Untangling Concurrent Systems for Analysis of Behavioral Properties

Artem Polyvyanyy[1], Marcello La Rosa[1,3],
Chun Ouyang[1,3], and Arthur H.M. ter Hofstede[1,2,3]

[1] Queensland University of Technology, Brisbane, Australia
[2] Eindhoven University of Technology, Eindhoven, The Netherlands
[3] NICTA Queensland Lab, Australia
{artem.polyvyanyy;m.larosa;c.ouyang;a.terhofstede}@qut.edu.au

Abstract. Substantial research efforts have been expended to deal with the state space explosion problem inherent to the analysis of concurrent systems. Approaches differ in the classes of properties that they are able to suitably check and this is largely a result of the way they balance the trade-off between time and space. One interesting class of properties is concerned with behavioral characteristics of a concurrent system. These properties are conveniently expressed in terms of runs through the system. In this paper, the theory of *untangling* is proposed that exploits a particular representation of collections of runs to facilitate the analysis. It is shown that no behavior is lost or added when untangling a concurrent system.

1 Introduction

Systems in which several computations are executing simultaneously, usually interacting with each other, are called *concurrent*. Naturally, concurrent systems are capable of providing a better throughput, computed output relative to the number and size of tasks at hand, as compared to systems in which all operations are performed in a centralized manner by a single worker, e.g., a processing unit. The design of a concurrent system deals with finding an effective coordination that encourages efficiency of individual workers in the name of a common goal.

Concurrent systems are often extremely complex. The complexity is primarily due to the amount of uncertainty about the order in which atomic operations can be executed by different processing units of a concurrent system. The number of all possible interleavings of operations defined by a concurrent system grows combinatorially with the number of its simultaneously enabled operations – a phenomenon known as the state space explosion problem. Consequently, techniques for analysis of concurrent systems constitute one of the most complex classes of practical problems in computer science. Many techniques for verification of behavioral characteristics, i.e., properties over all potential computations, are doomed to explore immense portions of operation interleavings described by a concurrent system. Besides, the complexity of these explorations depends on the particular formalism that is used to describe the system.

A number of formalisms for modeling and analyzing concurrent systems have been developed and systematized [1,2]. Some of them can be used through the entire development cycle while others target some particular phase, e.g., analysis. One of the first formalisms for describing concurrent systems, and by now one of the most studied, was proposed in Carl Adam Petri's seminal work on Petri nets. A concurrent system captured in the Petri net notation, or a net system, is often reminiscent of a mass of highly interwoven models for individual computations – a *tangle* of computations. Every time a Petri net gets instantiated it carves its way through a maze of pre-designed options into a collection of desired computations. This paper proposes theoretical foundations for a novel representation of concurrent systems. We propose to untie Petri net models of concurrent systems into their representative *untanglings*, i.e., to free them from the confusion of individual computations.

An untangling of a concurrent system is a set of its *processes* [3,4,5], i.e., abstract models for representing causality and concurrency of operations, each describing a collection of system computations. A representative untangling of a Petri net system describes the exact behavior of its originative system, is usually larger than the originative system (with respect to size), but is easier to analyze. We envision that the proposed theory will eventually lead towards a design of a novel index structure for representing behaviors of concurrent systems, suitable for their effective and efficient analysis.

The remainder of this paper is organized as follows. The next section introduces preliminary notions that will be used later to impart the findings. Section 3 proposes process set systems – abstract models capable of representing chunks of the non-sequential behavior of concurrent systems. Afterwards, Section 4 is devoted to the main artifact which is developed in this paper – a representative untangling of a concurrent system. A representative untangling can be used to induce a process set system that is behaviorally equivalent to the original concurrent system, but structurally different. Section 5 demonstrates use of representative untanglings for analysis of concurrent systems. The paper closes with a detailed discussion of related work, ideas for future work, and conclusions.

2 Preliminaries

This section introduces formalisms that will be used to support discussions in the subsequent sections. First, Section 2.1 presents *Petri nets* – a formalism for describing concurrent systems. Afterwards, Section 2.2 talks about *processes* – abstract models that capture the non-sequential behavior of concurrent systems.

2.1 Petri Nets

Petri nets are a well-known formalism for modeling concurrent systems. This section introduces the basic Petri net terminology and notations.

Definition 2.1 (Petri net) A *Petri net*, or a *net*, is an ordered triple $N := (P, T, F)$, where P and T are finite disjoint sets of *places* and *transitions*, respectively, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow* relation. ⌟

An element $x \in P \cup T$ is a *node* of N. A node $x \in P \cup T$ is an *input* node of a node $y \in P \cup T$ iff x and y are in the flow relation, i.e., $(x, y) \in F$. Similarly, a node

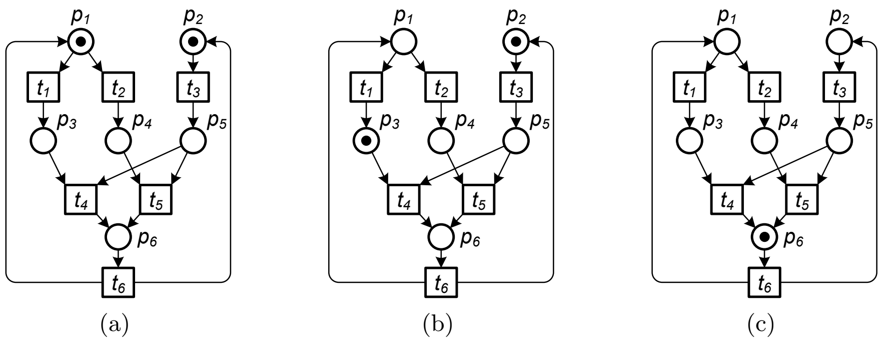

Fig. 1. Net systems (a net at different markings)

$x \in P \cup T$ is an *output* node of a node $y \in P \cup T$ iff y and x are in the flow relation, i.e., $(y, x) \in F$. By $\bullet x$, $x \in P \cup T$, we denote the *preset* of x – the set of all input nodes of x, i.e., $\bullet x := \{y \in P \cup T \mid (y, x) \in F\}$. Likewise, by $x\bullet$, $x \in P \cup T$, we refer to the *postset* of x – the set of all output nodes of x, i.e., $x\bullet := \{y \in P \cup T \mid (x, y) \in F\}$. For a set of nodes $X \subseteq P \cup T$, $\bullet X := \bigcup_{x \in X} \bullet x$ and $X\bullet := \bigcup_{x \in X} x\bullet$. A node $x \in P \cup T$ is a *source* (*sink*) node of N iff $\bullet x = \varnothing$ ($x\bullet = \varnothing$).

Given a net $N := (P, T, F)$, by $Min(N)$ we denote the set of all source nodes of N, i.e., $Min(N) := \{x \in P \cup T \mid \bullet x = \varnothing\}$. Similarly, $Max(N)$ denotes the set of all sink nodes of N, i.e., $Max(N) := \{x \in P \cup T \mid x\bullet = \varnothing\}$. For technical convenience, we require all nets to be T-restricted. A net N is *T-restricted* iff the preset and postset of every transition is non-empty[4], i.e., $\forall\, t \in T : \bullet t \neq \varnothing \neq t\bullet$.

Execution semantics of a Petri net is based on the notion of a *marking.* A *marking* of a Petri net is the distribution of *tokens* over the net's places.

Let $\mathbb{K}$ be a universe of tokens.

Definition 2.2 (Marking of a net)
A *marking* of a net $N := (P, T, F)$ is an ordered pair $M := (K, \mu)$, where $K \subseteq \mathbb{K}$ is a set of *tokens* and $\mu : K \to P$ assigns a place to each token. ⌟

Finally, a net system is a net at a certain marking.

Definition 2.3 (Net system) A *net system*, or a *system*, is an ordered pair $S := (N, M)$, where N is a net and M is a marking of N. ⌟

In the graphical notation, a common practice is to visualize places as circles, transitions as rectangles, the flow relation as directed edges, and tokens as black dots inside assigned places; see Fig. 1 for visualization examples of net systems.

Whether a transition is *enabled* depends on tokens in its input places. An enabled transition can *occur*, which leads to a fresh marking.

Definition 2.4 (Semantics of a net system)
Let $S := (N, M)$, $N := (P, T, F)$, $M := (K, \mu)$, be a net system.

- ◦ A transition $t \in T$ is *enabled* in S, denoted by $S[t\rangle$, iff every input place of t contains at least one token, i.e., $\forall\, p \in \bullet t\ \exists\, k \in K : \mu(k) = p$.

[4] Every Petri net can be trivially T-restricted by adding a single input (output) place to every transition with empty preset (postset).

- If a transition $t \in T$ is enabled in S then t can *occur*, which leads to a net system $S' := (N, M')$, where $M' := (K', \mu')$ is a marking obtained by removing one token from every input place of t and putting one fresh token at every output place of t as follows:

  Let $\hat{K} \subseteq \mathbb{K} \setminus K$ be a set of tokens such that $|\hat{K}| = |t\bullet|$ and let $g : \hat{K} \to t\bullet$ be a bijection between $\hat{K}$ and $t\bullet$. Let $f \subseteq \mu \rhd \bullet t$, with f a bijection and $|dom(f)| = |\bullet t|$. Then, $K' := K \bigtriangleup (dom(f) \cup dom(g))$ and $\mu' := \mu \bigtriangleup (f \cup g)$.[5] ⌟

By $S[t\rangle S'$, we denote the fact that there exists an occurrence of transition t that leads from S to S'. Observe that the semantics of net systems relies on holdings of tokens at places and is independent from the tokens' identities. This fact gives rise to the following relation on markings. Two markings $M_1 := (K_1, \mu_1)$ and $M_2 := (K_2, \mu_2)$ of a net are *equivalent*, denoted by $M_1 \equiv M_2$, if and only if there exists a bijection $\beta : K_1 \to K_2$ such that $\mu_1(k) = (\mu_2 \circ \beta)(k)$, $k \in K_1$. Clearly, the $\equiv$ relation is an equivalence relation. Every equivalence class of this relation specifies a *state* of the net (system) that is best identified as a multiset of places.

Definition 2.5 (State of a net system) A *state of a net* $N := (P, T, F)$ *induced by a marking* $M := (K, \mu)$ of N is a multiset H of places of N, where every place $p \in P$ appears $|\{k \in K \mid \mu(k) = p\}|$ times in H. ⌟

A state of a net system $S := (N, M)$ is a state of N induced by M. Let M_1 and M_2 be two markings of a net and let H_1 and H_2 be two states of the net induced by M_1 and M_2, respectively. It is easy to see that $H_1 = H_2$ if and only if $M_1 \equiv M_2$. Let M_1 and M_2 be equivalent and let (N, M_1) and (N, M_2) be two net systems, $N := (P, T, F)$. It is obvious that $(N, M_1)[t\rangle$ holds if and only if $(N, M_2)[t\rangle$ holds, $t \in T$. Finally, let (N, M_1') and (N, M_2') be net systems obtained after occurrence of t in S_1 and S_2, respectively. Clearly, it holds that $M_1' \equiv M_2'$.

The above observations lead to a notion of a *step* that describes equivalent transition occurrences. A *step* in a net $N := (P, T, F)$ is an ordered triple $\chi := (H_1, t, H_2)$, where H_1 and H_2 are states of N induced by some markings M_1 and M_2 of N, respectively, and $t \in T$, such that $(N, M_1)[t\rangle(N, M_2)$ holds.

A net system induces a set of its reachable states/markings/systems.

Definition 2.6 (Run, Reachable state, Occurrence sequence)
Let $S_0 := (N, M_0)$, $N := (P, T, F)$, be a net system.

- A sequence of steps $\delta := (H_0, t_1, H_1) \ldots (H_{n-1}, t_n, H_n)$, $n \in \mathbb{N}_0$, in N is a *run* in S_0, iff δ is empty or H_0 is a state of S_0.
- A state H of N is *reachable* from S_0, denoted by $H \in [S_0\rangle$, iff H is induced by M_0 or there exists a run $(H_0, t_1, H_1) \ldots (H_{n-1}, t_n, H_n)$, $n \in \mathbb{N}_0$, in S_0 such that $H = H_n$. A marking M of N is *reachable* from S_0 iff there exists a state $H \in [S_0\rangle$ such that H is induced by M. A net system (N, M) is *reachable* from S_0 iff M is reachable from S_0.
- A sequence of transitions $\sigma := t_1 \ldots t_n$, $n \in \mathbb{N}_0$, of N is an *occurrence sequence* in S_0, iff σ is empty or there is a run $(H_0, t_1, H_1) \ldots (H_{n-1}, t_n, H_n)$ in S_0. ⌟

[5] $f \rhd X$ denotes the range restriction of function $f : Y \to Z$ to the subset of its codomain $X \subseteq Z$, i.e., $f \rhd X := \{(y, z) \in f \mid z \in X\}$. $dom(f)$ denotes the domain of function f. Finally, $X \bigtriangleup Y$ denotes the symmetric difference of sets X and Y.

Let $S \coloneqq (N, M)$ be a net system. A step (H, t, H') in N is a step in S iff $H \in [S\rangle$. A net system $S \coloneqq (N, M)$ is *bounded* iff there exists a number $n \in \mathbb{N}_0$ such that for every marking M reachable from S it holds that the amount of tokens at each place of N is at most n, i.e., the set $[S\rangle$ is finite.

The net system in Fig. 1(a) is at a marking that induces state $[p_1\, p_2]$. This net system enables transitions t_1, t_2, and t_3. An occurrence of t_1 leads to the net system in Fig. 1(b). Observe that the system in Fig. 1(c) is reachable from the net system in Fig. 1(b), e.g., via t_3, t_4 or $t_3, t_4, t_6, t_3, t_2, t_5$ occurrence sequence.

2.2 Processes of Net Systems

A common way to trace dependencies between transition occurrences in net systems is by means of runs, cf. Definition 2.6. Runs suit well when it comes to describing *orderings* of transition occurrences. Let $\delta \coloneqq \chi_1 \dots \chi_n$, $n \in \mathbb{N}_0$, be a run in a net system $S \coloneqq (N, M_0)$, $N \coloneqq (P, T, F)$, let H_0 be a state of N induced by M_0, and let $H_1 \dots H_n$ be states of N such that for every position i in the run $\chi_i \coloneqq (H_{i-1}, t_i, H_i)$, $t_i \in T$. Then, two subsequent steps χ_{i-1} and χ_i, $i \in [2\,..\,n]$ represent two subsequent transition occurrences, such that an occurrence of t_{i-1} leads to a marking that induces H_{i-1} and an occurrence of t_i takes place at a marking that induces H_{i-1}. However, runs of net systems cannot be used to capture other behavioral phenomena such as *causality* and *concurrency.*

In this section, we discuss *processes* of net systems [3,4,5]. One can rely on processes to adequately represent causality and concurrency relations on transition occurrences. A process of a net system is a net of a particular kind, called *causal net* (or sometimes occurrence net, see e.g., [5]), together with a mapping from elements of the causal net to elements of the net system. The mapping allows interpreting the causal net as a *concurrent* run of the net system.

Definition 2.7 (Causal net) A net $N \coloneqq (B, E, G)$, $B \subseteq \mathbb{K}$, is a *causal* net[6], iff:
- ∘ for every $b \in B$ it holds that $|\bullet b| \leq 1$ and $|b \bullet| \leq 1$ (N is conflict-free), and
- ∘ G^+ is irreflexive (N is acyclic).[7] ⌟

Elements of E are called *events* and elements of B are called *conditions* of N.

One can utilize events of causal nets to represent transition occurrences. Consequently, conditions in the preset and postset of an event can be interpreted as tokens consumed and produced, respectively, by the transition occurrence that corresponds to the event. The intuition discussed above can be formalized in the notion of a *process* of a net system as follows.

Definition 2.8 (Process of a net system)
A *process* of a net system $S \coloneqq (N, M_0)$, $N \coloneqq (P, T, F)$, is an ordered pair $\pi \coloneqq (N_\pi, \rho)$, where $N_\pi \coloneqq (B, E, G)$ is a causal net and $\rho : B \cup E \to P \cup T$ is s.t.:
- ∘ $\rho(B) \subseteq P$, $\rho(E) \subseteq T$ (ρ preserves the nature of nodes),
- ∘ $M_0 \equiv (Min(N_\pi), \rho|_{Min(N_\pi)})$ (π starts at M_0), and

[6] For technical convenience, we require that every element of B is an element of $\mathbb{K}$.
[7] R^+ denotes the transitive closure of a binary relation R.

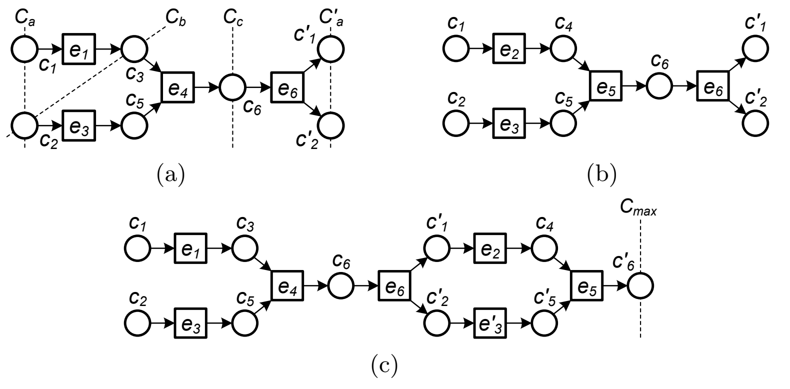

Fig. 2. Processes of the net system in Fig. 1(a)

- for every event $e \in E$ and for every place $p \in P$ it holds that $|\{(p,t) \in F \mid t = \rho(e)\}| = |\rho^{-1}(p) \cap \bullet e|$ and $|\{(t,p) \in F \mid t = \rho(e)\}| = |\rho^{-1}(p) \cap e\bullet|$ (ρ respects the environment of transitions). ⌟

An event $e \in E$ represents an occurrence of transition $\rho(e)$; conditions $\bullet e$ and $e\bullet$ relate to tokens that are consumed and produced by the respective transition occurrence. More precisely, for each condition $c \in \bullet e$ one token should be removed from place $\rho(c)$ and for each condition $c \in e\bullet$ one token should be put at place $\rho(c)$. The set of all processes of a net system collectively defines its *behavior*.

Fig. 2 shows three processes of the net system in Fig. 1(a). When visualizing processes, we use $e_i, e'_i \ldots$ to denote events that refer to transition t_i; similarly, conditions $c_i, c'_i \ldots$ refer to place p_i. Processes capture dependencies between transition occurrences as follows. Two nodes x and y of a causal net $N := (B, E, G)$ are *causal*, iff $(x,y) \in G^+$; otherwise x and y are *concurrent*. For instance, in Fig. 2(a), events e_1 and e_6 are causal, i.e., an occurrence of transition t_1 is a prerequisite for an occurrence of transition t_6, whereas events e_1 and e_3 are concurrent, i.e., t_1 and t_3 can be enabled at the same time and occur in any order.

In addition to transition occurrences, processes encode reachable markings of corresponding net systems by means of cuts. A *cut* of a causal net is a maximal (with respect to set inclusion) set of its pairwise concurrent conditions.

Theorem 2.9 (Cuts and reachable markings, cf. [5, Theorem 3.5]) *Let $\pi := (N_\pi, \rho)$, $N_\pi := (B, E, G)$, be a process of a net system S. If $C \subseteq B$ is a cut of N_π, then $M := (C, \rho|_C)$ is a marking that is reachable from S.* ⌟

Theorem 2.9 allows interpreting a process as a space-efficient data structure that stores markings that are reachable from the corresponding net system. Fig. 2(a) shows four (out of six) cuts of the causal net; a cut is a set of conditions that intersect a dashed line. Cuts C_a and C'_a describe the marking in Fig. 1(a), whereas cuts C_b and C_c refer to the markings in Fig. 1(b) and Fig. 1(c), respectively.

Let $\pi := (N_\pi, \rho)$ be a process of a net system $S := (N, M)$. It is obvious that $Min(N_\pi)$ is a cut [5]. Moreover, by definition, $Min(N_\pi)$ is the cut that describes marking M, i.e., $M \equiv (Min(N_\pi), \rho|_{Min(N_\pi)})$, cf. cut C_a in Fig. 2(a).

3 Process (Set) Systems

A simple structure of processes, i.e., they are static models captured as causal nets, permits simple analysis. Processes allow precise reasoning about *causality* and *concurrency* of transition occurrences [5]. However, this simple analysis comes at a price, as time is traded for space. A net system can often have an infinite number of processes making any type of analysis on the set of all processes an infeasible task, e.g., this is the case of the net systems in Fig. 1.

The discussion above triggers a question: Is it possible to represent the behavior of a net system with a finite number of processes? Clearly, one can give a positive answer to this question only if the space-efficiency of each individual process is "high", i.e., it should be possible to interpret a process in such a way that it captures an infinite portion of the system's behavior. The initial insights into the feasibility of such an interpretation of a process come from the notion of a *reproduction process* [6]. A reproduction process (N_π, ρ) captures a repetitive behavior as its minimal and maximal cuts, i.e., $\mathit{Min}(N_\pi)$ and $\mathit{Max}(N_\pi)$ both refer to the same marking; observe that processes in Fig. 2(a) and Fig. 2(b) are reproduction processes. We learn from these insights. Our intent is to maximize encoding of the repetitive behaviors in a single process.

Given a process of a net system, the behavior encoded in the process can be decoded in terms of the restricted behavior of the corresponding net system. Formally, we implement the above intuition by means of *process systems*.

Definition 3.1 (Process system)
A *process system* is an ordered triple $\mathcal{S}_\pi := (N, M, \pi)$, where M is a marking of a net N and π is a process of a net system (N, M'); M' is a marking of N. ⌟

The semantics of process systems – similarly to the semantics of net systems, cf. Definition 2.4 – consists of the transition enablement and transition occurrence rules. The enablement rule of a net system (N, M) depends on the structure of the net N, i.e., on tokens in presets of transitions of the net. The enablement rule of a process system (N, M, π) relies on the structure of the process π. The exact formulation of the rule is due to Theorem 2.9 and the following result.

Proposition 3.2 (Process restricted transition enablement)
Let $\pi := (N_\pi, \rho)$, $N_\pi := (B, E, G)$, be a process of a net system $S := (N, M_0)$, let $D \subseteq B$ be a set of conditions, and let $e \in E$ be an event. If $\bullet e \subseteq D$, then transition $t := \rho(e)$ is enabled in N at marking $M := (D, \rho|_D)$, i.e., $(N, M)[t\rangle$ holds. ⌟

Proposition 3.2 follows immediately from the fact that ρ preserves the nature of nodes and environment of transitions, cf. Definition 2.8. Consequently, we propose the following semantics for process systems.

Definition 3.3 (Semantics of a process system) Let $\mathcal{S}_\pi := (N, M, \pi)$, $N := (P, T, F)$, $\pi := (N_\pi, \rho)$, $N_\pi := (B, E, G)$, be a process system.

- A transition $t \in T$ is *enabled* in $\mathcal{S}_\pi$, denoted by $\mathcal{S}_\pi[t\rangle$, iff there exists a cut $C \subseteq B$ of N_π and an event $e \in E$ such that $M \equiv (C, \rho|_C)$, $\bullet e \subseteq C$, and $t = \rho(e)$.
- If a transition $t \in T$ is enabled in $\mathcal{S}_\pi$ then t can *occur*, which leads to a process system (N, M', π), where M' is a marking s.t. $(N, M)[t\rangle(N, M')$ holds. ⌟

The semantics of a process system is restricted to those markings that are equivalent with cuts of the process π and to those transition occurrences encoded in the process π. For instance, a process system composed of the net and marking in Fig. 1(a) and the process in Fig. 2(a) enables transitions t_1 and t_3. Indeed, the process contains a cut $C_a \coloneqq \{c_1, c_2\}$ that describes the marking in Fig. 1(a) and it holds that $\bullet e_1 \subseteq C_a \supseteq \bullet e_3$. Observe that the net system in Fig. 1(a) enables transition t_2, in addition to transitions t_1 and t_3.

The semantics of a process system $\mathcal{S}_\pi$ has its natural boundaries on what portion of system's behavior can be described by $\mathcal{S}_\pi$. In order to overcome these boundaries, we introduce *process set systems* that capture behavior of a net system restricted by a collection of its processes, rather than by a single process.

Definition 3.4 (Process set system)
A *process set system* is an ordered triple $\mathcal{S} \coloneqq (N, M, \mathit{\Pi})$, where M is a marking of a net N and $\mathit{\Pi}$ is a set of processes (an untangling) of a net system (N, M'). ⌟

For technical considerations, we expect that for every two distinct processes $\pi_1, \pi_2 \in \mathit{\Pi}$, where $\pi_1 \coloneqq (N_1, \rho_1)$, $N_1 \coloneqq (B_1, E_1, G_1)$, and $\pi_2 \coloneqq (N_2, \rho_2)$, $N_2 \coloneqq (B_2, E_2, G_2)$, it holds that $(B_1 \cup E_1) \cap (B_2 \cup E_2) = \emptyset$. When visualizing processes, we use the same label though assuming distinct elements, e.g., refer to Fig. 2.

The semantics of a process set system $\mathcal{S} \coloneqq (N, M, \mathit{\Pi})$ is "composed" of the semantics of process systems induced by processes in $\mathit{\Pi}$.

Definition 3.5 (Semantics of a process set system)
Let $\mathcal{S} \coloneqq (N, M, \mathit{\Pi})$, $N \coloneqq (P, T, F)$, be a process set system.

- A transition $t \in T$ is *enabled* in $\mathcal{S}$, denoted by $\mathcal{S}[t\rangle$, iff there exists a process $\pi \in \mathit{\Pi}$ such that $\mathcal{S}_\pi[t\rangle$ holds, where $\mathcal{S}_\pi \coloneqq (N, M, \pi)$.
- If a transition $t \in T$ is enabled in $\mathcal{S}$, then t can *occur*, which leads to a process set system $(N, M', \mathit{\Pi}')$, where M' is a marking s.t. $(N, M)[t\rangle(N, M')$ holds and $\mathit{\Pi}' \coloneqq \{\pi \in \mathit{\Pi} \mid (N, M, \pi)[t\rangle\}$. ⌟

By $\mathcal{S}[t\rangle\mathcal{S}'$, we denote the fact that there exists an occurrence of transition $t \in T$ that leads from a process (set) system $\mathcal{S}$ to a process (set) system $\mathcal{S}'$. Let $\mathcal{S}_0 \coloneqq (N, M, \mathit{\Gamma})$, $N \coloneqq (P, T, F)$, be a process (set) system. Similar to net systems, the state H of N induced by M is the state of $\mathcal{S}_0$. A sequence of steps $\delta \coloneqq (H_0, t_1, H_1) \ldots (H_{n-1}, t_n, H_n)$, $n \in \mathbb{N}_0$, is a *run* in $\mathcal{S}_0$, iff δ is empty or there exists a sequence of process (set) systems $\mathcal{S}_1 \ldots \mathcal{S}_n$ such that for every position i in δ it holds that $\mathcal{S}_{i-1}[t_i\rangle\mathcal{S}_i$ and H_{i-1} and H_i are states of $\mathcal{S}_{i-1}$ and $\mathcal{S}_i$, respectively. Accordingly, such notions as a reachable state/marking/process (set) system, and an occurrence sequence in a process (set) system are defined similar to those for net systems, cf. Definition 2.6, but considering runs in process (set) systems.

Let $\mathcal{S}' \coloneqq (N, M, \mathit{\Pi})$ be a process set system reachable from a process set system $\mathcal{S}$ via a run δ. Then, for each $\pi \in \mathit{\Pi}$ it holds that δ is a run in (N, M, π). A process set system $\mathcal{S}$ that is composed of the net and marking in Fig. 1(a) and the processes in Fig. 2(a) and Fig. 2(b) enables transitions t_1, t_2, and t_3. An occurrence of transition t_1 leads to the process set system composed of the net and marking in Fig. 1(b) and the process in Fig. 2(a), which enables transition t_3. Indeed, the process in Fig. 2(b) does not describe an occurrence of transition t_1.

4 Representative Untanglings

This section proposes the notion of *representative* untangling. A representative untangling of a net system is a set of its processes that collectively allow the behavior of this net system and disallow any other. The next section defines the notion of a representative untangling of a net system. Afterwards, Section 4.2 proposes an algorithm for constructing representative untanglings.

4.1 Definition

A net system can be untangled into a set of processes in many possible ways. Every such set contains information on some portion of the net system's behavior. The precise definition of this portion depends on a particular semantics of processes. If one expects to employ the untangled processes for analysis, it is demanding that they capture the exact behavior (recall the discussion in Section 3). Next, we characterize those sets of processes that according to the semantics from Section 3 represent the *exact* behavior of the corresponding net systems.

Definition 4.1 (Representative untangling)
Let $S := (N, M)$ be a net system and let Π be a set of processes of S.

- A process $\pi \in \Pi$ *represents* a step (H, t, H') in S iff it holds that $(N, M', \pi)[t\rangle$, where M' is a marking of N that induces H.
- A process $\pi \in \Pi$ *represents* a run δ in S, either finite or infinite, iff π represents every step in δ.
- The set Π is a *representative untangling* of S iff for every run δ in S there exists a process $\pi \in \Pi$ that represents δ. ⌟

Let Π be a representative untangling of a net system $S := (N, M)$. Then, the net system S and the process set system $\mathcal{S} := (N, M, \Pi)$ are in a strong behavior equivalence relation. In fact, from the point of view of an external observer, S and $\mathcal{S}$ specify the same system. Both S and $\mathcal{S}$ induce occurrences of transitions from the net N. Thus, whenever a transition occurs in either of two systems it occurs in the same environment, i.e., the same preset and postset of places. Because a run δ in S is represented by some process $\pi \in \Pi$, it holds that δ is a run in $\mathcal{S}$. The converse, i.e., the result that a run δ in $\mathcal{S}$ is also a run in S, can be obtained by referring to Proposition 3.2 at each step along the run δ. The above observations lead to a conclusion that S and $\mathcal{S}$ are *occurrence net equivalent* [7], which is the strongest behavioral equivalence notion for models of concurrent systems [8], that states that *unfoldings* [9,10] of both systems are isomorphic.

A representative untangling Π of a net system S induces a process set system that allows all the behavior of the net system and disallows any other. Moreover, the set Π provides an alternative specification of the behavior encoded in S that is particularly appealing for analysis purposes because of its relation between runs in S and processes in Π; every run in S is represented by some process $\pi \in \Pi$ which provides a dedicated object with a limited scope as input for analysis. That is, one can check if there exists a run in S with certain properties by checking if there exists a process in Π that allows a run with these properties. Likewise, it is possible to check if some property holds for all runs in S by validating it against all processes in Π. Finally, as processes adequately represent the ordering, causality, and concurrency relation on transition occurrences, one can explore these relations when designing properties to be checked; note that it is often possible to derive the conflict relation from events in different processes.

4.2 Construction

This section proposes an algorithm that given a bounded net system constructs its representative untangling. The section starts by suggesting a method for representing processes in pseudocode. Afterwards, this representation is employed in an algorithm that given a run in a net system constructs its induced process. The algorithm for constructing representative untanglings relies on this construction as it attempts to discover those runs in the input net system that induce set of processes with desired characteristics. After having presented the untangling algorithm, this section closes with its termination and correctness analysis.

Representation of Processes. One can represent processes in several ways. For instance, one can adopt the approach proposed in [11]. A process can be represented as a set of conditions and events, where a condition is captured as an ordered pair of a place it refers to and the only event in its preset (or $\varnothing$ in the case of a source condition), while an event is captured as an ordered pair of a transition it refers to and a set of conditions in its preset. This representation is designed to describe nets without backward conflicts. Processes are captured as causal nets and, thus, forbid backward and forward conflicts, cf. Definition 2.8. Consequently, we take a different approach and represent conditions (B) and events (E) as tokens and sets of tokens, respectively, i.e., $B \subseteq \mathbb{K}$ and $E \subseteq \mathcal{P}_{\geq 1}(\mathbb{K})$.[8] A binary relation $\rho \subseteq (B \times P) \cup (E \times T)$ is used to specify the mapping of conditions to places (P) and events to transitions (T). As usual, the structure of a process is given by the flow relation $G \subseteq (B \times E) \cup (E \times B)$. The idea of the process representation proposed above allows implementing an intuition of every condition being the holding of a token at a certain place and an event being the occurrence of a transition at a certain marking. Next, we realize this intuition.

From Runs to Processes. A natural way to address construction of a process is by iteratively appending fresh events to a causal net. The input to such a procedure is a sequence of transition occurrences – a run. Algorithm 1 summarizes a procedure that given a run of a net system constructs an induced process. The algorithm follows the intuition of the proof of Theorem 3.6 in [5].

The starting point for the construction is a process composed of conditions that correspond to tokens from the marking of the net system and no events, cf. lines 1–2 in Algorithm 1. The construction proceeds by appending events to the process stepwise, via the `for` loop of lines 4–11. Every appended event corresponds to a transition occurrence that is described by a step in the input run. Events get appended in the order in which respective steps appear in the run. Every fresh event that gets appended to the process and has corresponding transition t is appended together with output conditions that correspond to

[8] $\mathcal{P}_{\geq 1}(X)$ denotes the set of all non-empty subsets of set X, including X itself.

output places of t, which are constructed at line 6. Observe that line 6 ensures that output conditions are fresh with respect to the history of the run. The flow relation gets completed, respecting the environment of t at line 8. Note that at the start of every iteration of the `for` loop, the pair (K, μ) represents a marking that enables an occurrence of transition t_i, i.e., induces H_{i-1}.

Algorithm 1: *Process*(S, δ) – Construct process induced by run

```
   Input: A run $\delta := (H_0, t_1, H_1) \ldots (H_{n-1}, t_n, H_n)$, $n \in \mathbb{N}_0$, in a net system
          $S := (N, M_0)$, $N := (P, T, F)$, $M_0 := (K_0, \mu_0)$, $t_i \in T$, $i \in [1 \,..\, n]$
   Output: A process $\pi$ of $S$ induced by $\delta$
1  $B := K_0$; $E := \varnothing$; $G := \varnothing$; // initialize conditions, events, and flow
2  $\rho := \mu_0$; // initialize mapping of nodes
3  $K := K_0$; $\mu := \mu_0$;
4  for $i := 1$ to $n$ do // iterate over positions in $\delta$
       // prepare
5      $f \subseteq \mu \rhd \bullet t_i$ is a bijection such that $|dom(f)| = |\bullet t_i|$;
6      $X \subseteq \mathbb{K} \setminus B$ is a set of tokens such that $|X| = |t_i \bullet|$;
7      $g : X \to t_i \bullet$ is a bijection between $X$ and $t_i \bullet$;
       // construct
8      $B := B \cup X$; $E := E \cup \{K\}$; $G := G \cup (dom(f) \times \{K\}) \cup (\{K\} \times dom(g))$;
9      $\rho := \rho \cup \{(K, t_i)\} \cup g$;
10     $\mu := \mu \vartriangle (f \cup g)$; $K := dom(\mu)$;
11 end
12 return $\pi := ((B, E, G), \rho)$;
```

The construction in Algorithm 1 is not unique but is always possible. Moreover, a process resulting from the algorithm represents the run it was constructed from.

Proposition 4.2 (Processes and runs)
Let δ be a run in a net system S. Then, Process(S, δ) represents δ. ⌟

Let $\delta := (H_0, t_1, H_1) \ldots (H_{n-1}, t_n, H_n)$, $n \in \mathbb{N}_0$, be a run in a net system S. For every start of the i-th iteration of the `for` loop of lines 4–11, the set $C := Max((B, E, G))$ is a cut of (B, E, G) and $(C, \rho|_C)$ is a marking of N that induces state H_{i-1} cf. [5]. By the end of the i-th iteration, a fresh event that corresponds to transition t_i is appended to a process under construction with its preset completely in C. Hence, *Process*(S, δ) represents every step in δ, cf. Definition 4.1.

Algorithm. Next, an algorithm for the construction of representative untanglings is proposed. The algorithm expects a bounded net system as input and is a state space search algorithm that discovers runs of the input net system that induce its representative untangling. Searching a state space means systematically observing transition occurrences so as to visit the states of the net system. Subsequent transition occurrences make up runs of the net system. If one attempts to employ the discovered runs to induce a representative untangling of a net system, one must ensure that these runs contain sufficient amount of information on the net system's behavior. Construction of a run can terminate naturally, i.e., when a net system reachable via the run does not enable any transition. Additionally, we propose to terminate construction of a run if it encodes a repetitive behavior that can be reconstructed from other steps in the run – the run is *insignificant* with respect to repetitive behaviors. A significant run is defined as follows.

Definition 4.3 (Significant run)
A run $\chi_1 \dots \chi_n$, $n \in \mathbb{N}_0$, in a net system is *repetition significant*, or *significant*, iff:

$$\forall\, i,j \in [1\,..\,n], i < j, \chi_i = \chi_j \;\exists\, k \in (i\,..\,j)\; \forall\, m \in [1\,..\,i) \cup (j\,..\,n] \;:\; \chi_k \neq \chi_m .$$

⌟

A run that is not significant is *insignificant.* Considering the net system in Fig. 1(a), occurrence sequence $t_1, t_3, t_4, t_6, t_1, t_3, t_4, t_6$ is induced by the insignificant run. Note that the run which induces sequence $t_1, t_3, t_4, t_6, t_1, t_4, t_3, t_6$ is significant.

Let α and β be two sequences. Then, $\alpha + \beta$ is the sequence obtained by concatenating α and β, i.e., joining them end-to-end. Let $\delta := \chi_1 \dots \chi_n$, $n \in \mathbb{N}_0$, be a run in a net system S. By $PE(S, \delta)$ we denote the set of all *possible extensions* of δ, i.e., the set of all steps in S such that for every step $\chi \in PE(S, \delta)$ it holds that $\delta + \chi$ is a run in S. Finally, Algorithm 2 exploits the significant property of runs to construct a representative untangling of a bounded net system.

Algorithm 2: *RPS(S)* – Construct representative untangling

```
Input: A bounded net system $S := (N, M_0)$
Output: A representative untangling of $S$
1  $\varDelta := \{\varnothing\}$; // $\varDelta$ is a set of runs in $S$, $R := \{\varnothing\}$, $A := \{\varnothing\}$
2  $\mathcal{R}_n^S := \varnothing$; // initialize result with the empty set
3  while $\varDelta \neq \varnothing$ do
4      $\varDelta := \varDelta \setminus \{\delta\}$, $\delta \in \varDelta$; // $\delta$ is a run in $S$
5      if $PE(S, \delta) = \varnothing$ then $\mathcal{R}_n^S := \mathcal{R}_n^S \cup \{Process(S, \delta)\}$; // collect result
6      else
7          $allExtIns :=$ true;
8          foreach $\chi \in PE(S, \delta)$ do
9              if $\delta + \chi$ is significant then
10                 $\varDelta := \varDelta \cup \{\delta + \chi\}$; // $R := R \cup \{\delta + \chi\}$, $A := A \cup \{(\delta, \delta + \chi)\}$
11                 $allExtIns :=$ false;
12             end
13         end
14         if $allExtIns$ then $\mathcal{R}_n^S := \mathcal{R}_n^S \cup \{Process(S, \delta)\}$; // collect result
15     end
16 end
17 return $\mathcal{R}_n^S$;
```

Termination Analysis. The `while` loop of lines 3–16 iterates as long as there are runs in the set $\varDelta$. Prior to the first iteration of the loop, the set $\varDelta$ is initialized with the empty run at line 1. At every iteration of the `while` loop, one run δ is drawn from $\varDelta$ at line 4. If δ has no possible extensions (see the check at line 5), then no runs are added to $\varDelta$ in the same iteration of the loop. Otherwise, in the `foreach` loop of lines 8–13, every possible extension of δ that leads to a significant run (refer to the check at line 9) triggers the insertion of this extended run into the set $\mathit{\Delta}$ at line 10. Observe that no run is added to the set $\mathit{\Delta}$ within a single iteration of the `while` loop of lines 3–16 if every possible extension of δ leads to an insignificant run in S (check at line 9 does not evaluate to true).

Given a net system, Algorithm 2 systematically explores its runs. Let R be a set that collects runs that are added to the set $\mathit{\Delta}$ at lines 1 and 10 along a course of execution of Algorithm 2 – a set of *visited runs*; the set R can be constructed as specified in the comments of respective lines of the algorithm (if executed). It is easy to see that the `while` loop of lines 3–16 maintains the following invariant.

Invariant 4.4 (Fresh runs – the `while` loop of lines 3–16)
Let R be the set of visited runs up to the current execution point in Algorithm 2. Prior to every execution of line 10, it holds that the run $\delta + \chi$ is not in R. ⌟

Proof. Assume that before some execution of line 10, it holds that $\delta + \chi$ is in R. Then, in the course of the same execution of Algorithm 2 there exist two iterations of the `while` loop of lines 3–16 that selected the same run as δ at line 4. This means that there exists an iteration of the `while` loop in the course of this execution such that before line 10 it holds that δ is in R. By applying the above reasoning iteratively, one can conclude that there exist two iterations of the `while` loop that selected the empty run as δ at line 4. However, one can clearly see from the structure of the algorithm that the empty run is always selected at line 4 in the first iteration of the `while` loop and never afterwards. ∎

The logic of Algorithm 2 makes it apparent that its execution will never terminate if the input net system has an infinite significant run. Next, we show that a bounded net system has no infinite significant runs.

Lemma 4.5 (Infinite runs)
An infinite run in a bounded net system is insignificant. ⌟

Proof. Let $\delta := \chi_1, \chi_2 \ldots$ be an infinite run in a bounded net system $S := (N, M)$, $N := (P, T, F)$. Assume that σ is significant. Let $\mathit{\Sigma}$ be the set of all steps (H, t, H'), $t \in T$, in N, where H is a state of N reachable from S. Because S is bounded and T is finite, it holds that $\mathit{\Sigma}$ is finite. There exists a step $\chi \in \mathit{\Sigma}$ such that χ occurs infinitely often in δ; otherwise δ is finite. Then, there also exists an infinite sequence γ of positions in δ (in ascending order) such that for every element n in γ it holds that $\chi = \chi_n$. Consequently, for every pair of subsequent elements i and j in γ there exists $k \in (i \mathinner{..} j)$ such that for all $m \in [1 \mathinner{..} i) \cup (j \mathinner{..} \infty]$ it holds that $\chi_k \neq \chi_m$; otherwise δ is insignificant. Then, χ occurs at most $|\mathit{\Sigma}|$ times in δ. ∎

In other words, all significant runs in a bounded net system are finite. Another threat that can cause Algorithm 2 run forever stems from the necessity to explore an infinite number of finite significant runs. Next, we show that there is a finite number of significant runs in a bounded net system.

Lemma 4.6 (Significant runs)
The set of all significant runs in a bounded net system is finite. ⌟

Proof. Let $S := (N, M)$, $N := (P, T, F)$, be a bounded net system and let $\mathit{\Sigma}$ be the set of all steps (H, t, H'), $t \in T$, in S. As S is bounded and T is finite, it holds that $\mathit{\Sigma}$ is finite. As per Lemma 4.5, every significant run in S is finite. Then, there is $n \in \mathbb{N}_0$ such that every significant run in S is a sequence of at most n steps. Thus, the number of significant runs in S is bounded above by the size of a dictionary of all words of length at most n such that every word is composed of symbols drawn from the fixed alphabet Σ; this dictionary is finite. ▪

Finally, Theorem 4.7 collects all the above results to show that Algorithm 2 indeed terminates for every input bounded net system.

Theorem 4.7 (Termination) *Algorithm 2 always terminates.* ⌟

Proof. Algorithm 2 terminates if the `while` loop of lines 3–16 terminates. We associate a measure with every iteration of the `while` loop. Let R be a set of visited runs that gets constructed at lines 1 and 10 in a course of execution of Algorithm 2. Let the measure of the i-th iteration of the `while` loop be an ordered pair (Δ_i, R_i), where Δ_i and R_i are the values of Δ and R, respectively, at the time of the check at line 3 (prior to executing the body of the loop). For example, Δ_1 and R_1 are both equal to $\{\varnothing\}$. One can classify every scenario of a single iteration of the `while` loop into two cases, according to which relation between the current and the next iteration measure this scenario leads.

(i) $|\Delta_{i+1}| - |\Delta_i| = n$, $n \in N_0$, and $R_i \subset R_{i+1}$. This case corresponds to the scenario when a run δ chosen at line 4 has at least one significant extension and, hence, line 10 is executed one or more times in the i-th iteration of the `while` loop. Observe that both $|\Delta_i| \leq |\Delta_{i+1}|$ and $R_i \subset R_{i+1}$ hold because of Invariant 4.4.
(ii) $\Delta_{i+1} \subset \Delta_i$ and $R_i = R_{i+1}$. This case corresponds to the scenario when a run δ chosen at line 4 has no extensions or all its extensions result in insignificant runs and, thus, line 10 is not executed in the i-th iteration of the `while` loop.

The key observation here is that execution of the `while` loop can take scenario that falls under case (i) only a finite number of times. It is easy to see that the algorithm preserves the invariant of R being composed of significant runs and the set of all significant runs is finite, refer to Lemma 4.6. Every time the `while` loop is executed according to scenario (i), the size of the set Δ stays unchanged or is increased by some number n. Therefore, there exists an iteration of the loop from which on execution of every subsequent iteration will always follow scenario (ii) and at that moment in time the set Δ is finite. Thus, the set Δ will eventually become empty and the condition at line 3 will evaluate to false. ▪

Correctness Analysis. Let δ be a run in a net system S. The set of processes Π of S is representative if there exists $\pi \in \Pi$ that represents every step in δ, cf. Definition 4.1. According to Proposition 4.2, a process represents all the steps in a run that it is constructed from (as per Algorithm 1). Next, we show that for every run in a bounded net system S it holds that it is composed of steps that also participate in some run that is used to induce a process in the set $\mathcal{R}_n^S$.

Algorithm 2 explores runs in the input net system. Every fresh run that gets explored is obtained from the one priorly observed, refer to lines 4 and 10 of the algorithm. This fact gives rise to the following relation on runs of the net system.

Definition 4.8 (Graph of runs)
Let R be the set of visited runs constructed by Algorithm 2 for an input bounded

net system S. A *graph of runs* of S is an ordered pair $\mathcal{G} := (R, A)$, where A is the set of ordered pairs $(\delta, \delta + \chi)$ constructed for every pair of values for δ and χ observed at line 10 during execution of Algorithm 2 for input S. ⌟

Note that the precise construction of the set A is specified in the comments to lines 1 and 10 of Algorithm 2. Next, we point out two interesting invariants of the `while` loop of lines 3–16 with respect to the graph of runs.

Invariant 4.9 (Tree of runs – the `while` loop of lines 3–16)
Let $\varDelta_i$, $\mathcal{R}^S_{n,i}$, and $\mathcal{G}_i := (R_i, A_i)$, be the values for $\varDelta$, the set of runs used to induce the set of processes $\mathcal{R}^S_n$, and the graph of runs, respectively, at the start of the i-th iteration of the `while` loop of lines 3–16 in Algorithm 2. Then, the following statements hold at the start of the i-th iteration:
(i) $\mathcal{G}_i$ is a tree, (ii) $\varDelta_i \cup \mathcal{R}^S_{n,i}$ are all the leaves of the tree $\mathcal{G}_i$ rooted at $\varnothing \in R_i$. ⌟

Proof. We show that (i) and (ii) hold prior to the first iteration of the loop, and if (i) and (ii) hold before an iteration, then they hold before the next iteration.

Initialization: (i) $\mathcal{G}_1 := (\{\varnothing\}, \varnothing)$. (ii) $\varDelta_1 \cup \mathcal{R}^S_{n,1} = \{\varnothing\}$.

Maintenance: (i) Because of Invariant 4.4, it holds for every two subsequent iterations of the `while` loop that $|R_{i+1}| - |R_i| = |A_{i+1}| - |A_i|$. Therefore, it holds that $|A_{i+1}| = |R_{i+1}| - 1$. Moreover, it holds that $\mathcal{G}_{i+1}$ is connected. (ii) A run δ selected from $\varDelta_i$ at line 4 is a leaf node of $\mathcal{G}_i$ rooted at $\varnothing$ (empty run). Observe that δ is removed from $\varDelta_{i+1}$ at line 4. If δ has extensions that lead to significant runs then, because of Invariant 4.4, a child run is added to δ in $\mathcal{G}_{i+1}$ at line 10 (and also to $\varDelta_{i+1}$); otherwise no fresh run is added to R_{i+1} and, thus, δ is the leaf node of $\mathcal{G}_{i+1}$, and it is added to $\mathcal{R}^S_{n,i+1}$ (line 14). ∎

Moreover, it is immediate to see that at every moment in the course of execution of Algorithm 2, the set $\mathcal{R}^S_n$ is composed of processes induced by maximal significant runs, where a significant run is *maximal* if every its extension leads to an insignificant run. Indeed, a run induces a process in $\mathcal{R}^S_n$ either if it has no extensions (line 5) or all its extensions are insignificant (*allExtIns* flag is set to true at line 14). In the sequel, we propose several results on run significance that will be later orchestrated to show that Algorithm 2 is correct. We proceed with the claim that each extension of an insignificant run leads to an insignificant run.

Proposition 4.10 (Insignificance invariant) *Let δ be an insignificant run in a net system S. A run $\delta + \chi$, $\chi \in PE(S, \delta)$, is insignificant.* ⌟

The proof of Proposition 4.10 follows immediately from Definition 4.3.

Proposition 4.11 (Infinite and finite runs)
Let δ be an infinite run in a bounded net system S. There is a finite run δ' in S s.t. for every step χ in δ there is a step χ' in δ' for which it holds that $\chi = \chi'$. ⌟

Proof. Let $\delta := \chi_1, \chi_2 \dots$ be an infinite run in a bounded net system $S := (N, M)$, $N := (P, T, F)$. Let $\varSigma$ be the set of steps in δ. Because S is bounded and T is finite, it holds that $\varSigma$ is finite. Let $\varOmega$ be the set that for every step $\chi \in \varSigma$ contains the smallest position i in δ at which χ occurs, i.e., $\chi = \chi_i$. Then, the subsequence δ' of δ that removes all the elements after position $\max \varOmega$ is finite. Moreover, for every step χ in δ there is a step χ' in δ' for which $\chi = \chi'$. ∎

An important observation is that for every finite insignificant run δ there exists a significant run that "uses" all the steps from δ.

Lemma 4.12 (Significant and insignificant runs) *Let δ be a finite insignificant run in a net system S. There exists a significant run δ' in S such that for every step χ in δ there exists a step χ' in δ' for which it holds that $\chi = \chi'$.* ⌟

Proof. By infinite descent on subsequences of a run. Let $\delta \coloneqq \chi_1 \dots \chi_n$, $n \in \mathbb{N}_0$, be a finite insignificant run in S. For δ it holds that: (i) there exist two positions i and j in δ such that $i < j$ and $\chi_i = \chi_j$, and (ii) for every $k \in (i \,..\, j)$ there exists $m \in [1 \,..\, i) \cup (j \,..\, n]$ such that $\chi_k = \chi_m$. Let δ' be a subsequence of δ obtained via a reduction operation that removes all elements after position i up to and including element at position j, i.e., $\delta' \coloneqq (\chi_1 \dots \chi_i) + (\chi_{j+1} \dots \chi_n)$. Clearly δ' is a run in S. Moreover, because of (ii), for every step χ in δ there exists a step χ' in δ' for which it holds that $\chi = \chi'$. Observe that the length of δ' is strictly smaller than the length of δ. Assume that δ' is always insignificant. Then, one can construct an infinite sequence of insignificant runs in S that starts with δ and every other run in the sequence is obtained from the previous run via the reduction operation proposed above, which is impossible. ∎

Observe that the proof of Lemma 4.12 defines a construction to obtain a significant run with all the steps of a given insignificant run. The following corollary is an immediate consequence of Lemma 4.11 and Lemma 4.12.

Corollary 4.13 (Runs and significant runs) Let δ be a run in a bounded net system S, either finite or infinite. There is a significant run δ' in S such that for every step χ in δ there exists a step χ' in δ' for which it holds that $\chi = \chi'$ ⌟

Finally, Theorem 4.14 collects all the above results to demonstrate that Algorithm 2 is correct in the sense that given a bounded net system it, certainly, computes its representative untangling.

Theorem 4.14 (Correctness)
Let S be a bounded net system. $\mathcal{R}_n^S$ is a representative untangling of S. ⌟

Proof. Upon termination of Algorithm 2, it holds that $\Delta = \varnothing$ and, hence, the set $\mathcal{R}_n^S$ contains processes induced by all the runs that correspond to the leaf nodes in the tree of runs $\mathcal{G} \coloneqq (R, A)$ of S, refer to Invariant 4.9. Therefore, for every run δ in S that is composed of steps that also participate in some leaf run in $\mathcal{G}$ it holds that there exists a process in $\mathcal{R}_n^S$ that represents δ, refer to Proposition 4.2. Next, we show that the above statement holds for every significant run in S. Observe that R is composed of significant runs, see the check at line 9 of Algorithm 2, and every internal run in $\mathcal{G}$ is a subrun (a subsequence) of some leaf run in $\mathcal{G}$. Moreover, as leaf runs in $\mathcal{G}$ are maximal and because of Proposition 4.10, it holds that R is the set of all significant runs in S. Finally, according to Corollary 4.13, it holds that every insignificant run is composed from steps of some significant run that, in turn, is composed from steps of some leaf run in $\mathcal{G}$. ∎

Example. Algorithm 2 provides a theoretic foundation for constructing representative untanglings. However, in practice, it may build a high number of processes. For instance, the tree of runs of the net system in Fig. 1(a) consists of 508 significant runs, 152 of which are maximal. If one prunes the tree by iteratively removing those leaves that are composed of steps observed in some internal run in the tree, one can use the maximal runs in the pruned tree to construct eleven unique processes out of which six contain the other five as subgraphs. These six processes consitute a representative untangling of the system. Observe that the set with exactly one process in Fig. 2(c) is another representative untangling of the net system in Fig. 1(a). It encodes all 152 significant runs of the system and, thus, all its runs. In future work, we plan to develop fast untangling algorithms that are interesting from the practical point of view, as well as look for the canonical (or minimal) representative untanglings.

5 Behavioral Properties

This section demonstrates how representative untanglings can be utilized for efficient verification of behavioral properties. We keep extensive studies for future work and, in the following, rather give some illustrative examples that should provide the reader with a grip on untangling-based analysis of concurrent systems.

The *executability* problem deals with deciding whether a system can execute any transition out of a given set of transitions. It is a fundamental problem in the concurrency theory as many others can be reduced to this one, e.g., a solution to the executability problem can help deciding *reachability* and *safety* [12,13].

Lemma 5.1 (Executability) *Let Π be a representative untangling of a bounded net system $S := (N, M)$, $N := (P, T, F)$. A transition $t \in U \subseteq T$ can be executed in S, i.e., is part of some occurrence sequence in S, iff there is a process $\pi := (N_\pi, \rho)$, $N_\pi := (B, E, G)$, in Π that contains an event $e \in E$ for which $\rho(e) = t$.* ⌟

The proof of Lemma 5.1 follows immediately from the definition of a representative untangling, which can be always constructed for a bounded net system.

Another important problem in the concurrency theory is *deadlock freedom*. It deals with answering the question whether there exists a deadlock marking reachable from a net system. A marking M of a net $N := (P, T, F)$ is a *deadlock* marking of N if and only if it does not enable any transition in N, i.e., the set $\{t \in T \mid (N, M)[t\rangle\}$ is empty. A net system $S := (N, M)$ is *deadlock free* if and only if none of the markings reachable from S is a deadlock marking of N.

Lemma 5.2 (Deadlock freedom) *Let Π be a representative untangling of a bounded net system $S := (N, M)$. S is deadlock free iff for each process $\pi := (N_\pi, \rho)$ in Π it holds that $(C, \rho|_C)$, with $C := Max(N_\pi)$, is not a deadlock marking of N.* ⌟

Proof. We prove each direction of the statement separately.

($\Rightarrow$) If S is deadlock free then every marking that is reachable from S is not a deadlock marking of N. Because every maximal cut of every process in Π corresponds to a marking reachable from S, refer to Theorem 2.9, it holds that all maximal cuts of processes in Π do not correspond to deadlock markings.

($\Leftarrow$) Assume that for every process $\pi := (N_\pi, \rho)$ in Π it holds that $(C, \rho|_C)$, where $C := Max(N_\pi)$, is not a deadlock marking of N, but S is not deadlock free. Then, there exists a deadlock marking M' reachable from S via some run δ in S. There also exists a process $\pi := (N_\pi, \rho)$ in Π that represents δ.

Because π represents every step in δ, it holds that there exists a cut C of N_π that corresponds to marking M'. If C is not the maximal cut of N_π, then M' is not a deadlock marking of N (M' enables transition $\rho(e)$, where $\bullet e \subseteq C$). Then, C is the maximal cut of N_π, which leads to a contradiction. ■

Executability and deadlock freedom of a concurrent system can be checked efficiently on its representative untangling.

Proposition 5.3 *Given a representative untangling of a bounded net system S, the following problems can be solved in linear time:*

- *To decide if a transition (from a set of transitions) can be executed in S.*
- *To decide if S is deadlock free.* ⌟

The proof of Proposition 5.3 is a direct consequence of Lemma 5.1 and Lemma 5.2. Indeed, in the worst case, one can verify executability by visiting every event of a representative untangling once, whereas deadlock freedom can be decided by checking maximal cuts of untangled processes. Observe that the results in Proposition 5.3 are due to the representative property that we enforce on untanglings; executability is due to the fact that every step needs to be represented and deadlock freedom is partly owed to the fact that every run must be represented. We believe that, in a similar way, many other behavioral properties of concurrent systems can exploit the representative property of untanglings leading to efficient (in practice) implementations of verification algorithms.

Considering the representative untangling in Fig. 2(c), one can conclude that the net system in Fig. 1(a) is deadlock free and confirm executability of every transition; the maximal cut $C_{max} \coloneqq \{c'_6\}$ in Fig. 2(c) describes the marking in Fig. 1(c) that is not a deadlock marking as it enables transition t_6.

6 Related Work

A representative untangling of a system is a novel mathematical formalism for the description and analysis of behavior encoded in the system. In the following, we discuss several existing formalisms that address the same problematics.

State space techniques are popular when it comes to the automatic analysis and verification of concurrent systems. Rather than performing analysis directly on a given concurrent system, these methods explore its induced representation called *transition system*. In a nutshell, a transition system induced by a concurrent system S is a graph with states reachable from S as its nodes and an edge from state H to state H' whenever there is a step from H to H' in S. Unfortunately, state space techniques suffer from the state space explosion problem. In the worst case, all nodes of the induced transition system must be explored to accomplish the envisioned analysis task, and there can be exponentially many nodes based on the number of concurrent components of the system under analysis.

An *unfolding* of a concurrent system is a mathematical structure (often infinite) that explicitly represents concurrency and causal relations between system operations, as well as the points of choice between qualitatively different behaviors. Unfoldings are special partially ordered graphs that describe reachable states of a system as combinations of nodes rather than dedicating every node to a single state. In [9,10], McMillan observed that one can represent all the information contained in an unfolding in its truncated finite initial part, called *complete prefix unfolding*. If adequate construction algorithms are employed, size of a complete prefix unfolding of a concurrent system is never larger – and in practice is by far smaller – than the size of the transition system induced by the same system. A classical condition for prefix truncation is *completeness*, where a prefix induced by a concurrent system is complete if it encodes all the steps (reachable states and possible moves) in the system. Completeness of prefixes helps to "unveil" some of the behavioral properties of concurrent systems by allowing their validation in time that is linear in the size of the prefix, e.g., executability [12], while preserving other properties sufficiently "concealed", e.g., the problem of deciding deadlock freedom [14] using complete prefix unfoldings is NP-complete [10]. In order to unveil these hidden behavioral properties in finite parts of unfoldings, one must rely on special unfolding truncation criteria. In [12], truncation criteria that address executability, repeated executability, livelock, and properties expressible in linear temporal logic, are systematized. The strong coupling of behavioral properties of a system with different constellations of finite prefixes of its unfolding is mainly due to implicit dependencies between transition occurrences encoded in finite prefixes which stem from unfolding truncations.

Merged processes, proposed in [15], are compressed representations of complete prefix unfoldings with most of the advantages and disadvantages of unfoldings that were discussed above. The compression is achieved by addressing such sources of state space explosion as sequences of choices and non-safeness. Many results initially proposed for unfoldings can be transferred to merged processes.

Similar to unfoldings, *untanglings* are partially ordered graphs that represent concurrency and causal relations between events of individual transition occurrences. Thus, fundamentally, untanglings address state space explosion to a similar extent as unfoldings. *Representative untanglings* describe all the steps of a system and, additionally to unfoldings, provide clear scopes for analysis of systems that target individual computations (each run is represented by some untangled process). Some implications of such characterization of behavior are demonstrated in Section 5. We believe that further studies of representative untanglings should confirm their applicability as general purpose index structures for behavioral analysis of highly concurrent and repetitive systems.

7 Conclusion and Future Work

This paper develops theoretical foundations that lead to a novel characterization of behavior encoded in a concurrent system. A representative untangling of a concurrent system is our proposal for a new compromise between size of a model that describes behavior and time required for its analysis. Behavioral properties of concurrent systems often relate to existence of an instance, or a collection of instances, of the system with certain characteristics. We rely on the partial-order semantics of concurrent systems in order to minimize size of untanglings, and enforce a strong correspondence between (parts of) representative untanglings and instances of behavior that they specify, in order to facilitate analysis.

This work is the first step in the journey towards effective and efficient untanglings of concurrent systems. Preliminary experiments show that untanglings can be computed fast on reduced systems; reductions can then be efficiently reverted, similar to the approach proposed in [16], to obtain untanglings of original systems. Observe that a representative untangling of a net system can be as small as its complete prefix unfolding. It is evident that size of untanglings can often be noticeably decreased by representing them as branching processes [17] which can be constructed by merging isomorphic histories of untangled processes. Inspired by some of the ideas in [15], we foresee that the obtained branching processes can be further packed into novel acyclic models that will inherit analysis effectiveness of untanglings. This could be achieved by merging isomorphic futures of the original untangled processes. In future work, we plan to study all the above stated ideas in depth.

Acknowledgments. This research is supported by the ARC Linkage Project LP110100252 “Facilitating Business Process Standardisation and Reuse”, and NICTA Queensland Lab, Australia.

References

1. Sassone, V., Nielsen, M., Winskel, G.: Models for concurrency: Towards a classification. TCS **170**(1–2) (1996) 297–348
2. Lee, E.A., Sangiovanni-Vincentelli, A.L.: A framework for comparing models of computation. TCAD **17**(12) (1998) 1217–1229
3. Petri, C.A.: Non-Sequential Processes. GMD ISF. Gesellschaft für Mathematik und Datenverarbeitung (1977)
4. Nielsen, M., Plotkin, G.D., Winskel, G.: Petri nets, event structures and domains, Part I. TCS **13** (1981) 85–108
5. Goltz, U., Reisig, W.: The non-sequential behavior of Petri nets. IANDC **57**(2/3) (1983) 125–147
6. Best, E., Desel, J.: Partial order behaviour and structure of Petri nets. FAC **2**(2) (1990) 123–138
7. van Glabbeek, R.J., Vaandrager, F.W.: Petri net models for algebraic theories of concurrency. In: PARLE. Volume 259 of LNCS., Springer (1987) 224–242
8. Tarasyuk, I.V.: Equivalence Notions for Models of Concurrent and Distributed Systems. PhD thesis, A.P. Ershov Institute of Informatics Systems, Russia (1997)
9. McMillan, K.L.: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In: CAV. Volume 663 of LNCS., Springer (1992)
10. McMillan, K.L.: A technique of state space search based on unfolding. FMSD **6**(1) (1995) 45–65
11. Esparza, J., Römer, S., Vogler, W.: An improvement of McMillan’s unfolding algorithm. FMSD **20**(3) (2002) 285–310
12. Esparza, J., Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. EATCS Monographs in Theoretical Computer Science. Springer (2008)
13. Godefroid, P., Wolper, P.: Using partial orders for the efficient verification of deadlock freedom and safety properties. FMSD **2**(2) (1993) 149–164
14. Melzer, S., Römer, S.: Deadlock checking using net unfoldings. In: CAV. Volume 1254 of LNCS., Springer (1997) 352–363
15. Khomenko, V., Kondratyev, A., Koutny, M., Vogler, W.: Merged processes: A new condensed representation of Petri net behaviour. ACTA **43**(5) (2006) 307–330
16. Khomenko, V.: Behaviour-preserving transition insertions in unfolding prefixes. In: ATPN. Volume 4546 of LNCS., Springer (2007) 204–222
17. Engelfriet, J.: Branching processes of Petri nets. ACTA **28**(6) (1991) 575–591