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

Clemens Grabmayer

Gran Sasso Science Institute,
L'Aquila, Abruzzo, Italy
clemens.grabmayer@gssi.it

**Abstract**

Process graphs that are interpretations of '1-free' regular expressions in Milner's process semantics for regular expressions have the Loop Existence and Elimination Property (LEE). Hereby a process graph satisfies LEE if the procedure of loop elimination, in which in every step a loop subgraph is decoupled by removing its entry transitions and then garbage collection is performed, terminates in a process graph without an infinite trace. Loop elimination in finite process graphs is terminating, but typically not confluent.

We explain that loop elimination can be turned into a confluent rewrite system on all process graphs by adding pruning steps that remove transitions to deadlocking states. For this purpose we perform a critical-pair like analysis that involves bi-loop subgraphs, and use the decreasing diagram method. This confluence result has two expedient consequences: for finite process graphs, LEE can be decided in polynomial time, and a layered version of LEE (no loops are eliminated from bodies of already eliminated ones) coincides with LEE.

We report on an aspect of work on the process semantics of regular expressions that concerns a procedure for analysing the structure of process graphs by decomposition. Formalising this procedure via rewrite relations helped us to clarify the situation. While the confluence result that we describe here does not require new techniques, it has some tangible consequences.

## 1 Introduction

Starting from a formalisation of regular (finite-state) processes as $\mu$-terms, Milner (1984, [7]) defined an interpretation $P$ of regular expressions $e$ as regular processes $P(e)$. The interpretations of 0, letters $a$, sums $e_1 + e_2$, products $e_1 \cdot e_2$, and iterations $e^*$ are as follows: $P(0)$ denotes deadlock, $P(a)$ performs a single action named "$a$" before terminating successfully, $P(e_1 + e_2)$ chooses to continue as $P(e_1)$ or $P(e_2)$, $P(e_1 \cdot e_2)$ executes $P(e_1)$, and upon its successful termination executes $P(e_2)$, and $P(e^*)$ iterates $P(e)$ (if it terminates successfully) arbitrarily often, but with the option to terminate successfully before each iteration. For $0^*$ we also use the constant 1 whose interpretation $P(1) = P(0^*)$ is the process that immediately terminates successfully. Based on this interpretation, Milner then defined the process semantics $[\![e]\!]_P$ of a regular expression $e$ as the bisimilarity equivalence class $[P(e)]_{\leftrightarrows}$ of its process interpretation $P(e)$.

Unlike for the language interpretation of regular expressions, in which every language that is accepted by a finite-state automaton is the interpretation of a regular expression, not every finite-state process is bisimilar to the process interpretation of a regular expression. There are finite process graphs that are neither $P$-expressible (the interpretation of a regular expression) nor $[\![\cdot]\!]_P$-expressible (bisimilar to a $P$-expressible graph). In order to recognise, and reason about $[\![\cdot]\!]_P$-expressible graphs it was therefore desirable to characterise the structure of at least the $P$-expressible graphs. For this purpose Fokkink and I formulated the Loop Existence and Elimination Condition LEE [6], which checks the structure of a process graph by loop-elimination decomposition steps. These steps remove infinite-trace 'loop subgraphs', which correspond

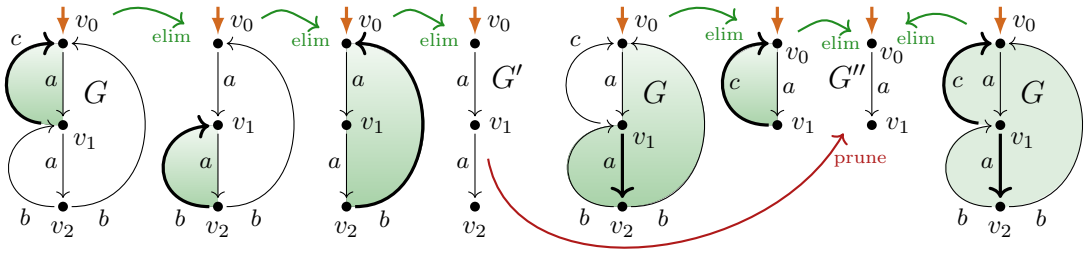


Figure 1: Three different $\to_{\mathrm{elim}}$ loop elimination sequences from the process interpretation $G = P(e)$ of the regular expression $e = ((1\cdot a)\cdot(c\cdot a + a\cdot(b+b\cdot a))^*)\cdot 0$. For every $\to_{\mathrm{elim}}$ step the eliminated loop subchart is indicated by drawing its loop-entry transition(s) in boldface, and colouring the area within its body. These rewrite sequences end in the normal form graphs $G'$ and $G''$ neither of which permits an infinite trace. Therefore $G = P(e)$ satisfies LEE. But as $G$ has two normal forms, $\to_{\mathrm{elim}}$ is not confluent. However, a $\to_{\mathrm{prune}}$ step can join the $\to_{\mathrm{elim}}$ fork. We note that none of the vertices of the graphs above permit immediate successful termination.

to interpretations $P(f^*)$ of iterations $f^*$ for expressions $f$ that are star-free and normed (i.e., $P(f)$ has a trace to successful termination). Loop subgraphs are peeled off from a given graph $G$ one by one, as long as such steps are possible. If a normal form graph without an infinite trace is found, then LEE holds for $G$, and otherwise it fails. See Fig. 1 for a graph $G$ that satisfies LEE. It can be shown (see [5]) that LEE characterises process interpretations of 'under-star-1-free' regular expressions (with restricted use of iterations within iterations), and that a generalisation 1-LEE [3] for graphs with 1-transitions (for empty steps) characterises process interpretations. In Sect. 2 we recapitulate the loop-elimination rewrite relation $\to_{\mathrm{elim}}$, and the property LEE.

Loop elimination steps can typically be performed in several ways, thereby 'parsing' a given $P$-expressible graph $G$ differently while intuitively synthesising from $G$ different regular expressions $e$ with $P(e) = G$ or $P(e) \leftrightarroweq G$. Typically $\to_{\mathrm{elim}}$ steps are not confluent, see Fig. 1 for an example. Here we report that a confluent rewrite system can be obtained by adding $\to_{\mathrm{prune}}$ steps that remove transitions to deadlock states. Such transitions cannot be part of infinite traces enabled by loop subgraphs, and therefore they do not hinder the applicability of $\to_{\mathrm{elim}}$ steps.

In Sect. 3 we give an outline for why $\to_{\mathrm{elim}}$ and $\to_{\mathrm{prune}}$ steps form a decreasing abstract rewriting system. For this purpose we focus attention on a lemma concerning $\to_{\mathrm{elim}}$ forks. The decreasing diagram method by van Oostrom [10, 9, 11] and de Bruijn then guarantees that $\to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ is confluent. Finally in Sect. 4 we describe two significant consequences of this result. First, that LEE can be decided in polynomial time, and second, that a restriction of $\to_{\mathrm{elim}}$ for which only such loop subgraphs are eliminated whose start vertices are not in the body of previously eliminated loops defines a specialisation LLEE of LEE that is equivalent with LEE.

The exposition here is supplemented in Appendix A with more details and some of the proofs.

## 2 Loop Elimination and Pruning

A *(process) graph (with an immediate termination predicate)* is a tuple $G = \langle V, A, v_{\mathrm{s}}, \to, \Downarrow\rangle$ that consists of a set $V$ of *vertices*, a set $A$ of *actions*, a start vertex $v_{\mathrm{s}} \in V$ (hence $V \neq \varnothing$), a *(labelled) transition relation* $\to \subseteq V \times A \times V$, and a predicate $\Downarrow \subseteq V$ on the states that indicates immediate termination. We write $v_1 \xrightarrow{a} v_2$ for a transition $\langle v_1, a, v_2\rangle \in \to$, and $v\Downarrow$ for a state $v \in \Downarrow$ with immediate termination permitted, as well as $v\not\downarrow$ if $v \notin \Downarrow$. We say that a vertex $v \in V$ is *start-vertex connected* if there is a path from $v_{\mathrm{s}}$ to $v$. We usually assume, with the exception

of the definition of rewrite steps, graphs to be such that all vertices are start-vertex connected.

Let $G = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph. We define the result of *garbage collection in* $G$ based on the subset $V_0 \subseteq V$ of vertices of $G$ that are start-vertex connected as $\mathsf{gc}(G) := \langle V_0, A, v_s, \to \cap (V_0 \times A \times V_0), \Downarrow \cap V_0\rangle$. Now let $v \in V$. By $G{\downarrow_*^v} := \mathsf{gc}(\langle V, A, v, \to, \Downarrow\rangle)$ we define the *subgraph of* $G$ *at* $v$ that results from $G$ by changing the start vertex to $v$, and then performing garbage collection. Let again $v \in V$, and additionally $T \subseteq \langle v\to\rangle := \{\langle v, a, w\rangle \mid \langle v, a, w\rangle \in \to\}$ be a set of transitions from $v$ in $G$. We define by $G{\downarrow_*^{v,T}} := \langle V_0, A, v, T \cup ((V_0 \setminus \{v\}) \times A \times V_0), \Downarrow \cap V_0\rangle$ the *generated subgraph of* $G$ *from/until* $v$ *with respect to* $T$ where $V_0 \subseteq V$ is the set of vertices on traces in $G$ that start in $v$, take a transition from $T$, and then continue onwards as long as possible but stop whenever $v$ is reached again.

**Definition 2.1** (loop graphs, loop subgraphs, bi-loops)**.** Let $G = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph.

We say that $G$ is a *loop (process graph)* if it satisfies the following three conditions:

(L1) There is an infinite trace from the start vertex $v_s$ of $G$ (denoted by $\infty(v_s)$ and by $\infty(G)$).

(L2) Every infinite trace from the start vertex $v_s$ of $G$ returns to $v_s$.

(L3) Immediate termination is only permitted at the start vertex of $G$, i.e. $v\Downarrow$ only if $v = v_s$.

Then we call transitions from $v_s$ *loop-entry transitions*, and the other ones *loop-body transitions*.

By a *loop subgraph* of $G$ (where $G$ does not have to be a loop itself) we mean a generated subgraph $G{\downarrow_*^{v,T}}$ of $G$ from/until $v \in V$ with respect to $T \subseteq \langle v\to\rangle$ such that $G{\downarrow_*^{v,T}}$ is a loop.

Let $v_1, v_2 \in V$. We say that $G$ is a *bi-loop with respect to* $v_1, v_2$ if $v_s = v_1 \neq v_2$, and $G$ is a start-vertex connected loop chart that remains a start-vertex connected loop chart if its start vertex is changed to $v_2$, that is, for each $i \in \{1, 2\}$, $G{\downarrow_*^{v_i}} = \langle V, A, v_i, \to, \Downarrow\rangle$ (note the absence of garbage collection) is a start-vertex connected loop chart.

Note that for a bi-loop $G$ with respect to $v_1, v_2$ it holds that $G = G{\downarrow_*^{v_1}}$, $v_2$ is reachable from $v_1$, and $v_1$ is reachable from $v_2$, in each of $G = G{\downarrow_*^{v_1}}$ and $G{\downarrow_*^{v_2}}$. The following lemma states that bi-loops can also be defined by three conditions similar to the loop chart conditions (L1)–(L3).

**Lemma 2.2.** *Let $B = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph, and let $v_1, v_2 \in V$ be such that $v_1 \neq v_2$.*

*Then $B$ is a bi-loop with respect to $v_1, v_2$ if and only if the following specialised variant of the loop conditions (L1)–(L3) hold:* (B1) *there is an infinite trace from $v_1$,* (B2) *every infinite trace from $v_1$ visits $v_2$ before it visits $v_1$ again, and every infinite trace from $v_2$ visits $v_1$ before it visits $v_2$ again,* (B3) *no vertex of $B$ permits immediate termination, i.e. $v\not\Downarrow$ for all $v \in V$.*

For defining decoupling $\mathsf{dcpl}(G, T)$ of sets $T$ of transitions from $G$, and garbage collection $\mathsf{gc}(G)$ in $G$, we let $G = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph.

For a subset $T \subseteq \to$ of transitions of $G$, we define the (result of) *decoupling of* $T$ *from* $G$ as the graph $\mathsf{dcpl}(G, T) := \langle V, A, v_s, \to \setminus T, \Downarrow\rangle$ that results from $G$ by removing the transitions in $T$.

**Definition 2.3** (loop elimination $\to_{\mathrm{elim}}$, pruning $\to_{\mathrm{prune}}$, union $\to_{\mathrm{ep}}$)**.** We define loop elimination steps $\to_{\mathrm{elim}}$, and pruning steps $\to_{\mathrm{prune}}$ between graphs $G$ and $G'$ as follows:

$$\begin{aligned} G \to_{\mathrm{elim}} G' \;&:\Longleftrightarrow\; \exists v \in V\, \exists T \subseteq \langle v\to\rangle \big[\, G{\downarrow_*^{v,T}} \text{ is a loop} \\ &\qquad\qquad \wedge\; G' = \mathsf{gc}(\mathsf{dcpl}(G, T))\,\big], \\ \text{(where } G = \langle V, A, v_s, \to, \Downarrow\rangle\text{)} \end{aligned}$$

$$\begin{aligned} G \to_{\mathrm{prune}} G' \;&:\Longleftrightarrow\; \exists t = \langle v, a, v'\rangle \in \to \big[\, \langle v'\to\rangle = \varnothing \;\wedge\; v'\not\Downarrow \\ &\qquad\qquad \wedge\; G' = \mathsf{gc}(\mathsf{dcpl}(G, \{t\}))\,\big]. \\ \text{(where } G = \langle V, A, v_s, \to, \Downarrow\rangle\text{)} \end{aligned}$$

By an $\to_{\mathrm{ep}}$ step we mean an $\to_{\mathrm{elim}}$ step or a $\to_{\mathrm{prune}}$ step. We will also use $\to_{\mathrm{elim}}$, $\to_{\mathrm{prune}}$, and $\to_{\mathrm{ep}}$ as suggestive notation for an(y) abstract rewriting system $\langle \mathcal{G}, \to\rangle$ with $\mathcal{G}$ a set of graphs that is closed under steps $\to$, and with rewrite relation $\to \in \{\to_{\mathrm{elim}}, \to_{\mathrm{prune}}, \to_{\mathrm{ep}}\}$.

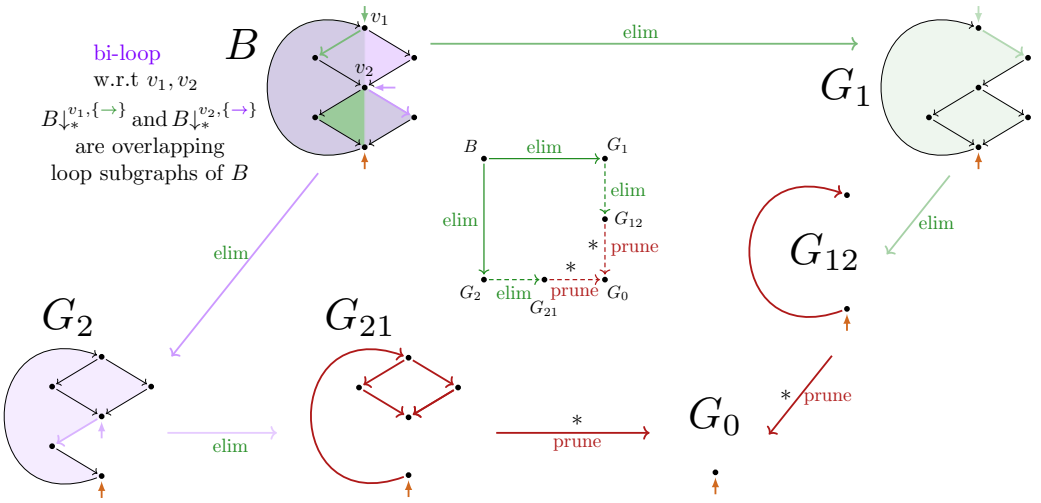


Figure 2: Prototypical bi-loop example justifying the elementary diagram (in the middle) for joining an elimination step fork $G_2 \leftarrow_{\text{elim}} B \rightarrow_{\text{elim}} G_1$ in the critical-pair like case of eliminating loop subgraphs with overlapping cyclic parts. The fork is joined by $\rightarrow_{\text{elim}}$ steps, and $\rightarrow_{\text{prune}}$ steps that 'eat up' the remaining bi-loop body: $G_2 \rightarrow_{\text{elim}} G_{21} \rightarrow^*_{\text{prune}} G_0 \leftarrow^*_{\text{prune}} G_{12} \leftarrow_{\text{elim}} G_1$.

**Example 2.4.** In Fig. 1 we illustrate, for a graph $G$, three maximal $\rightarrow_{\text{elim}}$ rewrite sequences, and a $\rightarrow_{\text{prune}}$ step that links the obtained $\rightarrow_{\text{elim}}$ normal forms. For each $\rightarrow_{\text{elim}}$ step the set $T$ of loop-entry transitions that define the decoupled loop subchart is indicated in boldface. Two $\rightarrow_{\text{elim}}$ steps from a bi-loop with respect to overlapping loop subgraphs are depicted in Fig. 2.

**Proposition 2.5.** $\rightarrow_{\text{elim}}$ *is not confluent.* (See Fig. 1 for a counterexample to confluence.)

**Definition 2.6** (LEE)**.** We say that a process graph $G$ satisfies the *loop existence and elimination property* LEE, denoted by LEE($G$), if $G$ has an $\rightarrow_{\text{elim}}$ normal form graph that does not enable an infinite trace. More formally, we stipulate:

$$\text{LEE}(G) \;:\Longleftrightarrow\; \exists G_0 \big( (\, G \rightarrow^*_{\text{elim}} G_0 \not\rightarrow_{\text{elim}}) \;\wedge\; \neg\infty(G_0) \big)\,.$$

**Lemma 2.7.** *On finite process graphs,* $\rightarrow_{\text{ep}} = \rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$ *is terminating.*

# 3 Loop Elimination with Pruning is Confluent

While the motivation for loop elimination stems from finite process graphs, we prove confluence of $\rightarrow_{\text{ep}}$ for all process graphs, including infinite ones. In order to obtain confluence of $\rightarrow_{\text{ep}}$ on only finite graphs it would suffice to show local confluence of $\rightarrow_{\text{ep}}$ (follows from Lem. 3.2 below), because Newman's lemma can be applied due to termination of $\rightarrow_{\text{ep}}$ in that case (Lem. 2.7). But we obtain confluence of $\rightarrow_{\text{ep}}$ on all graphs by showing that $\rightarrow_{\text{ep}} = \rightarrow_{\text{elim}} \cup \rightarrow_{\text{prune}}$ is decreasing, and by appealing to the Decreasing Diagram Theorem [10, 9]. Only the case of finding elementary diagrams for $\rightarrow_{\text{elim}}$ forks is non-trivial. We formulate that case as Lem. 3.1 below. It bears a similarity with critical pair lemmas in term rewriting theory. For the crucial case of eliminating non-root overlapping loop subgraphs we illustrate a typical example in Fig. 2.

**Lemma 3.1** (joining $\rightarrow_{\text{elim}}$ forks)**.** *Let* $G_2 \leftarrow_{\text{elim}} G \rightarrow_{\text{elim}} G_1$ *be a fork of* $\rightarrow_{\text{elim}}$ *steps in which the loop subgraphs* $G\downarrow_*^{v_2,T_2}$*, and* $G\downarrow_*^{v_1,T_1}$ *of* $G$ *are eliminated, respectively.*

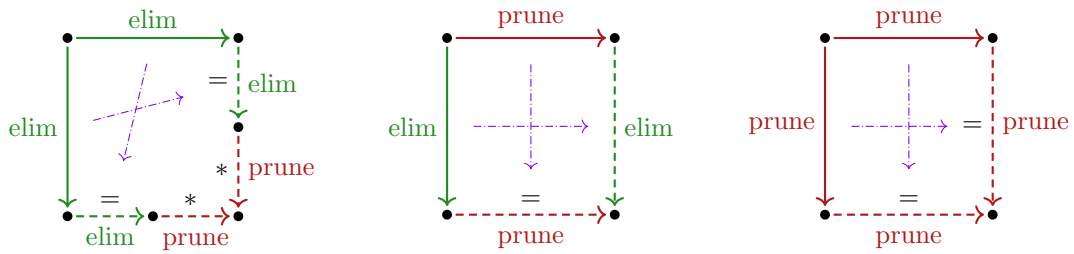


Figure 3: Elementary diagrams for showing that $\langle \mathcal{G}, \{\to_\ell\}_{\ell\in\{\mathrm{elim},\mathrm{prune}\}}\rangle$, for a set $\mathcal{G}$ of graphs that is closed under $\to_{\mathrm{elim}}$ and $\to_{\mathrm{prune}}$, is an abstract rewriting system that is decreasing with respect to precedence order prune $<$ elim. That these elementary diagrams are decreasing is indicated by the dash-dotted trace relation arrows.

*Then this $\to_{\mathrm{elim}}$ fork can be joined appropriately with $\to_{\mathrm{prune}}$ steps towards a graph $G_0$ in the following three mutually exclusive, and exhaustive situations:*

*(i)* Root-overlap*: $v_1 = v_2$, and both loops are contained in the 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: $G_2 \to_{\mathrm{elim}}^{=} \cdot \to_{\mathrm{prune}}^{*} G_0 \leftarrow_{\mathrm{prune}}^{*} \cdot \leftarrow_{\mathrm{elim}}^{=} G_1$.*

*(ii)* Parallel/nested*: $v_1 \neq v_2$, and the start-vertex connected cyclic parts $\circlearrowright(G{\downarrow_*^{v_1,T_1}})$ and $\circlearrowright(G{\downarrow_*^{v_2,T_2}})$ of the loops are vertex-disjoint. Then the $\to_{\mathrm{elim}}$ 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: $G_2 \to_{\mathrm{elim}}^{=} G_0 \leftarrow_{\mathrm{elim}}^{=} G_1$.*

*(iii)* Non-root overlap *(see Fig. 2): $v_1 \neq v_2$, the start-vertex connected cyclic parts $\circlearrowright(G{\downarrow_*^{v_1,T_1}})$, and $\circlearrowright(G{\downarrow_*^{v_2,T_2}})$ have common vertices. The two loops together form a bi-loop w.r.t. $v_1, v_2$. By removing possibly remaining loop-entries at $v_2$ after the step $G \to_{\mathrm{elim}} G_2$, and at $v_1$ after the step $G \to_{\mathrm{elim}} G_1$, and then removing all transitions of the bi-loop with $\to_{\mathrm{prune}}$ steps, the fork can always be joined as follows: $G_2 \to_{\mathrm{elim}}^{=} \cdot \to_{\mathrm{prune}}^{*} G_0 \leftarrow_{\mathrm{prune}}^{*} \cdot \leftarrow_{\mathrm{elim}}^{=} G_1$.*

**Lemma 3.2.** *$\to_{\mathrm{ep}}$ is (locally) decreasing with respect to the family $\{\to_\ell\}_{\ell\in\{\mathrm{elim},\mathrm{prune}\}}$ of rewrite relations, with the precedence order prune $<$ elim.*

*Proof.* In Fig. 3 we illustrate the full set of elementary diagrams for the abstract rewriting system with the family $\{\to_\ell\}_{\ell\in\{\mathrm{elim},\mathrm{prune}\}}$ of rewriting relations.

The leftmost elementary diagram follows from the analysis of the three possible, mutually exclusive cases for $\to_{\mathrm{elim}}$ forks in Lem. 3.1. It describes the joining steps in the cases (i) and (iii) in Lem. 3.1, encompassing the situation in easier case (ii). The second elementary diagram in Fig. 3 concerning joining $\leftarrow_{\mathrm{prune}} \cdot \to_{\mathrm{elim}}$ forks by subcommutation of the two involved steps is due to the fact that transitions to deadlock states which are removed in $\to_{\mathrm{prune}}$ steps cannot be part of the infinite traces in the cyclic parts of loop subgraphs. The third elementary diagram in Fig. 3 expresses commutation of $\to_{\mathrm{prune}}$ steps: if the two $\to_{\mathrm{prune}}$ steps coincide, then they have the same targets, which are joined by empty steps; if the steps are different, then they commute.

All of these diagrams are decreasing with respect to the precedence order prune $<$ elim of reduction labels, as witnessed by the trace relation indicated by arrows in Fig. 3. □

**Theorem 3.3.** *$\to_{\mathrm{ep}} = \to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ is confluent.*

*Proof.* From the fact that $\to_{\mathrm{ep}}$ is decreasing by Lem. 3.2 it follows by the Decreasing Diagrams Theorem (see [10, Cor. 3.9], [9, Thm. 14.2.8], [11, Thm. 1]) that $\to_{\mathrm{ep}}$ is confluent. □

**Corollary 3.4.** *When restricted to finite graphs,* $\to_{\mathrm{ep}} = \to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ *is complete (that is,* $\to_{\mathrm{ep}}$ *confluent and terminating).*

**Corollary 3.5.** *Every finite graph has a unique* $\to_{\mathrm{ep}}$ *normal form graph.*

# 4 Consequences

From confluence of $\to_{\mathrm{ep}}$ by Thm. 3.3 we obtain the following corollary by utilising basic rewriting properties of $\to_{\mathrm{elim}}$ and $\to_{\mathrm{prune}}$ that are easy to establish: $\to_{\mathrm{prune}}$ postpones over $\to_{\mathrm{elim}}$, and $\to_{\mathrm{prune}}$ steps preserve as well as reflect $\to_{\mathrm{elim}}$ normal forms, and the properties $\infty(\cdot)$ and $\neg\infty(\cdot)$.

**Corollary 4.1.** *For every finite process graph* $G$ *the following statements are equivalent:*

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

*(ii) The (by Cor. 3.5) 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.*

Now Cor. 4.1, (i) $\Leftrightarrow$ (iii), entails that we can verify $\mathrm{LEE}(G)$, for every finite graph $G$, simply by constructing an arbitrary $\to_{\mathrm{elim}}$ rewrite sequence to a normal form $G_0$, and then confirming whether $\neg\infty(G_0)$ holds; if the test fails, we can conclude $\neg\mathrm{LEE}(G)$. As $\to_{\mathrm{elim}}$ rewrite sequences to normal form and the test are implementable in polynomial time, we obtain the next theorem.

**Theorem 4.2.** *For finite graphs,* LEE *is decidable in* PTIME *with the graph size as input size.*

For extracting regular expressions $e$ with $P(e) \leftrightarrows G$ from $\to_{\mathrm{elim}}$ rewrite sequences that witness that a graph $G$ satisfies $\mathrm{LEE}(G)$ it is expedient to only use 'layered' loop elimination $\overset{\circ}{\to}_{\mathrm{elim}}$ steps. In such steps, loops are never eliminated from bodies of already eliminated ones. Defining $\overset{\circ}{\to}_{\mathrm{elim}}$ requires history awareness, which can be formalised by vertex-marked versions ${}^{\bullet}G$ of graphs $G$ in which the vertices in the body of already eliminated loops are marked.

**Definition 4.3** (LLEE)**.** A graph $G$ has the *layered* loop existence and elimination property LLEE, denoted by $\mathrm{LLEE}(G)$, if $G$ has an $\overset{\circ}{\to}_{\mathrm{elim}}$ normal form graph without an infinite trace:

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

**Theorem 4.4.** LLEE *coincides with* LEE.

*Proof (Idea).* For every sequence ${}^{\circ}G \overset{\circ}{\to}{}^{*}_{\mathrm{elim}} {}^{\bullet}G_1$ from an initially unmarked graph ${}^{\circ}G$ it holds that every $\to_{\mathrm{elim}}$ step from ${}^{\bullet}G_1$ that does not correspond to an $\overset{\circ}{\to}_{\mathrm{elim}}$ step gives rise to an $\overset{\circ}{\to}_{\mathrm{elim}}$ step from ${}^{\bullet}G_1$: the two steps eliminate overlapping loops in the same bi-loop, see Lem. 3.1, (iii). □

# 5 Conclusion

Three further questions that came up during our work on the confluence of $\to_{\mathrm{ep}}$, and its consequences are the following: (1) While a rough estimate (cf. version with appendix) of the complexity of deciding LEE according to Thm. 4.2 yields $\mathrm{LEE} \in \mathrm{O}(|G|^3)$, there is much room for improvement. How fast can an algorithm for implementing $\to_{\mathrm{elim}}$, and deciding LEE be? (2) We want to analyse systematically how, for a graph $G$ with LEE, any two regular expressions $e_1, e_2$ with $P(e_1) = P(e_2) \simeq G$ that can be extracted from two maximal $\to_{\mathrm{elim}}$ rewrite sequences from $G$ (see [6, 2, 4]) can be proved equivalent with respect to $[\![\cdot]\!]_P$. (3) Just as postponement of $\overset{\circ}{\to}_{\mathrm{elim}}$ over $\to_{\mathrm{elim}}$ is closely connected to confluence of $\to_{\mathrm{ep}}$ (Thm. 3.3), we wonder how the decreasing diagram method (perhaps in its 'converted' version by van Oostrom [11]) can best be adapted in general for showing postponement of a family of rewrite relations over another family.[1]

[1] In the meantime van Oostrom referred me to his IWC 2020 article [12], in which he explains the relationship between commutation of rewrite relations and their factorisation, which depending on the relations' order is also

# References


[1] Doeko Bosscher. *Grammars Modulo Bisimulation*. PhD thesis, University of Amsterdam, 1997.

[2] Clemens Grabmayer. A Coinductive Version of Milner's Proof System for Regular Expressions Modulo Bisimilarity. In Fabio Gadducci and Alexandra Silva, editors, *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, volume 211 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:23, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.CALCO.2021.16.

[3] Clemens Grabmayer. Milner's Proof System for Regular Expressions Modulo Bisimilarity is Complete (Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions). In *Proceedings of the 37th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '22, pages 1–13, New York, NY, USA, 2022. Association for Computing Machinery. Extended version as report arXiv:2209.12188 on arxiv.org, September, 2022. doi:10.1145/3531130.3532430.

[4] Clemens Grabmayer. A Coinductive Reformulation of Milner's Proof System for Regular Expressions Modulo Bisimilarity. *Logical Methods in Computer Science*, Volume 19, Issue 2, Jun 2023. URL: https://lmcs.episciences.org/11519, doi:10.46298/lmcs-19(2:17)2023.

[5] Clemens Grabmayer. Towards a Characterisation of the Graph Structure of Process Interpretations of Regular Expressions. Extended Abstract for TERMGRAPH 2026 (15th Int. Workshop on Computing with Terms and Graphs), published on the workshop's webpage http://termgraph.org.uk/2026, 2026. Also available at https://clegra.github.io/lf/pi-struc-TG-26.pdf.

[6] Clemens Grabmayer and Wan Fokkink. A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity. In *Proceedings of the 35th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '20, page 465–478, New York, NY, USA, 2020. Association for Computing Machinery. doi:10.1145/3373718.3394744.

[7] Robin Milner. A Complete Inference System for a Class of Regular Behaviours. *Journal of Computer and System Sciences*, 28(3):439–466, 1984. doi:10.1016/0022-0000(84)90023-0.

[8] Robert Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1(2):146–160, 1972. doi:10.1137/0201010.

[9] Terese. *Term Rewriting Systems*. Number 55 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2003. doi:10.1017/S095679680400526X.

[10] Vincent van Oostrom. Confluence by Decreasing Diagrams. *Theoretical Computer Science*, 126(2):259–280, 1994. doi:10.1016/0304-3975(92)00023-K.

[11] Vincent van Oostrom. Confluence by Decreasing Diagrams (Converted). In Andrei Voronkov, editor, *Rewriting Techniques and Applications*, pages 306–320, Berlin, Heidelberg, 2008. Springer Berlin Heidelberg. doi:10.1007/978-3-540-70590-1_21.

[12] Vincent van Oostrom. Some Symmetries of Commutation Diamonds. In http://iwc2020.cic.unb.br/iwc2020_proceedings.pdf, editor, *Proceedings of the 9th International Workshop on Confluence, June 30th, 2020, online*, pages 1–6, http://iwc2020.cic.unb.br, 2020. Mauricio Ayala-Rincón and Samuel Mimram. URL: http://iwc2020.cic.unb.br/iwc2020_proceedings.pdf.


called preponement or postponement.

# A Appendix

In this appendix we supplement the exposition in the extended abstract with some more details. Hereby the four subsections A.1–A.4 refer to the four sections 1–4 in the extended abstract.

## A.1 Supplements for Section 1

The example below concerns the passage in the Introduction: "There are finite process graphs that are neither $P$-expressible (the interpretation of a regular expression) nor $[\![\cdot]\!]_P$-expressible (bisimilar to a $P$-expressible graph)."

**Example A.1** (not $[\![\cdot]\!]_P$-expressible graphs, with failure of LEE)**.** Milner proved in [7] that the graph $G_1$ below is not only not $P$-expressible (i.e. $G_1$ is not the process interpretation of a regular expression) but that it is also not $[\![\cdot]\!]_P$-expressible (i.e., $G_1$ is also not bisimilar to the process interpretation of a regular expression). Bosscher showed later in [1] that the graph $G_2$ below is also not $[\![\cdot]\!]_P$-expressible.[2]


Above, and in all of our pictures the start vertex of a process graph is highlighted by a brown arrow ➙ , and a vertex $v$ with immediate termination is emphasised in brown as ◉ including a boldface ring. $G_1$ and $G_2$ witness the incompleteness of the process semantics $[\![\cdot]\!]_P$ with respect to finite-state processes. This contrasts with completeness of the language semantics of regular expressions with respect to regular languages, which means that every language that is accepted by a finite-state automaton is the language interpretation of some regular expression.

Neither of $G_1$ and $G_2$ satisfies LEE, because both process graphs do not contain loop subgraphs. In $G_1$, every transition from a vertex $v$ in $G_1$ enables an infinite trace that does not return to $v$ (see an example in red), which creates violations of (L2) for generated subgraphs from and until $v$. In $G_2$, every infinite trace from one vertex in $G_2$ visits the other, which permits immediate termination, thereby violating (L3) for the only possible loop candidate, $G_2$ itself. Therefore $G_1$ and $G_2$ are $\to_{\mathrm{elim}}$ normal forms. As both graphs enable infinite traces, $\infty(G_1)$ and $\infty(G_2)$ holds, entailing $\neg\mathrm{LEE}(G_1)$ and $\neg\mathrm{LEE}(G_2)$.

## A.2 Supplements for Section 2

We start with some more definitions of concepts for graphs. Let $G = \langle V, A, v_{\mathrm{s}}, \to, \Downarrow\rangle$ be a graph.

We introduce the convention of notation, concerning a graph $G'$ that we have not fixed by its defining tuple (like $G$ above), to refer by $V(G')$ to its set of vertices.

[2] For $G_2$ this also follows from the necessary condition for graphs to be $[\![\cdot]\!]_P$-expressible that Milner showed and used to recognise $G_1$ as not $[\![\cdot]\!]_P$-expressible in [7]: Infinite star-behaviours always have a 'loop' (in Milner's weaker sense) derivative. Yet bot $G_1$ and $G_2$ have infinite traces but do not have a loop derivative (also not) in Milner's sense. However, Milner uses a variant graph $G_2'$ of $G_2$ with additional selfloops with different labels say $c$ and $d$ at the two vertices. Showing that such a variant $G_2'$ of $G_2$ is not $[\![\cdot]\!]_P$-expressible, neither, is more difficult, but Bosscher succeeded in doing so in [1].

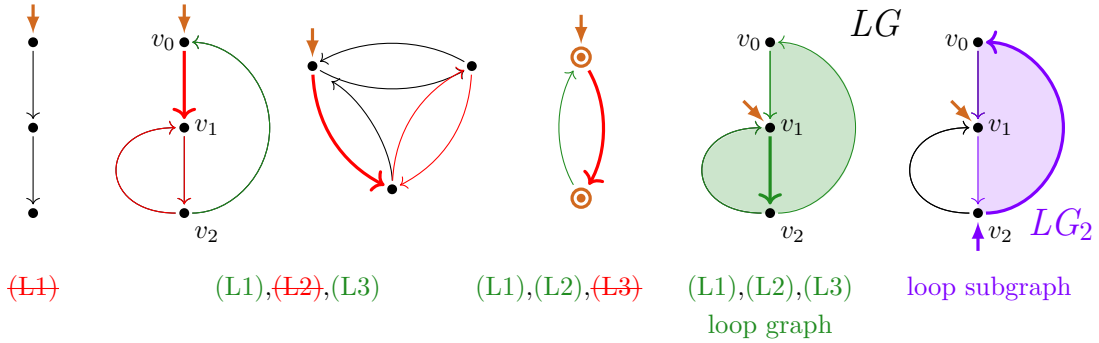


Figure 4: Four process graphs (action labels ignored) that violate at least one of the loop graph conditions (L1), (L2), and (L3), then a loop graph $LG$, and finally a loop subgraph $LG_2$ of $LG$.

We call the graph $G$ *finite* if its sets of vertices and actions are finite (then also its set of transitions is finite). For a finite graph $G$ as above we define its *size* by $|G| := |V| + |{\to}|$, that is, as the sum of the cardinality of its vertices and the cardinality of its transitions.

By a *trace from* $v$ in $G$ we mean a finite or infinite sequence $\langle v_0, a_1, v_1, a_2, \ldots\rangle$ that consists of vertices alternated with actions of $G$ with $v_0 = v$, and $v_0, v_1, v_2, \ldots \in V$ and $a_1, a_2, \ldots \in A$ so that a transition sequence $v = v_0 \xrightarrow{a_1} v_1 \xrightarrow{a_2} \ldots$ from $v$ in $G$ is represented. By a *path from* $v$ in $G$ we mean a finite or infinite sequence $\langle v_0, v_1, \ldots\rangle$ of vertices that results from a trace $\langle v_0, a_1, v_1, a_2, \ldots\rangle$ from $v$ in $G$ by dropping the actions of transitions.

We denote by $\infty(v)$ (resp. by $\neg\infty(v)$) that there is an infinite trace from $v$ in $G$ (resp. that there is <u>not</u> an infinite trace from $v$ in $G$).

Let $v \in V$. By $G{\downarrow_*^v} := \mathsf{gc}(\langle V, A, v, {\to}, {\Downarrow}\rangle)$ we define the *subgraph of $G$ at $v$* that results from $G$ by changing the start vertex to $v$, and then performing garbage collection.

We also recapitulate the definition of generated subgraphs from and until given vertices.

Let $v \in V$ be a vertex. We denote by $\langle v{\to}\rangle := \{\langle v, a, w\rangle \mid \langle v, a, w\rangle \in {\to}\}$ the set of transitions from $v$ in $G$. Now let $v \in V$, and $T \subseteq \langle v{\to}\rangle \subseteq {\to}$ be a set of transitions from $v$ in $G$.

We define the *generated subgraph of $G$ from and until $v$ with respect to $T$* by $G{\downarrow_*^{v,T}} := \langle V_0, A, v, T \cup ((V_0 \setminus \{v\}) \times A \times V_0), {\Downarrow} \cap V_0\rangle$ where $V_0 \subseteq V$ is the set of vertices on traces in $G$ that start in $v$, take a transition from $T$, and then continue onwards as long as possible but stop whenever $v$ is reached again. Note that $G{\downarrow_*^{v,T}}$ is start-vertex connected by the definition of $V_0$.

**Definition** (= Def. 2.1 on loop graphs, loop subgraphs, bi-loops)**.** Let $G = \langle V, A, v_{\mathrm{s}}, {\to}, {\Downarrow}\rangle$ be a graph. We say that $G$ is a *loop (process graph)* if it satisfies the following three conditions:

(L1) There is an infinite trace from the start vertex $v_{\mathrm{s}}$ of $G$, i.e. $\infty(v_{\mathrm{s}})$ holds.

(L2) Every infinite trace from the start vertex $v_{\mathrm{s}}$ of $G$ returns to $v_{\mathrm{s}}$.

(L3) Immediate termination is only permitted at the start vertex of $G$, i.e. $v{\Downarrow}$ iff $v = v_{\mathrm{s}}$.

Then we call transitions from $v_{\mathrm{s}}$ *loop-entry transitions*, and the other ones *loop-body transitions*.

By a *loop subgraph* of $G$ (where $G$ does not have to be a loop itself) we mean a generated subgraph $G{\downarrow_*^{v,T}}$ of $G$ from/until $v \in V$ with respect to $T \subseteq \langle v{\to}\rangle$ such that $G{\downarrow_*^{v,T}}$ is a loop.

Let $v_1, v_2 \in V$ be such that $v_1 \neq v_2$. We say that $G$ is a *bi-loop with respect to $v_1, v_2$* if $G{\downarrow_*^{v_1,\langle v_1{\to}\rangle}}$, $G{\downarrow_*^{v_2,\langle v_2{\to}\rangle}}$, and $G$ differ at most with respect to their start vertices $v_1$, $v_2$, and $v$.

**Example A.2.** In Fig. 4 we illustrate four non-examples for loop graphs, each of which violates one of the loop conditions (L1), (L2), or (L3), and furthermore an example of a loop graph, and of a loop subgraph.

*Remark* A.3 (paths suffice to define loops, and LEE). Since in the defining clauses (L1)–(L3) of loop graphs the actions on the assumed traces in the graph do not play any role, the use of 'infinite trace' in (L1) and (L2) (and thus everywhere in Def. 2.1) can be replaced by 'infinite path' everywhere. It follows that the action labels in graphs are therefore also irrelevant for the definition of the property LEE in Def. 2.6. As a consequence, LEE can be understood as a property of the path structure of process graphs.

**Lemma** (= Lem. 2.2)**.** *Let $B = \langle V, A, v_s, \to, \Downarrow\rangle$ be a graph, and let $v_1, v_2 \in V$ be s.th. $v_1 \neq v_2$.*

*Then $B$ is a bi-loop with respect to $v_1, v_2$ if and only if the following specialised variant of the loop conditions (L1)–(L3) hold:* (B1) *there is an infinite trace from $v_1$,* (B2) *every infinite trace from $v_1$ visits $v_2$ before it visits $v_1$ again, and every infinite trace from $v_2$ visits $v_1$ before it visits $v_2$ again,* (B3) *no vertex of $B$ permits immediate termination, i.e. $v\not\downarrow$ for all $v \in V$.*

*Proof.* For the proof we use the abbreviations $LG_1 := B{\downarrow_*^{v_1,\langle v_1\to\rangle}}$ and $LG_2 := B{\downarrow_*^{v_2,\langle v_2\to\rangle}}$.

For proving the 'if' part of the bi-implication in the lemma we assume that $B$ satisfies the bi-loop conditions (B1), (B2), and (B3). We have to show that $LG_1$ and $LG_2$ satisfy the loop conditions (L1), (L2), and (L3).

The condition (L1) for $LG_1$ coincides with (B1). Since there is an infinite trace $\tau_1$ from $v_1$ due to (B1), which due to (B2) must pass through $v_2$, there is also an infinite trace $\tau_2$ from $v_2$. This shows (L1) for $LG_2$.

For verifying the condition (L2) for $LG_1$, let $\tau_1$ be an infinite trace from $v_1$. We have to show that $\tau_1$ returns to $v_1$. By (B2) $\tau_1$ visits $v_2$ before (if so) returning to $v_1$. Let $\tau_2$ be the (infinite) part of $\tau_1$ after first visiting $v_2$. Then it follows again by (B2) that $\tau_2$ must visit $v_1$ (before it may return to $v_2$ again). But that means that $\tau_1$, of which $\tau_2$ is an infinite final segment, must return to $v_2$, just what we had to show. For $LG_2$ the condition (L2) can be argued analogously.

For verifying the condition (L3) for $LG_1$ we have to show that $v\not\downarrow$ holds for all vertices $v \neq v_1$, and for verifying (L3) for $LG_2$ we have to show that $v\not\downarrow$ holds for all vertices $v \neq v_2$. Since $v_1 \neq v_2$, we must therefore show that $v\not\downarrow$ holds for all vertices $v \in V$ of $B$. That, however, is just the bi-loop condition (B3) that we have assumed.

For showing the 'only if' part of the bi-implication in the lemma, we assume that $B$ is a bi-loop. That is, $LG_1$ and $LG_2$ are loops, and furthermore, $LG_1$, $LG_2$, and $G$ differ at most with respect to their start vertices $v_1$, $v_2$, and $v$. We have to show the bi-loop conditions (B1), (B2), and (B3) for $B$.

The condition (B1) for $B$ follows directly from (L1) for $LG_1$.

For showing (B2) for $B$ we have to prove that every infinite trace from $v_1$ visits $v_2$ before (perhaps) returning to $v_1$, and that every infinite trace from $v_2$ visits $v_1$ before (perhaps) returning to $v_2$. We only argue for the first part, because the second can be established analogously.

For showing the first part, we argue indirectly, and assume that there is an infinite trace $\tau_1$ in $B$ from $v_1$ that does not visit $v_2$. We will derive a contradiction to our assumptions, the loop conditions for $LG_1$ and $LG_2$. Since $LG_1$ and $LG_2$ have the same vertices, $v_1$ is a vertex of $LG_2$, and since $LG_2 = B{\downarrow_*^{v_2,\langle v_2\to\rangle}}$ is start-vertex connected to its start vertex $v_2$, there is a finite trace $\tau_{21}$ from $v_2$ to $v_1$. We may assume that $\tau_{21}$ visits $v_2$ only at its start, and so does not return to $v_2$ before its end in $v_1$; otherwise we take an initial segment with that property of the trace that we picked. But now the infinite trace in $B$ defined by $\tau := \tau_{21} \cdot \tau_1 \cdot \tau_1 \cdot \ldots$, which repeats $\tau_1$ infinitely often, visits, by its construction from $\tau_{21}$ (which visits $v_2$ only at the

start) and $\tau_1$ (which does not visit $v_2$ at all), $v_2$ only at the start, but does never return to it. This trace $\tau$, however, now violates the loop condition (L2) for $LG_2$ (that infinite traces from $v_2$ always return to $v_2$ eventually) since it also is a trace in $LG_2 = B{\downarrow_*^{v_2,\langle v_2\to\rangle}}$. We have reached a contradiction with our assumptions. As this contradiction cannot stand, we conclude that every infinite trace from $v_1$ must visit $v_2$ before returning to $v_1$.

For showing the condition (B3) for $B$ we have to show that $v\not\downarrow$ holds for all $v \in V$. Now since $LG_1$ and $LG_2$ are loops with the same set $V$ of vertices, it follows from (L3) for $LG_1$ that $v\not\downarrow$ for all vertices except for $v_1$, and from (L3) for $LG_2$ that $v\not\downarrow$ for all vertices except for $v_2$. Therefore $v\not\downarrow$ holds indeed for all $v \in V$. □

**Lemma** (= Lem. 2.7)**.** *On finite process graphs,* $\to_{\mathrm{ep}} = \to_{\mathrm{elim}} \cup \to_{\mathrm{prune}}$ *is terminating.*

*Proof.* In every $\to_{\mathrm{elim}}$ or $\to_{\mathrm{prune}}$ step on a finite graph the graph size decreases by at least 1, because one transition is removed (decoupled or pruned) in any case, and possibly more transitions and/or vertices become unreachable and are removed afterwards by garbage collection. □

## A.3 Supplements for Section 3

**Definition A.4** (start-vertex connected cyclic graph part)**.** Let $G = \langle V, A, v_{\mathrm{s}}, \to, \Downarrow\rangle$ be a graph.

By the *start-vertex connected cyclic part* $\circlearrowright(G)$ *of* $G$ we mean the graph that consists of all vertices and transitions of $G$ that are bi-connected with the start vertex $v_{\mathrm{s}}$ (with traces from $v_{\mathrm{s}}$ and traces to $v_{\mathrm{s}}$), and that is defined by:

$$\begin{aligned}\circlearrowright(G) &:= \langle \circlearrowright(V), A, v_{\mathrm{s}}, \to \cap (\circlearrowright(V) \times A \times \circlearrowright(V)), \Downarrow \cap \circlearrowright(V)\rangle\,,\\ \text{where: } \circlearrowright(V) &:= \{v \in V \mid v_{\mathrm{s}} \to_G^* v \to_G^* v_{\mathrm{s}}\}\ (\to_G \text{ means a step via a transition of } G).\end{aligned}$$

Hereby we mean by saying that two vertices $v_1, v_2 \in V$ are *bi-connected in* $G$ if $v_1 \to_G^* v_2 \to_G^* v_1$ holds, that is, if there are traces from $v_1$ to $v_2$ and from $v_2$ to $v_1$.

For Lem. 3.1, which provides a complete overview of $\to_{\mathrm{elim}}$ forks and how they can be joined, we here only explain the crucial argument underlying joining $\to_{\mathrm{elim}}$ forks in its case (iii). That case concerns loop subgraphs with overlapping start-vertex connected cyclic parts. The reasoning for this situation flows forth from the following lemma. It states that loop subcharts with vertex-overlapping cyclic parts together form a bi-loop subgraph.

**Lemma A.5.** *Let* $G = \langle V, A, v_{\mathrm{s}}, \to, \Downarrow\rangle$ *be a graph. Let* $v_1, v_2 \in V$ *with* $v_1 \neq v_2$ *be two different vertices of* $G$*, and let* $T_1 \subseteq \langle v_1\to\rangle$ *and* $T_2 \subseteq \langle v_2\to\rangle$ *be sets of transitions from* $v_1$ *and* $v_2$ *such that the generated subgraphs* $G{\downarrow_*^{v_1,T_1}}$ *and* $G{\downarrow_*^{v_2,T_2}}$ *are loop subgraphs of* $G$*. Suppose further that for the start-vertex connected cyclic parts* $\circlearrowright(G{\downarrow_*^{v_1,T_1}})$*, and* $\circlearrowright(G{\downarrow_*^{v_2,T_2}})$ *it holds that* $V(\circlearrowright(G{\downarrow_*^{v_1,T_1}})) \cap V(\circlearrowright(G{\downarrow_*^{v_2,T_2}})) \neq \varnothing$*, that is, these parts have vertices in common.*

*Then the subgraphs* $G{\downarrow_*^{v_1}}$ *at* $v_1$*, and* $G{\downarrow_*^{v_2}}$ *at* $v_2$ *are bi-loops with respect to* $v_1, v_2$*, and with respect to* $v_2, v_1$*, which furthermore differ only by their start vertices* $v_1$ *and* $v_2$*.*

*Proof (Sketch).* We can pick a vertex $v \in V(\circlearrowright(G{\downarrow_*^{v_1,T_1}})) \cap V(\circlearrowright(G{\downarrow_*^{v_2,T_2}})) \neq \varnothing$. Then $v_1 \to_G^* v \to_G^* v_1$ and $v_2 \to_G^* v \to_G^* v_2$ follows, that is, $v$ is bi-connected with $v_1$ and with $v_2$. It follows that the start-vertex connected, cyclic part $\circlearrowright(G{\downarrow_*^{v_1}})$ of $G{\downarrow_*^{v_1}}$ (the subgraph of $G$ at $v_1$), and in the start-vertex connected, cyclic part $\circlearrowright(G{\downarrow_*^{v_2}})$ of $G{\downarrow_*^{v_2}}$ (the subgraph of $G$ at $v_2$) are non-empty, and that every vertex in either of them is bi-connected with each of $v$, $v_1$, $v_2$. Therefore $G{\downarrow_*^{v_1}}$ and $G{\downarrow_*^{v_2}}$ encompass the loop subcharts $G{\downarrow_*^{v_1,T_1}}$ and $G{\downarrow_*^{v_2,T_2}}$. Furthermore they only differ with respect to their start vertex. Crucially, it can be shown that $G{\downarrow_*^{v_1}}$ and $G{\downarrow_*^{v_2}}$ are themselves loop subcharts of $G$. From this now Lem. 2.2 implies that $G{\downarrow_*^{v_1}}$ and $G{\downarrow_*^{v_2}}$ are bi-loops that differ only with respect to their start vertices $v_1$ and $v_2$. □

**Lemma** (= Lem. 3.2). $\to_{\mathrm{ep}}$ *is (locally) decreasing with respect to the family* $\{\to_\ell\}_{\ell\in\{\mathrm{elim},\mathrm{prune}\}}$ *of rewrite relations, with the precedence order* prune $<$ elim.

*Proof (slightly extended version).* The full set of elementary diagrams for the abstract rewriting system with the family $\{\to_\ell\}_{\ell\in\{\mathrm{elim},\mathrm{prune}\}}$ of rewriting relations is illustrated in Fig. 3. All of these diagrams are decreasing with respect to the precedence order prune $<$ elim of reduction labels, as witnessed by the trace relation indicated by arrows in Fig. 3. Thereby the precedence order is only used for recognising that the first elementary diagram is decreasing.

This first elementary diagram in Fig. 3 concerns $\to_{\mathrm{elim}}$ forks, and is based on the analysis of the three possible, mutually exclusive cases for $\to_{\mathrm{elim}}$ forks that is stated by Lem. 3.1. Indeed it corresponds to the most general joining steps in the cases (i) and (iii) in Lem. 3.1.

The second elementary diagram in Fig. 3 concerning joining $\leftarrow_{\mathrm{prune}}\cdot\to_{\mathrm{elim}}$ forks by sub-commutation of the two involved steps is due to the fact that transitions to deadlock states which are removed in $\to_{\mathrm{prune}}$ steps cannot be part of the infinite traces in the cyclic parts of loop subgraphs. Therefore every $G_0\to_{\mathrm{elim}}G_1$ has, after a step $G_0\to_{\mathrm{prune}}G_0'$, a residual step $G_0'\to_{\mathrm{elim}}G_2'$ such that furthermore $G_2\to^{=}_{\mathrm{prune}}G_2'$ holds.

The third elementary diagram in Fig. 3 concerning $\to_{\mathrm{prune}}$ forks is easy to understand for the following reason: if the two $\to_{\mathrm{prune}}$ steps coincide, then they have the same targets, which are joined by empty steps; if the steps are different, then they commute. □

**Theorem** (= Thm. 3.3). $\to_{\mathrm{ep}}=\to_{\mathrm{elim}}\cup\to_{\mathrm{prune}}$ *is confluent.*

*Proof.* From the fact that $\to_{\mathrm{ep}}$ is decreasing by Lem. 3.2 it follows by the Decreasing Diagrams Theorem (see [10, Cor. 3.9], [9, Thm. 14.2.8], [11, Thm. 1]) that $\to_{\mathrm{ep}}$ is confluent. □

**Corollary** (= Cor. 3.4). $\to_{\mathrm{ep}}=\to_{\mathrm{elim}}\cup\to_{\mathrm{prune}}$ *is complete (confluent and terminating).*

*Proof.* Follows from Thm. 3.3 on confluence of $\to_{\mathrm{ep}}$, and Lem. 2.7 on termination of $\to_{\mathrm{ep}}$. □

**Corollary** (= Cor. 3.5). *Every finite graph has a unique* $\to_{\mathrm{ep}}$ *normal form graph.*

*Proof.* Follows from confluence of $\to_{\mathrm{ep}}$ due to Thm. 3.3 in view of that confluence implies uniqueness of normal forms in abstract rewrite systems. □

## A.4 Supplements for Section 4

We provide a number of proofs of the results in Sect. 4, and sketch the main ideas of others.

The basic rewrite properties of $\to_{\mathrm{elim}}$ and $\to_{\mathrm{prune}}$ that we mentioned right before Cor. 4.1 are formulated in the lemma below.

**Lemma A.6.** *The following statements hold:*

*(i)* $\to_{\mathrm{prune}}\cdot\to_{\mathrm{elim}}=\to_{\mathrm{elim}}\cdot\to^{=}_{\mathrm{prune}}$, *that is,* $\to_{\mathrm{prune}}$ *one-step postpones over* $\to_{\mathrm{elim}}$.

*(ii)* $\to^*_{\mathrm{ep}}\subseteq\to^*_{\mathrm{elim}}\cdot\to^*_{\mathrm{prune}}$, *that is,* $\to_{\mathrm{prune}}$ *postpones over* $\to_{\mathrm{elim}}$ *in* $\to^*_{\mathrm{ep}}$ *rewrite sequences.*

*(iii)* $\to_{\mathrm{prune}}$ *steps preserve and reflect* $\to_{\mathrm{elim}}$ *normal forms.*

*(iv)* $\to_{\mathrm{prune}}$ *steps preserve and reflect the properties* $\infty(\cdot)$ *and* $\neg\infty(\cdot)$.

*Proof (Idea).* Statement (ii) can be shown from statement (i) by induction on the number of $\to_{\mathrm{elim}}$ steps in given $\to_{\mathrm{ep}}$ rewrite sequences. Statements (i), (iii), and (iv) are again (similar to the second elementary diagram in Fig. 3 for Lem. 3.2) quite direct consequences of the fact that transitions to deadlock states that are removed in $\to_{\mathrm{prune}}$ steps cannot be part of infinite

traces like in the cyclic parts of loop subgraphs. In particular, a $\to_{\mathrm{prune}}$ step does not affect the cyclic part of any loop subgraph, nor can it lead to the creation of a new loop subgraph. Therefore every step $G_1 \to_{\mathrm{elim}} G_2$ after a step $G_0 \to_{\mathrm{prune}} G_1$ gives rise to a step $G_0 \to_{\mathrm{elim}} G_1'$ already from $G_0$ such that $G_1' \to^{=}_{\mathrm{prune}} G_2$. □

**Corollary** (= Cor. 4.1)**.** *For finite process graphs $G$ the following statements are equivalent:*

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

*(ii) The (by Cor. 3.5) 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.*

*Proof.* For showing (i) ⇒ (ii), we assume (i), and let $G_0$ be the $\to_{\mathrm{ep}}$ normal form of a graph $G$. We have to show $\neg\infty(G_0)$. Due to postponement of $\to_{\mathrm{prune}}$ over $\to_{\mathrm{elim}}$ steps (by Lem. A.6, (ii)) there is a graph $G'$ such that $G \to^*_{\mathrm{elim}} G' \to^*_{\mathrm{prune}} G_0 \not\to_{\mathrm{ep}}$. Since $\to_{\mathrm{prune}}$ steps reflect $\to_{\mathrm{elim}}$ normal forms (by Lem. A.6, (iii)), also $G'$ is an $\to_{\mathrm{elim}}$ normal form. Then (i) yields $\neg\infty(G')$. Finally, since $\to_{\mathrm{prune}}$ steps also preserve $\neg\infty(\cdot)$ (by Lem. A.6, (iv)), $\neg\infty(G_0)$ follows.

For showing (ii) ⇒ (iii), we assume that the unique $\to_{\mathrm{ep}}$ normal form $G_0$ of $G$ satisfies $\neg\infty(G_0)$, and we let $G'$ be an arbitrary $\to_{\mathrm{elim}}$ normal from $G$. We have to show $\neg\infty(G')$ Then we find an $\to_{\mathrm{elim}}$ fork of the form $\not\leftarrow_{\mathrm{elim}} G' \leftarrow^*_{\mathrm{elim}} G \to^*_{\mathrm{ep}} G_0 \not\to_{\mathrm{ep}}$. Due to confluence of $\to_{\mathrm{ep}}$ (by Thm. 3.3), and $G_0$ is an $\to_{\mathrm{ep}}$ normal from, the fork must be joined by the $\to_{\mathrm{ep}}$ sequence $G' \to^*_{\mathrm{ep}} G_0$. This sequence, however, can only consist of $\to_{\mathrm{prune}}$ steps, because $G'$ is an $\to_{\mathrm{elim}}$ normal form and $\to_{\mathrm{elim}}$ normal forms are preserved along $\to_{\mathrm{prune}}$ steps (by Lem. A.6, (iii)). Thus we have $G' \to^*_{\mathrm{prune}} G_0$ and $\neg\infty(G_0)$. From this $\neg\infty(G_0)$ follows from Lem. A.6, (iv).

The implication (iii) ⇒ (i) follows from termination of $\to_{\mathrm{elim}}$ on finite graphs, see Lem. 2.7. □

**Theorem** (= Thm. 4.2)**.** *For finite graphs,* LEE *is decidable in* PTIME *with the graph size as input size.*

*Proof.* In order to decide LEE($G$) for a finite graph $G$ it suffices to compute an arbitrary $\to_{\mathrm{elim}}$ normal form $G_0$ of $G$, and check whether $G_0$ enables an infinite trace: if $\neg\infty(G_0)$ holds, then LEE($G$) follows by definition; if $\infty(G_0)$ holds, then ¬LEE($G$) follows, because LEE($G$) would imply $\neg\infty(G_0)$ by Cor. 4.1, (i) ⇒ (iii). It is not necessary to check all $\to_{\mathrm{elim}}$ normal forms of $G$.

For every vertex $v$ and every transition $t$ of a graph $G$, it can be determined in time $\mathrm{O}(|G|)$ whether $G{\downarrow}^{v,\{t\}}_{*}$ is a loop, by adapting Tarjan's depth-first search algorithm [8] for determining scc's. It follows that checking whether $G$ is an $\to_{\mathrm{elim}}$ normal form, or finding an $\to_{\mathrm{elim}}$ step and performing it can be done in $\mathrm{O}(|G|^2)$ time. Since $\to_{\mathrm{elim}}$ steps decrease the graph size by at least 1, $\to_{\mathrm{elim}}$ normal forms of $G$ can be constructed in $\mathrm{O}(|G|^3)$ time. Checking whether a normal form graph has an infinite trace is again computable in linear time by using a DFS. – This very rough estimate shows that LEE($G$) can be computed in time $\mathrm{O}(|G|^3)$, i.e. LEE ∈ PTIME. □

For the definition of layered loop elimination, we extend the concept of graphs to that of 'vertex-marked graphs' in which the marked vertices are recorded in a separate set. On such vertex-marked graphs we define 'layered loop elimination' steps $\overset{\circ}{\to}_{\mathrm{elim}}$ via keeping track of vertices in the loop bodies of eliminated loops by marking them (note that start vertices of eliminated loops are not marked), and then by permitting the elimination of loop subcharts only at unmarked vertices (and disallowing eliminations at marked vertices). Then layered LEE (LLEE) is defined by using the elimination relation $\overset{\circ}{\to}_{\mathrm{elim}}$ on marked graphs instead of $\to_{\mathrm{elim}}$.

**Definition A.7** (marked graphs)**.** A *(vertex-)marked graph* is a tuple $^\bullet G = \langle V, {}^\bullet V, A, v_s, \to, \Downarrow\rangle$ such that the *unmarking* $|{}^\bullet G|_\circ := \langle V, A, v_s, \to, \Downarrow\rangle$ of $^\bullet G$ is a graph, and $^\bullet V \subseteq V$ is a subset of *marked* vertices. For such a marked graph $^\bullet G$ and a subset $W \subseteq V$ of its vertices we define by $\mathsf{mk}({}^\bullet G, W) := \langle V, {}^\bullet V \cup W, A, v_s, \to, \Downarrow\rangle$ the result of marking also the vertices in $W$ in $^\bullet G$.

For every graph $G = \langle V, A, v_s, \to, \Downarrow\rangle$ we define its (trivial) marking by $^\circ G = \langle V, \varnothing, A, v_s, \to, \Downarrow\rangle$.

The operations of decoupling $\mathsf{dcpl}$ of sets of transitions (by dropping them), and garbage collection $\mathsf{gc}$ (by removing unreachable vertices and transitions) are extended to marked graphs in the obvious way. For marked graphs $^\bullet G$ the statement $\infty({}^\bullet G)$ that $^\bullet G$ enables an infinite trace is defined as the statement $\infty(|{}^\bullet G|_\circ)$ that the unmarking of $^\bullet G$ enables an infinite trace.

**Definition A.8.** We define *layered* loop elimination steps $\overset{\circ}{\to}_{\mathrm{elim}}$ between marked graphs $^\bullet G$ and $^\bullet G'$ as follows:

$$^\bullet G \overset{\circ}{\to}_{\mathrm{elim}} {}^\bullet G' \;:\Longleftrightarrow\; \exists v \in V \setminus {}^\bullet V\, \exists T \subseteq \langle v\to\rangle \big[\, G{\downarrow_*^{v,T}} \text{ is a loop} \;\wedge\; {}^\bullet G' = \mathsf{gc}(\mathsf{dcpl}(\mathsf{mk}({}^\bullet G, V(G{\downarrow_*^{v,T}}) \setminus \{v\}), T))\,\big]\,.$$
(where $^\bullet G = \langle V, {}^\bullet V, A, v_s, \to, \Downarrow\rangle$)

**Definition** (= Def. 2.6 (LLEE))**.** We say that a process graph $G$ satisfies the *layered loop existence and elimination property* LLEE, denoted by LLEE($G$), if $G$ has an $\overset{\circ}{\to}_{\mathrm{elim}}$ normal form graph that does not enable an infinite trace. More formally, we stipulate:

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

For showing that LLEE coincides with LEE we need a technical lemma that states that $\overset{\circ}{\to}_{\mathrm{elim}}$ steps project to $\to_{\mathrm{elim}}$ steps via unmarking projection $|\cdot|_\circ$, and also that $\to_{\mathrm{elim}}$ rewrite sequences on graphs 'lift' (in opposite direction of the unmarking projection) to $\overset{\circ}{\to}_{\mathrm{elim}}$ rewrite sequences on marked graphs modulo the $\to_{\mathrm{prune}}$ step convergence $\leftrightarrow^*_{\mathrm{prune}}$. The technically crucial part hereby is statement (4). It asserts that, after an arbitrary $\overset{\circ}{\to}_{\mathrm{elim}}$ rewrite sequence $^\circ G \overset{\circ}{\to}_{\mathrm{elim}} {}^\bullet G_1$ from the trivial marking $^\circ G$ of a not marked graph $G$, every $\to_{\mathrm{elim}}$ step $|G_1|_\circ \to_{\mathrm{elim}} G_1'$ from the unmarking $|G_1|_\circ$ of $G_1$ can be lifted, also if it is not itself the unmarking projection of a $\overset{\circ}{\to}_{\mathrm{elim}}$ step from $G_1$, to a $\overset{\circ}{\to}_{\mathrm{elim}}$ step $|G_1|_\circ \overset{\circ}{\to}_{\mathrm{elim}} {}^\bullet G_1''$ from $|G_1|_\circ$ such that $|{}^\bullet G_1''|_\circ \leftrightarrow^*_{\mathrm{prune}} G_1'$.

**Lemma A.9.** *For all graphs $G$, $G'$, and $G_1'$, and for all marked graphs $^\bullet G_1$ and $^\bullet G_1'$ the following statements hold:*

$$\infty({}^\bullet G) \iff \infty(|{}^\bullet G|_\circ)\,, \tag{1}$$

$$^\bullet G_1 \overset{\circ}{\to}_{\mathrm{elim}} {}^\bullet G_1' \implies |{}^\bullet G_1|_\circ \to_{\mathrm{elim}} |{}^\bullet G_1'|_\circ\,, \tag{2}$$

$$^\bullet G_1 \overset{\circ}{\to}{}^*_{\mathrm{elim}} {}^\bullet G_1' \implies |{}^\bullet G_1|_\circ \to^*_{\mathrm{elim}} |{}^\bullet G_1'|_\circ\,, \tag{3}$$

$$^\circ G \overset{\circ}{\to}{}^*_{\mathrm{elim}} {}^\bullet G_1 \implies \Big[\exists {}^\bullet G_1''\Big[\begin{smallmatrix} {}^\bullet G_1 \overset{\circ}{\to}_{\mathrm{elim}} {}^\bullet G_1'' \\ \wedge\; |{}^\bullet G_1''|_\circ \to^*_{\mathrm{prune}} \cdot \leftarrow^*_{\mathrm{prune}} \cdot \leftarrow^=_{\mathrm{elim}} G_1' \end{smallmatrix}\Big] \impliedby |{}^\bullet G_1|_\circ \to_{\mathrm{elim}} G_1'\Big]\,, \tag{4}$$

$$\exists {}^\bullet G_1\big[\, {}^\circ G \overset{\circ}{\to}_{\mathrm{elim}} {}^\bullet G_1 \;\wedge\; |{}^\bullet G_1|_\circ \leftrightarrow^*_{\mathrm{prune}} G_1\,\big] \impliedby G \to^*_{\mathrm{elim}} G_1\,, \tag{5}$$

$$^\circ G \overset{\circ}{\to}{}^*_{\mathrm{elim}} {}^\bullet G_1 \implies \Big[{}^\bullet G_1 \textit{ is a } \overset{\circ}{\to}_{\mathrm{elim}} \textit{normal form} \iff \begin{smallmatrix} |{}^\bullet G_1|_\circ \textit{ is an } \to_{\mathrm{elim}} \\ \textit{normal form} \end{smallmatrix}\Big]. \tag{6}$$

*Proof.* Statement (1) is a consequence of the fact that traces from a marked graph $^\bullet G$ project, via unmarking $|\cdot|_\circ$ of sources and targets of steps, to traces from the unmarking $|{}^\bullet G|_\circ$ of $^\bullet G$.

Statement (2) can be understood as the obvious fact that every layered elimination step $\overset{\circ}{\to}_{\mathrm{elim}}$ projects to an elimination step $\to_{\mathrm{elim}}$ on the unmarked source and target graphs. Then statement (3) follows from statement (2) by induction on the length of the $\overset{\circ}{\to}_{\mathrm{elim}}$ rewrite sequence on the left-hand side.

We sketch the proof of statement (4) as follows. Suppose that we have given a layered loop elimination rewrite sequence ${}^{\circ}G \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_1$ from the trivial marking ${}^{\circ}G$ of a (not marked) graph $G$. Let us further assume that we have given a step $|{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} G_1'$ that eliminates a loop $LG = (|{}^{\bullet}G_1|_{\circ})\!\downarrow_{*}^{v,T}$ such that $v$ is a marked vertex in ${}^{\bullet}G_1$. (If $v$ is not marked, then this $\to_{\mathrm{elim}}$ step directly corresponds to a $\overset{\circ}{\to}_{\mathrm{elim}}$ step on ${}^{\bullet}G_1$.) Since $LG$ is a loop subchart, there is a non-empty, cyclic trace $\tau$ from $v$ to $v$, of which we may assume that it does not visit $v$ in between. The cyclic trace $\tau$ is also a trace in ${}^{\bullet}G_1$, which there starts in the marked vertex $v$. Now it cannot be the case that all vertices on $\tau$ in ${}^{\bullet}G_1$ are marked: this is because otherwise all of the vertices on $\tau$ are in the body of some loop subchart that has been eliminated before. But no infinite trace is possible strictly within the body of a loop subchart without visiting the appertaining loop vertex. We conclude that there must be a vertex on $\tau$ that is not marked in ${}^{\bullet}G_1$. We pick the first not marked vertex on $\tau$, and denote it by $v_0$. Then one can show that all infinite traces from $v$ have to pass through $v_0$, and moreover that $LG$ is a bi-loop with respect to $v, v_0$. Consequently also $LG_0 := (|{}^{\bullet}G_1|_{\circ})\!\downarrow_{*}^{v_0,\langle v_0 \to\rangle}$ is a loop subgraph of $|{}^{\bullet}G_1|_{\circ}$, and one that differs from $LG$ only with respect to its new start vertex $v_0$. Now since $v_0$ is unmarked in ${}^{\bullet}G_1$, there is also a layered loop elimination step ${}^{\bullet}G_1 \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_1''$ from ${}^{\bullet}G_1$ that eliminates ${}^{\bullet}G_1\!\downarrow_{*}^{v_0,\langle v_0\to\rangle}$, and projects to an $\to_{\mathrm{elim}}$ step $|{}^{\bullet}G_1|_{\circ} \to_{\mathrm{elim}} |{}^{\bullet}G_1''|_{\circ}$ that eliminates $LG_0$. Analogously to case (iii) in Lem. 3.1 concerning the joining of $\to_{\mathrm{elim}}$ forks that in loop subcharts that overlap within a bi-loop, now the $\to_{\mathrm{elim}}$ fork:

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

can be joined as follows:

$$|{}^{\bullet}G_1''|_{\circ} \to_{\mathrm{prune}}^{*} \cdot \leftarrow_{\mathrm{prune}}^{*} \cdot \leftarrow_{\mathrm{elim}}^{=} G_1' \,.$$

(This form underlies the left-right unified form in Lem. 3.1, (iii).) This joined fork shows the conclusion of the implication "$\Leftarrow$" in (4).

Statement (5) follows from statement (6) by induction on the length of the $\to_{\mathrm{elim}}$ rewrite sequence on the right-hand side. In that proof also postponement of $\to_{\mathrm{prune}}$ steps over $\to_{\mathrm{elim}}$ steps, see Lem. A.6, (i), and confluence of $\to_{\mathrm{ep}}$, see Thm. 3.3, are used.

Statement (6), which states that $\overset{\circ}{\to}_{\mathrm{elim}}$ normal forms ${}^{\bullet}G_1$ correspond to $\to_{\mathrm{elim}}$ normal forms $|{}^{\bullet}G_1|_{\circ}$ under the unmarking projection $|\cdot|_{\circ}$ provided that ${}^{\bullet}G_1$ results by a $\overset{\circ}{\to}_{\mathrm{elim}}$ normal form from an initially unmarked graph, follows directly from (2) (the projection of $\overset{\circ}{\to}_{\mathrm{elim}}$ steps to $\to_{\mathrm{elim}}$ steps), and (4) (lifting of $\to_{\mathrm{elim}}$ steps to $\overset{\circ}{\to}_{\mathrm{elim}}$ steps modulo $\leftrightarrow_{\mathrm{prune}}^{*}$). □

**Theorem** (= Thm. 4.4)**.** LLEE *coincides with* LEE.

*Proof.* Let $G = \langle V, A, v_{\mathrm{s}}, \to, \Downarrow\rangle$ be a graph.

For showing LLEE($G$) $\Rightarrow$ LEE($G$), we assume that LLEE($G$) holds. This means that there is a marked graph ${}^{\bullet}G_0$ such that ${}^{\circ}G \overset{\circ}{\to}_{\mathrm{elim}} {}^{\bullet}G_0 \not\overset{\circ}{\to}_{\mathrm{elim}}$ and $\neg\infty({}^{\bullet}G_0)$ hold. Then it follows from Lem. A.9, (3), that $G = |{}^{\circ}G|_{\circ} \to_{\mathrm{elim}}^{*} |{}^{\bullet}G_0|_{\circ}$, and by Lem. A.9, (6), that $|{}^{\bullet}G_0|_{\circ} \not\to_{\mathrm{elim}}$. Furthermore $\neg\infty({}^{\bullet}G_0)$ implies $\neg\infty(|{}^{\bullet}G_0|_{\circ})$ by Lem. A.9, (1). In this way we have demonstrated $G \to_{\mathrm{elim}}^{*} |{}^{\bullet}G_0|_{\circ} \not\to_{\mathrm{elim}}$ and $\neg\infty(|{}^{\bullet}G_0|_{\circ})$. Thus we can conclude that LEE($G$) holds.

For showing LEE($G$) $\Rightarrow$ LLEE($G$), we assume that LEE($G$) holds. Then there is a graph $G_0$ such that $G \to_{\mathrm{elim}}^{*} G_0 \not\to_{\mathrm{elim}}$ and $\neg\infty(G_0)$ holds. By statement (5) in Lem. A.9 it follows that there is a marked graph ${}^{\bullet}G_0$ such that ${}^{\circ}G \overset{\circ}{\to}_{\mathrm{elim}}^{*} {}^{\bullet}G_0$ and $|{}^{\bullet}G_0|_{\circ} \leftrightarrow_{\mathrm{prune}}^{*} G_0$. Then $\neg\infty(G_0)$ and $G_0 \leftrightarrow_{\mathrm{prune}}^{*} |{}^{\bullet}G_0|_{\circ}$ entail $\neg\infty(|{}^{\bullet}G_0|_{\circ})$ by Lem. A.6, (iv), and furthermore $\neg\infty({}^{\bullet}G_0)$ by (1) in Lem. A.9. Since $\to_{\mathrm{prune}}$ steps preserve and reflect $\to_{\mathrm{elim}}$ normal forms by Lem. A.6, (iii), it follows from $|{}^{\bullet}G_0|_{\circ} \leftrightarrow_{\mathrm{prune}}^{*} G_0 \not\to_{\mathrm{elim}}$ that also $|{}^{\bullet}G_0|_{\circ}$ is an $\to_{\mathrm{elim}}$ normal form. Then (6) in Lem. A.9 entails that ${}^{\bullet}G_0$ is a $\overset{\circ}{\to}_{\mathrm{elim}}$ normal form. In this way have found that ${}^{\circ}G \overset{\circ}{\to}_{\mathrm{elim}}^{*} {}^{\bullet}G_0 \not\overset{\circ}{\to}_{\mathrm{elim}}$ and $\neg\infty({}^{\bullet}G_0)$ holds. This shows that LLEE($G$) holds. □