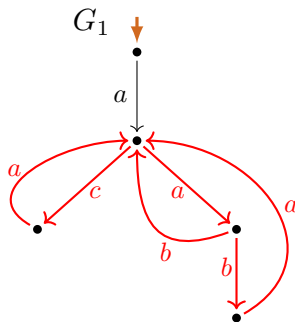
$$P\left(\overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^*}^f \cdot 0\right)$$

$P$ -expressibility and  $\llbracket \cdot \rrbracket_P$ -expressibility (example, informally)



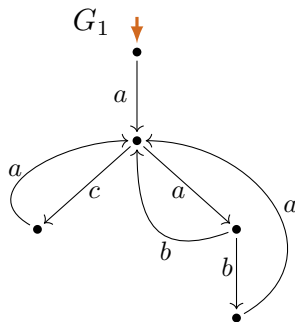
$$\begin{array}{c} \overbrace{P((a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^* \cdot 0)}^f \\ P(a \cdot (c \cdot a + a \cdot (b + b \cdot a))^{\otimes} 0) \end{array}$$

# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



$$\begin{array}{c}
 \overbrace{P\left( (a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^* \cdot 0 \right)}^f \\
 P\left( a \cdot (c \cdot a + a \cdot (b + b \cdot a))^{\otimes 0} \right)
 \end{array}$$

# $P$ -expressibility and $\llbracket \cdot \rrbracket_P$ -expressibility (example, informally)

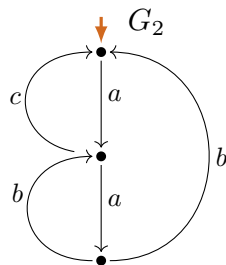
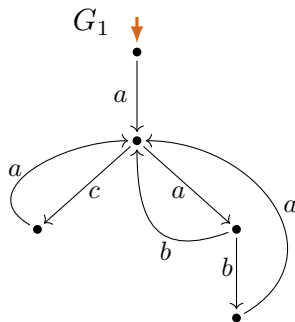


$$\overbrace{P\left( (a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^* \cdot 0 \right)}^f$$

$$P\left( a \cdot (c \cdot a + a \cdot (b + b \cdot a))^{\otimes 0} \right)$$

$$G_1 \in \llbracket f \rrbracket_P$$

# $P$ -expressibility and $\llbracket \cdot \rrbracket_P$ -expressibility (example, informally)



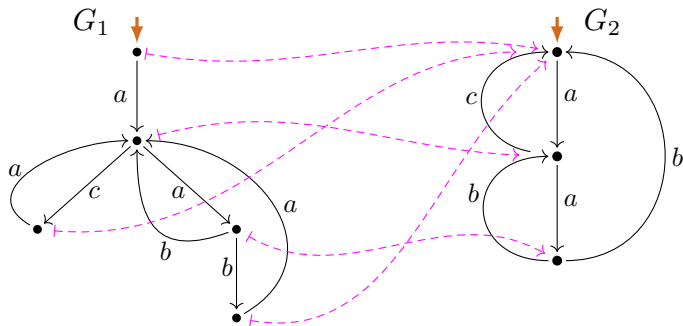
$$\overbrace{P\left( (a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^* \cdot 0 \right)}^f$$

$$P\left( a \cdot (c \cdot a + a \cdot (b + b \cdot a))^{\otimes 0} \right)$$

$$G_1 \in \llbracket f \rrbracket_P$$



# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)

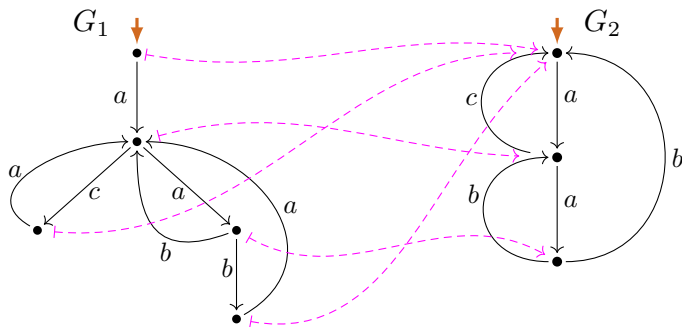


$$P\left(\overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a)))^*}^f \cdot 0\right)$$

$$P\left(a \cdot (c \cdot a + a \cdot (b + b \cdot a))^{\otimes 0}\right)$$

$$G_1 \in [[f]]_P$$

# $P$ -expressibility and $\llbracket \cdot \rrbracket_P$ -expressibility (example, informally)



$$P\left( \overbrace{(a \cdot (c \cdot a + a \cdot (b + b \cdot a))^*) \cdot 0}^f \right)$$

$$P\left( a \cdot (c \cdot a + a \cdot (b + b \cdot a))^{\otimes} 0 \right)$$

$$G_2 \in \llbracket f \rrbracket_P$$

$$G_1 \in \llbracket f \rrbracket_P$$

# Process interpretation $\mathcal{P}$ (formally)

Definition (Transition system specification  $\mathcal{T}$ )

$$\frac{}{a \xrightarrow{\mathcal{A}} 1} \quad \frac{e_i \xrightarrow{\mathcal{A}} e'_i}{e_1 + e_2 \xrightarrow{\mathcal{A}} e'_i} \quad (i \in \{1, 2\})$$

# Process interpretation $P$ (formally)

Definition (Transition system specification  $\mathcal{T}$ )

$$\begin{array}{c} \frac{}{a \xrightarrow{a} 1} \quad \frac{e_i \xrightarrow{a} e'_i}{e_1 + e_2 \xrightarrow{a} e'_i} \quad (i \in \{1, 2\}) \\[2ex] \frac{e \xrightarrow{a} e'}{e^* \xrightarrow{a} e' \cdot e^*} \end{array}$$

# Process interpretation $P$ (formally)

## Definition (Transition system specification $\mathcal{T}$ )

$$\begin{array}{c}
 \frac{}{1 \Downarrow} \qquad \frac{e_i \Downarrow}{(e_1 + e_2) \Downarrow} \ (i \in \{1, 2\}) \qquad \frac{e_1 \Downarrow \quad e_2 \Downarrow}{(e_1 \cdot e_2) \Downarrow} \qquad \frac{}{(e^*) \Downarrow} \\
 \\
 \frac{}{a \xrightarrow{a} 1} \qquad \frac{e_i \xrightarrow{a} e'_i}{e_1 + e_2 \xrightarrow{a} e'_i} \ (i \in \{1, 2\}) \\
 \\
 \frac{e \xrightarrow{a} e'}{e^* \xrightarrow{a} e' \cdot e^*}
 \end{array}$$

# Process interpretation $P$ (formally)

## Definition (Transition system specification $\mathcal{T}$ )

$$\begin{array}{c}
 \overline{1 \Downarrow} \qquad \overline{e_i \Downarrow} \quad (i \in \{1, 2\}) \quad \overline{e_1 \Downarrow \quad e_2 \Downarrow} \quad \overline{(e^*) \Downarrow} \\
 \overline{(e_1 + e_2) \Downarrow} \qquad \overline{(e_1 \cdot e_2) \Downarrow} \\
 \\
 \overline{a \xrightarrow{a} 1} \qquad \overline{e_i \xrightarrow{a} e'_i} \quad (i \in \{1, 2\}) \\
 \overline{e_1 + e_2 \xrightarrow{a} e'_i} \\
 \\
 \overline{e_1 \xrightarrow{a} e'_1} \quad \overline{e_1 \Downarrow \quad e_2 \xrightarrow{a} e'_2} \quad \overline{e \xrightarrow{a} e'} \\
 \overline{e_1 \cdot e_2 \xrightarrow{a} e'_1 \cdot e_2} \quad \overline{e_1 \cdot e_2 \xrightarrow{a} e'_2} \quad \overline{e^* \xrightarrow{a} e' \cdot e^*}
 \end{array}$$

# Process interpretation $P$ (formally)

## Definition (Transition system specification $\mathcal{T}$ )

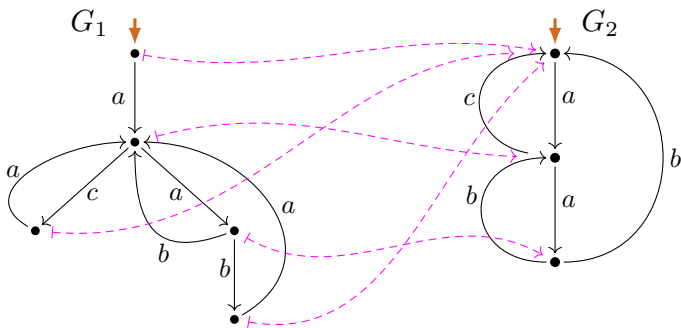
$$\begin{array}{c}
 \frac{}{1 \Downarrow} \qquad \frac{e_i \Downarrow}{(e_1 + e_2) \Downarrow} \ (i \in \{1, 2\}) \qquad \frac{e_1 \Downarrow \quad e_2 \Downarrow}{(e_1 \cdot e_2) \Downarrow} \qquad \frac{}{(e^*) \Downarrow} \\
 \\
 \frac{}{a \xrightarrow{a} 1} \qquad \frac{e_i \xrightarrow{a} e'_i}{e_1 + e_2 \xrightarrow{a} e'_i} \ (i \in \{1, 2\}) \\
 \\
 \frac{e_1 \xrightarrow{a} e'_1}{e_1 \cdot e_2 \xrightarrow{a} e'_1 \cdot e_2} \qquad \frac{e_1 \Downarrow \quad e_2 \xrightarrow{a} e'_2}{e_1 \cdot e_2 \xrightarrow{a} e'_2} \qquad \frac{e \xrightarrow{a} e'}{e^* \xrightarrow{a} e' \cdot e^*}
 \end{array}$$

## Definition

The **process (graph) interpretation**  $P(e)$  of a regular expression  $e$ :

$P(e) :=$  **labeled transition graph** generated by  $e$  by derivations in  $\mathcal{T}$ .

# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, informally)



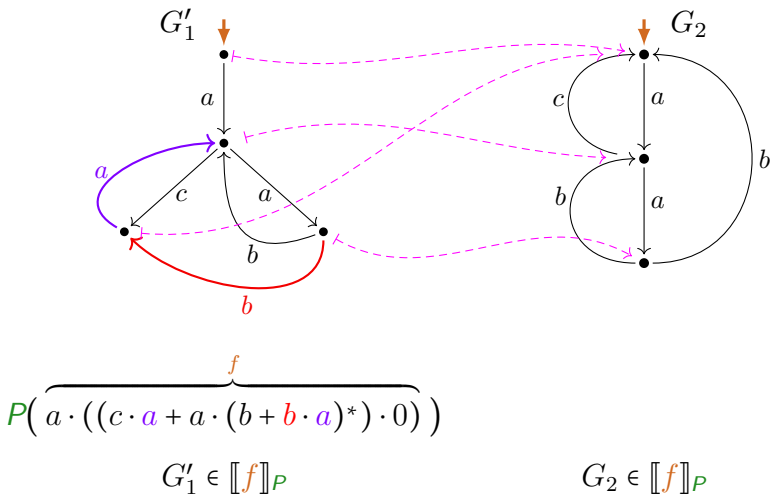
$$P\left(\overbrace{a \cdot ((c \cdot a + a \cdot (b + b \cdot a)^*) \cdot 0))}^f\right)$$

$$G_1 \in [[f]]_P$$

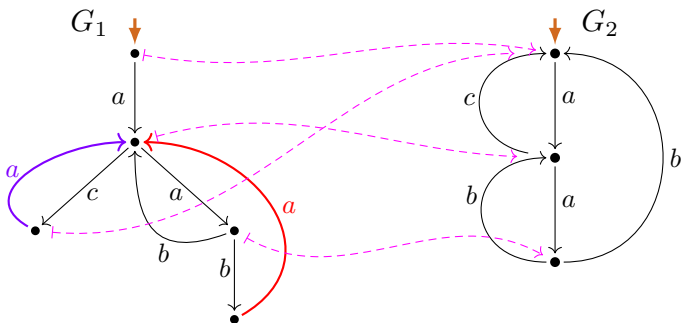
$$G_2 \in [[f]]_P$$



# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (example, formally)



# $P$ -expressibility and $\llbracket \cdot \rrbracket_P$ -expressibility (example, formally)

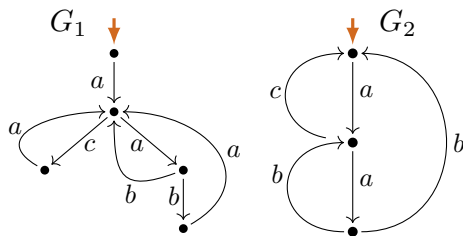


$$P\left(\overbrace{a \cdot ((c \cdot a + a \cdot (b + b \cdot (a + a)))^*)}^f \cdot 0\right)$$

$$G_1 \in \llbracket f \rrbracket_P$$

$$G_2 \in \llbracket f \rrbracket_P$$

# $P$ -expressibility and $[[\cdot]]_P$ -expressibility (examples)

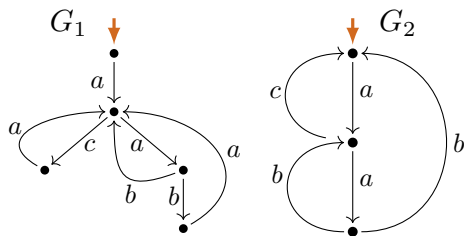


$P$ -expressible

$[[\cdot]]_P$ -expressible

$[[\cdot]]_P$ -expressible

# $P$ -expressibility and $\llbracket \cdot \rrbracket_P$ -expressibility (examples)



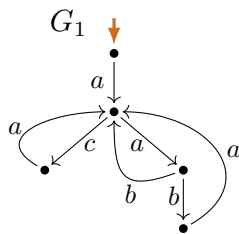
$P$ -expressible

?

$\llbracket \cdot \rrbracket_P$ -expressible

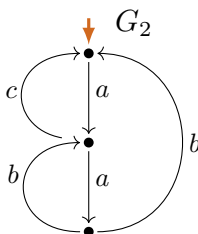
$\llbracket \cdot \rrbracket_P$ -expressible

# $P$ -expressibility and $\llbracket \cdot \rrbracket_P$ -expressibility (examples)



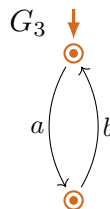
$P$ -expressible

$\llbracket \cdot \rrbracket_P$ -expressible



?

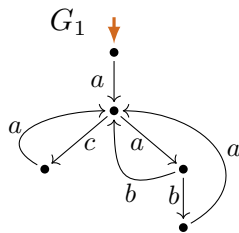
$\llbracket \cdot \rrbracket_P$ -expressible



**not**  $P$ -expressible

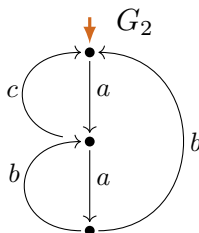
**not**  $\llbracket \cdot \rrbracket_P$ -expressible

# $P$ -expressibility and $\llbracket \cdot \rrbracket_P$ -expressibility (examples)



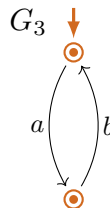
$P$ -expressible

$\llbracket \cdot \rrbracket_P$ -expressible



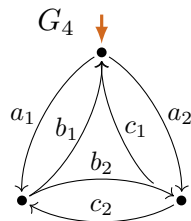
?

$\llbracket \cdot \rrbracket_P$ -expressible

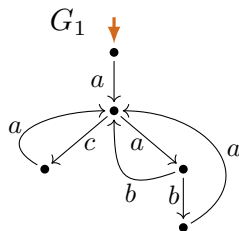


**not**  $P$ -expressible

**not**  $\llbracket \cdot \rrbracket_P$ -expressible

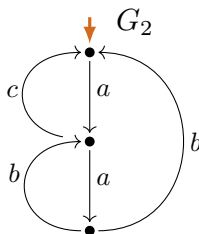


# $P$ -expressibility and $\llbracket \cdot \rrbracket_P$ -expressibility (examples)



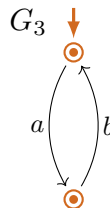
$P$ -expressible

$\llbracket \cdot \rrbracket_P$ -expressible



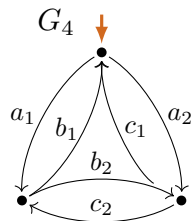
?

$\llbracket \cdot \rrbracket_P$ -expressible



**not**  $P$ -expressible

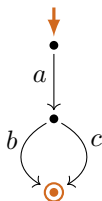
**not**  $\llbracket \cdot \rrbracket_P$ -expressible



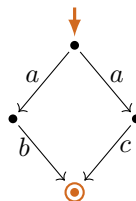
Q2: How can  $P$ -expressibility and  $\llbracket \cdot \rrbracket_P$ -expressibility be characterized?

# Process semantics equality $=_{\llbracket \cdot \rrbracket_P}$

- **Fewer** identities hold for  $=_{\llbracket \cdot \rrbracket_P}$  than for  $=_{\llbracket \cdot \rrbracket_L}$  :



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

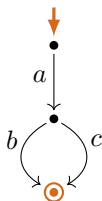


$$P(a \cdot b + a \cdot c)$$

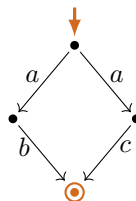


# Process semantics equality $=_{\llbracket \cdot \rrbracket_P}$

- Fewer identities hold for  $=_{\llbracket \cdot \rrbracket_P}$  than for  $=_{\llbracket \cdot \rrbracket_L}$  :



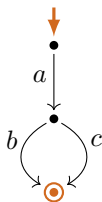
$$P(a \cdot (b + c))$$



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

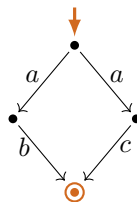
# Process semantics equality $=_{\llbracket \cdot \rrbracket_P}$

- **Fewer** identities hold for  $=_{\llbracket \cdot \rrbracket_P}$  than for  $=_{\llbracket \cdot \rrbracket_L}$  :



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

$\neq_{\llbracket \cdot \rrbracket_P}$

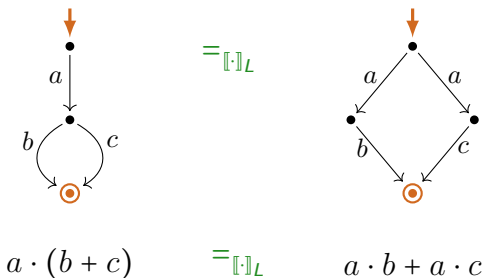


$$a \cdot b + a \cdot c$$

$\neq_{\llbracket \cdot \rrbracket_P}$

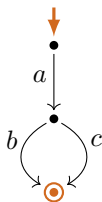
# Process semantics equality $=_{\llbracket \cdot \rrbracket_P}$

- Fewer identities hold for  $=_{\llbracket \cdot \rrbracket_P}$  than for  $=_{\llbracket \cdot \rrbracket_L}$  :



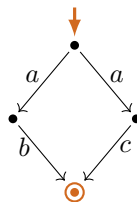
# Process semantics equality $=_{[\cdot]_P}$

► Fewer identities hold for  $=_{[\cdot]_P}$  than for  $=_{[\cdot]_L}$ :  $=_{[\cdot]_P} \not\subseteq =_{[\cdot]_L}$ .



$a \cdot (b + c)$

$\neq_{[\cdot]_P}$



$a \cdot b + a \cdot c$

$\neq_{[\cdot]_P}$

# Milner's proof system **Mil**

*Axioms:*

$$(A1) \quad e + (f + g) = (e + f) + g$$

$$(A2) \quad e + 0 = e$$

$$(A3) \quad e + f = f + e$$

$$(A4) \quad e + e = e$$

$$(A5) \quad e \cdot (f \cdot g) = (e \cdot f) \cdot g$$

$$(A6) \quad (e + f) \cdot g = e \cdot g + f \cdot g$$

$$(A7) \quad e = 1 \cdot e$$

$$(A8) \quad e = e \cdot 1$$

$$(A9) \quad 0 = 0 \cdot e$$

$$(A10) \quad e^* = 1 + e \cdot e^*$$

$$(A11) \quad e^* = (1 + e)^*$$

$$\text{But: } e \cdot (f + g) \neq e \cdot f + e \cdot g$$

$$\text{But: } e \cdot 0 \neq 0$$

*Inference rules:* rules of equational logic *plus*

$$\frac{e = f \cdot e + g}{e = f^* \cdot g} \text{RSP}^* \text{ (if } f \text{ does not terminate immediately)}$$

## Milner's Question (Q1)

Is **Mil** complete with respect to  $=_{[\cdot]_P}$ ? (Does  $e =_{[\cdot]_P} f \implies e =_{\text{Mil}} f$  hold?)

# Milner's questions

(Q1) Complete axiomatization:

*Is the proof system Mil complete for  $\llbracket \cdot \rrbracket_P$ ?*

(Q2)  $\llbracket \cdot \rrbracket_P$ -Expressibility:

*What structural property characterizes process graphs that are  $\llbracket \cdot \rrbracket_P$ -expressible?*

# Milner's questions

## (Q1) Complete axiomatization:

*Is the proof system Mil complete for  $\llbracket \cdot \rrbracket_P$ ?*

## (Q2) $\llbracket \cdot \rrbracket_P$ -Expressibility:

*What structural property characterizes process graphs that are  $\llbracket \cdot \rrbracket_P$ -expressible?*

- ▶ is decidable (Baeten/Corradini/G, 2007)

# Milner's questions

## (Q1) Complete axiomatization:

*Is the proof system **Mil** **complete** for  $=_{\llbracket \cdot \rrbracket_P}$ ?*

## (Q2) $\llbracket \cdot \rrbracket_P$ -Expressibility:

*What **structural property** characterizes  
process graphs that are  $\llbracket \cdot \rrbracket_P$ -expressible ?*

- ▶ is decidable (Baeten/Corradini/G, 2007)
- ▶ partial new answer (G/Fokkink, 2020):
  - ▶ bisimulation collapse has **loop existence & elimination property (LEE)**  
if expressible by **under-star-1-free** regular expression



# Milner's questions

## (Q1) Complete axiomatization:

Is the proof system *Mil complete* for  $\llbracket \cdot \rrbracket_P$ ?

- ▶ series of partial completeness results for:
  - ▶ exitless iterations (Fokkink, 1998)
  - ▶ with a stronger fixed-point rule (G, 2006)
  - ▶ under-star 1-free, and without 0 (Corradini/de Nicola/Labella, 2004)
  - ▶ with 0 but under-star-1-free (G/Fokkink, 2020)

## (Q2) $\llbracket \cdot \rrbracket_P$ -Expressibility:

What *structural property* characterizes process graphs that are  $\llbracket \cdot \rrbracket_P$ -expressible?

- ▶ is decidable (Baeten/Corradini/G, 2007)
- ▶ partial new answer (G/Fokkink, 2020):
  - ▶ bisimulation collapse has loop existence & elimination property (LEE) if expressible by under-star-1-free regular expression

# Milner's questions

## (Q1) Complete axiomatization:

Is the proof system *Mil complete* for  $\llbracket \cdot \rrbracket_P$ ?

- ▶ Yes! (G, 2022, proof summary, employing LEE and crystallization)
- ▶ series of partial completeness results for:
  - ▶ exitless iterations (Fokkink, 1998)
  - ▶ with a stronger fixed-point rule (G, 2006)
  - ▶ under-star 1-free, and without 0 (Corradini/de Nicola/Labella, 2004)
  - ▶ with 0 but under-star-1-free (G/Fokkink, 2020)

## (Q2) $\llbracket \cdot \rrbracket_P$ -Expressibility:

What *structural property* characterizes process graphs that are  $\llbracket \cdot \rrbracket_P$ -expressible?

- ▶ is decidable (Baeten/Corradini/G, 2007)
- ▶ partial new answer (G/Fokkink, 2020):
  - ▶ bisimulation collapse has loop existence & elimination property (LEE) if expressible by under-star-1-free regular expression

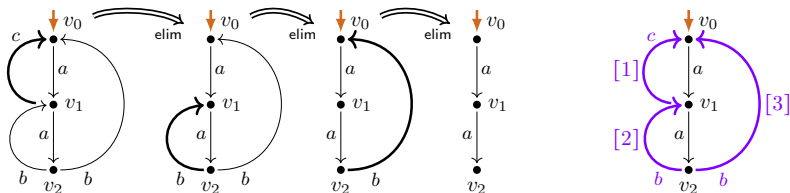
## Question (Q2) specialized

(Q2)<sub>0</sub>  $P$ -Expressibility and  $P$ - $(*/\perp)$ -Expressibility:

What *structural property* characterizes:

- ▶ process graphs that are  $P$ -expressible ?  
(... in the *image of  $P$* ?)
- ▶ process graphs that are  $P$ -expressible by  $(*/\perp)$  regular expressions?  
(... in the *image of  $(*/\perp)$  expressions under  $P$* ?)

# Loop Existence and Elimination (LEE)



# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to **it**.
- (L3) Termination is **only** possible at the **start vertex**.

# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to **it**.
- (L3) Termination is **only** possible at the **start vertex**.

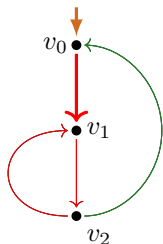


# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~

# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to **it**.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~

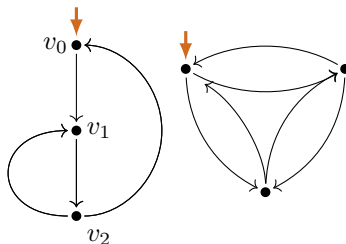


# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



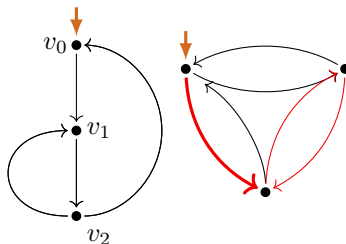
(L1), ~~(L2)~~

# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



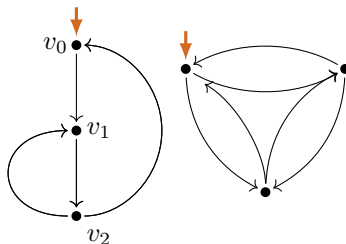
(L1), ~~(L2)~~

# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



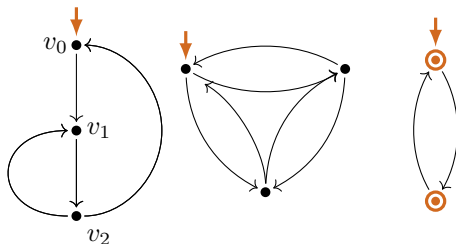
(L1), ~~(L2)~~

# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



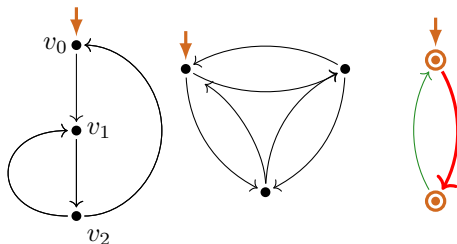
(L1), ~~(L2)~~

# Loop graphs (interpretations of innermost iterations without 1)

## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



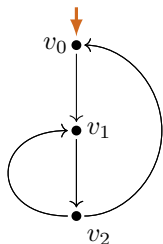
(L1), ~~(L2)~~

# Loop graphs (interpretations of innermost iterations without 1)

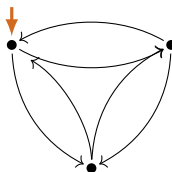
## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~

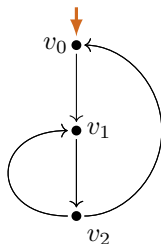


# Loop graphs (interpretations of innermost iterations without 1)

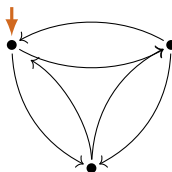
## Definition

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

- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~



# Loop graphs (interpretations of innermost iterations without 1)

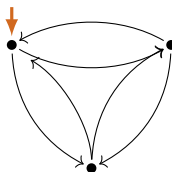
## Definition

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

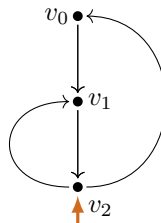
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~



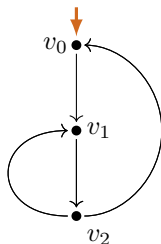


# Loop graphs (interpretations of innermost iterations without 1)

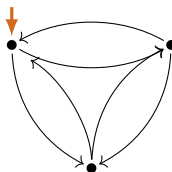
## Definition

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

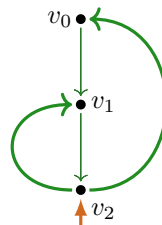
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~

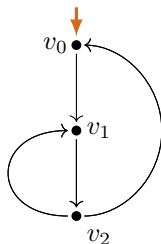


# Loop graphs (interpretations of innermost iterations without 1)

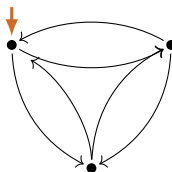
## Definition

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

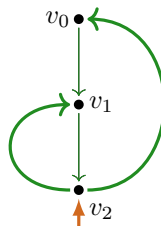
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~



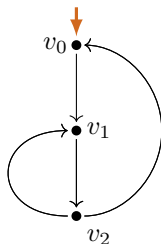
**loop chart**

# Loop graphs (interpretations of innermost iterations without 1)

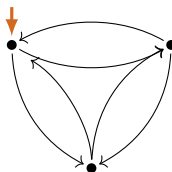
## Definition

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

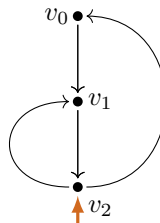
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~



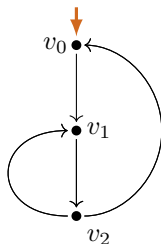
**loop chart**

# Loop graphs (interpretations of innermost iterations without 1)

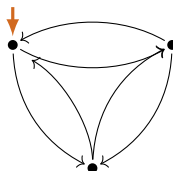
## Definition

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

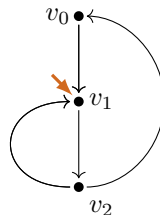
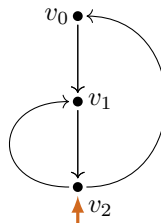
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~ **loop chart**

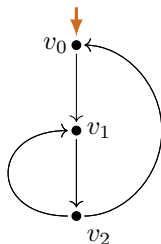


# Loop graphs (interpretations of innermost iterations without 1)

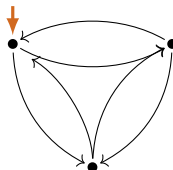
## Definition

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

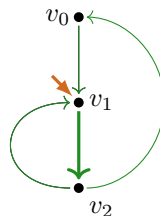
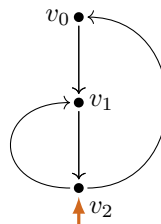
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~ **loop chart**



# Loop graphs (interpretations of innermost iterations without 1)

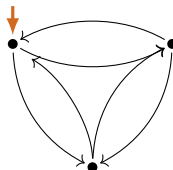
## Definition

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

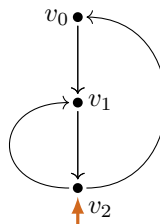
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



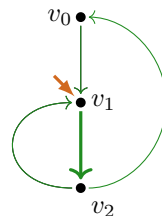
(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~



loop chart



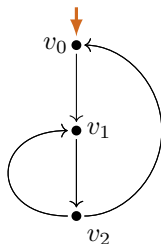
loop chart

# Loop graphs (interpretations of innermost iterations without 1)

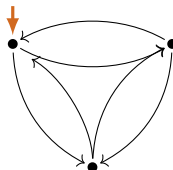
## Definition

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

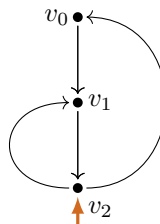
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



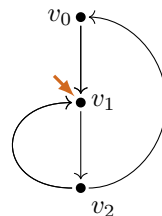
(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~



loop chart



loop chart

# Loop graphs (interpretations of innermost iterations without 1)

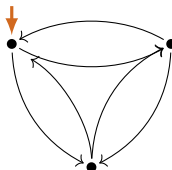
## Definition

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

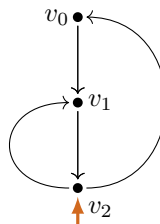
- (L1) There is an infinite path from the **start vertex**.
- (L2) Every infinite path from the **start vertex** returns to it.
- (L3) Termination is **only** possible at the **start vertex**.



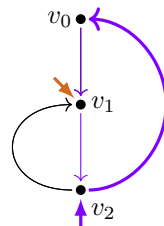
(L1), ~~(L2)~~



(L1), (L2), ~~(L3)~~



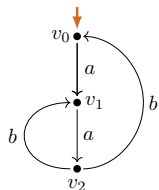
loop chart



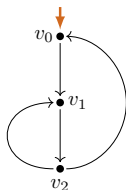
loop subchart



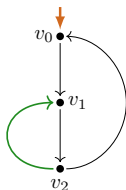
# Loop existence and elimination (example 1)



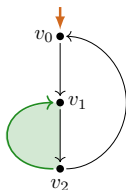
# Loop existence and elimination (example 1)



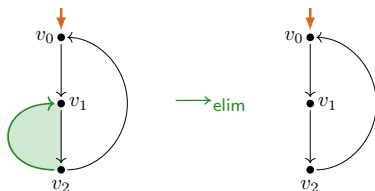
# Loop existence and elimination (example 1)



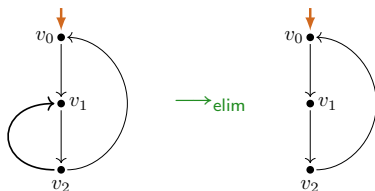
# Loop existence and elimination (example 1)



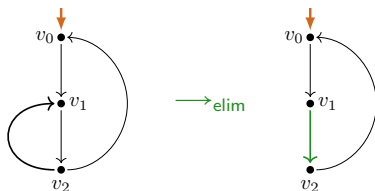
# Loop existence and elimination (example 1)



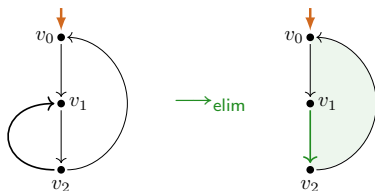
# Loop existence and elimination (example 1)



# Loop existence and elimination (example 1)

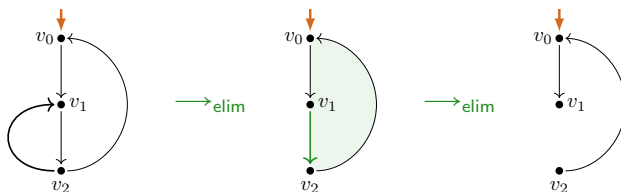


# Loop existence and elimination (example 1)

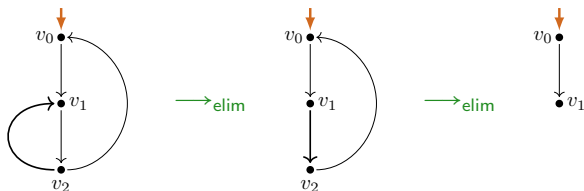




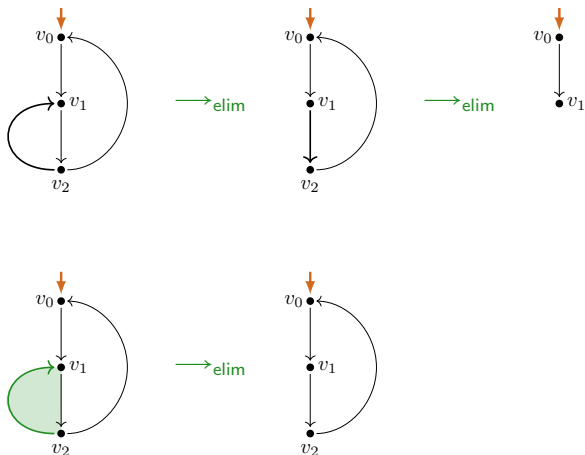
# Loop existence and elimination (example 1)



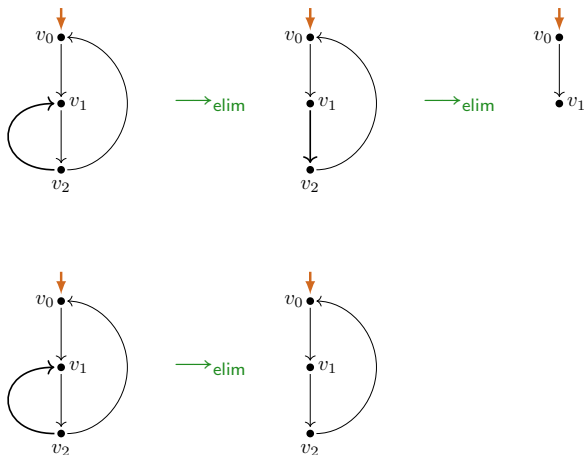
# Loop existence and elimination (example 1)



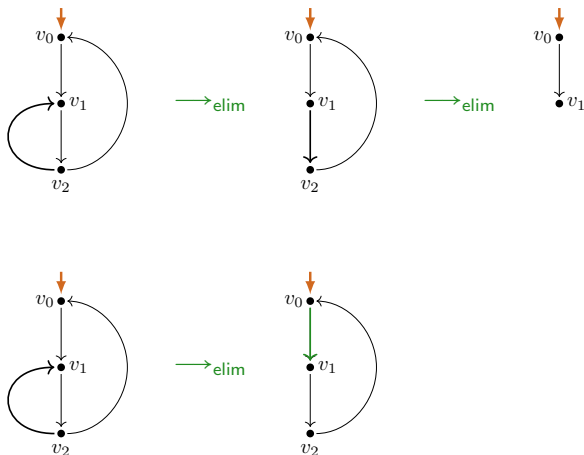
# Loop existence and elimination (example 1)



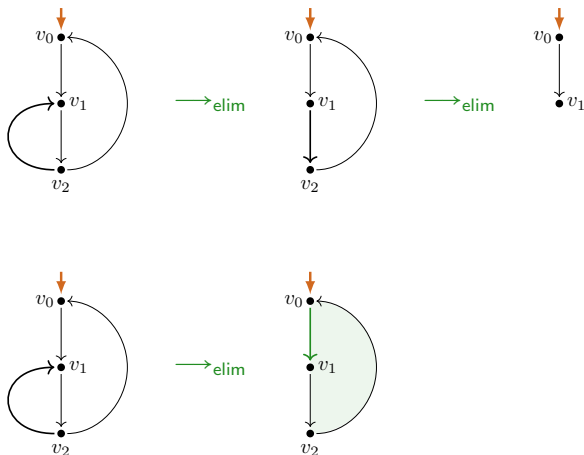
# Loop existence and elimination (example 1)



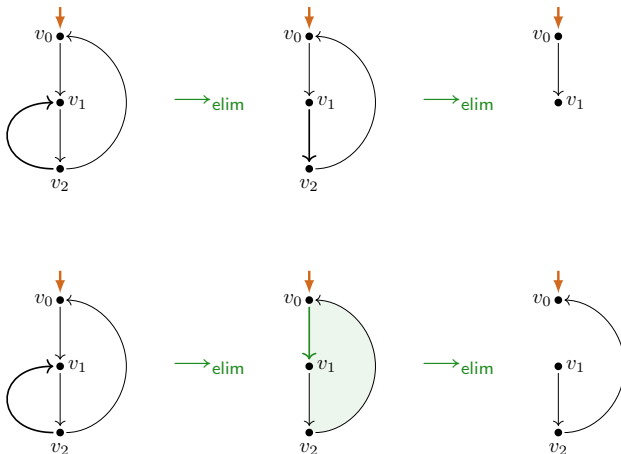
# Loop existence and elimination (example 1)



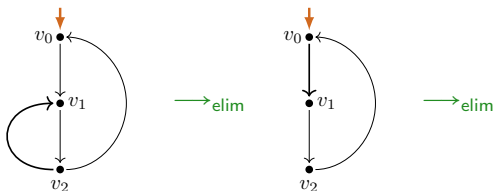
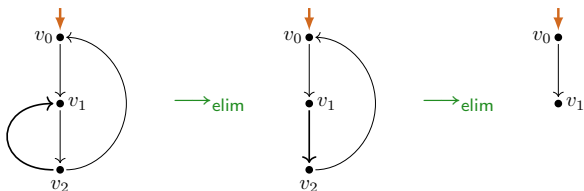
# Loop existence and elimination (example 1)



# Loop existence and elimination (example 1)

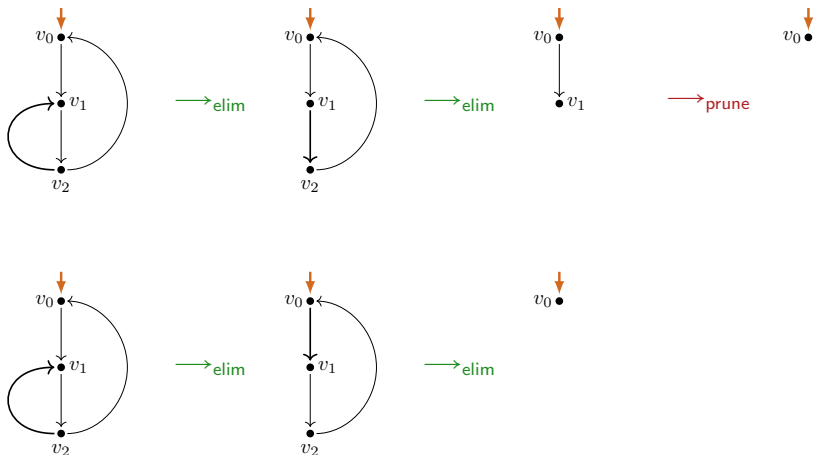


# Loop existence and elimination (example 1)

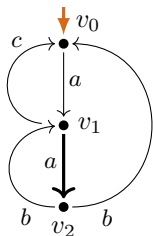




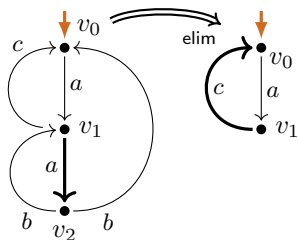
# Loop existence and elimination (example 1)



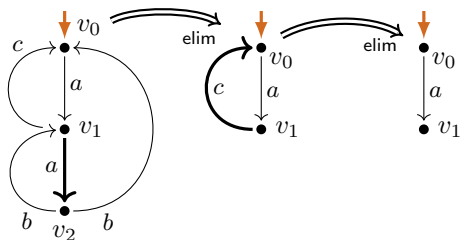
## Loop existence and elimination (example 2)



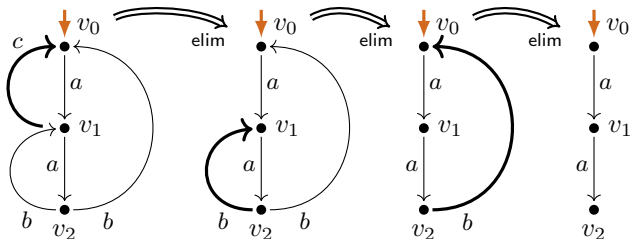
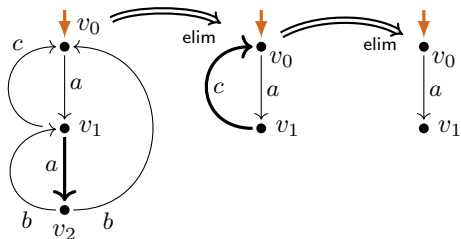
## Loop existence and elimination (example 2)



## Loop existence and elimination (example 2)



## Loop existence and elimination (example 2)



# LEE

## Definition

A chart  $\mathcal{C}$  satisfies **LEE** (*loop existence and elimination*) if:

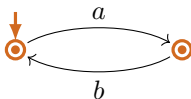
$$\exists \mathcal{C}_0 \left( \mathcal{C} \xrightarrow[\text{elim}]{*} \mathcal{C}_0 \not\rightarrow_{\text{elim}} \wedge \mathcal{C}_0 \text{ permits no infinite path} \right).$$

# LEE

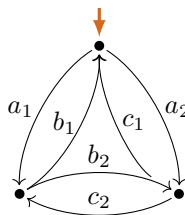
## Definition

A chart  $\mathcal{C}$  satisfies **LEE** (*loop existence and elimination*) if:

$$\exists \mathcal{C}_0 \left( \mathcal{C} \xrightarrow{*}_{\text{elim}} \mathcal{C}_0 \not\rightarrow_{\text{elim}} \right. \\ \left. \wedge \mathcal{C}_0 \text{ permits no infinite path} \right).$$

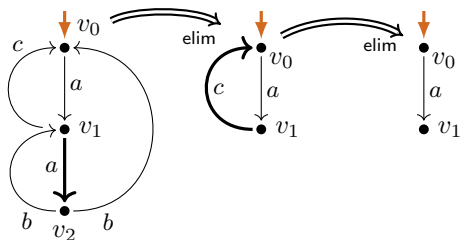


**LEE**

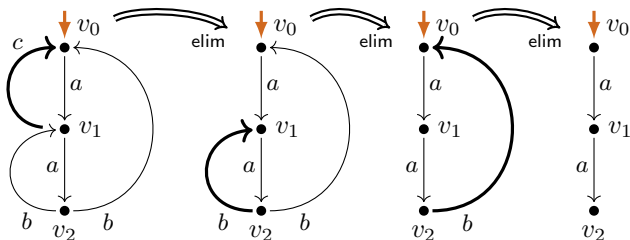
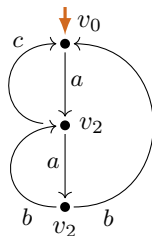


**LEE**

# LEE

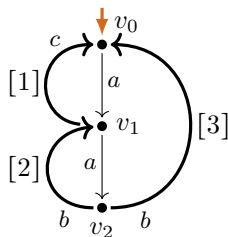
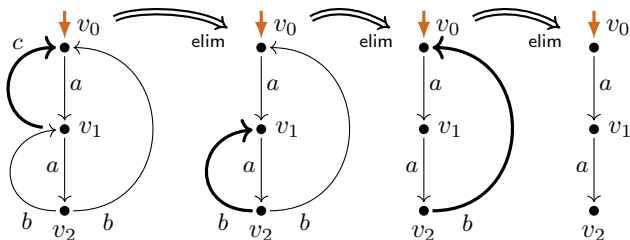
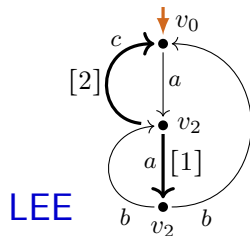
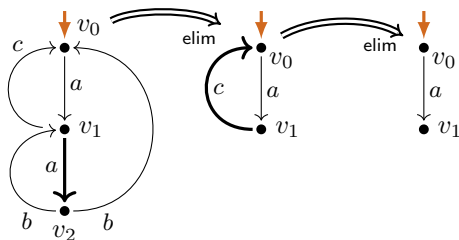


LEE

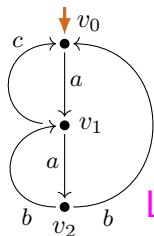




# LEE

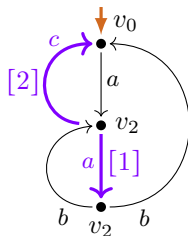


# LEE

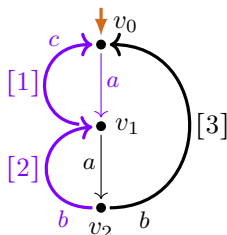


LEE-graph

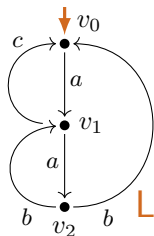
LEE-witness



LEE-witness

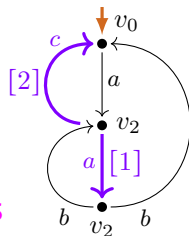


# Layered LEE

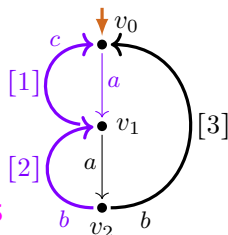


LLEE-graph

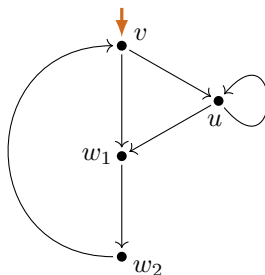
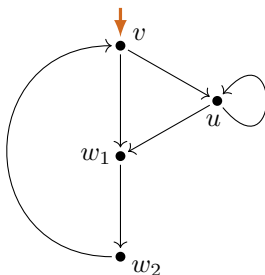
layered  
LLEE-witness



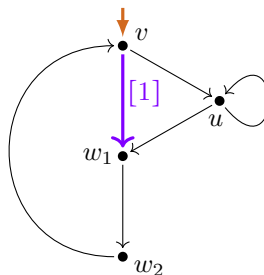
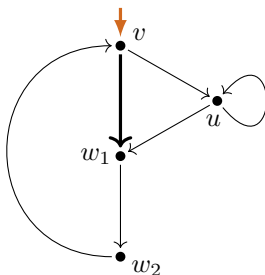
layered  
LLEE-witness



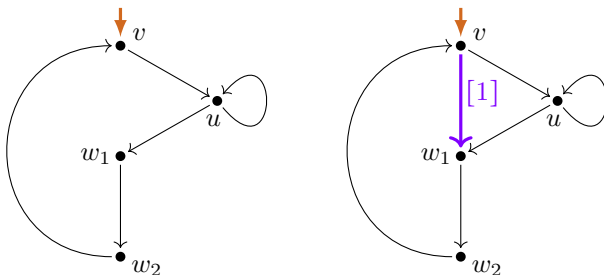
# Layered LEE-witness (LLEE-witness)



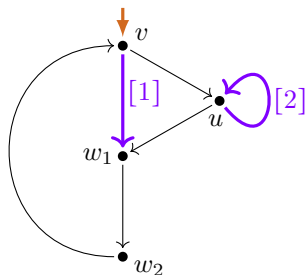
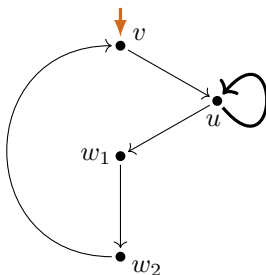
# Layered LEE-witness (LLEE-witness)



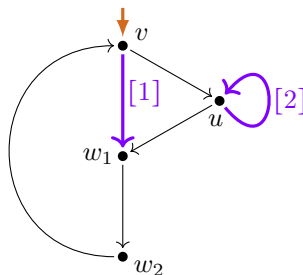
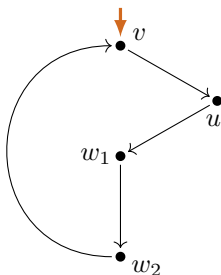
# Layered LEE-witness (LLEE-witness)



# Layered LEE-witness (LLEE-witness)

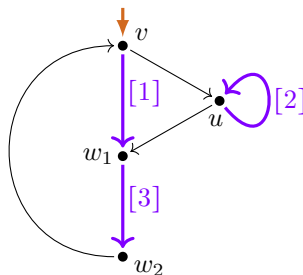
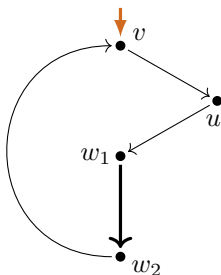


# Layered LEE-witness (LLEE-witness)

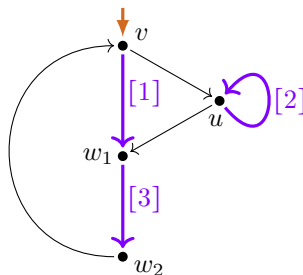
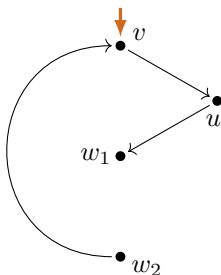




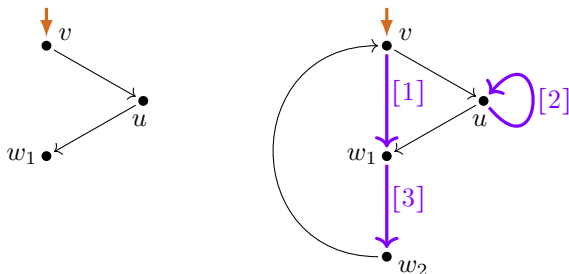
# Layered LEE-witness (LLEE-witness)



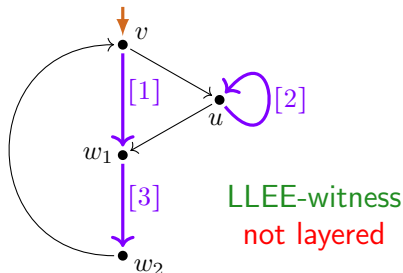
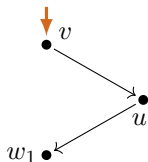
# Layered LEE-witness (LLEE-witness)



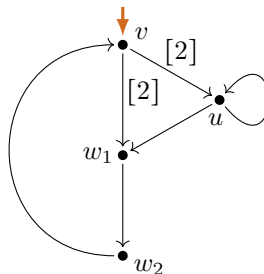
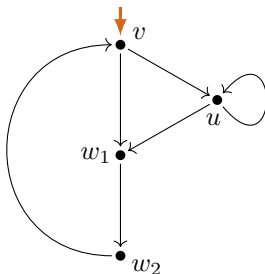
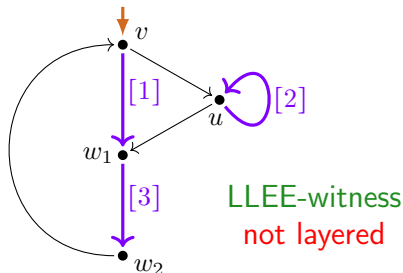
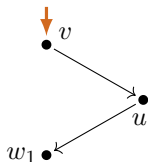
# Layered LEE-witness (LLEE-witness)



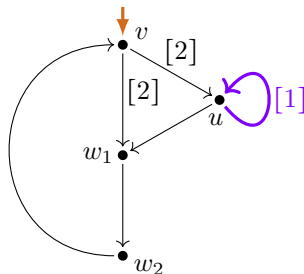
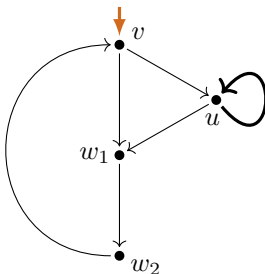
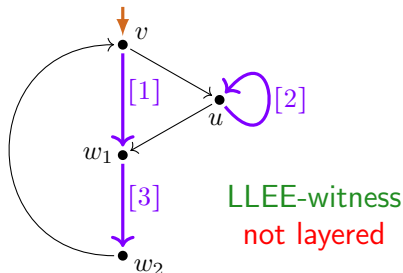
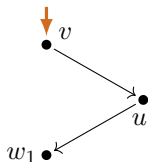
# Layered LEE-witness (LEEE-witness)



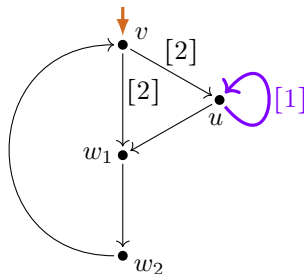
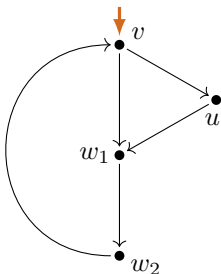
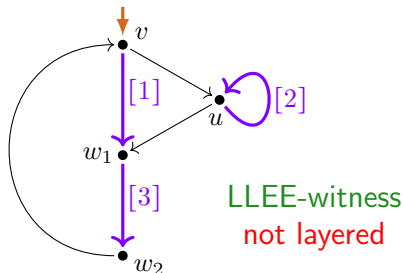
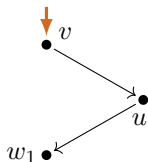
# Layered LEE-witness (LLEE-witness)



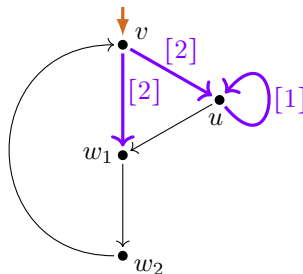
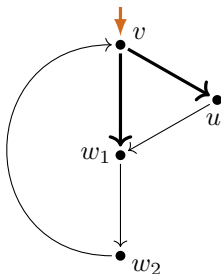
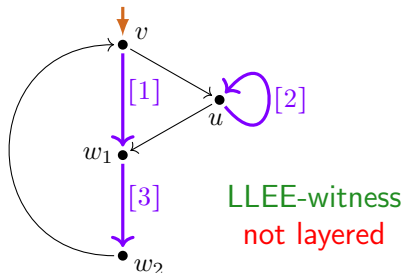
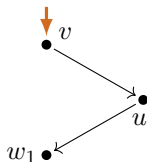
# Layered LEE-witness (LLEE-witness)



# Layered LEE-witness (LLEE-witness)

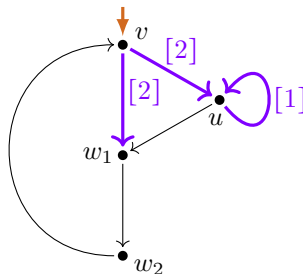
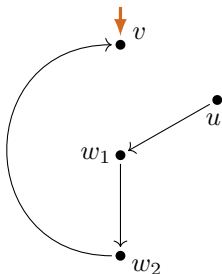
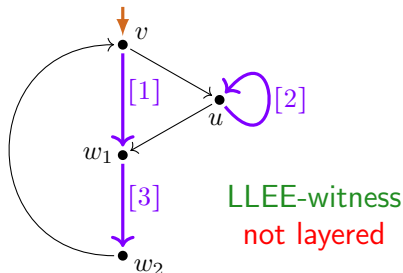
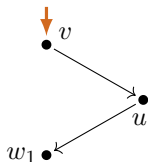


# Layered LEE-witness (LLEE-witness)

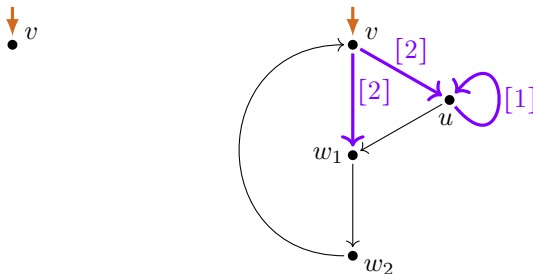
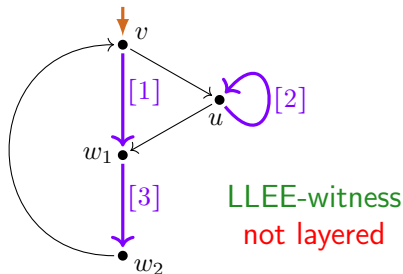
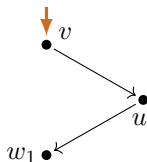




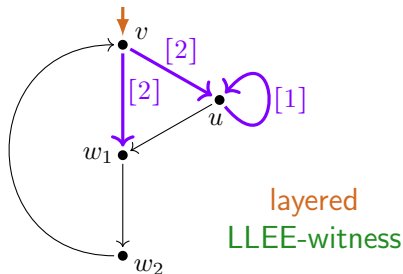
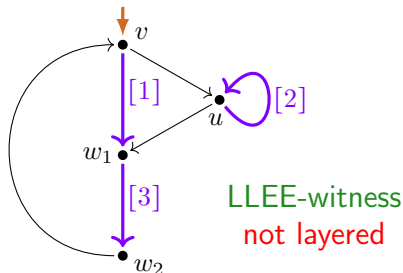
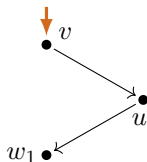
# Layered LEE-witness (LLEE-witness)



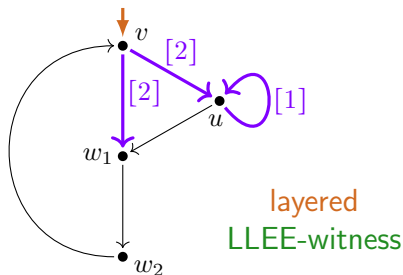
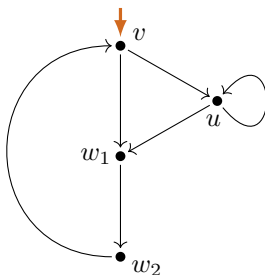
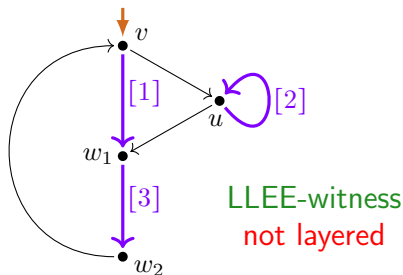
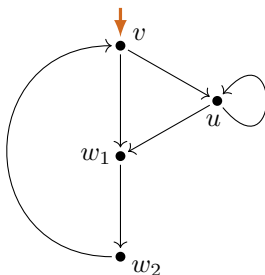
# Layered LEE-witness (LLEE-witness)



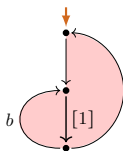
# Layered LEE-witness (LLEE-witness)



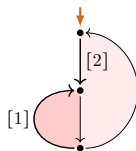
# Layered LEE-witness (LLEE-witness)



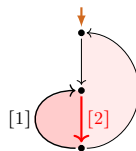
## 7 LEE-witnesses



layered

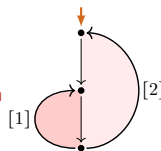


layered

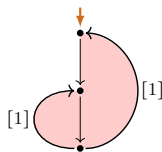
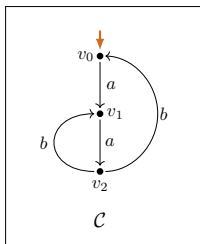


not layered

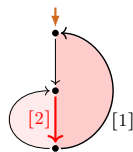
make layered



layered

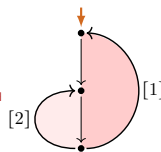


layered



not layered

make layered



layered

# Loop elimination: properties

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

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

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

## Lemma

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

# Loop elimination: properties

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

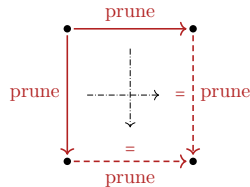
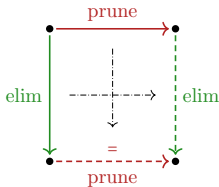
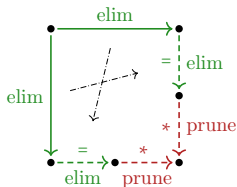
- ▶ removing the loop-entry transition(s)
- ▶ garbage collection

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

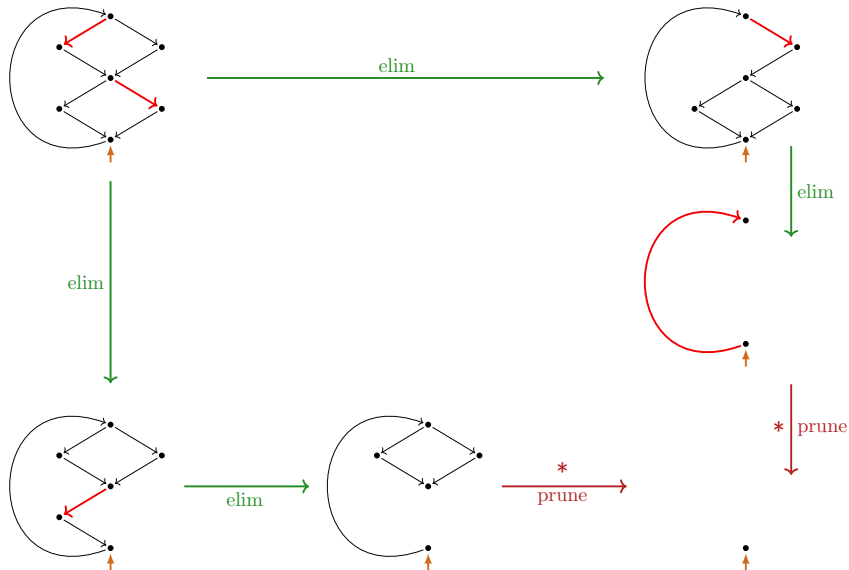
## Lemma

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

(ii)  $\xrightarrow{\text{elim}} \cup \xrightarrow{\text{prune}}$  is decreasing [Van Oostrom, de Bruijn]



# 'Critical pair': bi-loop elimination





# Loop elimination, and properties

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

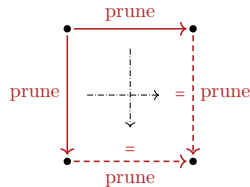
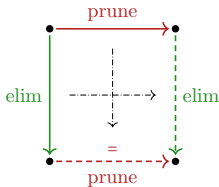
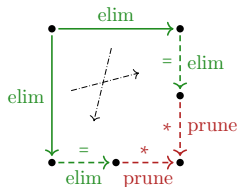
- ▶ removing the loop-entry transition(s)
- ▶ garbage collection

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

## Lemma

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

(ii)  $\xrightarrow{\text{elim}} \cup \xrightarrow{\text{prune}}$  is decreasing, and so due to (i) locally confluent.



# Loop elimination, and properties

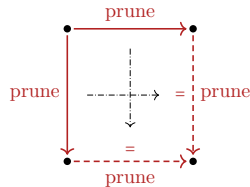
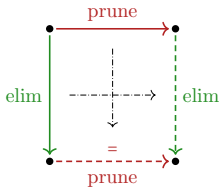
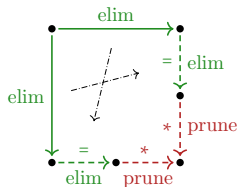
$\xrightarrow{\text{elim}}$  : eliminate a transition-induced loop by:

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

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

## Lemma

- (i)  $\xrightarrow{\text{elim}} \cup \xrightarrow{\text{prune}}$  is terminating.
- (ii)  $\xrightarrow{\text{elim}} \cup \xrightarrow{\text{prune}}$  is decreasing, and so due to (i) locally confluent.
- (iii)  $\xrightarrow{\text{elim}} \cup \xrightarrow{\text{prune}}$  is confluent.



# Structure property LEE

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

**Lemma** (by using termination and confluence)

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

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

# Structure property LEE

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

**Lemma** (by using termination and confluence)

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

- (i)  $\text{LEE}(G)$ .
- (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

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

**Lemma** (by using termination and confluence)

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

- (i)  $\text{LEE}(G)$ .
- (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

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

**Lemma** (by using termination and confluence)

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

- (i)  $\text{LEE}(G)$ .
- (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  $\text{LEE}(G)$  for process graphs  $G$  is in **PTIME**.

# Interpretation/extraction correspondences with LEE

( $\Leftarrow$  G/Fokkink 2020, G 2021)

**(Int)**<sub>P</sub><sup>(\*/±)</sup>: *P*-(\*/±)-expressible graphs have the *structural property* LEE.

Process **interpretations** *P*(*e*) of (\*/±) regular expressions *e* are finite process graphs that **satisfy** LEE.

**(Extr)**<sub>P</sub>: LEE *implies*  $\llbracket \cdot \rrbracket_P$ -*expressibility*

From every finite process graph *G* with LEE  
a regular expression *e* can be **extracted** such that  $G \Leftrightarrow P(e)$ .

# Interpretation/extraction correspondences with LEE

( $\Leftarrow$  G/Fokkink 2020, G 2021)

**(Int)**<sub>P</sub><sup>(\*/ $\perp$ )</sup>: *P*-(\*/ $\perp$ )-expressible graphs have the *structural property* LEE.

Process **interpretations**  $P(e)$  of (\*/ $\perp$ ) regular expressions  $e$  are finite process graphs that **satisfy** LEE.

**(Extr)**<sub>P</sub>: LEE *implies*  $\llbracket \cdot \rrbracket_P$ -*expressibility*

From every finite process graph  $G$  with LEE  
a regular expression  $e$  can be **extracted** such that  $G \rightleftharpoons P(e)$ .

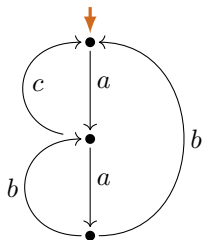
**(Coll)**: LEE *is preserved under collapse*

The class of finite process graphs with LEE  
is **closed under bisimulation collapse**.

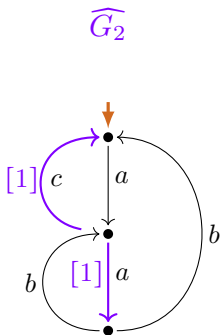


# Expression extraction using **LLEE** (G/Fokkink 2020, G 2021/22)

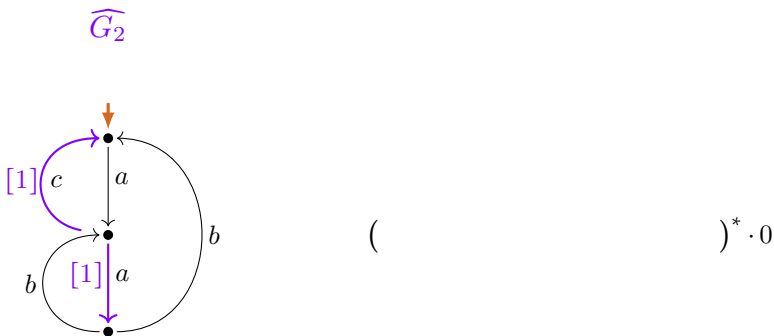
$G_2$



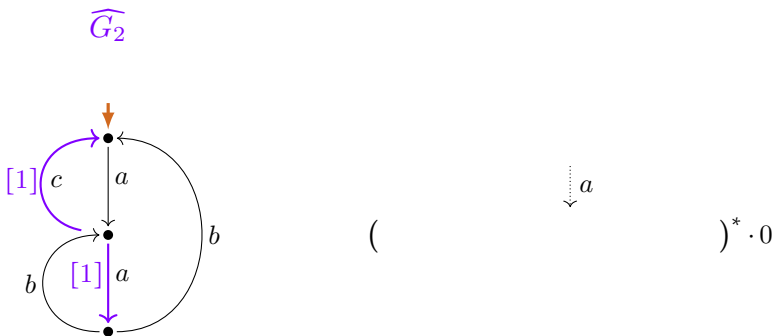
# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)



# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)

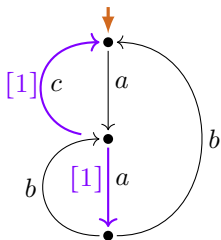


# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)



# Expression extraction using **LLEE** (G/Fokkink 2020, G 2021/22)

$\widehat{G}_2$

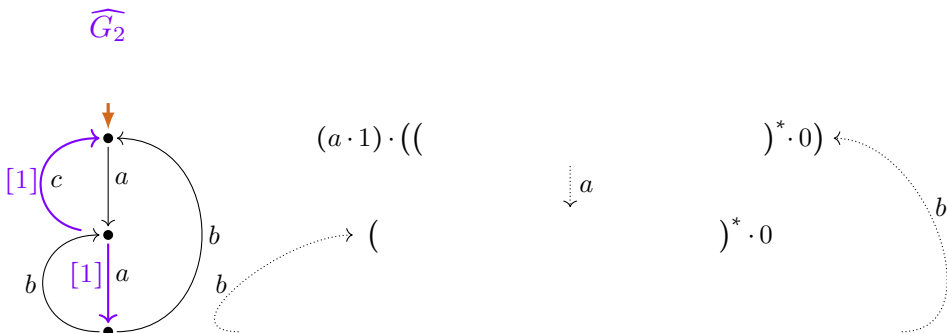


$$(a \cdot 1) \cdot (( \quad )^* \cdot 0)$$

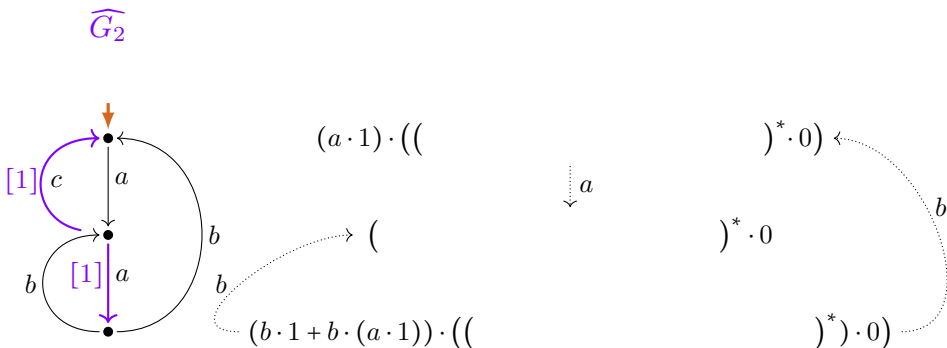
$$( \quad )^* \cdot 0$$

$\downarrow a$

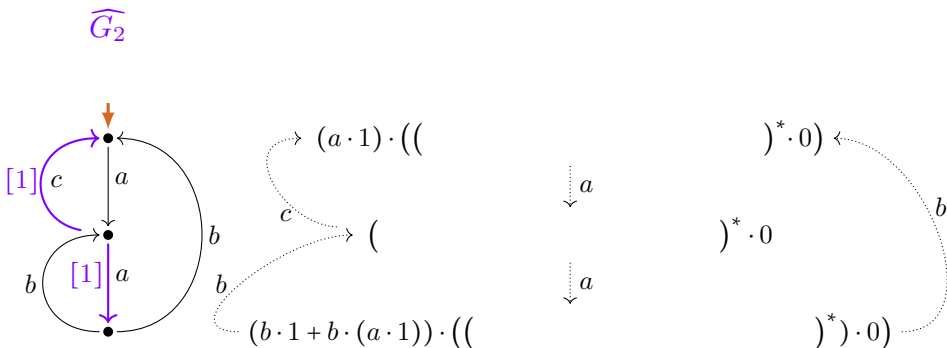
# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)



# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)

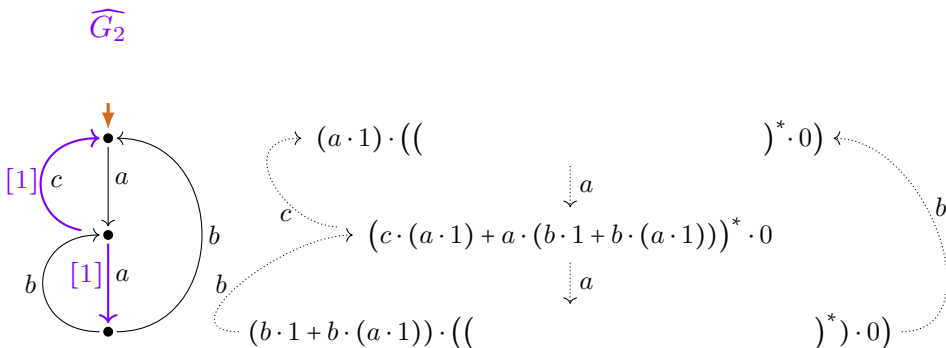


# Expression extraction using **LEE** (G/Fokkink 2020, G 2021/22)



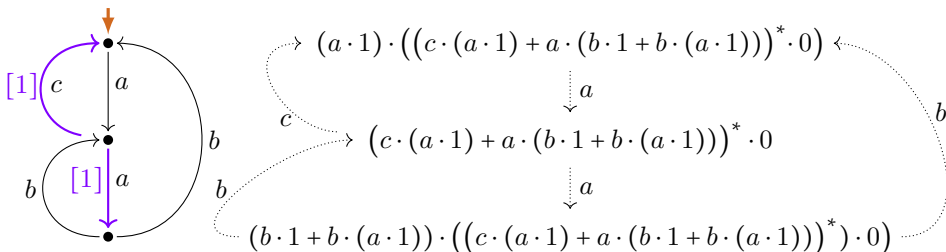


# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)

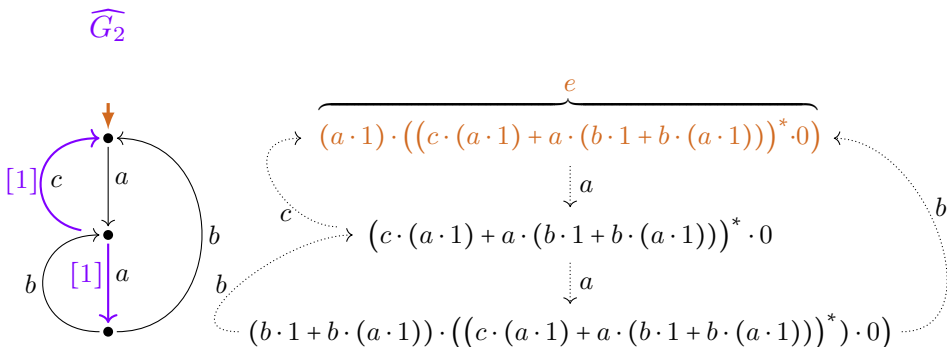


# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)

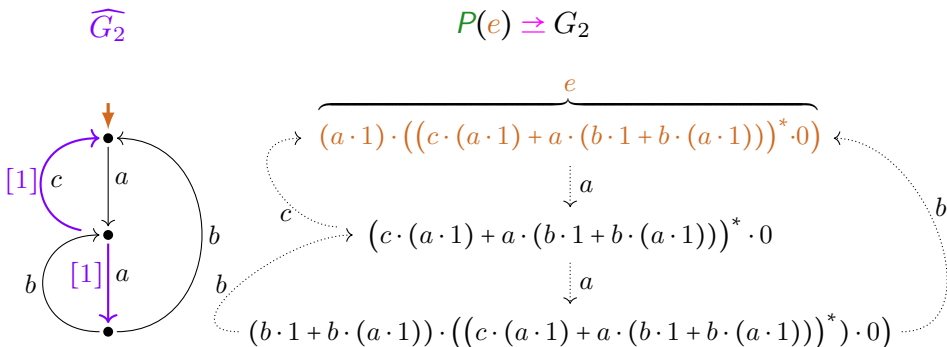
$\widehat{G}_2$



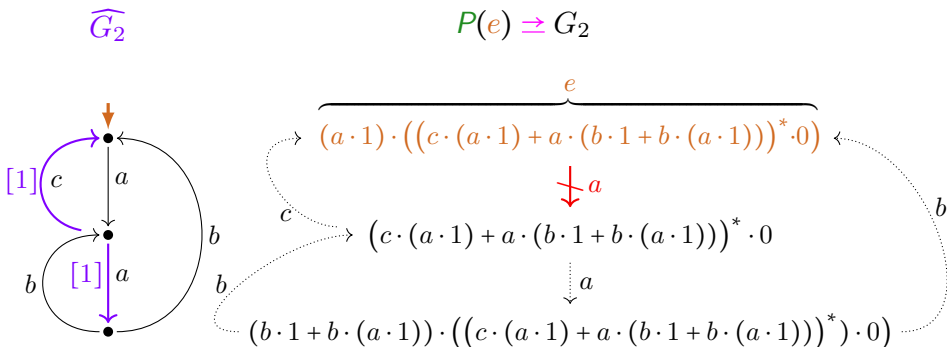
# Expression extraction using **LLEE** (G/Fokkink 2020, G 2021/22)



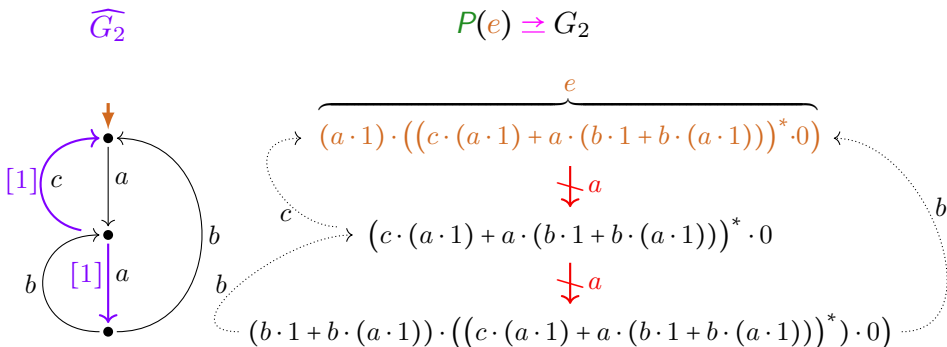
# Expression extraction using **LLEE** (G/Fokkink 2020, G 2021/22)



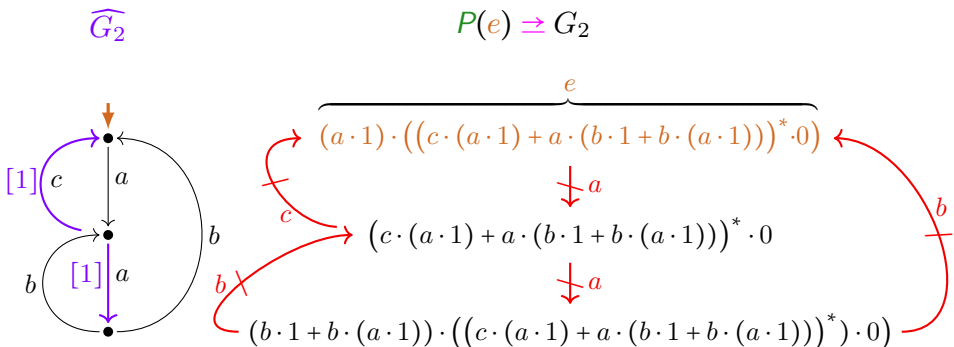
# Expression extraction using **LLEE** (G/Fokkink 2020, G 2021/22)



# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)



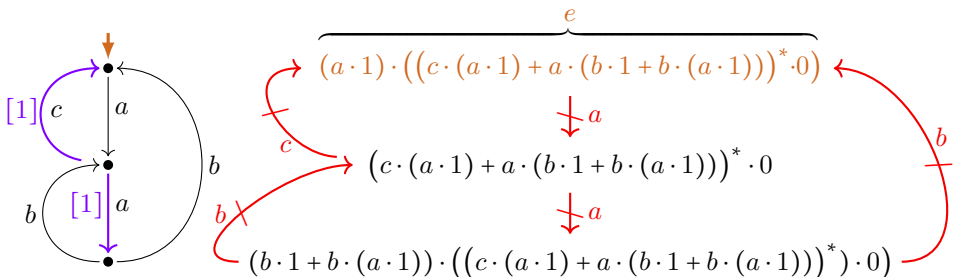
# Expression extraction using **LLEE** (G/Fokkink 2020, G 2021/22)



# Expression extraction using LLEE (G/Fokkink 2020, G 2021/22)

$\widehat{G}_2$

$P(e) \Rightarrow G_2 \not\Rightarrow P(e)$





# Interpretation of extracted expression

$G'_2$

$P(e) = G'_2$

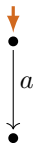


$$\overbrace{(a \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)}^e$$

# Interpretation of extracted expression

$G'_2$

$P(e) = G'_2$



$$\begin{array}{c}
 \overbrace{(a \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)}^e \\
 \downarrow a \\
 (1 \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)
 \end{array}$$

# Interpretation of extracted expression

$G'_2$

$P(e) = G'_2$



$$\overbrace{(a \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)}^e$$

$\downarrow a$

$$(1 \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)$$

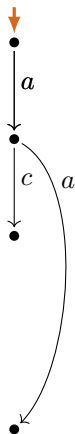
$\downarrow c$

$$((1 \cdot (a \cdot 1)) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1))))^* \cdot 0$$

# Interpretation of extracted expression

$G'_2$

$P(e) = G'_2$



$$\overbrace{(a \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)}^e$$

$\downarrow a$

$$(1 \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)$$

$\downarrow c$

$$((1 \cdot (a \cdot 1)) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0$$

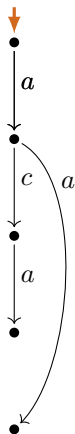
$a$

$$((1 \cdot (b \cdot 1 + b \cdot (a \cdot 1))) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0$$

# Interpretation of extracted expression

$G'_2$

$P(e) = G'_2$



$$\overbrace{(a \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)}^e$$

$\downarrow a$

$$(1 \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)$$

$\downarrow c$

$$((1 \cdot (a \cdot 1)) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)$$

$\downarrow a$

$$((1 \cdot 1) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)$$

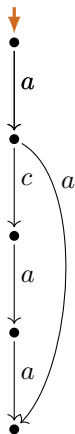
$a$

$$((1 \cdot (b \cdot 1 + b \cdot (a \cdot 1))) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)$$

# Interpretation of extracted expression

$G'_2$

$P(e) = G'_2$

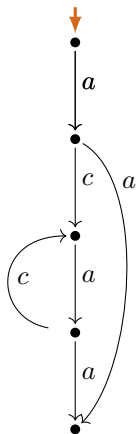


$$\begin{array}{c}
 \overbrace{(a \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)}^e \\
 \downarrow a \\
 (1 \cdot 1) \cdot ((c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0) \\
 \downarrow c \\
 ((1 \cdot (a \cdot 1)) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0) \\
 \downarrow a \\
 ((1 \cdot 1) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0) \\
 \downarrow a \\
 ((1 \cdot (b \cdot 1 + b \cdot (a \cdot 1))) \cdot (c \cdot (a \cdot 1) + a \cdot (b \cdot 1 + b \cdot (a \cdot 1)))^* \cdot 0)
 \end{array}$$

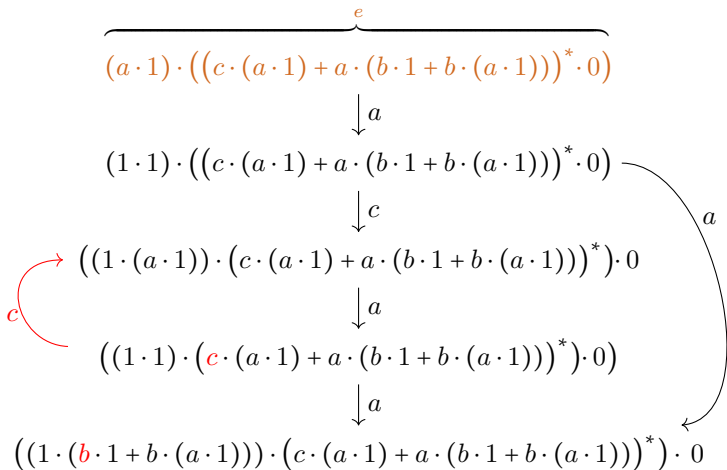
A curved arrow labeled 'a' connects the second expression to the final expression.

# Interpretation of extracted expression

$G'_2$

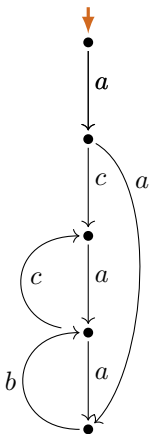


$P(e) = G'_2$

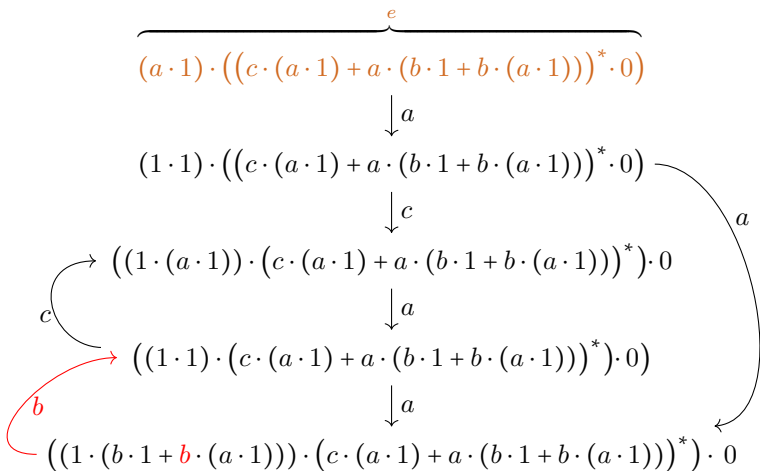


# Interpretation of extracted expression

$G'_2$



$P(e) = G'_2$

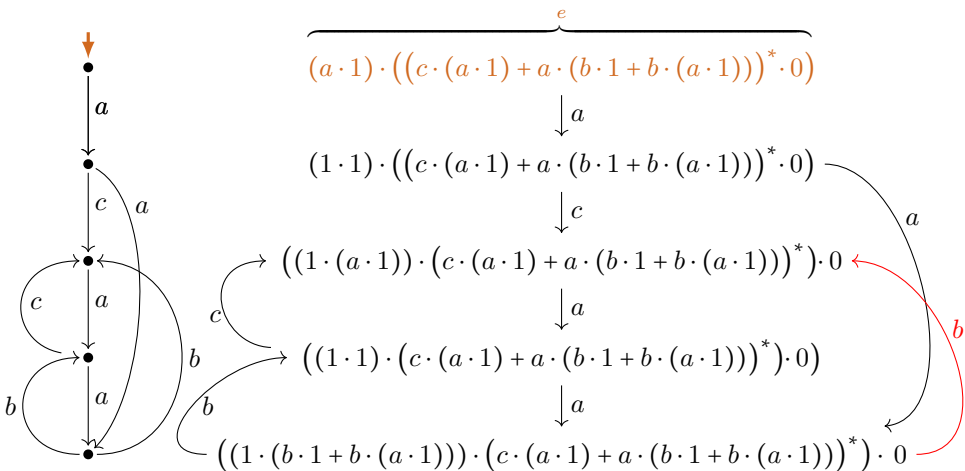




# Interpretation of extracted expression

$G'_2$

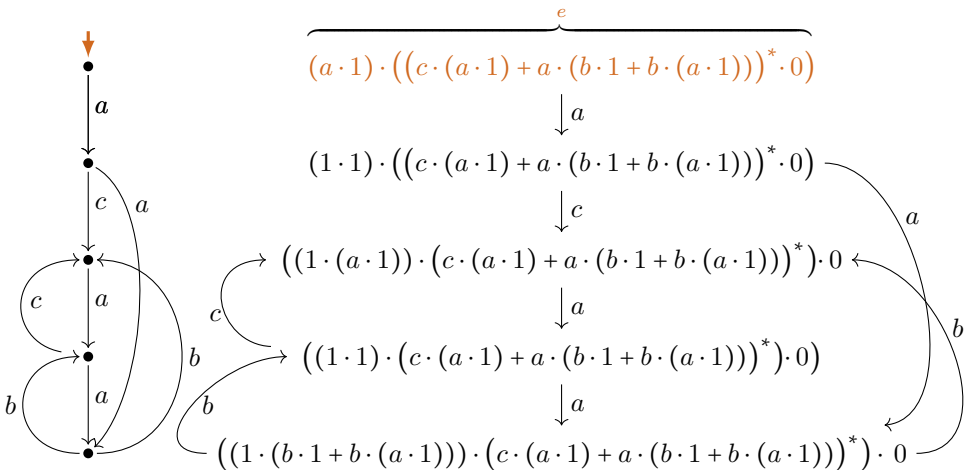
$P(e) = G'_2$



# Interpretation of extracted expression

$G'_2$

$$P(e) = G'_2 \xrightarrow{\text{pink}} G_2$$

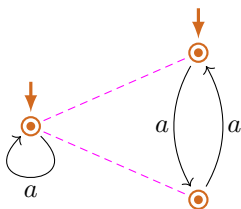




# LEE under bisimulation

## Observation

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



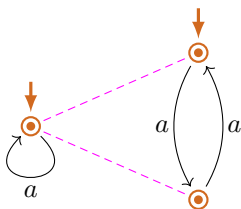
LEE

¬LEE

# LEE under bisimulation

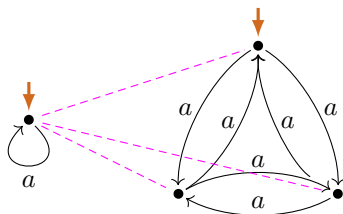
## Observation

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



LEE

¬LEE



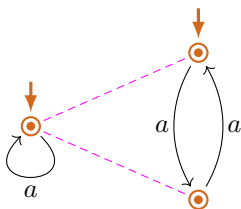
LEE

¬LEE

# LEE under bisimulation

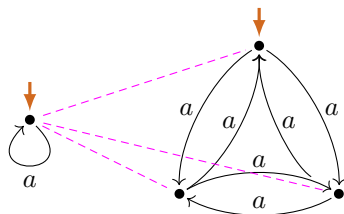
## Observation

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



LEE

¬LEE



LEE

¬LEE

# LEE under functional bisimulation

## Lemma

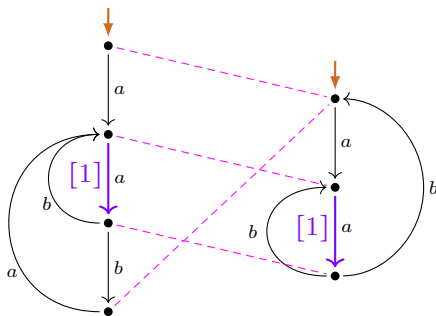
(i) LEE is preserved by *functional bisimulations*:

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

## Proof (Idea).

Use loop elimination in  $G_1$  to carry out loop elimination in  $G_2$ .

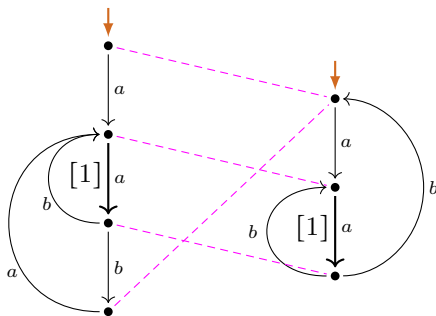
# Collapsing LEE-witnesses



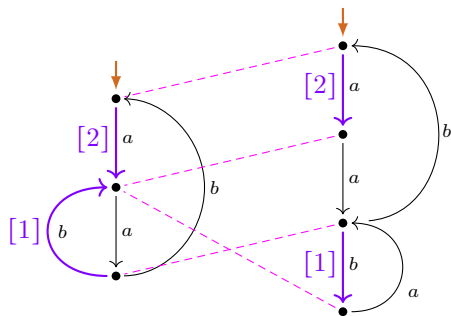
$$P(a(a(b + ba))^* \cdot 0)$$



# Collapsing LEE-witnesses



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



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

# LEE under functional bisimulation / bisimulation collapse

## Lemma

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

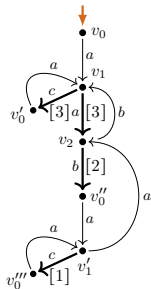
$$\text{LEE}(G_1) \wedge G_1 \rightrightarrows G_2 \implies \text{LEE}(G_2).$$

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

$$\text{LEE}(G) \wedge G \text{ has bisimulation collapse } C \implies \text{LEE}(C).$$

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

(no 1-transitions!)



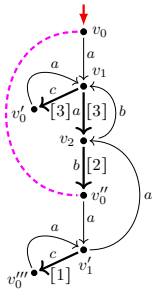
(C1.1)

## Lemma

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

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

(no 1-transitions!)

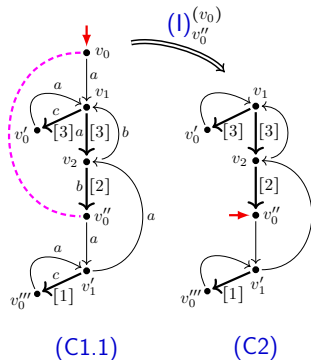


(C1.1)

## Lemma

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

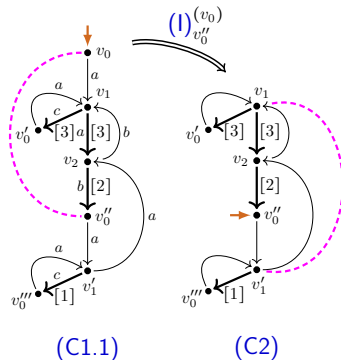
# LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20) (no 1-transitions!)



## Lemma

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

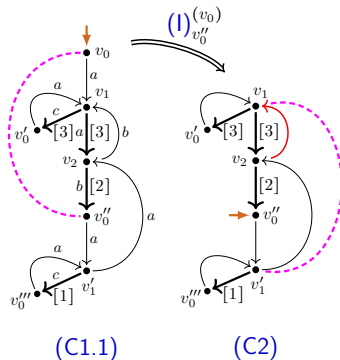
# LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20) (no 1-transitions!)



## Lemma

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

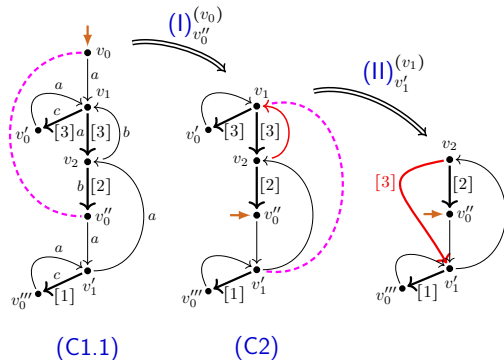
# LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20) (no 1-transitions!)



## Lemma

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

# LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20) (no 1-transitions!)

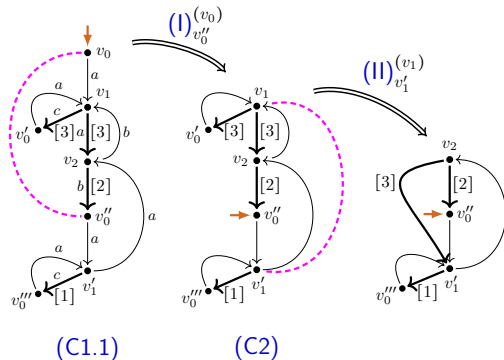


## Lemma

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



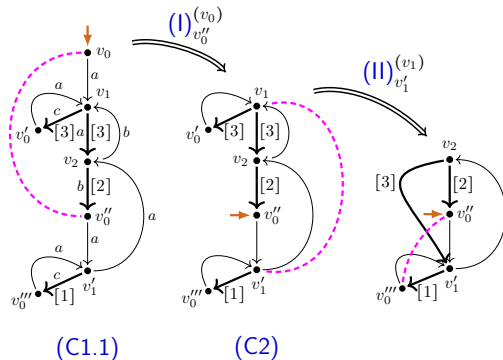
# LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20) (no 1-transitions!)



## Lemma

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

# LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20) (no 1-transitions!)

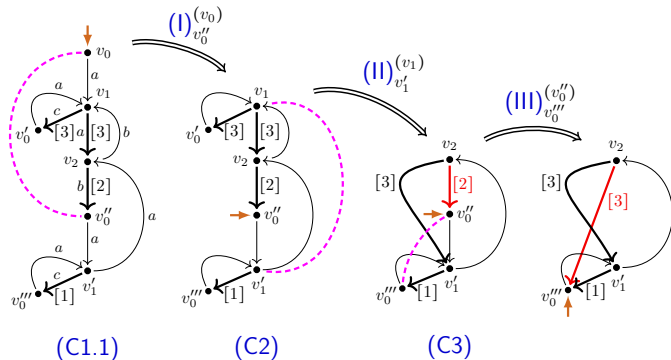


## Lemma

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

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

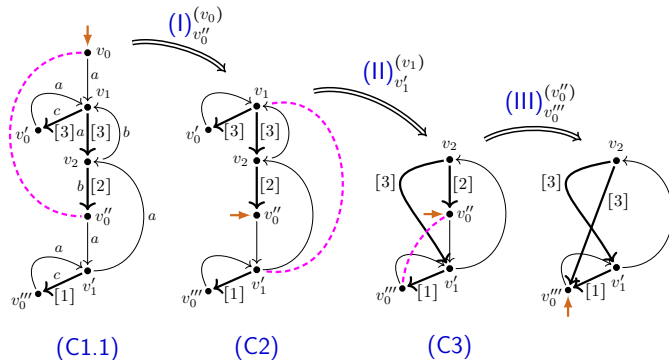
(no 1-transitions!)



## Lemma

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

# LLEE-preserving collapse of LLEE-charts (G/Fokkink, LICS'20) (no 1-transitions!)



## Lemma

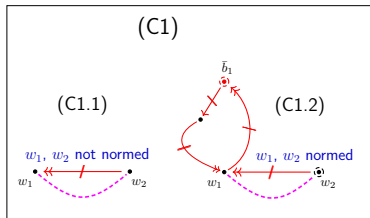
*The bisimulation collapse of a LLEE-chart is again a LLEE-chart.*



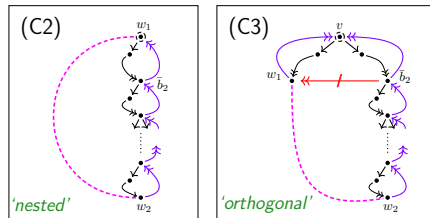
The bisimulation collapse of a LLEE-chart is again a LLEE-chart.

# Reduced bisimilarity redundancies in LLEE-graphs (no 1-trans.!) (G/Fokkink, LICS'20)

$w_1, w_2$  in different scc's



$w_1, w_2$  in the same scc

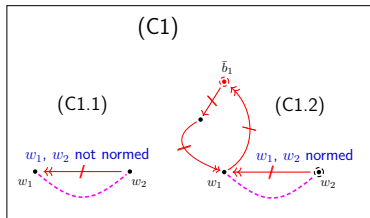


## Lemma

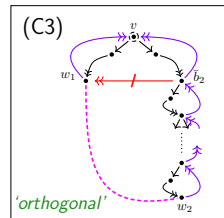
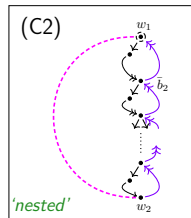
Every *not collapsed* LLEE-graph contains bisimilar vertices  $w_1 \neq w_2$  of kind (C1), (C2), or (C3) (a *reduced bisimilarity redundancy*  $\langle w_1, w_2 \rangle$ ):

# Reduced bisimilarity redundancies in LLEE-graphs (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-graph 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-graph can be eliminated LLEE-preservingly.

# Properties of LEE-charts

Theorem ( $\Leftarrow$  G/Fokkink, 2020)

A process graph  $G$

is  $\llbracket \cdot \rrbracket_P$ -expressible by an under-star-1-free regular expression

(i.e.  $P$ -expressible modulo bisimilarity by an  $(\pm \backslash *)$  reg. expr.)

if and only if

the bisimulation collapse of  $G$  satisfies LEE.



# Properties of LEE-charts

Theorem ( $\Leftarrow$  G/Fokkink, 2020)

A process graph  $G$

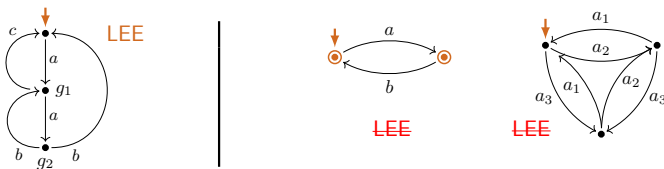
is  $\llbracket \cdot \rrbracket_P$ -expressible by an under-star-1-free regular expression

(i.e.  $P$ -expressible modulo bisimilarity by an  $(\pm \setminus *)$  reg. expr.)

if and only if

the bisimulation collapse of  $G$  satisfies LEE.

Hence  $\llbracket \cdot \rrbracket_P$ -expressible | **not**  $\llbracket \cdot \rrbracket_P$ -expressible by 1-free regular expressions:



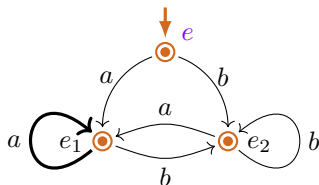
# 1-LEE

$\hat{=}$  sharing via 1-transitions facilitates LEE

# Failure of LEE in general (example)

$(a^* \cdot b^*)^*$  not  $(*/\perp)$  !

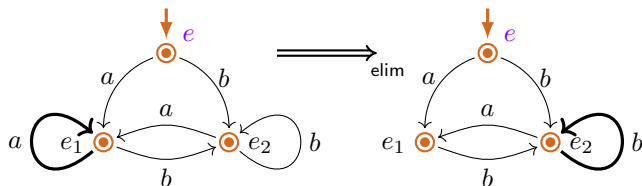
$P((a^* \cdot b^*)^*)$



# Failure of LEE in general (example)

$(a^* \cdot b^*)^*$  not  $(*/\perp)$  !

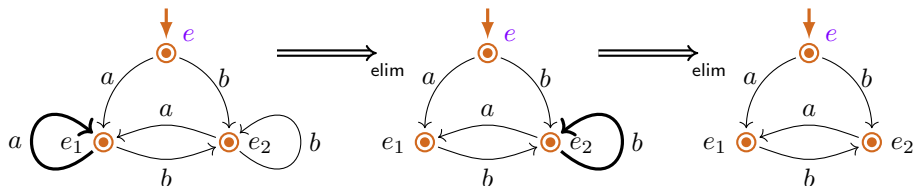
$P((a^* \cdot b^*)^*)$



# Failure of LEE in general (example)

$(a^* \cdot b^*)^*$  not  $(*/\perp)$  !

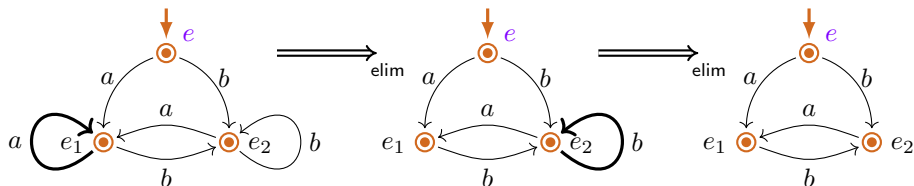
$P((a^* \cdot b^*)^*)$



# Failure of LEE in general (example)

$(a^* \cdot b^*)^*$  not  $(*/\perp)$  !

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

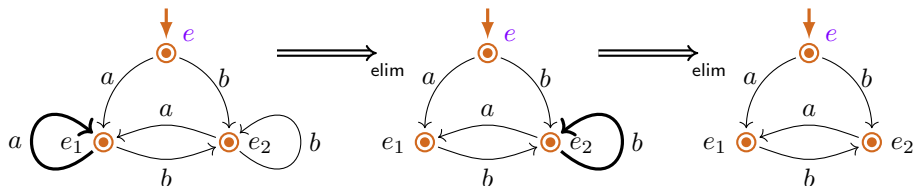


no loop subchart,  
but infinite paths

# Failure of LEE in general (example)

$(a^* \cdot b^*)^*$  not  $(*/\perp)$  !

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



no loop subchart,  
but infinite paths

# 1-Graphs and induced graphs

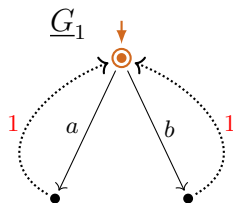
## Definition

$$\xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \xrightarrow{a} \quad \hat{=} \quad \xrightarrow{(a)}$$

induced  $a$ -transitions, for  $a \in A$

$$\xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \Downarrow \quad \hat{=} \quad \Downarrow^{(1)}$$

induced termination.





# 1-Graphs and induced graphs

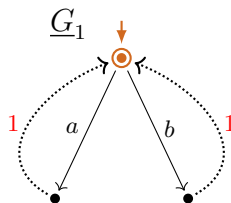
## Definition

$$v_1 \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \xrightarrow{a} v_2 \quad \hat{=} \quad v_1 \xrightarrow{(a)} v_2$$

induced  $a$ -transitions, for  $a \in A$

$$v \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \Downarrow \quad \hat{=} \quad v \Downarrow^{(1)}$$

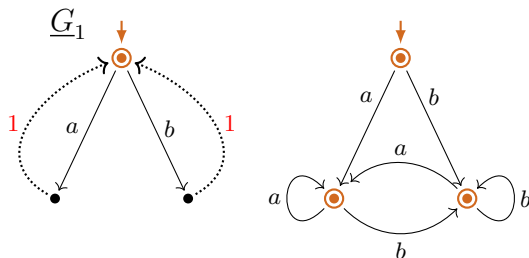
induced termination.



# 1-Graphs and induced graphs

## Definition

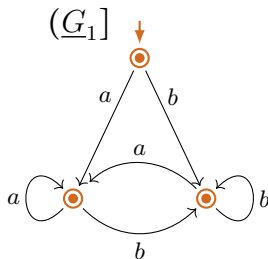
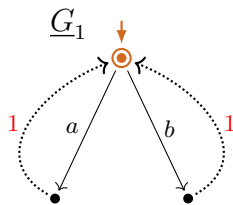
$$\begin{aligned}
 v_1 \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \xrightarrow{a} v_2 &\hat{=} v_1 \xrightarrow{(a)} v_2 && \text{induced } a\text{-transitions, for } a \in A \\
 v \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \Downarrow &\hat{=} v \Downarrow^{(1)} && \text{induced termination.}
 \end{aligned}$$



# 1-Graphs and induced graphs

## Definition

$$\begin{aligned}
 v_1 \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \xrightarrow{a} v_2 &\equiv v_1 \xrightarrow{(a)} v_2 && \text{induced } a\text{-transitions, for } a \in A \\
 v \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \Downarrow &\equiv v \Downarrow^{(1)} && \text{induced termination.}
 \end{aligned}$$



# 1-Graphs and induced graphs

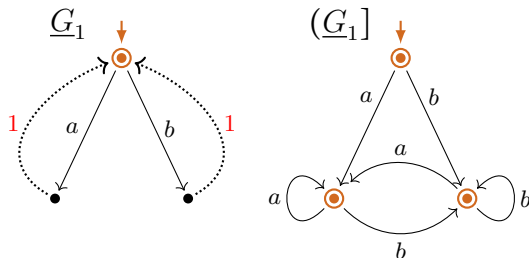
## Definition

$$\begin{aligned}
 v_1 \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \xrightarrow{a} v_2 &\equiv v_1 \xrightarrow{[a]} v_2 && \text{induced } a\text{-transitions, for } a \in A \\
 v \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \Downarrow &\equiv v \Downarrow^{(1)} && \text{induced termination.}
 \end{aligned}$$

## Definition

The induced (process) graph of a 1-graph  $\underline{G} = \langle V, A, 1, v_s, \rightarrow, \Downarrow \rangle$  is:

$$(\underline{G}) = \langle V, A, v_s, \xrightarrow{[\cdot]}, \Downarrow^{(1)} \rangle.$$



# 1-Graphs and induced graphs

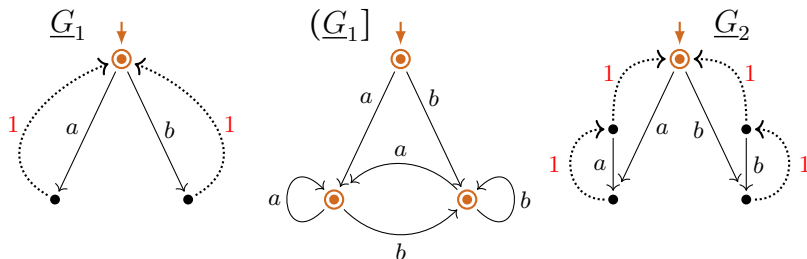
## Definition

$$\begin{aligned}
 v_1 \xrightarrow{1} \cdot \cdots \cdot \xrightarrow{1} \cdot \xrightarrow{a} v_2 &\hat{=} v_1 \xrightarrow{[a]} v_2 && \text{induced } a\text{-transitions, for } a \in A \\
 v \xrightarrow{1} \cdot \cdots \cdot \xrightarrow{1} \cdot \Downarrow &\hat{=} v \Downarrow^{(1)} && \text{induced termination.}
 \end{aligned}$$

## Definition

The induced (process) graph of a 1-graph  $\underline{G} = \langle V, A, 1, v_s, \rightarrow, \Downarrow \rangle$  is:

$$(\underline{G}) = \langle V, A, v_s, \xrightarrow{[\cdot]}, \Downarrow^{(1)} \rangle.$$



# 1-Graphs and induced graphs

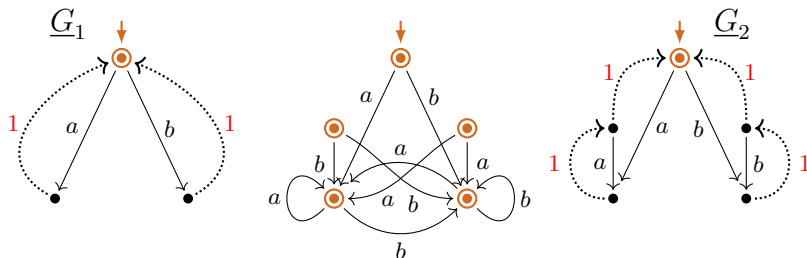
## Definition

$$\begin{aligned}
 v_1 \xrightarrow{1} \cdot \cdots \cdot \xrightarrow{1} \cdot \xrightarrow{a} v_2 &\hat{=} v_1 \xrightarrow{(a)} v_2 && \text{induced } a\text{-transitions, for } a \in A \\
 v \xrightarrow{1} \cdot \cdots \cdot \xrightarrow{1} \cdot \Downarrow &\hat{=} v \Downarrow^{(1)} && \text{induced termination.}
 \end{aligned}$$

## Definition

The induced (process) graph of a 1-graph  $\underline{G} = \langle V, A, 1, v_s, \rightarrow, \Downarrow \rangle$  is:

$$(\underline{G}) = \langle V, A, v_s, \xrightarrow{[\cdot]}, \Downarrow^{(1)} \rangle.$$



# 1-Graphs and induced graphs

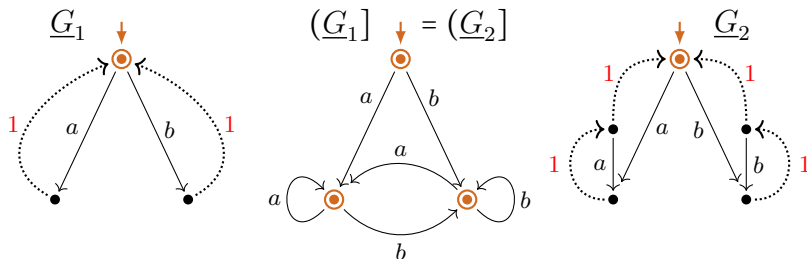
## Definition

$$\begin{aligned}
 v_1 \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \xrightarrow{a} v_2 &\hat{=} v_1 \xrightarrow{[a]} v_2 && \text{induced } a\text{-transitions, for } a \in A \\
 v \xrightarrow{1} \cdot \dots \cdot \xrightarrow{1} \cdot \Downarrow &\hat{=} v \Downarrow^{(1)} && \text{induced termination.}
 \end{aligned}$$

## Definition

The induced (process) graph of a 1-graph  $\underline{G} = \langle V, A, 1, v_s, \rightarrow, \Downarrow \rangle$  is:

$$(\underline{G}) = \langle V, A, v_s, \xrightarrow{[\cdot]}, \Downarrow^{(1)} \rangle.$$



# 1-LEE

## Definition

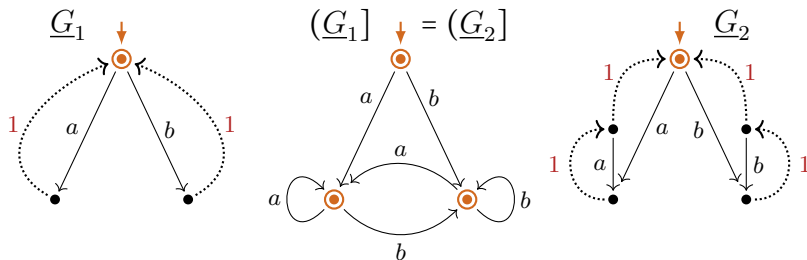
$1\text{-LEE}(G)$  holds for a graph  $G$ ,  
if  $G = (\underline{G}]$  for some weakly-guarded 1-graph  $\underline{G}$ .



# 1-LEE

## Definition

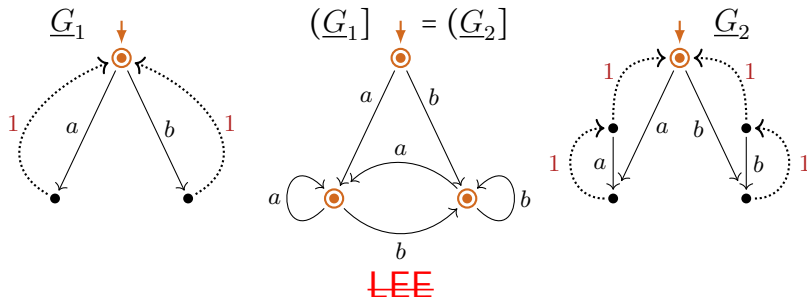
$1\text{-LEE}(G)$  holds for a graph  $G$ ,  
if  $G = (\underline{G}]$  for some weakly-guarded 1-graph  $\underline{G}$ .



# 1-LEE

## Definition

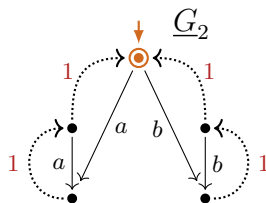
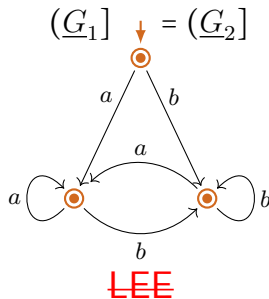
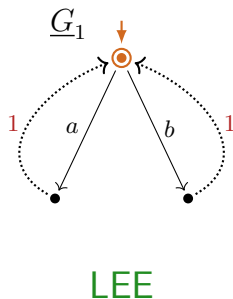
$1\text{-LEE}(G)$  holds for a graph  $G$ ,  
if  $G = (\underline{G}]$  for some weakly-guarded 1-graph  $\underline{G}$ .



# 1-LEE

## Definition

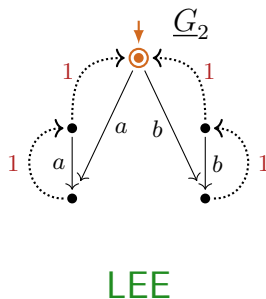
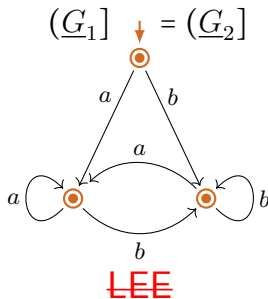
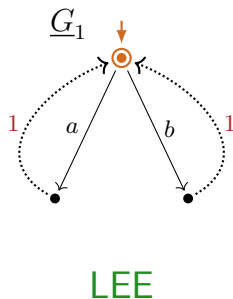
$1\text{-LEE}(G)$  holds for a graph  $G$ ,  
if  $G = (\underline{G}]$  for some **weakly-guarded 1-graph**  $\underline{G}$ .



# 1-LEE

## Definition

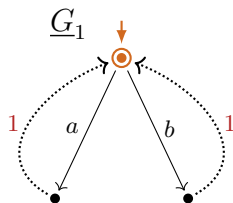
$1\text{-LEE}(G)$  holds for a graph  $G$ ,  
if  $G = (\underline{G}]$  for some **weakly-guarded 1-graph**  $\underline{G}$ .



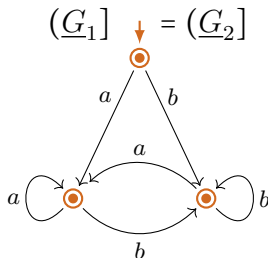
# 1-LEE

## Definition

$1\text{-LEE}(G)$  holds for a graph  $G$ ,  
if  $G = (\underline{G})$  for some weakly-guarded 1-graph  $\underline{G}$ .

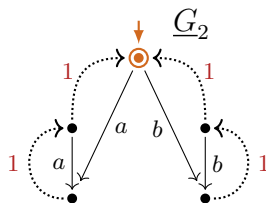


LEE



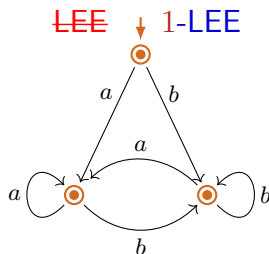
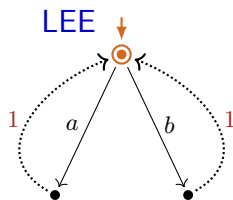
~~LEE~~

1-LEE



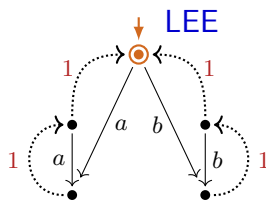
LEE

# 1-LEE holds for process interpretations



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

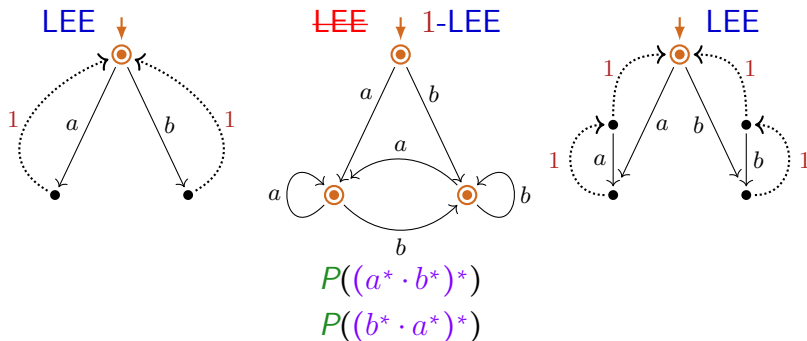
$P((b^* \cdot a^*)^*)$



# 1-LEE holds for process interpretations

## Lemma

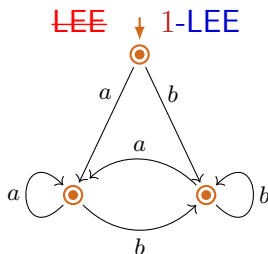
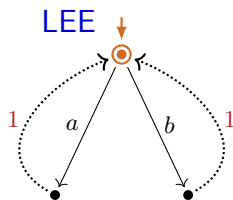
There is a 1-graph interpretation  $\underline{P}$  of reg. expression  $e$  as 1-graphs  $\underline{P}(e)$  such that for all  $e \in RExp$ : (i):  $LEE(\underline{P}(e))$ , (ii):  $(\underline{P}(e)) = \underline{P}(e)$ .



# 1-LEE holds for process interpretations

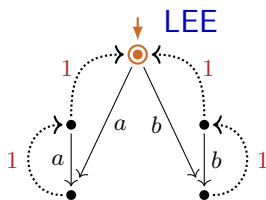
## Lemma

There is a **1-graph interpretation**  $\underline{P}$  of reg. expression  $e$  as 1-graphs  $\underline{P}(e)$  such that for all  $e \in RExp$ : (i):  $LEE(\underline{P}(e))$ , (ii):  $(\underline{P}(e)) = P(e)$ .



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

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



$\underline{P}((a^* \cdot b^*)^*)$

$\underline{P}((b^* \cdot a^*)^*)$



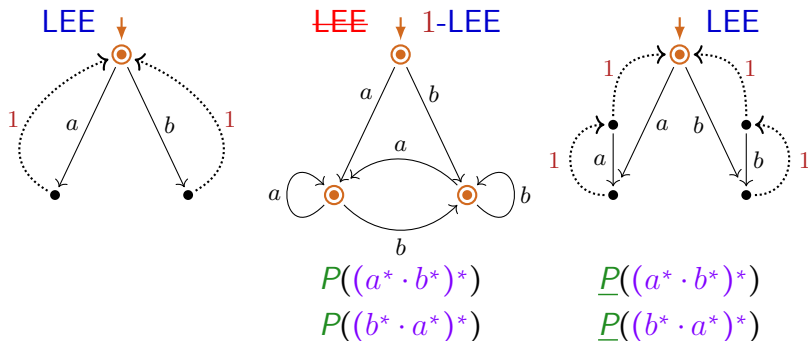
# 1-LEE holds for process interpretations

## Lemma

There is a **1-graph interpretation**  $\underline{P}$  of reg. expression  $e$  as 1-graphs  $\underline{P}(e)$  such that for all  $e \in RExp$ : (i):  $LEE(\underline{P}(e))$ , (ii):  $(\underline{P}(e)) = \underline{P}(e)$ .

## Theorem

**1-LEE**( $\underline{P}(e)$ ) holds for all regular expressions  $e$ .



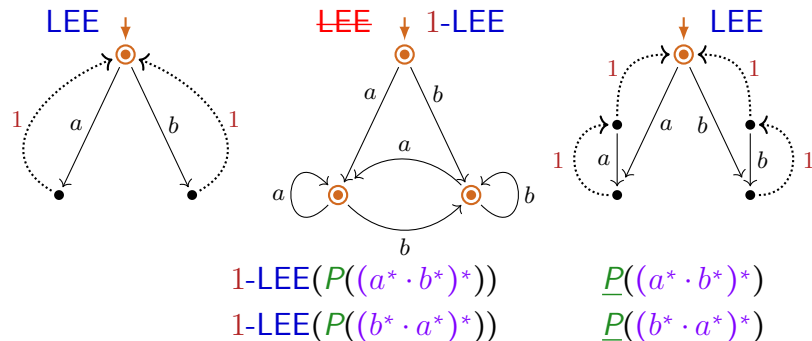
# 1-LEE holds for process interpretations

## Lemma

There is a **1-graph interpretation**  $\underline{P}$  of reg. expression  $e$  as 1-graphs  $\underline{P}(e)$  such that for all  $e \in RExp$ : (i):  $LEE(\underline{P}(e))$ , (ii):  $(\underline{P}(e)) = \underline{P}(e)$ .

## Theorem

**1-LEE**( $\underline{P}(e)$ ) holds for all regular expressions  $e$ .



# Interpretation/extraction correspondences with 1-LEE

( $\Leftarrow$  G 2021/22/23)

**(Int)<sub>P</sub>**: *P-expressible* graphs have the *structural property* 1-LEE

Process **interpretations**  $P(e)$  of regular expressions  $e$  are finite process graphs that satisfy 1-LEE.

**(Extr)<sub>P</sub>**: 1-LEE implies  $\llbracket \cdot \rrbracket_P$ -*expressibility*

From every finite 1-process-graph  $\underline{G}$  with 1-LEE a regular expression  $e$  can be **extracted** such that  $\underline{G} \Leftrightarrow P(e)$ .

# Interpretation/extraction correspondences with 1-LEE

( $\Leftarrow$  G 2021/22/23)

**(Int)<sub>P</sub>**: *P-expressible* graphs have the *structural property* 1-LEE

Process **interpretations**  $P(e)$  of regular expressions  $e$  are finite process graphs that satisfy 1-LEE.

**(Extr)<sub>P</sub>**: 1-LEE implies  $\llbracket \cdot \rrbracket_P$ -expressibility

From every finite 1-process-graph  $\underline{G}$  with 1-LEE a regular expression  $e$  can be **extracted** such that  $\underline{G} \Leftrightarrow P(e)$ .

**(Coll)**: 1-LEE is *not preserved under collapse*

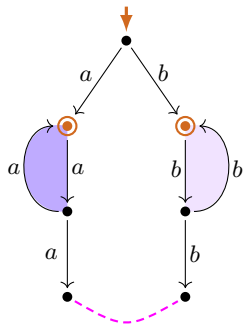
The class of finite process graphs with 1-LEE is **not closed under bisimulation collapse**.

1-LEE/ LEE characterize

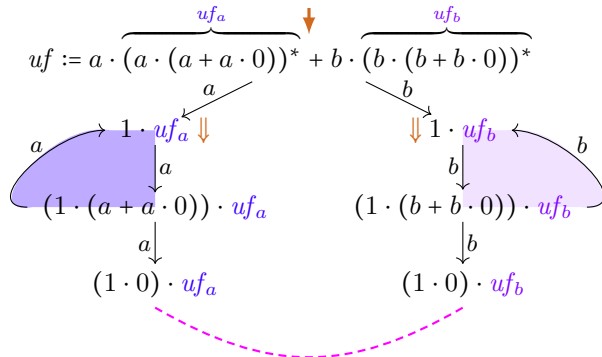
the un-/restricted image of compact version  $P^\bullet$  of  $P$

Image of  $P$  is **not closed** under bisimulation collapse  
**not even for**  $(*/\perp)$  regular expressions (example)

$P(uf)$



$P(uf)$



# Compact process interpretation $P^\bullet$

Definition (Transition system specification  $\mathcal{T}^\bullet$ , changed rules w.r.t.  $\mathcal{T}$ )

$$\begin{array}{c} \frac{e_1 \xrightarrow{a} e'_1}{e_1 \cdot e_2 \xrightarrow{a} e'_1 \cdot e_2} \text{ (if } e'_1 \text{ is normed)} \qquad \frac{e_1 \xrightarrow{a} e'_1}{e_1 \cdot e_2 \xrightarrow{a} e'_1} \text{ (if } e'_1 \text{ is not normed)} \\[2ex] \frac{e \xrightarrow{a} e'}{e^* \xrightarrow{a} e' \cdot e^*} \text{ (if } e' \text{ is normed)} \qquad \frac{e \xrightarrow{a} e'}{e^* \xrightarrow{a} e'} \text{ (if } e' \text{ is not normed)} \end{array}$$

Definition

The compact process (graph) interpretation  $P^\bullet(e)$  of a reg. expr's  $e$ :

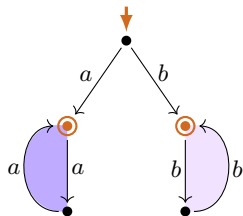
$P^\bullet(e) :=$  labeled transition graph generated by  $e$  by derivations in  $\mathcal{T}^\bullet$ .

Lemma ( $P^\bullet$  increases sharing;  $P^\bullet, P$  have same bisimulation semantics)

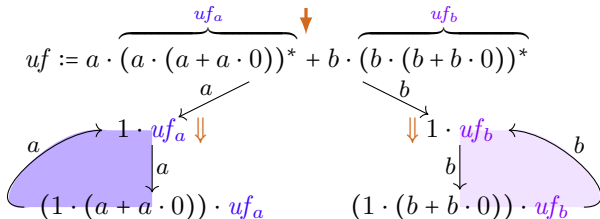
- (i)  $P(e) \rightleftharpoons P^\bullet(e)$  for all regular expressions  $e$ .
- (ii) ( $G$  is  $\llbracket \cdot \rrbracket_{P^\bullet}$ -expressible  $\iff G$  is  $\llbracket \cdot \rrbracket_P$ -expressible) for all graphs  $G$ .

# Image of $P$ restricted to $(*/1)$ regular expressions ... contains all of its bisimulation collapses (example)

$P^\bullet(uf)$



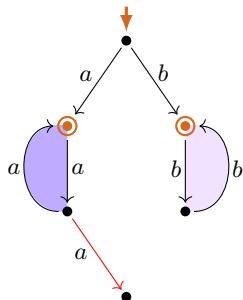
$P^\bullet(uf)$



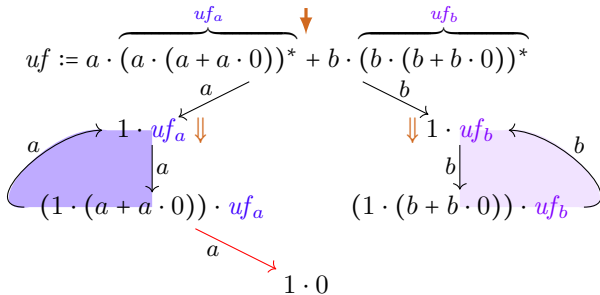


# Image of $P$ restricted to $(*/1)$ regular expressions ... contains all of its bisimulation collapses (example)

$P^\bullet(uf)$

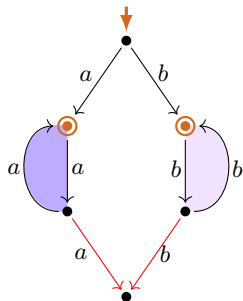


$P^\bullet(uf)$

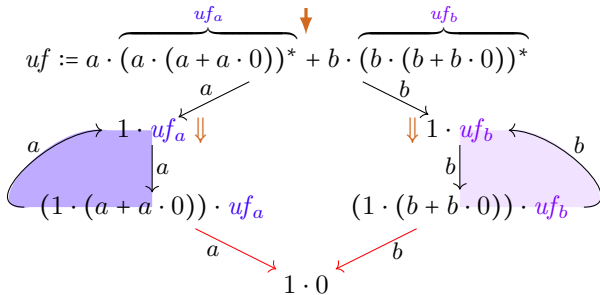


# Image of $P$ restricted to $(*/1)$ regular expressions ... contains all of its bisimulation collapses (example)

$P^\bullet(uf)$



$P^\bullet(uf)$



# Interpretation correspondence of $P^\bullet$ with LEE

**(Int)** $_{P^\bullet}^{(*/+)}:$  By *under-star-1-free* expressions  $P^\bullet$ -expressible graphs satisfy LEE:

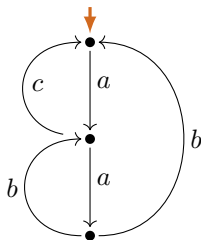
Compact process interpretations  $P^\bullet(uf)$   
 of *under-star-1-free* regular expressions  $uf$   
 are finite process graphs that satisfy LEE.

**(Extr)** $_{P^\bullet}^{(*/+)}:$  LEE implies  $\llbracket \cdot \rrbracket_{P^\bullet}$ -expressibility by *under-star-1-free* reg. expr's:

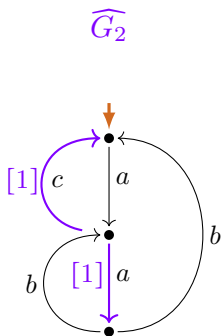
From every finite process graph  $G$  with LEE  
 an *under-star-1-free* regular expression  $uf$  can be extracted  
 such that  $G \rightrightarrows P^\bullet(uf)$ .

# Refined extraction expression (example)

$G_2$

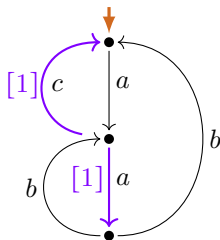


# Refined extraction expression (example)



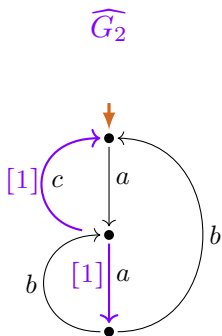
# Refined extraction expression (example)

$\widehat{G}_2$



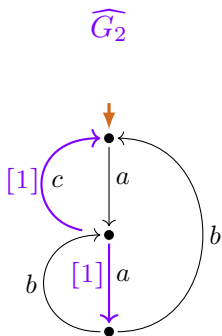
$$(1 \cdot ( \quad )^*) \cdot 0$$

# Refined extraction expression (example)



$$(1 \cdot ( \begin{array}{c} \vdots \\ a \\ \vee \end{array} )^* ) \cdot 0$$

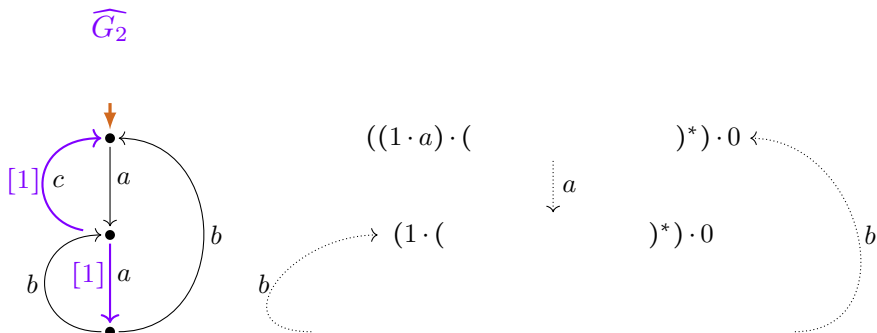
# Refined extraction expression (example)



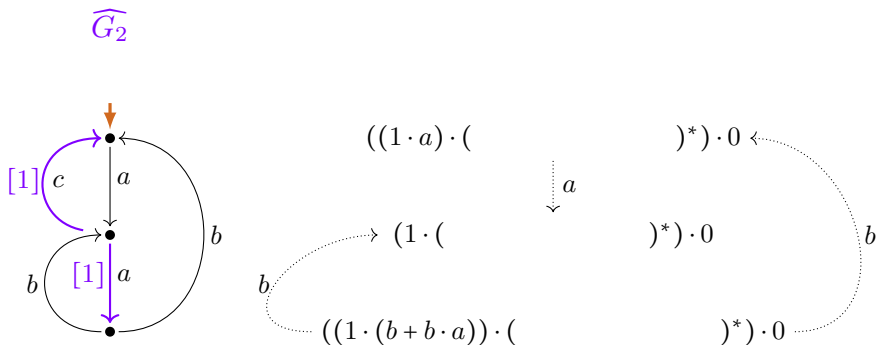
$$\begin{array}{ccc}
 ((1 \cdot a) \cdot ( & & )^*) \cdot 0 \\
 \downarrow a & & \\
 (1 \cdot ( & & )^*) \cdot 0
 \end{array}$$



# Refined extraction expression (example)

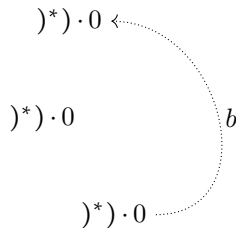
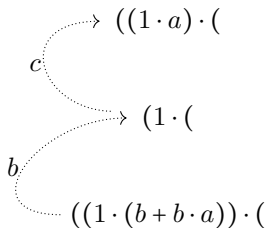
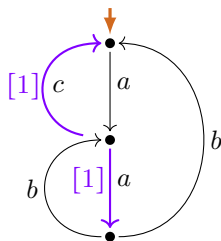


# Refined extraction expression (example)

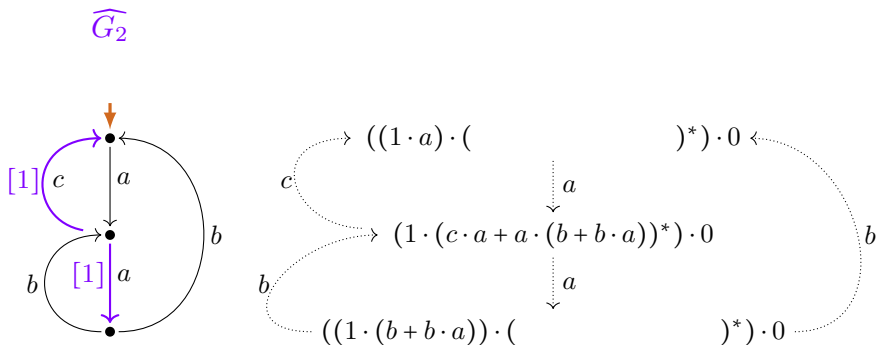


# Refined extraction expression (example)

$\widehat{G}_2$

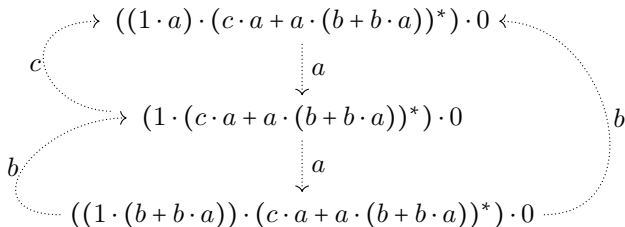
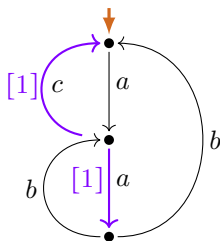


# Refined extraction expression (example)



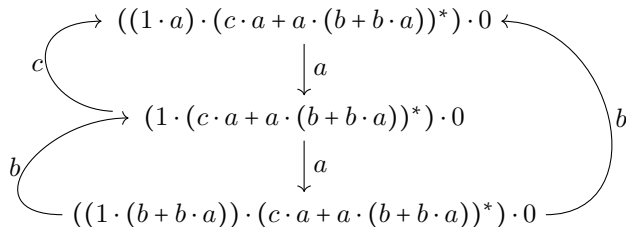
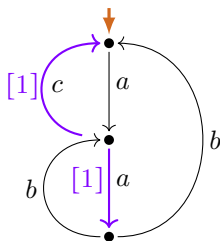
# Refined extraction expression (example)

$\widehat{G}_2$



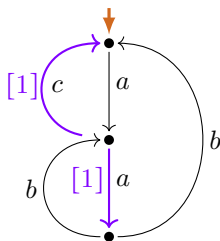
# Refined extraction expression (example)

$\widehat{G}_2$

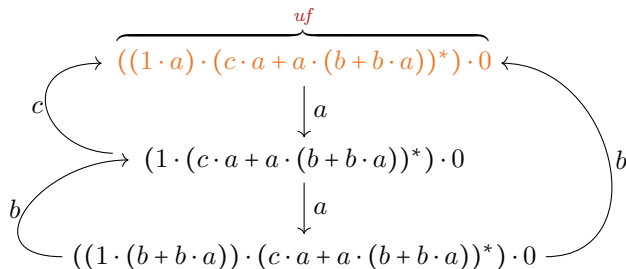


# Refined extraction expression (example)

$\widehat{G}_2$



$$P^\bullet(uf) = P(uf) \simeq G_2$$



# Interpretation/extraction correspondences of $P^\bullet$ with LEE

**(Int)** $_{P^\bullet}^{(*/\pm)}$ : By *under-star-1-free* expressions  $P^\bullet$ -expressible graphs satisfy LEE:

Compact process interpretations  $P^\bullet(uf)$   
 of *under-star-1-free* regular expressions  $uf$   
 are finite process graphs that satisfy LEE.

**(Extr)** $_{P^\bullet}^{(*/\pm)}$ : LEE implies  $\llbracket \cdot \rrbracket_{P^\bullet}$ -expressibility by *under-star-1-free* reg. expr's:

From every finite process graph  $G$  with LEE  
 an *under-star-1-free* regular expression  $uf$  can be extracted  
 such that  $G \rightrightarrows P^\bullet(uf)$ .

From every finite collapsed process graph  $G$  with LEE  
 an *under-star-1-free* regular expression  $uf$  can be extracted  
 such that  $G \simeq P^\bullet(uf)$ .



# Interpretation/extraction correspondences of $P^\bullet$ with LEE

**(Int)** $_{P^\bullet}^{(*/\pm)}$ : By *under-star-1-free* expressions  $P^\bullet$ -expressible graphs satisfy LEE:

Compact process interpretations  $P^\bullet(uf)$   
 of *under-star-1-free* regular expressions  $uf$   
 are finite process graphs that satisfy LEE.

**(Extr)** $_{P^\bullet}^{(*/\pm)}$ : LEE implies  $\llbracket \cdot \rrbracket_{P^\bullet}$ -expressibility by *under-star-1-free* reg. expr's:

From every finite process graph  $G$  with LEE  
 an *under-star-1-free* regular expression  $uf$  can be extracted  
 such that  $G \rightrightarrows P^\bullet(uf)$ .

From every finite collapsed process graph  $G$  with LEE  
 an *under-star-1-free* regular expression  $uf$  can be extracted  
 such that  $G \simeq P^\bullet(uf)$ .

**(ImColl)** $_{P^\bullet}^{(*/\pm)}$ : The image of  $P^\bullet$ ,  
 restricted to *under-star-1-free* regular expressions,  
 is closed under bisimulation collapse.

# Interpretation/extraction correspondences of $P^\bullet$ with 1-LEE

**(Int)** $_{P^\bullet}$ :  $P^\bullet$ -expressible graphs satisfy 1-LEE:

Compact process interpretations  $P^\bullet(e)$  of regular expressions  $e$  are finite process graphs that satisfy 1-LEE.

**(Extr)** $_{P^\bullet}$ : 1-LEE implies  $\llbracket \cdot \rrbracket_{P^\bullet}$ -expressibility:

From every finite process graph  $G$  with 1-LEE  
an regular expression  $e$  can be extracted  
such that  $G \rhd P^\bullet(e)$ .

From every finite collapsed process graph  $G$  with 1-LEE  
a regular expression  $e$  can be extracted  
such that  $G \simeq P^\bullet(e)$ .

# Interpretation/extraction correspondences of $P^\bullet$ with 1-LEE

**(Int) $_{P^\bullet}$ :**  $P^\bullet$ -expressible graphs satisfy 1-LEE:

Compact process interpretations  $P^\bullet(e)$  of regular expressions  $e$  are finite process graphs that satisfy 1-LEE.

**(Extr) $_{P^\bullet}$ :** LEE implies  $\llbracket \cdot \rrbracket_{P^\bullet}$ -expressibility:

From every finite process graph  $G$  with 1-LEE  
an regular expression  $e$  can be extracted  
such that  $G \rhd P^\bullet(e)$ .

From every finite collapsed process graph  $G$  with 1-LEE  
a regular expression  $e$  can be extracted  
such that  $G \simeq P^\bullet(e)$ .

**(ImColl) $_{P^\bullet}$ :** The image of  $P^\bullet$  is not closed under bisimulation collapse.

$$\text{LEE} \stackrel{\wedge}{=} \text{image of } P^\bullet \big|_{\text{RExp}^{(*/\perp)}}$$

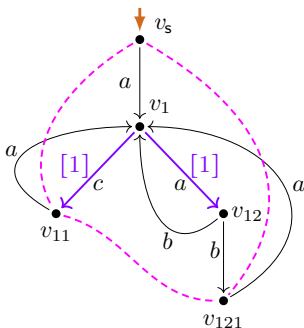
## Theorem

For every process graph  $G$  TFAE:

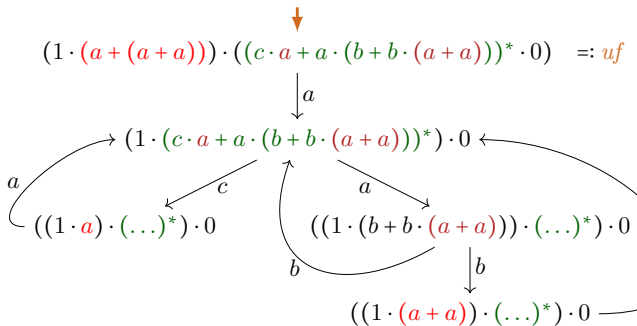
- (i)  $\text{LEE}(G)$ .
- (ii)  $G$  is  $P^\bullet$ -expressible by an  $(*/\perp)$  regular expression  
(i.e.  $G \simeq P^\bullet(e)$  for some  $e \in \text{RExp}^{(*/\perp)}$ ).
- (iii)  $G$  is isomorphic to a graph in the image of  $P^\bullet$  on  $(*/\perp)$  reg. expr's  
(i.e.  $G \simeq G'$  for some  $G' \in \text{im}(P^\bullet \big|_{\text{RExp}^{(*/\perp)})}$ ).

# Adapted (refined) extraction from LLEE-graph

$$G_1 / \widehat{G}_1$$



$$P^\bullet(uf) = P(uf) \simeq G_1$$



# 1-LEE $\stackrel{\wedge}{=}$ image of $P^\bullet$

## Theorem

For every process graph  $G$  TFAE:

- (i) 1-LEE( $G$ )  
(i.e.  $G = (\underline{G}]$  for some 1-transition-process-graph  $\underline{G}$  with LEE( $\underline{G}$ )).
- (ii)  $G$  is  $P^\bullet$ -expressible by a regular expression  
(i.e.  $G \simeq P^\bullet(e)$  for some  $e \in RExp$ ).
- (iii)  $G$  is isomorphic to a graph in the image of  $P^\bullet$   
(i.e.  $G \simeq G'$  for some  $G' \in im(P^\bullet)$ ).

# Summary

- ▶ Characterizations of the image of  $P^\bullet$  (refinement of  $P$ ):
  - ▶ LEE  $\hat{=}$  image of  $P^\bullet|_{\text{RExp}(*/\textcolor{red}{+})} \not\equiv$  image of  $P|_{\text{RExp}(*/\textcolor{red}{+})}$
  - ▶ 1-LEE  $\hat{=}$  image of  $P^\bullet \not\equiv$  image of  $P$

# Summary

- ▶ process interpretation  $P$ /semantics  $\llbracket \cdot \rrbracket_P$  of regular expressions
  - ▶ expressibility and completeness questions
- ▶ loop existence and elimination (LEE)
  - ▶ loop elimination rewrite system can be completed
  - ▶ interpretation/extraction correspondences with  $(*/\pm)$  reg. expr.s
  - ▶ LEE-witnesses: labelings of graphs with LEE
  - ▶ stepwise LEE-preserving bisimulation collapse
- ▶ 1-LEE = sharing via 1-transitions facilitates LEE
  - ▶ interpretation/extraction correspondences with all regular expressions
  - ▶ not preserved under bisim. collapse (approximation possible)
- ▶ Characterizations of the image of  $P^\bullet$  (refinement of  $P$ ):
  - ▶  $\text{LEE} \triangleq \text{image of } P^\bullet|_{\text{RExp}(*\pm)} \not\sqsubseteq \text{image of } P|_{\text{RExp}(*\pm)}$
  - ▶  $1\text{-LEE} \triangleq \text{image of } P^\bullet \not\sqsubseteq \text{image of } P$
- ▶ outlook on work-to-do



# My next aims

Completeness problem, [solution](#) ([journal articles](#)):

**A1:** graph structure of regular expression processes ([LEE](#)/[1-LEE](#))

**A2:** motivation of crystallization

**A4:** details of crystallization procedure,  
and completeness of Milner's proof system

Expressibility problem

**A3:** [LEE](#) is decidable in polynomial time ([conference article](#)).

**Q:** Is [1-LEE](#) decidable in polynomial time?

**P:** Is expressibility by a regular expression, for a finite process graph,  
decidable in polynomial time/fixed-parameter tractable time?

# Resources

- ▶ Slides/abstract on [clegra.github.io](https://clegra.github.io)
  - ▶ slides: [.../1f/IFIP-1\\_6-2024.pdf](https://clegra.github.io/1f/IFIP-1_6-2024.pdf)
  - ▶ abstract: [.../1f/abstract-IFIP-1\\_6-2024.pdf](https://clegra.github.io/1f/abstract-IFIP-1_6-2024.pdf)
- ▶ CG: Closing the Image of the Process Interpretation of 1-Free Regular Expressions Under Bisimulation Collapse
  - ▶ TERMGRAPH 2024, [extended abstract](#).
- ▶ CG: The Image of the Process Interpretation of Regular Expressions is Not Closed under Bisimulation Collapse,
  - ▶ [arXiv:2303.08553](#), 2021/2023.
- ▶ CG: Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete,
  - ▶ LICS 2022, [arXiv:2209.12188](#), [poster](#).
- ▶ CG, Wan Fokkink: A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity,
  - ▶ LICS 2020, [arXiv:2004.12740](#), [video on youtube](#).
- ▶ CG: Modeling Terms by Graphs with Structure Constraints,
  - ▶ TERMGRAPH 2018, [EPTCS 288](#), [arXiv:1902.02010](#).

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

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

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

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

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

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

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

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

$e_1 + e_2 \xrightarrow{L} \text{union of } L(e_1) \text{ and } L(e_2)$

$e_1 \cdot e_2 \xrightarrow{L} \text{element-wise concatenation of } L(e_1) \text{ and } L(e_2)$

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

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

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

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

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

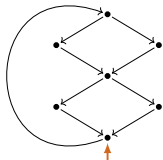
$e_1 + e_2 \xrightarrow{L} \text{union of } L(e_1) \text{ and } L(e_2)$

$e_1 \cdot e_2 \xrightarrow{L} \text{element-wise concatenation of } L(e_1) \text{ and } L(e_2)$

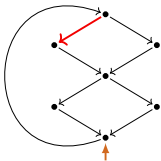
$e^* \xrightarrow{L} \text{set of words formed by concatenating words in } L(e),$   
and adding the empty word  $\epsilon$

$\llbracket e \rrbracket_L := L(e) \quad (\text{language defined by } e)$

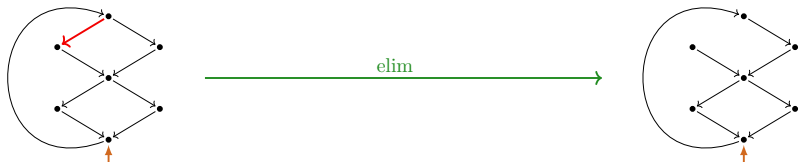
# 'Critical pair': bi-loop elimination



# 'Critical pair': bi-loop elimination

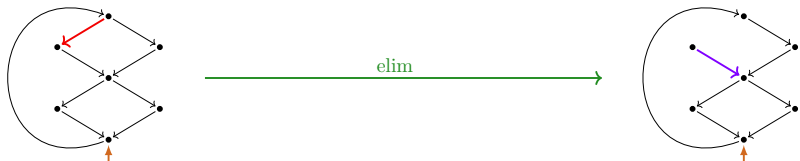


# 'Critical pair': bi-loop elimination

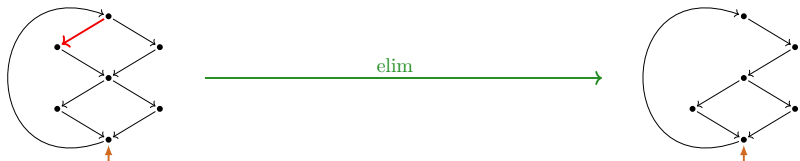




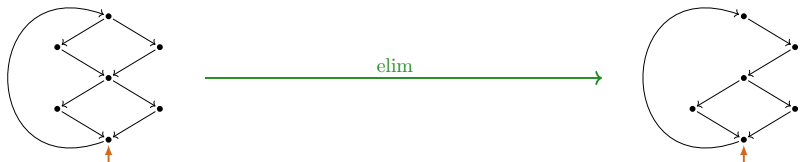
# 'Critical pair': bi-loop elimination



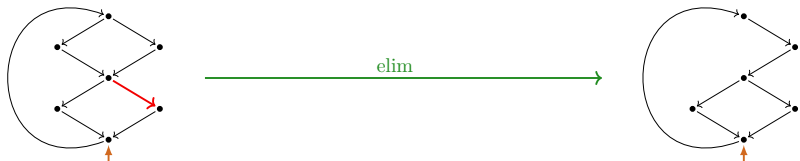
# 'Critical pair': bi-loop elimination



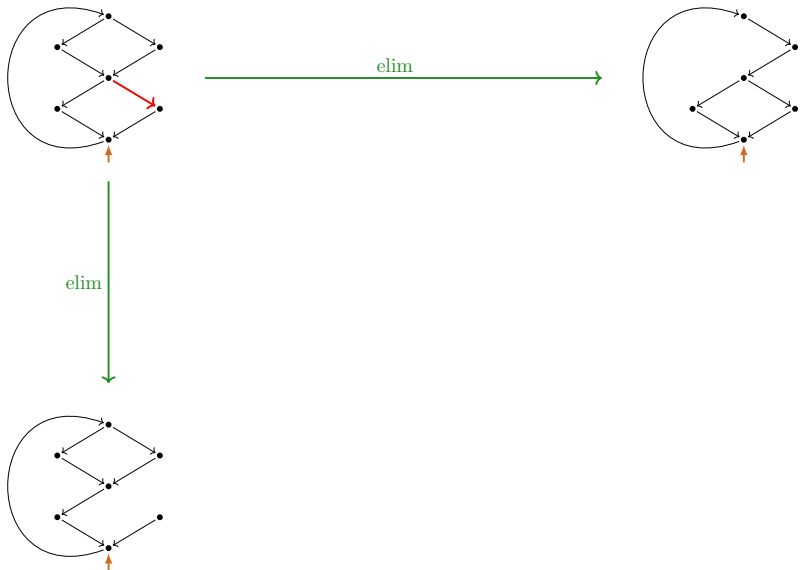
# 'Critical pair': bi-loop elimination



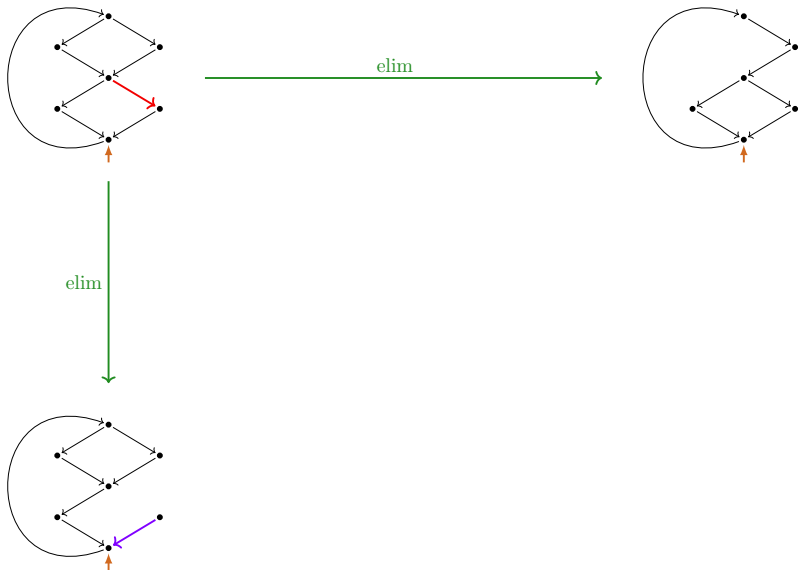
# 'Critical pair': bi-loop elimination



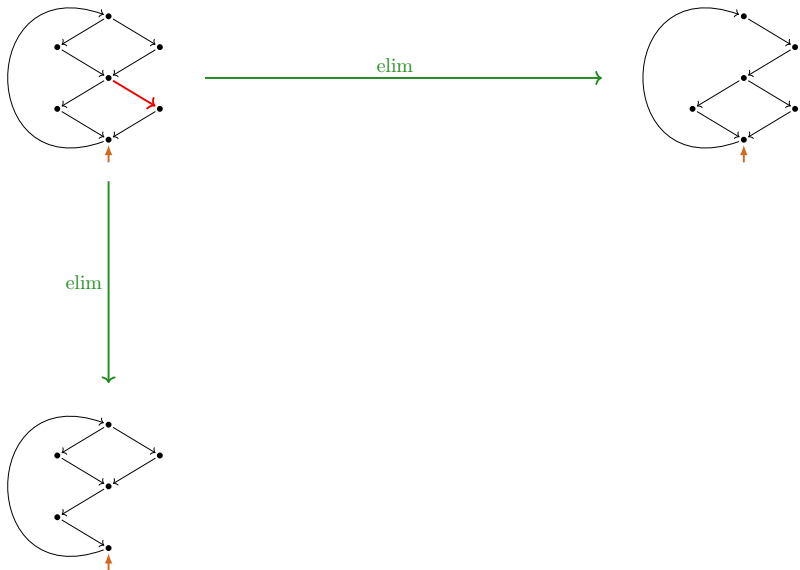
# 'Critical pair': bi-loop elimination



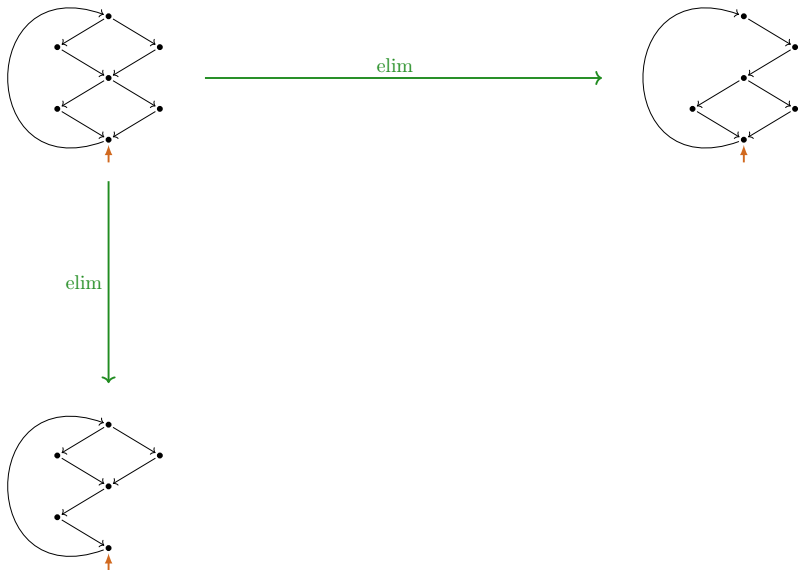
# 'Critical pair': bi-loop elimination



# 'Critical pair': bi-loop elimination

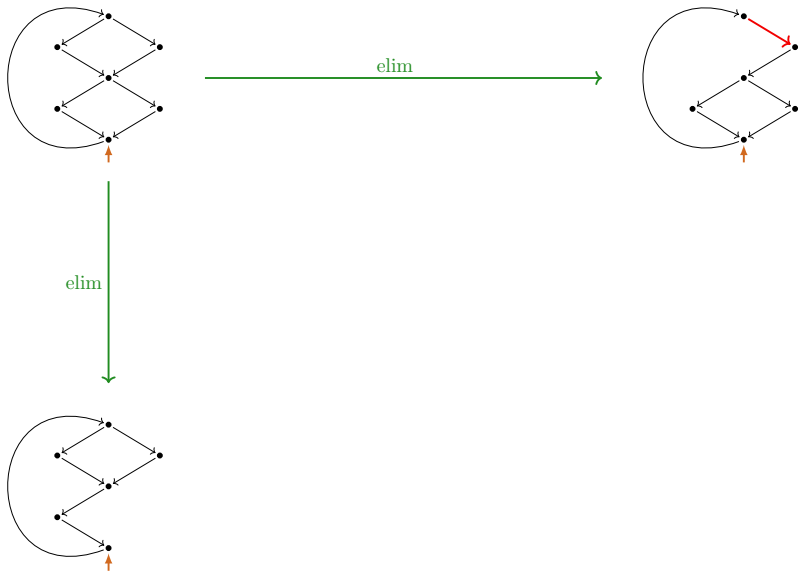


# 'Critical pair': bi-loop elimination

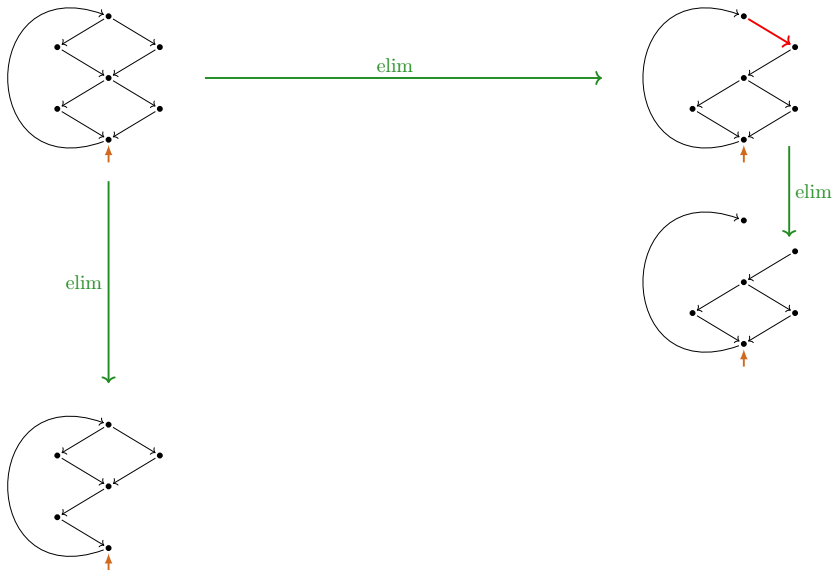




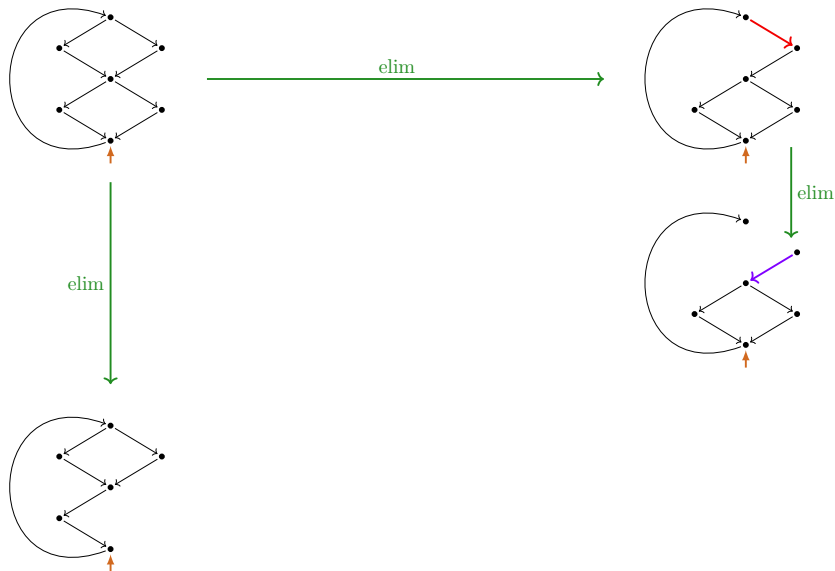
# 'Critical pair': bi-loop elimination



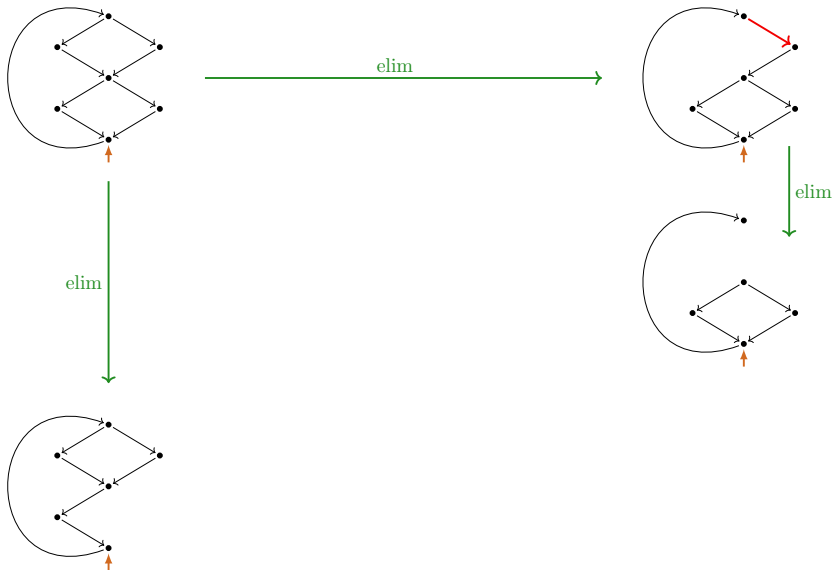
# 'Critical pair': bi-loop elimination



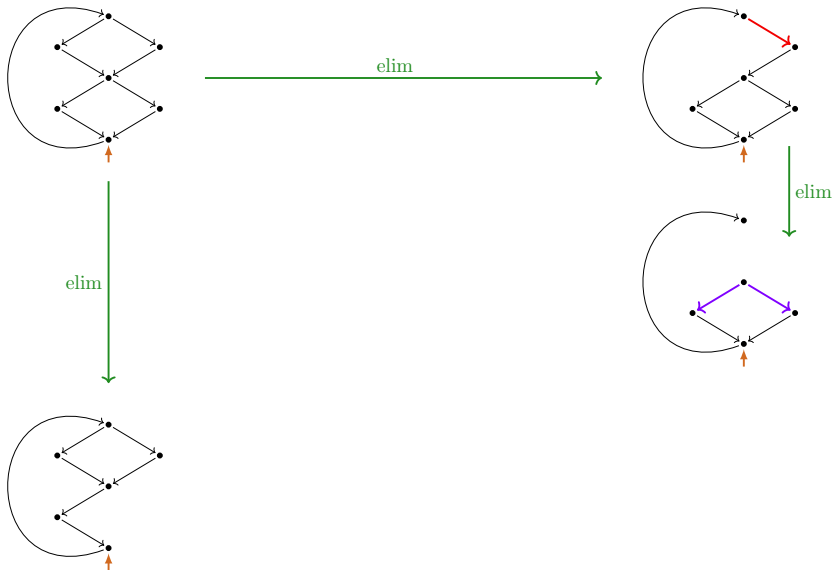
# 'Critical pair': bi-loop elimination



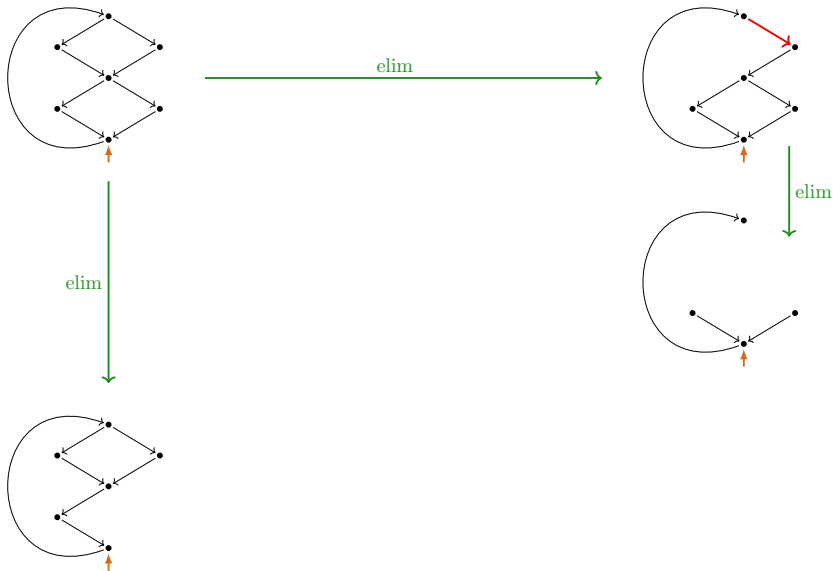
# 'Critical pair': bi-loop elimination



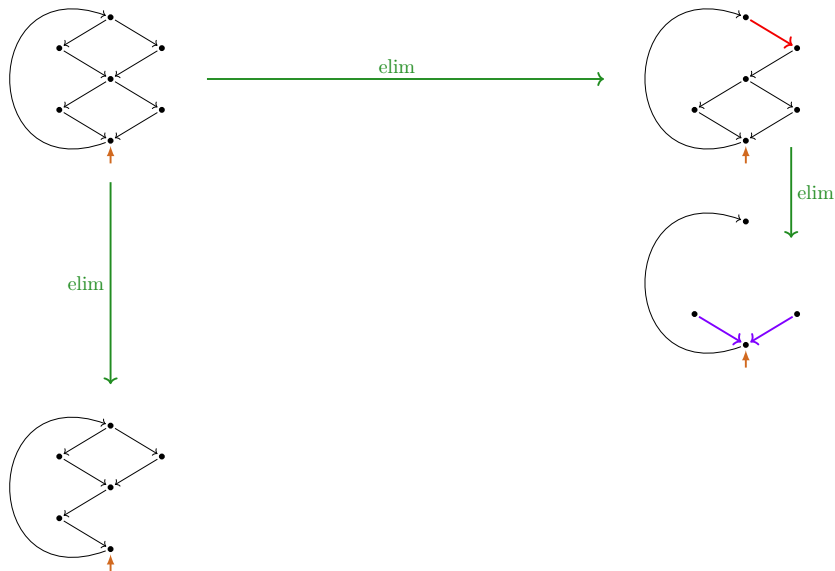
# 'Critical pair': bi-loop elimination



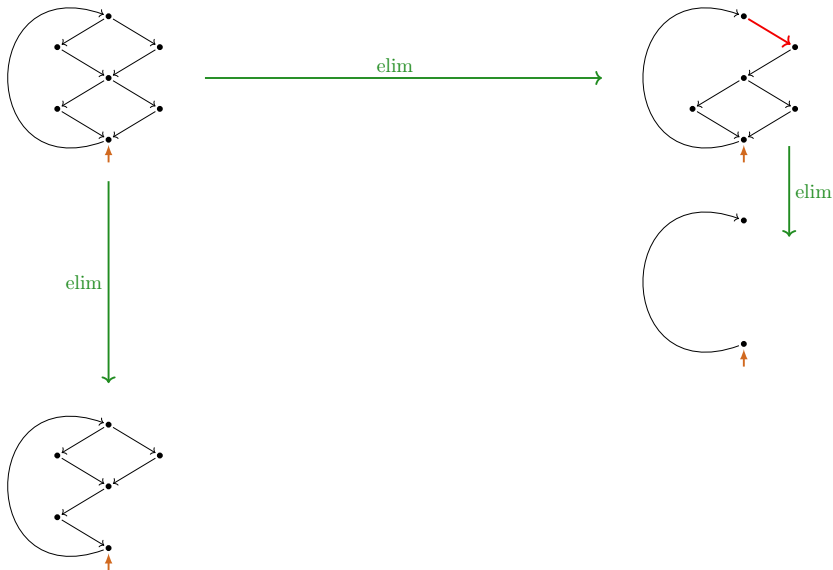
# 'Critical pair': bi-loop elimination



# 'Critical pair': bi-loop elimination

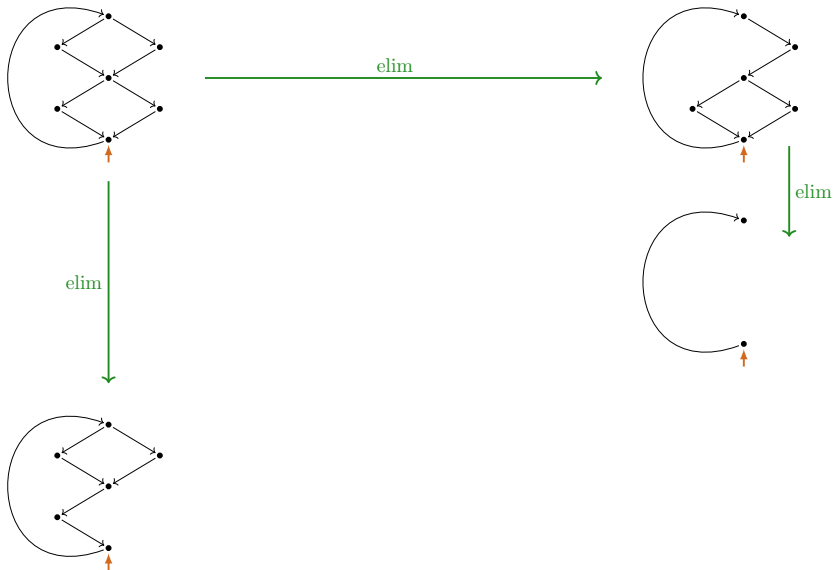


# 'Critical pair': bi-loop elimination

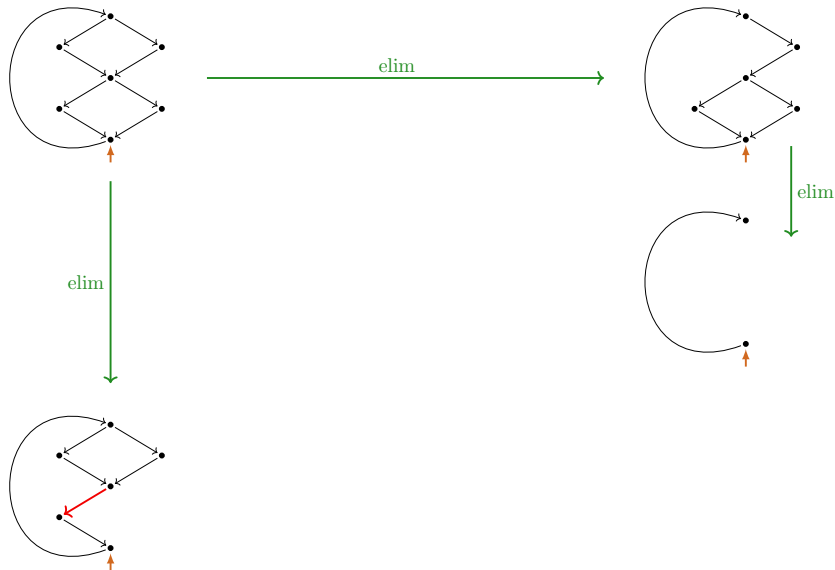




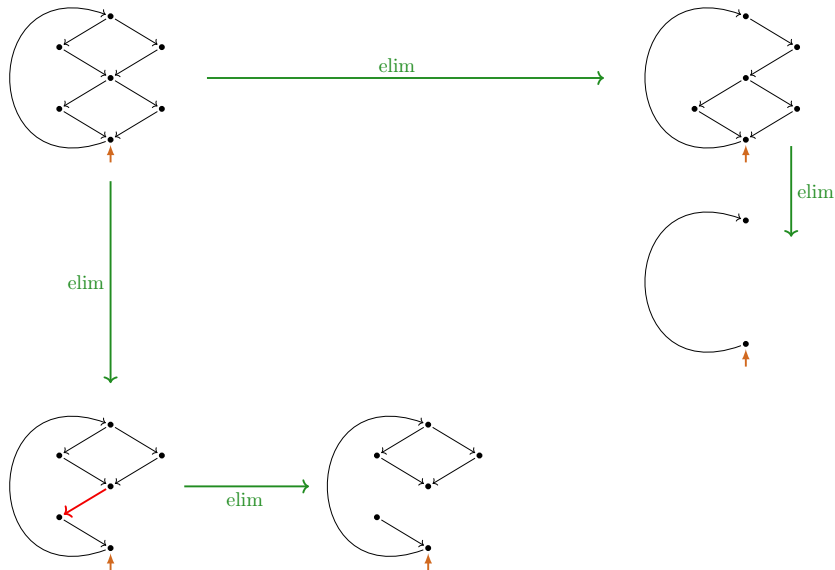
# 'Critical pair': bi-loop elimination



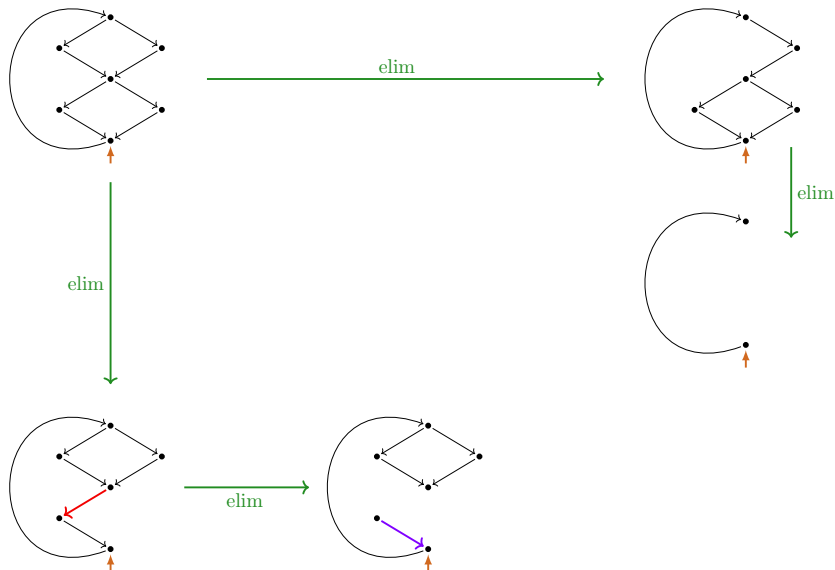
# 'Critical pair': bi-loop elimination



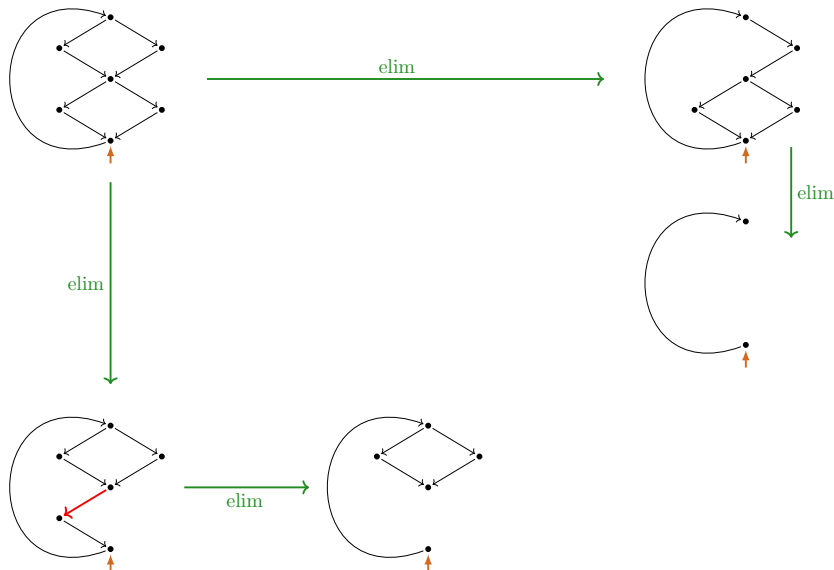
# 'Critical pair': bi-loop elimination



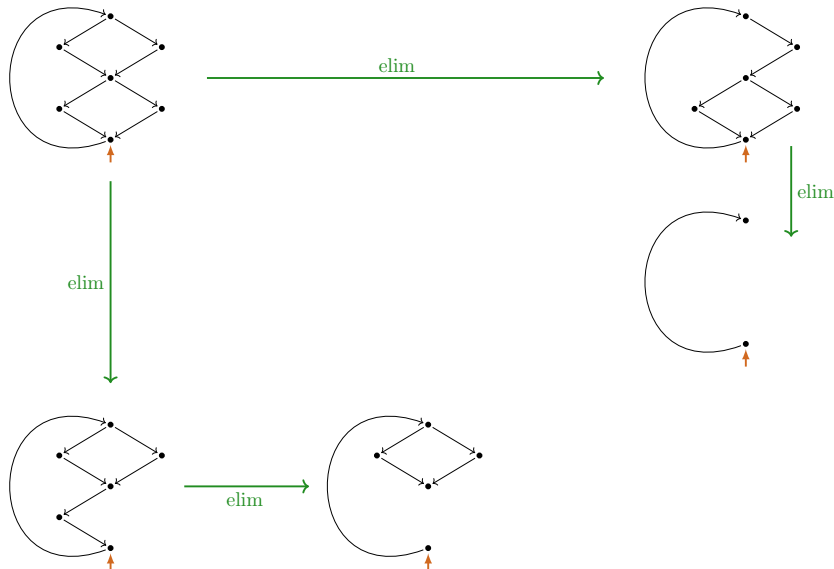
# 'Critical pair': bi-loop elimination



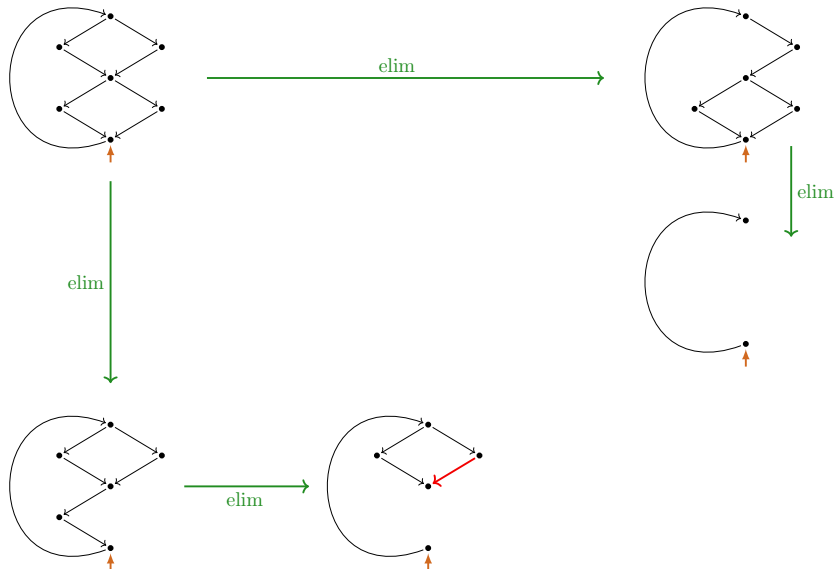
# 'Critical pair': bi-loop elimination



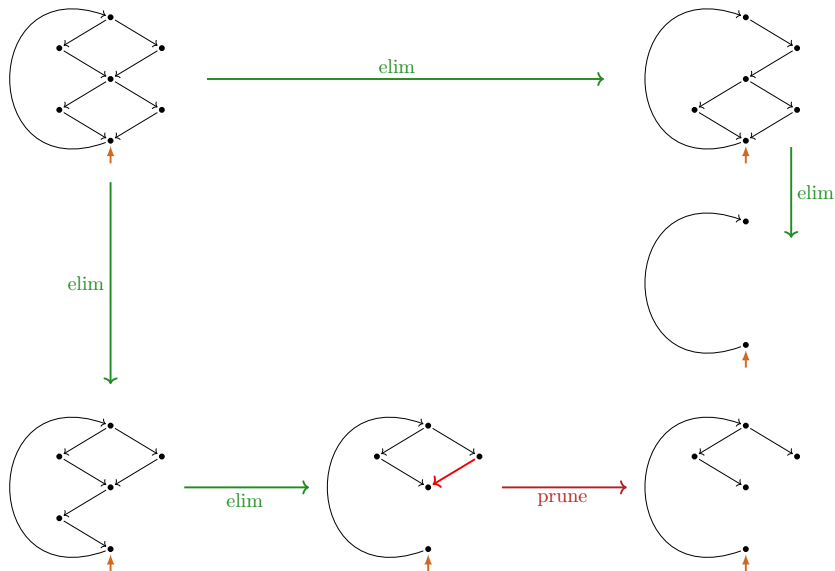
# 'Critical pair': bi-loop elimination



# 'Critical pair': bi-loop elimination

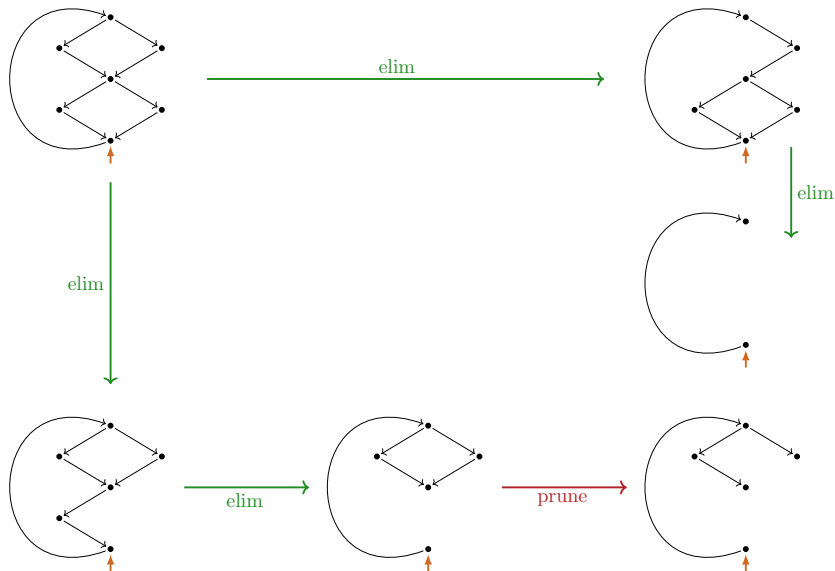


# 'Critical pair': bi-loop elimination

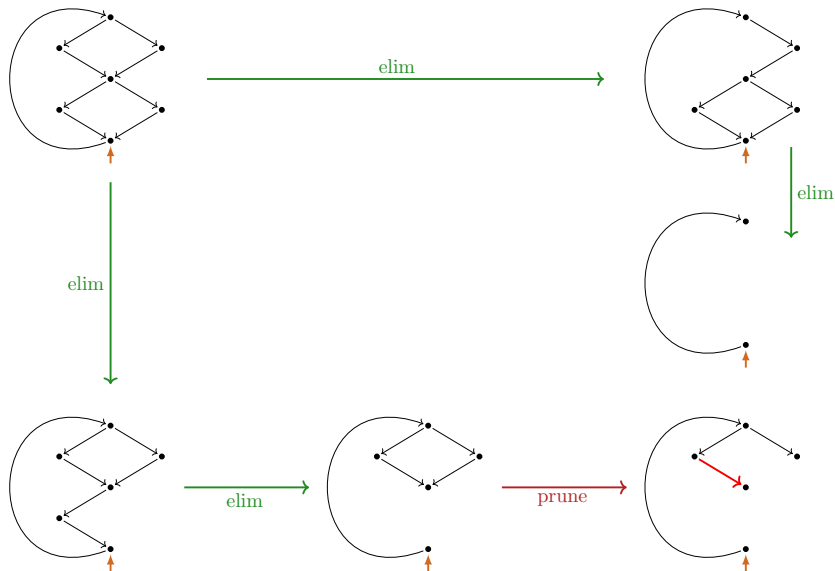




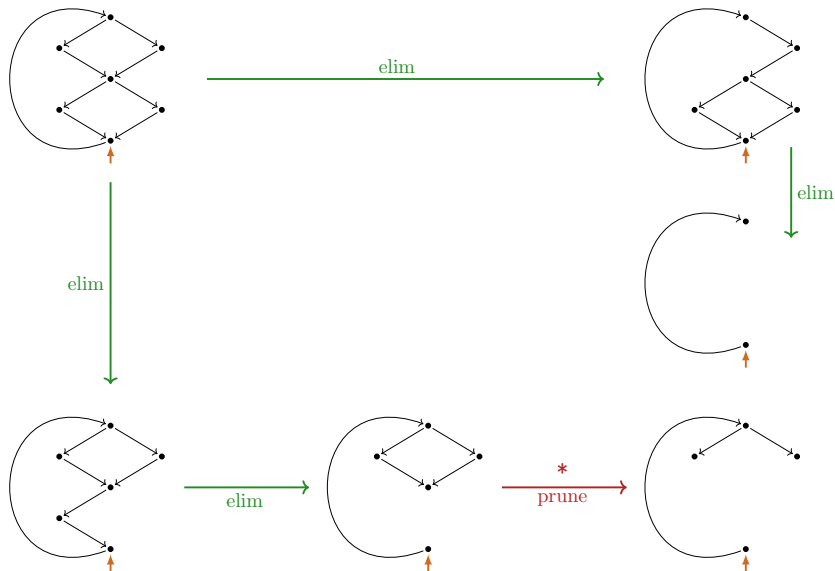
# 'Critical pair': bi-loop elimination



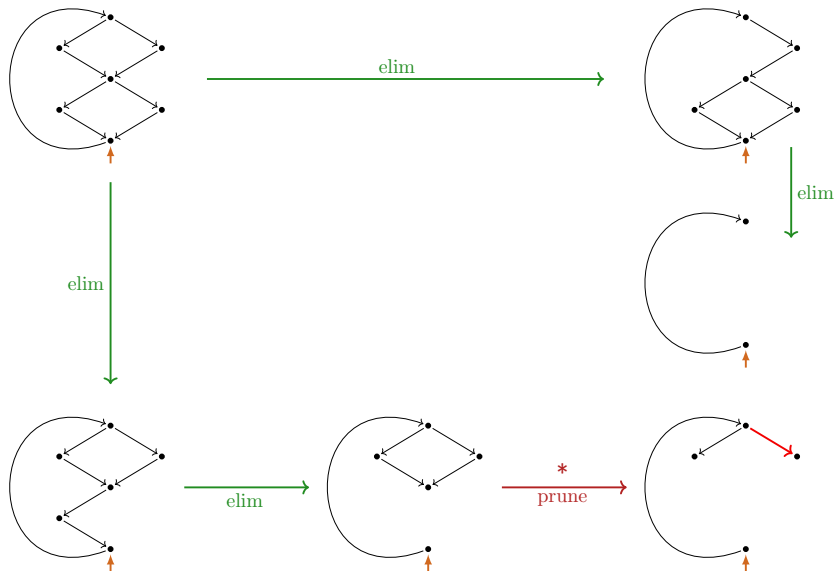
# 'Critical pair': bi-loop elimination



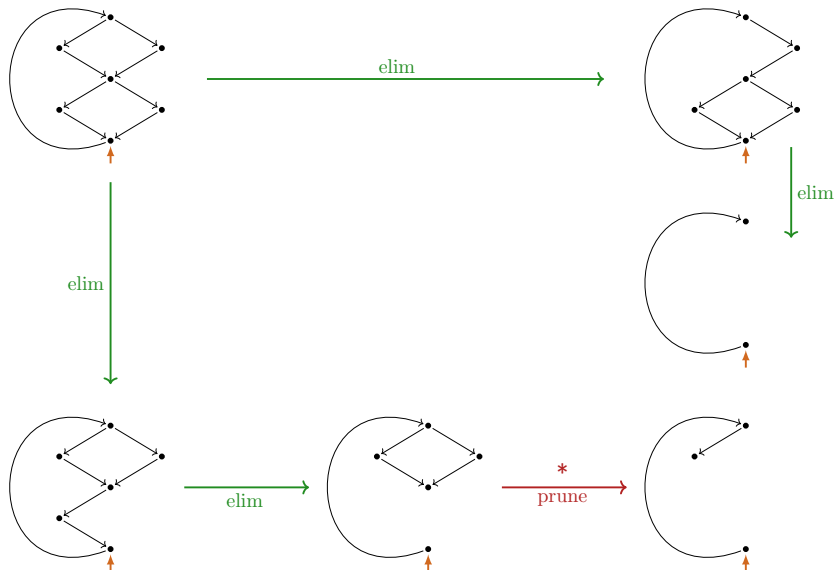
# 'Critical pair': bi-loop elimination



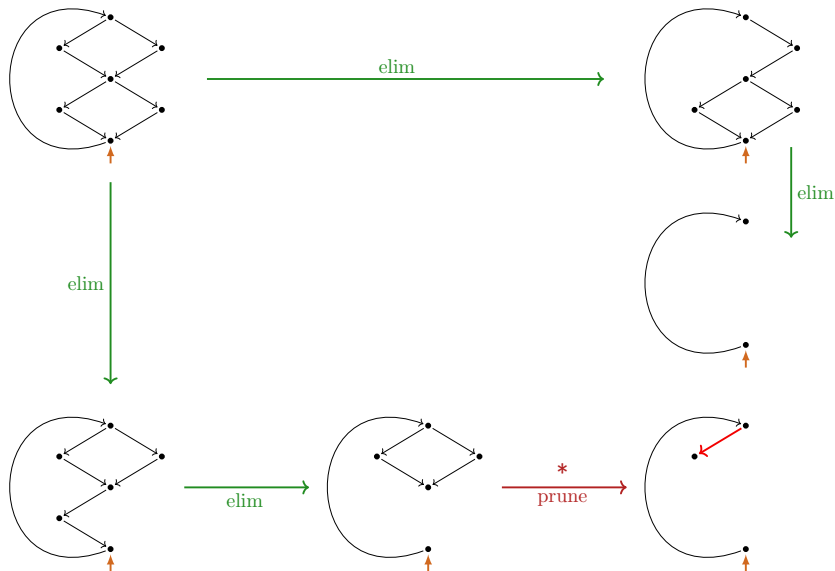
# 'Critical pair': bi-loop elimination



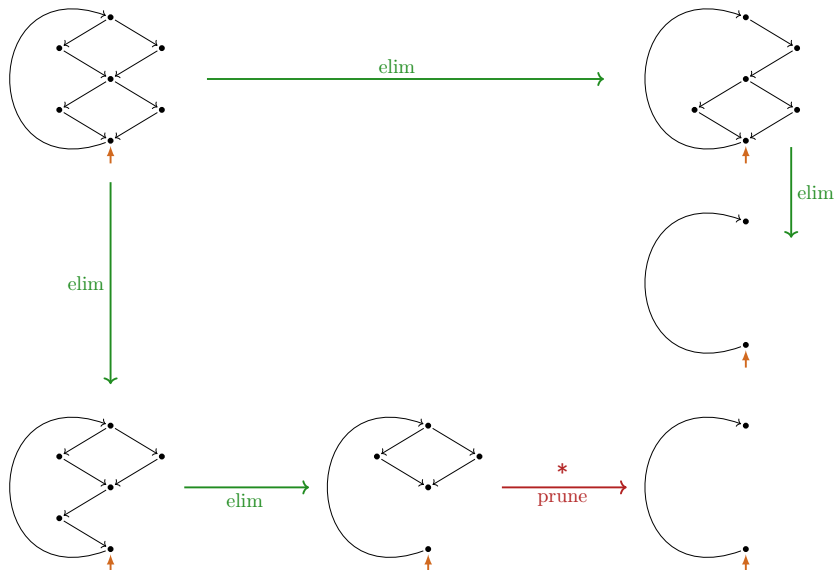
# 'Critical pair': bi-loop elimination



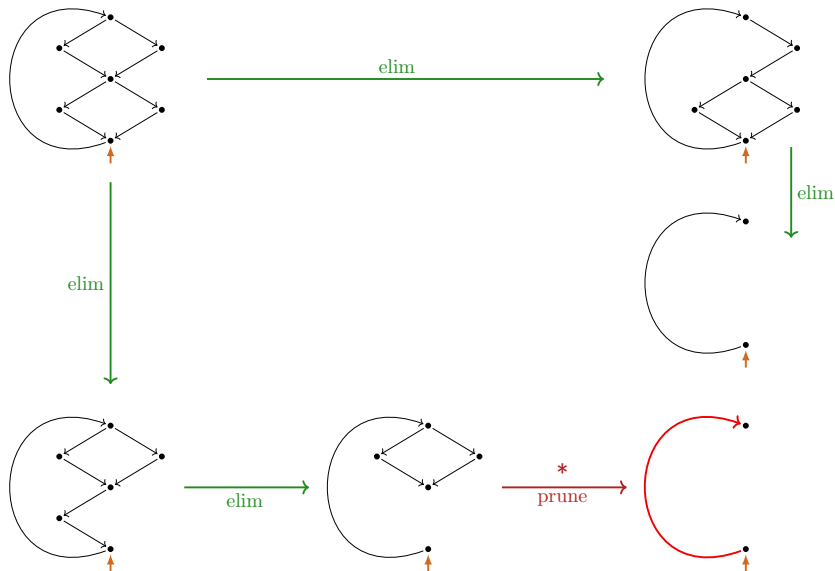
# 'Critical pair': bi-loop elimination



# 'Critical pair': bi-loop elimination

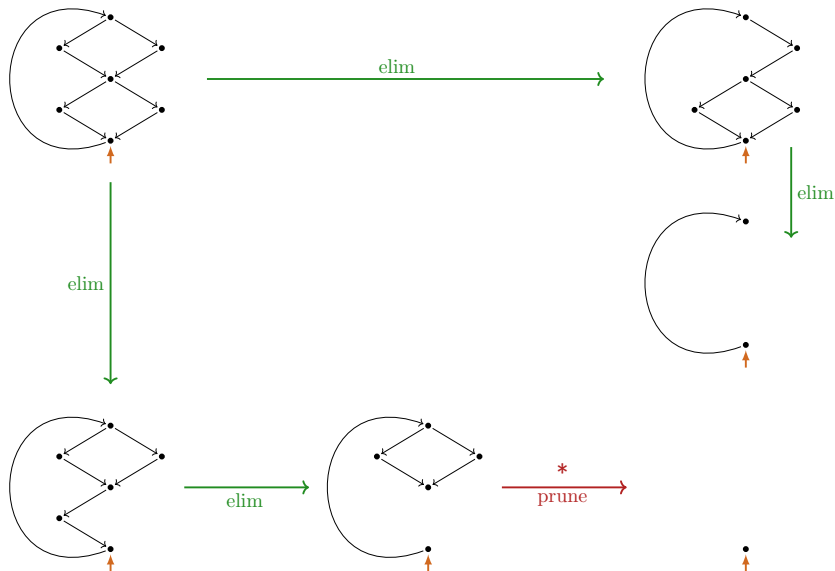


# 'Critical pair': bi-loop elimination

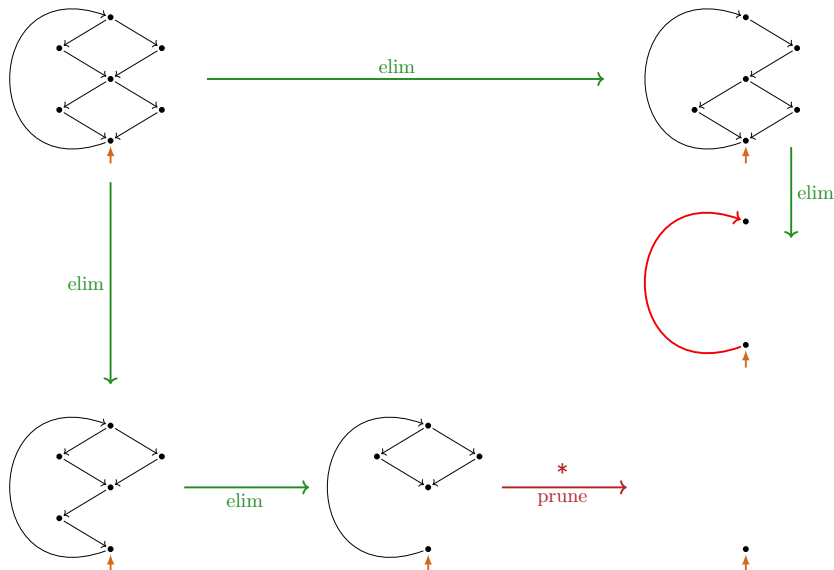




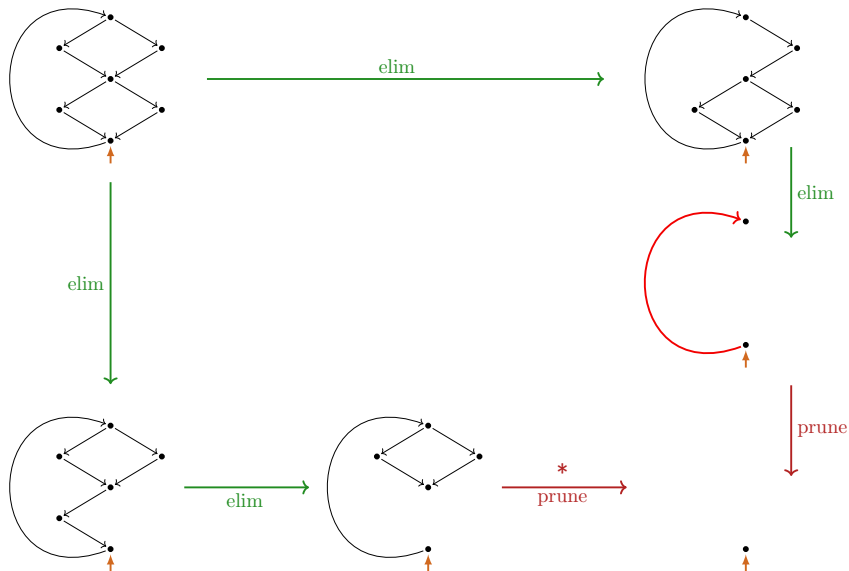
# 'Critical pair': bi-loop elimination



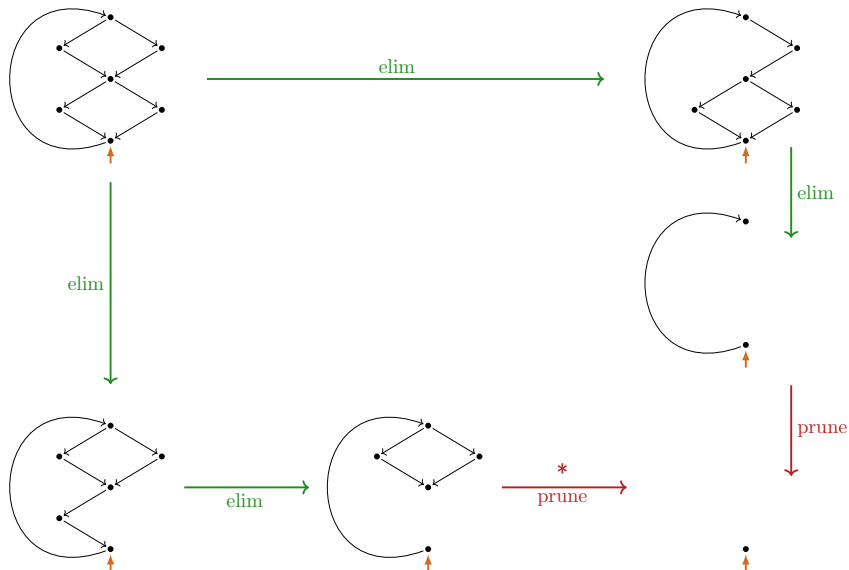
# 'Critical pair': bi-loop elimination



# 'Critical pair': bi-loop elimination



# 'Critical pair': bi-loop elimination



# 'Critical pair': bi-loop elimination

