Towards a Characterisation of the Graph Structure of Process Interpretations of Regular Expressions

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

We report on work in progress for the characterisation of the structure of process graphs that are process interpretations of regular expressions. Hereby we only aim to recognise the structure of graphs in the image of the interpretation (modulo isomorphism), without permitting to extend this image modulo bisimilarity (or another congruence) as in Milner's process semantics of regular expressions.

At the previous workshop we presented the result that, for a slightly more compact variant of the process interpretation as is commonly used, the image of 'under-star-1-free' regular expressions is closed under the operation of bisimulation collapse. Here we report two significant extensions that we work out now, and at the moment can only label as statements. First, a finite process graph is isomorphic to a compact process interpretation of an under-star-1-free regular expression iff it satisfies the Loop Existence and Elimination property (LEE). Second, a finite process graph G is isomorphic to a compact process interpretation of a regular expression iff G satisfies a generalisation 1-LEE of LEE, which requires that LEE holds for a representation of G with 1-transitions (empty steps).

In this extended abstract we report on two significant extensions of a result that we described at the TERMGRAPH workshop in 2024 [11]. We do so largely informally here, and focus on motivations, intuitions, and examples. Despite of our high confidence in the new results, we cautiously label them as statements at the moment, because we are still writing up the lengthy technical proofs in detail.

1 Introduction

Starting from a formalisation of finite-state processes as μ-terms, Milner (1984, [15]) defined an interpretation P of regular expressions e as regular processes $P(e)$. The interpretations of the constant 0, letters a, sums e_1+e_2, products $e_1 \cdot e_2$, and iterations e^* of expressions e_1, e_2, and 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 the interpretation P, Milner then defined the process semantics $[\![e]\!]_P$ of a regular expression e as a 'behaviour': the bisimilarity equivalence class $[P(e)]_{\leftrightarrows}$ of its process interpretation $P(e)$. Later Antimirov (1996, [1]) introduced, from a different motivation, an interpretation of regular expressions via 'partial derivatives' as non-deterministic finite-state automata that, when understood as transition systems, correspond precisely to the process interpretation.

The process interpretation $P(e)$ of a regular expression e refines the language semantics $[\![e]\!]_L$ of e (defined first by Copi, Elgot, Wright [4]) in the sense that the words of actions on traces to termination of the process $P(e)$ correspond to the words of the regular language $[\![e]\!]_P$. Between these two kinds of semantics there is, however, a striking difference in image completeness in relation to natural classes of

Submitted to:
TERMGRAPH 2026

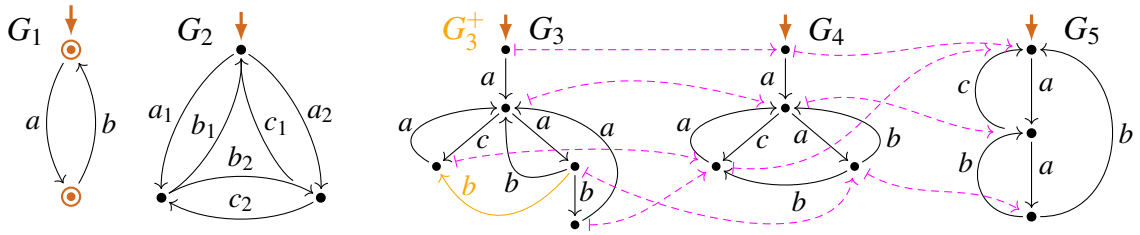

Figure 1: Two neither P- nor $[\![\cdot]\!]_P$-expressible process graphs, and four graphs that are $[\![\cdot]\!]_P$-expressible because G_4 is P-expressible (as $G_4 = P((a\cdot(c\cdot a + a\cdot(b+b\cdot a))^*)\cdot 0)$), and G_3, G_3^+, and G_5 are linked to G_4 via the indicated functional bisimulations. Whether G_3, G_3^+, and G_5 (a bisimulation collapse, solved in [11]) are P-expressible is less obvious. For each graph it is indicated whether LLEE is satisfied. (Start vertices are marked by a brown arrow ➙, vertices with immediate termination by ◉ with a brown ring.)

languages and processes defined by finite-state automata. Whereas the image of the language semantics is complete with respect to languages that are accepted by finite-state automata, the image of the process semantics is incomplete with respect to behaviours of finite-state processes. This is because 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). Examples are the graphs G_1 and G_2 in Fig. 1 (recognised as such by Milner [15] and Bosscher [3]). This phenomenon motivated Milner in [15] to ask for a structural characterization of $[\![\cdot]\!]_P$-expressible graphs. Taken in its strict meaning, however, this question is difficult to answer, because only few structural properties of graphs are preserved under arbitrary bisimulations. For this reason a characterisation of the structure of P-expressible graphs may form a significant step. We provide two characterisations here, a partial one and a global one, based on a concept from previous work. Before that, we and other authors[1] have long settled for workable criteria that recognise P-expressible graphs and that are 'sufficiently general'.

Specifically, Fokkink and I formulated the Loop Existence and Elimination Property LEE [13, 14], which is based on a procedure of eliminating 'loop subgraphs', shorter called 'loops'. These loops are interpretations $P(f^*)$ of iterations f^* of expressions f that are star-free and normed (i.e. $P(f)$ has a trace to successful termination). Loop subgraphs can be peeled off (be eliminated) from a given graph G one by one, as long as such steps are possible. If eventually a graph without an infinite trace (and therefore also without loop subgraphs) is obtained, then LEE holds for G; otherwise it fails. An equivalent, but technically expedient, variant of LEE is 'Layered LEE' (LLEE), which is based on a restricted form of loop elimination: a loop subgraph is eliminated only if its start vertex is not in the body of a previously eliminated loop. The connection between the properties LEE and LLEE and P -expressibility rests, intuitively, on the fact that every successful run of loop elimination on a graph G corresponds to a successful 'parsing' of G while synthesising a regular expression e with $P(e) = G$ or with $P(e) \leftrightarrows G$.

The property LLEE turned out to be a valuable tool for solving axiomatisation problems for the process semantics $[\![\cdot]\!]_P$ of regular expressions [13, 14, 8], based on lemmas such as the following:

(P-1) Process interpretations $P_{\circledast}(f)$ of '1-free' regular expressions with binary star $\circledast$ satisfy LLEE [13].

[1] Apart from the properties LEE and LLEE that we describe and refer to here, Corradini and Baeten had earlier introduced in [2], for this purpose, graphs with 1-transitions (denoting empty steps) that are specified by 'well-behaved' specifications.

(P-2) From every graph G with the property LLEE a regular expression e can be extracted such that $G \in [\![e]\!]_P$ (i.e. G is bisimilar to $[\![e]\!]_P$) [13, 14, 9].

(P-3) The property LLEE is preserved under the operation of bisimulation collapse [13, 14].

While (P-1) shows that graphs with LLEE encompass the process interpretations of 1-free regular expressions (with binary star), statement (P-2) does (by far) not settle a containment in the other direction. For example, while for the four bisimilar graphs G_3, G_3^+, G_4, and G_5 in Fig. 1 the fact that G_4 is P-expressible implies that all are $[\![\cdot]\!]_P$-expressible, it may not be clear immediately whether the graphs G_3, G_3^+, and G_5 are also P-expressible or not. This highlights the question of how the structure of process interpretations of 1-free and of arbitrary regular expressions can be characterised. We report solutions here (see ($P^\bullet$-6) and ($P^\bullet$-4) below) for a compact version $P^\bullet$ of P that slightly increases sharing in $P(e)$ via a functional bisimulation to $P^\bullet(e)$. We motivated $P^\bullet$ at the previous workshop [11], and obtained as refined lemmas:

($P^\bullet$-1) Compact process interpretations $P^\bullet(f)$ of 'under-star-1-free' regular expressions satisfy LLEE.

($P^\bullet$-2) From every graph G with the property LLEE an under-star-1-free regular expression uf can be extracted such that $G \rightrightarrows P^\bullet(uf)$ holds (that is, there is a functional bisimulation from G to $P^\bullet(uf)$).

($P^\bullet$-3) The image under $P^\bullet$ of the under-star-1-free regular expressions is closed under bisim. collapse.

While these lemmas provide insight and are useful, they do not yet settle the characterisation questions. For the graphs in Fig. 1, ($P^\bullet$-3) entails that G_5 is $P^\bullet$-expressible by an under-star-1-free regular expression since it is the collapse of the graph G_4 which is $P^\bullet$-expressible by an under-star-1-free expression; also ($P^\bullet$-1) entails that G_3^+ is not $P^\bullet$-expressible by an under-star-1-free regular expression, because G_3^+ does not satisfy LLEE. However, it remains unclear (and may be difficult to ascertain directly) whether G_3, which satisfies LLEE, is $P^\bullet$-expressible. (It will turn out that G_3^+ is not $P^\bullet$-expressible.)

Based on a slight extension of the 'under-star-1-free' regular expressions (see Def. 2.1) we report here the following characterisation statements of the restricted, and the full image of $P^\bullet$, and a corollary:

($P^\bullet$-4) A process graph G is $P^\bullet$-expressible by an 'under-star-1-free' regular expression if and only if G has the property LLEE. (See Stmt. 4.1.)

($P^\bullet$-5) A process graph G is $[\![\cdot]\!]_P$-expressible (the same as $[\![\cdot]\!]_{P^\bullet}$-expressible) by an 'under-star-1-free' regular expression if and only if the bisimulation collapse of G is $P^\bullet$-expressible. (See Conseq. 4.1)

($P^\bullet$-6) A process graph G is $P^\bullet$-expressible by a regular expression if and only if G is the induced graph of a graph $\underline{G}$ with 1-transitions that has the property LLEE. (See Stmt. 5.1.)

Now it follows from ($P^\bullet$-4) that the graph G_3 with LLEE in Fig. 1 is $P^\bullet$-expressible, and indeed by a under-star-1-free regular expression (see also the expression uf_3 extracted from G_3 later in Fig. 4). The statements ($P^\bullet$-4) and ($P^\bullet$-6) can be viewed as explaining the structure of process graphs in the image of P and $P^\bullet$. Statement ($P^\bullet$-5) is a corollary to ($P^\bullet$-4) in view of ($P^\bullet$-1) and (P-3).

Overview. In Sect. 2 we recapitulate the process interpretation P and its compact variant $P^\bullet$. In Sect. 3 we briefly recall loop elimination, and the property LLEE. In Sect. 4 we report on the characterisation of the image of the process interpretation of under-star-1-free regular expressions. In Sect. 5, we explain the generalisation of this characterisation to the full class of regular expressions. In Sect. 6 we outline two lines of questions that arise from the characterisations as topics for further work. Finally in Appendix A we gather additional explanations of the concepts from previous work on which the new results are based.

2 The process interpretation of reg. expressions, and its compact variant

We first recapitulate from [11] the definitions of (under-star-)1-free (here slightly extended) and normed regular expressions, of the process interpretation P, and of the compact process interpretation $P^\bullet$.

Definition 2.1 (regular expressions)**.** For every set A of actions, the set $RExp(A)$ of *regular expressions over* A, and its subsets $RExp^{(\mathbb{1})}(A)$, $RExp^{(*/\mathbb{1})}(A)$, and $nd\text{-}RExp(A)$ of regular expressions over A that are 1*-free*, *under-star*-1*-free* (abbreviated by $(*/\mathbb{1})$), and *normed*, respectively, are defined by the grammars:

$$\begin{array}{rcll} e,e_1,e_2 & ::= & 0 \mid 1 \mid a \mid e_1+e_2 \mid e_1\cdot e_2 \mid e^* & (\text{where } a\in A) \quad RExp(A) \\ f,f_1,f_2,f_3 & ::= & 0 \mid a \mid f_1+f_2 \mid f_1\cdot f_2 \mid (f_1)^*\cdot f_2 \mid (f_1\cdot(f_2)^*)\cdot f_3 & (\text{where } a\in A) \quad RExp^{(\mathbb{1})}(A) \\ uf,uf_1,uf_2 & ::= & 0 \mid 1 \mid a \mid uf_1+uf_2 \mid uf_1\cdot uf_2 \mid f^* & (\text{where } a\in A) \quad RExp^{(*/\mathbb{1})}(A) \\ n,n_1,n_2 & ::= & 1 \mid a \mid n+e \mid e+n \mid n_1\cdot n_2 \mid e^* & (\text{where } a\in A) \quad nd\text{-}RExp(A) \end{array}$$

Definition 2.2 (process graphs)**.** A *(process) graph* G is a tuple $G=\langle V,A,v_s,\to,\Downarrow\rangle$ that consists of a set V of *vertices*, a set A of *actions*, a specified *start vertex* $v_s\in V$ (hence $V\neq\varnothing$), a *(labelled) transition relation* $\to\subseteq V\times A\times V$, and a predicate $\Downarrow\subseteq V$ on the vertices that indicates immediate successful termination. We write $w_1\xrightarrow{a}w_2$ for a transition $\langle w_1,a,w_2\rangle\in\to$, and $w\Downarrow$ for a vertex $w\in\Downarrow$ that permits immediate successful termination.

Definition 2.3 ((functional) bisimulations, (functionally) bisimilar between process graphs)**.** We consider process graphs $G_i=\langle V_i,A,v_{s,i},\to_i,\Downarrow_i\rangle$, for $i\in\{1,2\}$.

By a *bisimulation between* G_1 *and* G_2 we mean a binary relation $B\subseteq V_1\times V_2$ with the properties that it is non-empty, that is, $B\neq\varnothing$, that relates their start vertices, i.e. $\langle v_{s,1},v_{s,2}\rangle\in B$ holds, and that for every $\langle w_1,w_2\rangle\in B$ the following three conditions hold:

$$\begin{array}{rl} \text{(forth)} & \forall w_1'\in V_1\forall a\in A\big(w_1\xrightarrow{a}_1 w_1' \implies \exists w_2'\in V_2\big(w_2\xrightarrow{a}_2 w_2'\wedge\langle w_1',w_2'\rangle\in B\big)\big), \\ \text{(back)} & \forall w_2'\in V_2\forall a\in A\big(\exists w_1'\in V_1\big(w_1\xrightarrow{a}_1 w_1'\wedge\langle w_1',w_2'\rangle\in B\big) \impliedby w_2\xrightarrow{a}_2 w_2'\big), \\ \text{(termination)} & w_1\Downarrow_1 \iff w_2\Downarrow_2. \end{array}$$

For a partial function $f:V_1\rightharpoonup V_2$ we say that f *defines* a bisimulation between G_1 and G_2 if its graph $\{\langle v,f(v)\rangle\mid v\in V_1\}$ is a bisimulation between G_1 and G_2. We call this graph a *functional* bisimulation.

We denote by $G_1\leftrightarroweq G_2$, and say that G_1 and G_2 are *bisimilar*, if there is a bisimulation between G_1 and G_2. By $G_1\rightrightarrows G_2$ we denote the stronger statement that there is a partial function $f:V_1\rightharpoonup V_2$ whose graph $\{\langle v,f(v)\rangle\mid v\in V_1\}$ is a bisimulation between G_1 and G_2.

Definition 2.4 (process interpretation P)**.** The transition relation $\to\subseteq RExp(A)\times A\times RExp(A)$ and the termination predicate $\Downarrow\subseteq RExp(A)$ for process interpretations of regular expressions $gRExp(A)$ are defined via derivability in the following transition system specification (TSS):

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

$$\frac{e_i\xrightarrow{a}ed}{e_1+e_2\xrightarrow{a}ed}\ (i\in\{1,2\}) \qquad \frac{e_1\Downarrow \quad e_2\xrightarrow{a}ed}{e_1\cdot e_2\xrightarrow{a}ed} \qquad \frac{e_1\xrightarrow{a}ed}{e_1\cdot e_2\xrightarrow{a}ed\cdot e_2} \qquad \frac{e\xrightarrow{a}ed}{e^*\xrightarrow{a}ed\cdot e^*}$$

For every $g\in RExp(A)$, the *process interpretation* $P(g)$ *of* g is then defined as $P(g)=\langle V(g),A,g,\to\cap(V(g)\times A\times V(g)),\Downarrow\cap V(g)\rangle$, where $V(g)$ is the set of vertices that are reachable from g via transitions of $\to$.

We say that a graph G with actions in A is P*-expressible* (resp., *is* P*-expressible by a* $(*/\mathbb{1})$ *regular expression*) if $G=P(e)$ for some $e\in RExp(A)$ (resp., if $G=P(uf)$ for some $uf\in RExp^{(*/\mathbb{1})}(A)$). We say that a graph G is $[\![\cdot]\!]_P$*-expressible* (resp., *is* $[\![\cdot]\!]_P$*-expressible by a* $(*/\mathbb{1})$ *regular expression*) if $G\in[\![e]\!]_P$, and hence $G\leftrightarroweq[\![e]\!]_P$, for some $e\in RExp(A)$ (resp., if $G=[\![uf]\!]_P$, hence $G\leftrightarroweq[\![uf]\!]_P$, for some $uf\in RExp^{(*/\mathbb{1})}(A)$).

As a remedy for an obvious reason why the image of P is not closed under bisimulation collapse (recalled in the appendix) we defined the compact process interpretation $P^\bullet$ in [11]. It arises by preventing the TSS rules from creating unreachable factors in product expressions. That can be implemented with a pruning operation that drops such unreachable factors from product expressions at outermost positions.

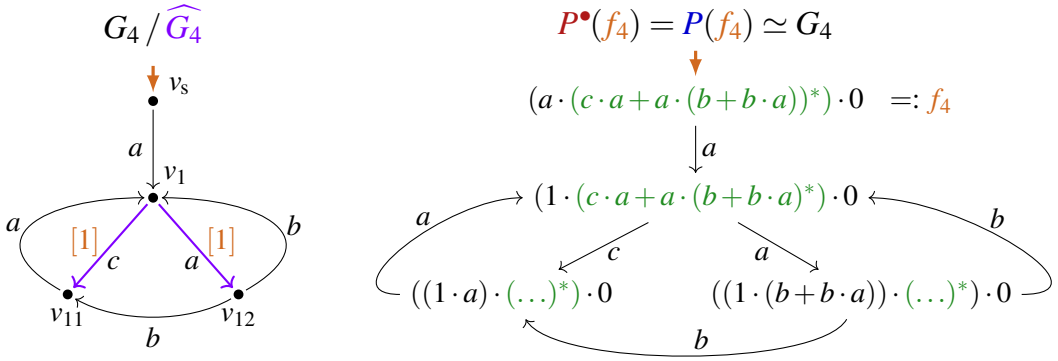

Figure 2: Illustration for recognising the graph G_4 in Fig. 1 as the process interpretation $P(f_4)$, and the compact process interpretation $P^\bullet(f_4)$ of the 1-free regular expression $f_4 := a\cdot(c\cdot a + a\cdot(b+b\cdot a))^*\cdot 0$.

Definition 2.5 (pruning operation π)**.** We define the *pruning operation* $\pi : \mathit{RExp}(A) \to \mathit{RExp}(A)$ recursively by induction on the generation depth of regular expressions e by:

$$\pi(e) := \begin{cases} e_1 \cdot e_2 & \text{if } e = e_1 \cdot e_2 \text{ and } e_1 \text{ is normed,} \\ \pi(e_1) & \text{if } e = e_1 \cdot e_2 \text{ and } e_1 \text{ is not normed,} \\ e & \text{otherwise.} \end{cases}$$

Definition 2.6 (compact process interpretation $P^\bullet$)**.** The transition relation $\to \subseteq \mathit{RExp}(A) \times A \times \mathit{RExp}(A)$ of compact process interpretations $P^\bullet(g)$ of regular expressions g is defined via derivations in the TSS that arises from the TSS in Def. 2.4 by replacing the fourth, and the fifth rule (for products, and iterations) for generating transitions by the corresponding rule that applies the pruning operation to transition targets:

$$\frac{e_1 \xrightarrow{a} ed}{e_1 \cdot e_2 \xrightarrow{a} \pi(ed \cdot e_2)} \qquad \frac{e \xrightarrow{a} ed}{e^* \xrightarrow{a} \pi(ed \cdot e^*)}$$

For the termination predicate $\Downarrow$ for $P^\bullet$ the rules for P apply. For every regular expression $g \in \mathit{RExp}(A)$, the *compact process interpretation* $P^\bullet(g)$ *of* g is then defined as $P^\bullet(g) = \langle V(g), A, \pi(g), \to \cap (V(g) \times A \times V(g)), \Downarrow \cap V(g) \rangle$ where $V(g)$ is the set of vertices that are reachable from $\pi(g)$ via transitions in $\to$.

We say that a process graph G with actions in A is $P^\bullet$*-expressible* (resp., *is* $P^\bullet$*-expressible by a* $(*/1)$ *regular expression*) if $G = P^\bullet(e)$ for some $e \in \mathit{RExp}(A)$ (resp., if $G = P^\bullet(uf)$ for some $uf \in \mathit{RExp}^{(*/1)}(A)$).

As an example for both versions P and $P^\bullet$ of the process interpretation we illustrate in Fig. 2 that the graph G_4 from Fig. 1 is P- and $P^\bullet$-expressible.

Lemma 2.7. *$P(e) \rightleftarrows P^\bullet(e)$, for all $e \in \mathit{RExp}(A)$. Consequently also $P^\bullet(e)$ is finite, for all $e \in \mathit{RExp}(A)$.*

3 Layered Loop Existence and Elimination

The Loop Existence and Elimination Condition LEE, and its layered form LLEE are based on the concept of 'loop subgraph', and an elimination procedure of loop subgraphs. Here we only briefly recall the crucial concepts from [13, 14], illustrate an example in Fig. 3, and refer to more details in the appendix.

Definition 3.1 (loop graphs, loop subgraphs)**.** 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.

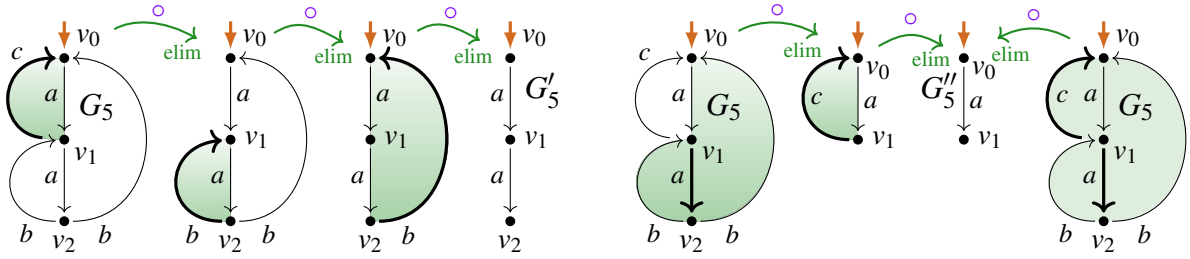

Figure 3: Three different $\overset{\circ}{\to}_{\text{elim}}$ layered loop elimination sequences from the graph G_5 in Fig. 1. For every $\overset{\circ}{\to}_{\text{elim}}$ step the eliminated loop subchart is indicated. These rewrite sequences end in the normal form graphs G'_5 and G''_5 neither of which permits infinite traces. They witness that G_5 satisfies LLEE.

(L2) Every infinite trace from the start vertex v_s of G returns to v_s.
(L3) Immediate successful 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 all other ones *loop-body transitions*.

By $G\downarrow_*^{v,T}$ we denote the subgraph of G with start vertex v that consists of all vertices and transitions that are reachable from v by taking a transition in T, and continue onwards until v (if so) is reached again. 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 a subset $T \subseteq \to$ of transitions from v such that $G\downarrow_*^{v,T}$ is a loop.

A loop elimination step $G \to_{\text{elim}} G'$ in a graph G consists of identifying a loop subgraph $G\downarrow_*^{v,T}$ of G, and then removing (decoupling) the transitions in T, and performing garbage collection (by which all vertices are removed that have become unreachable from the start vertex), obtaining G' as the result. Layered loop elimination steps $\overset{\circ}{\to}_{\text{elim}}$ are history-aware variants of $\to_{\text{elim}}$ on vertex-marked graphs that do not remove loop subgraphs from the bodies (the remnants) of previously eliminated loops (for details see the appendix and [12]). We only define the layered variant LLEE of LEE, which is equivalent to LEE (see also [12]), but technically more convenient for establishing connections with regular expressions.

Definition 3.2 (LLEE)**.** A process graph G satisfies the *layered loop existence and elimination property* LLEE, denoted by LLEE(G), if G has an $\overset{\circ}{\to}_{\text{elim}}$ normal form graph that does not enable an infinite trace.

In Fig. 3 we illustrate three layered loop elimination $\overset{\circ}{\to}_{\text{elim}}$ sequences from the graph G_5 in Fig. 1 to normal form graphs without infinite traces, all of which witness that G_5 satisfies LLEE.

We note that the definition of loop graphs ignores the specific actions on transition traces. Hence the use 'trace' can be replaced everywhere by 'path' in Def. 3.1. It follows that the properties LEE and LLEE are in fact structural properties of the graph structure which are oblivious to action labels of transitions.

The two lemmas below can be shown similar to lemmas in [13, 14]. They guarantee the statements (P-1), (P-2), and ($P^\bullet$-1), that compact process interpretations of $(*/+)$ regular expressions satisfy LLEE.

Lemma 3.3 ($\sim$[13, 14])**.** *Process interpretations $P(uf)$, and compact process interpretations $P^\bullet(uf)$, of $(*/+)$ regular expressions uf satisfy* LLEE*, i.e.* LLEE($P(uf)$), LLEE($P^\bullet(uf)$) *hold for all $uf \in RExp^{(*/+)}(A)$.*

Lemma 3.4 ($\sim$[13, 14] (with Fokkink))**.** *From every graph G with actions in A that satisfies* LLEE *a regular expression $e \in RExp(A)$ can be extracted such that $G \leftrightarroweq P(e) \leftrightarroweq P^\bullet(e)$ (i.e. $G \in [\![e]\!]_P = [\![e]\!]_{P^\bullet}$).*

4 Towards a characterisation of compact process interpretations of under-star-1-free regular expressions

A significant technical refinement of the previously used procedure for extracting from a graph G with LLEE a regular expression e with $P(e) \leftrightarroweq G$, which was used in the proof of Lem. 3.4, made it possible

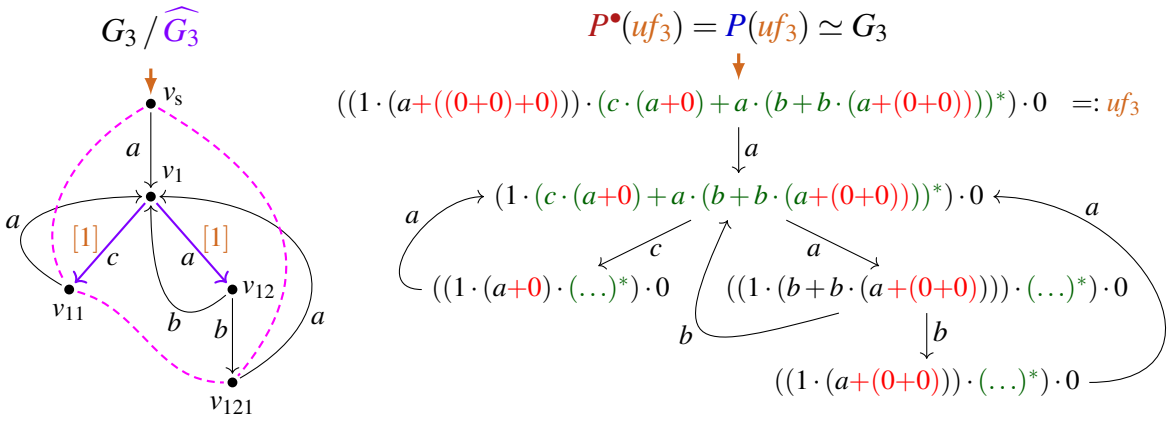

Figure 4: Illustration of the idea underlying the proof of Stmt. 4.1 for the adaptation of refined extraction from the 'LLEE-witness' $\widehat{G_3}$ of the not collapsed graph G_3 in Fig. 1, in order to obtain the $(*/+)$ regular expression uf_3 that represents G_3 via $P^\bullet$ and P. The adaptation distinguishes the values extracted from the three bisimilar vertices v_s, v_{11}, and v_{121} by adding the deadlocking terms 0, $(0+0)$, and $(0+0)+0$ to the extracted solution. Note that G_3 misses, compared to G_3^+ in Fig. 1, one b-transition, but because of that satisfies LLEE, and (like G_3^+) is bisimilar to G_4 and G_5 in Fig. 1. Indeed, G_3^+ and G_3 are not isomorphic, but they are functionally bisimilar to each other via the identity function, and they permit functional bisimulations to G_4 and G_5, that is more formally, G_3^+ $\{\leftleftarrows, \rightrightarrows\}$ $G_3 \rightrightarrows G_4 \rightrightarrows G_5$ holds.

to prove (2024, [11]) the properties ($P^\bullet$-1)–($P^\bullet$-3) for the compact process interpretation $P^\bullet$. Indeed, as a sharpened form of Lem. 3.4 we proved Lem. 4.1 below, which then lead directly to Thm. 4.2 concerning bisimulation collapse (by using that functional bisimulations from collapsed graphs are isomorphisms).

Lemma 4.1 ($=(P^\bullet\text{-}2)$)**.** *From every graph G with actions in A that satisfies* LLEE *a $(*/+)$ regular expression $uf \in RExp^{(*/+)}(A)$ can be extracted such that $G \rightrightarrows P^\bullet(uf)$.*

Theorem 4.2 ($=(P^\bullet\text{-}3)$, [10])**.** *The image of the compact process interpretation $P^\bullet$ when it is restricted to $(*/+)$ regular expressions is closed under bisimulation collapse.*

The extension that we report here is based on an adaptation of the refined extraction procedure. Here we only hint at the basic underlying idea, and demonstrate it for an example in Fig. 4. During bottom-up extraction from a graph G with LLEE it is taken into account which vertices are bisimilar. The locally extracted expressions at bisimilar vertices are now made distinguishable by using syntactically different terms without behaviour. We provide an example in Fig. 4. There, disambiguation is obtained by different summands of 0, which are used for a bottom-up extraction process along a spanning tree described by the LLEE-witness. This technique permits to show the following adaptation, and sharpening, of Lem. 4.1, which directly leads to a characterisation of the image of $P^\bullet$ for $(*/+)$ regular expressions.

Lemma 4.3. *From every graph G with actions in A that satisfies* LLEE *a $(*/+)$ regular expression $uf \in RExp^{(*/+)}(A)$ can be extracted such that $G \simeq P^\bullet(uf)$.*

Statement 4.1 ($=(P^\bullet\text{-}4)$)**.** *A process graph G is $P^\bullet$-expressible by an $(*/+)$ regular expression if and only if G is finite, and satisfies* LLEE*. In other words, the image of the compact process interpretation $P^\bullet$ when restricted to $(*/+)$ regular expressions coincides with the set of finite graphs that satisfy* LLEE*.*

Example 4.4. With Stmt. 4.1 we can now conclude that G_3 and G_5 in Fig. 1, which satisfy LLEE, are $P^\bullet$-expressible, but that G_3^+ is not $P^\bullet$-expressible, because it does not satisfy LEE (nor LLEE).

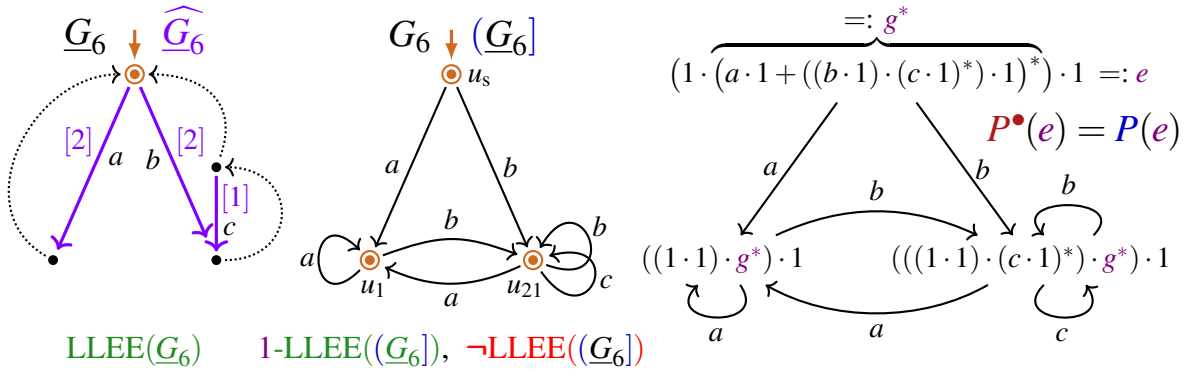

Figure 5: Illustration of a 1-graph $\underline{G_6}$ (see left) with 1-transitions represented by dotted arrows where $\widehat{\underline{G_6}}$ witnesses the property 1-LLEE for its induced graph $G_6 = (\underline{G_6}]$ (see center left), which does not satisfy LLEE. Refined extraction from $\underline{G_6}$ yields the regular expression $e = (1 \cdot g^*) \cdot 1$ with $P(e) \simeq P^\bullet(e) \simeq (\underline{G_6}]$, that is, whose process interpretation (compact and original) coincides with $(\underline{G_6}]$ (see labeled graph right).

From this statement we obtain, with preservation of LLEE under bisimulation collapse (see (P-3)) and Lem. 2.7, the following characterisation of graphs that $[\![\cdot]\!]_P$-expressible by $(*/\dotplus)$ regular expressions.

Consequence 4.1 ($\sim(P^\bullet$-5)). *For every process graph G, the following three statements are equivalent:*

(i) G is $[\![\cdot]\!]_P$-expressible by an $(*/\dotplus)$ regular expression.

(ii) Every bisimulation collapse of G is finite, and satisfies LLEE.

(iii) Every bisimulation collapse of G is $P^\bullet$-expressible by a $(*/\dotplus)$ regular expression.

5 Towards a characterisation of compact process interpretations

For explaining the generalisation of the characterisation of the image of the compact process interpretation $P^\bullet$ from $(*/\dotplus)$ regular expressions to the full class, we first need to introduce the generalised concept of '1-graph' with '1-transitions' that denote empty steps, and define how such a 1-graph $\underline{G}$ represents a graph G via its defined 'induced transitions'. Then we will formulate the generalisation of Stmt. 4.1.

A *process graph G with 1-transitions*, for short a *1-graph*, is a 6-tuple $G = \langle V, A, 1, v_s, \to, \Downarrow \rangle$ that consists of a set V of *vertices*, a set A of *proper actions*, an additional action $1 \notin A$ that is the specified as *empty step label*, a start vertex $v_s \in V$ (hence $V \neq \varnothing$), a labelled transition relation $\to \subseteq V \times \underline{A} \times V$, where $\underline{A} := A \cup \{1\}$ is the set of action labels including 1, and a termination predicate $\Downarrow \subseteq V$.

Definition 5.1 (induced graph of 1-graph). Let $\underline{G} = \langle V, A, 1, v_s, \to, \Downarrow \rangle$ be a 1-graph.

By an *induced a-transition* $v \xrightarrow{(a)} w$, for a proper action $a \in A$, in $\underline{G}$ we mean a trace of the form $v \xrightarrow{1} \cdots \xrightarrow{1} \cdot \xrightarrow{a} w$ in $\underline{G}$ that consists of a finite number of 1-transitions that ends with a proper a-transition. By *induced termination* $v{\downarrow^{(1)}}$, for $v \in V$ we mean that there is a path $v \xrightarrow{1} \cdots \xrightarrow{1} \tilde{v}$ with $\tilde{v}\Downarrow$ in $\underline{G}$.

We define by $(\underline{G}] := \langle V, A, v_s, \xrightarrow{(\cdot)}, \downarrow^{(1)} \rangle$ *the induced* (1-free) *graph of* $\underline{G}$ whose transitions are the induced transitions of $\underline{G}$, and whose terminating vertices are the vertices of $\underline{G}$ with induced termination.

For an example we refer to the graph $\underline{G}$ and its induced graph $(\underline{G}] = G$ in Fig. 5. Based on induced transitions, bisimilarity $\leftrightarroweq$ between graphs can be generalised to 1-bisimilarity $\overset{1}{\leftrightarroweq}$ so that it is guaranteed that $\underline{G_1} \overset{1}{\leftrightarroweq} \underline{G_2}$ if and only if $(\underline{G_1}] \leftrightarroweq (\underline{G_2}]$, for all 1-graphs $\underline{G_1}, \underline{G_2}$. We now define the property 1-LLEE, and recall two of its properties that are analogous to, and generalizations of, Lem. 3.3 and Lem. 3.4.

Definition 5.2 (1-LLEE)**.** We say that a graph G *satisfies the property* 1-LLEE, denoted by 1-LLEE(G), if G is the induced graph of a 1-graph $\underline{G}$ with LLEE (i.e. $\underline{G}$ is such that $(\underline{G}] = G$ and LLEE($\underline{G}$) holds).

We note here that, unlike the property LLEE which is an 'action-oblivious' requirement on the path structure of process graphs, the property 1-LLEE usually takes actions on the transitions of the considered graph into account. Indeed that is the case, typically, for some graphs G that do not satisfy LLEE, but that have a '1-transition refinement' $\underline{G}$ whose induced graph $(\underline{G}]$ is G and satisfies LLEE (for example, see the graph $G_6 = (\underline{G_6}]$ in Fig. 5). When, in such a case, G is refined into $\underline{G}$ only transitions with the same actions can be shared via added 1-transitions.

Lemma 5.3 (generalises Lem. 3.3, [8])**.** *Process interpretations $P(e)$ of regular expressions e satisfy* 1-LLEE*, that is more formally,* 1-LLEE($P(e)$) *holds for all $e \in RExp(A)$.*

Lemma 5.4 (generalises Lem. 3.4, [6, 5, 10])**.** *From every 1-graph $\underline{G}$ with actions in A that satisfies* LLEE *a regular expression $e \in RExp(A)$ can be extracted such that $\underline{G} \leftrightarrows P(e)$.*

Now the key step to showing that 1-LEE characterises P-expressible graphs consists in the following lemma, which can be shown in analogy to Lem. 4.3 by an adaptation of the same form of the refined extraction procedure that underlies Lem. 4.1, but now applied to 1-graphs and their induced graphs.

Lemma 5.5 (generalises Lem. 4.3, refines Lem. 5.4)**.** *From every 1-graph $\underline{G}$ with actions in A that satisfies* LLEE *a regular expression $e \in RExp(A)$ can be extracted such that $(\underline{G}] \simeq P^\bullet(e)$.*

Statement 5.1 (=($P^\bullet$-6))**.** *A process graph G is $P^\bullet$-expressible by a regular expression if and only if G is finite, and G satisfies* 1-LLEE*. Equivalently, the image of the compact process interpretation coincides with the set of finite process graphs that satisfy* 1-LLEE*.*

6 Conclusion

Our first aim is to finish writing up the proofs for statements Stmt. 4.1 and Stmt. 5.1 in detailed, and yet convincing form. But furthermore we are interested in two further lines of questions.

First, the two characterisations reported here are likely not on the same level with respect to computational complexity. We can argue that Stmt. 4.1 may be practically useful, because LEE and LLEE can be recognised in cubic time depending on the graph size (but likely much faster algorithms are possible), as we argue in [12] where we show that loop elimination can be turned into a confluent rewrite system by adding a pruning operation. However, practicality of the recognition of 1-LLEE is less clear. To determine whether 1-LLEE(G) holds for a graph G, requires to check whether there is a 1-transition refinement $\underline{G}$ of G with LLEE and $(\underline{G}] = G$. While the search space for such a refinement is a priori exponential, there are several ways to reduce it. A limiting phenomenon, however, may be that, unlike LLEE (see (P-3)), 1-LLEE is not closed under bisimulation collapse, as we showed in [10]. For these reasons we want to investigate more closely the computational complexity of the characterisation in Stmt. 5.1.

Second, the refined extraction procedure whose adaptation led to the new characterisations uses, as the previously used procedure, LLEE-witnesses, which can be viewed as certificates of successful *layered* loop elimination $\overset{\circ}{\to}_{\mathrm{elim}}$ sequences. Yet examples show that extraction of regular expressions is also possible from successful loop elimination $\to_{\mathrm{elim}}$ sequences. It is therefore desirable to generalise the (refined and adapted) extraction procedures from using LLEE-witnesses to using 'LEE-witnesses'.

Acknowledgment. We sincerely thank the reviewers for their observations and questions. In particular, for their close reading, for the detailed comments that pointed out overlooks, for suggestions to improve readability, and for the question about additional motivation. We are acutely aware that we will be able to make good on some of those points only in the presentation at the workshop and later.

References

[1] Valentin Antimirov (1996): *Partial Derivatives of Regular Expressions and Finite Automaton Constructions*. *Theoretical Computer Science* 155(2), pp. 291–319, doi:https://doi.org/10.1016/0304-3975(95)00182-4.

[2] Jos Baeten & Flavio Corradini (2005): *Regular Expressions in Process Algebra*. In: *Proceedings of LICS 2005*, IEEE Computer Society 2005, pp. 12–19, doi:10.1109/LICS.2005.43.

[3] Doeko Bosscher (1997): *Grammars Modulo Bisimulation*. Ph.D. thesis, University of Amsterdam.

[4] Irving M. Copi, Calvin C. Elgot & Jesse B. Wright (1958): *Realization of Events by Logical Nets*. *Journal of the ACM* 5(2), doi:10.1007/978-1-4613-8177-8_1.

[5] Clemens Grabmayer (2021): *A Coinductive Version of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.2108.13104. arXiv:2108.13104.

[6] Clemens Grabmayer (2021): *A Coinductive Version of Milner's Proof System for Regular Expressions Modulo Bisimilarity*. In Fabio Gadducci & Alexandra Silva, editors: *9th Conference on Algebra and Coalgebra in Computer Science (CALCO 2021)*, *Leibniz International Proceedings in Informatics (LIPIcs)* 211, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, pp. 16:1–16:23, doi:10.4230/LIPIcs.CALCO.2021.16. Extended report see [5].

[7] Clemens Grabmayer (2021): *Structure-Constrained Process Graphs for the Process Semantics of Regular Expressions*. *Electronic Proceedings in Theoretical Computer Science* 334, p. 29–45, doi:10.4204/eptcs.334.3.

[8] Clemens Grabmayer (2022): *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, Association for Computing Machinery, New York, NY, USA, pp. 1–13.

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

[10] Clemens Grabmayer (2023): *The Image of the Process Interpretation of Regular Expressions is Not Closed under Bisimulation Collapse*. Technical Report `arXiv:2303.08553`, `arxiv.org`. Version 2 (November), `https://arxiv.org/pdf/2303.08553`.

[11] Clemens Grabmayer (2024): *Closing the Image of the Process Interpretation of 1-Free Regular Expressions under Bisimulation Collapse*. Extended abstract and slides for the Workshop TERMGRAPH 2024, satellite event of ETAPS 2024, 7 April 2024, Luxembourg City, Luxembourg. Available at `https://clegra.github.io/lf/closing-bc-i-pi-us1f.pdf` (extended abstract) and `https://clegra.github.io/lf/TG-2024.pdf` (slides).

[12] Clemens Grabmayer (2026): *Loop Elimination in Process Graphs is Confluence when Pruning Steps are Added*. In: *Proceedings of IWC 2026 (15th Int. Workshop on Confluence)*, p. (known only after publication). To appear. Also available at `https://clegra.github.io/lf/le-conf-IWC-26.pdf`.

[13] Clemens Grabmayer & Wan Fokkink (2020): *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, Association for Computing Machinery, New York, NY, USA, p. 465–478, doi:10.1145/3373718.3394744.

[14] Clemens Grabmayer & Wan Fokkink (2020): *A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity*. Technical Report, `arxiv.org`, doi:10.48550/arXiv.2004.12740. arXiv:2004.12740. Report version of [13].

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

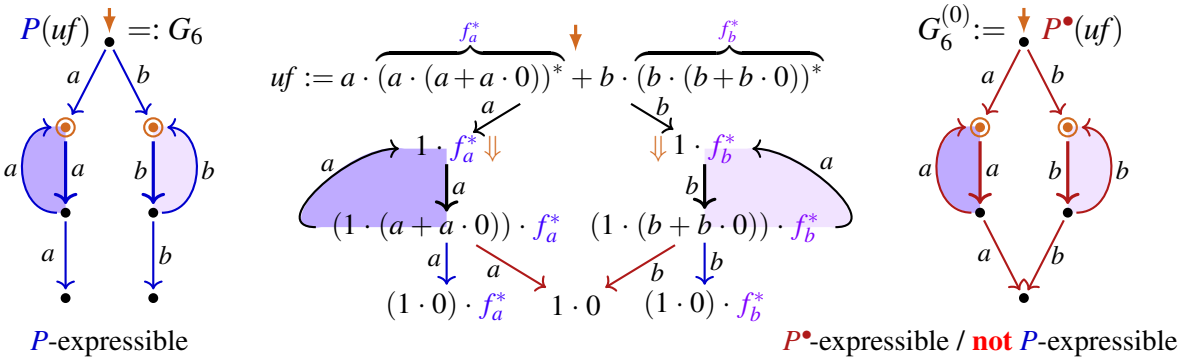

Figure 6: Example of an $(*/+)$ regular expression uf that highlights why the process interpretation P cannot express all graphs with LLEE (see the loop entries in bold), while the compact process interpretation $P^\bullet$ can: For P, transitions that depart from strongly connected components that are in parallel positions (here colored indigo and darkcyan) carry trailing subexpressions of the different iterations that describe (also) the scc's (here f_a^* and f_b^*) although these iterations denote unreachable parts in the not normed target expressions at the bottom left and right. This does not happen for $P^\bullet$, because in it trailing unreachable parts are dropped from expressions that are not normed.

A Appendix

In this appendix we supplement the exposition in the extended abstract with some more details for the Sections 2, 3, 4, and 5 in the Subsections A.1, A.2, A.3, and A.4, respectively. In the central Subsection A.2 we recapitulate the definitions of the Layered Loop Elimination Condition LLEE and of the concept of LLEE-witness from our previous work in [13, 6, 9].

A.1 Supplements for Section 2: the process interpretation and its compact variant

For the well-definedness as a finite process graph of the process interpretation $P(e)$ of a regular expression as defined in Def. 2.4 it is possible to refer to Antimirov's equivalent definition of P by means of partial derivatives of regular expressions.

Proposition A.1 (Antimirov [1]). *$P(e)$ is a finite graph, for every $e \in RExp(A)$.*

Here we recall the example and a picture that we used at TERMGRAPH 2024 for explaining why the image of the process interpretation P restricted to under-star-1-free regular expression is not closed under bisimulation collapse, but that the compact process interpretation $P^\bullet$ can provide a remedy.

Example A.2. The $(*/+)$ regular expression:

$$uf := a \cdot (a \cdot (a + a \cdot 0))^* + b \cdot (b \cdot (b + b \cdot 0))^* \in RExp^{(*/\mathbf{1})}(\{a,b\})$$

has the process interpretation $P(uf) = G_6$ in Fig. 6 on the left. We note that $P(uf)$ is not a bisimulation collapse, because the two vertices on the bottom are bisimilar.

Since the transitions to deadlock vertices from within the graph parts of the iterations f_1^* and f_2^* take place not from the loop vertices, but from the bodies of their loop, and since both loop vertices permit immediate termination, there does not exist another regular expression ug whose process interpretation $P(ug)$ is the bisimilar LLEE-graph $G_6^{(0)}$ on the right in Fig. 6.

We summarise these observations in the following proposition.

Proposition A.3. *The image of the process interpretation P, even when restricted to $(*/+)$ regular expressions, is not closed under the operation of taking a bisimulation collapse. This is witnessed by the $(*/+)$ expression $uf \in RExp^{(*/\mathbf{1})}(\{a,b\})$ in Exa. A.2 and Fig. 6 with process interpretation $P(uf) = G_6$ and bisimulation collapse $G_6^{(0)}$.*

A.2 Supplements for Section 3: Layered Loop Existence and Elimination

First in Sect. A.2.1 we recapitulate the definition of LEE and LLEE in more detail. Subsequently in Sect. A.2.2 we explain the definition of witnesses of these two properties (of which LLEE-witnesses are instrumental for the definition of extraction procedures of regular expressions from graphs with LEE).

A.2.1 The (Layered) Loop Existence and Elimination condition (L)LEE

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

We call the graph G *finite* if its set V of vertices and its set A of actions are finite (then also its set $\to$ of transitions is finite).

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 not an infinite trace from v in G). Furthermore we denote by $\infty(G)$ (resp. by $\neg\infty(G)$) that $\infty(v_s)$ holds (and resp. that $\neg\infty(v_s)$ holds) for the start vertex v_s of G.

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 A.4 (= Def. 3.1 on loop graphs, loop subgraphs)**.** 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, i.e. $\infty(v_s)$ holds.

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

(L3) Immediate successful 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.

Example A.5. In Fig. 7 we illustrate four non-examples for loop graphs, each of which violates one of the loop conditions (L1), (L2), or (L3), and also an example of a loop graph, and of a loop subgraph.

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. 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 $\mathsf{gc}(G) := \langle V_0, A, v_s, \to \cap (V_0 \times A \times V_0), \Downarrow \cap V_0 \rangle$.

Definition A.6 (loop elimination $\to_{\mathrm{elim}}$)**.** We define loop elimination steps $\to_{\mathrm{elim}}$ as follows:

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

(where $G = \langle V, A, v_s, \to, \Downarrow \rangle$)

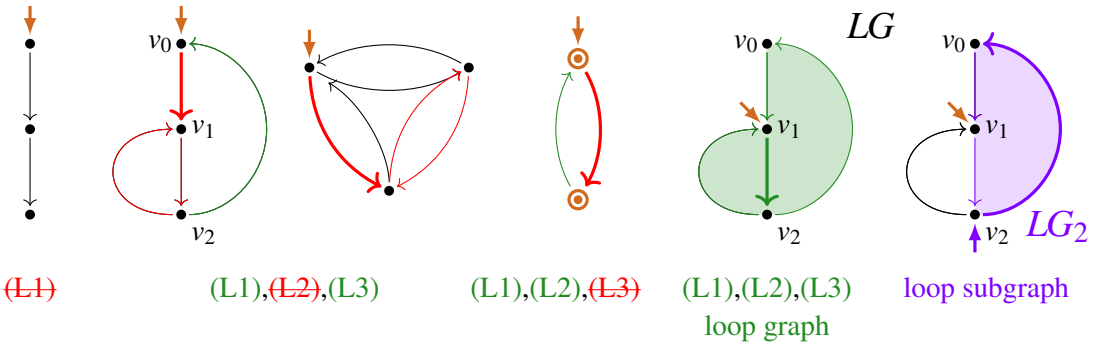

Figure 7: Four process graphs (action labels ignored) that violate at least one of the loop graph conditions (L1), (L2), and (L3), then a loop graph LG, and finally a loop subgraph LG_2 of LG.

It can be shown that the loop elimination rewrite relation $\to_{\text{elim}}$ can be extended into a confluent rewrite relations by adding prune steps $\to_{\text{prune}}$ which remove single transitions to deadlocking vertices (i.e. to vertices without immediate termination, and without departing transitions). We present that related result at the FLoC'26 workshop IWC, see the extended abstract [12].

Definition A.7 (LEE)**.** We say that a process graph G satisfies the *loop existence and elimination property* LEE, denoted by $\mathrm{LEE}(G)$, if G has an $\to_{\text{elim}}$ normal form graph that does not enable an infinite trace. More formally, we stipulate:

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

Layered loop elimination steps $\overset{\circ}{\to}_{\text{elim}}$ are history-aware variants of $\to_{\text{elim}}$. They can be defined on vertex-marked graphs, in which the body vertices of already eliminated loop subcharts (but not their loop vertices) are recorded as marked. Then layered elimination steps $\overset{\circ}{\to}_{\text{elim}}$ are defined by permitting the elimination of loop subgraphs only at yet unmarked vertices, while forbidding to remove loop subgraphs at marked vertices. Please also see the extended abstract [12] for IWC 2026 for more details on $\overset{\circ}{\to}_{\text{elim}}$.

Definition A.8 (LLEE)**.** We say that a process graph G satisfies the *Layered Loop Existence and Elimination Property* LLEE, denoted by $\mathrm{LLEE}(G)$, if G has an $\overset{\circ}{\to}_{\text{elim}}$ normal form graph that does not enable an infinite trace. More formally, we stipulate:

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

where ${}^{\circ}G$ is G viewed as marked graph without marked vertices, and ${}^{\bullet}G_0$ denotes a vertex-marked graph.

We mention that it can be shown that although the property LLEE is a formally stronger requirement than the property LEE, which often helps to simplify proofs, both properties are equivalent.

Example A.9. For the process graph G in Fig. 8 the properties LEE and LLEE can be witnessed by a sequence of three loop elimination steps there that lead to graph G''' without infinite traces.

The not $[\![\cdot]\!]_P$-expressible graphs G_1 and G_2 in Fig. 1 do not satisfy LLEE and LEE, since loop elimination is not successful on them: they do not enable loop-elimination steps, but facilitate infinite traces.

The induced graph $(G]$ of G in Fig. 5 does not satisfy LEE nor LLEE. This is because after elimination of the three (self-)loops at u_1 and u_{21} no further loop subcharts remain (the vertices with immediate successful termination create violations of the loop condition (L3) for generated subgraphs that fulfill loop condition (L1)), and yet the then remaining graph still permits an infinite trace.

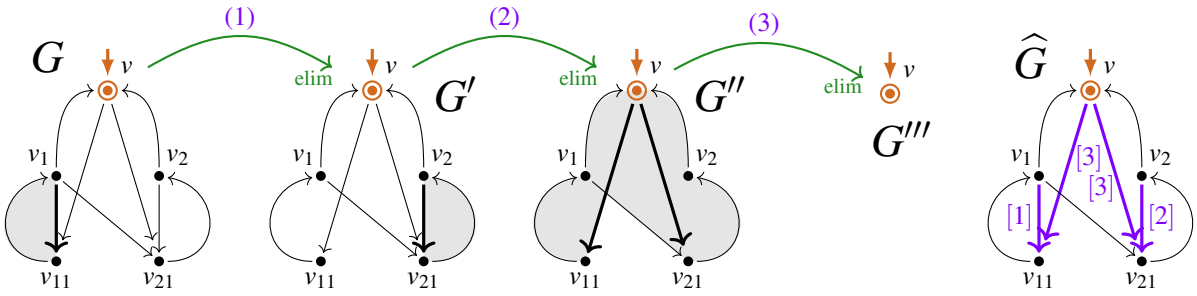

Figure 8: Example of successful loop elimination from the process graph G: three elimination steps of loop subcharts, which are represented as shaded gray areas, lead to the process graph G''' without infinite behaviour. These steps witness that G satisfies the properties LEE and LLEE (as well as do G', G'', G'''). Then the marking labeled version $\widehat{G}$ on the right is a LEE-witness (also a LLEE-witness) of G in which the loop-entry transitions and the step numbers in which they have been eliminated have been recorded.

Remark A.10 (paths suffice to define loops, and (L)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. 3.1) can be replaced by 'infinite path'. It follows that the action labels in graphs are also irrelevant for the definition of the property LEE in Def. A.7. As a consequence, LEE and LLEE can be understood as properties of the path structure of process graphs.

We mention that it can be shown that although the property LLEE is a formally stronger requirement than the property LEE, which often helps to simplify proofs, both properties are equivalent.

Proposition A.11. *A process graph satisfies* LLEE *if and only if it satisfies* LEE*.*

A.2.2 (L)LEE-witnesses

The idea behind the concept of a witness of one of the properties LEE or LLEE (layered LEE) is illustrated in Fig. 8 for the example process graph G there for which LEE and LLEE were recognised to hold due to a successful 3-step loop elimination process. A LEE-witness $\widehat{G}$ of the graph G in Fig. 8, which in this case is also a LLEE-witness, can be viewed as the 'recording' of the successful 3-step run of the loop-elimination procedure on G by adding additional labels on to the transitions of G that represent the run. More precisely, for each loop-entry transition that is removed in one of the loop-elimination steps the number n of the step in the successful run of the loop-elimination procedure is added to the action label of the transition as an additional marking label $[n]$. Finally all other transitions are labelled as 'body transitions' (which we do not draw differently in pictures in contrast with the added number annotations of loop-entry transitions). More formally, LEE-witnesses and LLEE-witnesses are defined in Def. A.13 below. For the purpose of their definition we use a multi-step version $\multimap\!\!\circ\!\!\rightarrow_{\mathrm{elim}}$ of the single-step loop-elimination rewrite relation $\rightarrow_{\mathrm{elim}}$. Hereby $\multimap\!\!\circ\!\!\rightarrow_{\mathrm{elim}}$ permits to eliminate, in a single step of $\multimap\!\!\circ\!\!\rightarrow_{\mathrm{elim}}$, several loop-entry transitions at different vertices in a process graph.

Definition A.12 (entry/body-labeling)**.** Let $G = \langle V, A, v_s, \rightarrow, \Downarrow \rangle$ be a labelled transition graph with immediate-termination predicate.

By an *entry/body-labeling* of G we mean a labelled transition graph $\widehat{G} = \langle V, A \times \mathbb{N}, v_s, \hat{\rightarrow}, \Downarrow \rangle$ with actions in $A \times \mathbb{N}$ (and with immediate-termination predicate) that results from G by attaching to every transition of G an additional *marking label* in $\mathbb{N}$ (the labelled transitions in $\hat{\rightarrow}$ are marking-labelled

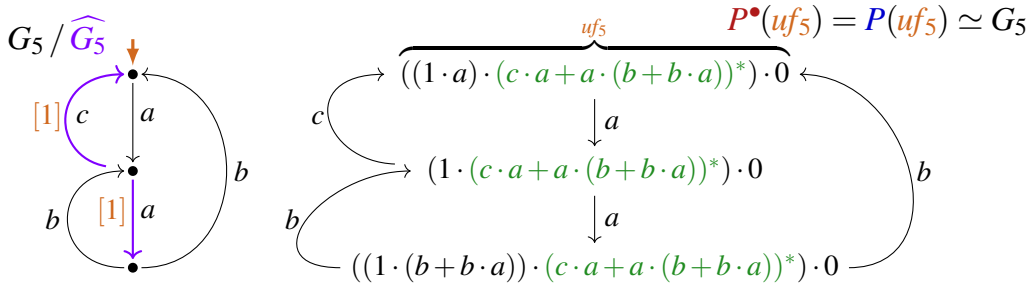

Figure 9: Refined extraction from the LLEE-witness $\widehat{G_5}$ of the graph G_5, the bisimulation collapse of G_3, G_3^+, and G_4, in Fig. 1) yields a $(*/+)$ regular expression of which uf_5 above is a slight simplification. From the illustration right it can be checked that $G_5 \simeq P(uf_5) = P^\bullet(uf_5)$, confirming Conseq. 4.1 in this case: collapsed graphs with LLEE are $P^\bullet$-expressible. Note the subtle difference between uf_5 and the 1-free regular expression $f_4 = (a\cdot(c\cdot a + a\cdot(b+b\cdot a))^*)\cdot 0$ with $P^\bullet(f_4) = P(f_4) = G_4$ in Fig. 2.

versions of the labelled transitions in $\to$). In an entry/body-labeling we call transitions with marking label 0 *body transitions*, and transitions with marking labels in $\mathbb{N}^+$ *entry transitions*.

Definition A.13 (LEE-witness, and LLEE-witness (layered LEE-witness))**.** Let $G = \langle V, A, v_s, \to, \Downarrow\rangle$ be a labelled transition graph with immediate-termination predicate.

A LEE-*witness* $\widehat{G}$ *of* G (and respectively, a LLEE-*witness (layered* LEE-*witness)* $\widehat{G}$ *of* G) is an entry/body-labeling $\widehat{G}$ of G that is the recording of a LEE-guaranteeing (and resp., a LLEE-guaranteeing), successful run $G \multimap\!\!\to^*_{\mathrm{elim}} G'$ of the multi-step loop elimination procedure on G: it results by attaching to a transition τ of G with action label $a \in A$ the marking label n for $n \in \mathbb{N}^+$, obtaining aggregated label $\langle a, n\rangle$ and forming a *loop-entry transition*, if τ is eliminated in the n-th multi-step, and by attaching marking label 0 to all other transitions of G, obtaining aggregated label $\langle a, 0\rangle$ and thereby forming a *body transition*. In pictures we indicate loop-entry transition marking labels $n \in \mathbb{N}^+$ as $[n]$, in steps as $\to_{[n]}$, and we usually neglect body transition marking labels, but in steps we write $\to_{\mathrm{bo}}$.

Example A.14. It is easy to see that the graph $\widehat{G}$ in Fig. 8 illustrates an entry/body-labeling of the graph G there, and that $\widehat{G}$ is actually a LEE-witness as well as a LLEE-witness of G.

A.3 Supplements for Section 4

In order to facilitate a comparison with the adapted refined extraction in Fig. 4 from the graph G_3 in Fig. 1, we repeat here the extraction example concerning G_5 in Fig. 4 that we have used before in [11].

Example A.15. In Fig. 9 we illustrate the completed refined extraction of the regular expression:

$$uf_5 := ((1\cdot a)\cdot(c\cdot a + a\cdot(b+b\cdot a))^*)\cdot 0$$

from a LLEE-witness $\widehat{G_5}$ of the bisimulation collapse graph G_5 in Fig. 1.

A.4 Supplement for Section 5

The following lemma is a more explicit form of Lem. 5.3, which states that compact process interpretations satisfy 1-LLEE. We proved it first in the article [7] for TERMGRAPH 2020. Later in [10, 8] we modified and simplified the TSS that we first introduced in [7] for producing, for process interpretations $P(e)$ of regular expressions from the full class, 1-transition refinements that satisfy LLEE.

Lemma A.16 ($\sim$[7, 10, 8])**.** *There is a* 1*-transition refinement* $\underline{P}$ *of the process interpretation* P *that maps any regular expressions* $e \in RExp(A)$ *to a* 1*-graph* $\underline{P}(e)$ *that satisfies* LLEE*, and whose induced graph is isomorphic to the process interpretation* $P(e)$ *of* e*, more formally,* LLEE$(\underline{P}(e))$*, and* $(\underline{P}(e)] \simeq P(e)$ *hold.*