Loop Elimination in Process Graphs is Confluent when Pruning Steps are Added

Clemens Grabmayer
https://clegra.github.io

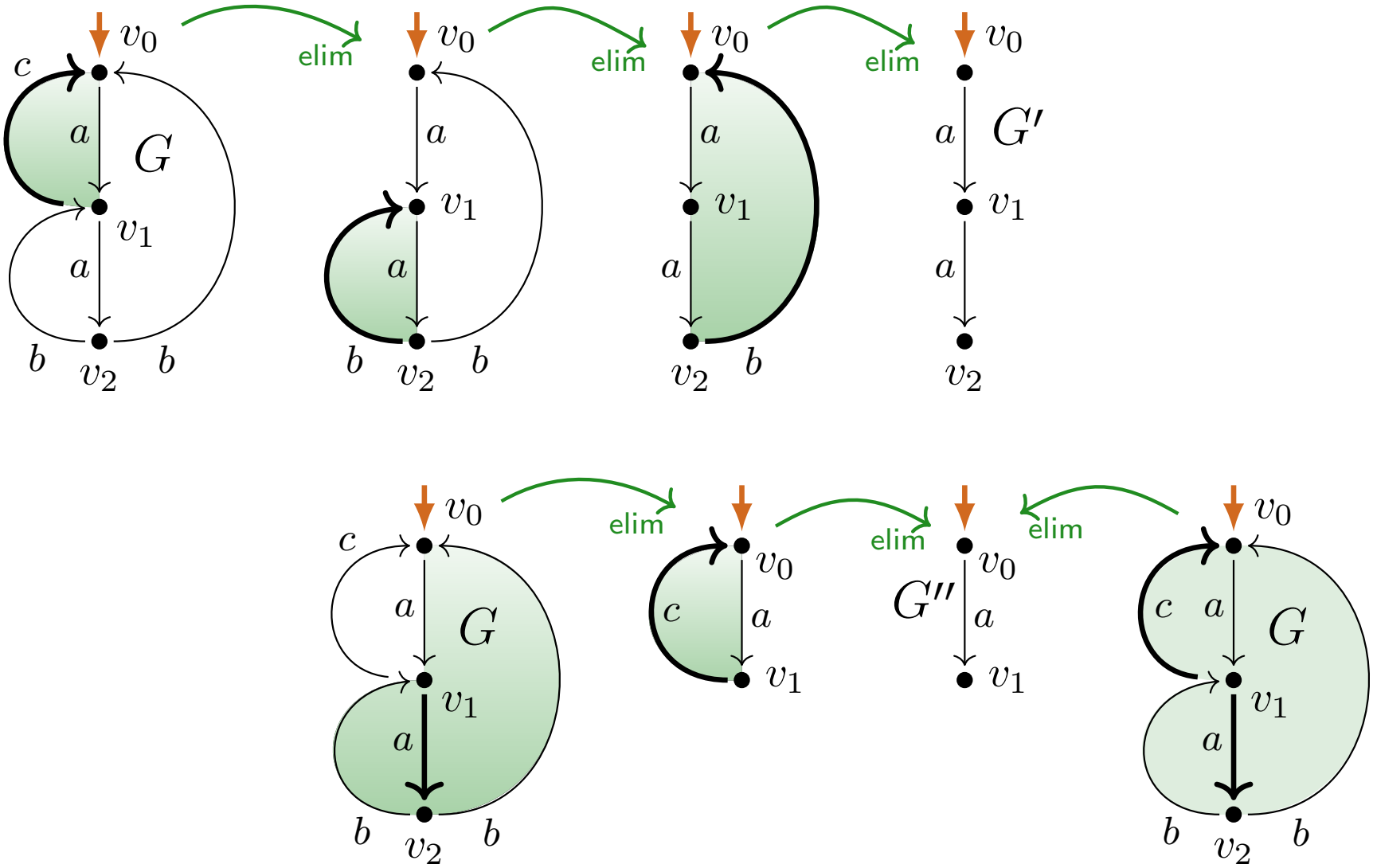
Department of Computer Science
GRAN SASSO SCIENCE INSTITUTE
SCHOOL OF ADVANCED STUDIES
Scuola Universitaria Superiore
L'Aquila, Italy

IWC 2026
Universidade de Lisboa
July 24, 2026

Overview

- Loop elimination $\to_{\mathrm{elim}}$ in process graphs
  - Milner's process interpretation P of regular expressions
  - Loop existence and elimination property LEE
    LEE $\hat{=}$ interpretations of $(*/\pm)$ reg. expr's
  - $\to_{\mathrm{elim}}$ is not confluent
- Completion of loop elimination $\to_{\mathrm{elim}}$ with pruning steps $\to_{\mathrm{prune}}$
  - 'critical pair' lemma for 'bi-loop' elimination steps
  - use of decreasing diagrams
- Application 1: LEE is decidable in polynomial time
- Application 2: Layered LEE = LEE
- Further questions

Loop elimination $\to_{\text{elim}}$



Process interpretation P of regular expressions *(Milner, 1984)*

$\mathbf{0} \xmapsto{P}$ deadlock δ, no observable behaviour

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

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

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

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

Process interpretation P of regular expressions *(Milner, 1984)*

$\mathbf{0} \overset{P}{\longmapsto}$ deadlock δ, no observable behaviour

$\mathbf{0}^* \overset{P}{\longmapsto}$ empty-step process ϵ, then terminate

$a \overset{P}{\longmapsto}$ atomic action a, then terminate

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

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

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

Process semantics $[\![\cdot]\!]_P$ of regular expressions *(Milner, 1984)*

$\mathbf{0} \xrightarrow{P}$ deadlock $\boldsymbol{\delta}$, no observable behaviour

$\mathbf{1} \xrightarrow{P}$ empty-step process $\boldsymbol{\epsilon}$, 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 semantics $[\![\cdot]\!]_P$ of regular expressions *(Milner, 1984)*

$\mathbf{0} \xrightarrow{P}$ deadlock δ, no observable behaviour

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

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

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

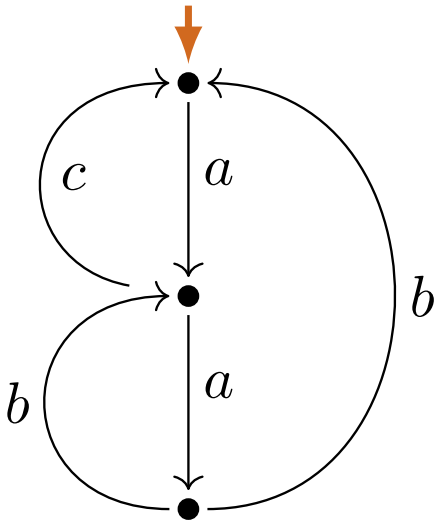
$e_1 \cdot e_2 \xrightarrow{P}$ *(sequentialization)* execute $P(e_1)$, then $P(e_2)$

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

$[\![e]\!]_P := [P(e)]_{\leftrightarrow}$ (bisimilarity equivalence class of process $P(e)$)

Process interpretation P (example)

G



$P(uf) \simeq G$

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

$$\downarrow a$$

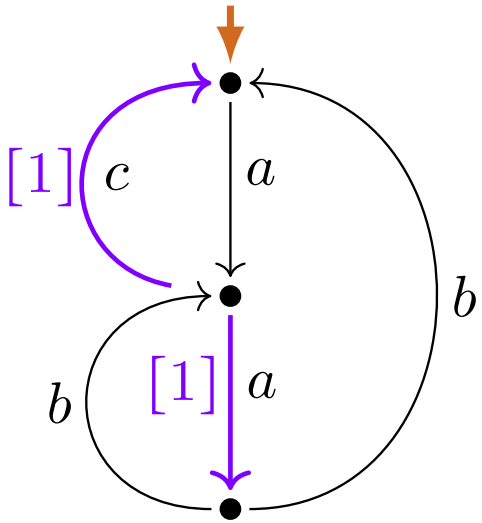
$$(1 \cdot (c \cdot a + a \cdot (b + b \cdot a))^*) \cdot 0$$

$$\downarrow a$$

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

Process interpretation P (example) has property LEE

G



$P(uf) \simeq G$

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

$$\downarrow a$$

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

$$\downarrow a$$

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

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) Immediate termination is only possible at the start vertex.

Loop graphs (interpretations of innermost iterations without 1)

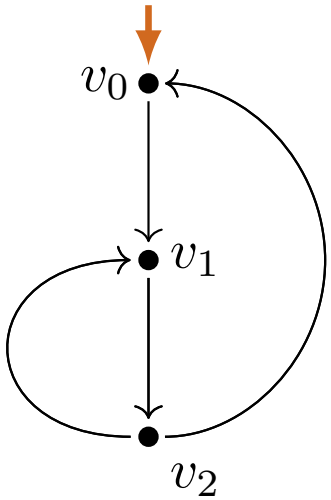
Definition

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) Immediate termination is only possible at the start vertex.



Loop graphs (interpretations of innermost iterations without 1)

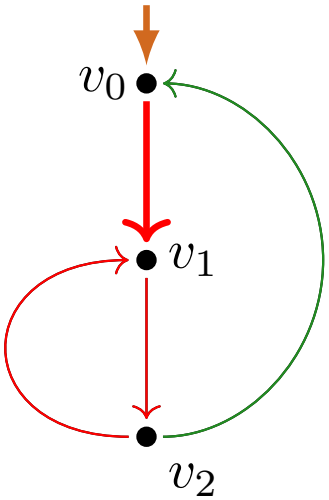
Definition

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) Immediate termination is only possible at the start vertex.



(L1),~~(L2)~~

Loop graphs (interpretations of innermost iterations without 1)

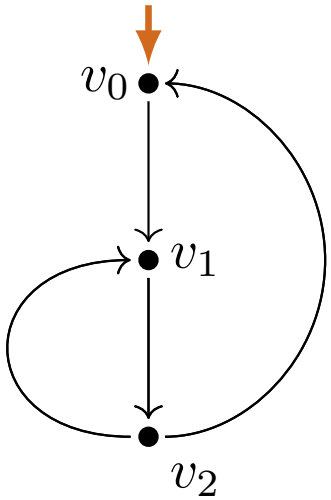
Definition

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) Immediate termination is only possible at the start vertex.



(L1),~~(L2)~~

Loop graphs (interpretations of innermost iterations without 1)

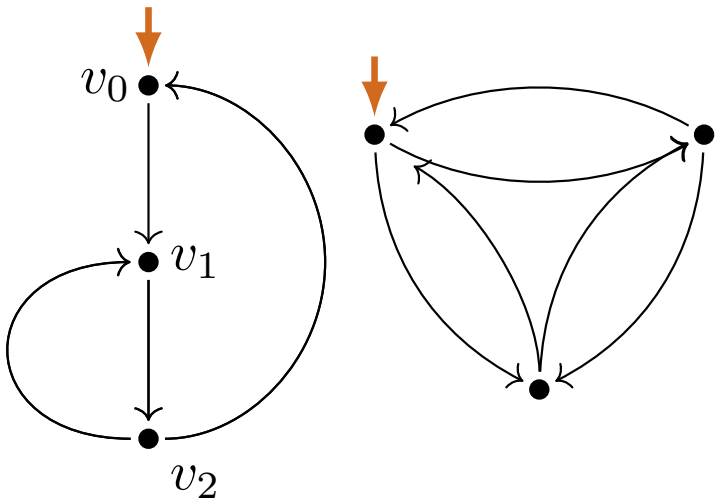
Definition

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) Immediate termination is only possible at the start vertex.



(L1),~~(L2)~~

Loop graphs (interpretations of innermost iterations without 1)

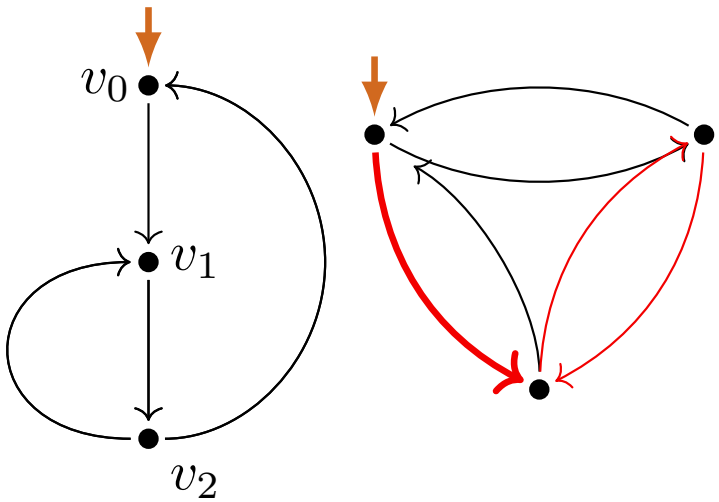
Definition

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) Immediate termination is only possible at the start vertex.



(L1),~~(L2)~~

Loop graphs (interpretations of innermost iterations without 1)

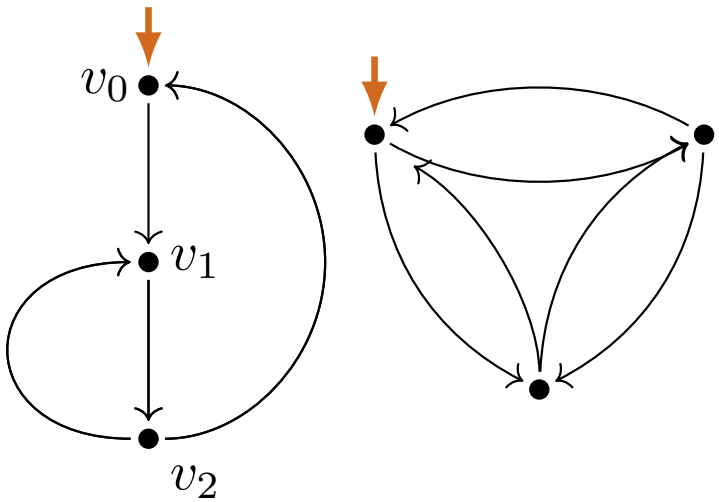
Definition

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) Immediate termination is only possible at the start vertex.



(L1),~~(L2)~~

Loop graphs (interpretations of innermost iterations without 1)

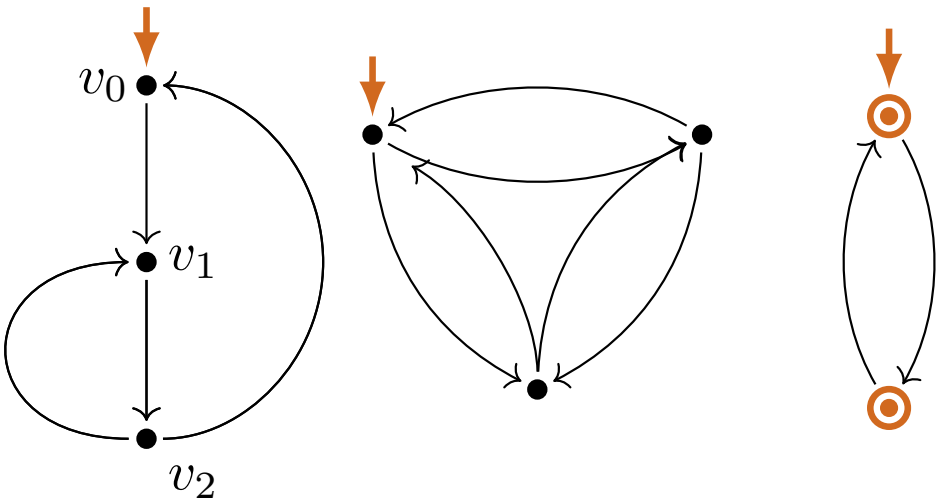
Definition

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) Immediate termination is only possible at the start vertex.



(L1),~~(L2)~~

Loop graphs (interpretations of innermost iterations without 1)

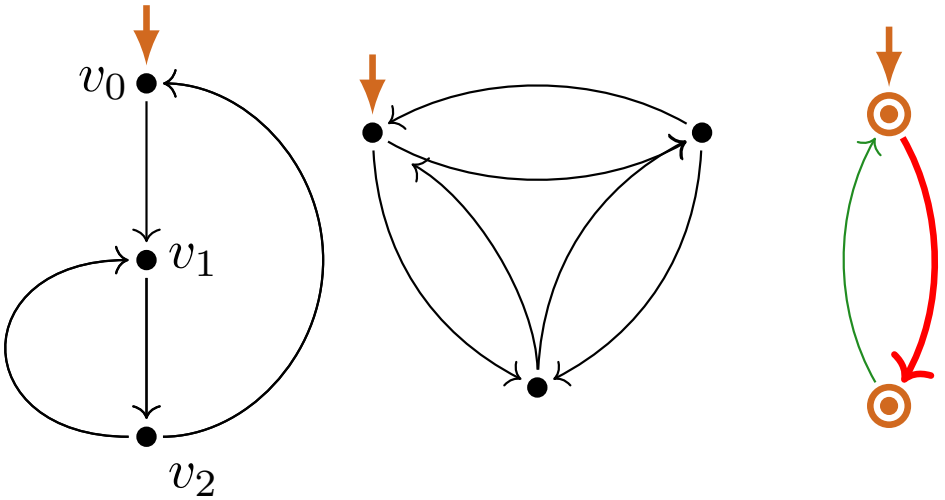
Definition

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) Immediate termination is only possible at the start vertex.



(L1),~~(L2)~~

Loop graphs (interpretations of innermost iterations without 1)

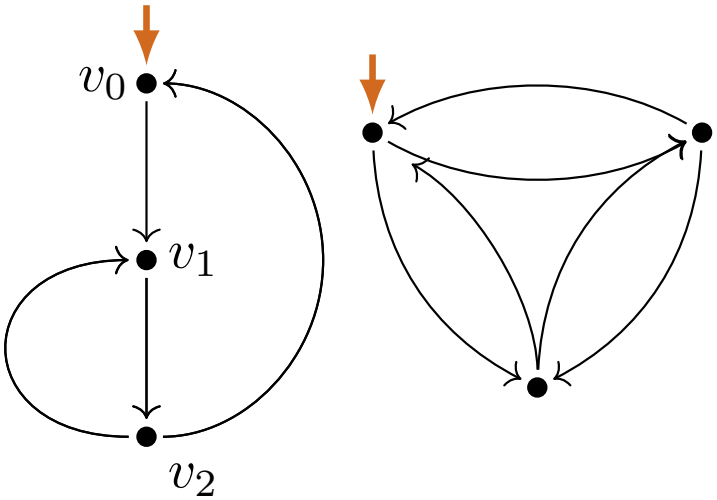
Definition

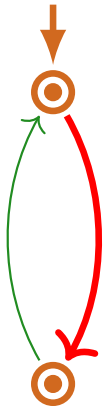
A process graph is a loop graph if:

(L1) There is an infinite path from the start vertex.

(L2) Every infinite path from the start vertex returns to it.

(L3) Immediate termination is only possible at the start vertex.





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

Loop graphs (interpretations of innermost iterations without 1)

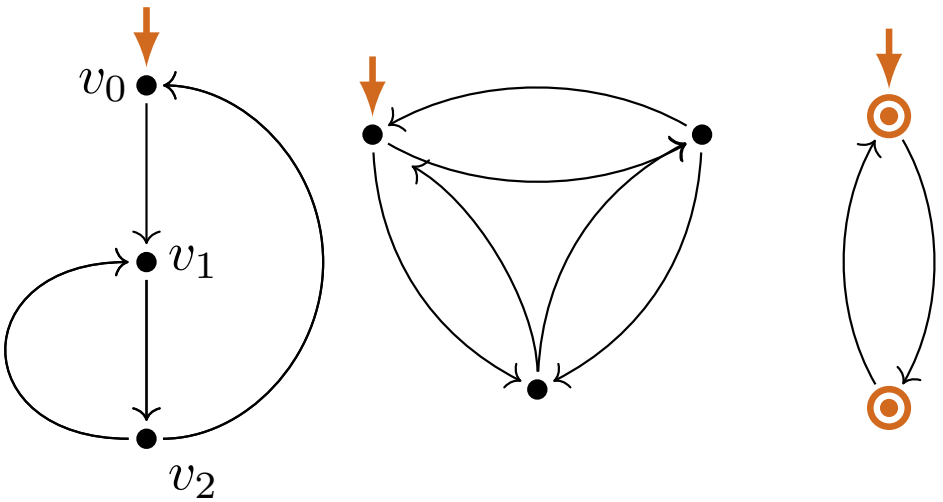
Definition

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) Immediate termination is only possible at the start vertex.



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

Loop graphs (interpretations of innermost iterations without 1)

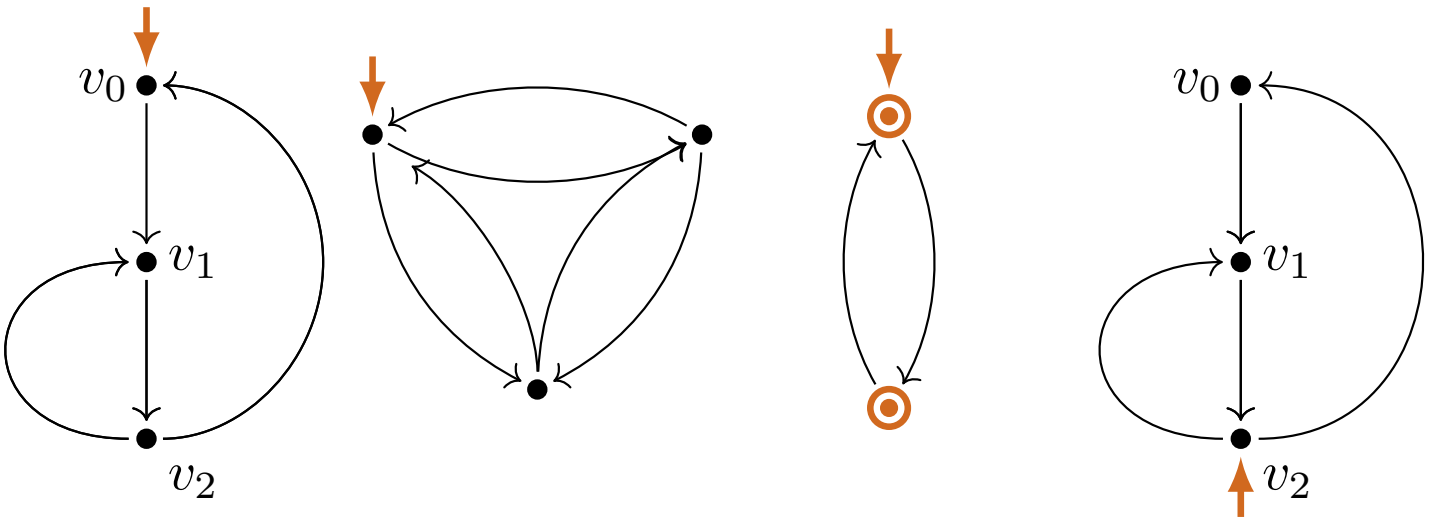
Definition

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) Immediate termination is only possible at the start vertex.



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

Loop graphs (interpretations of innermost iterations without 1)

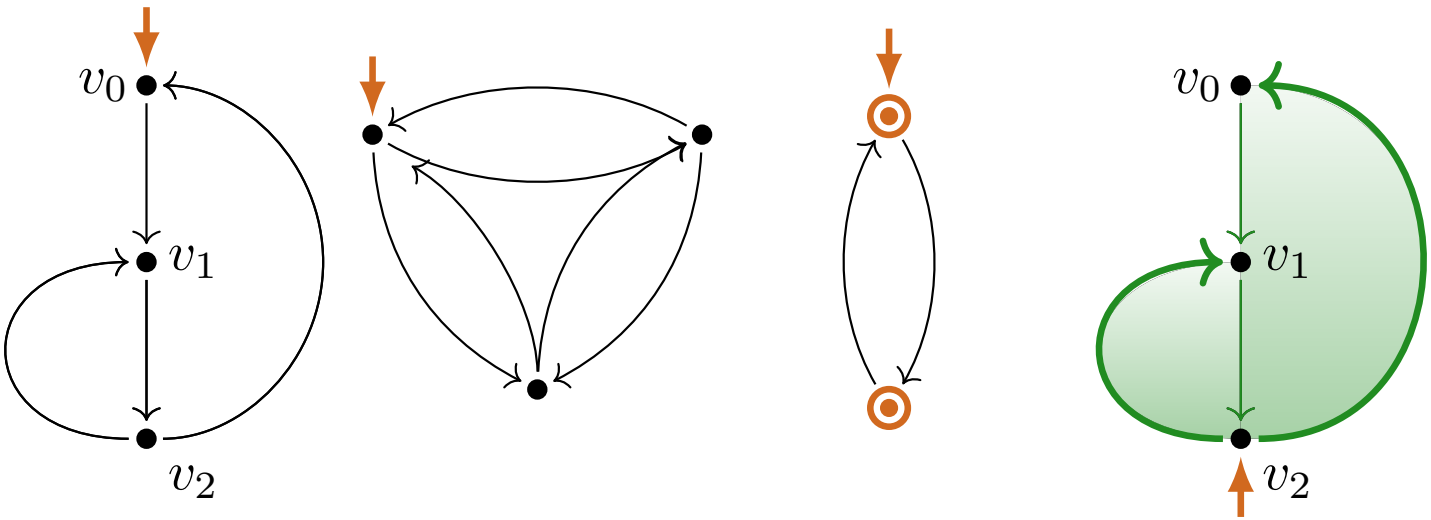
Definition

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) Immediate termination is only possible at the start vertex.



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

Loop graphs (interpretations of innermost iterations without 1)

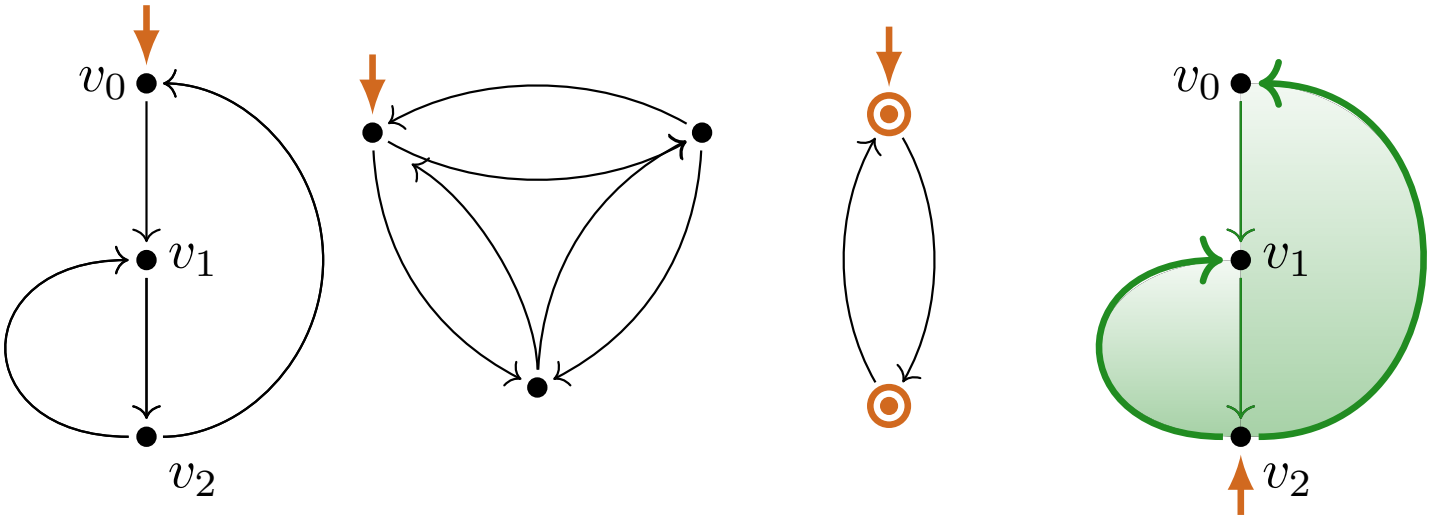
Definition

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) Immediate termination is only possible at the start vertex.



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

Loop graphs (interpretations of innermost iterations without 1)

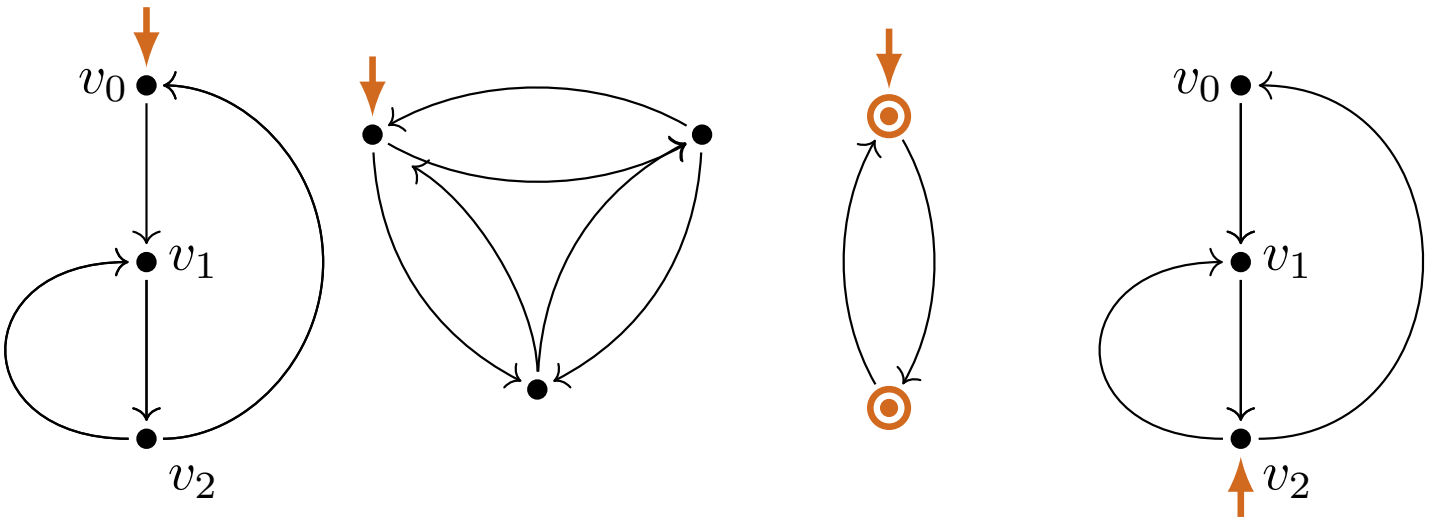
Definition

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) Immediate termination is only possible at the start vertex.



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

Loop graphs (interpretations of innermost iterations without 1)

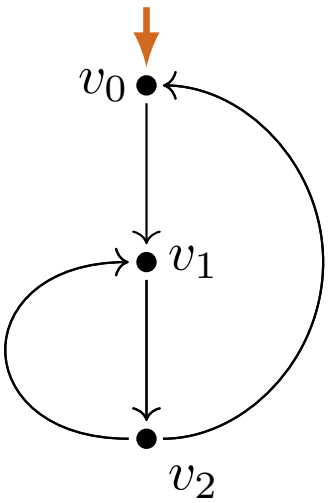
Definition

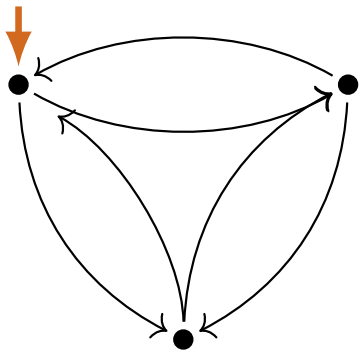
A process graph is a loop graph if:

(L1) There is an infinite path from the start vertex.

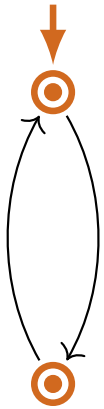
(L2) Every infinite path from the start vertex returns to it.

(L3) Immediate termination is only possible at the start vertex.

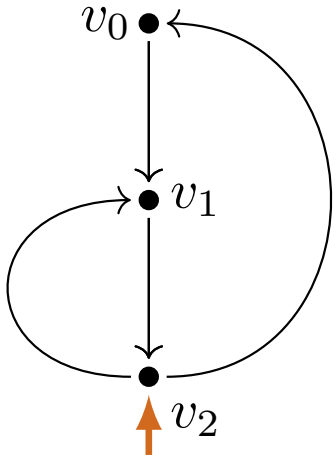


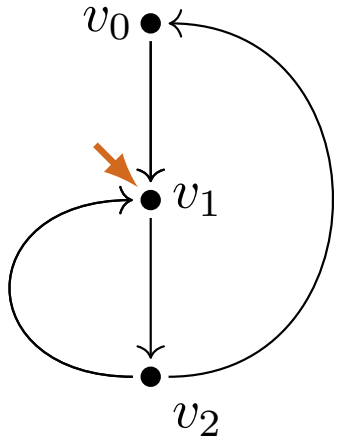


(L1),~~(L2)~~



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





loop chart

Loop graphs (interpretations of innermost iterations without 1)

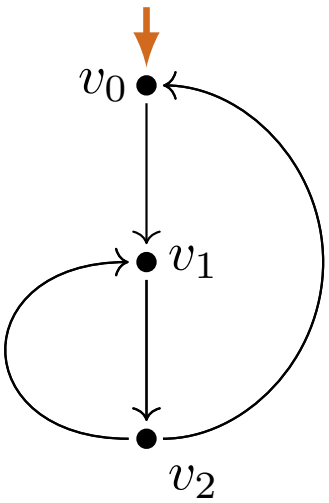
Definition

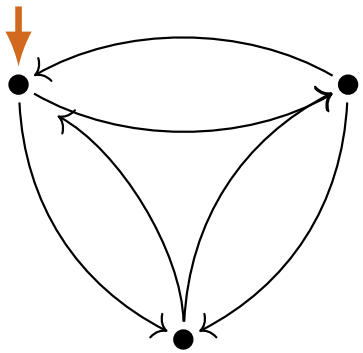
A process graph is a loop graph if:

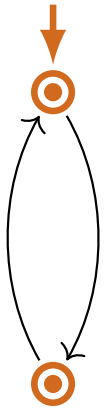
(L1) There is an infinite path from the start vertex.

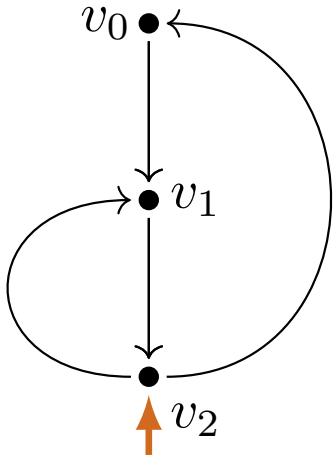
(L2) Every infinite path from the start vertex returns to it.

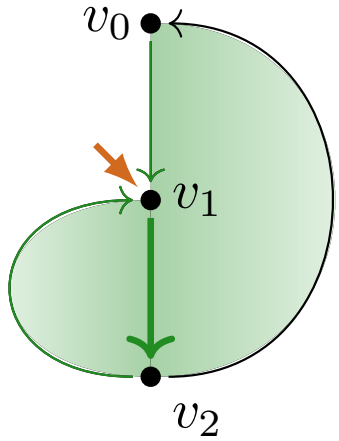
(L3) Immediate termination is only possible at the start vertex.











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

Loop graphs (interpretations of innermost iterations without 1)

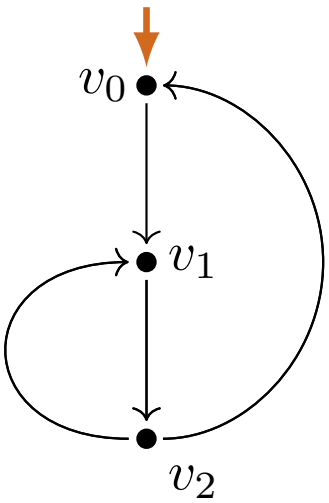
Definition

A process graph is a loop graph if:

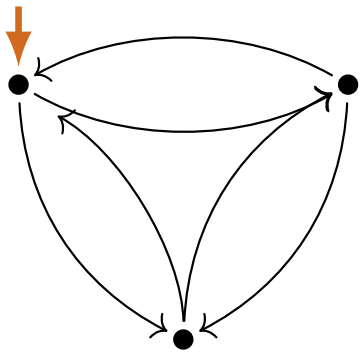
(L1) There is an infinite path from the start vertex.

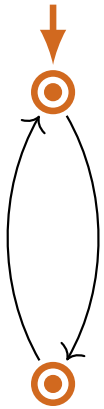
(L2) Every infinite path from the start vertex returns to it.

(L3) Immediate termination is only possible at the start vertex.

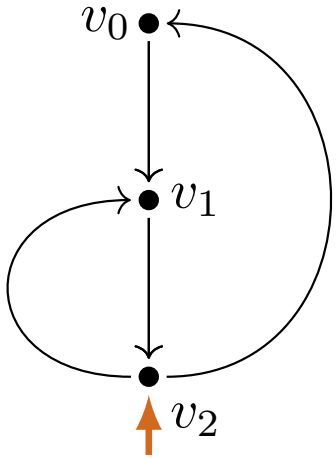


(L1),~~(L2)~~

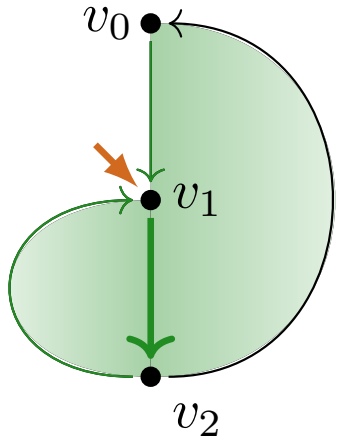




(L1),(L2),~~(L3)~~



loop chart



loop chart

Loop graphs (interpretations of innermost iterations without 1)

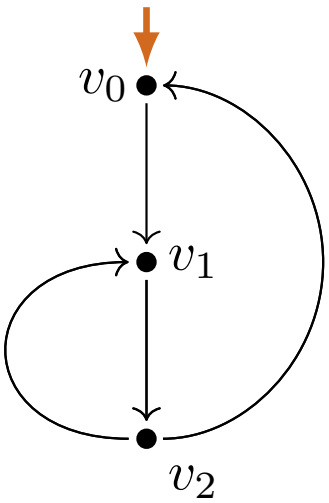
Definition

A process graph is a loop graph if:

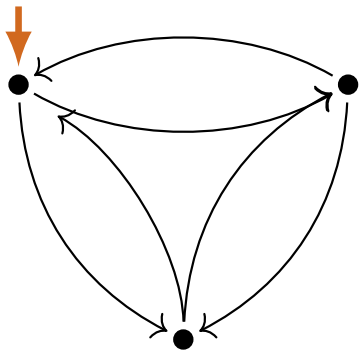
(L1) There is an infinite path from the start vertex.

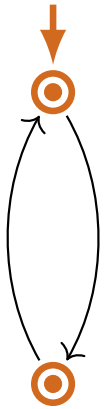
(L2) Every infinite path from the start vertex returns to it.

(L3) Immediate termination is only possible at the start vertex.

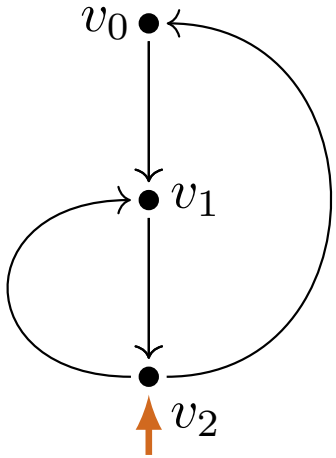


(L1),~~(L2)~~

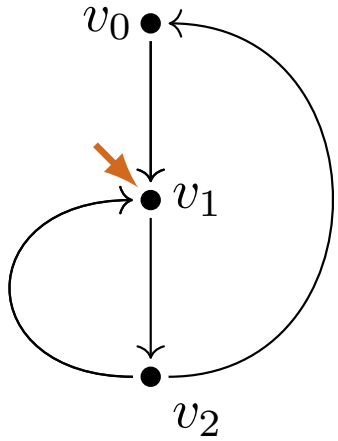




(L1),(L2),~~(L3)~~



loop chart



loop chart

Loop graphs (interpretations of innermost iterations without 1)

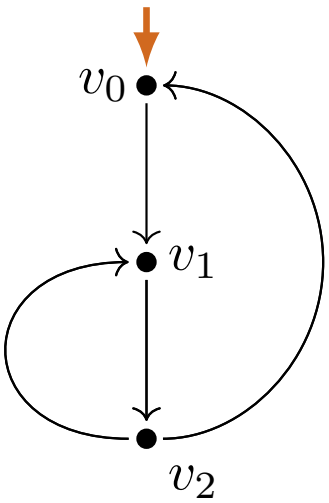
Definition

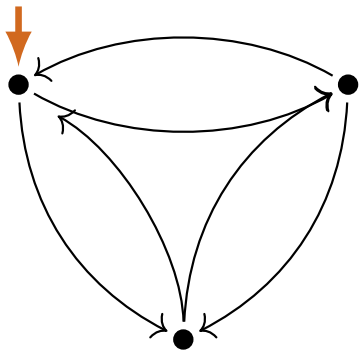
A process graph is a loop graph if:

(L1) There is an infinite path from the start vertex.

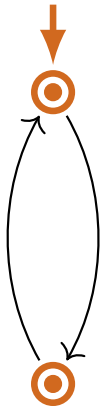
(L2) Every infinite path from the start vertex returns to it.

(L3) Immediate termination is only possible at the start vertex.

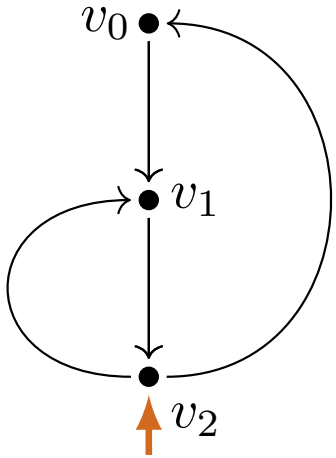


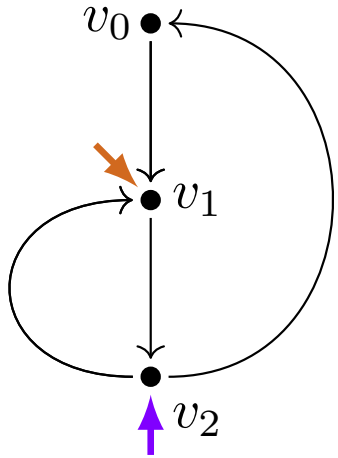


(L1),~~(L2)~~



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





loop chart

Loop graphs (interpretations of innermost iterations without 1)

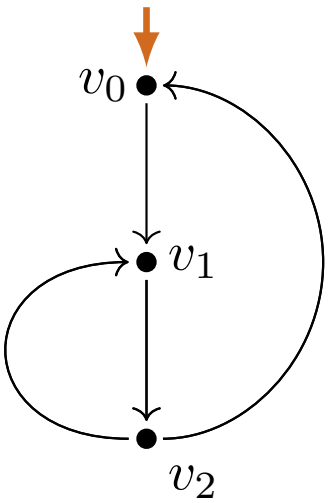
Definition

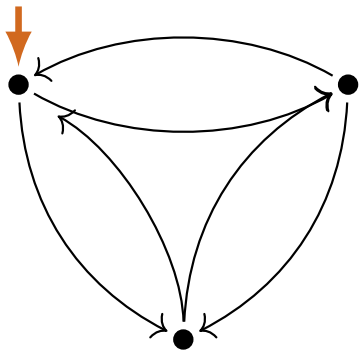
A process graph is a loop graph if:

(L1) There is an infinite path from the start vertex.

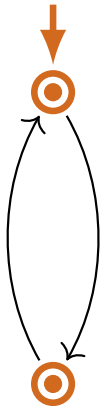
(L2) Every infinite path from the start vertex returns to it.

(L3) Immediate termination is only possible at the start vertex.

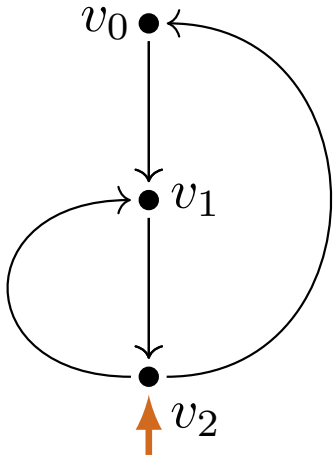




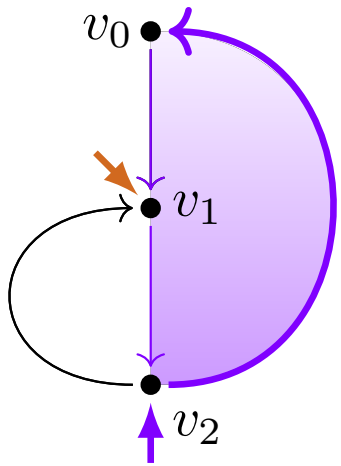
(L1),~~(L2)~~



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



loop chart



Loop graphs (interpretations of innermost iterations without 1)

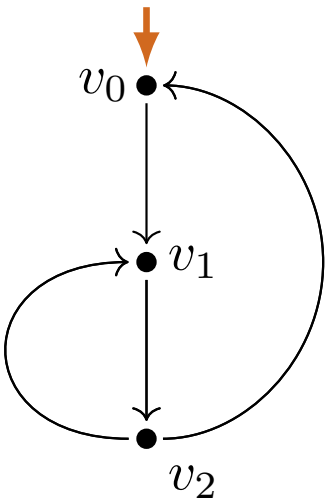
Definition

A process graph is a loop graph if:

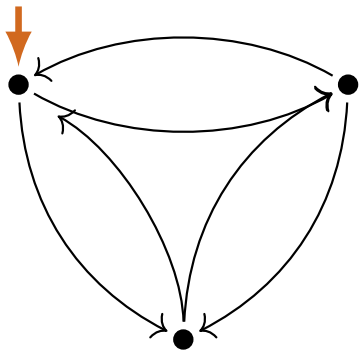
(L1) There is an infinite path from the start vertex.

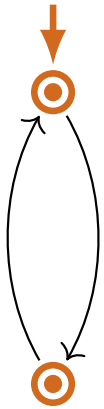
(L2) Every infinite path from the start vertex returns to it.

(L3) Immediate termination is only possible at the start vertex.

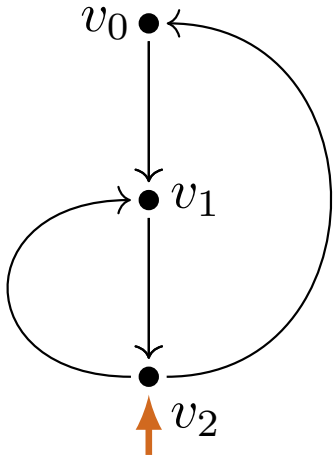


(L1),~~(L2)~~

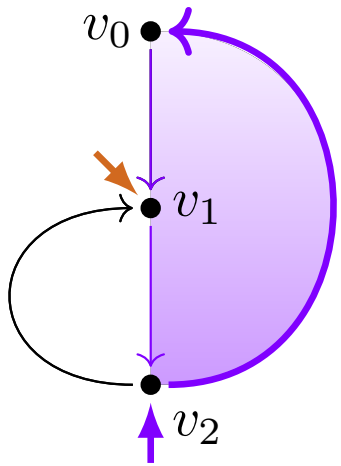




(L1),(L2),~~(L3)~~



loop chart



loop subchart

Loop graphs (interpretations of innermost iterations without 1)

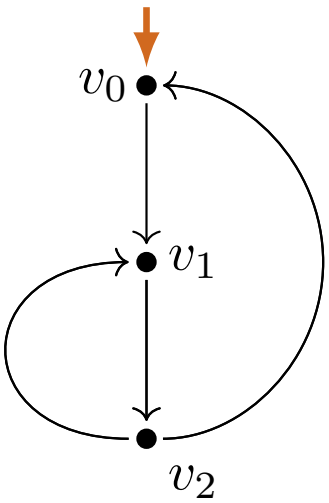
Definition

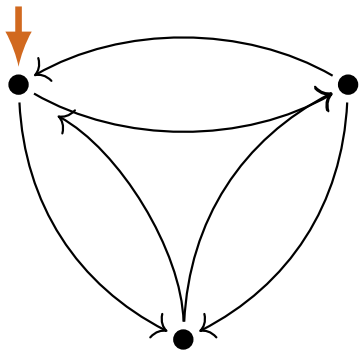
A process graph is a loop graph if:

(L1) There is an infinite path from the start vertex.

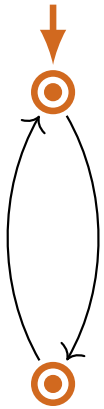
(L2) Every infinite path from the start vertex returns to it.

(L3) Immediate termination is only possible at the start vertex.

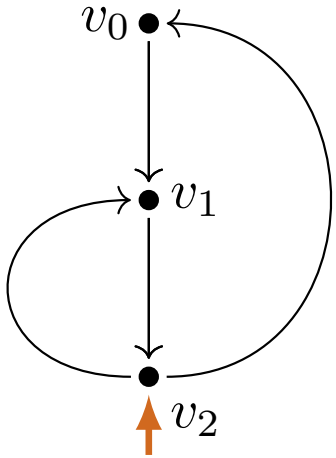


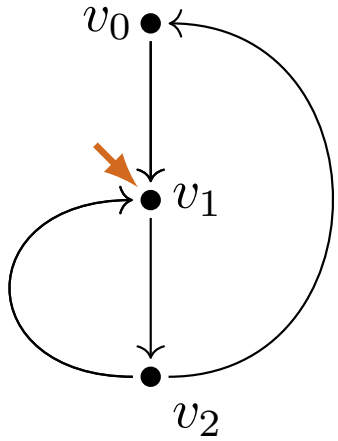


(L1),~~(L2)~~



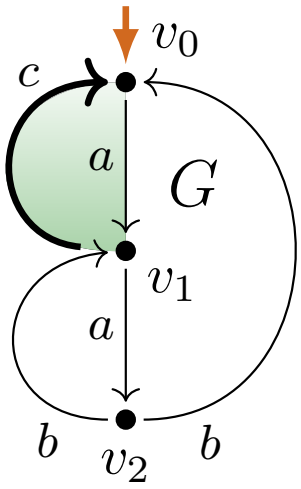
(L1),(L2),~~(L3)~~



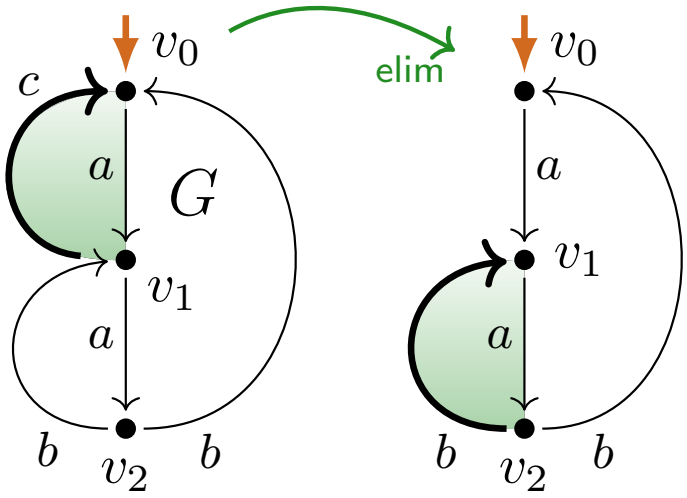


loop chart

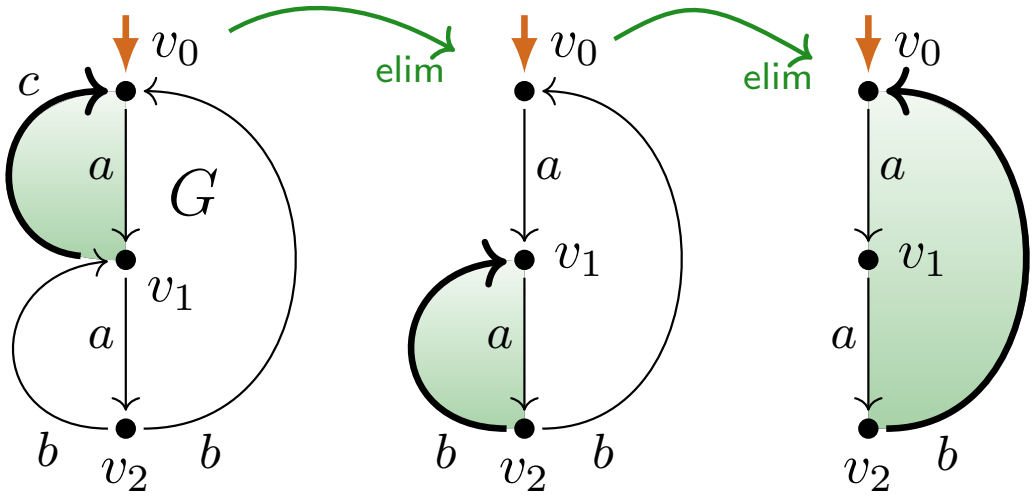
Loop elimination $\to_{\mathrm{elim}}$



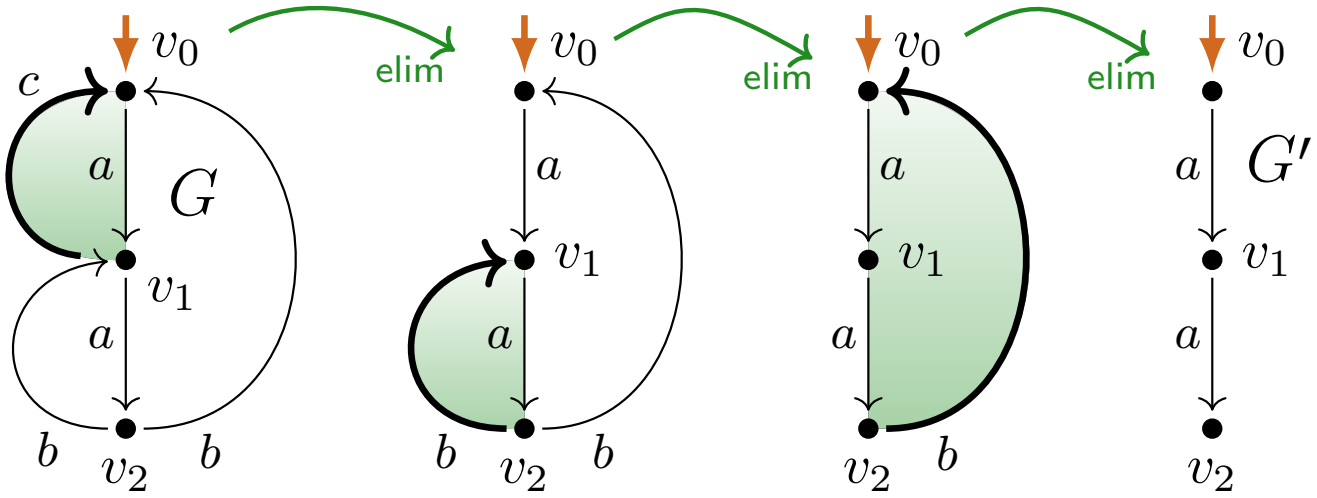
Loop elimination $\rightarrow_{\text{elim}}$



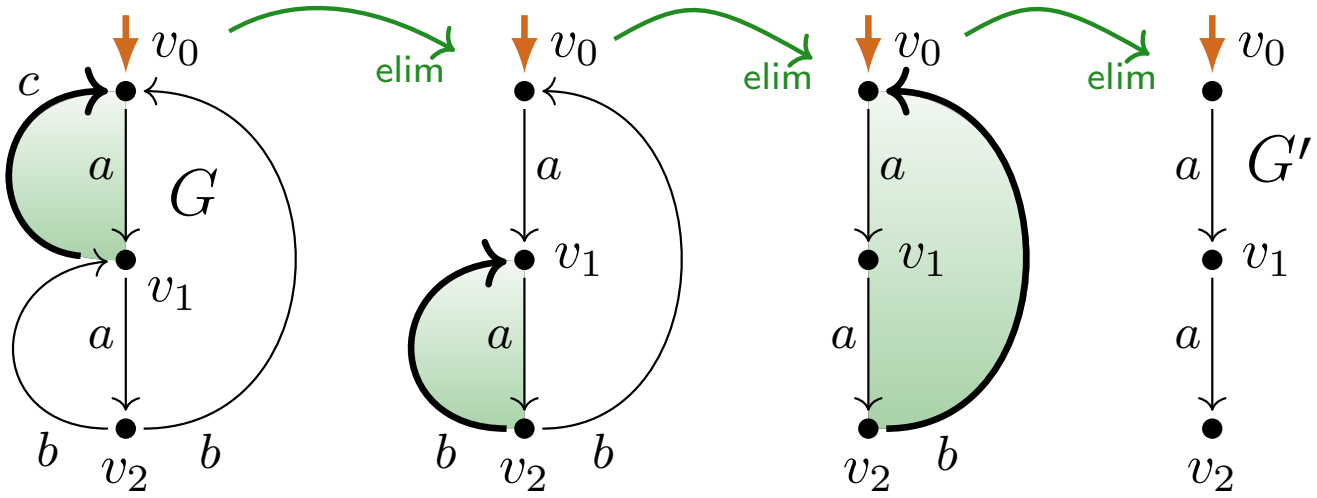
Loop elimination $\to_{\text{elim}}$

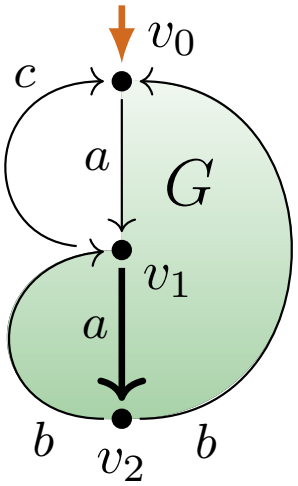


Loop elimination $\rightarrow_{\mathrm{elim}}$

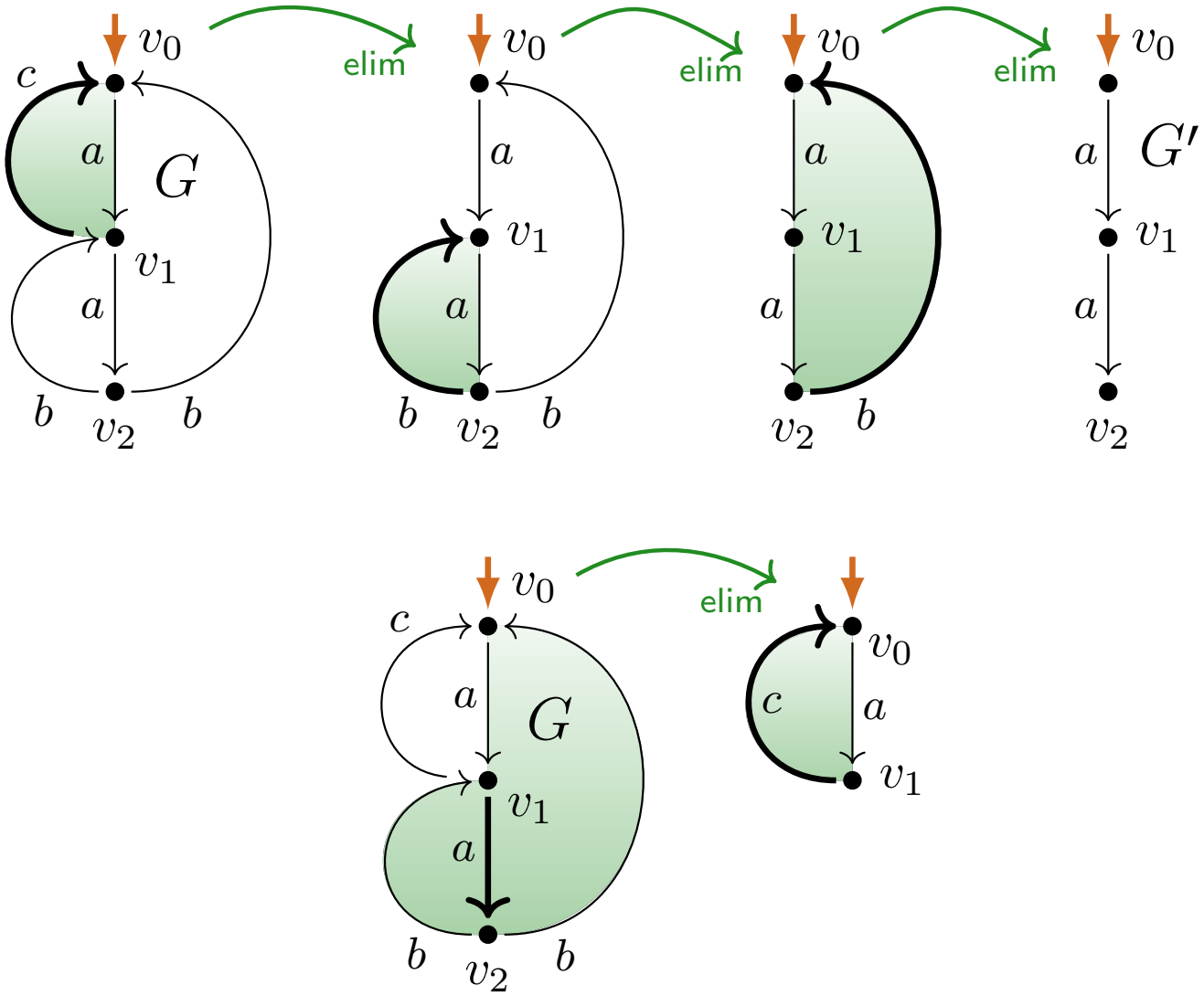


Loop elimination $\rightarrow_{\text{elim}}$

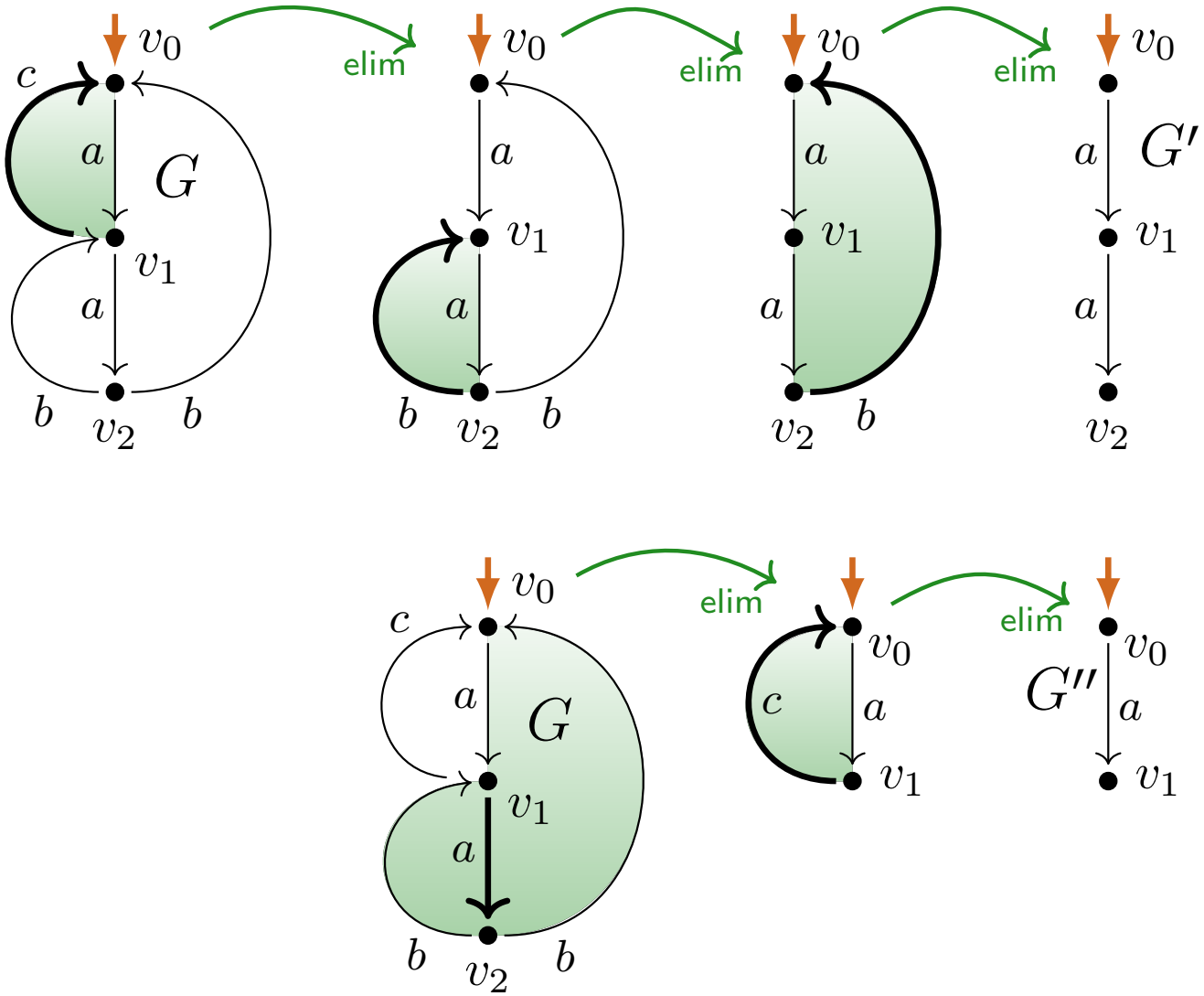




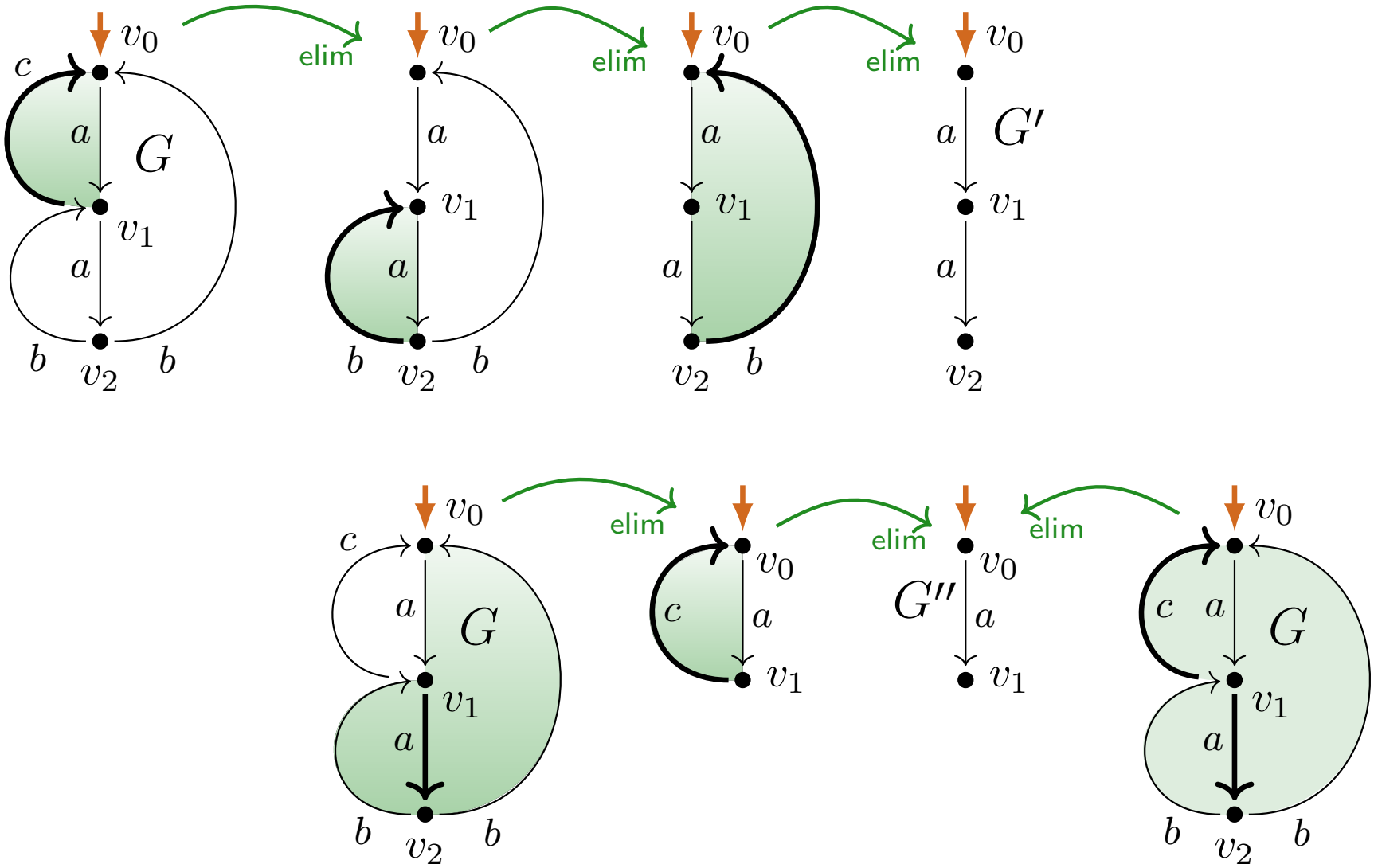
Loop elimination $\rightarrow_{\text{elim}}$



Loop elimination $\rightarrow_{\text{elim}}$



Loop elimination $\rightarrow_{\text{elim}}$



LEE

Definition

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

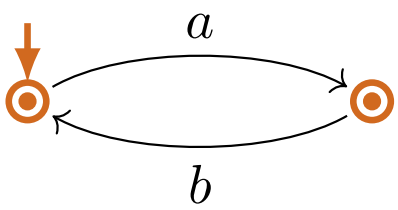
$$\exists G_0 \left(G \to^*_{\text{elim}} G_0 \not\to_{\text{elim}} \quad \wedge \quad \underbrace{\neg\infty(G_0)}_{G_0 \text{ permits no infinite trace}} \right).$$

LEE

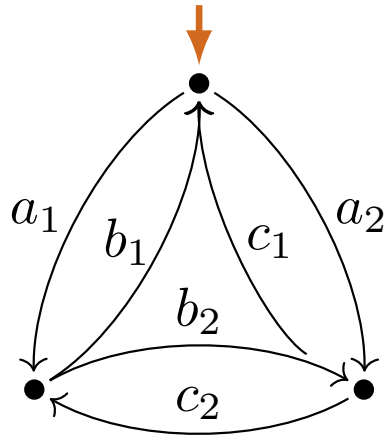
Definition

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

$$\exists G_0 \left(G \to^*_{\text{elim}} G_0 \not\to_{\text{elim}} \ \wedge \ \underbrace{\neg\infty(G_0)}_{G_0 \text{ permits no infinite trace}} \right).$$

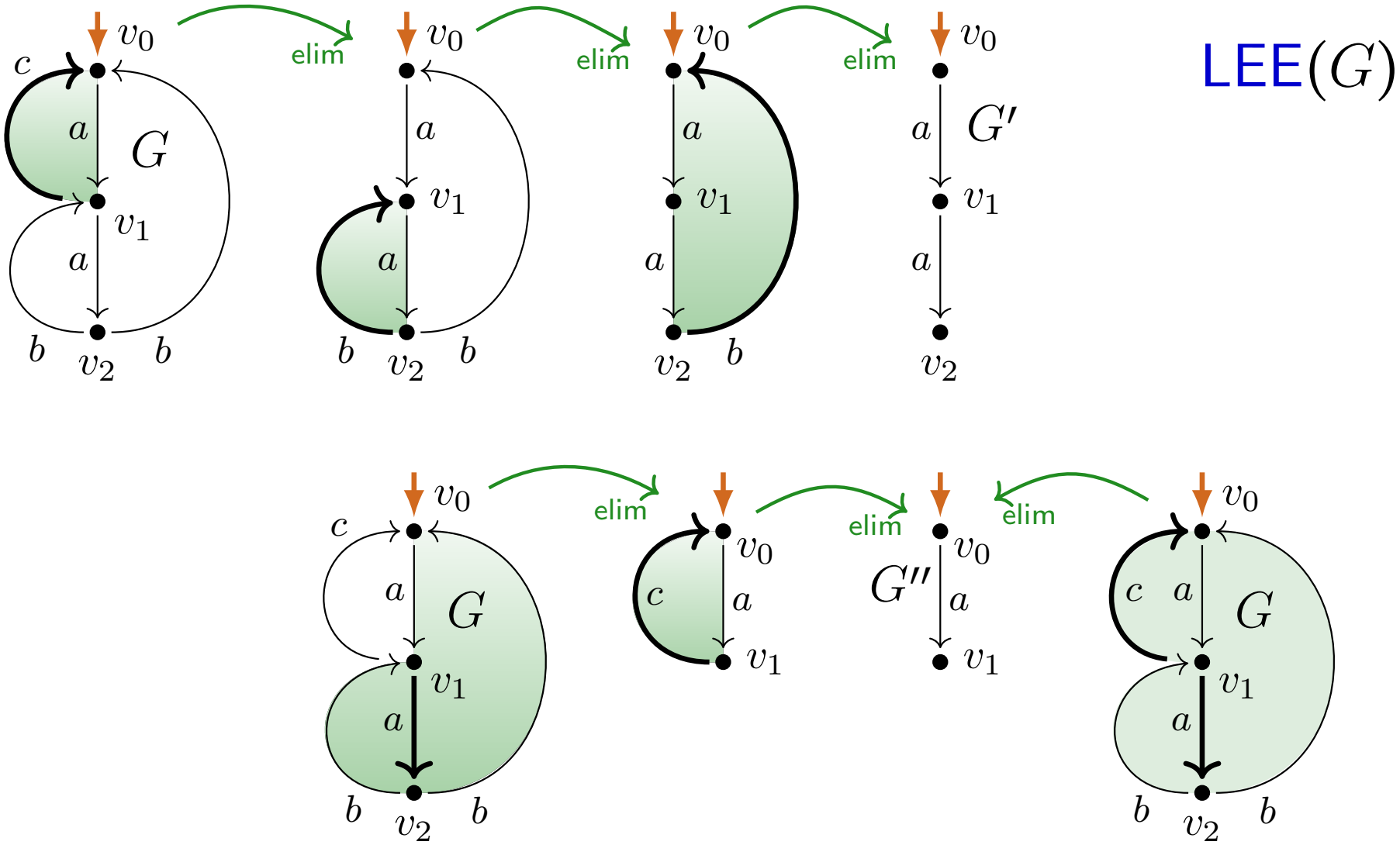


not LEE



not LEE

Loop elimination $\to_{\text{elim}}$



LEE $\triangleq$ P-/$P^\bullet$-expressible graphs for $(*/\pm)$ reg. expr's

Lemma (*G–Fokkink, LICS'20*)

For all process graphs G:

$$\mathsf{LEE}(G) \implies \exists\, uf \in \mathit{RExp}^{(*/\pm)} \big(\underbrace{G \leftrightarrows P(uf)}_{G \in [\![uf]\!]_P} \big),$$

$$\mathsf{LEE}(G) \impliedby \exists\, uf \in \mathit{RExp}^{(*/\pm)} \big(G \simeq P(uf) \big).$$

LEE $\triangleq$ P-/$P^\bullet$-expressible graphs for $(*/\pm)$ reg. expr's

Lemma (*G–Fokkink, LICS'20*)

For all process graphs G:

$$\mathsf{LEE}(G) \implies \exists\, uf \in \mathit{RExp}^{(*/\pm)} \big(\underbrace{G \leftrightarrows P(uf)}_{G \in [\![uf]\!]_P} \big),$$

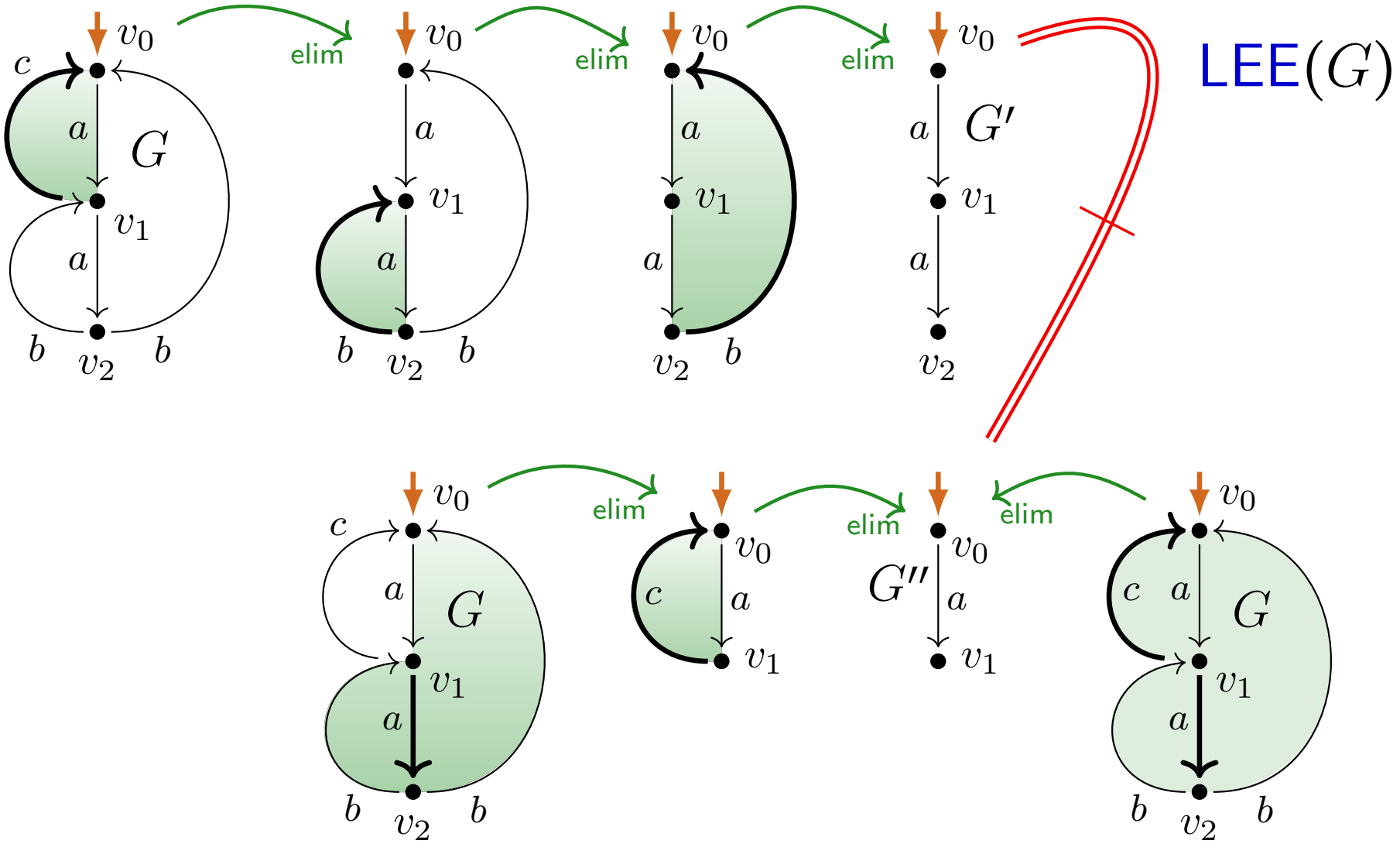
$$\mathsf{LEE}(G) \impliedby \exists\, uf \in \mathit{RExp}^{(*/\pm)} \big(G \simeq P(uf) \big).$$

Theorem (*TERMGRAPH '26*, LEE = LLEE *from IWC'26*)

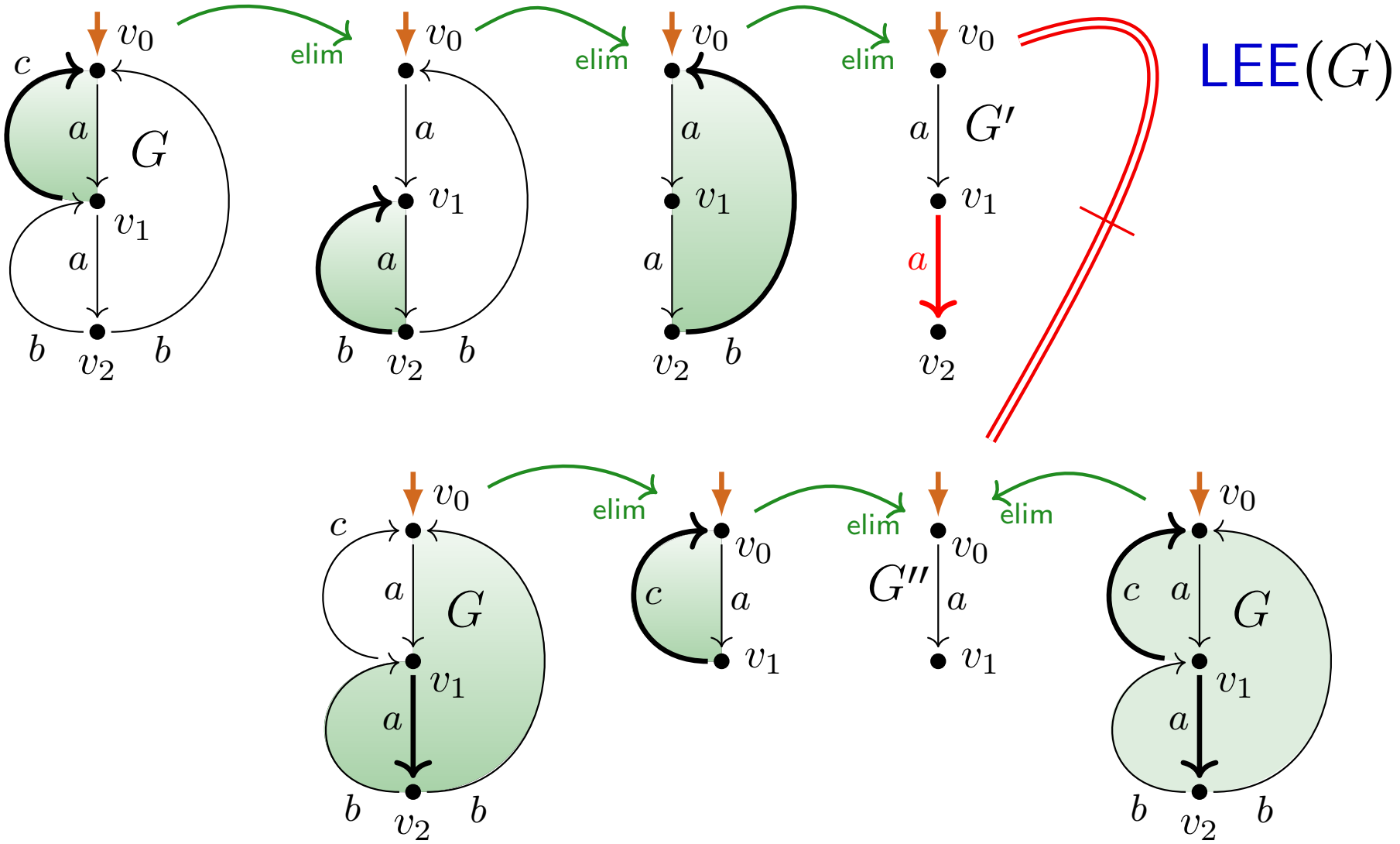
LEE $\triangleq$ compact process interpretations $P^\bullet(uf)$ of $(*/\pm)$ reg. expr's uf:

$$\mathsf{LEE}(G) \iff \exists\, uf \in \mathit{RExp}^{(*/\pm)} \big(G \simeq P^\bullet(uf) \big).$$

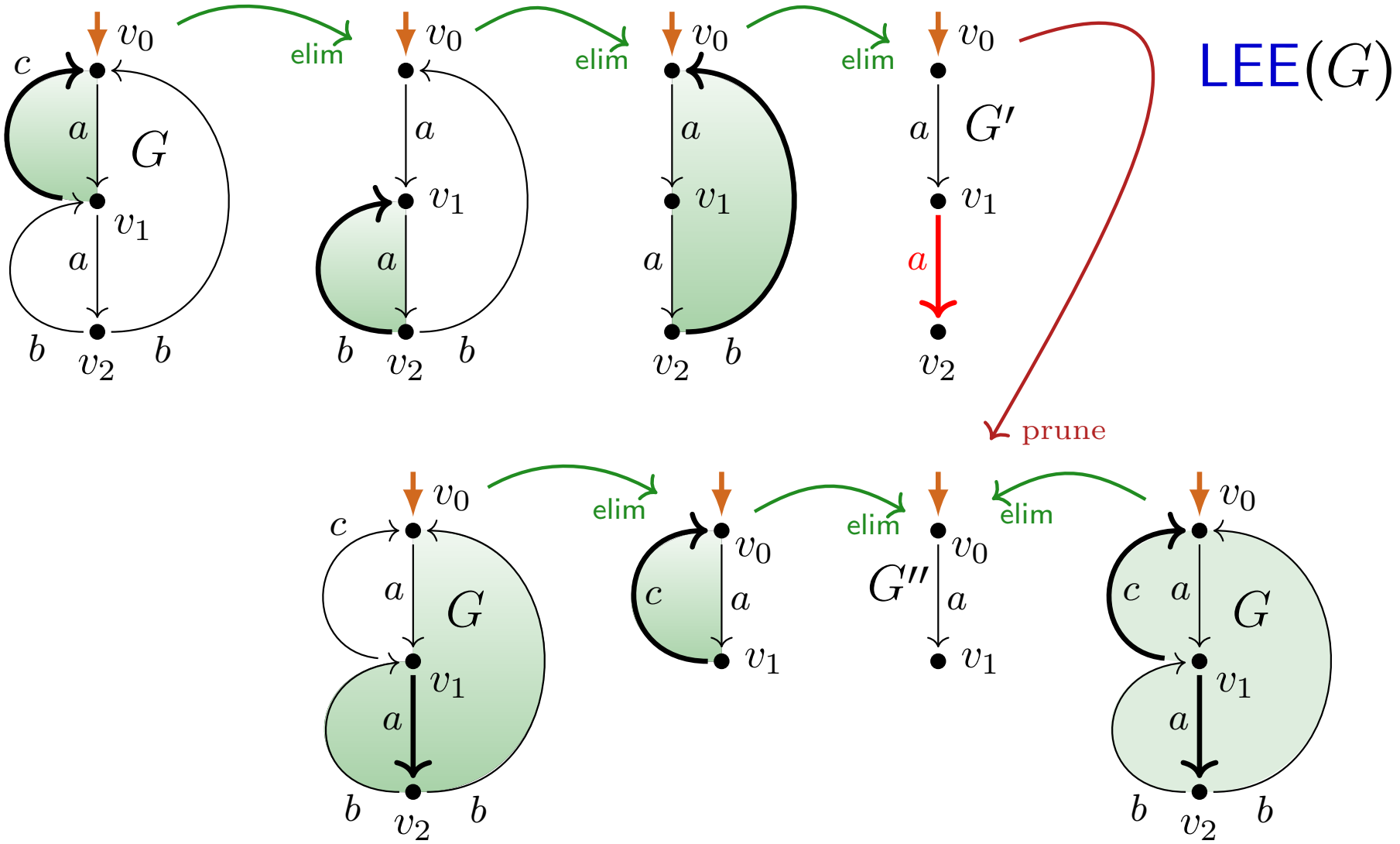
Loop elimination $\rightarrow_{\text{elim}}$ is **not** confluent



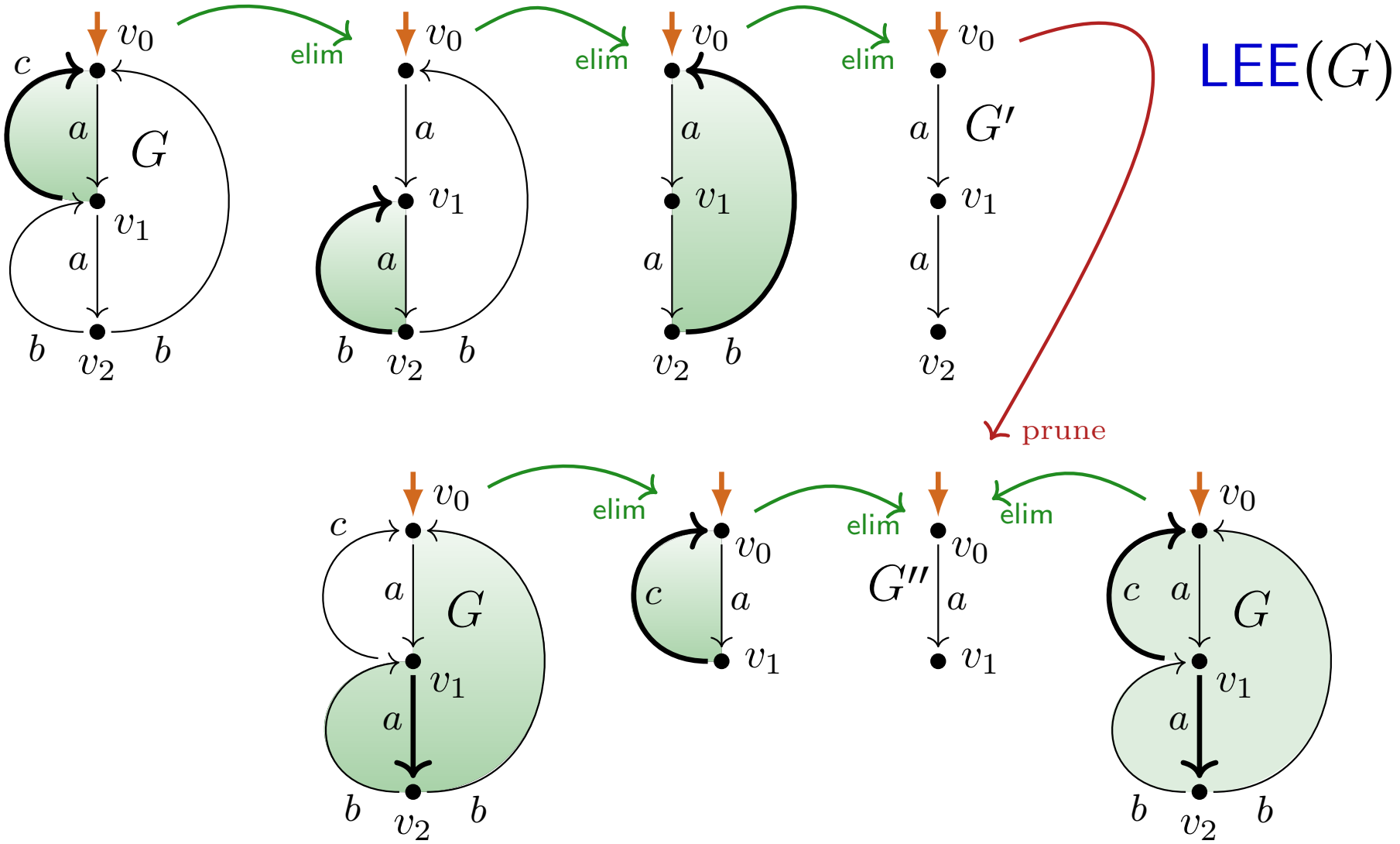
Loop elimination $\rightarrow_{\text{elim}}$ is **not** confluent



Joining an $\rightarrow_{\text{elim}}$ fork with a pruning $\rightarrow_{\text{prune}}$ step



Joining an $\rightarrow_{\text{elim}}$ fork with a pruning $\rightarrow_{\text{prune}}$ step



Overview

- Loop elimination $\to_{\mathrm{elim}}$ in process graphs
  - Milner's process interpretation P of regular expressions
  - Loop existence and elimination property LEE
    LEE $\triangleq$ interpretations of $(*/1)$ reg. expr's
  - $\to_{\mathrm{elim}}$ is not confluent
- Completion of loop elimination $\to_{\mathrm{elim}}$ with pruning steps $\to_{\mathrm{prune}}$
  - 'critical pair' lemma for 'bi-loop' elimination steps
  - use of decreasing diagrams
- Application 1: LEE is decidable in polynomial time
- Application 2: Layered LEE = LEE
- Further questions

‘Critical Pair Lemma’

Lemma (joining $\rightarrow_{\mathrm{elim}}$ forks)

Forks $G_2 \leftarrow_{\mathrm{elim}} G \rightarrow_{\mathrm{elim}} G_1$ in which loop subgraphs $G{\downarrow_*^{v_2,T_2}}$, $G{\downarrow_*^{v_1,T_1}}$ of G are eliminated, can be joined with $\rightarrow_{\mathrm{prune}}$ steps in the three situations:

‘Critical Pair Lemma’

Lemma (joining $\to_{\mathrm{elim}}$ forks)

Forks $G_2 \leftarrow_{\mathrm{elim}} G \to_{\mathrm{elim}} G_1$ in which loop subgraphs $G{\downarrow_*^{v_2,T_2}}$, $G{\downarrow_*^{v_1,T_1}}$ of G are eliminated, can be joined with $\to_{\mathrm{prune}}$ steps in the three situations:

(i) Root-overlap: $v_1 = v_2$, and the loops form bigger loop $G{\downarrow_*^{v_1,T_1 \cup T_2}}$.

(ii) Parallel/nested: $v_1 \neq v_2$, the cyclic parts $\circlearrowleft(G{\downarrow_*^{v_1,T_1}})$, $\circlearrowleft(G{\downarrow_*^{v_2,T_2}})$ of the loops are vertex-disjoint.

(iii) Non-root overlap: $v_1 \neq v_2$, the cyclic parts $\circlearrowleft(G{\downarrow_*^{v_1,T_1}})$, and $\circlearrowleft(G{\downarrow_*^{v_2,T_2}})$ have common vertices.

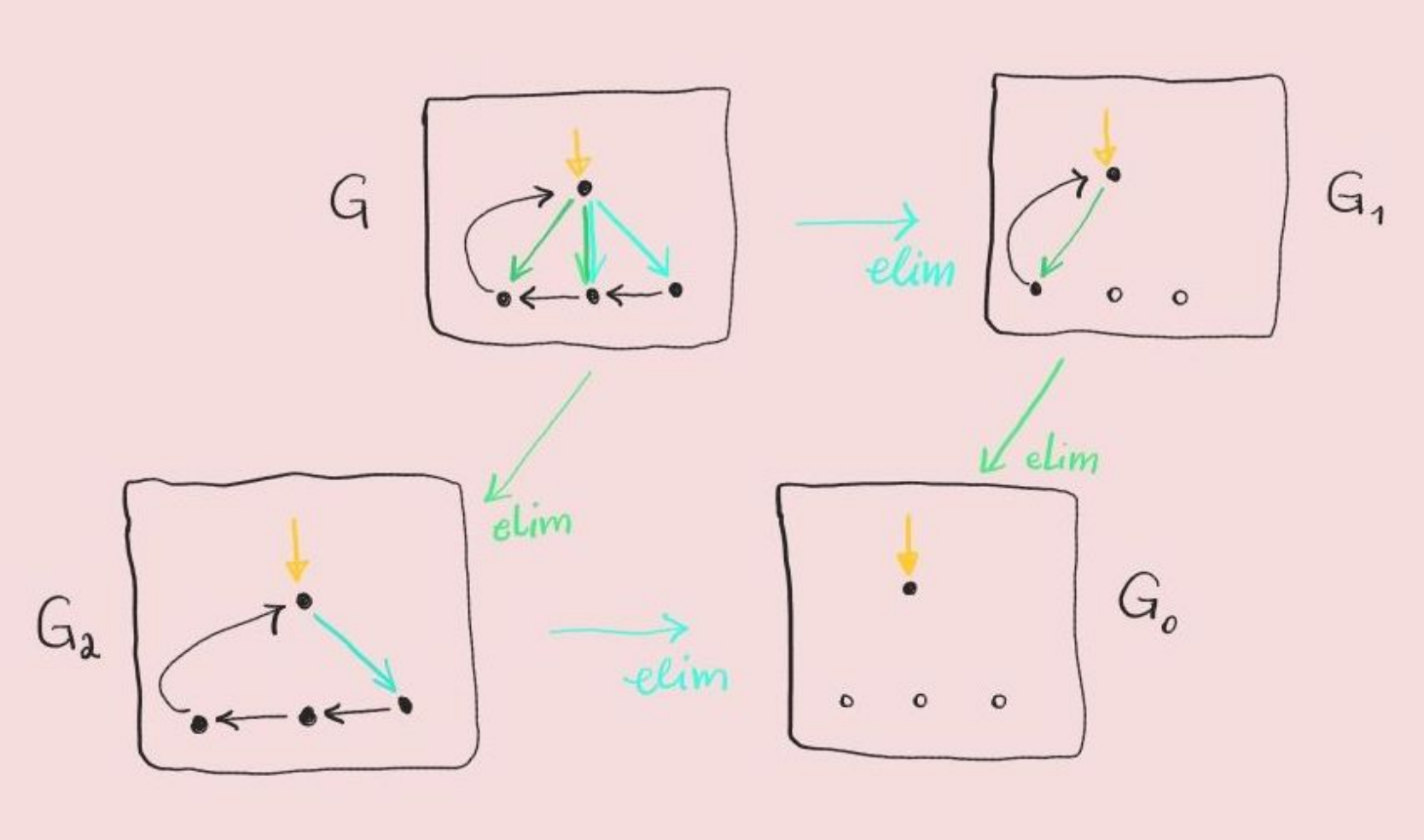
‘Critical Pair Lemma’

Lemma (joining $\to_{\mathrm{elim}}$ forks)

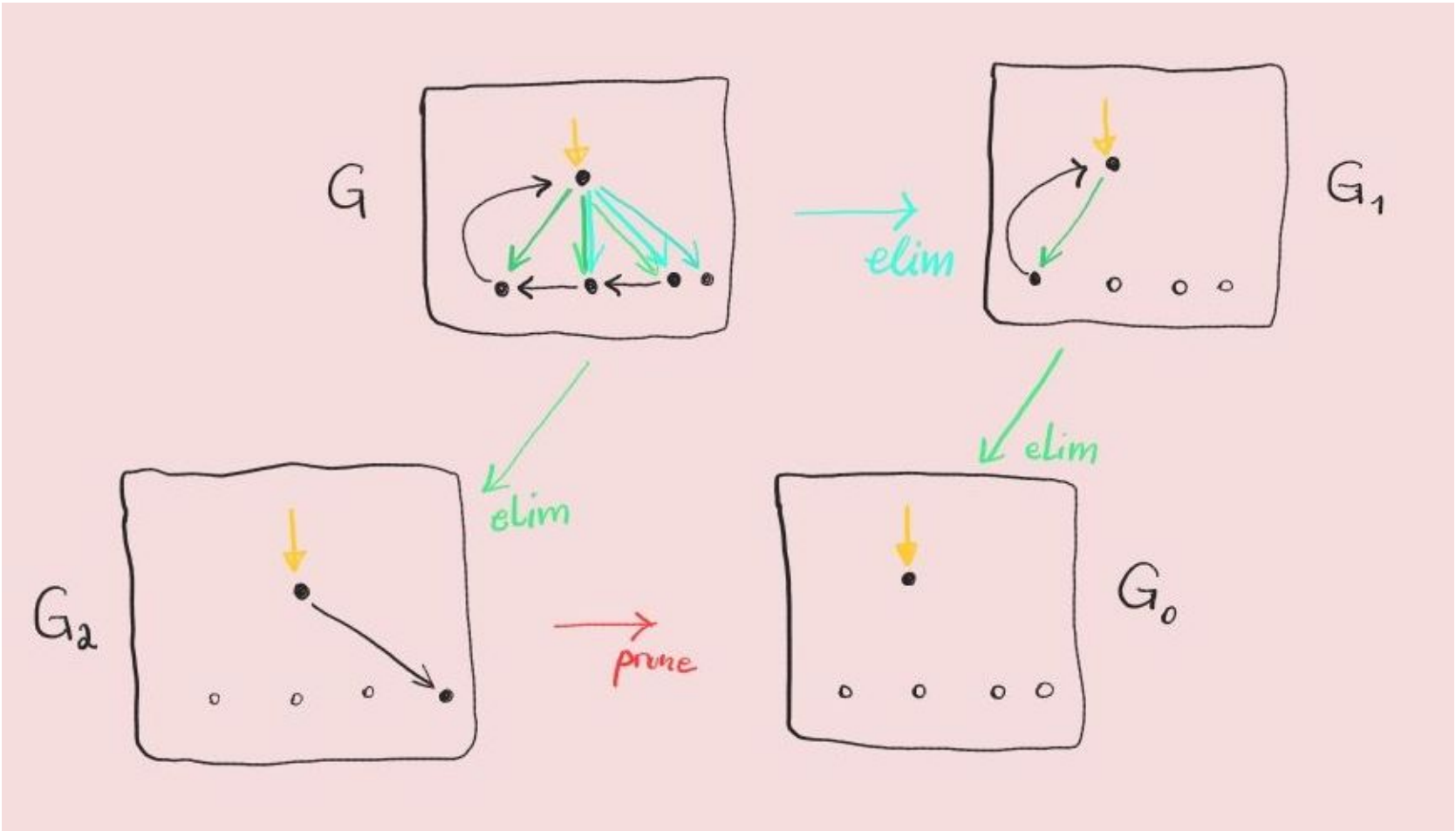
Forks $G_2 \leftarrow_{\mathrm{elim}} G \to_{\mathrm{elim}} G_1$ in which loop subgraphs $G{\downarrow_*^{v_2,T_2}}$, $G{\downarrow_*^{v_1,T_1}}$ of G are eliminated, can be joined with $\to_{\mathrm{prune}}$ steps in the three situations:

(i) Root-overlap: $v_1 = v_2$, and the loops form bigger loop $G{\downarrow_*^{v_1,T_1 \cup T_2}}$. The $\to_{\mathrm{elim}}$ steps may overlap in their loop-entry transitions, but the fork can always be joined by eliminating residual loop-entries as follows: $G_2 \to_{\mathrm{elim}}^{=} \cdot \to_{\mathrm{prune}}^{*} G_0 \leftarrow_{\mathrm{prune}}^{*} \cdot \leftarrow_{\mathrm{elim}}^{=} G_1$.

Root-overlapping loop subgraphs (example a)



Root-overlap of loop subgraphs (example b)



‘Critical Pair Lemma’

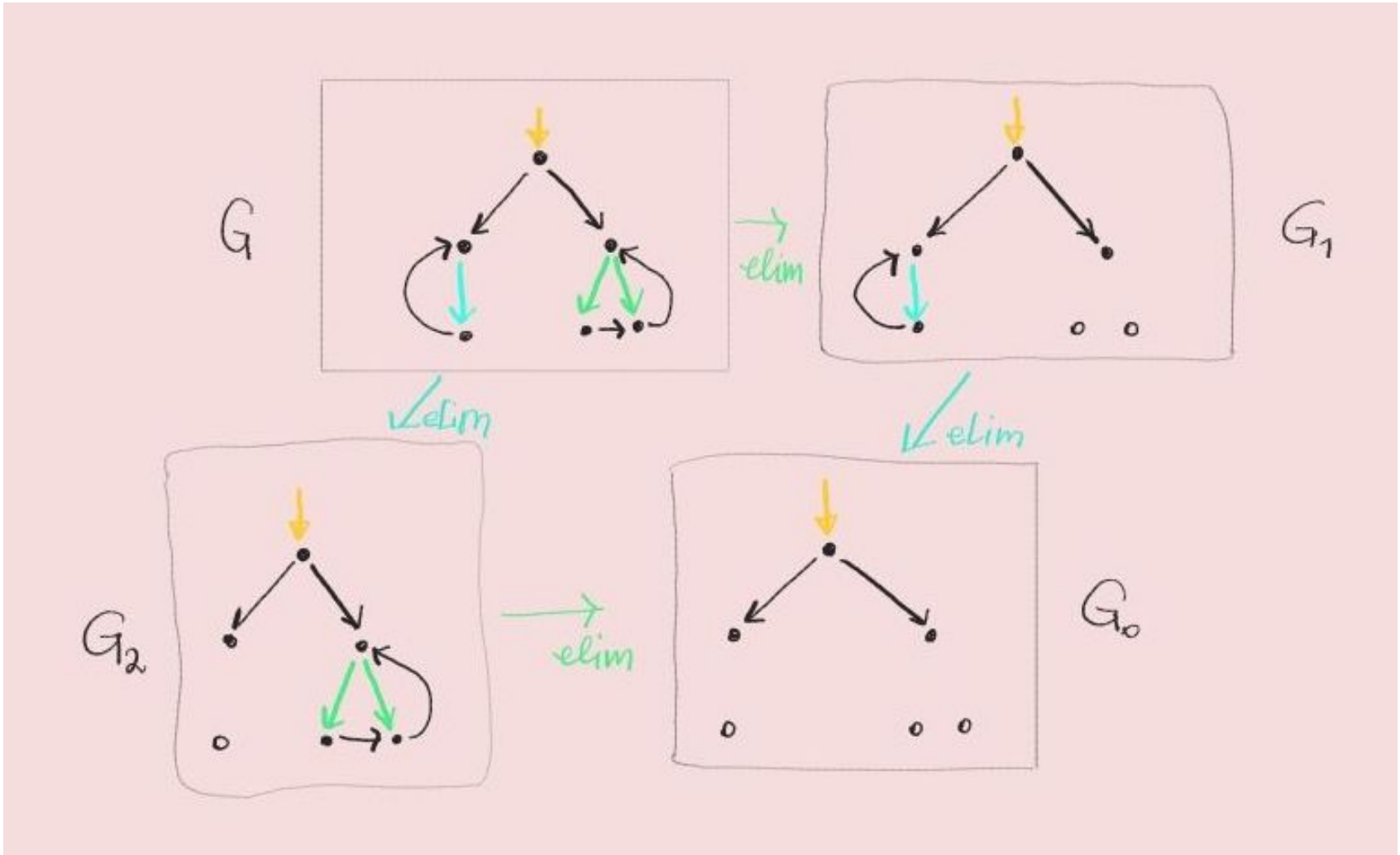
Lemma (joining $\to_{\text{elim}}$ forks)

Forks $G_2 \leftarrow_{\text{elim}} G \to_{\text{elim}} G_1$ in which loop subgraphs $G{\downarrow_*^{v_2,T_2}}$, $G{\downarrow_*^{v_1,T_1}}$ of G are eliminated, can be joined with $\to_{\text{prune}}$ steps in the three situations:

(i) Root-overlap: $v_1 = v_2$, and the loops form bigger loop $G{\downarrow_*^{v_1,T_1 \cup T_2}}$. The $\to_{\text{elim}}$ steps may overlap in their loop-entry transitions, but the fork can always be joined by eliminating residual loop-entries as follows: $G_2 \to_{\text{elim}}^{=} \cdot \to_{\text{prune}}^{*} G_0 \leftarrow_{\text{prune}}^{*} \cdot \leftarrow_{\text{elim}}^{=} G_1$.

(ii) Parallel/nested: $v_1 \neq v_2$, & the cyclic parts $\circlearrowleft(G{\downarrow_*^{v_1,T_1}})$, $\circlearrowleft(G{\downarrow_*^{v_2,T_2}})$ of the loops are vertex-disjoint. Then the $\to_{\text{elim}}$ steps are independent of each other, except that the elimination of one loop may eliminate the other in the garbage collection operation. The fork can always be joined as follows: $G_2 \to_{\text{elim}}^{=} G_0 \leftarrow_{\text{elim}}^{=} G_1$.

Parallel loop subgraphs (example)



'Critical Pair Lemma'

Lemma (joining $\to_{\text{elim}}$ forks)

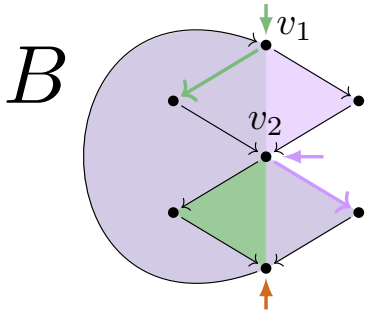
Forks $G_2 \leftarrow_{\text{elim}} G \to_{\text{elim}} G_1$ in which loop subgraphs $G{\downarrow_*^{v_2,T_2}}$, $G{\downarrow_*^{v_1,T_1}}$ of G are eliminated, can be joined with $\to_{\text{prune}}$ steps in the three situations:

(i) Root-overlap: $v_1 = v_2$, and the loops form bigger loop $G{\downarrow_*^{v_1,T_1 \cup T_2}}$. The $\to_{\text{elim}}$ steps may overlap in their loop-entry transitions, but the fork can always be joined by eliminating residual loop-entries as follows: $G_2 \to_{\text{elim}}^{=} \cdot \to_{\text{prune}}^{*} G_0 \leftarrow_{\text{prune}}^{*} \cdot \leftarrow_{\text{elim}}^{=} G_1$.

(ii) Parallel/nested: $v_1 \neq v_2$, and the cyclic parts $\circlearrowleft(G{\downarrow_*^{v_1,T_1}})$, $\circlearrowleft(G{\downarrow_*^{v_2,T_2}})$ of the loops are vertex-disjoint. Then the $\to_{\text{elim}}$ steps are independent of each other, except that the elimination of one loop may eliminate the other in the garbage collection operation. The fork can always be joined as follows: $G_2 \to_{\text{elim}}^{=} G_0 \leftarrow_{\text{elim}}^{=} G_1$.

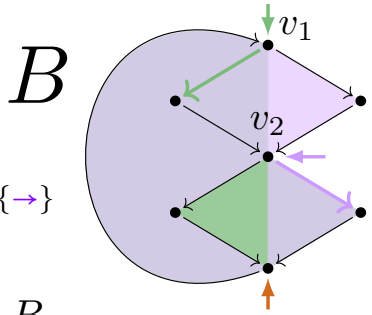
(iii) Non-root overlap: $v_1 \neq v_2$, the start-vertex connected cyclic parts $\circlearrowleft(G{\downarrow_*^{v_1,T_1}})$, and $\circlearrowleft(G{\downarrow_*^{v_2,T_2}})$ have common vertices. The two loops together form a bi-loop w.r.t. v_1, v_2. The fork can always be joined as follows: $G_2 \to_{\text{elim}}^{=} \cdot \to_{\text{prune}}^{*} G_0 \leftarrow_{\text{prune}}^{*} \cdot \leftarrow_{\text{elim}}^{=} G_1$.

Non-Root overlap /'critical pair' (example)



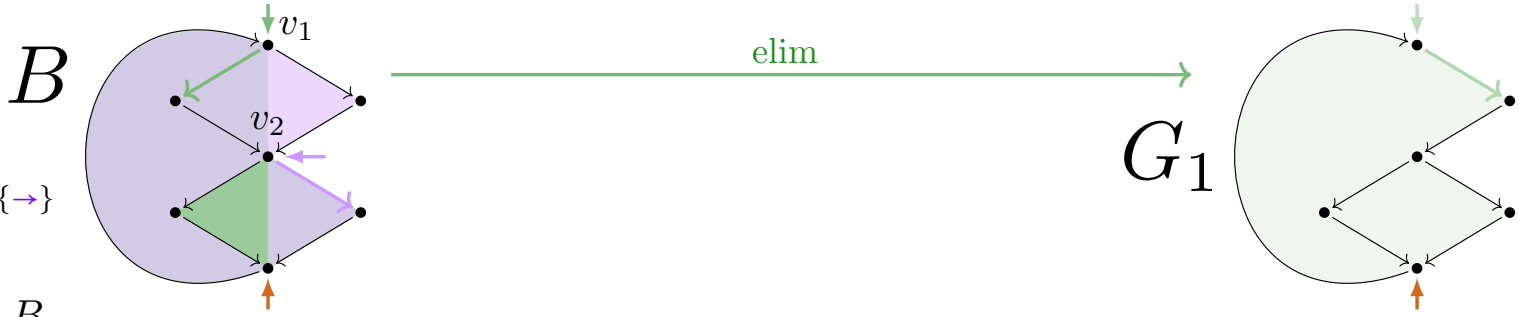
Non-Root overlap /'critical pair' (example)

bi-loop
w.r.t v_1, v_2
$B{\downarrow_*^{v_1,\{\rightarrow\}}}$ and $B{\downarrow_*^{v_2,\{\rightarrow\}}}$
are overlapping
loop subgraphs of B



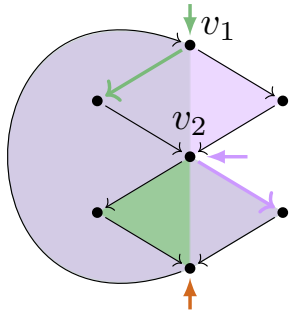
Non-Root overlap /'critical pair' (example)

bi-loop
w.r.t v_1, v_2
$B{\downarrow_*^{v_1,\{\rightarrow\}}}$ and $B{\downarrow_*^{v_2,\{\rightarrow\}}}$
are overlapping
loop subgraphs of B

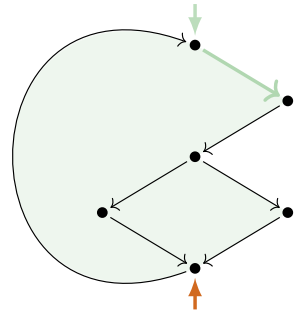


Non-Root overlap /'critical pair' (example)

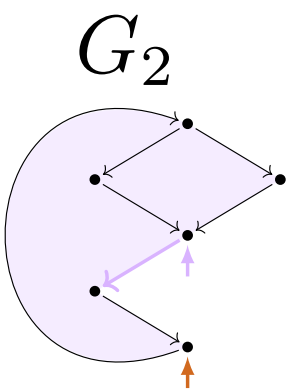
bi-loop
w.r.t v_1, v_2
$B{\downarrow_*^{v_1,\{\rightarrow\}}}$ and $B{\downarrow_*^{v_2,\{\rightarrow\}}}$
are overlapping
loop subgraphs of B



elim

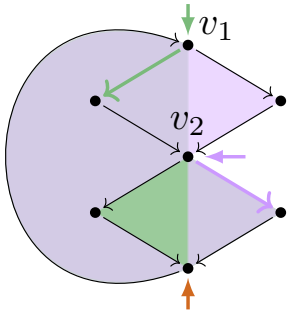


elim

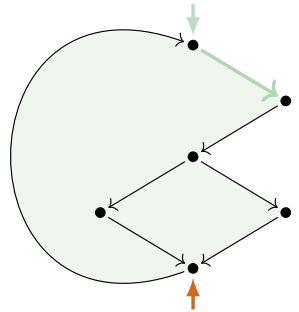


Non-Root overlap /'critical pair' (example)

bi-loop
w.r.t v_1, v_2
$B{\downarrow_*^{v_1,\{\rightarrow\}}}$ and $B{\downarrow_*^{v_2,\{\rightarrow\}}}$
are overlapping
loop subgraphs of B

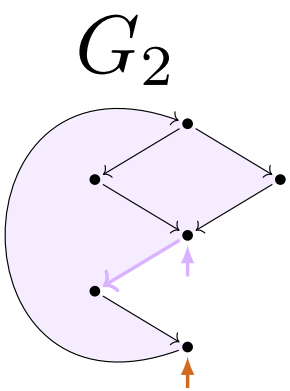


elim

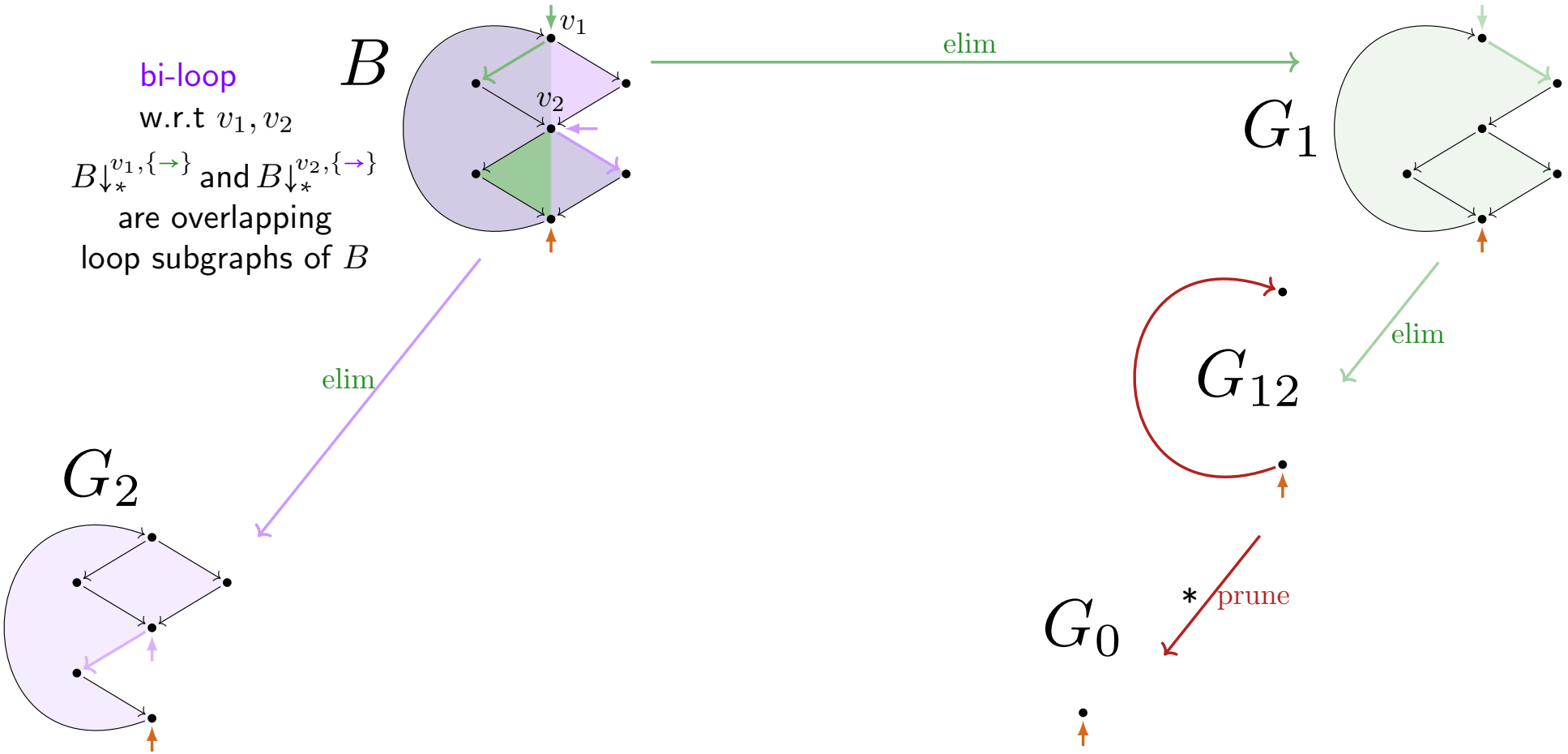


elim

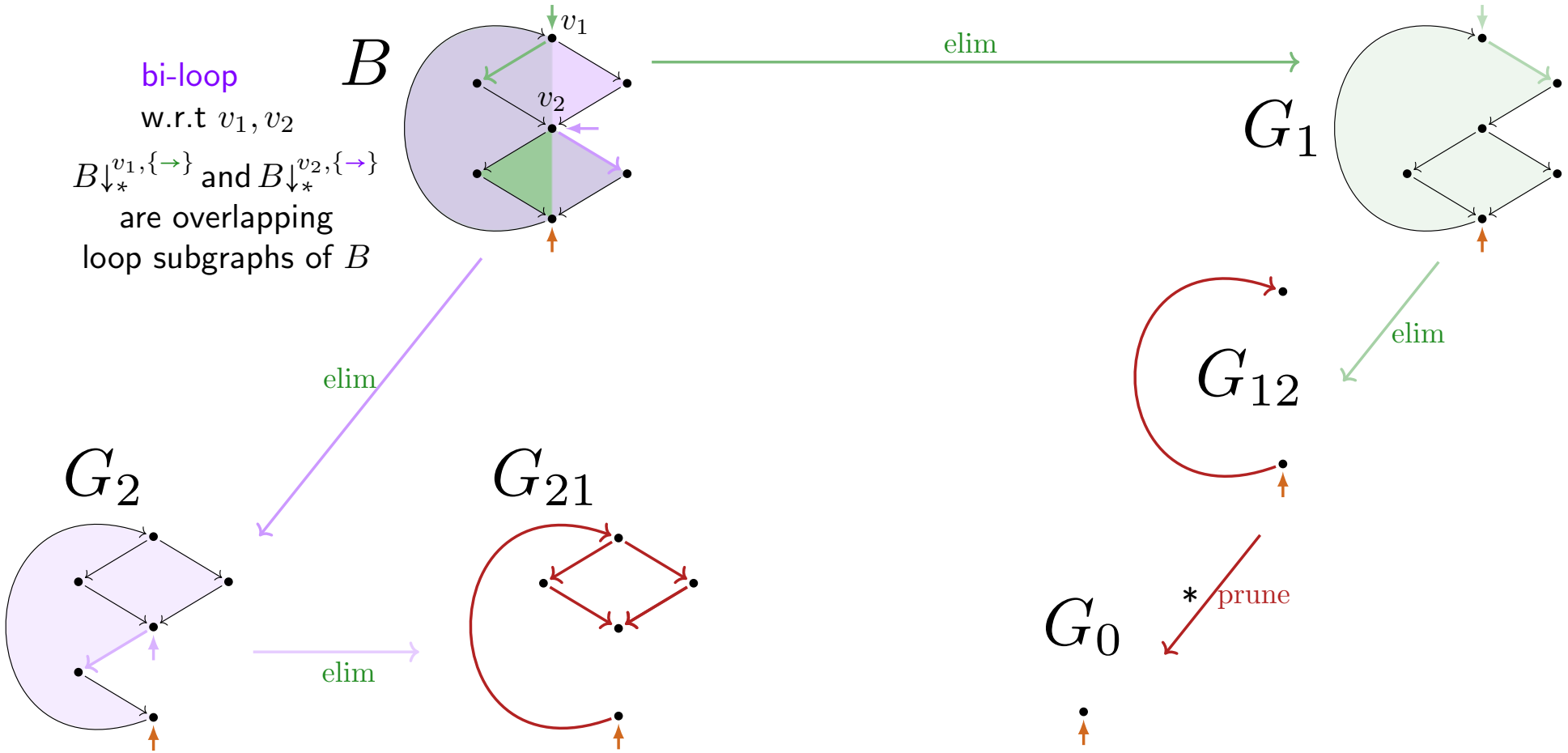
elim



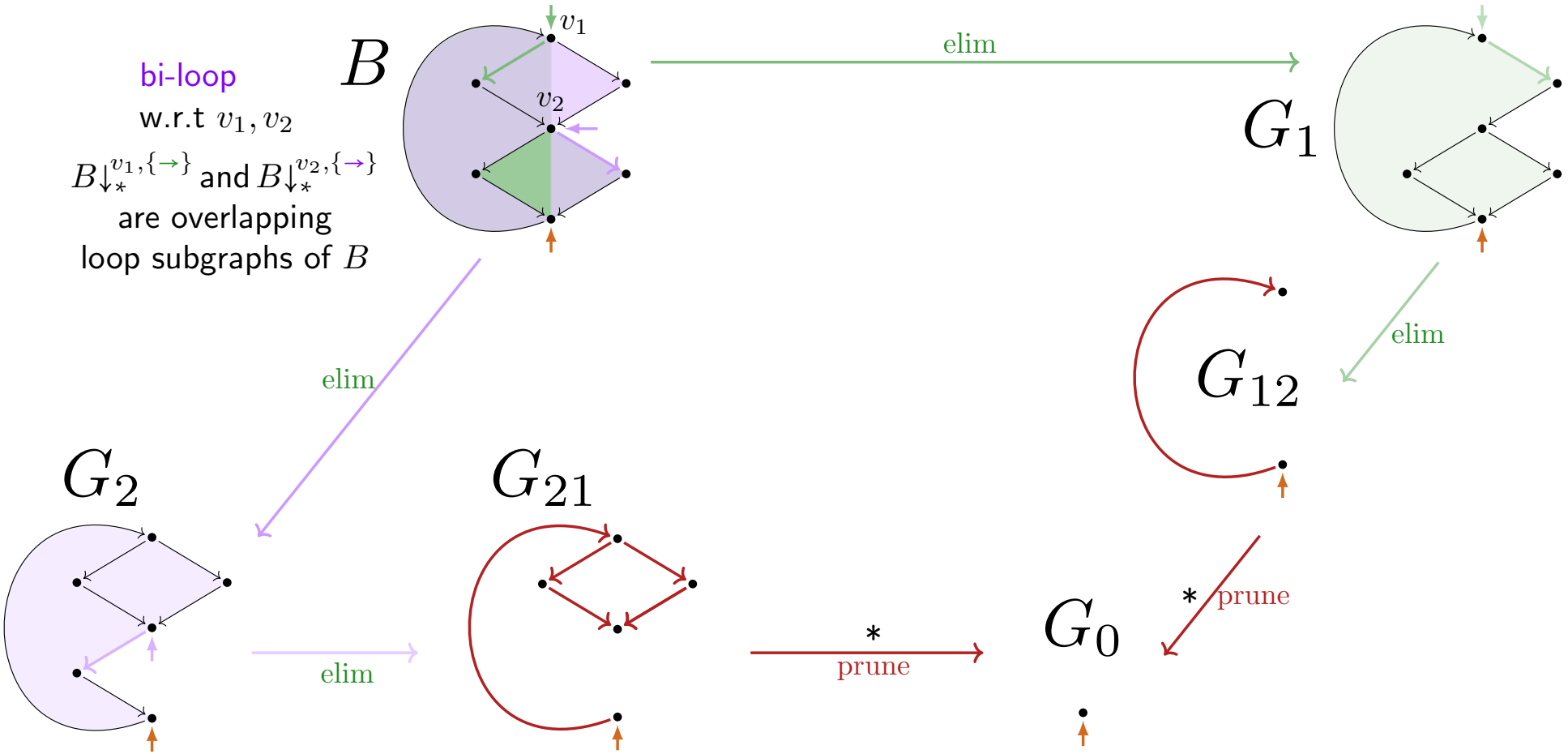
Non-Root overlap /'critical pair' (example)



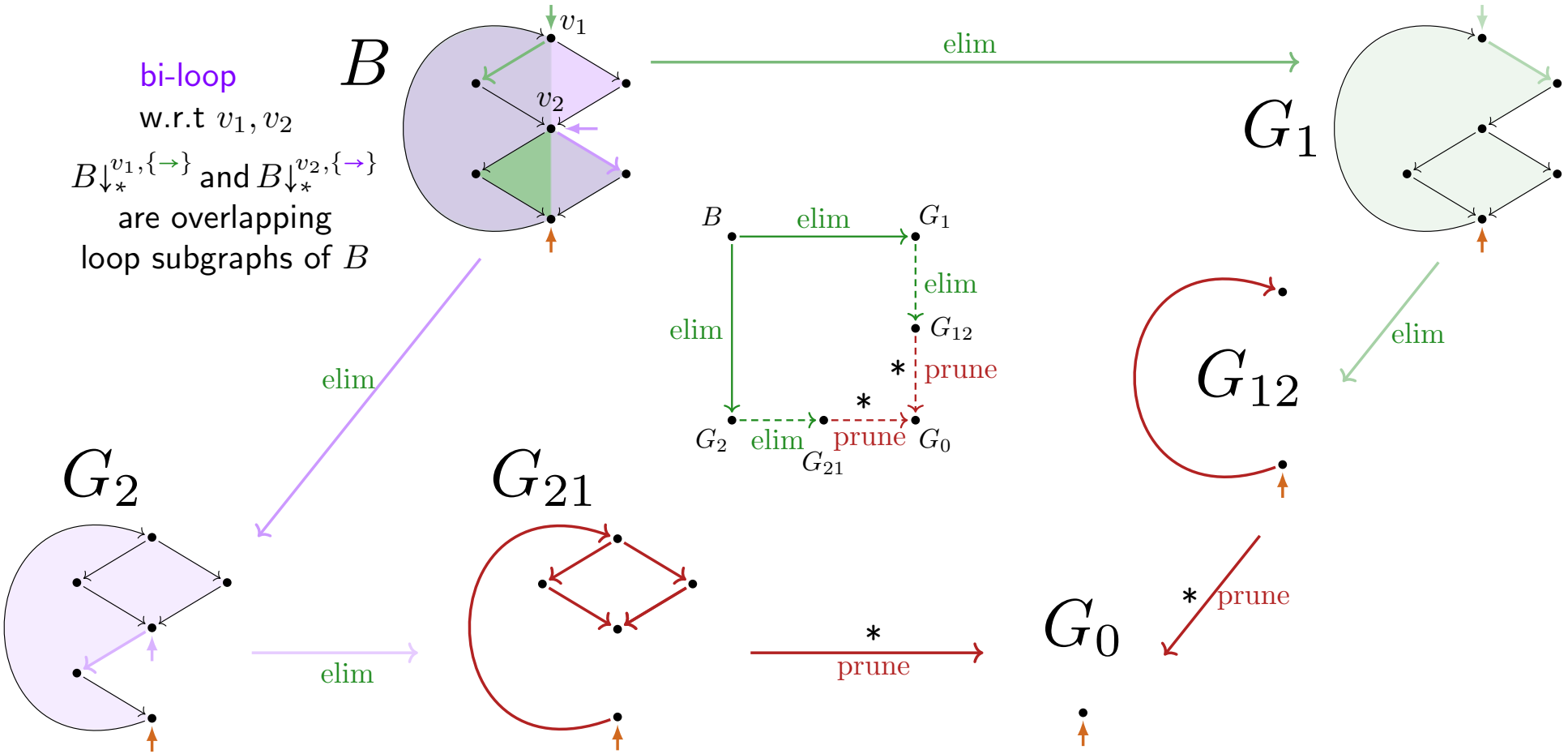
Non-Root overlap /'critical pair' (example)



Non-Root overlap /'critical pair' (example)



Non-Root overlap /'critical pair' (example)



'Critical Pair Lemma'

Lemma (joining $\to_{\text{elim}}$ forks)

Forks $G_2 \leftarrow_{\text{elim}} G \to_{\text{elim}} G_1$ in which loop subgraphs $G{\downarrow_*^{v_2,T_2}}$, $G{\downarrow_*^{v_1,T_1}}$ of G are eliminated, can be joined with $\to_{\text{prune}}$ steps in the three situations:

(i) Root-overlap: $v_1 = v_2$, and the loops form bigger loop $G{\downarrow_*^{v_1,T_1 \cup T_2}}$. The $\to_{\text{elim}}$ steps may overlap in their loop-entry transitions, but the fork can always be joined by eliminating residual loop-entries as follows: $G_2 \to_{\text{elim}}^{=} \cdot \to_{\text{prune}}^{*} G_0 \leftarrow_{\text{prune}}^{*} \cdot \leftarrow_{\text{elim}}^{=} G_1$.

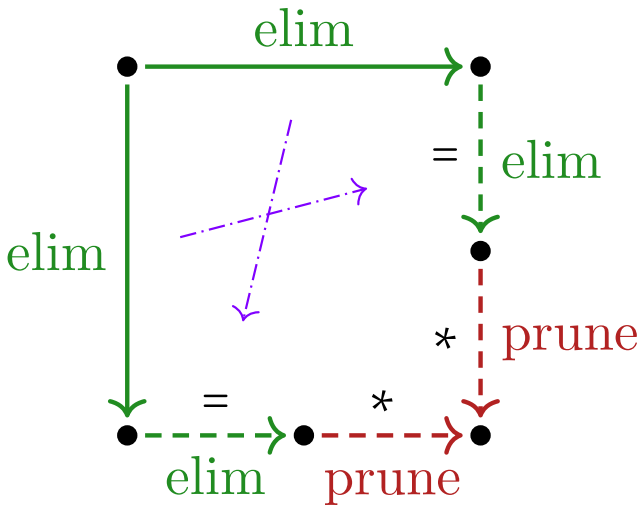
(ii) Parallel/nested: $v_1 \neq v_2$, and the cyclic parts $\circlearrowleft(G{\downarrow_*^{v_1,T_1}})$, $\circlearrowleft(G{\downarrow_*^{v_2,T_2}})$ of the loops are vertex-disjoint. Then the $\to_{\text{elim}}$ steps are independent of each other, except that the elimination of one loop may eliminate the other in the garbage collection operation. The fork can always be joined as follows: $G_2 \to_{\text{elim}}^{=} G_0 \leftarrow_{\text{elim}}^{=} G_1$.

(iii) Non-root overlap: $v_1 \neq v_2$, the start-vertex connected cyclic parts $\circlearrowleft(G{\downarrow_*^{v_1,T_1}})$, and $\circlearrowleft(G{\downarrow_*^{v_2,T_2}})$ have common vertices. The two loops together form a bi-loop w.r.t. v_1, v_2. The fork can always be joined as follows: $G_2 \to_{\text{elim}}^{=} \cdot \to_{\text{prune}}^{*} G_0 \leftarrow_{\text{prune}}^{*} \cdot \leftarrow_{\text{elim}}^{=} G_1$.

Elementary diagrams

Lemma

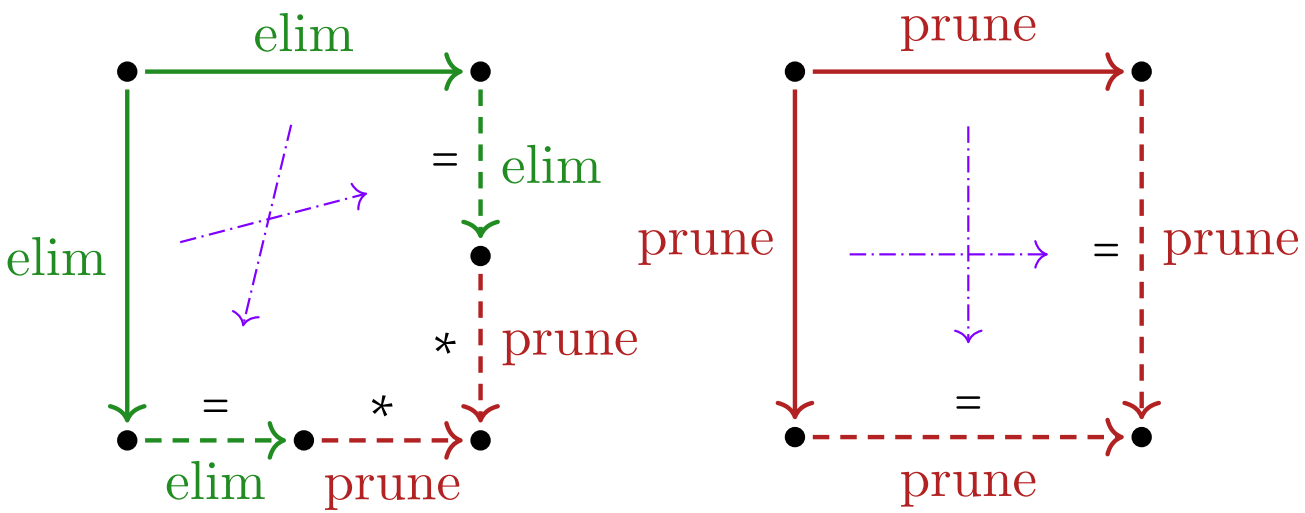
$\to_{\mathrm{ep}}$ is (locally) decreasing with respect to $\{\to_\ell\}_{\ell \in \{\mathrm{elim},\mathrm{prune}\}}$ for precedence $\mathrm{prune} < \mathrm{elim}$.



Elementary diagrams

Lemma

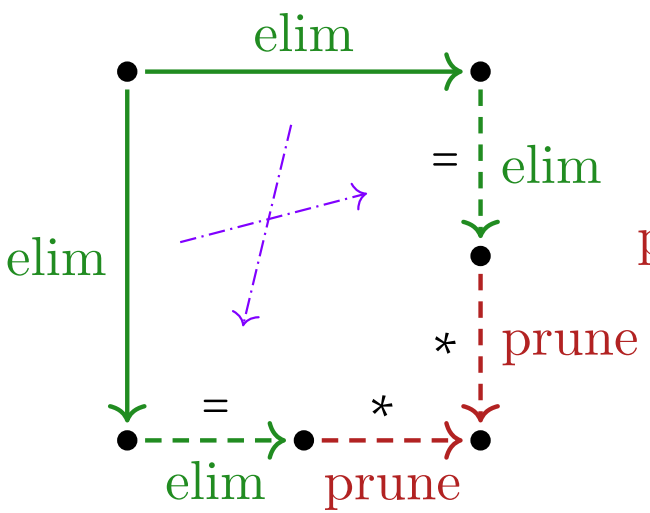
$\to_{\mathrm{ep}}$ is (locally) decreasing with respect to $\{\to_\ell\}_{\ell \in \{\mathrm{elim},\mathrm{prune}\}}$ for precedence $\mathrm{prune} < \mathrm{elim}$.

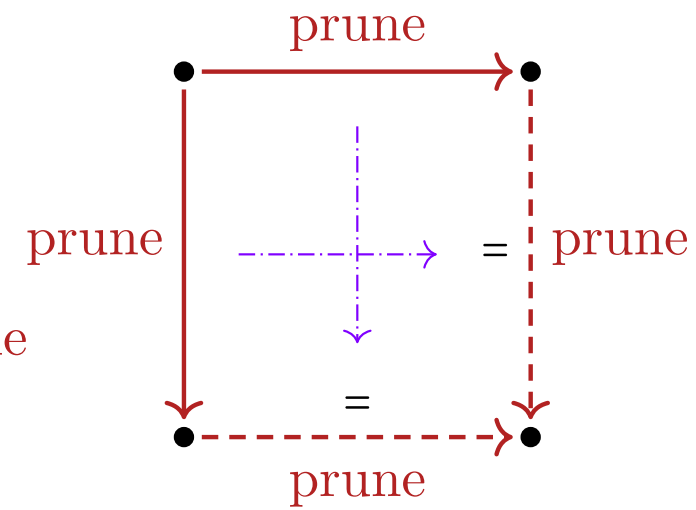


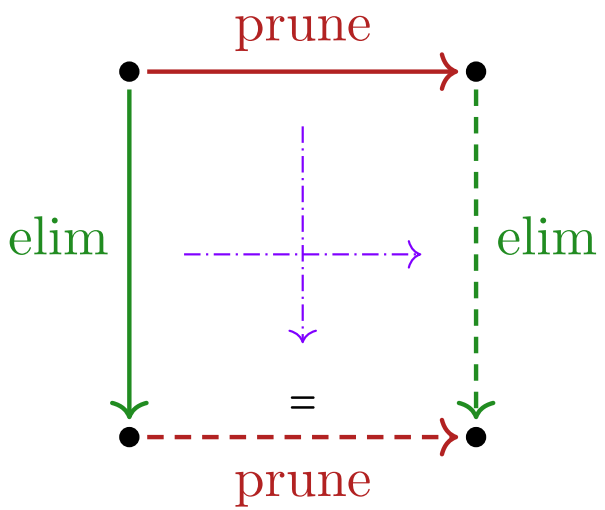
Elementary diagrams

Lemma

$\to_{\mathrm{ep}}$ is (locally) decreasing with respect to $\{\to_\ell\}_{\ell \in \{\mathrm{elim},\mathrm{prune}\}}$ for precedence $\mathrm{prune} < \mathrm{elim}$.



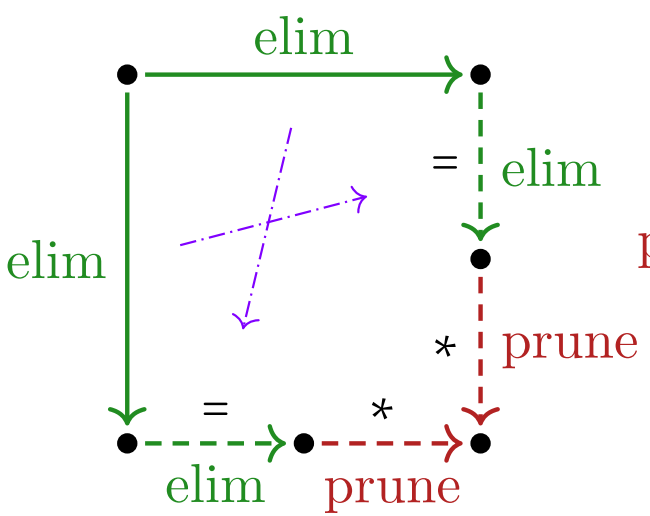


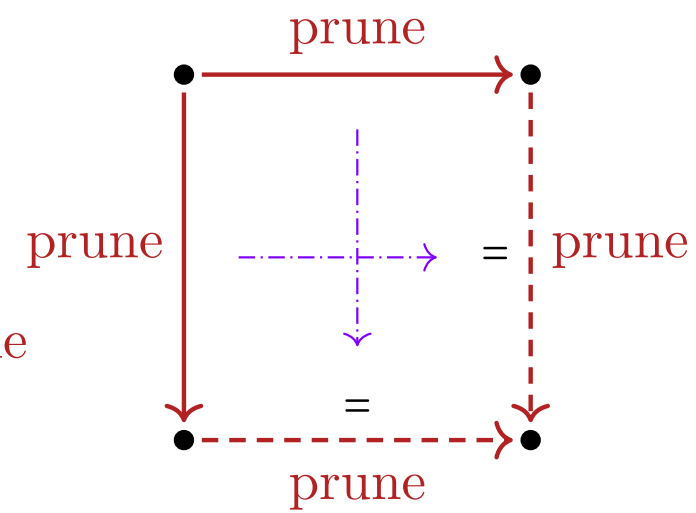


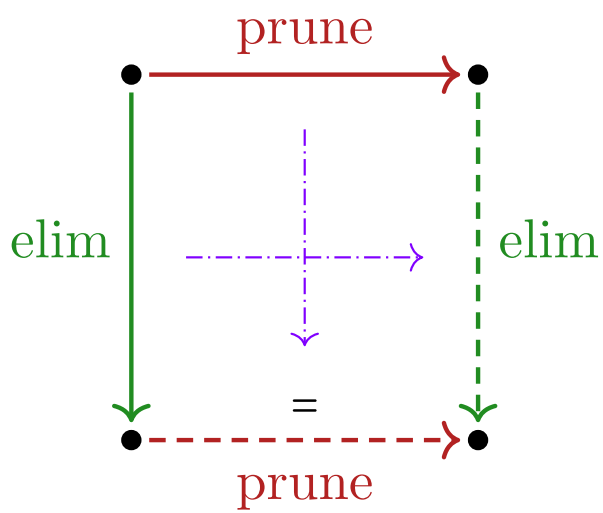
Elementary diagrams

Lemma

$\to_{\mathrm{ep}}$ is (locally) decreasing with respect to $\{\to_\ell\}_{\ell \in \{\mathrm{elim},\mathrm{prune}\}}$ for precedence $\mathrm{prune} < \mathrm{elim}$.







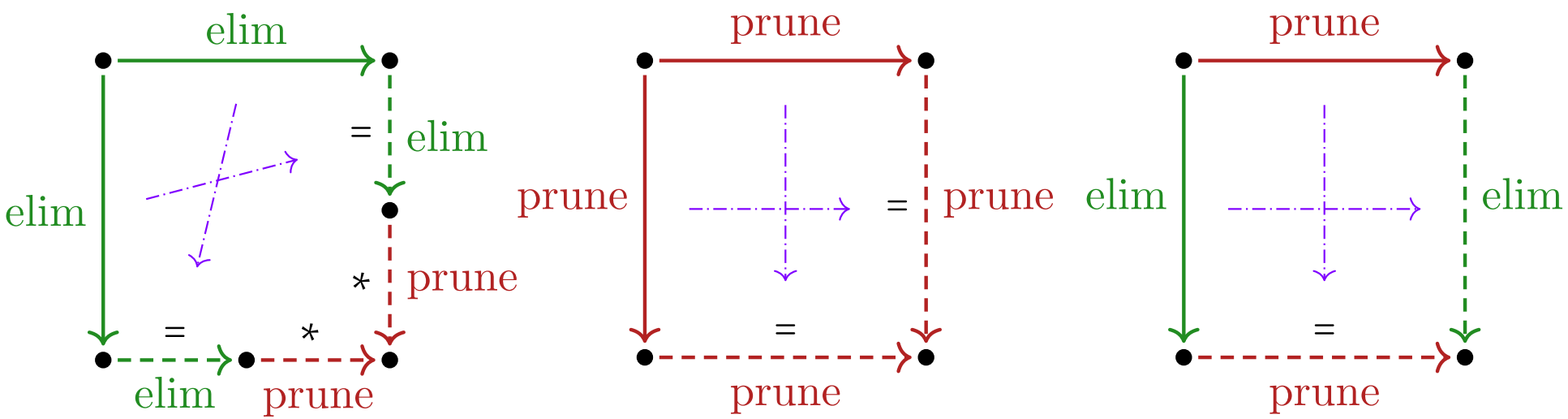
Theorem

$\to_{\mathrm{ep}} = \to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ is confluent.

Elementary diagrams

Lemma

$\to_{\mathrm{ep}}$ is (locally) decreasing with respect to $\{\to_\ell\}_{\ell \in \{\mathrm{elim},\mathrm{prune}\}}$ for precedence $\mathrm{prune} < \mathrm{elim}$.



Theorem

$\to_{\mathrm{ep}} = \to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ is confluent.

Proof.

By the Decreasing Diagrams Theorem (van Oostrom ~'94/de Bruijn). □

Overview

- Loop elimination $\to_{\mathrm{elim}}$ in process graphs
  - Milner's process interpretation P of regular expressions
  - Loop existence and elimination property LEE
    LEE $\hat{=}$ interpretations of $(*/\not{1})$ reg. expr's
  - $\to_{\mathrm{elim}}$ is not confluent
- Completion of loop elimination $\to_{\mathrm{elim}}$ with pruning steps $\to_{\mathrm{prune}}$
  - 'critical pair' lemma for 'bi-loop' elimination steps
  - use of decreasing diagrams
- Application 1: LEE is decidable in polynomial time
- Application 2: Layered LEE = LEE
- Further questions

Efficient decidability of LEE

Corollary (of confluence of $\to_{\mathrm{ep}}$)

For every finite process graph G, TFAE:

(i) LEE(G): there is an $\to_{\mathrm{elim}}$ normal form of G without infinite trace.

(ii) The unique $\to_{\mathrm{ep}}$ normal form of G does not enable an infinite trace.

(iii) Every $\to_{\mathrm{elim}}$ normal form of G does not enable an infinite trace.

Efficient decidability of LEE

Corollary (of confluence of $\to_{\mathrm{ep}}$)

For every finite process graph G, TFAE:

(i) LEE(G): there is an $\to_{\mathrm{elim}}$ normal form of G without infinite trace.

(ii) The unique $\to_{\mathrm{ep}}$ normal form of G does not enable an infinite trace.

(iii) Every $\to_{\mathrm{elim}}$ normal form of G does not enable an infinite trace.

Theorem

For finite graphs G, LEE is decidable in $\leq \mathrm{O}(|G|^3) \subseteq \mathrm{PTIME}$.

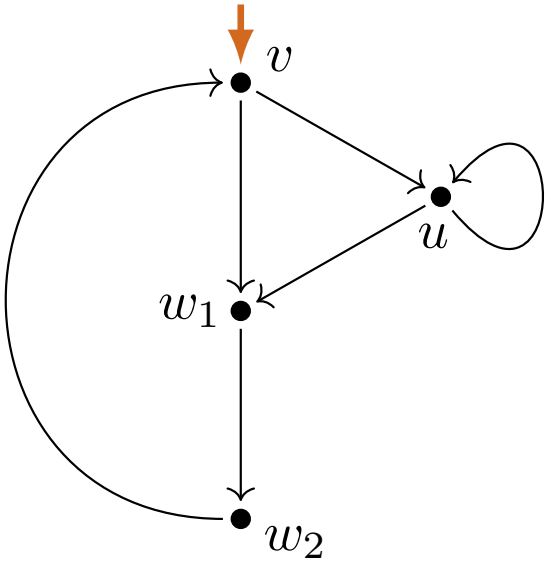
Proof

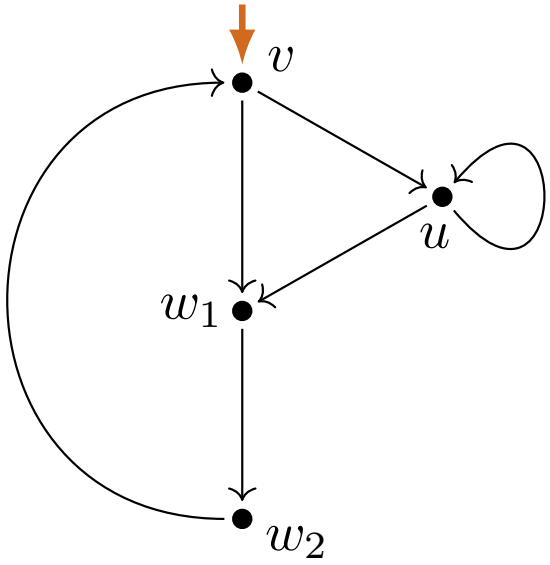
Due to (i) $\Rightarrow$ (iii) in the Corollary, to decide LEE(G) it suffices to construct any $\to_{\mathrm{elim}}$ rewrite sequence $G \to^*_{\mathrm{elim}} G_0 \not\to_{\mathrm{elim}}$ to normal form G_0, and check if $\neg\infty(G_0)$ holds. That is possible in $\leq \mathrm{O}(|G|^3)$ time. $\square$

Overview

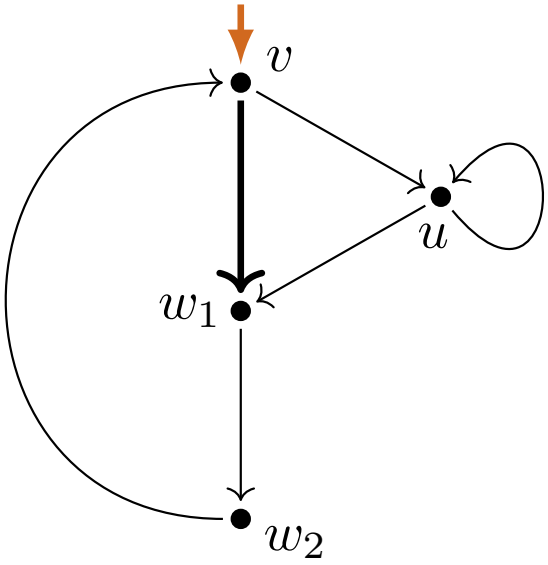
- Loop elimination $\to_{\mathrm{elim}}$ in process graphs
  - Milner's process interpretation P of regular expressions
  - Loop existence and elimination property LEE
    LEE $\hat{=}$ interpretations of $(*/\not{1})$ reg. expr's
  - $\to_{\mathrm{elim}}$ is not confluent
- Completion of loop elimination $\to_{\mathrm{elim}}$ with pruning steps $\to_{\mathrm{prune}}$
  - 'critical pair' lemma for 'bi-loop' elimination steps
  - use of decreasing diagrams
- Application 1: LEE is decidable in polynomial time
- Application 2: Layered LEE = LEE
- Further questions

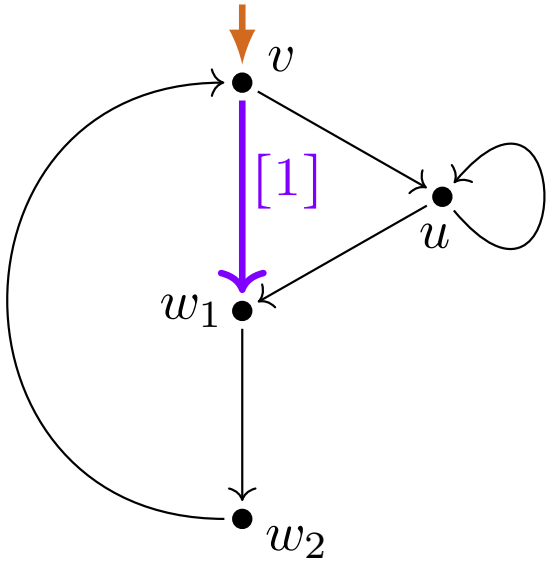
Not Layered / Layered Loop Elimination



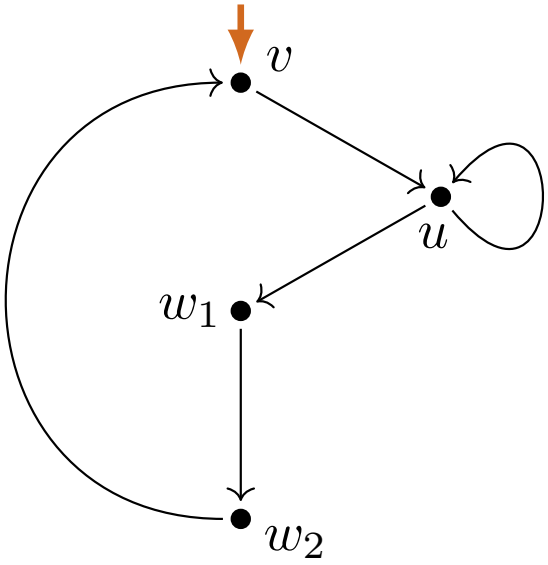


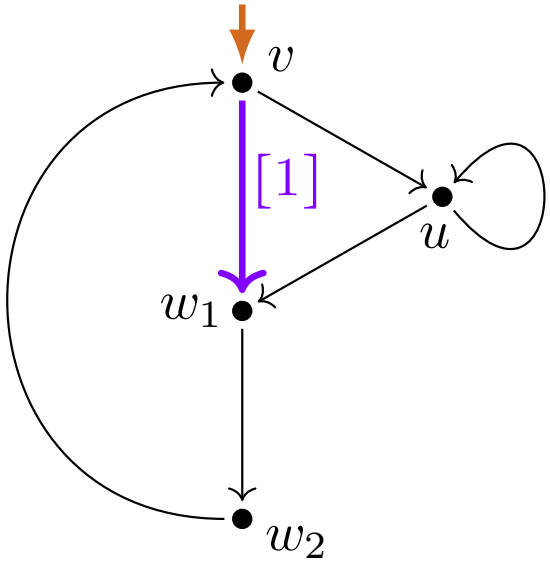
Not Layered / Layered Loop Elimination



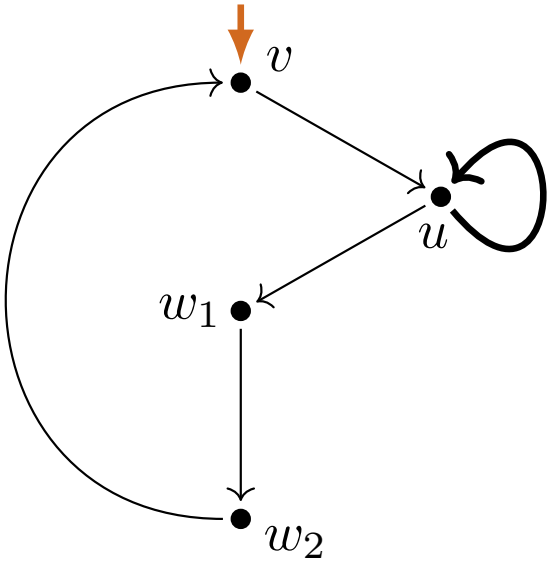


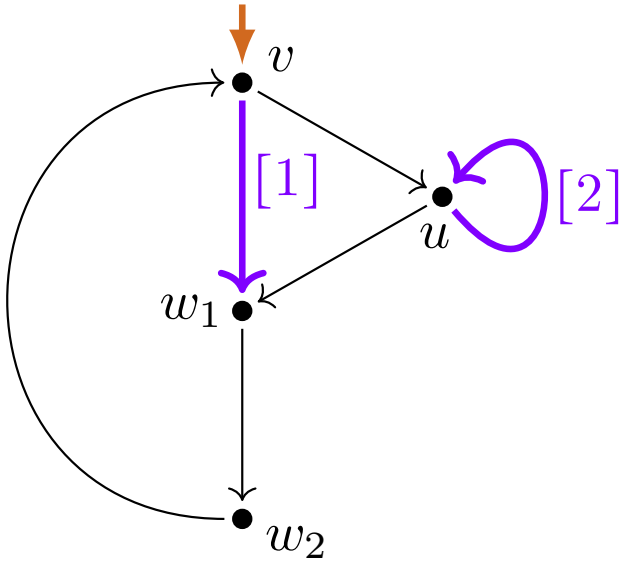
Not Layered / Layered Loop Elimination



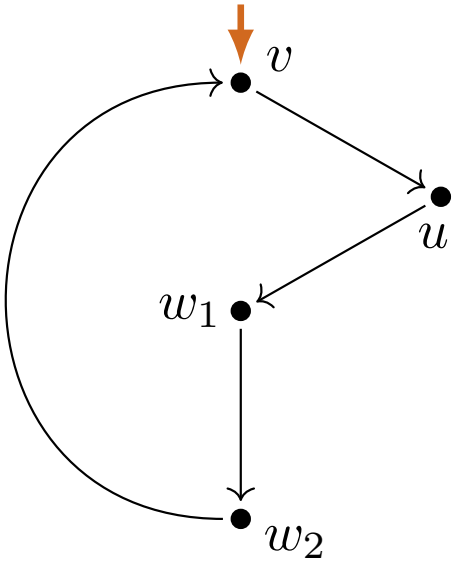


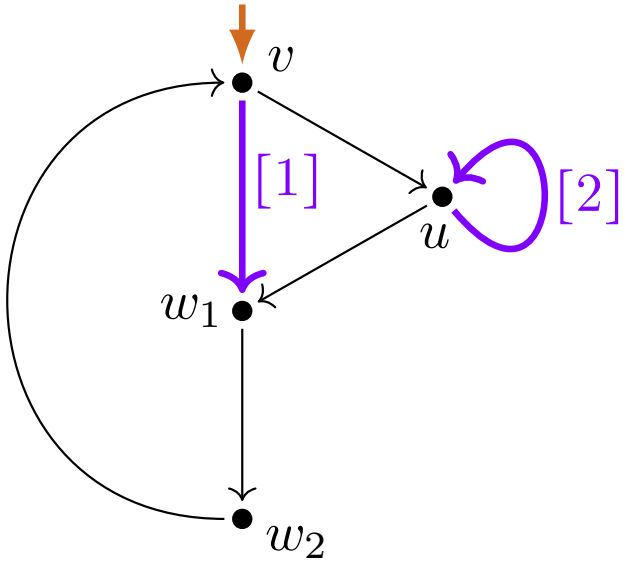
Not Layered / Layered Loop Elimination



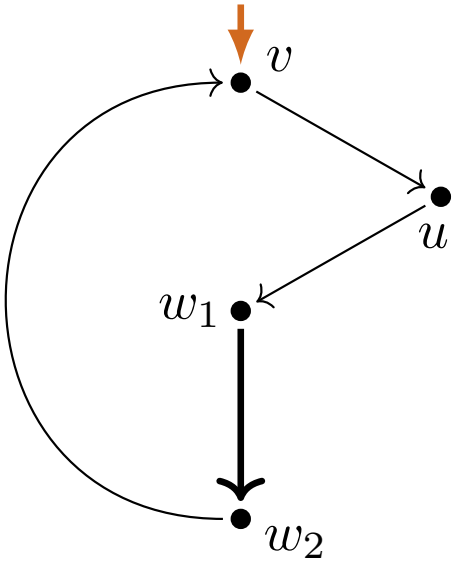


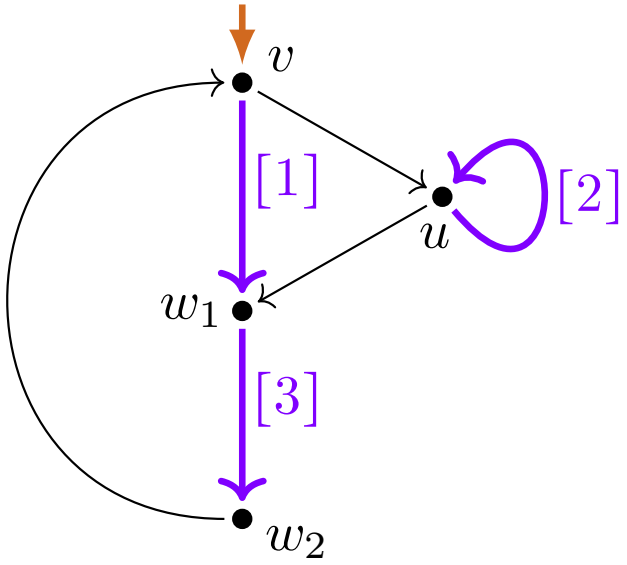
Not Layered / Layered Loop Elimination



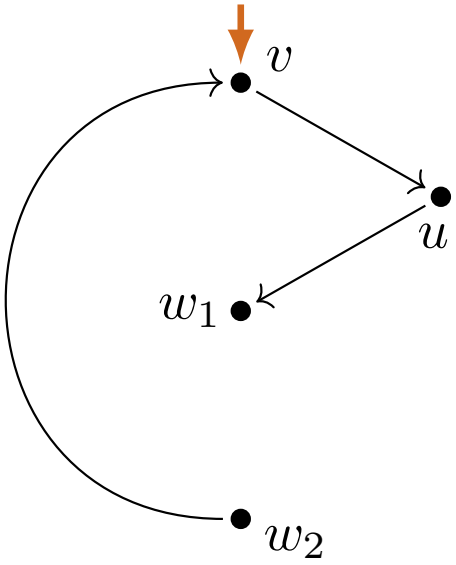


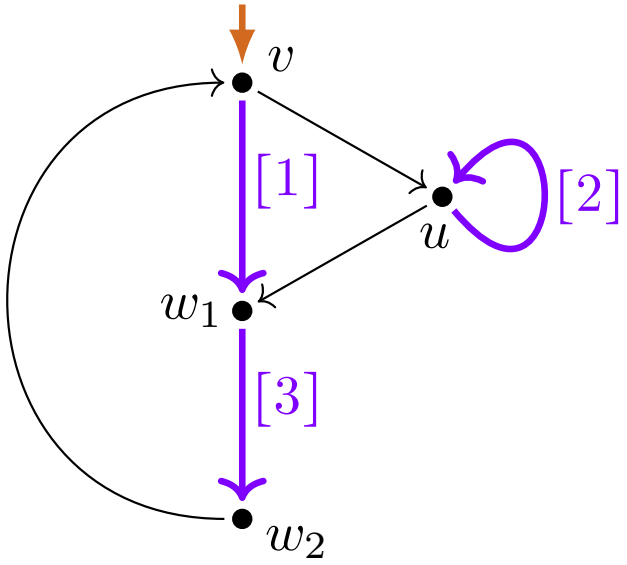
Not Layered / Layered Loop Elimination



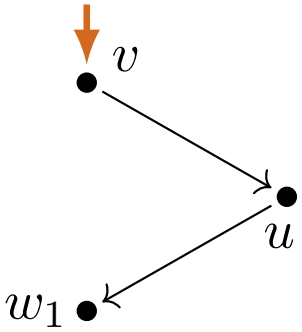


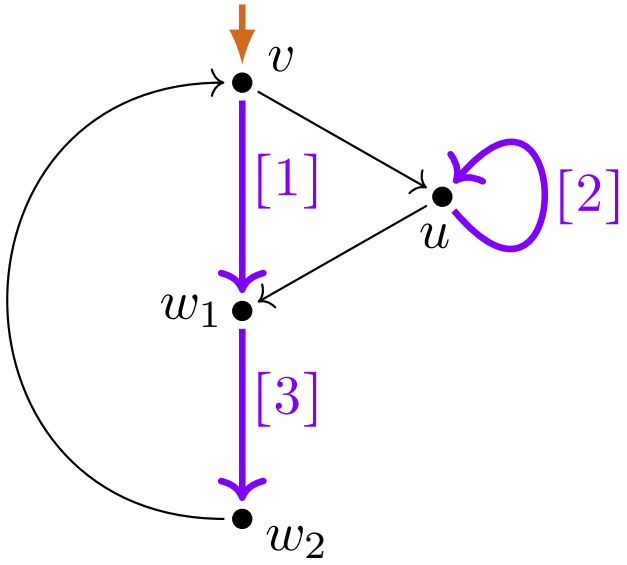
Not Layered / Layered Loop Elimination



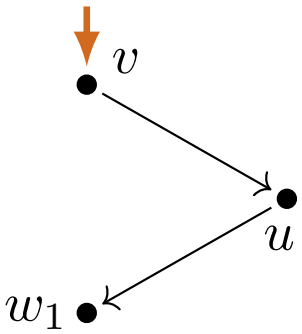


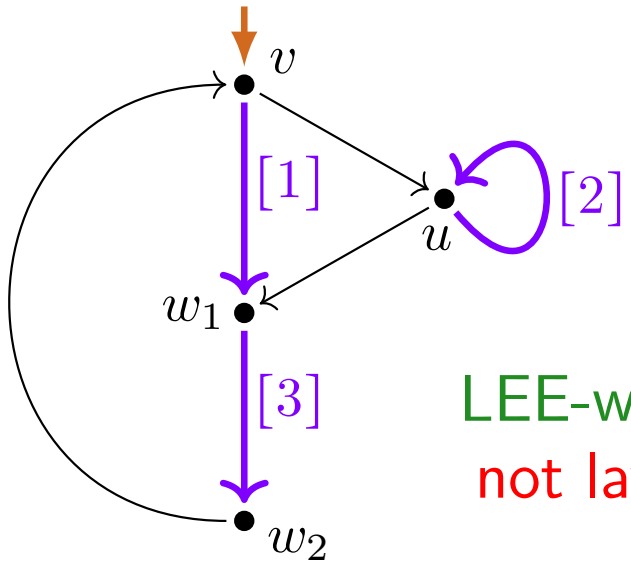
Not Layered / Layered Loop Elimination





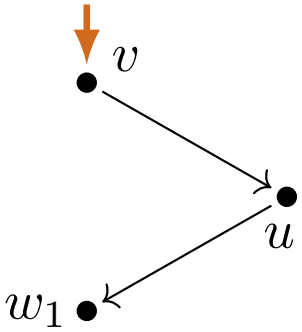
Not Layered / Layered Loop Elimination

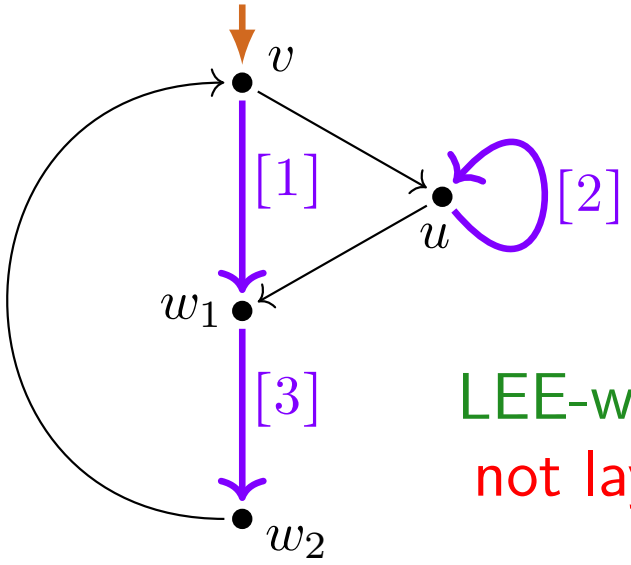




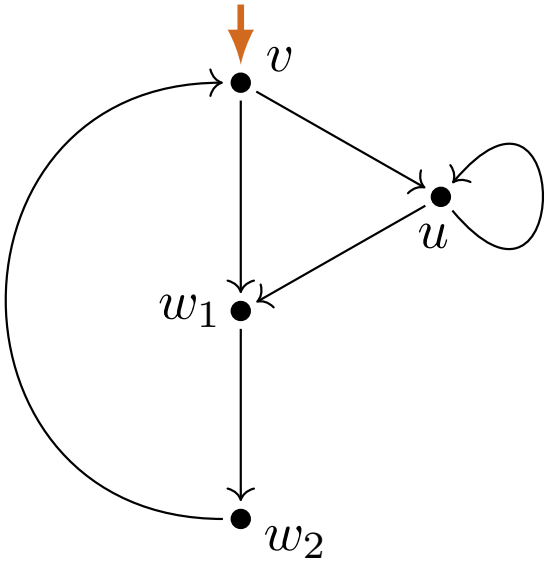
LEE-witness
not layered

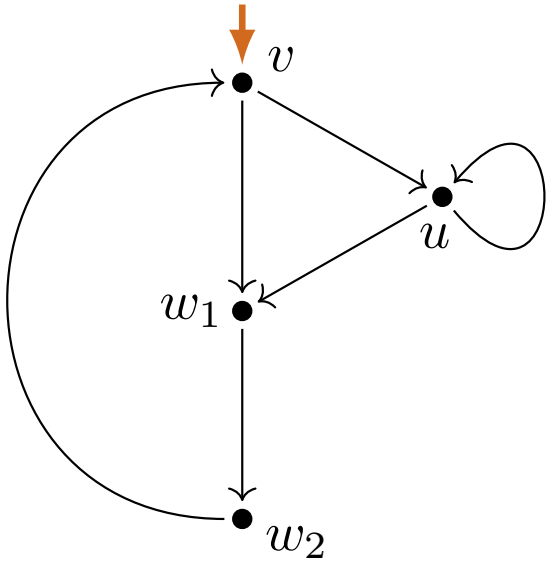
Not Layered / Layered Loop Elimination



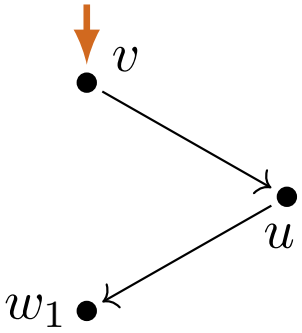


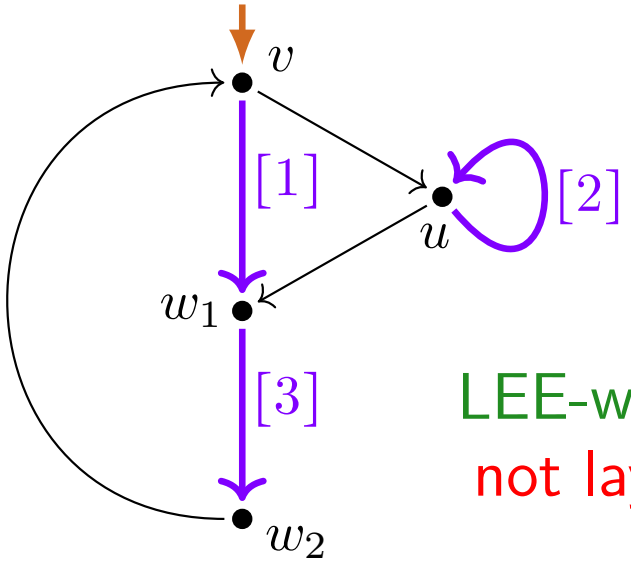
LEE-witness
not layered



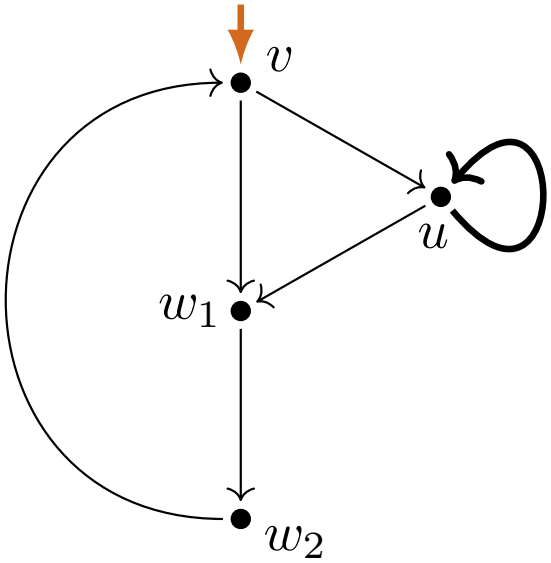


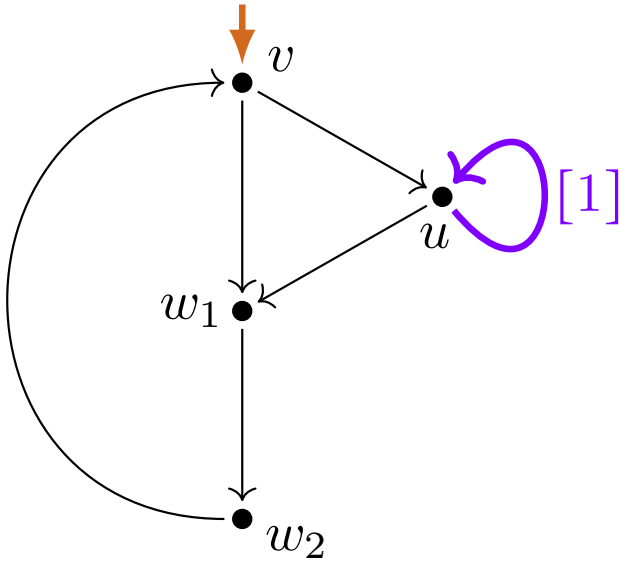
Not Layered / Layered Loop Elimination



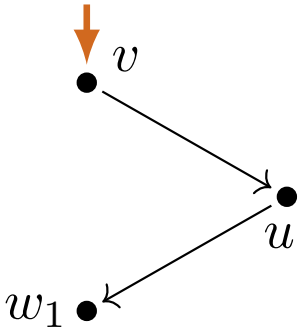


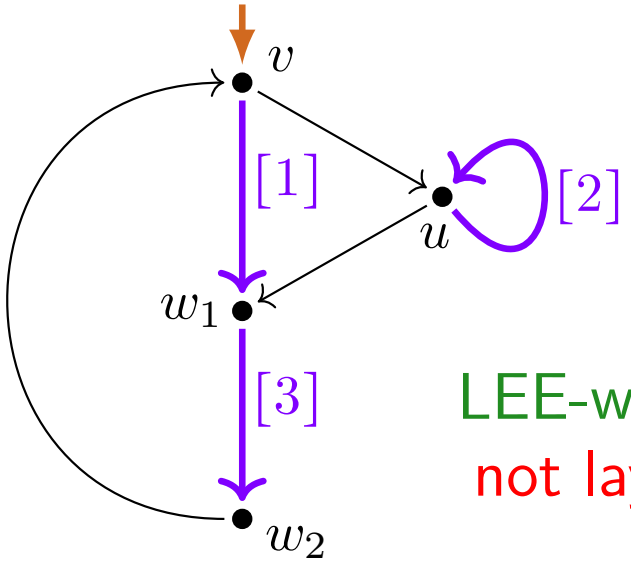
LEE-witness
not layered



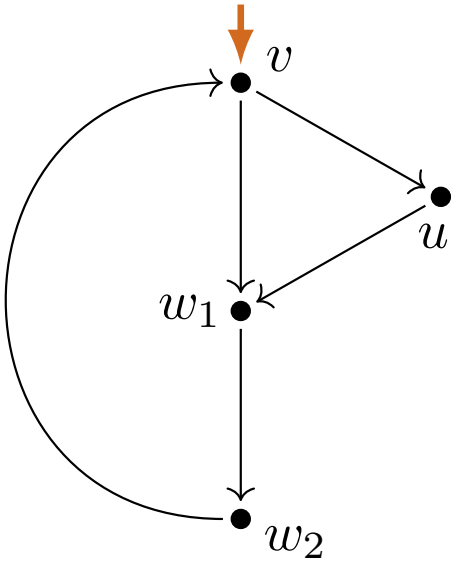


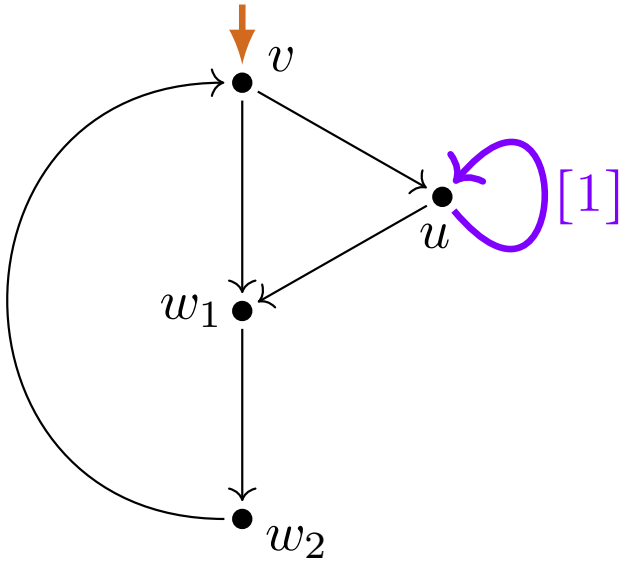
Not Layered / Layered Loop Elimination



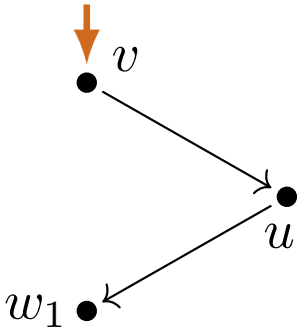


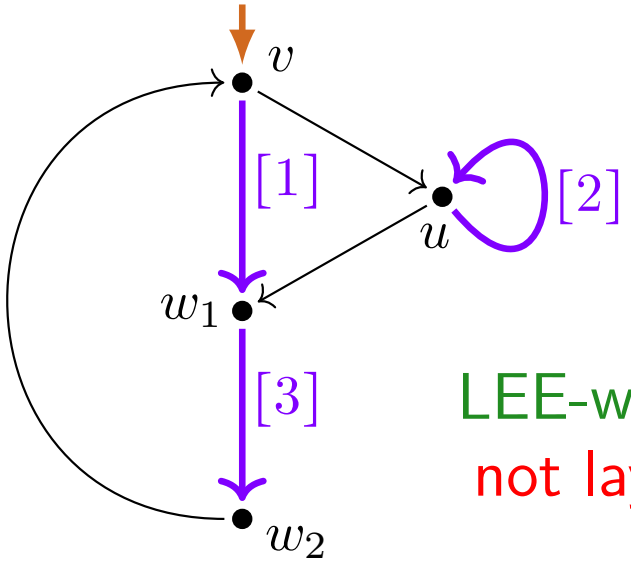
LEE-witness
not layered



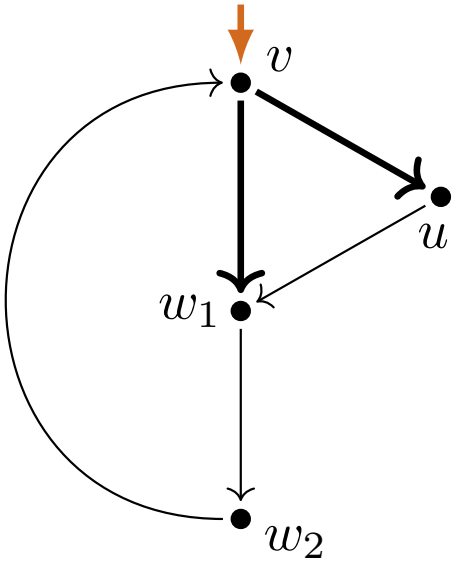


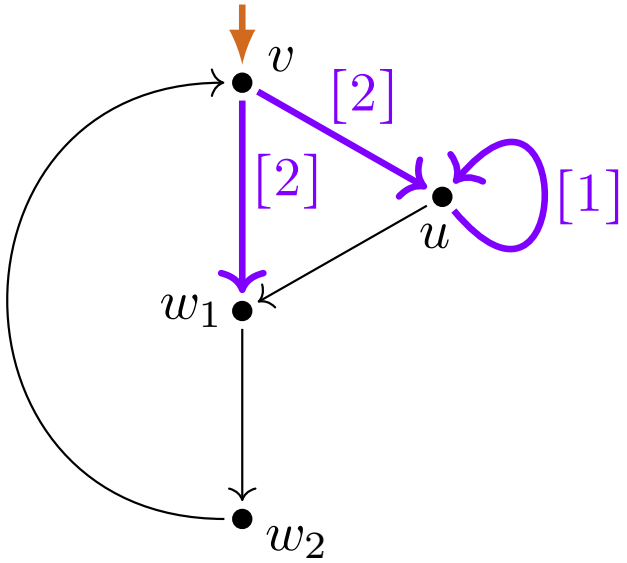
Not Layered / Layered Loop Elimination



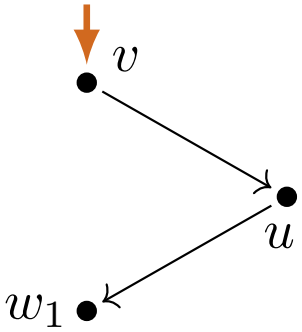


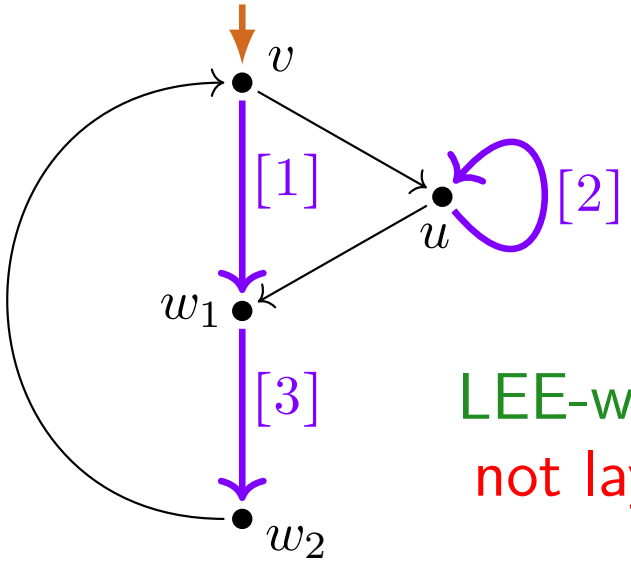
LEE-witness
not layered



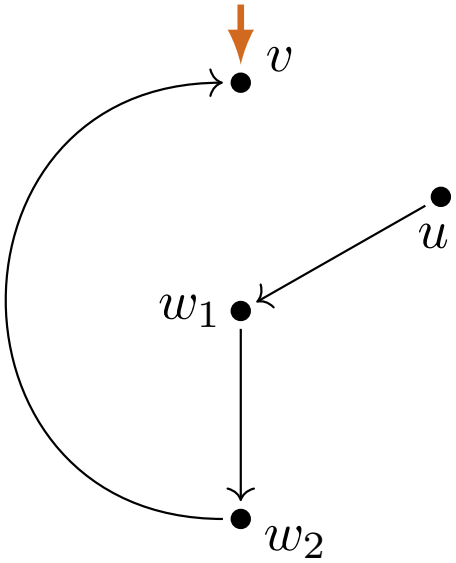


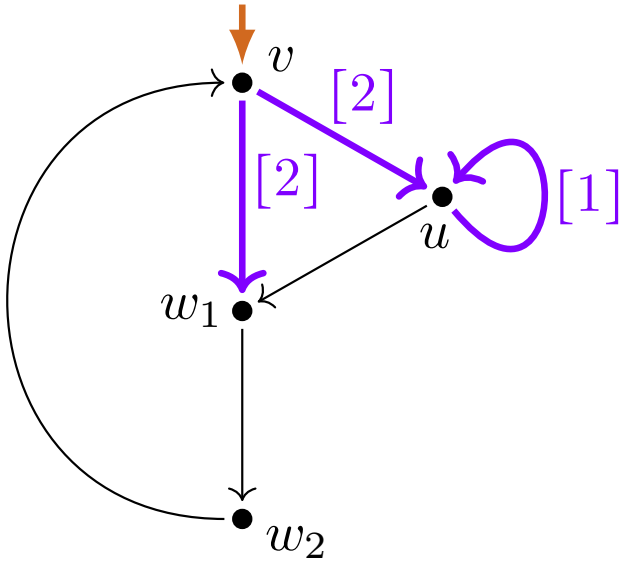
Not Layered / Layered Loop Elimination



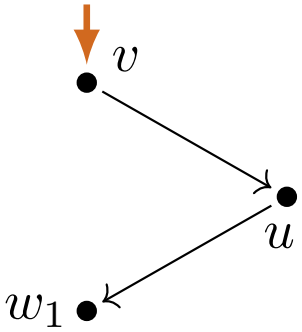


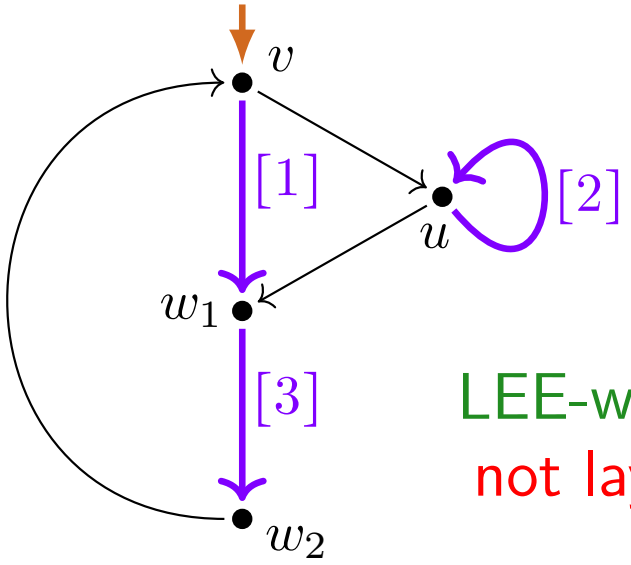
LEE-witness
not layered



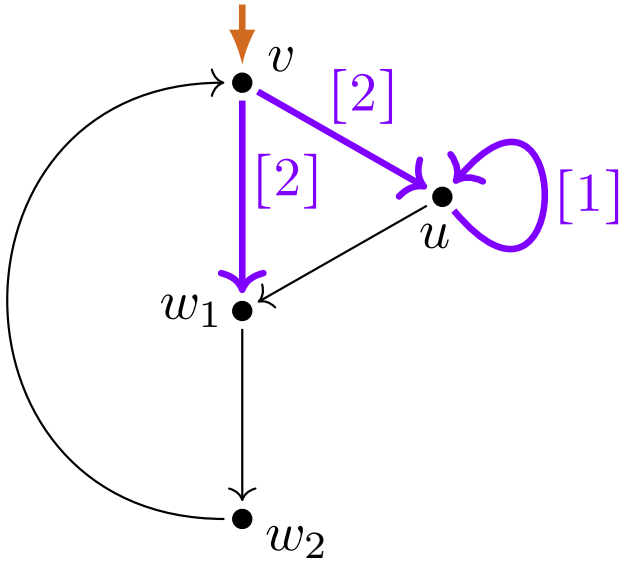


Not Layered / Layered Loop Elimination

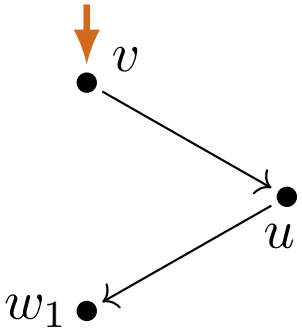


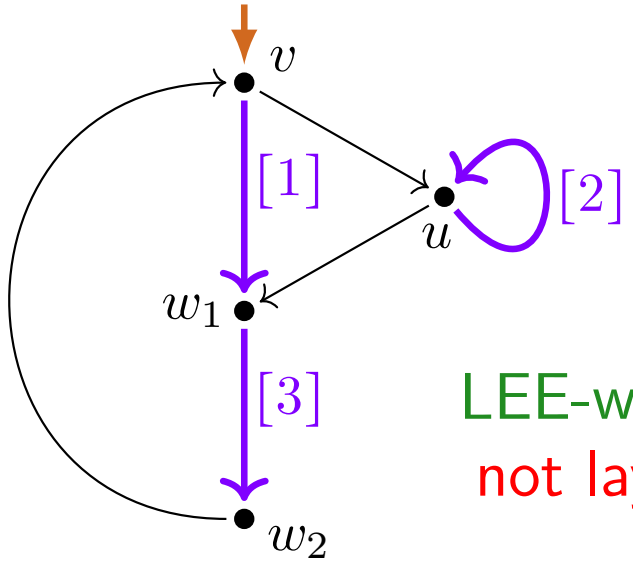


LEE-witness
not layered

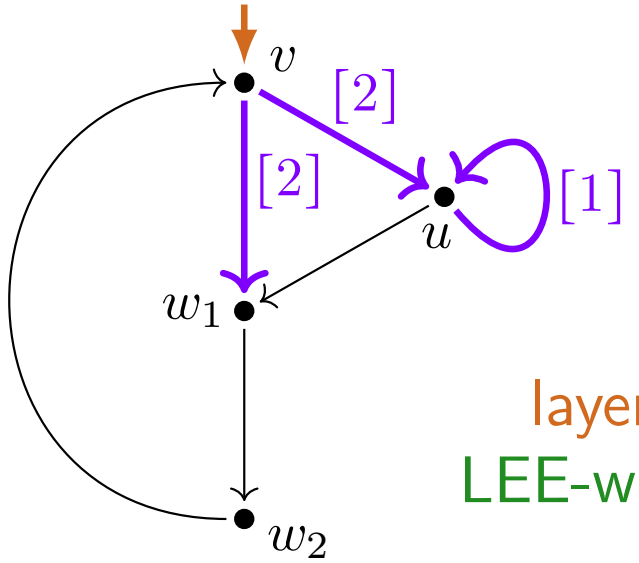


Not Layered / Layered Loop Elimination



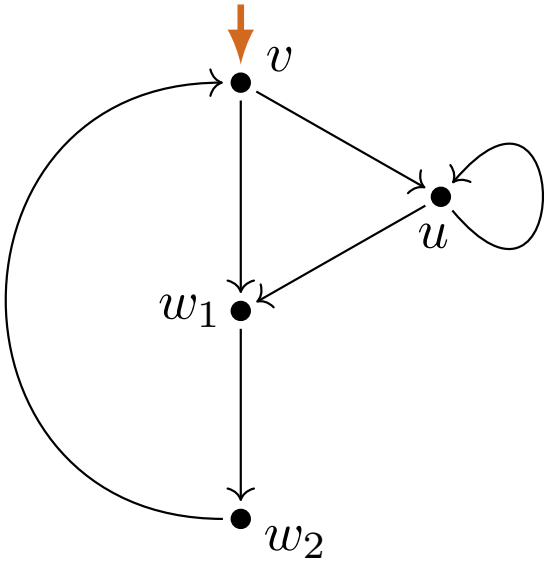


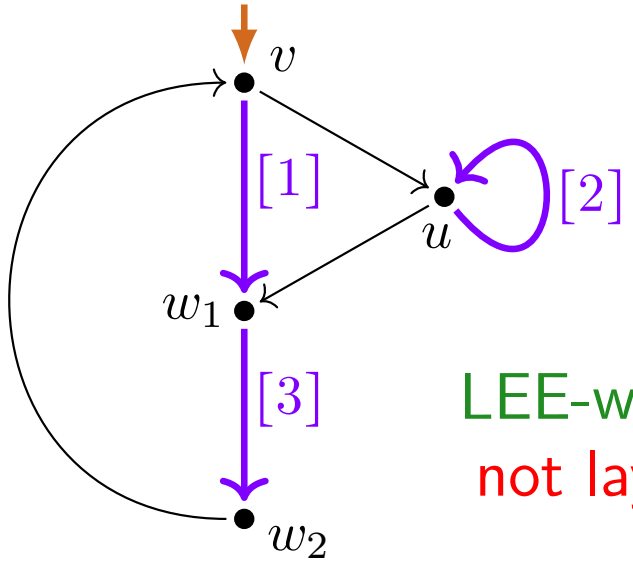
LEE-witness
not layered



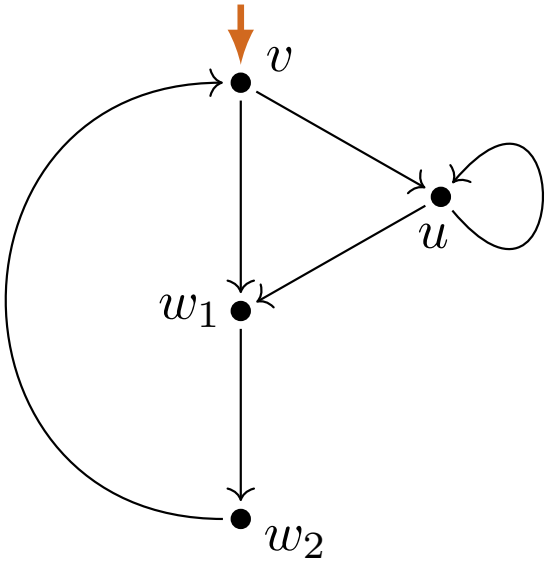
layered
LEE-witness

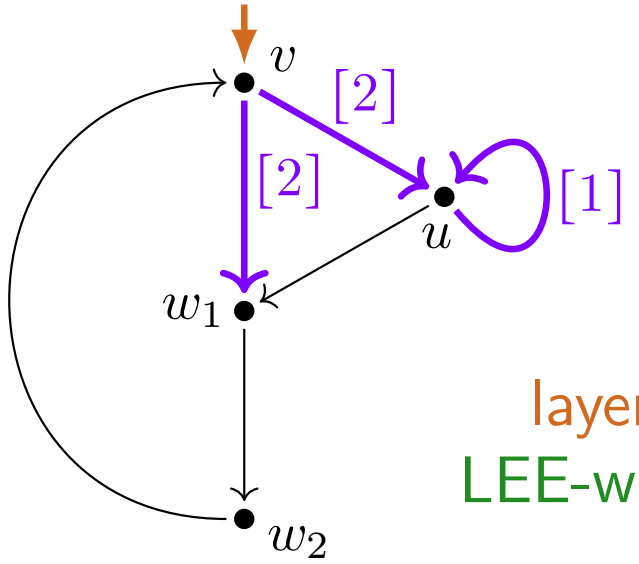
Not Layered / Layered Loop Elimination





LEE-witness
not layered





layered
LEE-witness

Layered Loop Elimination Condition LLEE

Layered loop elimination is defined on vertex-marked graphs $^{\circ}G$ in order to:

- mark vertices in the loop bodies of eliminated loops,
- permit the elimination of loop subcharts only at unmarked vertices.

Definition

The Layered Loop Existence & Elimination Condition LLEE for graphs G:

$$\mathrm{LLEE}(G) \;:\Longleftrightarrow\; \exists\, {}^{\bullet}G_0 \Big(\big({}^{\circ}G \overset{\circ}{\rightarrow}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_0 \overset{\circ}{\not\rightarrow}_{\mathrm{elim}} \big) \;\wedge\; \neg\infty\big({}^{\bullet}G_0\big) \Big) .$$

Layered Loop Elimination Condition LLEE

Layered loop elimination is defined on vertex-marked graphs $^{\circ}G$ in order to:

- mark vertices in the loop bodies of eliminated loops,
- permit the elimination of loop subcharts only at unmarked vertices.

Definition

The Layered Loop Existence & Elimination Condition LLEE for graphs G:

$$\mathrm{LLEE}(G) \;:\Longleftrightarrow\; \exists\, {}^{\bullet}G_0 \Big(\big({}^{\circ}G \overset{\circ}{\to}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_0 \overset{\circ}{\not\to}_{\mathrm{elim}} \big) \;\wedge\; \neg\infty({}^{\bullet}G_0) \Big) .$$

$${}^{\bullet}G_1 \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G'_1 \;\Longrightarrow\; |{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} |{}^{\bullet}G'_1|_{\circ}$$

Layered Loop Elimination Condition LLEE

Layered loop elimination is defined on vertex-marked graphs $^{\circ}G$ in order to:

- mark vertices in the loop bodies of eliminated loops,
- permit the elimination of loop subcharts only at unmarked vertices.

Definition

The Layered Loop Existence & Elimination Condition LLEE for graphs G:

$$\mathrm{LLEE}(G) \;:\Longleftrightarrow\; \exists\, {}^{\bullet}G_0 \big(\big({}^{\circ}G \xrightarrow{\circ}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_0 \not\xrightarrow{\circ}_{\mathrm{elim}} \big) \wedge \neg\infty({}^{\bullet}G_0) \big) .$$

$$ {}^{\bullet}G_1 \xrightarrow{\circ}_{\mathrm{elim}} {}^{\bullet}G_1' \implies |{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} |{}^{\bullet}G_1'|_{\circ} $$

$$ {}^{\circ}G \xrightarrow{\circ}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_1 \implies \Big[\exists\, {}^{\bullet}G_1'' \Big[\begin{matrix} {}^{\bullet}G_1 \xrightarrow{\circ}_{\mathrm{elim}} {}^{\bullet}G_1'' \\ \wedge\; |{}^{\bullet}G_1''|_{\circ} \to^{*}_{\mathrm{prune}} \cdot \leftarrow^{*}_{\mathrm{prune}} \cdot \leftarrow^{=}_{\mathrm{elim}} G_1' \end{matrix} \Big] \impliedby |{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} G_1' \Big] $$

Layered Loop Elimination Condition LLEE

Layered loop elimination is defined on vertex-marked graphs $^{\circ}G$ in order to:

- mark vertices in the loop bodies of eliminated loops,
- permit the elimination of loop subcharts only at unmarked vertices.

Definition

The Layered Loop Existence & Elimination Condition LLEE for graphs G:

$$\mathrm{LLEE}(G) \;:\Longleftrightarrow\; \exists\, {}^{\bullet}G_0 \big(\big({}^{\circ}G \overset{\circ}{\to}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_0 \overset{\circ}{\not\to}_{\mathrm{elim}} \big) \wedge \neg\infty({}^{\bullet}G_0) \big) .$$

$$ {}^{\bullet}G_1 \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_1' \implies |{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} |{}^{\bullet}G_1'|_{\circ} $$

$$ {}^{\circ}G \overset{\circ}{\to}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_1 \implies \Big[\exists\, {}^{\bullet}G_1'' \Big[\begin{array}{l} {}^{\bullet}G_1 \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_1'' \\ \wedge\; |{}^{\bullet}G_1''|_{\circ} \to^{*}_{\mathrm{prune}} \cdot \leftarrow^{*}_{\mathrm{prune}} \cdot \leftarrow^{=}_{\mathrm{elim}} G_1' \end{array} \Big] \impliedby |{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} G_1' \Big] $$

$$ \exists\, {}^{\bullet}G_1 \big[{}^{\circ}G \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_1 \wedge |{}^{\bullet}G_1|_{\circ} \leftrightarrow^{*}_{\mathrm{prune}} G_1 \big] \impliedby G \to^{*}_{\mathrm{elim}} G_1 $$

Layered Loop Elimination Condition LLEE

Layered loop elimination is defined on vertex-marked graphs ${}^{\circ}G$ in order to:

- mark vertices in the loop bodies of eliminated loops,
- permit the elimination of loop subcharts only at unmarked vertices.

Definition

The Layered Loop Existence & Elimination Condition LLEE for graphs G:

$$\mathrm{LLEE}(G) \;:\Longleftrightarrow\; \exists\, {}^{\bullet}G_0 \big(\big({}^{\circ}G \overset{\circ}{\to}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_0 \overset{\circ}{\not\to}_{\mathrm{elim}} \big) \wedge \neg\infty({}^{\bullet}G_0) \big) .$$

$$ {}^{\bullet}G_1 \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_1' \implies |{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} |{}^{\bullet}G_1'|_{\circ} $$

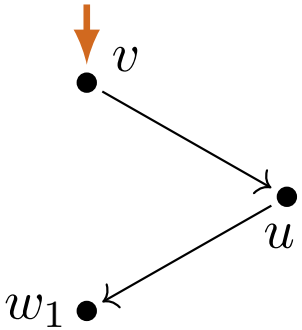
$$ {}^{\circ}G \overset{\circ}{\to}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_1 \implies \Big[\exists\, {}^{\bullet}G_1'' \Big[\begin{array}{l} {}^{\bullet}G_1 \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_1'' \\ \wedge\; |{}^{\bullet}G_1''|_{\circ} \to^{*}_{\mathrm{prune}} \cdot \leftarrow^{*}_{\mathrm{prune}} \cdot \leftarrow^{=}_{\mathrm{elim}} G_1' \end{array} \Big] \impliedby |{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} G_1' \Big] $$

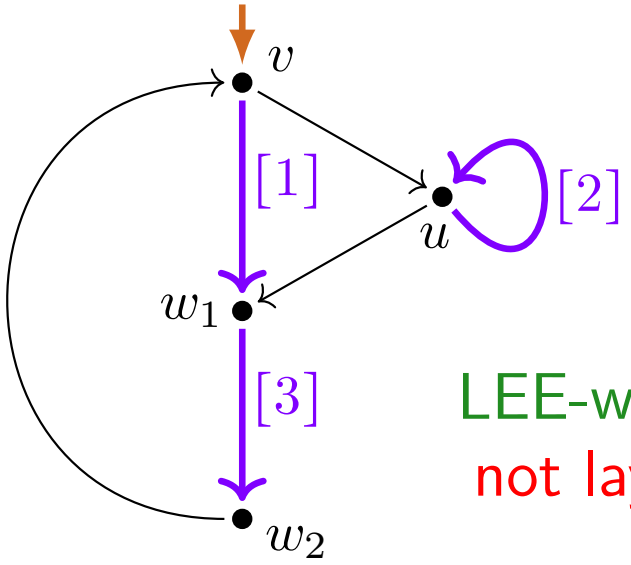
$$ \exists\, {}^{\bullet}G_1 \big[{}^{\circ}G \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_1 \wedge |{}^{\bullet}G_1|_{\circ} \leftrightarrow^{*}_{\mathrm{prune}} G_1 \big] \impliedby G \to^{*}_{\mathrm{elim}} G_1 $$

Theorem

$\mathrm{LLEE}(G) \Leftrightarrow \mathrm{LEE}(G)$ for all graphs G.

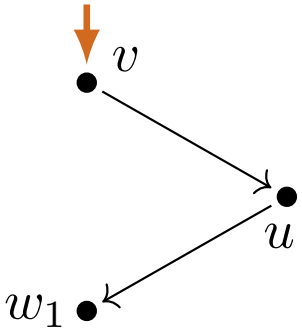
Making Layered

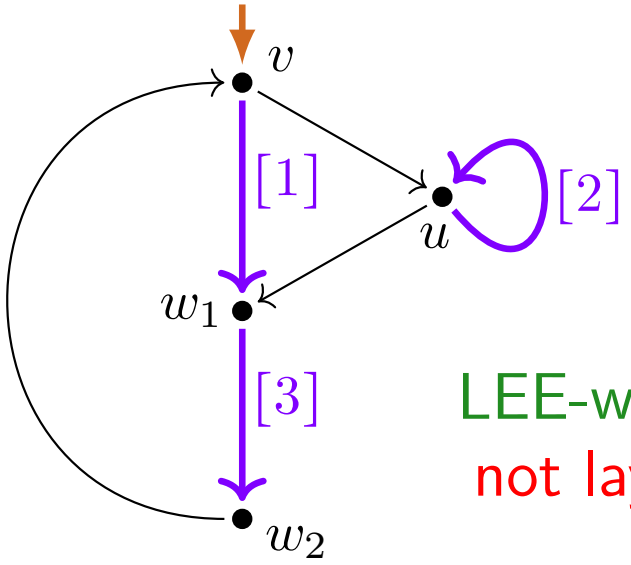




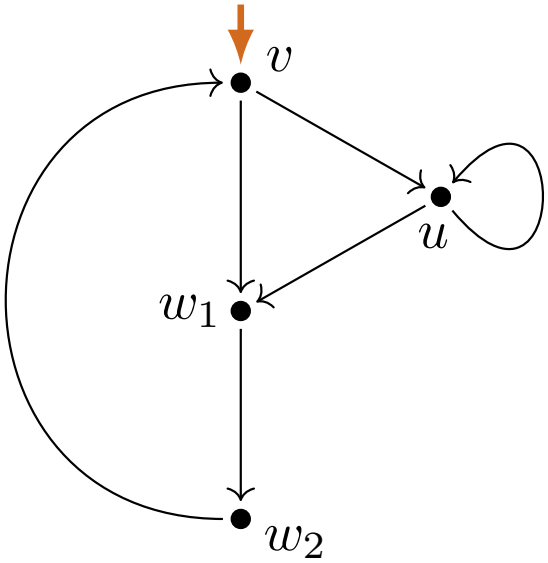
LEE-witness
not layered

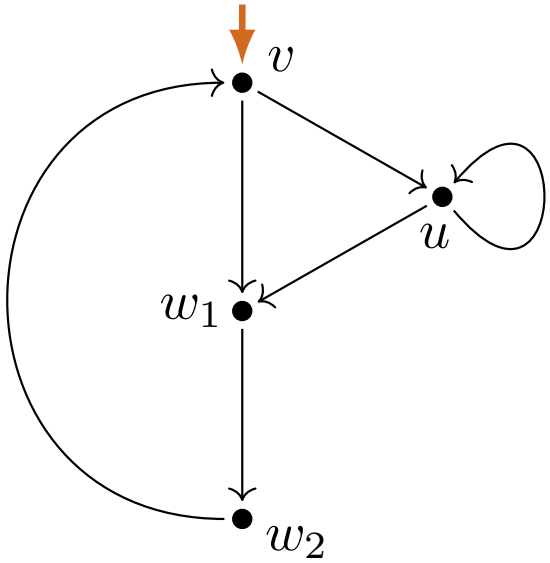
Making Layered



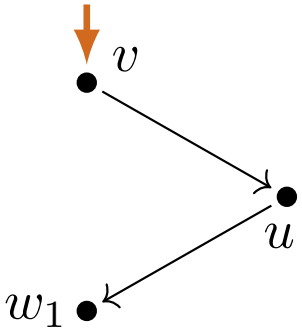


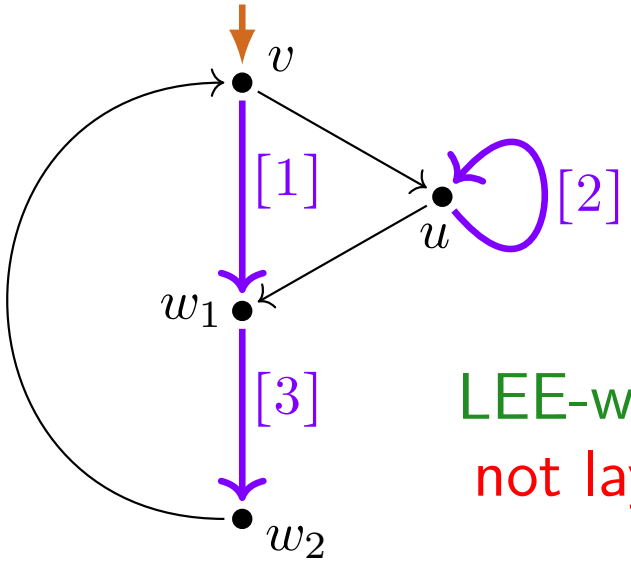
LEE-witness
not layered



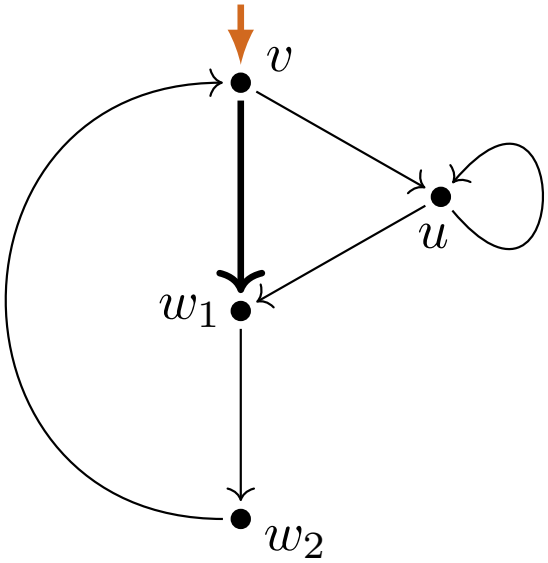


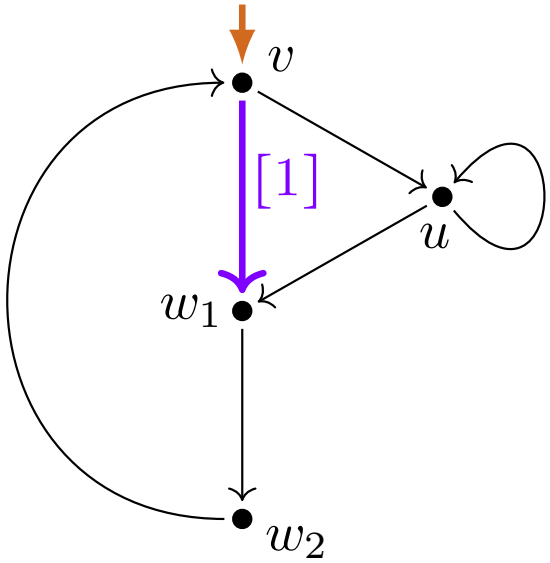
Making Layered



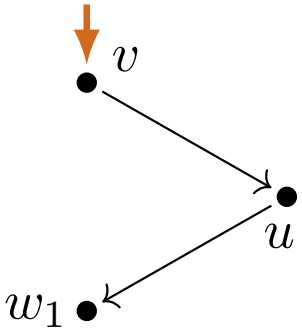


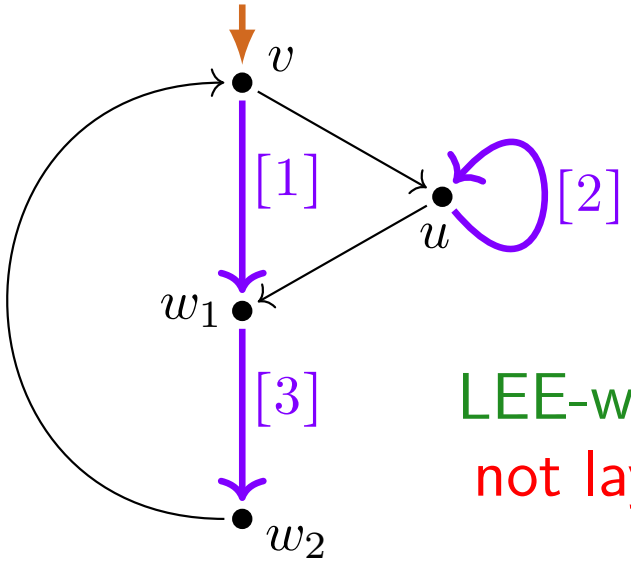
LEE-witness
not layered



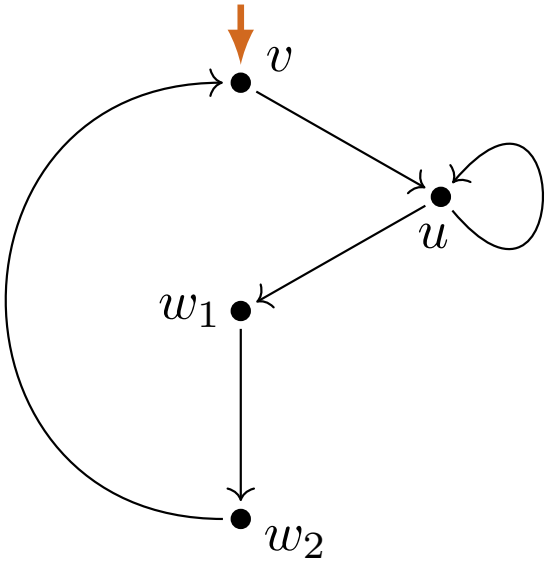


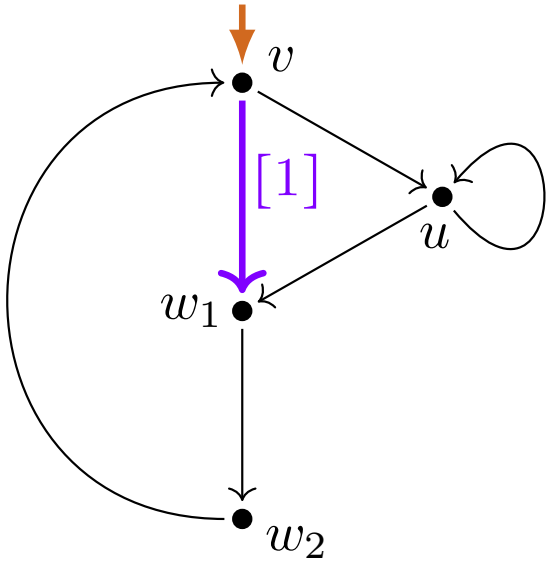
Making Layered



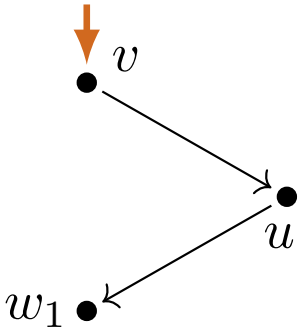


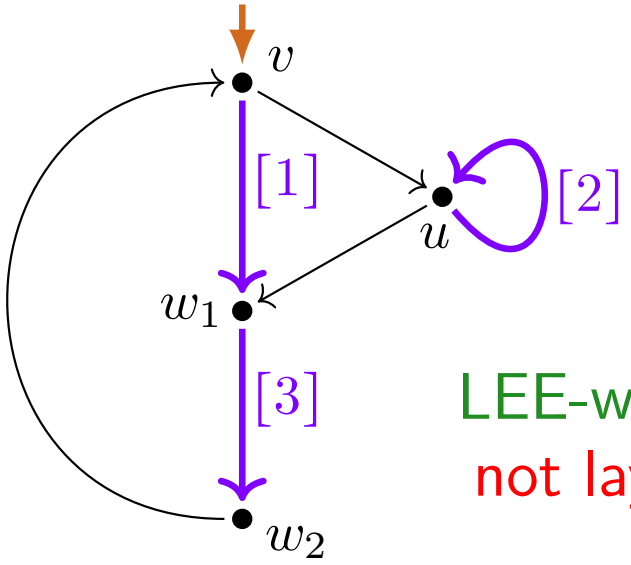
LEE-witness
not layered



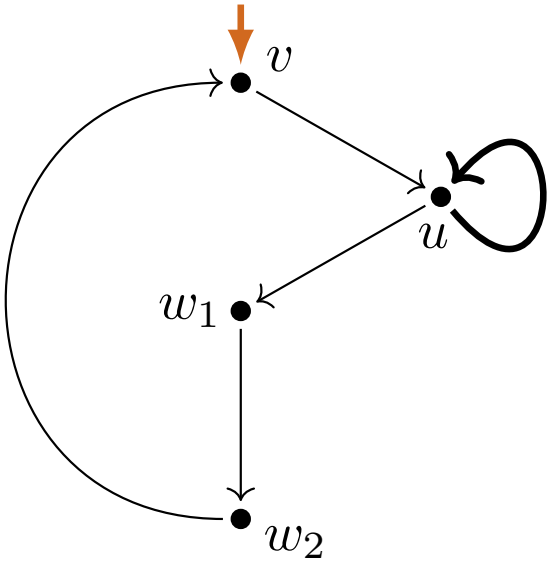


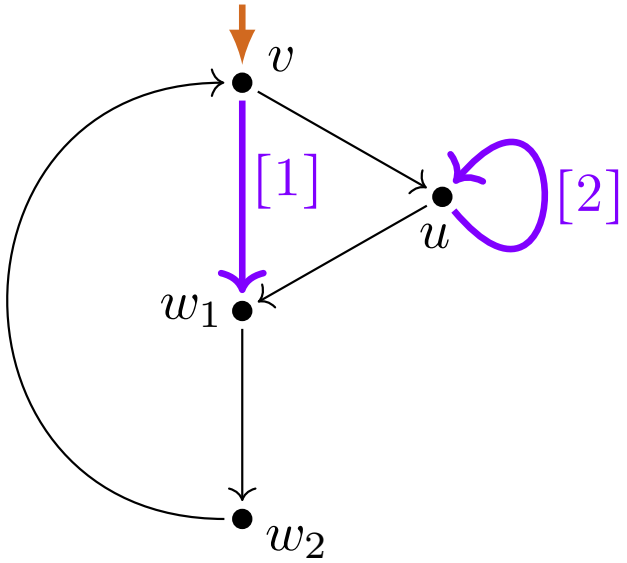
Making Layered



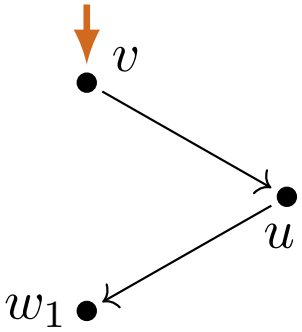


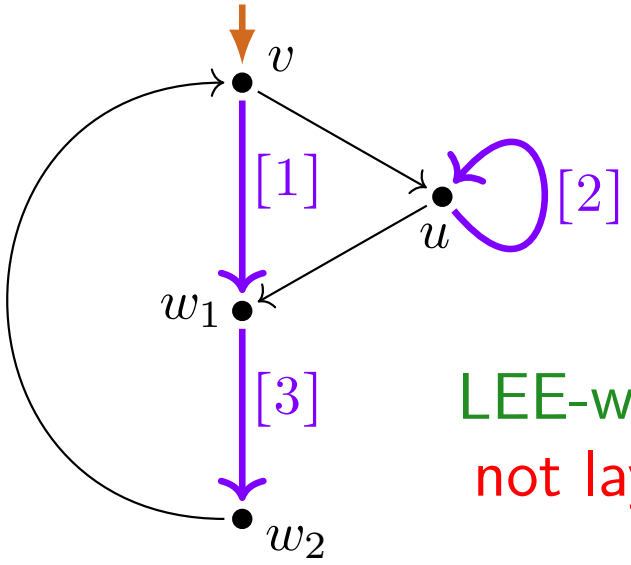
LEE-witness
not layered



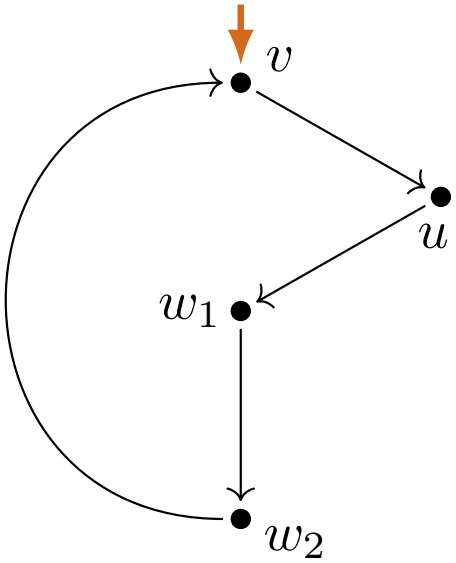


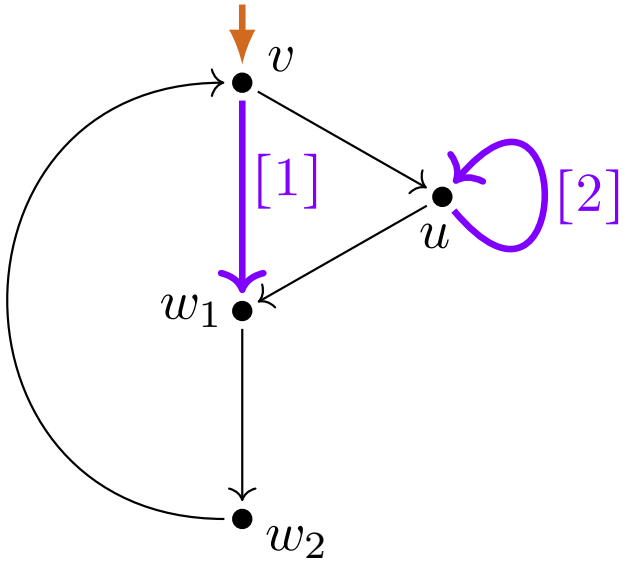
Making Layered



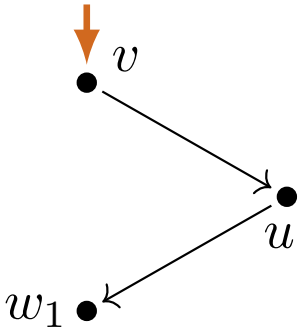


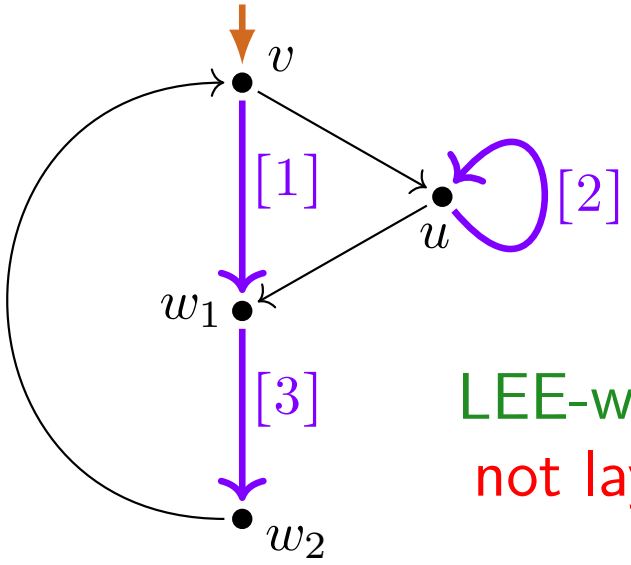
LEE-witness
not layered



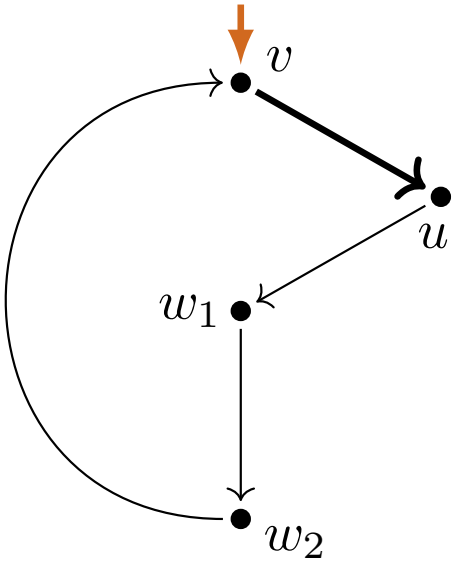


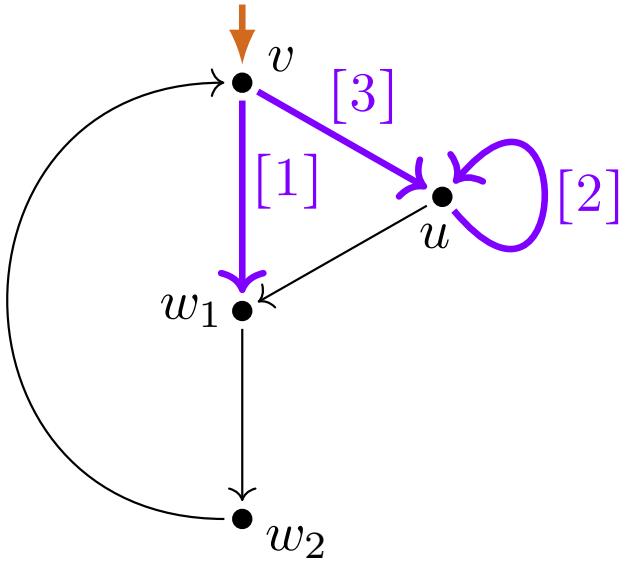
Making Layered



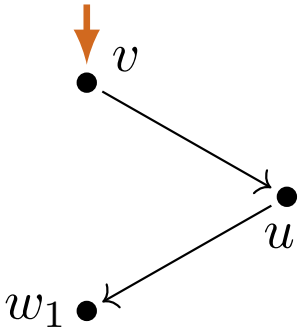


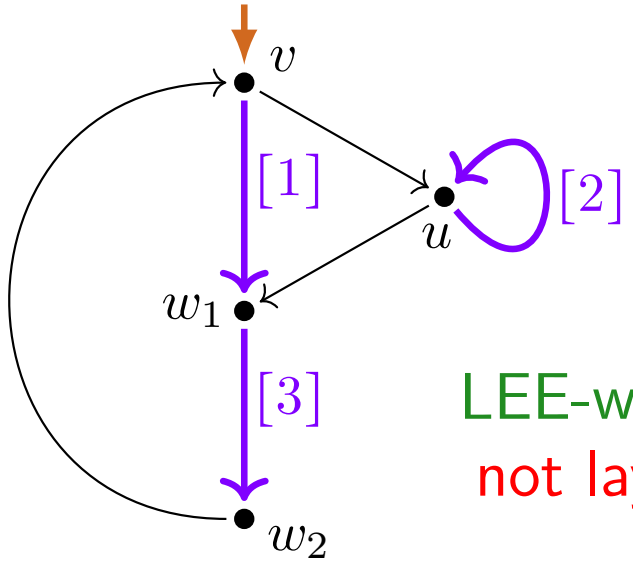
LEE-witness
not layered



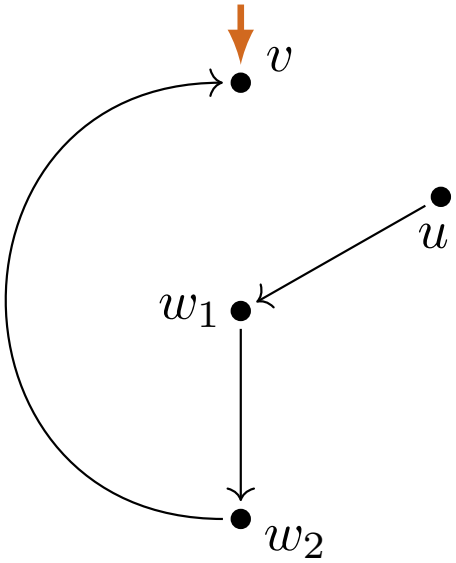


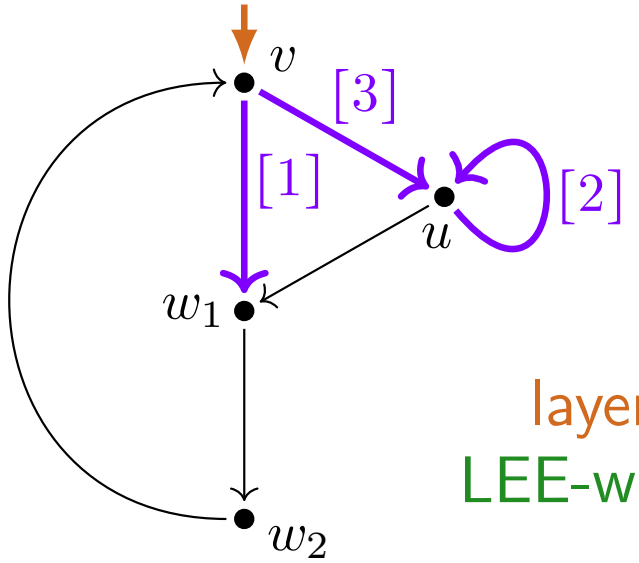
Making Layered





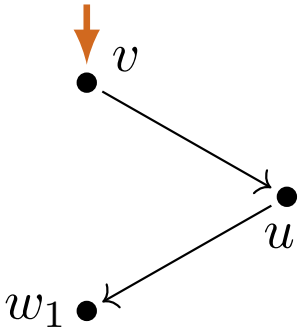
LEE-witness
not layered

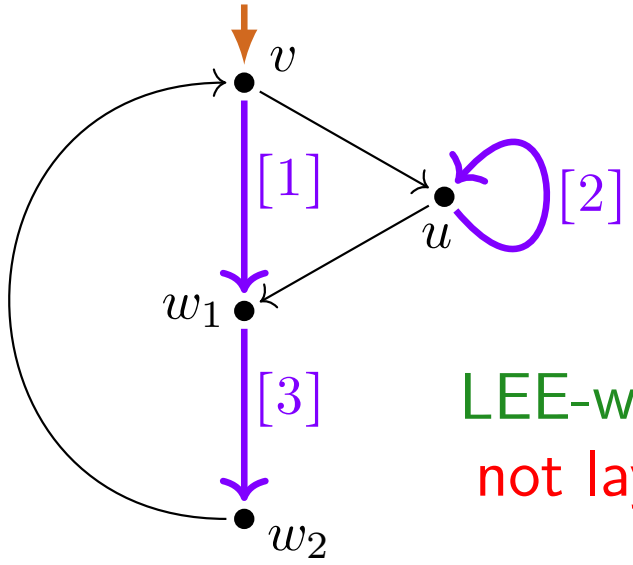




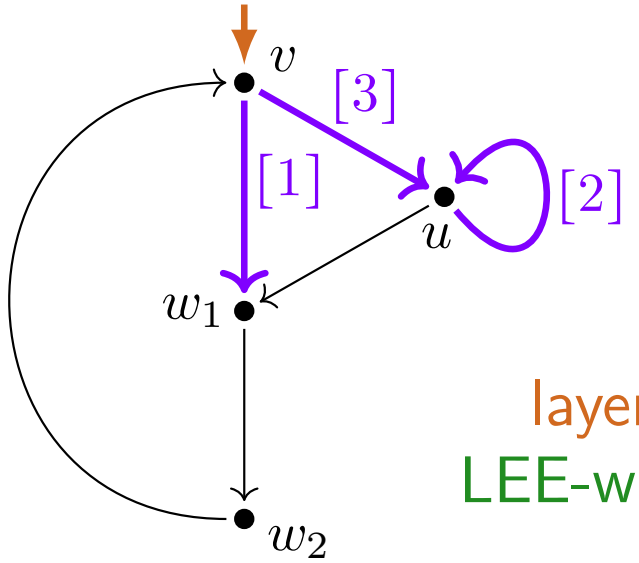
layered
LEE-witness

Making Layered





LEE-witness
not layered



layered
LEE-witness

Further questions

(Q1) How efficient ($\leq O((n+k)^3)$) can LEE be decided?

- To try: characterisation of LEE as flow-graph co-reducibility (Allen / Hecht–Ullman, 1970/72, Laarakker–Kappé, TERMGRAPH 2026).

(Q2) Is 1-LEE decidable in polynomial time?
(1-LEE $\hat{=}$ generalised loop elimination, which is not confluent.)

(Q3) How can, for a graph G with LEE, any two regular expressions e_1, e_2 with $P(e_1) = P(e_2) \simeq G$ that are extracted from maximal $\to_{\mathrm{elim}}$ rewrite sequences from G be proved equivalent with respect to $[\![\cdot]\!]_P$?

(Q4) Can the Decreasing Diagrams Method be adapted to show pre-/post-ponement? (Here postponement of $\overset{\circ}{\to}_{\mathrm{elim}}$ over $\to_{\mathrm{elim}}$ is closely connected to confluence of $\to_{\mathrm{ep}}$.)

- Vincent van Oostrom referred me to his IWC 2020 article.

Summary

- Loop elimination $\to_{\mathrm{elim}}$ in process graphs
  - Milner's process interpretation P of regular expressions
  - Loop existence and elimination property LEE
    LEE = compact process interpretations of $(*/\not{1})$ reg. expr's
  - $\to_{\mathrm{elim}}$ is not confluent
- Completion of loop elimination $\to_{\mathrm{elim}}$ with pruning steps $\to_{\mathrm{prune}}$
  - 'critical pair' lemma for joining elimination forks $\leftarrow_{\mathrm{elim}} \cdot \to_{\mathrm{elim}}$
  - use of decreasing diagrams for $\to_{\mathrm{ep}} = \to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$
- Application 1: LEE is decidable in polynomial time
- Application 2: Layered LEE = LEE

IFIP-1.6 and TERMGRAPH talks

TERMGRAPH (last Sunday)

CG: *Towards a Characterisation of the Graph Structure of Process Interpretations of Regular Expressions*

Ext. Abstract https://clegra.github.io/lf/pi-struc-TG-26.pdf

Slides https://clegra.github.io/lf/TG-26.pdf

- LEE $\hat{=}$ image of $P^\bullet|_{RExp^{(*/\not{1})}}$
- 1-LEE $\hat{=}$ image of $P^\bullet$

IFIP-1.6 and TERMGRAPH talks

TERMGRAPH (last Sunday)

CG: *Towards a Characterisation of the Graph Structure of Process Interpretations of Regular Expressions*

Ext. Abstract https://clegra.github.io/lf/pi-struc-TG-26.pdf

Slides https://clegra.github.io/lf/TG-26.pdf

- LEE $\hat{=}$ image of $P^\bullet|_{RExp^{(*/\not{1})}}$
- 1-LEE $\hat{=}$ image of $P^\bullet$

IFIP-1.6 meeting (last Saturday)

CG: *Finding Small Steps for Hard Problems, in Work on the Process Interpretation of Regular Expressions*

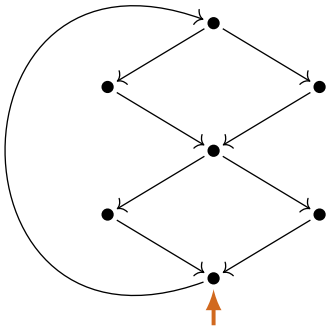
Slides https://clegra.github.io/lf/IFIP-1_6-2026.pdf

- Reducing for Graphs Representing Regular Expressions under Bisimilarity

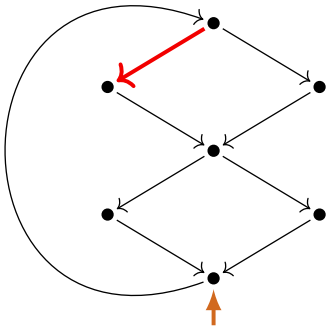
Resources

- Slides/extended abstract on clegra.github.io
  - slides: .../lf/IWC-26.pdf
  - extended abstract: .../lf/le-conf-IWC-26.pdf
- CG: Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete
  - LICS 2022, arXiv:2209.12188, poster.
- CG: The Image of the Process Interpretation of Regular Expressions is Not Closed under Bisimulation Collapse
  - arXiv:2303.08553, 2021/2023.
- CG, Wan Fokkink: A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity,
  - LICS 2020, arXiv:2004.12740, video on youtube.
- CG: Structure-Constrained Process Graphs for the Process Semantics of Regular Expressions
  - TERMGRAPH 2020, EPTCS 334, arXiv:2012.10869.
- CG: Modeling Terms by Graphs with Structure Constraints
  - TERMGRAPH 2018, EPTCS 288, arXiv:1902.02010.

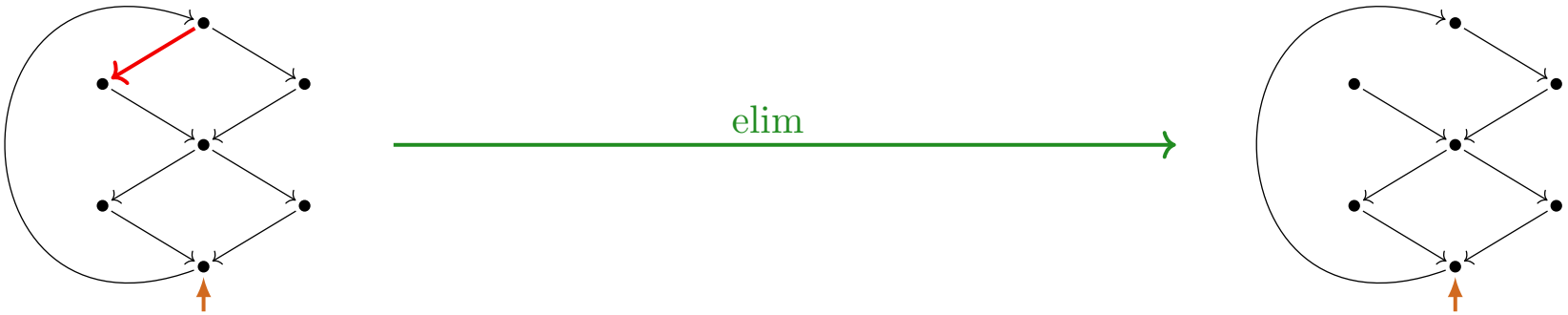
‘Critical pair’: bi-loop elimination



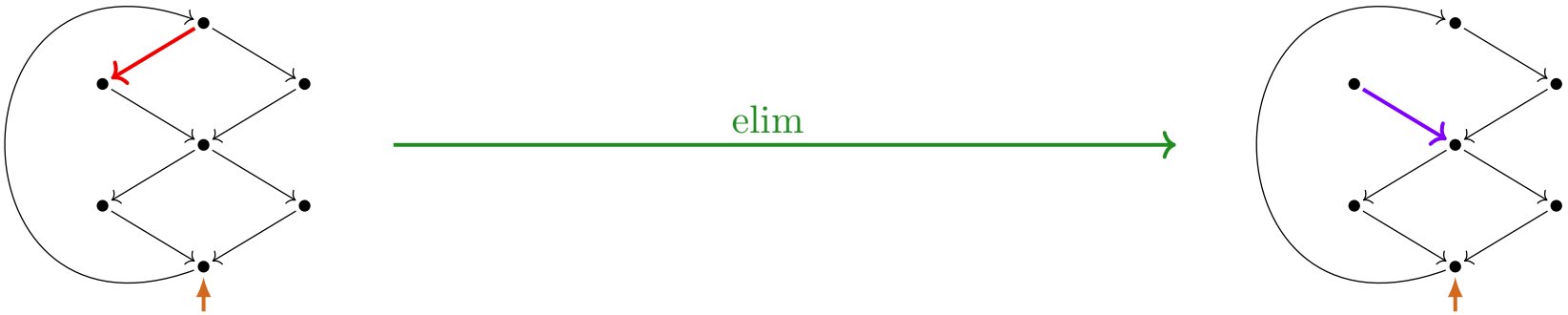
‘Critical pair’: bi-loop elimination



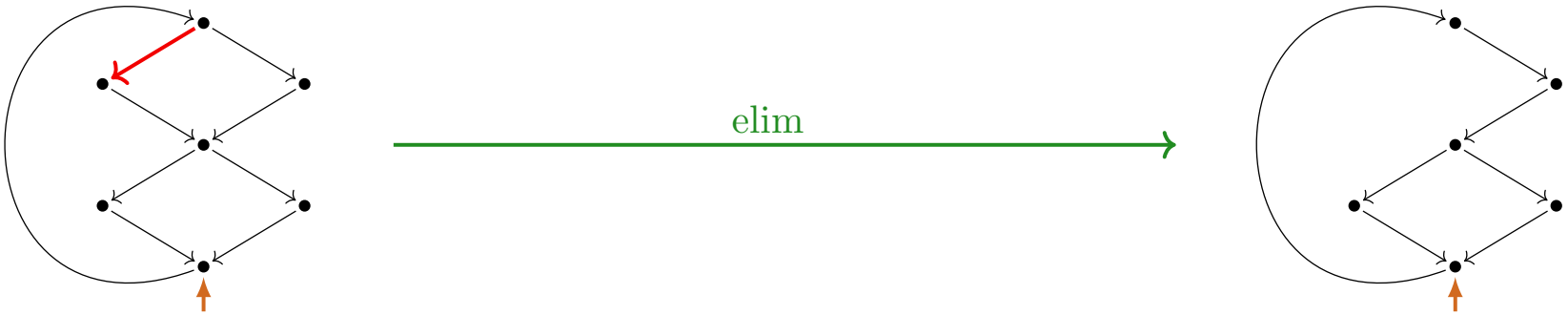
‘Critical pair’: bi-loop elimination



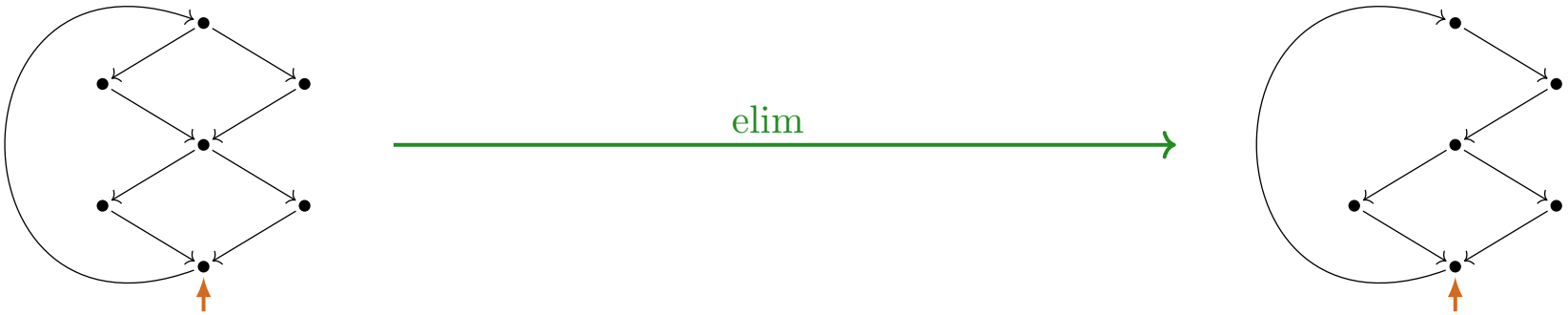
‘Critical pair’: bi-loop elimination



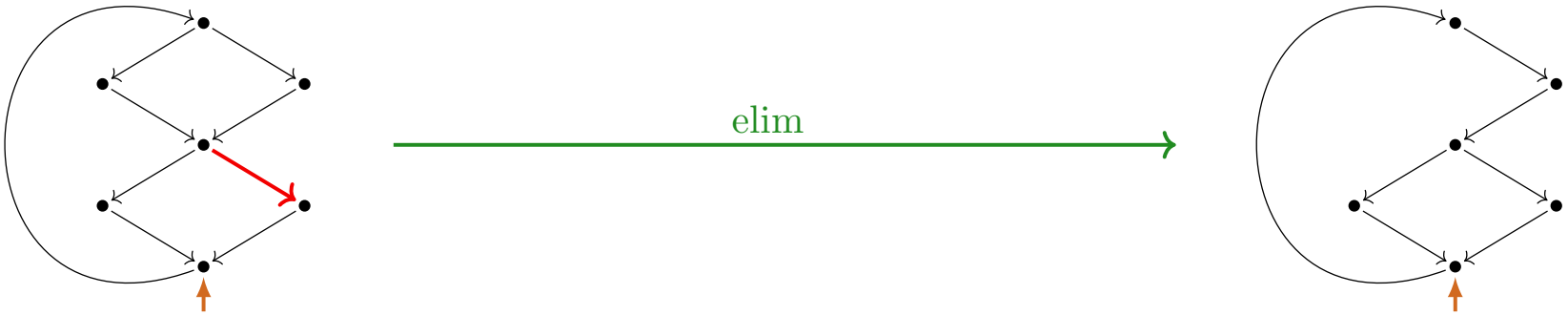
‘Critical pair’: bi-loop elimination



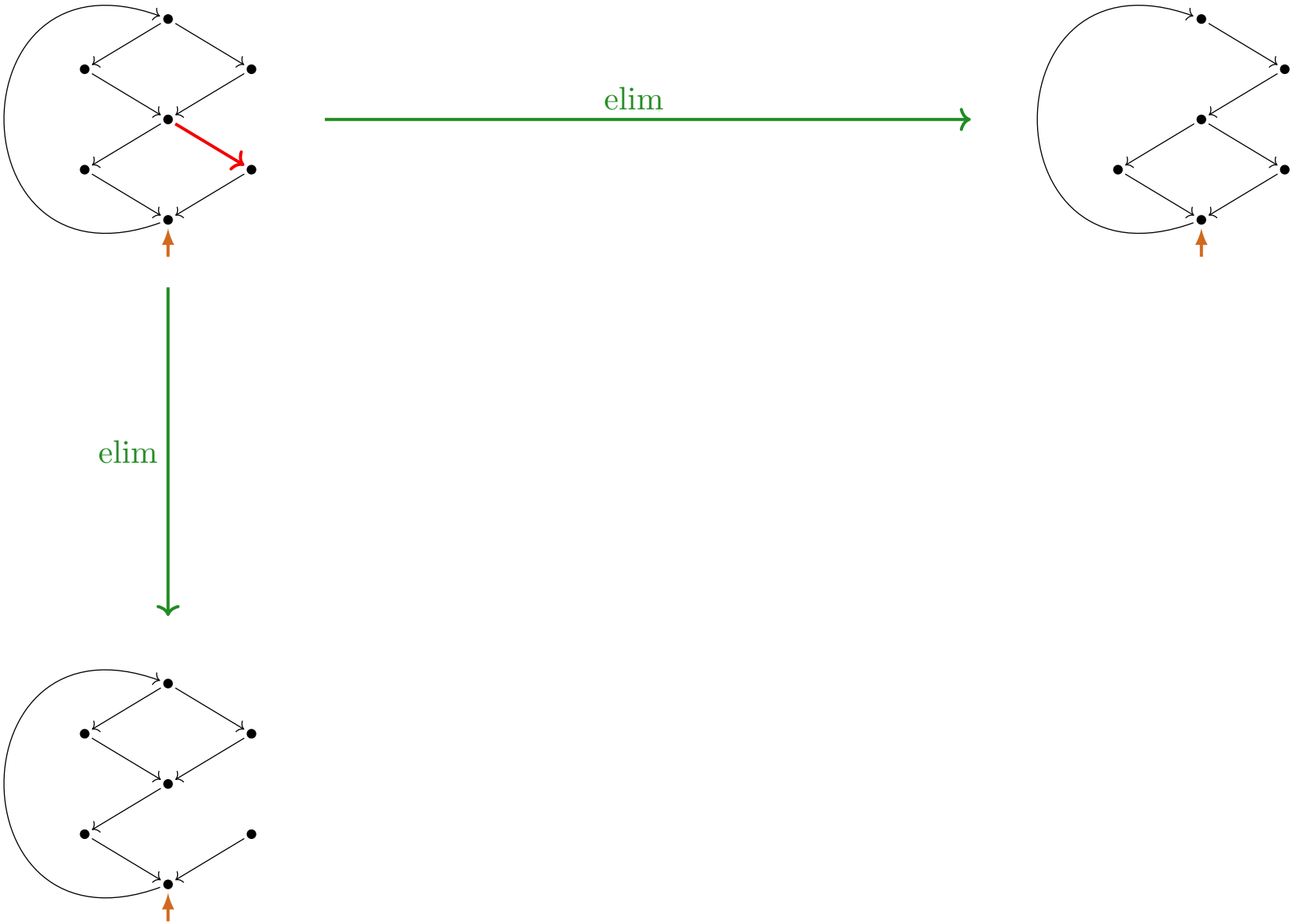
‘Critical pair’: bi-loop elimination



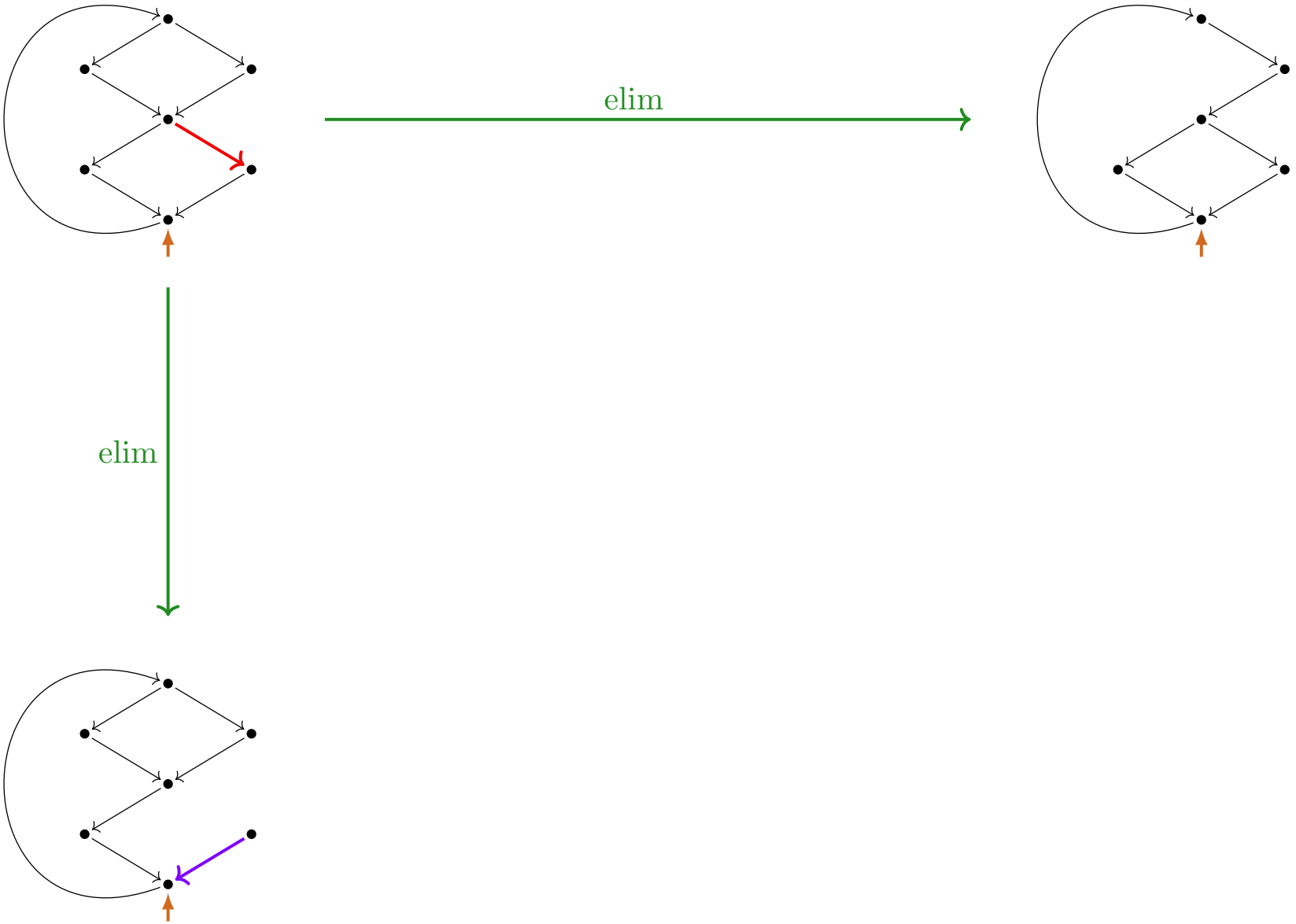
‘Critical pair’: bi-loop elimination



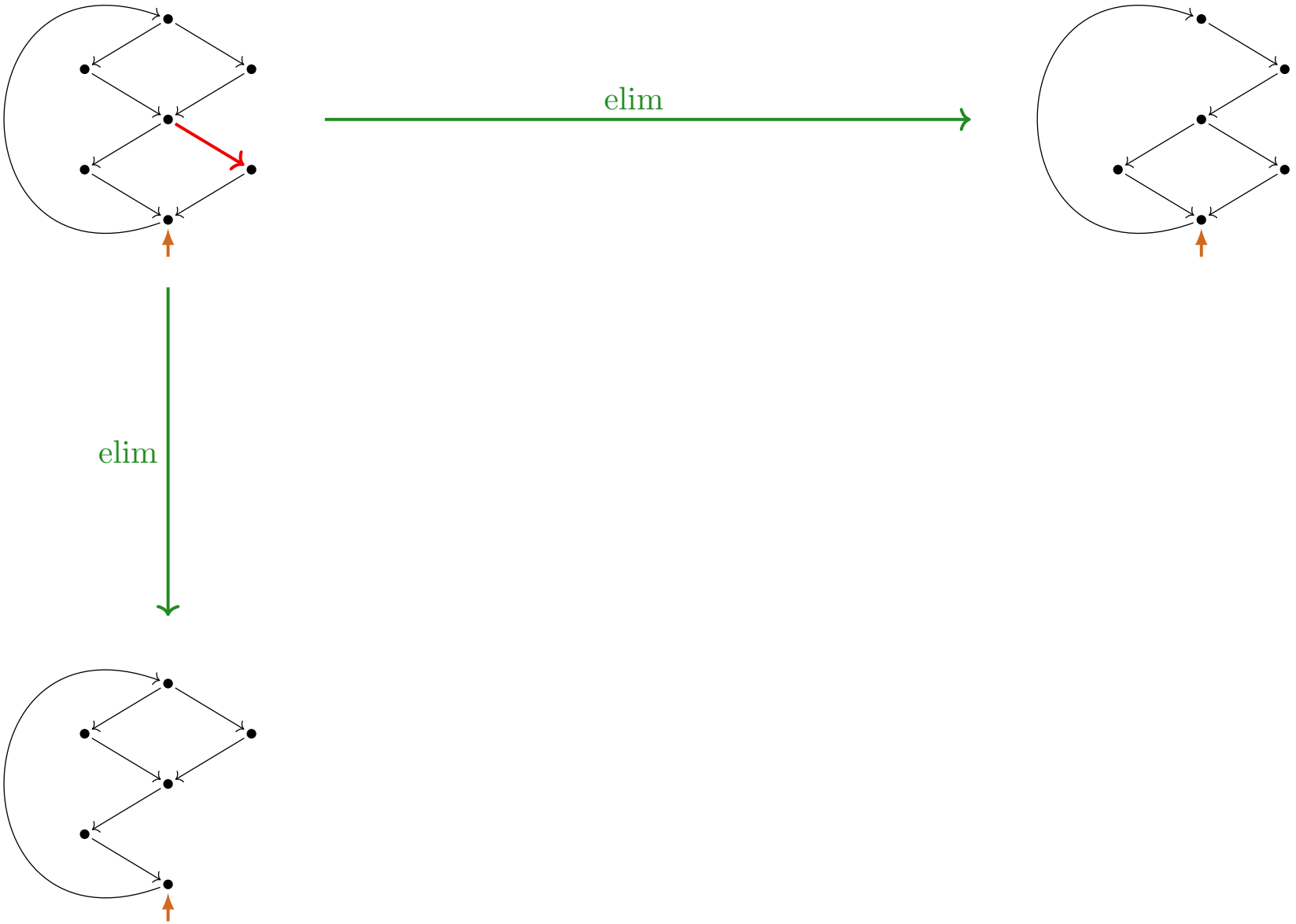
‘Critical pair’: bi-loop elimination



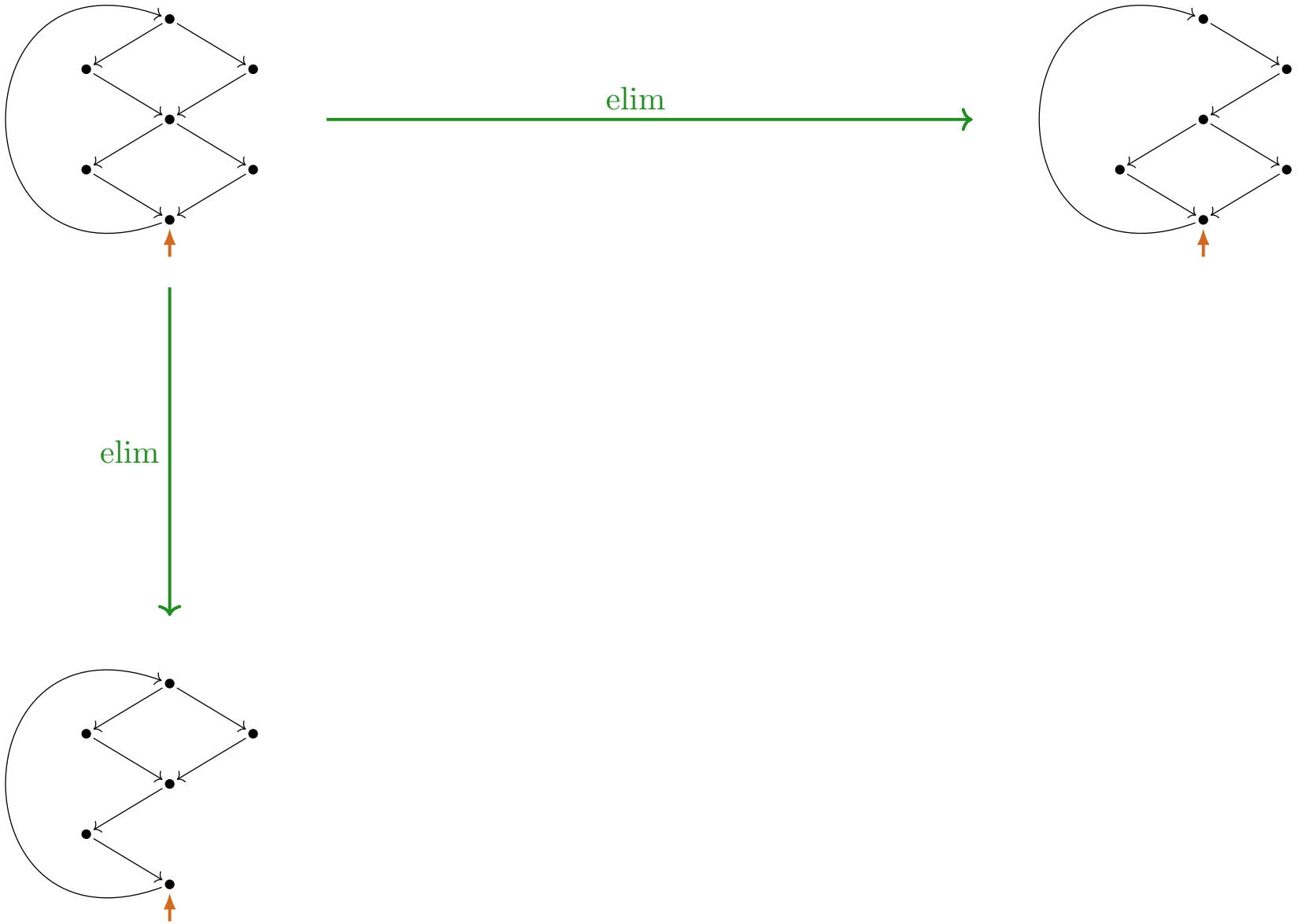
‘Critical pair’: bi-loop elimination



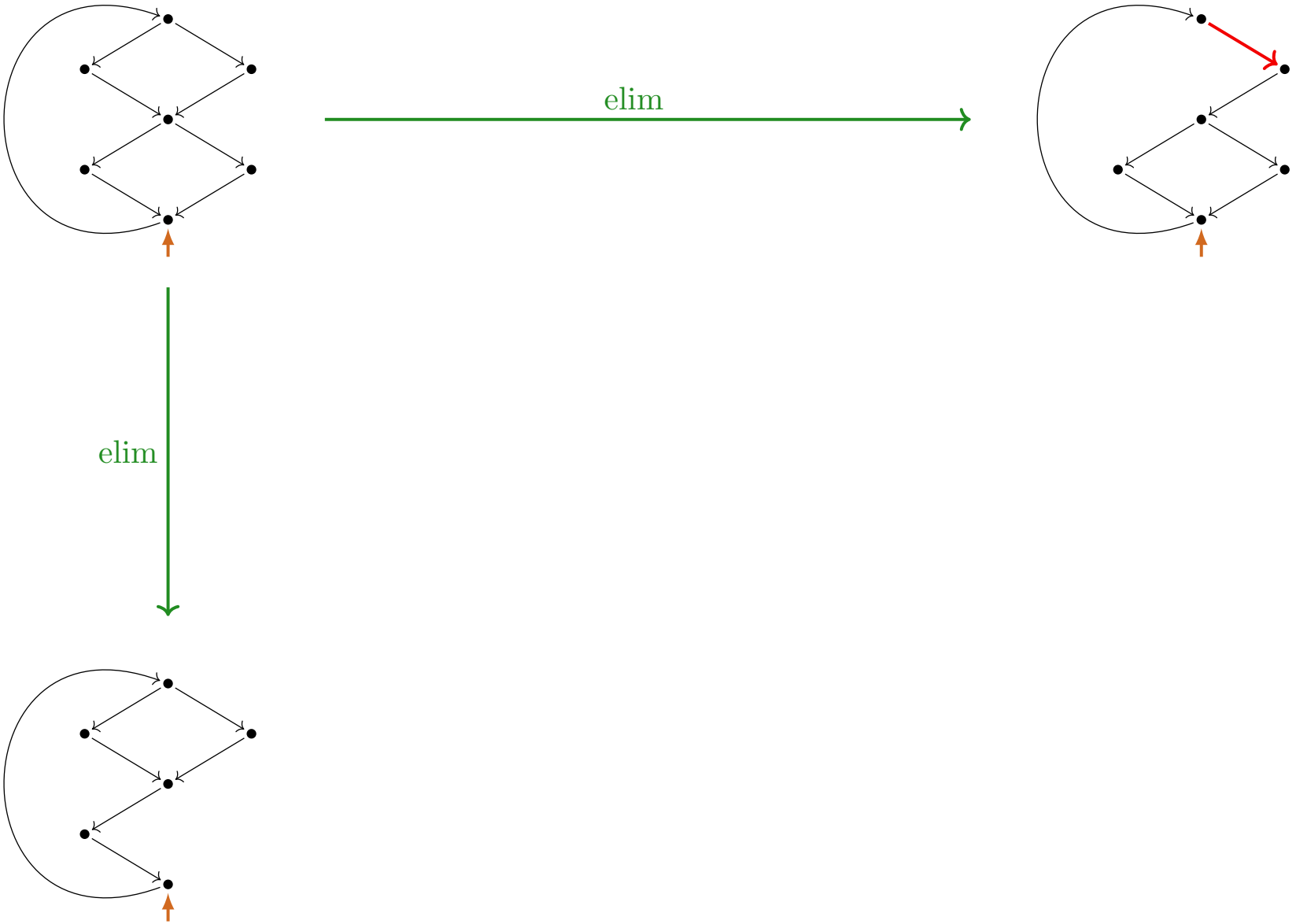
‘Critical pair’: bi-loop elimination



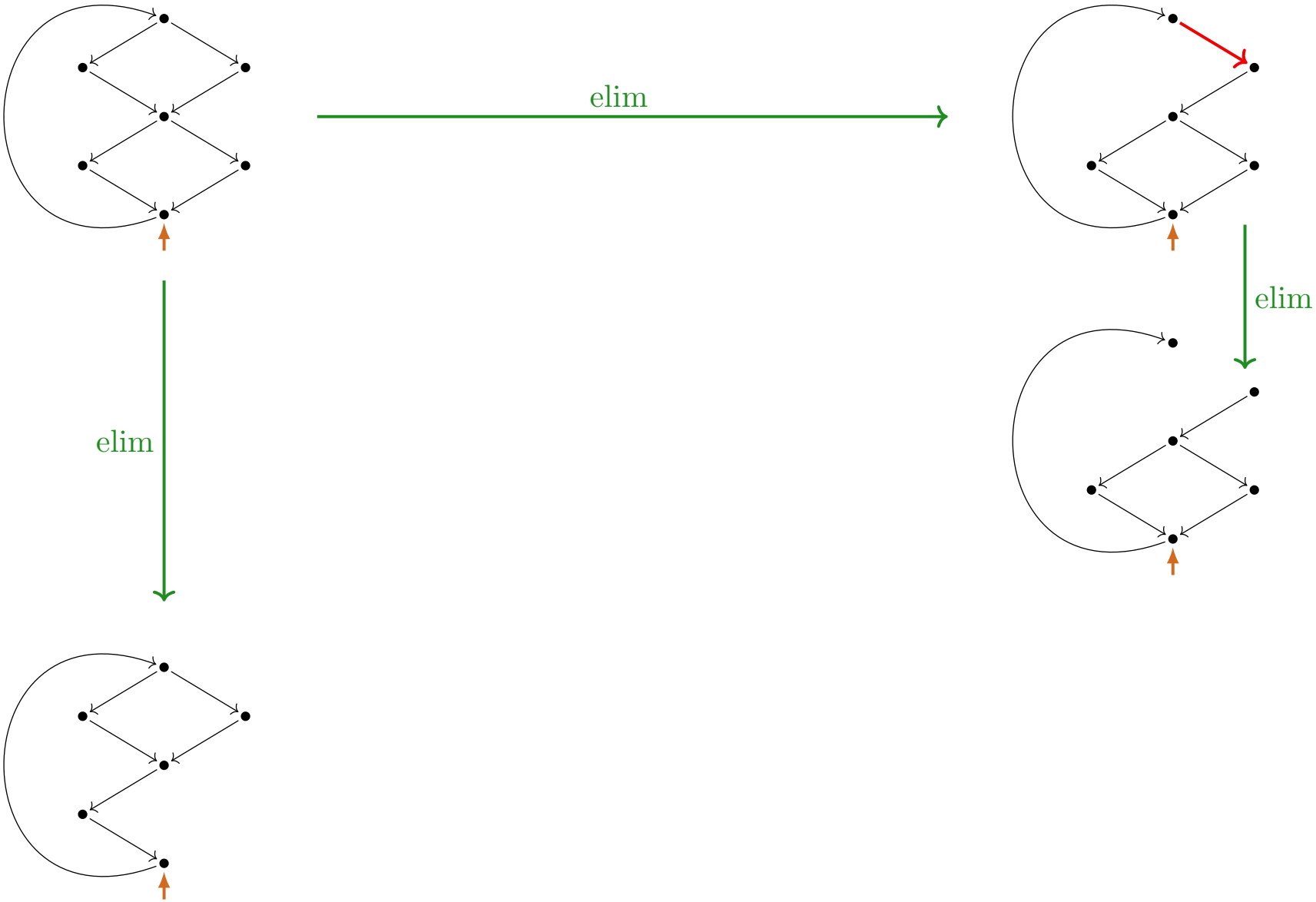
‘Critical pair’: bi-loop elimination



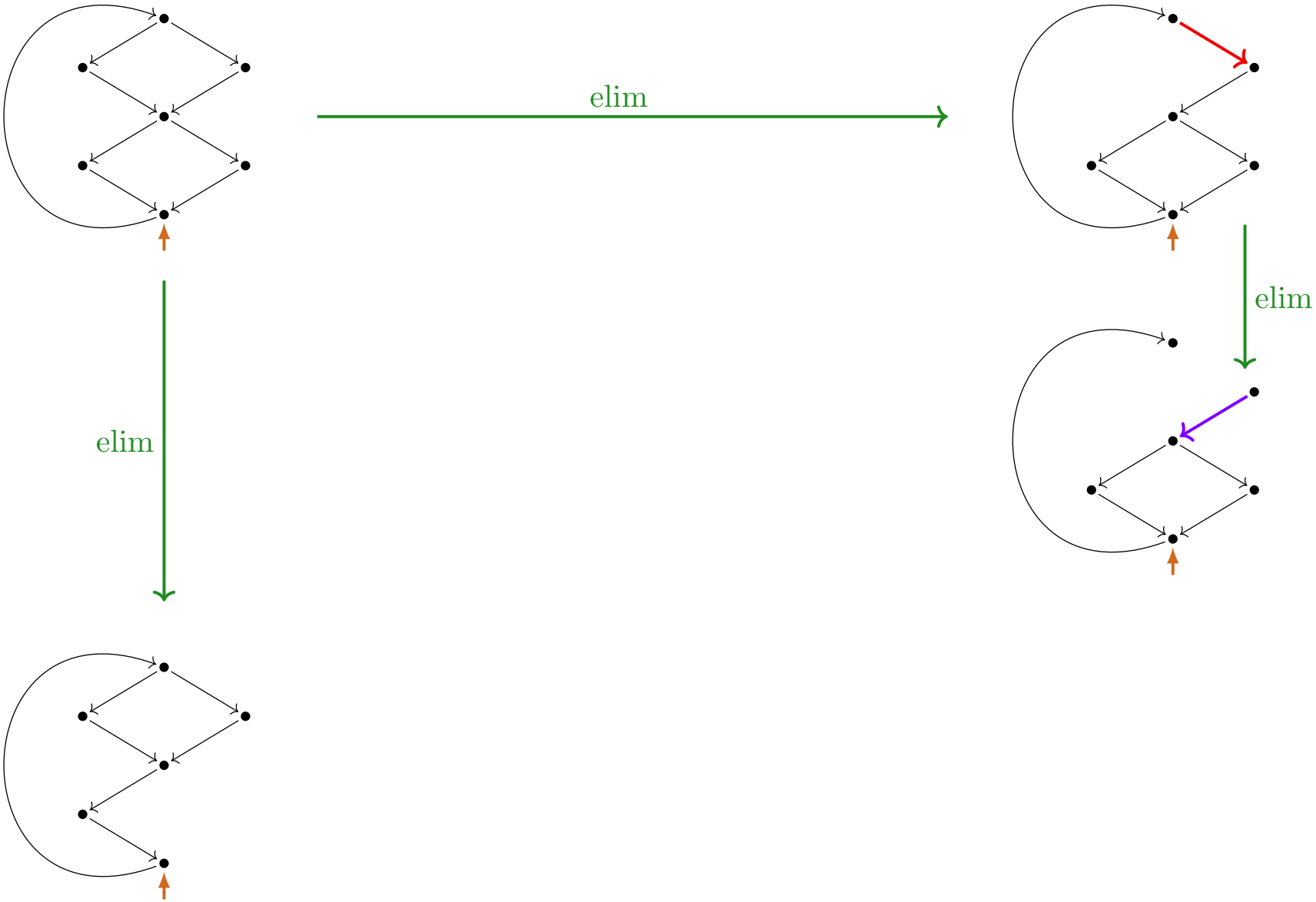
‘Critical pair’: bi-loop elimination



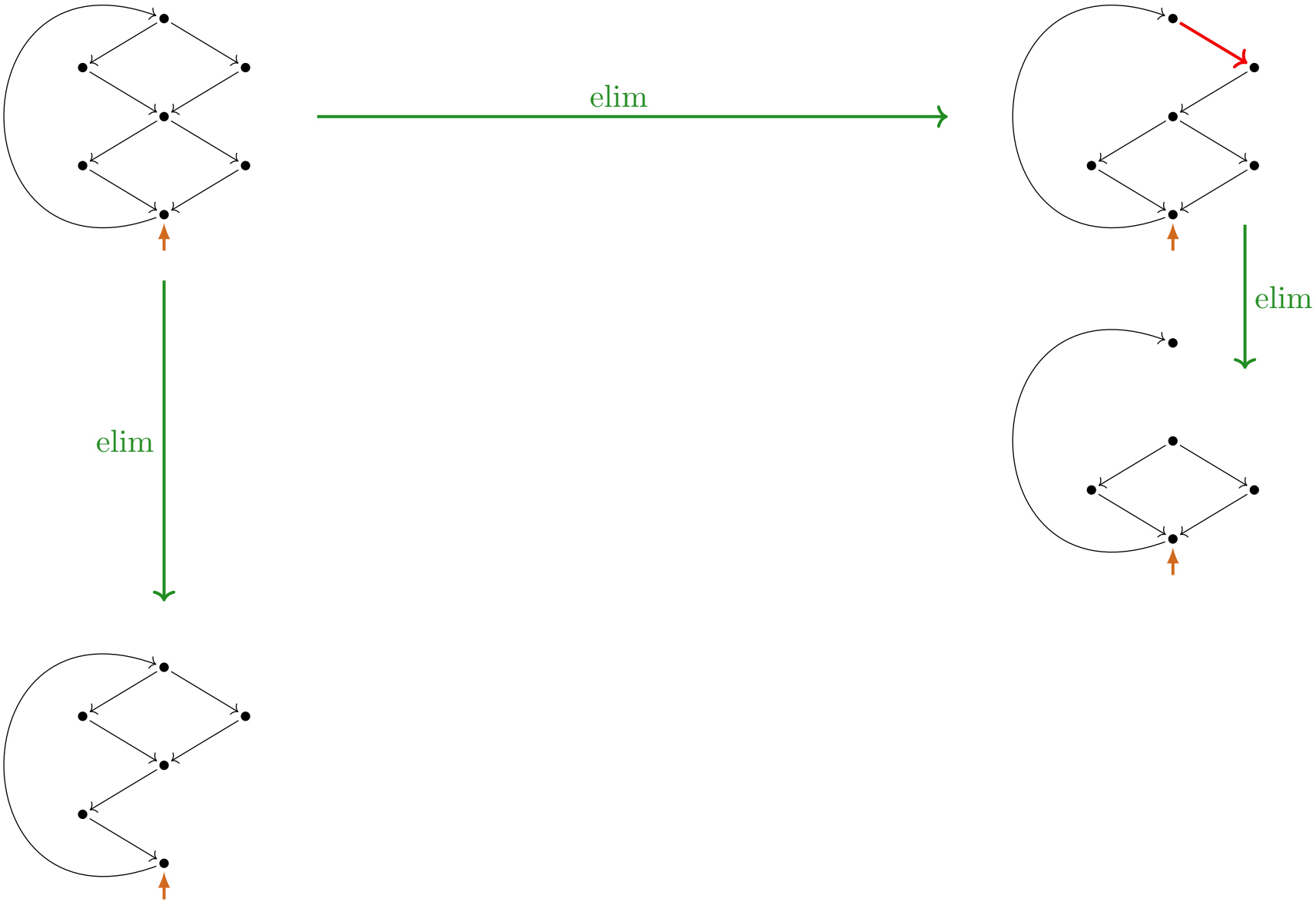
‘Critical pair’: bi-loop elimination



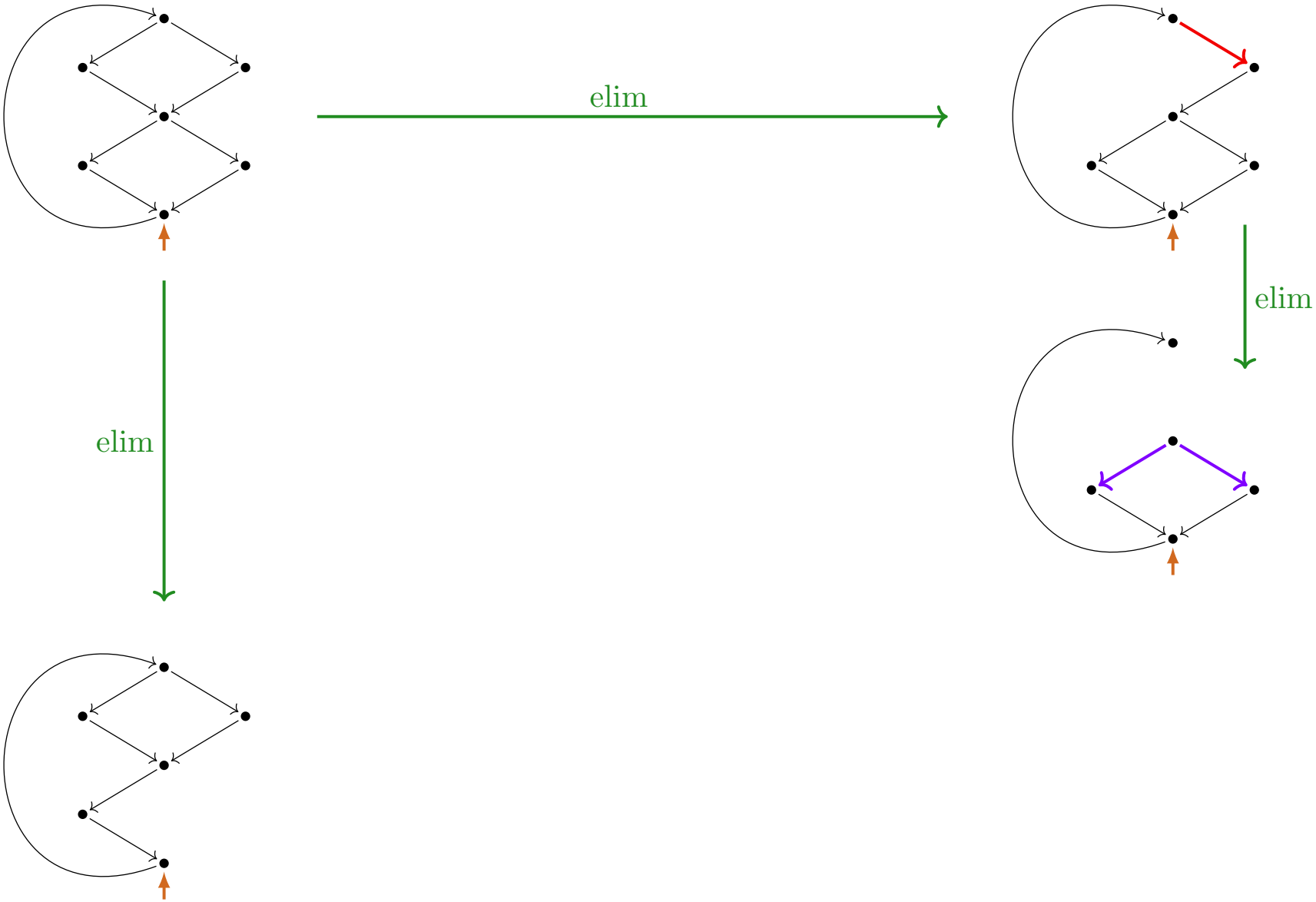
‘Critical pair’: bi-loop elimination



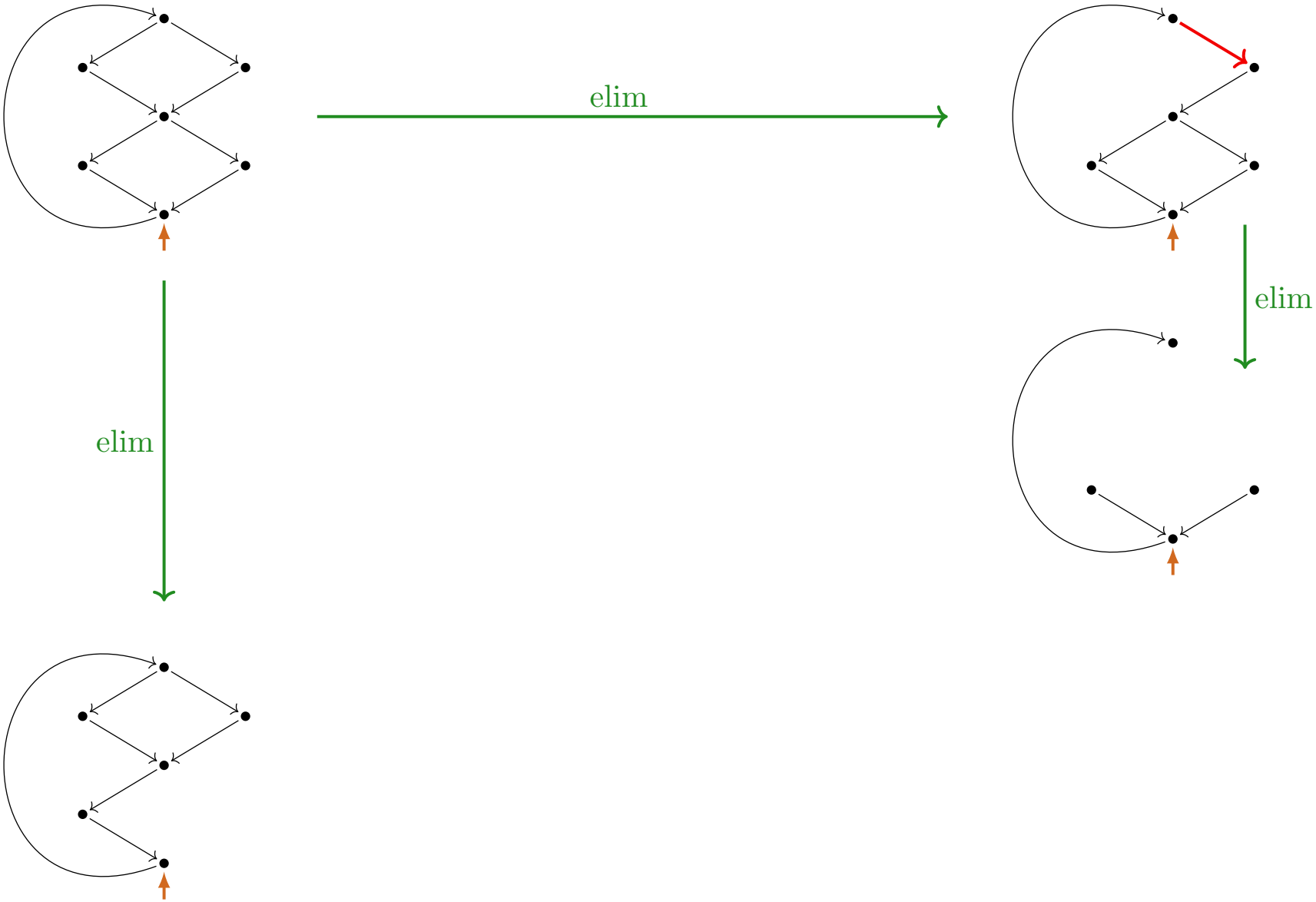
‘Critical pair’: bi-loop elimination



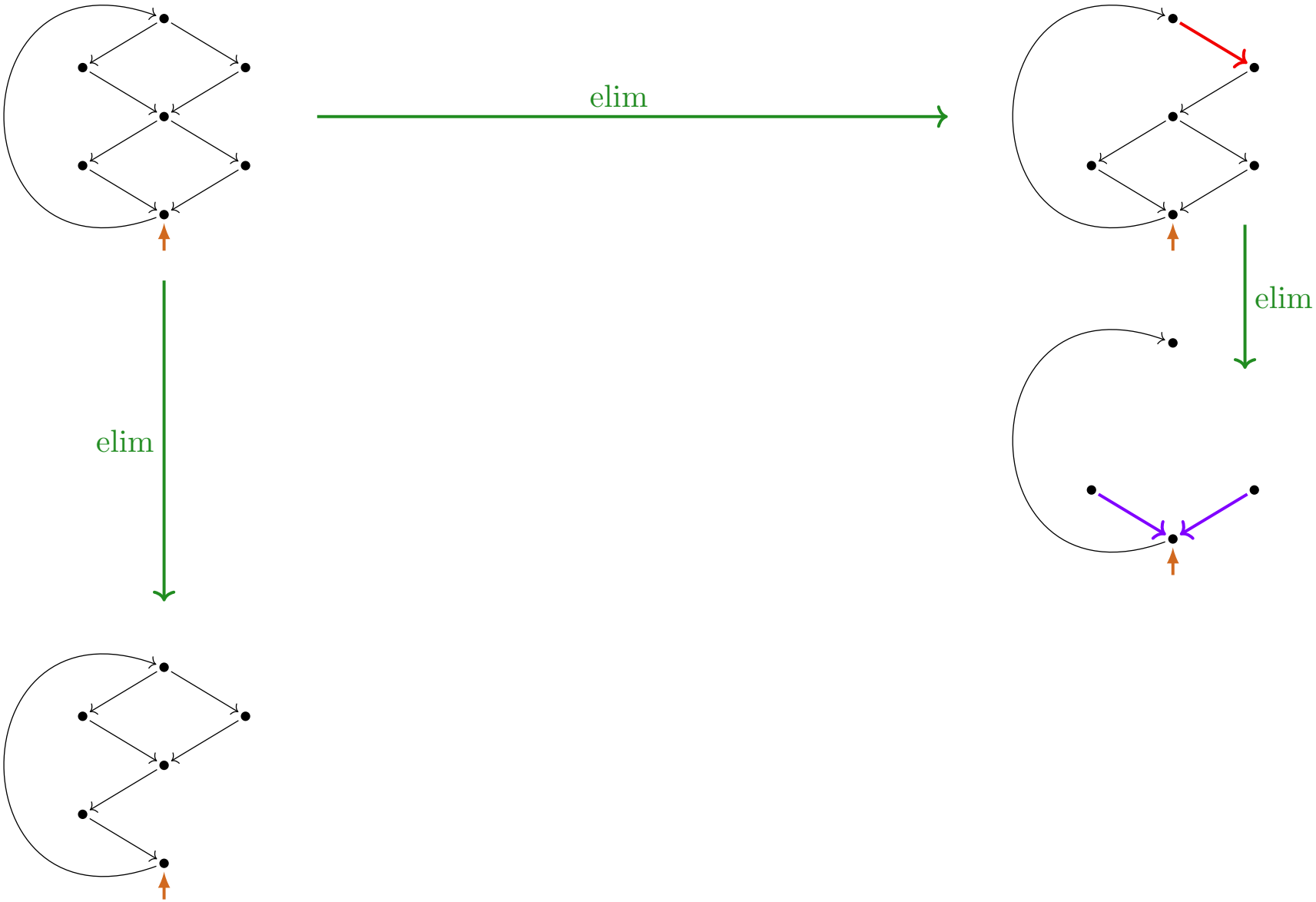
‘Critical pair’: bi-loop elimination



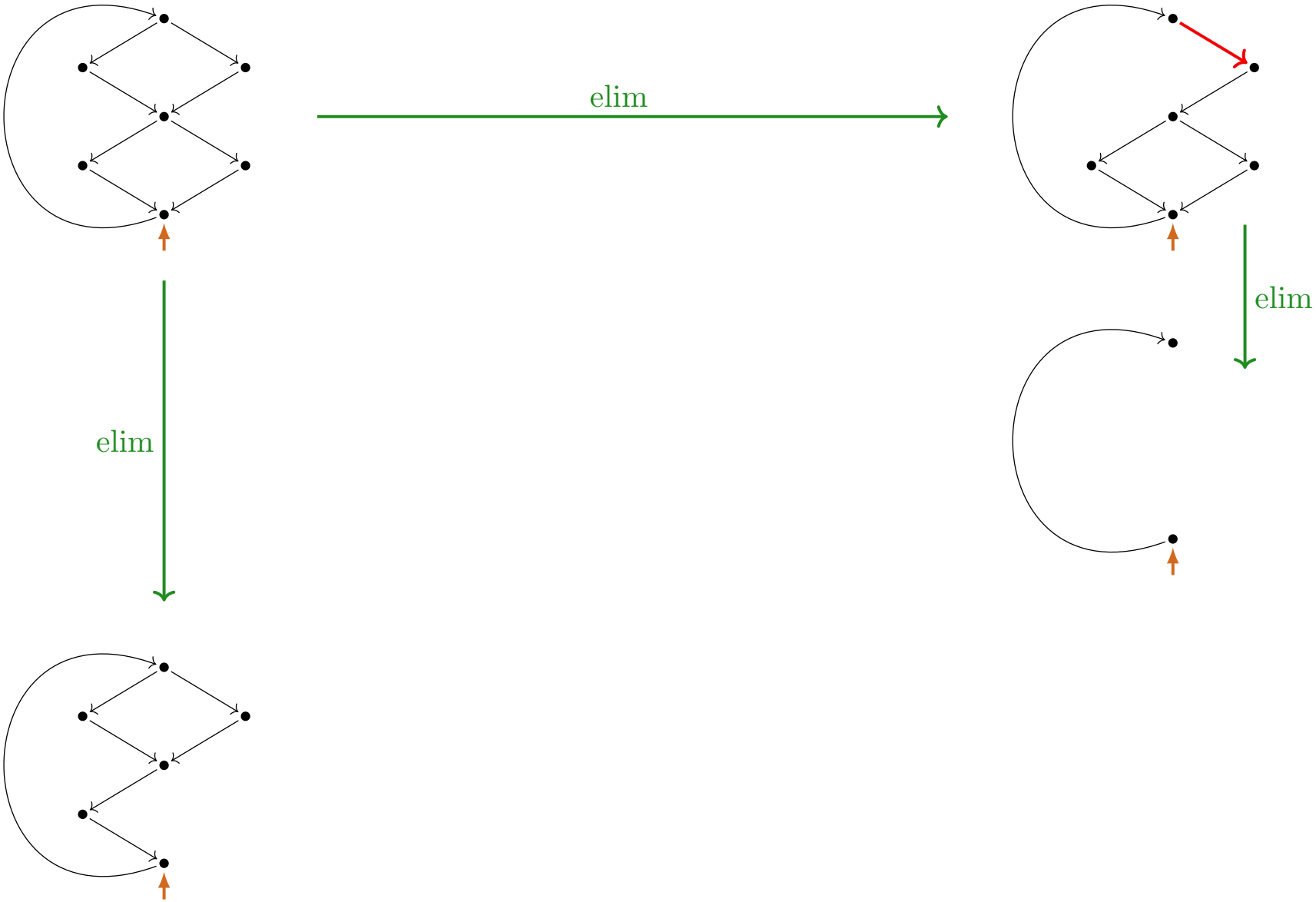
‘Critical pair’: bi-loop elimination



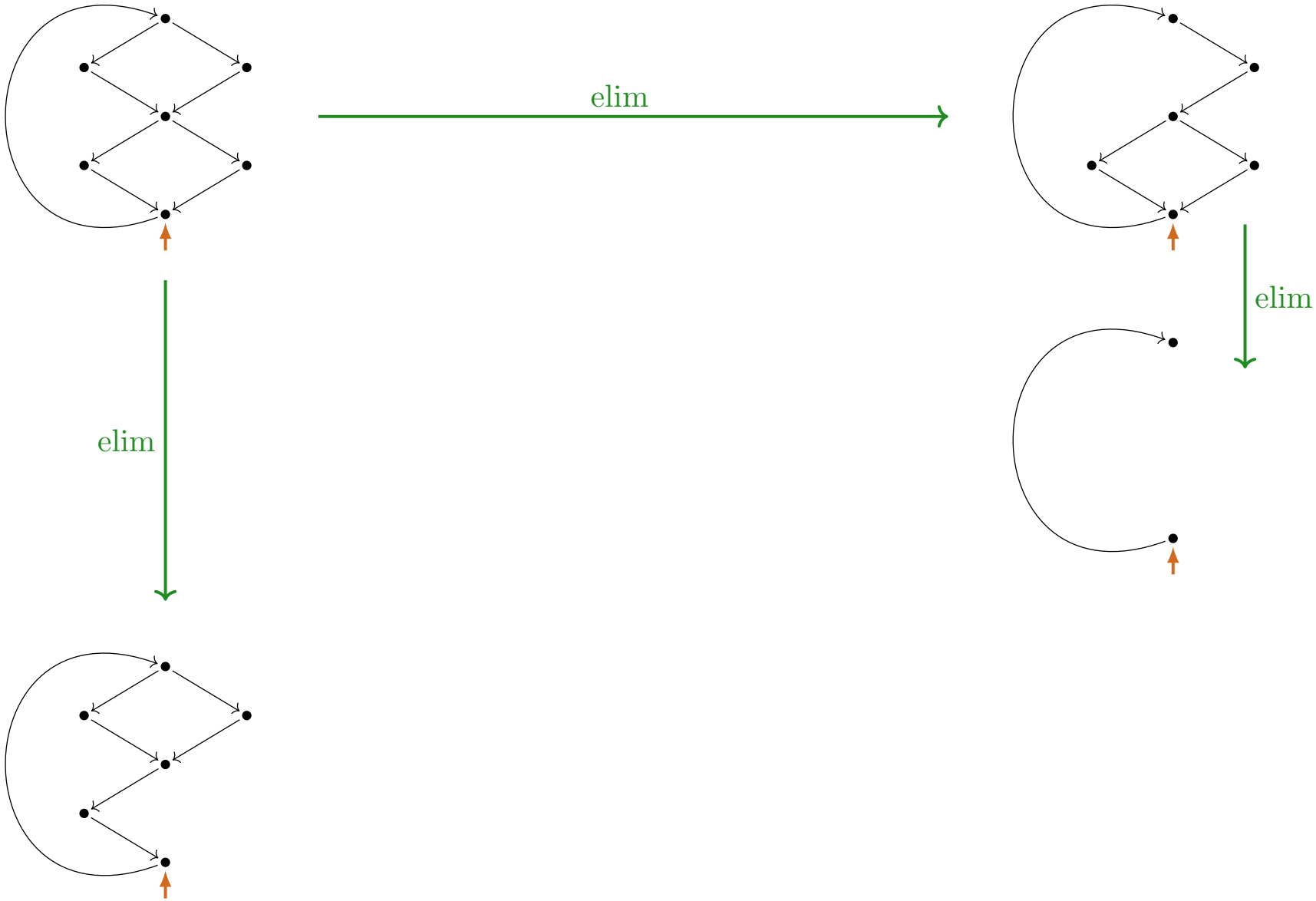
‘Critical pair’: bi-loop elimination



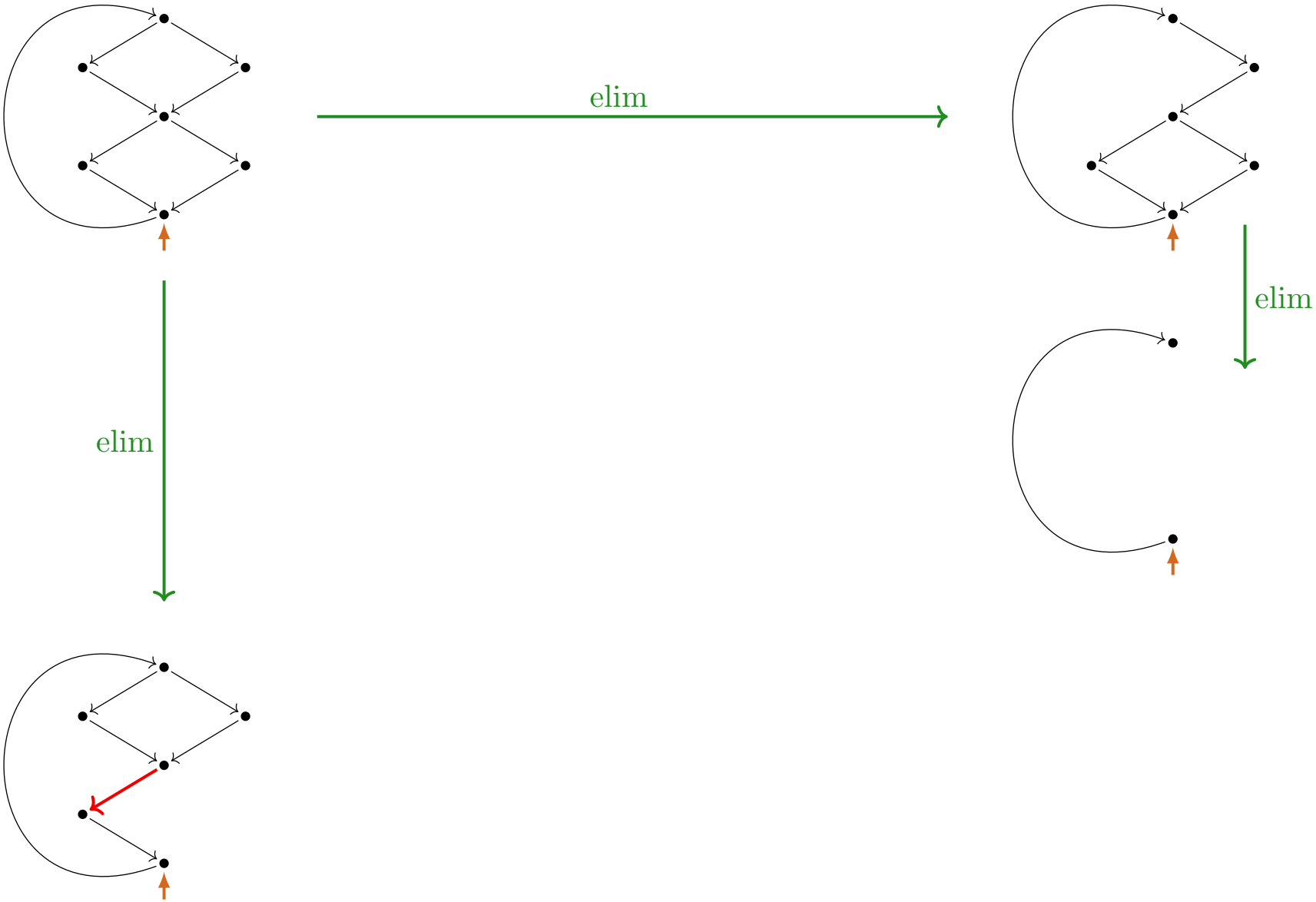
‘Critical pair’: bi-loop elimination



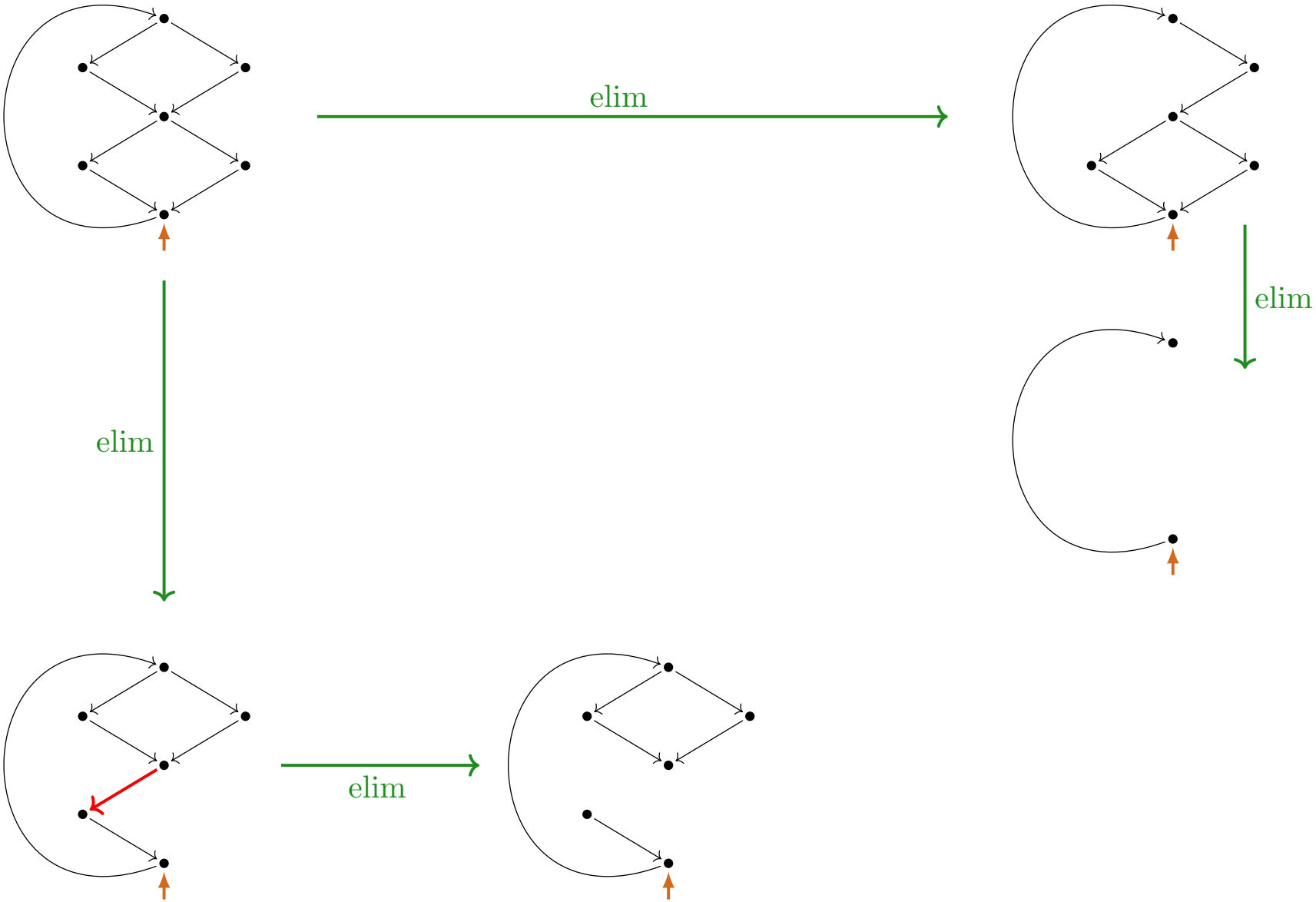
‘Critical pair’: bi-loop elimination



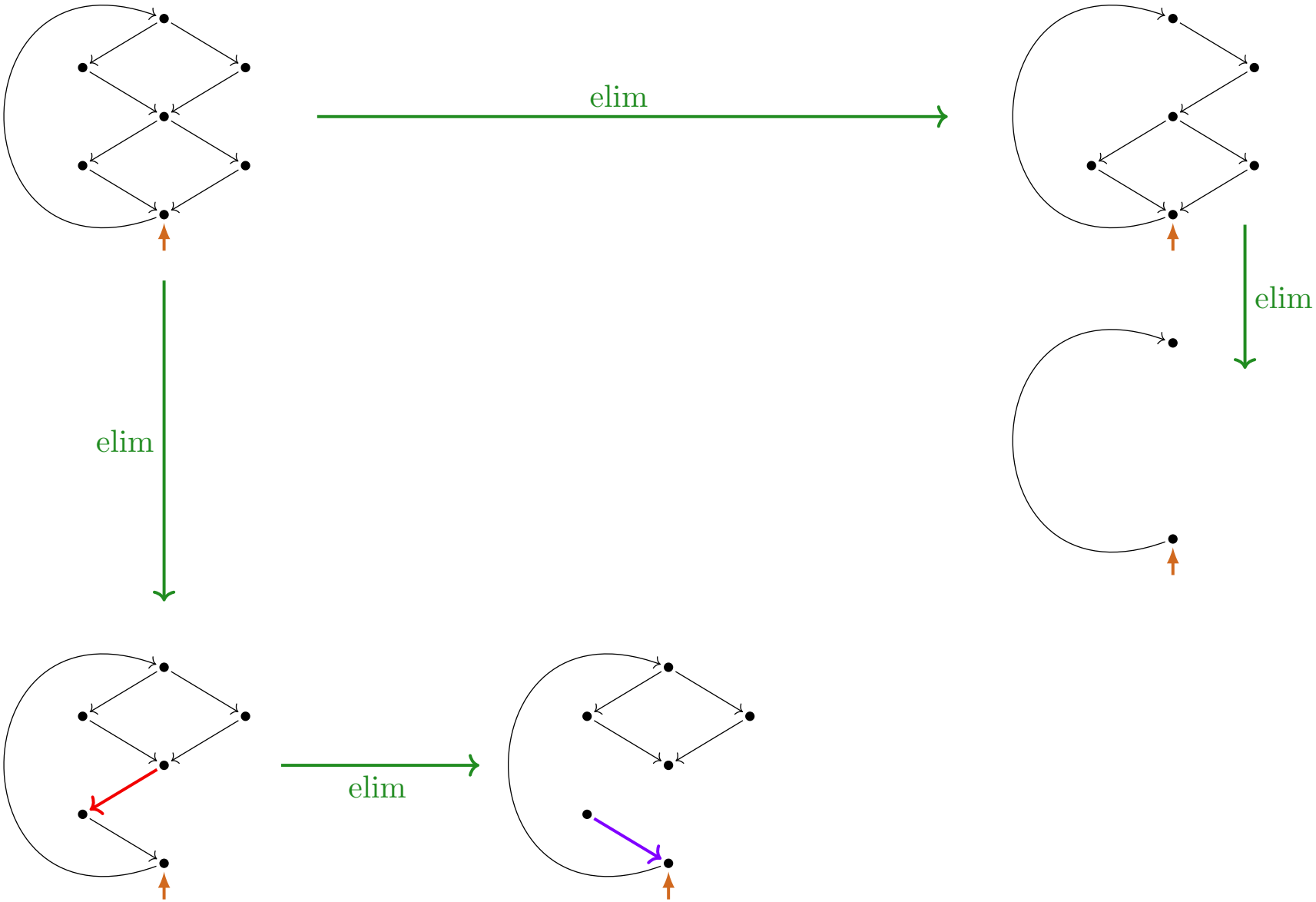
‘Critical pair’: bi-loop elimination



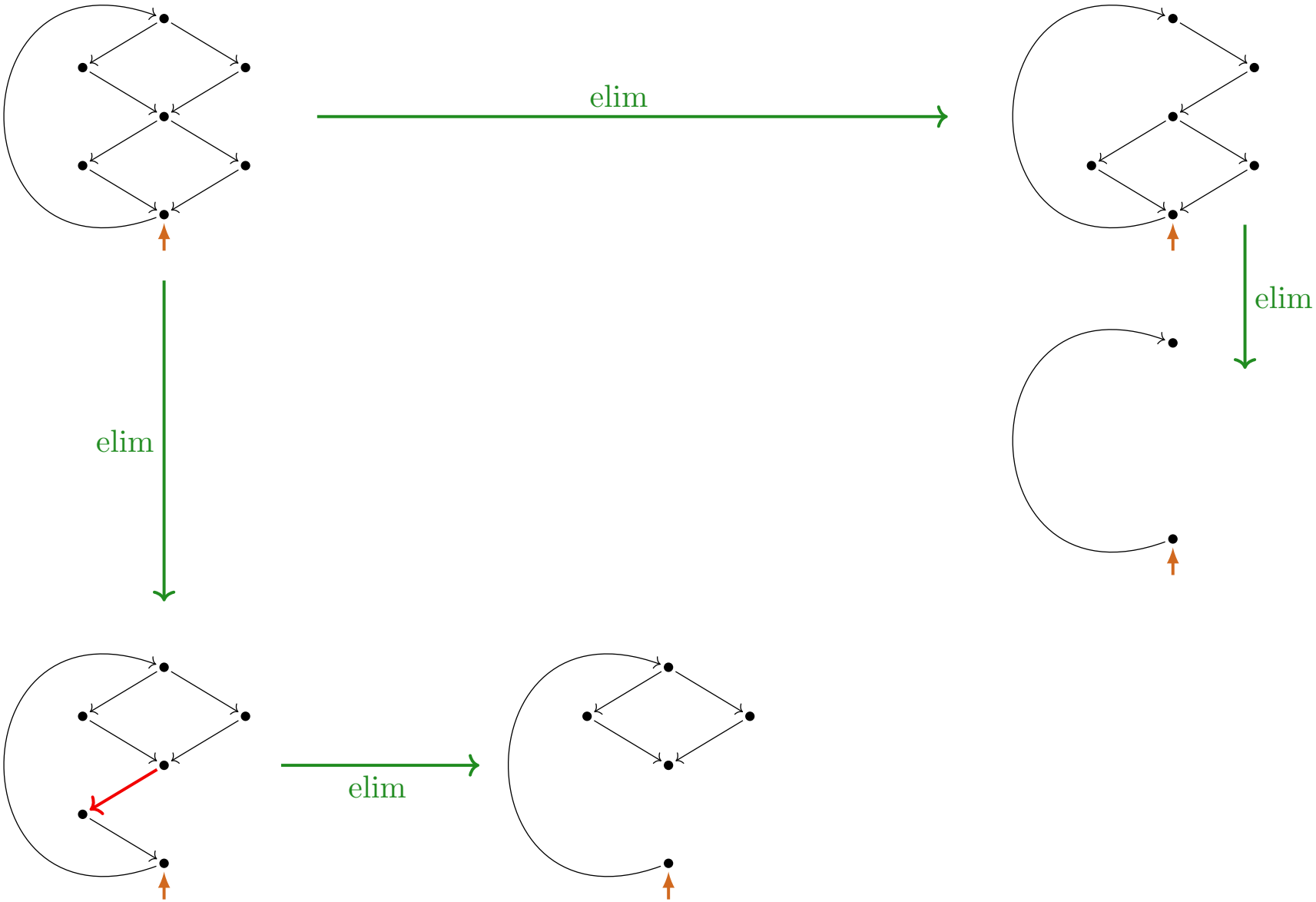
‘Critical pair’: bi-loop elimination



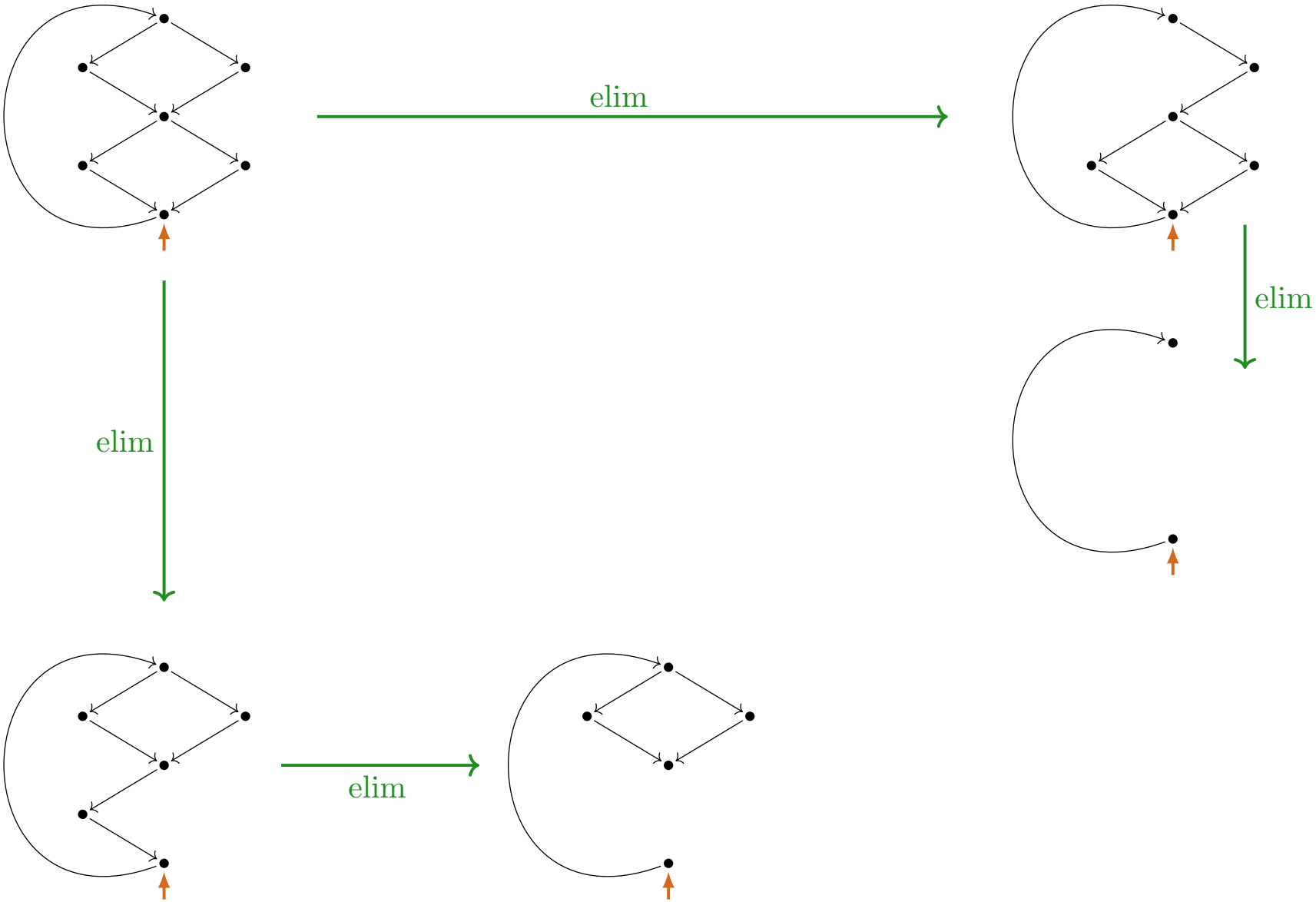
‘Critical pair’: bi-loop elimination



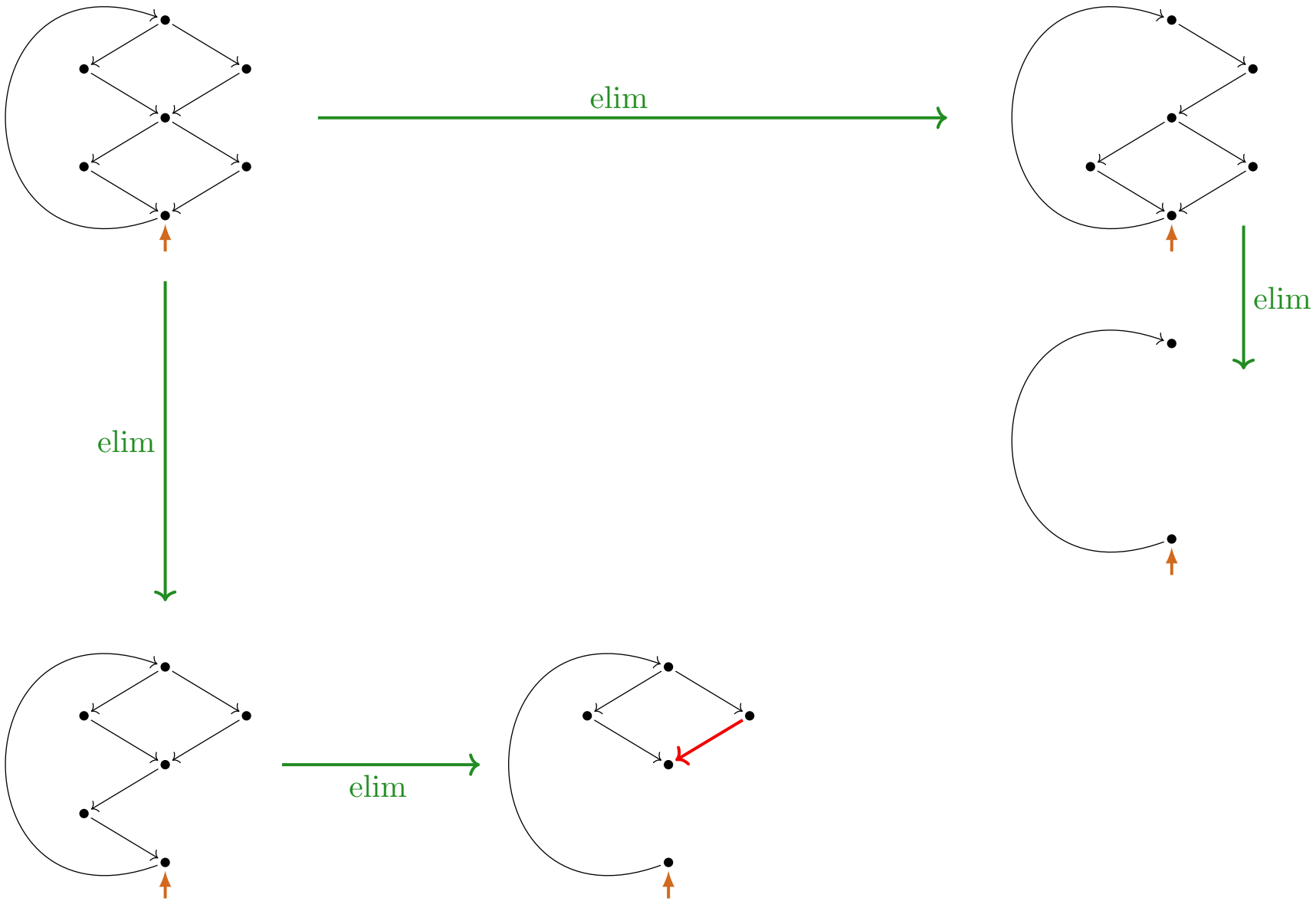
‘Critical pair’: bi-loop elimination



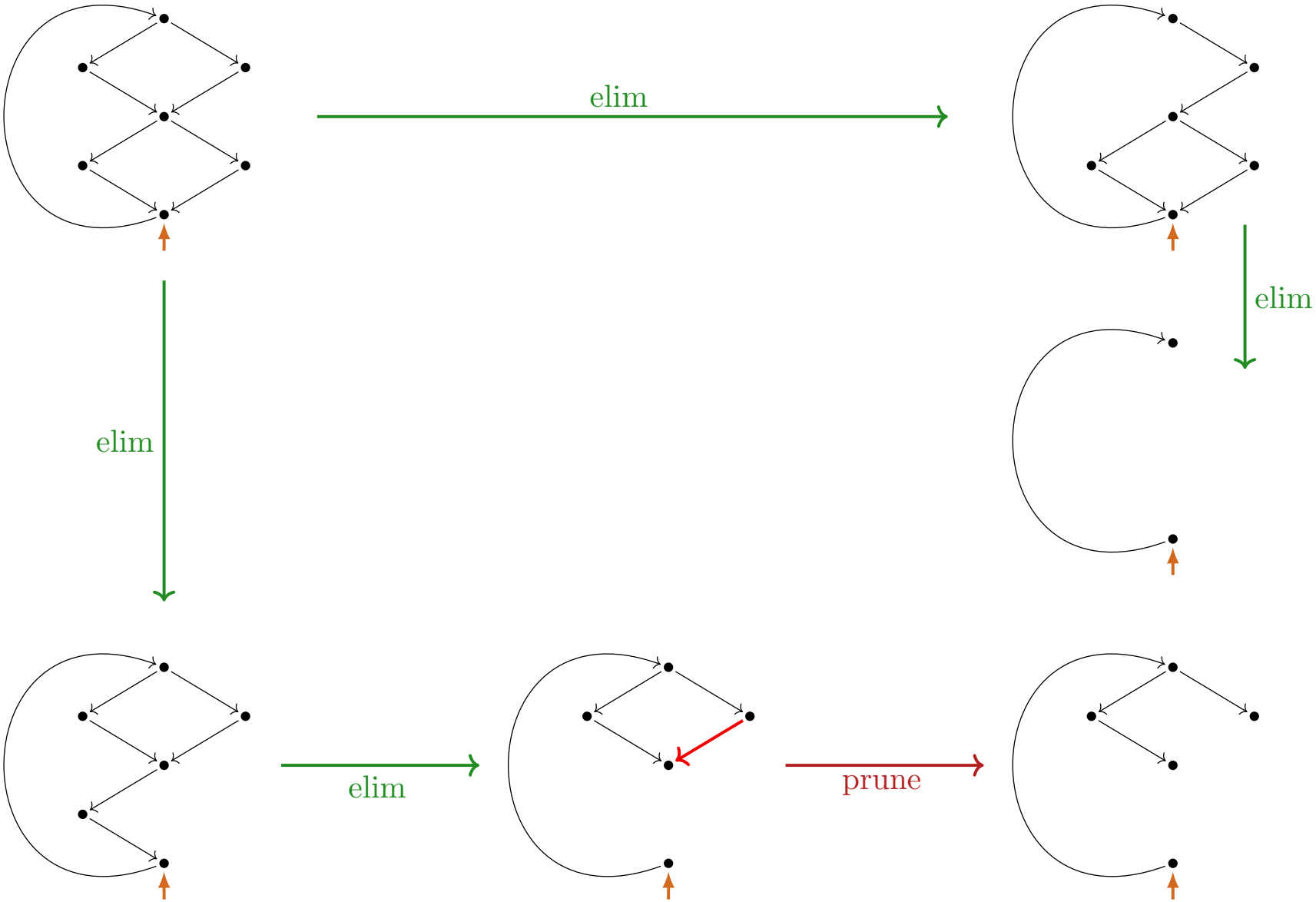
‘Critical pair’: bi-loop elimination



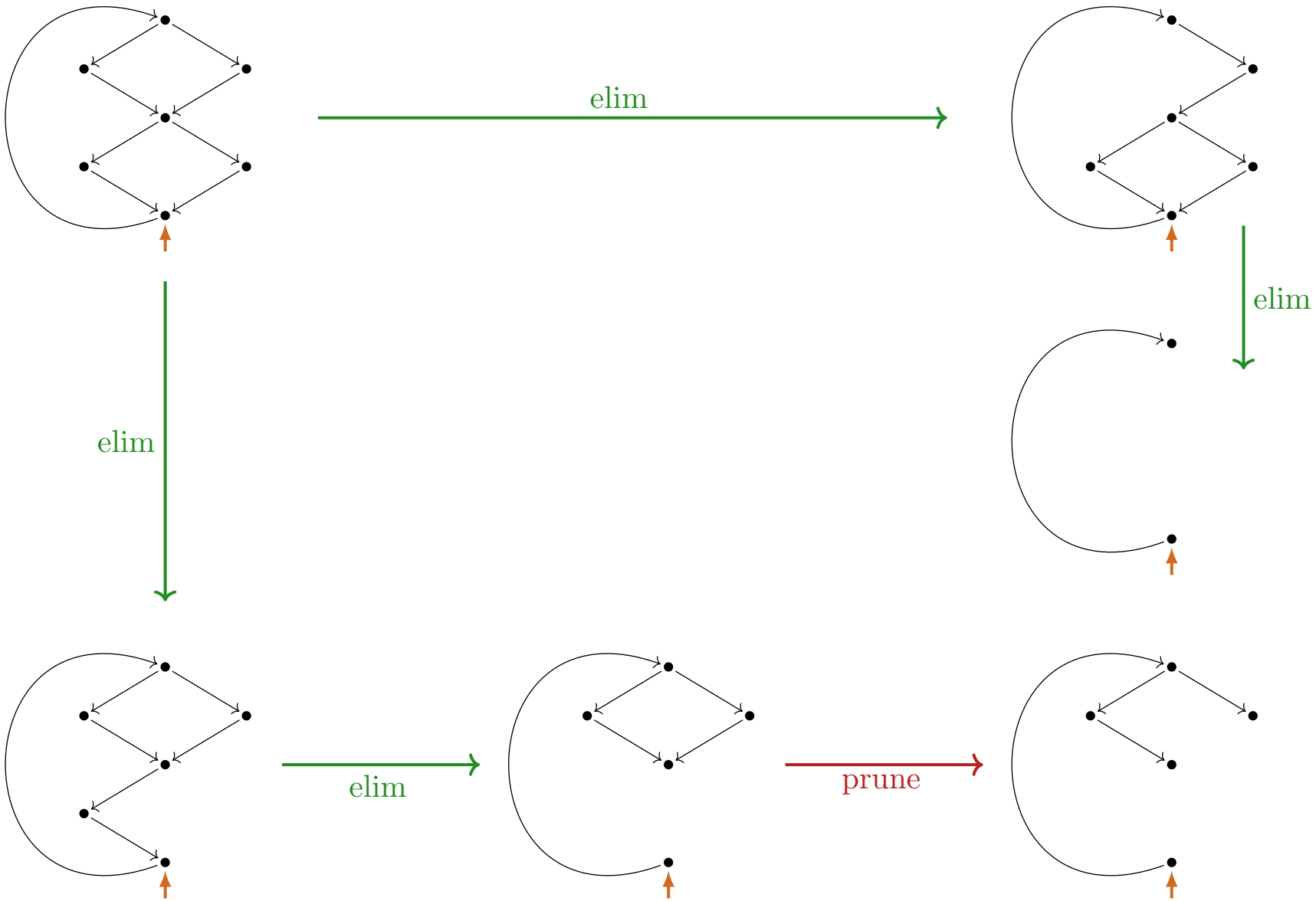
‘Critical pair’: bi-loop elimination



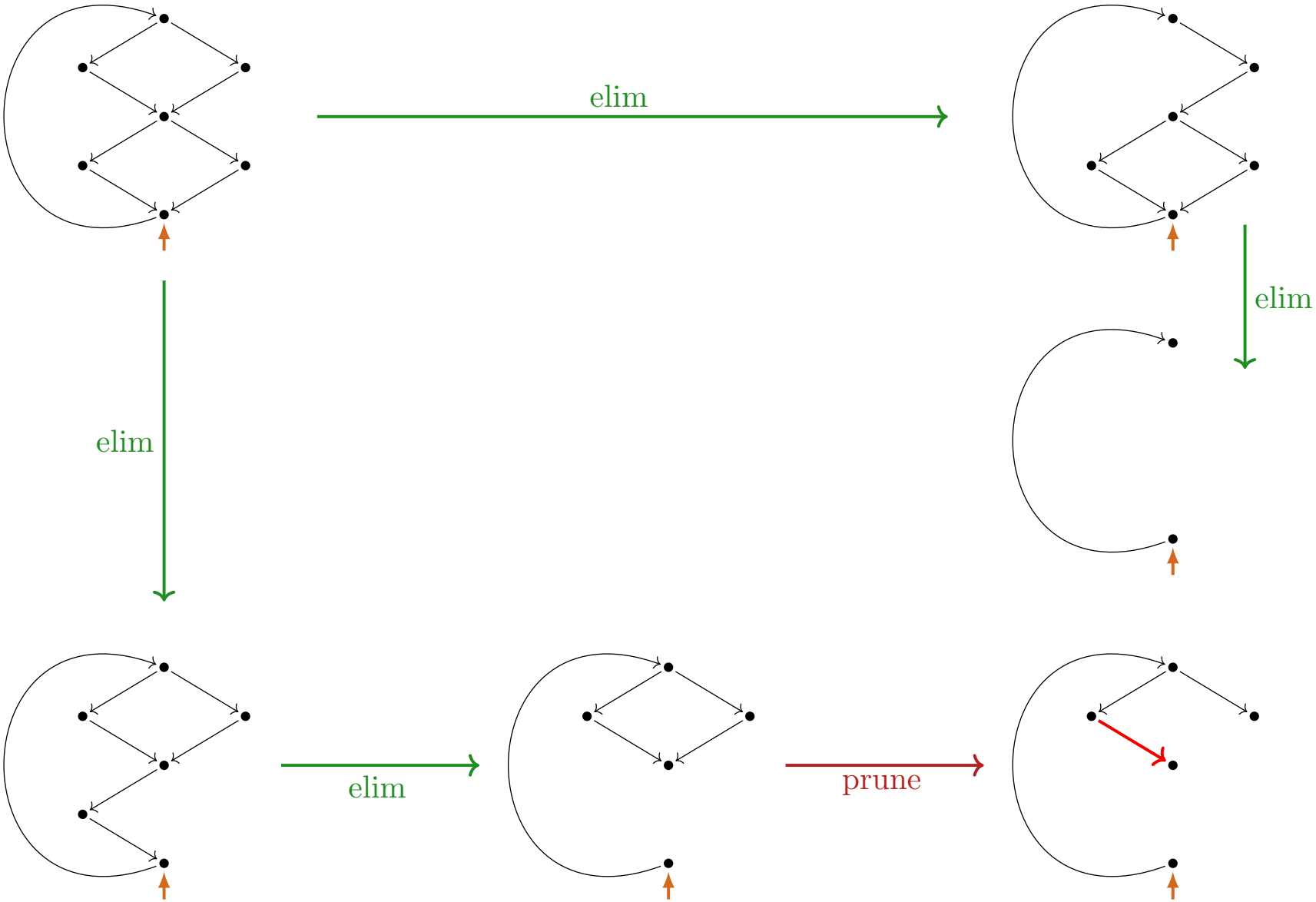
‘Critical pair’: bi-loop elimination



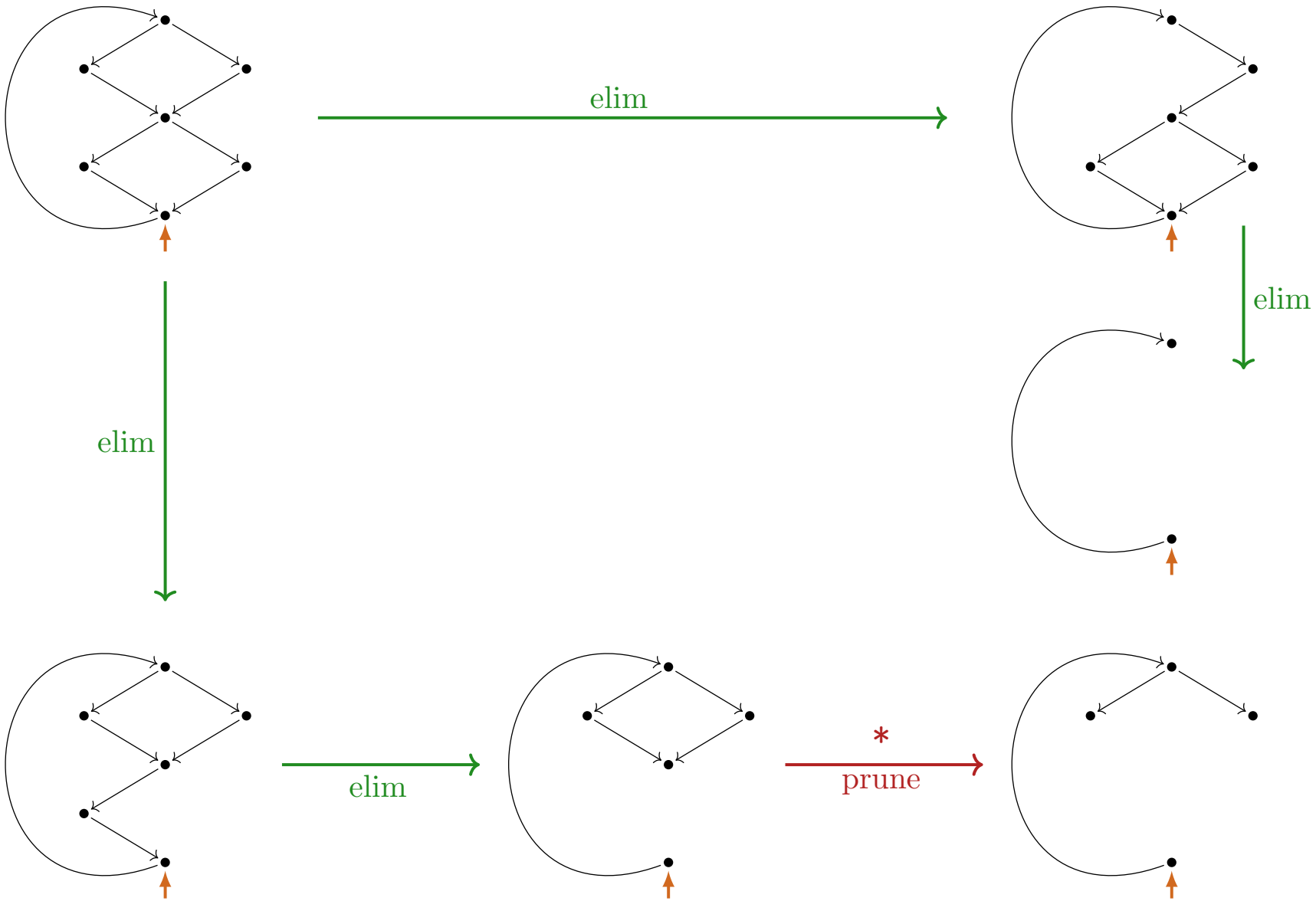
‘Critical pair’: bi-loop elimination



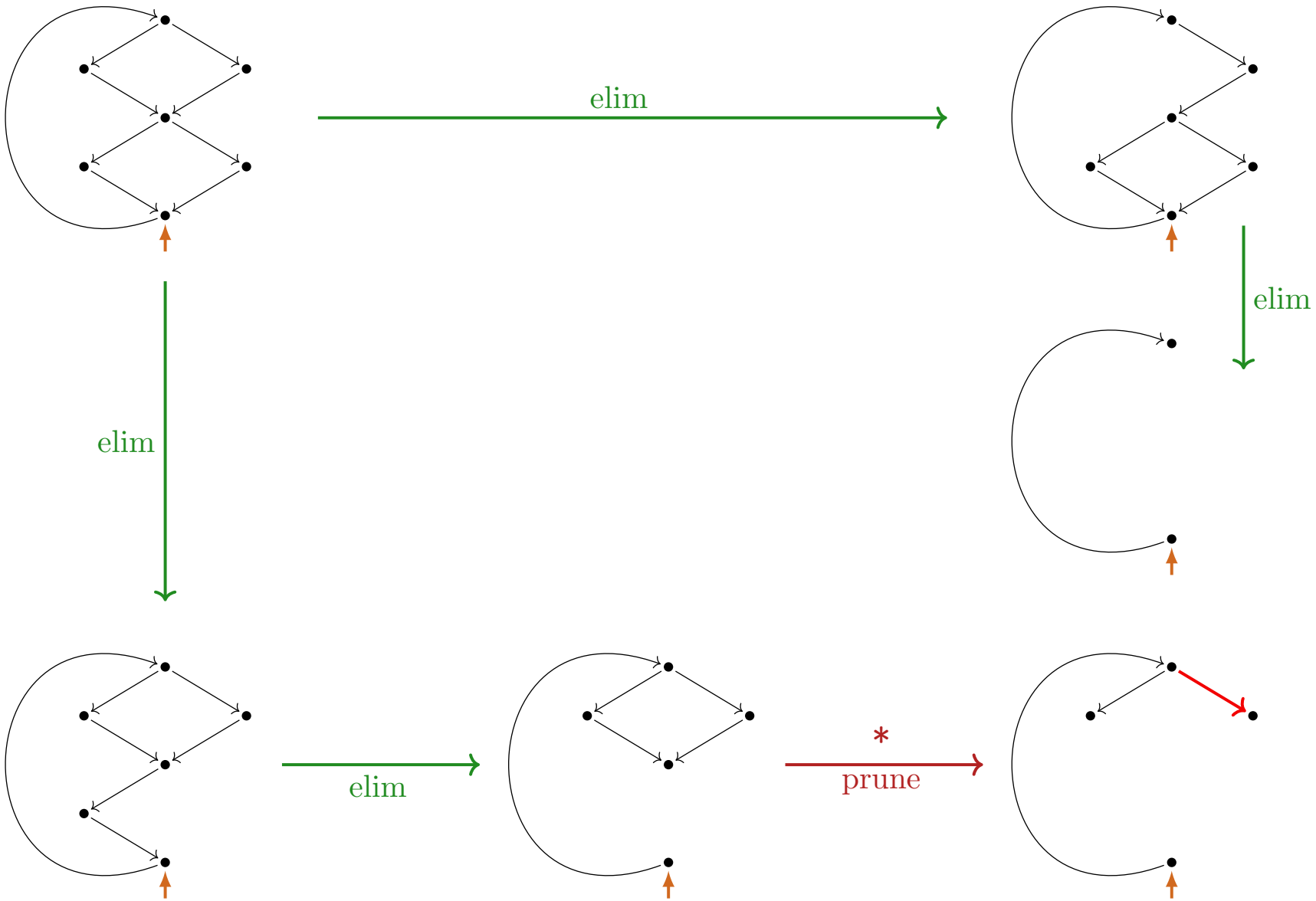
‘Critical pair’: bi-loop elimination



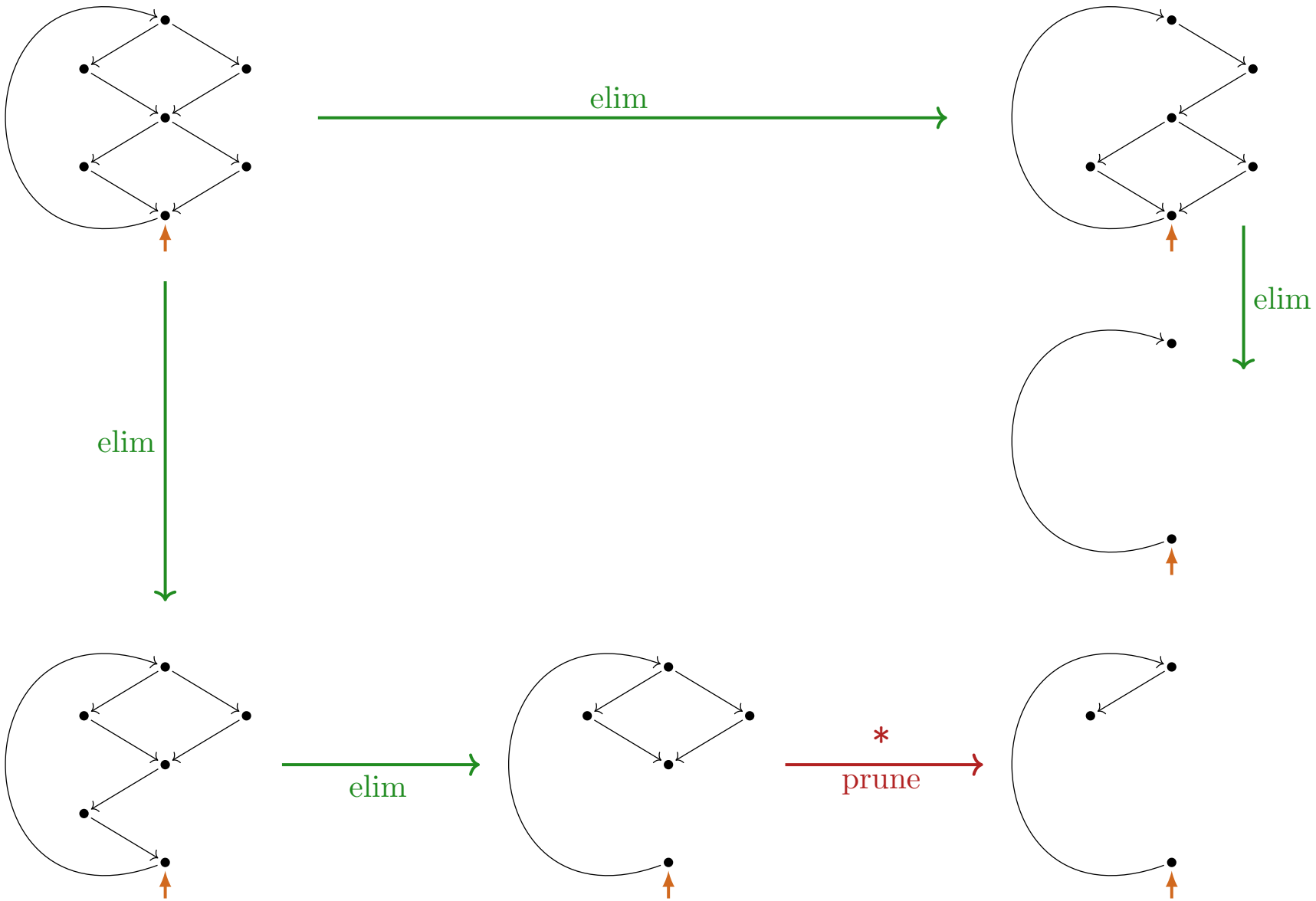
‘Critical pair’: bi-loop elimination



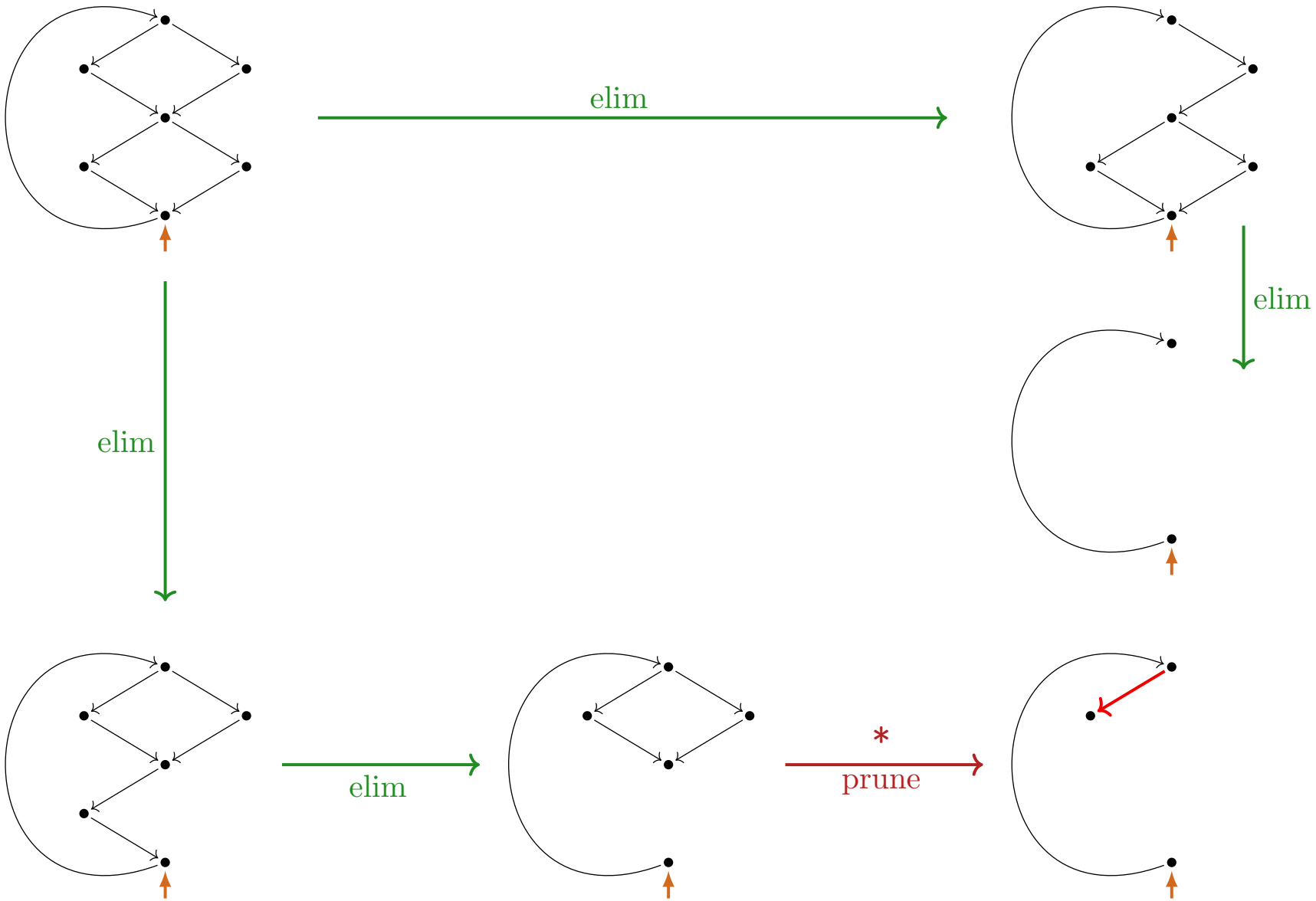
‘Critical pair’: bi-loop elimination



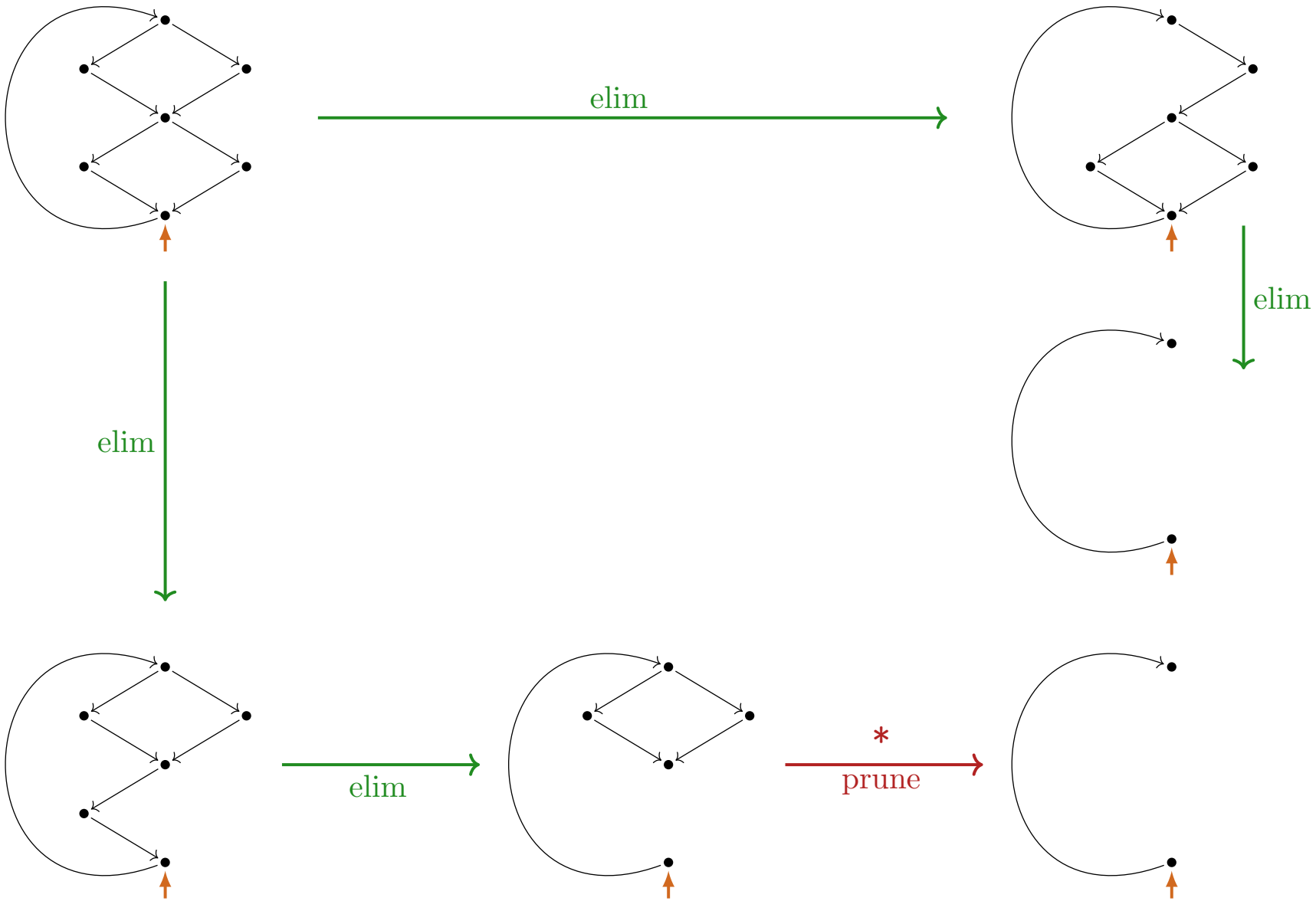
‘Critical pair’: bi-loop elimination



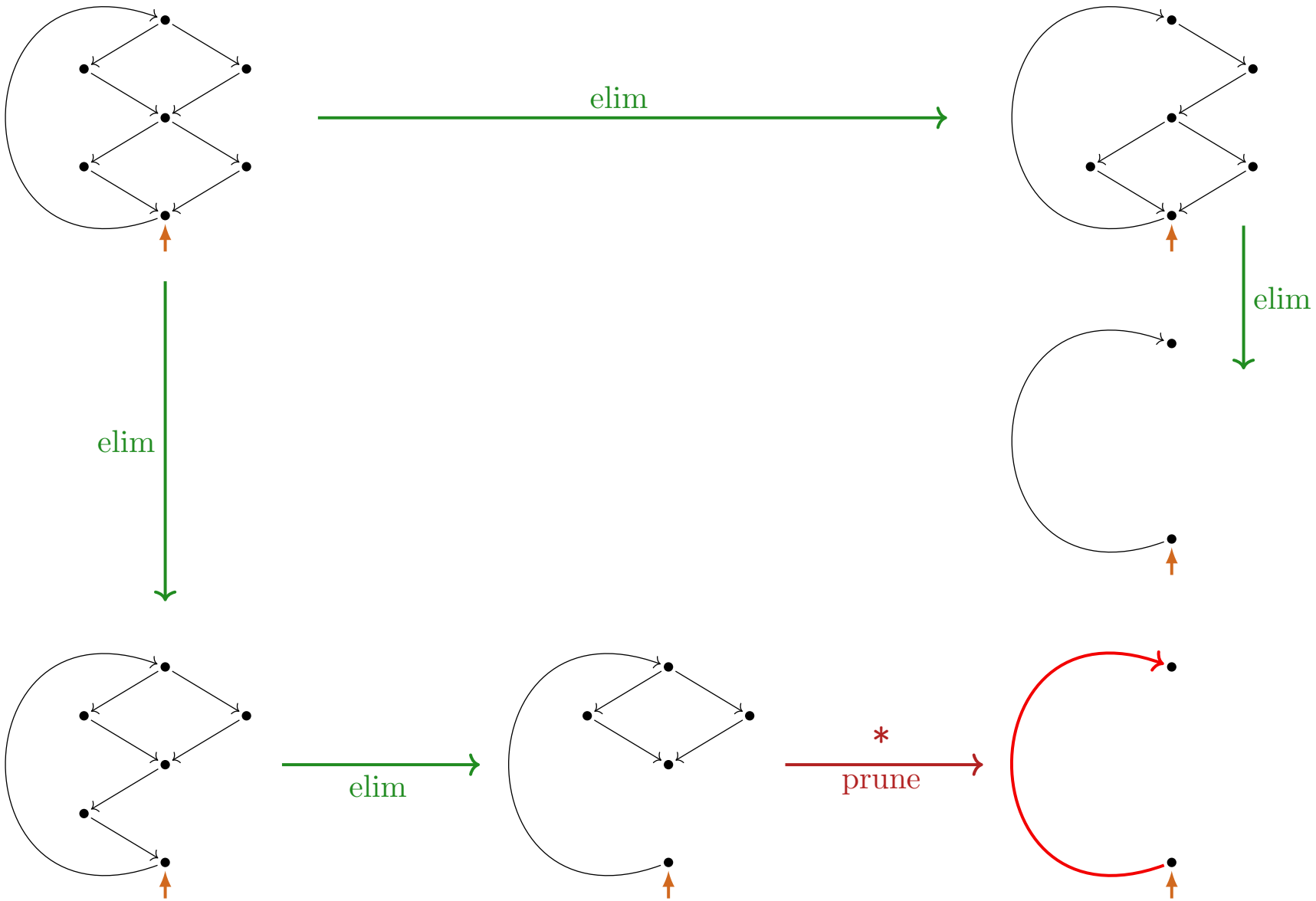
‘Critical pair’: bi-loop elimination



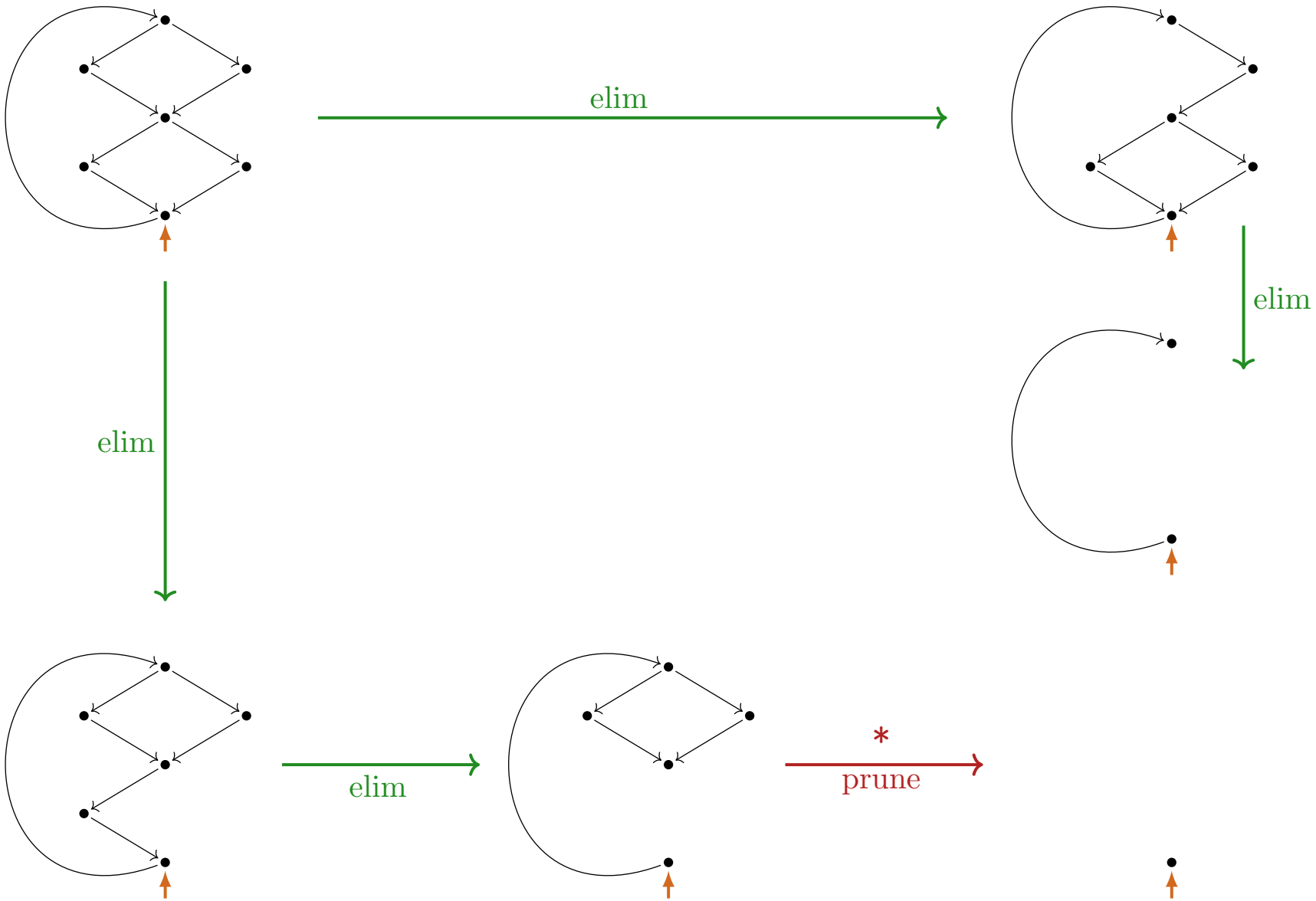
‘Critical pair’: bi-loop elimination



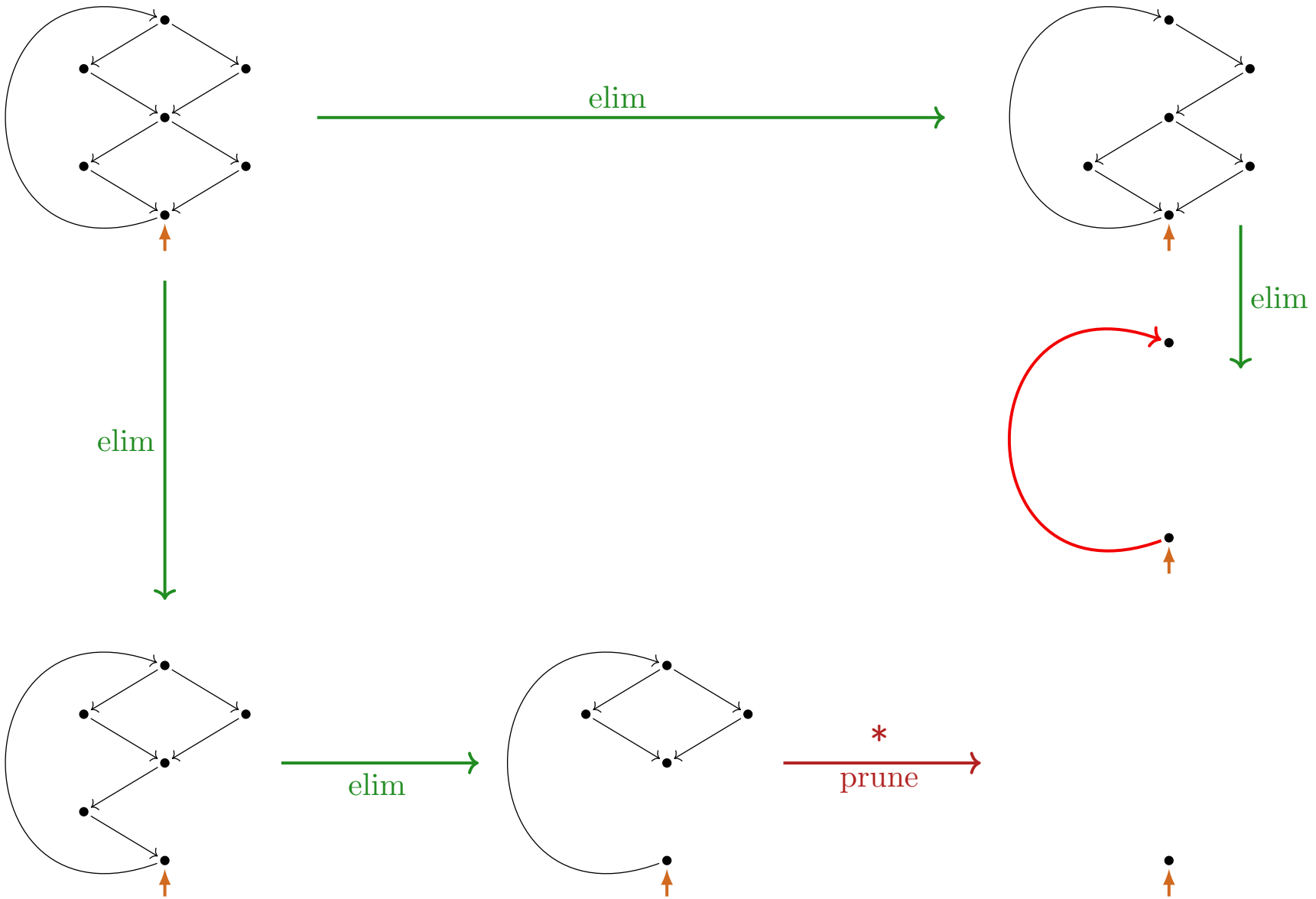
‘Critical pair’: bi-loop elimination



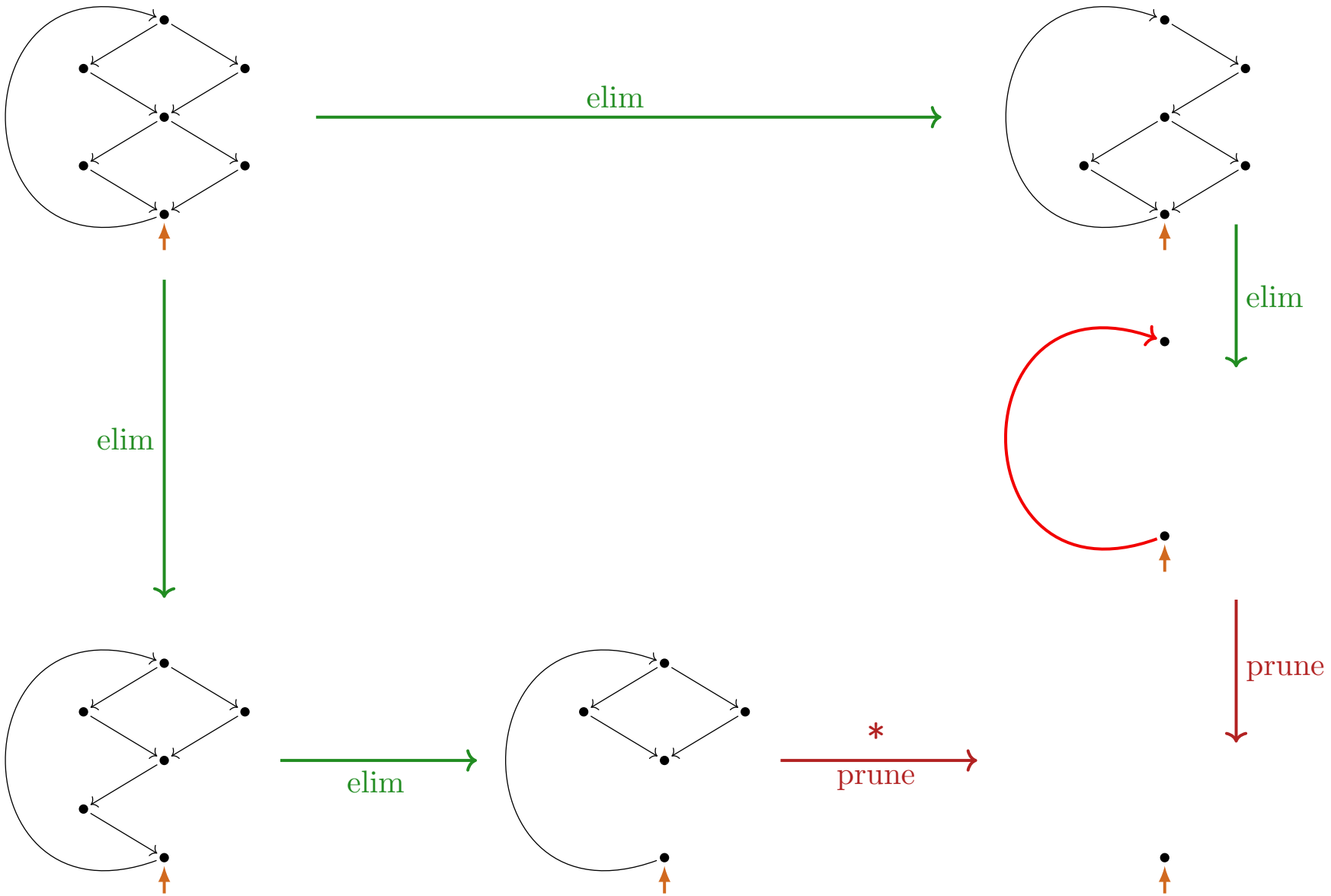
‘Critical pair’: bi-loop elimination



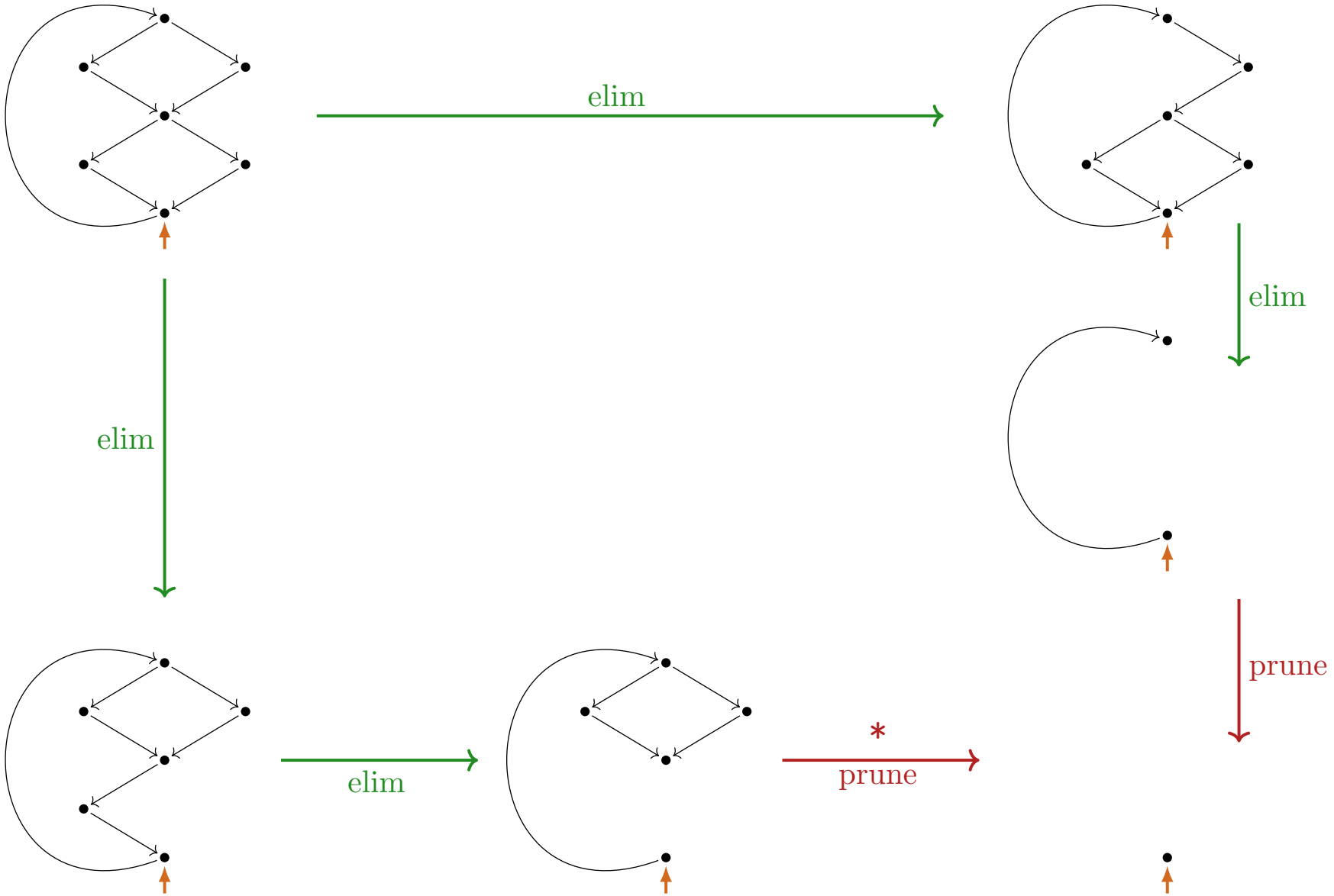
‘Critical pair’: bi-loop elimination



‘Critical pair’: bi-loop elimination



‘Critical pair’: bi-loop elimination



‘Critical pair’: bi-loop elimination