Acta Informatica manuscript No.
(will be inserted by the editor)

Merged Processes — a New Condensed Representation of Petri Net Behaviour

Victor Khomenko[1⋆], Alex Kondratyev[2], Maciej Koutny[1⋆⋆], Walter Vogler[3⋆⋆⋆]

[1] School of Computing Science, University of Newcastle, NE1 7RU, U.K.
e-mail: {Victor.Khomenko, Maciej.Koutny}@ncl.ac.uk
[2] Cadence Berkeley Labs, Berkeley, CA 94704, USA
e-mail: kalex@cadence.com
[3] Institut für Informatik, Universität Augsburg, D-86135 Germany
e-mail: Walter.Vogler@informatik.uni-augsburg.de

The date of receipt and acceptance will be inserted by the editor

Abstract. Model checking based on Petri net unfoldings is an approach widely applied to cope with the state space explosion problem. In this paper, we propose a new condensed representation of a Petri net's behaviour called *merged processes*, which copes well not only with concurrency, but also with other sources of state space explosion, viz. sequences of choices and non-safeness. Moreover, this representation is sufficiently similar to the traditional unfoldings, so that a large body of results developed for the latter can be re-used. Experimental results indicate that the proposed representation of a Petri net's behaviour alleviates the state space explosion problem to a significant degree and is suitable for model checking.
Keywords: Merged processes, Petri net unravelling, Petri net unfolding, state space explosion, model checking, formal verification.

1 Introduction

A reactive system is commonly described by a set of concurrent processes that interact with each other [8,10]. Processes typically have

⋆ V. Khomenko is a Royal Academy of Engineering/EPSRC Research Fellow supported by the RAEng/EPSRC grant EP/C53400X/1 (DAVAC).
⋆⋆ M. Koutny is supported by the EC IST grant 511599 (RODIN).
⋆⋆⋆ W. Vogler is supported by the DFG-project STG-Dekomposition VO 615/7-1.

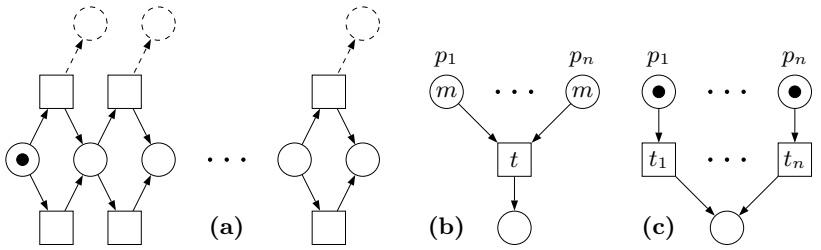


Fig. 1 Examples of Petri nets.

descriptions which are short and manageable, and the complexity of the behaviour of the system as a whole comes from highly complicated interactions between them. One way of coping with this complexity problem is to use formal methods and, especially, computer aided verification tools implementing model checking [1] — a technique in which the verification of a system is carried out using a finite representation of its state space.

The main drawback of model checking is that it suffers from the *state space explosion* problem [20]. That is, even a relatively small system specification can (and often does) yield a very large state space. To cope with this, several techniques have been developed, which usually aim either at a compact representation of the full state space of the system, or at the generation of a reduced state space (that is still sufficient for a given verification task). Among them, a prominent technique is McMillan's (finite prefixes of) Petri net unfoldings (see, e.g., [7,11,15]). They rely on the partial order view of concurrent computation, and represent system states implicitly, using an acyclic *unfolding prefix*.

There are several common sources of state space explosion. One of them is concurrency, and the unfolding techniques were primarily designed for efficient verification of highly concurrent systems. Indeed, complete prefixes are often exponentially smaller than the corresponding reachability graphs, because they represent concurrency directly rather than by multidimensional 'diamonds' as it is done in reachability graphs. For example, if the original Petri net consists of 100 transitions which can fire once in parallel, the reachability graph will be a 100-dimensional hypercube with 2^{100} vertices, whereas the complete prefix will be isomorphic to the net itself. However, unfoldings do not cope well with some other important sources of state space explosion, in particular with sequences of choices and non-safeness. Below we consider examples illustrating this problem.

First, consider Figure 1(a) with the dashed part not taken into account. The cut-off condition proposed in [7] copes well with this

Petri net (since the marking reached after either choice on each stage is the same — in fact, the Petri net has very few reachable markings), and the resulting prefix is linear in the size of the original Petri net. However, if the dashed part of the figure is taken into account, the smallest complete prefix is exponential in the size of the Petri net, since no event can be declared a cut-off (intuitively, each reachable marking of the Petri net 'remembers' its past). Thus Petri nets performing a sequence of choices leading to different markings may yield exponential prefixes.

Another problem arises when one tries to unfold non-safe Petri nets. Consider the Petri net in Figure 1(b). Its smallest complete unfolding prefix contains m^n instances of t, since the standard unfolding *distinguishes between different tokens on the same place.* One way to cope with non-safe nets is to convert them into safe ones and unfold the latter, as was proposed in [7]. However, such an approach destroys concurrency among executed transitions and can lead to very large prefixes; e.g., when applied to the Petri net in Figure 1(c) this approach would yield a prefix exponential in the size of the original Petri net, while the traditional unfolding technique would yield a prefix which is linear in its size [7].

The problems with unfolding prefixes described above should be viewed in the light of the fact that all the examples had a very simple structure — viz. they are all acyclic, and thus many model checking techniques, in particular those based on the *marking equation* [11, 17, 19], could be applied *directly to the original Petri nets.* And so it may happen that a prefix exponential in the size of the Petri net is built for a relatively simple problem!

In this paper, we propose a new condensed representation of a Petri net's behaviour called *merged processes,* which remedies the problems outlined above. It copes well not only with concurrency, but also with other sources of state space explosion we mentioned, viz. sequence of choices and non-safeness. Moreover, this representation is sufficiently similar to the traditional unfoldings, so that a large body of results developed for unfoldings can be re-used.

The main idea behind the new representation is to fuse some nodes in the complete prefix of the Petri net being verified, and use the resulting net as the basis for verification. For example, the unfolding of the net shown in Figure 1(a) (even with the dashed part taken into account) will collapse back to the original net after the fusion. In fact, this will happen in all the examples considered above.

The main body of the paper is devoted to formally defining the transformation based on the fusion of nodes in net unfoldings, and

solving some of the arising problems; in particular, fusing of nodes can create cycles and the marking equation alone is not sufficient for verification of such nets. The experimental results indicate that the new representation of a Petri net's behaviour alleviates the state space explosion problem to a significant degree and is suitable for model checking.

This paper is the full version of the conference paper [12].

2 Basic notions

A multiset over a set X is a function $\mathsf{m} : X \to \mathbb{N} \stackrel{\text{df}}{=} \{0, 1, 2, \ldots\}$, and a subset of X may be viewed through its characteristic function as a multiset over X. We denote $x \in \mathsf{m}$ if $\mathsf{m}(x) \geq 1$, and m is finite if there are finitely many $x \in \mathsf{m}$. A finite multiset may be represented by explicitly listing its elements between the $\{\!|\ldots|\!\}$ brackets.

The sum, intersection and difference of two multisets over X, m and m', are the multisets given by $(\mathsf{m} + \mathsf{m}')(x) \stackrel{\text{df}}{=} \mathsf{m}(x) + \mathsf{m}'(x)$, $(\mathsf{m} \cap \mathsf{m}')(x) \stackrel{\text{df}}{=} \min\{\mathsf{m}(x), \mathsf{m}'(x)\}$ and $(\mathsf{m} - \mathsf{m}')(x) \stackrel{\text{df}}{=} \mathsf{m}(x) - \mathsf{m}'(x)$, for all $x \in X$, respectively. (The difference is defined only if $\mathsf{m}(x) \geq \mathsf{m}'(x)$ for all $x \in X$.) Moreover, if $h : X \to Y$ then, for every multiset $\mathsf{m} = \{\!|x_1, \ldots, x_n|\!\}$ over X, $h\langle\mathsf{m}\rangle$ is a multiset over Y given by $h\langle\mathsf{m}\rangle \stackrel{\text{df}}{=} \{\!|h(x_1)|\!\} + \cdots + \{\!|h(x_n)|\!\}$. For example, if $h(x) = x^2 + 1$ and $\mathsf{m} = \{\!|-1, 0, 0, 1|\!\}$ then $h\langle\mathsf{m}\rangle = \{\!|1, 1, 2, 2|\!\}$.

In the rest of this section, we introduce the basic notions concerning Petri nets and their unfoldings (see also [5,7,11,13,15,17–19]).

Petri nets

A *net* is a triple $N \stackrel{\text{df}}{=} (P, T, F)$ such that P and T are disjoint sets of respectively *places* and *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow relation*, which we also regard as function $(P \times T) \cup (T \times P) \to \{0, 1\}$. The net is *finite* is both P and T are finite sets.

A *marking* of N is a multiset M of places, i.e., $M : P \to \mathbb{N}$. We adopt the standard rules about drawing nets, viz. places are represented as circles, transitions as boxes, the flow relation by arcs, and the marking is shown by placing tokens within circles. As usual, ${}^\bullet z \stackrel{\text{df}}{=} \{y \mid (y, z) \in F\}$ and $z^\bullet \stackrel{\text{df}}{=} \{y \mid (z, y) \in F\}$ denote the *pre-* and *postset* of $z \in P \cup T$. In this paper, the presets of transitions are restricted to be non-empty, i.e., ${}^\bullet t \neq \varnothing$ for every $t \in T$. For a finite net N, we define the *size* of N as $|N| \stackrel{\text{df}}{=} |P| + |T| + |F|$.

A *net system* (or *Petri net*) is a pair $\Sigma \stackrel{\text{df}}{=} (N, M_\Sigma)$ comprising a finite net $N = (P, T, F)$ and an (initial) marking M_Σ. A transition

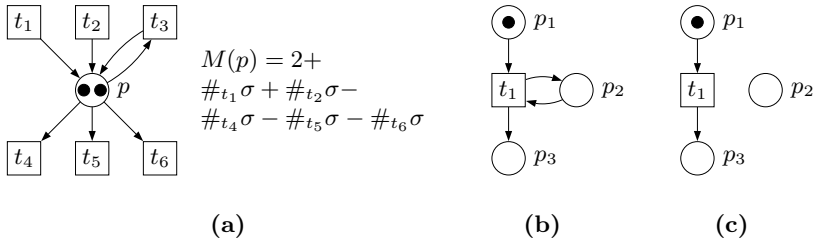


Fig. 2 Marking equation (only one place with its environment and initial marking is shown) **(a)**; and two net systems which have distinct sets of reachable markings but are indistinguishable by the marking equation **(b,c)**. (Note that these net systems have the same incidence matrix and the same initial marking, and so the same set of solutions of the marking equation.)

$t \in T$ is *enabled* at a marking M, denoted $M[t\rangle$, if for every $p \in {}^\bullet t$, $M(p) \geq 1$. Such a transition can be *executed* or *fired*, leading to a marking M' given by $M' \stackrel{\text{df}}{=} M - {}^\bullet t + t^\bullet$. We denote this by $M[t\rangle M'$. The set of *reachable* markings of Σ is the smallest (w.r.t. $\subset$) set $[M_\Sigma\rangle$ containing M_Σ and such that if $M \in [M_\Sigma\rangle$ and $M[t\rangle M'$ for some $t \in T$ then $M' \in [M_\Sigma\rangle$. For a finite sequence of transitions $\sigma = t_1 \dots t_k$ ($k \geq 0$), we write $M[\sigma\rangle M'$ if there are markings $M_1, \dots, M_{k+1}$ such that $M_1 = M$, $M_{k+1} = M'$ and $M_i[t_i\rangle M_{i+1}$, for $i = 1, \dots, k$. If $M = M_\Sigma$, we call σ an *execution* or *run* of Σ.

A marking is *deadlocked* if it does not enable any transitions. A net system Σ is *deadlock-free* if none of its reachable marking is deadlocked. A net system Σ is *k-bounded* if, for every reachable marking M and every place $p \in P$, $M(p) \leq k$, and *safe* if it is 1-bounded. Moreover, Σ is *bounded* if it is k-bounded for some $k \in \mathbb{N}$. One can show that $[M_\Sigma\rangle$ is finite iff Σ is bounded.

Marking equation

Let $\Sigma = (N, M_\Sigma)$ be a net system, p be one of its places, and σ be a run of Σ such that $M_\Sigma[\sigma\rangle M$. By counting the tokens brought to and taken from p by the transitions in σ it is possible to calculate $M(p)$ as follows:

$$M(p) = M_\Sigma(p) + \sum_{t \in T} F(t,p) \cdot \#_t\sigma - \sum_{t \in T} F(p,t) \cdot \#_t\sigma \ ,$$

where $\#_t\sigma$ denotes the number of times a transition t occurs in σ (see Figure 2(a)). This linear equation holds for every place of Σ, and can be written in the matrix form as follows.

Let $p_1, \ldots, p_m$ and $t_1, \ldots, t_n$ be respectively the places and transitions of the net system Σ. The *Parikh vector* of a finite sequence σ of transitions of Σ is the vector $x_\sigma \stackrel{\text{df}}{=} (\#_{t_1}\sigma, \ldots, \#_{t_n}\sigma)^{\mathbf{T}}$, and one can identify any marking M of Σ with the vector $(M(p_1), \ldots, M(p_m))^{\mathbf{T}}$. Moreover, the *incidence matrix* of Σ is an $m \times n$ matrix $\mathfrak{I} = (\mathfrak{I}_{ij})$ such that, for all $i \leq m$ and $j \leq n$,

$$\mathfrak{I}_{ij} \stackrel{\text{df}}{=} F(t_j, p_i) - F(p_i, t_j) \ .$$

One can then show that if σ is a run of Σ such that $M_\Sigma[\sigma\rangle M$ then $M = M_\Sigma + \mathfrak{I} \cdot x_\sigma$.

This provides a motivation for investigating the feasibility (or solvability) of the following *marking equation*[1] (see [11,14,17,19]):

$$\begin{cases} M = M_\Sigma + \mathfrak{I} \cdot x \\ M \in \mathbb{N}^m \text{ and } x \in \mathbb{N}^n \ . \end{cases} \tag{1}$$

If M is fixed then the feasibility of the above system is a necessary (but, in general, not sufficient) condition for M to be reachable from M_Σ. This is so because the Parikh vector of every sequence σ such that $M_\Sigma[\sigma\rangle M$ is a solution of (1), but, in general, (1) can have *spurious* solutions which do not correspond to any run of Σ (see Figure 2(b,c)).[2]

Therefore, the set of markings M for which (1) is feasible is an over-approximation of the set of reachable markings of Σ. However, for the class of acyclic nets (in particular, branching processes defined below), this equation provides an *exact* characterisation of the set of reachable markings [19, Theorem 16] — the fact which is crucial for the model checking based on unfoldings (see Section 4).

In some situations one is interested whether a vector $x \in \mathbb{N}^n$ is a Parikh vector for *some* marking M, i.e., the constraint

$$\exists M \in \mathbb{N}^m : M = M_\Sigma + \mathfrak{I} \cdot x$$

is investigated. It is clear that it is equivalent to

$$\begin{cases} M_\Sigma + \mathfrak{I} \cdot x \geq \mathbf{0} \\ x \in \mathbb{N}^n \ . \end{cases} \tag{2}$$

This form of marking equation will be used in Section 4.

[1] Also called the *token conservation equation.*

[2] Note that, in general, the presence of *self-loops,* i.e., pairs of arcs (p,t) and (t,p), is not the sole cause of spuriousness.

Branching processes

We will not recall all the technical details relating to branching processes and net unfoldings, which can be found in, e.g., [7,11,13]. Instead, we will introduce the main ideas behind them and quote few relevant results.

A *branching process* β of a Petri net Σ is a finite or infinite labelled acyclic net which can be obtained through unfolding Σ, by successive firings of transitions, under the following assumptions: (i) one starts from a set of places (called *conditions*), one for each token of the initial marking; (ii) for each new firing a fresh transition (called an *event*) is generated; and (iii) for each newly produced token a fresh place (also called a *condition*) is generated. Each event (resp. condition) is labelled by the corresponding transition (resp. place on which the corresponding token was present).

There exists a unique (up to isomorphism) maximal (w.r.t. the prefix relation) branching process of Σ called the *unfolding* of Σ [5, 7], and denoted here by β_Σ. For example, the unfolding of the Petri net in Figure 3(a) is shown in part (b) of this figure.

The unfolding β_Σ is infinite whenever Σ can execute an infinite sequence of transitions; however, if Σ has finitely many reachable states then the unfolding eventually starts to repeat itself and can be truncated (by identifying a set of *cut-off* events beyond which the unfolding procedure is not continued) without loss of essential information. For a branching process β obtained in this way, the sets of conditions, events, arcs and cut-off events of β will be denoted by B, E, G and E_{cut}, respectively, (note that $E_{cut} \subseteq E$), and the labelling function by h. Such an h is a *net homomorphism*, i.e., it maps the conditions in the preset (resp. postset) of an event e bijectively to the preset (resp. postset) of $h(e)$ and, intuitively, it maps the *implicit* initial marking of β, denoted here by M_β, obtained by putting a single token in each condition which does not have an incoming arc, to the initial marking of Σ (i.e., $h\langle M_\beta\rangle = M_\Sigma$). Note that when talking about a *run* of the branching process β, we will mean any run from its implicit marking M_β.

Being a net homomorphism, h maps runs of β to runs of Σ; more precisely, if $M_\beta[e_1 \dots e_n\rangle M$ then $M_\Sigma[h(e_1) \dots h(e_n)\rangle h\langle M\rangle$. As already explained, in acyclic nets like β, a marking is reachable iff the corresponding marking equation has a solution, and hence branching processes can be used for efficient model checking [9,11,14–17].

Since β is acyclic, the transitive closure of its flow relation is a partial order $<$ on $B \cup E$, called the *causality relation*. (The reflexive order corresponding to $<$ will be denoted by $\leq$.) Intuitively, all the

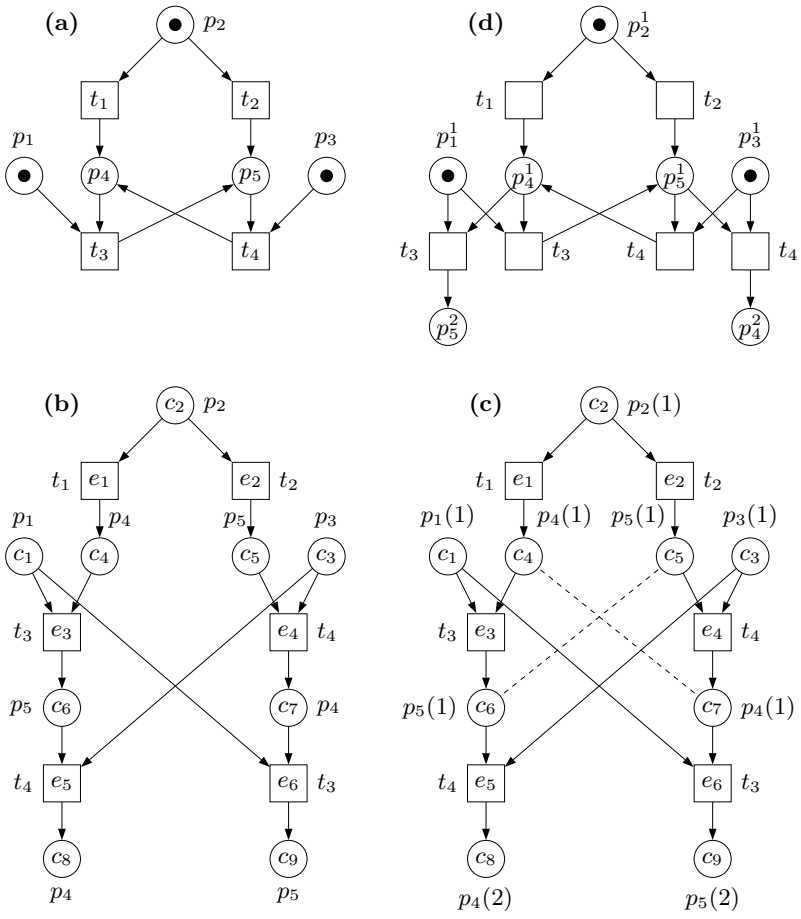


Fig. 3 A Petri net **(a)**; its unfolding **(b)**; its unfolding with the occurrence-depths of conditions shown in brackets and the conditions to be fused connected by dashed lines **(c)**; and its unravelling **(d)**.

events which are smaller than an event $e \in E$ w.r.t. $<$ must precede e in any run of β containing e.

Two nodes $x, y \in B \cup E$ are in *conflict*, denoted $x \# y$, if there are distinct events $e, f \in E$ such that ${}^\bullet e \cap {}^\bullet f \neq \varnothing$ and $e \leq x$ and $f \leq y$. Intuitively, no run of β can contain two events in conflict. Two nodes $x, y \in B \cup E$ are *concurrent*, denoted $x \| y$, if neither $x \# y$ nor $x \leq y$ nor $y \leq x$. Intuitively, two concurrent events can be enabled simultaneously, and executed in any order, or even concurrently, and two concurrent conditions can be simultaneously marked. For example,

in the branching process shown in Figure 3(b) the following relationships hold: $e_1 < e_5$, $e_3 \# e_4$ and $c_1 \| c_4$.

Due to structural properties of branching processes (such as acyclicity), the reachable markings of Σ can be represented using *configurations* of β. A *configuration* is a finite set of events $C \subseteq E$ such that for all $e, f \in C$, $\neg(e \# f)$ and, for every $e \in C$, $f < e$ implies $f \in C$. For example, in the branching process shown in Figure 3(b) $\{e_1, e_3, e_5\}$ is a configuration whereas $\{e_1, e_2, e_3\}$ and $\{e_1, e_5\}$ are not (the former includes events in conflict, $e_1 \# e_2$, while the latter does not include e_3, a causal predecessor of e_5). Intuitively, a configuration is a partial-order execution, i.e., an execution where the order of firing of some of its events (viz. concurrent ones) is not important.

After starting β from the implicit initial marking M_β and executing all the events[3] in C, one reaches the marking denoted by $Cut(C)$. $Mark(C) = h\langle Cut(C)\rangle$ denotes the corresponding marking of Σ, reached by firing a transition sequence corresponding to the events in C. Then β is *marking-complete w.r.t.* E_{cut} if, for every reachable marking M of Σ, there is a configuration C of β such that $C \cap E_{cut} = \varnothing$ and $Mark(C) = M$. Moreover, β is *complete* if it is marking-complete and, for each configuration C of β such that $C \cap E_{cut} = \varnothing$ and each event $e \notin C$ of β_Σ such that $C \cup \{e\}$ is a configuration of β_Σ, e is in E. This *preservation of firings* property is sometimes used for deadlock detection.

Complete branching processes are often called complete (unfolding) *prefixes*. One can build such a complete prefix in such a way that the number of non-cut-off events $|E \setminus E_{cut}|$ does not exceed the number of reachable markings of Σ [7,11].

3 Merged processes

In this section, we introduce the notion of a *merged process,* which is the main construction investigated in this paper.

Definition 1 (occurrence-depth). *Let β be a branching process of a Petri net Σ, and x be one of its nodes (condition or event). The* occurrence-depth *of x is defined as the maximum number of $h(x)$-labelled nodes on any directed path starting at a minimal (w.r.t. $<$) condition and terminating at x in the directed graph representing β.*

The above notion is well-defined since there is always at least one directed path starting at a minimal (w.r.t. $<$) condition and termi-

[3] I.e., each event in C is executed once, and no other event is executed — this is always possible.

nating at x, and the number of all such paths is finite. In Figure 3(c) the occurrence-depths of conditions are shown in brackets.

Definition 2 (merged process). *Given a branching process β, the corresponding* merged process $\mu = \mathfrak{Merge}(\beta)$ *is a Petri net which is obtained in two steps, as follows:*
Step 1: *the places of μ, called* mp-conditions, *are obtained by fusing together all the conditions of β which have the same labels and occurrence-depths; each mp-condition inherits its label and arcs from the fused conditions, and its initial marking is the total number of minimal (w.r.t. $<$) conditions which were fused into it.*
Step 2: *the transitions of μ, called* mp-events, *are obtained by merging all the events which have the same labels, presets and postsets (after step 1 was performed); each mp-event inherits its label from the merged events (and has exactly the same connectivity as either of them), and it is a* cut-off mp-event *iff* <u>*all*</u> *the events merged into it were cut-off events in β.*

Moreover, $\mu_\Sigma \stackrel{\text{df}}{=} \mathfrak{Merge}(\beta_\Sigma)$ is the merged process corresponding to the unfolding of Σ, called the unravelling *of Σ.*

Figure 3(c,d) illustrates this notion. In the sequel, $\hbar$ will denote the net homomorphism mapping the nodes of β to the corresponding nodes of μ, and $\widehat{E}$, $\widehat{B}$, $\widehat{G}$, $\widehat{M_\mu}$, $\widehat{E_{cut}}$ and $\widehat{h}$ will denote the set of its mp-events, the set of its mp-conditions, its flow relation, its initial marking, the set of its cut-off mp-events and the net homomorphism mapping the nodes of μ to the corresponding nodes of Σ (note that $\widehat{h} \circ \hbar = h$ and $\widehat{h}\langle\widehat{M_\mu}\rangle = M_\Sigma$).

Remark 1. A few properties of merged processes are listed below:

1. There is at most one mp-condition p^k resulting from the fusion of conditions labelled by place p of Σ occurring at depth $k \geq 1$.
2. Two distinct conditions in β having the same label and occurrence-depth are either concurrent or in conflict. Hence, if the original Petri net was safe then all the conditions in β which were fused into the same mp-condition p^k of μ were in conflict.
3. For two mp-conditions, p^k and p^{k+1}, there is a directed path from the former to latter. Moreover, if p^{k+1} is present and $k \geq 1$ then p^k is also present.
4. In general, μ is not acyclic (cycles can arise due to criss-cross fusions of conditions, as illustrated in Figure 3(c,d)). This, in turn, leads to complications for model checking; in particular, the marking equation can have spurious solutions not corresponding to any reachable marking.

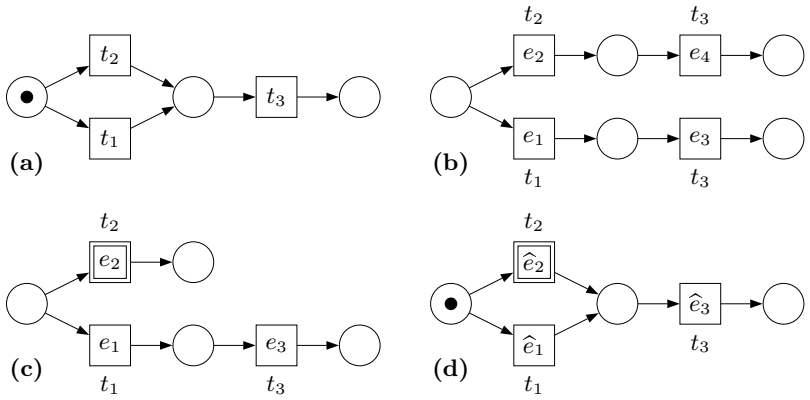


Fig. 4 A Petri net **(a)**; its unfolding **(b)**; one of its complete prefixes **(c)**; and the merged process corresponding to both the unfolding and the depicted prefix **(d)**. Cut-off events and cut-off mp-events are depicted as boxes with double borders.

To simplify model checking, one could refrain from fusing some conditions in Step 1 of Definition 2 if such a fusion would lead to a cycle. However, this would not be a satisfactory solution, since μ would not be uniquely defined, and it would result in a lower compression factor. So we decided to allow cycles, and strengthen the marking equation with additional constraints excluding spurious solutions of the marking equation (see Proposition 6 in the next section).

5. There can be events consuming conditions in the postset of a cut-off mp-event. Consider e.g., the Petri net in Figure 4(a) and its complete branching process shown in part (c) of this figure, where the event e_2 is a cut-off. In the corresponding merged process shown in Figure 4(d), the mp-event $\widehat{e}_2$ is a cut-off as well.
6. There is a strong correspondence between the runs of Σ and those of its unravelling: σ is a run of Σ iff $\sigma = \widehat{h}(\widehat{\sigma})$ for some run $\widehat{\sigma}$ of μ_Σ. To show this, we first note that σ is a run of Σ iff $\sigma = h(\widetilde{\sigma})$ for some run $\widetilde{\sigma}$ of β_Σ, and then observe the following:
   - For the ($\Longrightarrow$) implication, note that if a Petri net Σ'' is obtained from another Petri net Σ' by fusing places (and their tokens), then each run of Σ' is still a run of Σ''. Therefore, the Petri net obtained from β_Σ by performing just Step 1 of Definition 2 also satisfies the ($\Longrightarrow$) implication. Now, Step 2 simply removes duplicate events with the same label which does not affect the (images of) possible runs.
   - For the ($\Longleftarrow$) implication, recall that the labelling of nodes in β_Σ is a net homomorphism. Since no event has two equally labelled conditions in its preset (postset) respectively, the la-

belling after fusing conditions is still a net homomorphism, and so is the labelling after removing duplicate events. Hence, the implication follows from the general property that net homomorphisms map runs to runs.

A multiset $\widehat{C}$ of mp-events is an *mp-configuration* of μ if $\widehat{C} = \hbar\langle C\rangle$ for some configuration C of β_Σ. Note that there is a subtlety in this definition: we have to use the unfolding β_Σ of Σ rather than an arbitrary branching process β satisfying $\mu = \mathfrak{Merge}(\beta)$, since μ may contain mp-configurations which are not $\hbar$-images of any configurations in such a β, i.e., the mp-configurations of μ might be ill-defined if μ can arise from several different branching processes. E.g., for the Petri net in Figure 4(a), consider the unfolding β_Σ shown in part (b) and the branching process β shown in part (c) of this figure: both give rise to the same (up to isomorphism) merged process μ shown in part (d) of the figure, and the mp-configuration $\{\widehat{e}_2, \widehat{e}_3\}$ of μ is not an image of any configuration of β, but it is the image of the configuration $\{e_2, e_4\}$ of β_Σ.

If $\widehat{C}$ is an mp-configuration then the corresponding *mp-cut* $Cut(\widehat{C})$ is defined as the marking of μ reached by executing all[4] the events of $\widehat{C}$ starting from the initial marking $\widehat{M_\mu}$. Moreover, $Mark(\widehat{C}) \stackrel{\text{df}}{=} \widehat{h}\langle Cut(\widehat{C})\rangle$. Note that if $\widehat{C} = \hbar\langle C\rangle$ then $Mark(\widehat{C}) = Mark(C)$.

Canonical merged processes

Since $\mathfrak{Merge}$ is a deterministic transformation, one can easily define the *canonical* merged process as $\mathfrak{Merge}(\beta)$, where β is the canonical unfolding prefix of [13]. This allows for an easy import of the results of [11,13] related to the canonicity.

Completeness of a merged process

Marking-completeness of merged processes is defined similarly to that of branching processes. A merged process μ is *marking-complete w.r.t.* $\widehat{E_{cut}}$ if, for every reachable marking M of Σ, there exists an mp-configuration $\widehat{C}$ of μ such that $\widehat{C} \cap \widehat{E_{cut}} = \varnothing$ and $Mark(\widehat{C}) = M$.

Let C be a configuration of β and $\widehat{C} = \hbar\langle C\rangle$ be the corresponding configuration in μ. One can easily show that if C contains no cut-off

[4] I.e., each event in $\widehat{C}$ is executed as many times as it occurs in $\widehat{C}$, and no other event is executed — this is always possible. $Cut(\widehat{C})$ can be efficiently computed using, e.g., the marking equation (1).

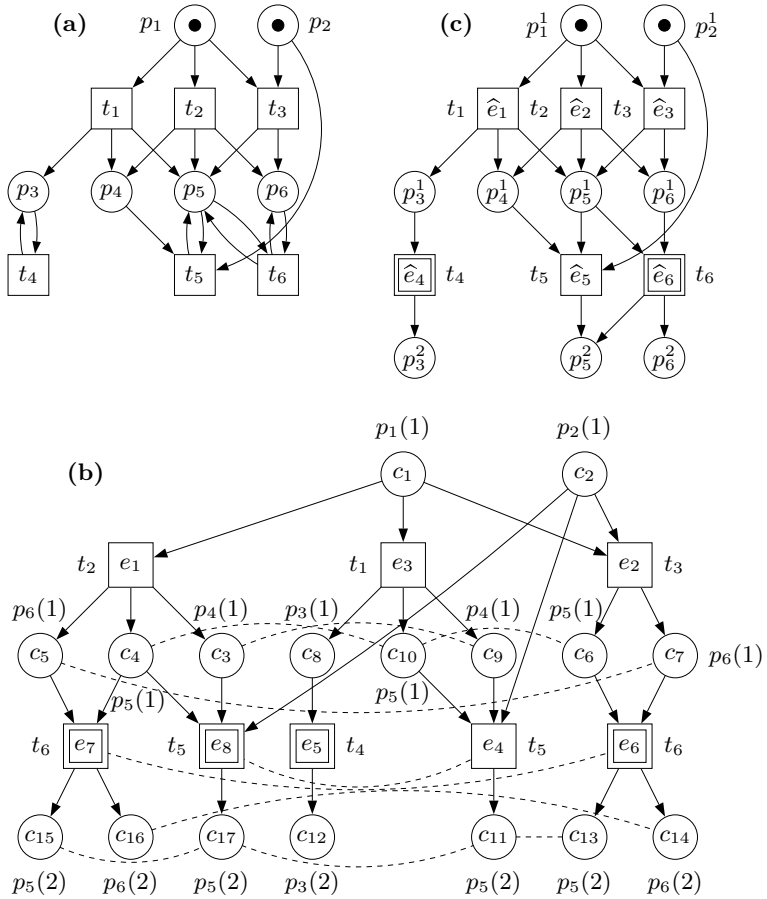


Fig. 5 A deadlock-free Petri net **(a)**; its unfolding with the occurrence-depths of conditions shown in brackets and the nodes to be fused connected by dashed lines **(b)**; and the corresponding merged process **(c)**. Cut-off events and cut-off mp-events are depicted as boxes with double borders.

event then $\widehat{C}$ contains no cut-off mp-events, and that $Mark(C) = Mark(\widehat{C})$. Hence the following holds:

Proposition 1. *If β is marking-complete branching process then μ is a marking-complete merged process.*

However, no such result holds for full completeness. Figure 5 shows a Petri net, its complete unfolding prefix, and the corresponding merged process. In the merged process, $\{\widehat{e}_2, \widehat{e}_5\}$ is an mp-configuration containing no cut-off mp-events and such that the corresponding mp-cut does not enable any mp-events. On the other hand, the Petri net

is deadlock-free, and so firings are obviously not preserved. Note that in the corresponding configuration $\{e_1, e_8\}$ of the unfolding, e_8 is a cut-off event, and so completeness is satisfied. However, e_8 is mapped to the same mp-event $\widehat{e}_5$ as a non-cut-off event e_4, and so $\widehat{e}_5$ is not a cut-off mp-event.

Since the completeness does not generally hold for merged processes, model checking algorithms developed for unfolding prefixes relying on the preservation of firings (e.g., some of the deadlock checking algorithms in [9,11,15–17]) cannot be easily transferred to merged processes. However, marking-completeness is sufficient for most purposes, as the transitions enabled by the final state of an mp-configuration can be easily found using the original Petri net.

The model checking algorithm proposed in Section 4 does not make use of cut-off mp-events, and so they can be removed from the merged process before model checking starts. Whether preservation of firings could be recovered and whether cut-off mp-events could be useful at all is a matter for further research.

The size of a merged process

The fusion of conditions in Definition 2 can only decrease the number of conditions without affecting the number of events or arcs; moreover, merging events can only decrease the number of events and arcs, without affecting the number of conditions. Hence, the following holds:

Proposition 2 (size). *If β is finite then μ is finite and $|\widehat{B}| \leq |B|$, $|\widehat{E}| \leq |E|$ and $|\widehat{G}| \leq |G|$.*

This result allows to import all the upper bounds proved for unfolding prefixes [7,11,13]. In particular, since for every safe Petri net Σ one can build a marking-complete branching process with the number of events not exceeding the number of reachable markings of Σ, the corresponding merged process μ has the same upper bound on the number of its events.

The upper bound given by Proposition 2 is rather pessimistic; in practice, merged processes turn out to be much more compact than the corresponding unfolding prefixes. Tables 1 and 2 show the results of our experiments. The popular set of benchmarks collected by J.C. Corbett [2] has been attempted. The meaning of the columns is as follows (from left to right): the name of the problem; the number of places and transitions in the original Petri net; the number of conditions, events and cut-off events in the unfolding prefix; the

Problem	Net		Unfolding				Unravelling				
	$\|P\|$	$\|T\|$	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	MC[s]	$\|\widehat{B}\|$	$\|\widehat{E}\|$	MC[s]	$\|\widehat{E}\|/\|T\|$	$\|E\|/\|\widehat{E}\|$
Q	163	194	16123	8417	1188	<1	248	256	<1	1.32	32.88
Speed	33	39	4929	2882	1219	<1	92	175	<1	4.49	16.47
Dac(6)	42	34	92	53	0	<1	42	35	<1	1.03	1.51
Dac(9)	63	52	167	95	0	<1	63	53	<1	1.02	1.79
Dac(12)	84	70	260	146	0	<1	84	71	<1	1.01	2.06
Dac(15)	105	88	371	206	0	<1	105	89	<1	1.01	2.31
Dp(6)	36	24	204	96	30	<1	60	37	<1	1.54	2.59
Dp(8)	48	32	368	176	56	<1	80	49	<1	1.53	3.59
Dp(10)	60	40	580	280	90	<1	100	61	<1	1.53	4.59
Dp(12)	72	48	840	408	132	<1	120	73	<1	1.52	5.59
Elev(1)	63	99	296	157	59	<1	73	89	<1	0.90	1.76
Elev(2)	146	299	1562	827	331	<1	150	241	<1	0.81	3.43
Elev(3)	327	783	7398	3895	1629	<1	304	588	<1	0.75	6.62
Elev(4)	736	1939	32354	16935	7337	<1	634	1387	<1	0.72	12.21
Hart(25)	127	77	179	102	1	<1	153	102	<1	1.32	1.00
Hart(50)	252	152	354	202	1	<1	303	202	<1	1.33	1.00
Hart(75)	377	227	529	302	1	<1	453	302	<1	1.33	1.00
Hart(100)	502	302	704	402	1	<1	603	402	<1	1.33	1.00
Key(2)	94	92	1310	653	199	<1	147	402	<1	4.37	1.62
Key(3)	129	133	13941	6968	2911	<1	201	1086	3	8.17	6.42
Key(4)	164	174	135914	67954	32049	<1	255	2054	25	11.80	33.08
Mmgt(1)	50	58	118	58	20	<1	61	58	<1	1.00	1.00
Mmgt(2)	86	114	1280	645	260	<1	111	282	<1	2.47	2.29
Mmgt(3)	122	172	11575	5841	2529	<1	159	662	<1	3.85	8.82
Mmgt(4)	158	232	92940	46902	20957	8	207	1206	67	5.20	38.89
Sent(25)	104	55	383	216	40	<1	120	81	<1	1.47	2.67
Sent(50)	179	80	458	241	40	<1	195	106	<1	1.33	2.27
Sent(75)	254	105	533	266	40	<1	270	131	<1	1.25	2.03
Sent(100)	329	130	608	291	40	<1	345	156	<1	1.20	1.87

Table 1 Experimental results for benchmarks with deadlocks.

time taken by deadlock checking based on unfoldings (discussed in the next section); the number of mp-conditions and mp-events in the corresponding merged process; the time taken by deadlock checking based on merged processes (discussed in the next section); and the ratios $|\widehat{E}|/|T|$ and $|E|/|\widehat{E}|$ giving measures of compactness of the merged process relative to the original Petri net and its unfolding prefix, respectively. The unfolding prefixes in our experiments were built using the algorithm described in [7,11,13], and the corresponding merged processes were obtained by applying the algorithm given in Definition 2. The time taken by this algorithm is not included in the tables because it was negligible (it did not exceed 0.1sec in all cases). The algorithm for building merged processes directly from Petri nets is a matter for future research — see the discussion in Section 5.

The last two columns in the tables show that merged processes can be by orders of magnitude smaller than unfolding prefixes, and, in many cases, are just slightly greater than the original Petri nets. In fact, in some of the examples merged processes are *smaller than the original Petri nets* due to the elimination of dead transitions and unreachable places.

Merged processes are much more amenable to model checking than general safe Petri nets — e.g., most of 'interesting' behaviourial prop-

Problem	Net		Unfolding				Unravelling				
	$\|P\|$	$\|T\|$	$\|B\|$	$\|E\|$	$\|E_{cut}\|$	MC[s]	$\|\widehat{B}\|$	$\|\widehat{E}\|$	MC[s]	$\|\widehat{E}\|/\|T\|$	$\|E\|/\|\widehat{E}\|$
ABP	43	95	337	167	56	<1	75	83	<1	0.87	2.01
BDS	53	59	12310	6330	3701	<1	145	359	<1	6.08	17.63
FTP	176	529	178085	89046	35197	3	304	875	3	1.65	101.77
CYCLIC(3)	23	17	52	23	4	<1	39	21	<1	1.24	1.10
CYCLIC(6)	47	35	112	50	7	<1	84	45	<1	1.29	1.11
CYCLIC(9)	71	53	172	77	10	<1	129	69	<1	1.30	1.12
CYCLIC(12)	95	71	232	104	13	<1	174	93	<1	1.31	1.12
DME(2)	135	98	487	122	4	<1	309	98	<1	1.00	1.24
DME(3)	202	147	1210	321	9	<1	463	148	<1	1.01	2.17
DME(4)	269	196	2381	652	16	<1	617	197	<1	1.01	3.31
DME(5)	336	245	4096	1145	25	<1	771	246	<1	1.00	4.65
DME(6)	403	294	6451	1830	36	<1	925	295	<1	1.00	6.20
DME(7)	470	343	9542	2737	49	<1	1079	344	<1	1.00	7.96
DME(8)	537	392	13465	3896	64	<1	1233	393	<1	1.00	9.91
DME(9)	604	441	18316	5337	81	<1	1387	442	<1	1.00	12.07
DME(10)	671	490	24191	7090	100	<1	1541	491	<1	1.00	14.44
DME(11)	738	539	31186	9185	121	1	1695	540	<1	1.00	17.01
DPD(4)	36	36	594	296	81	<1	81	78	<1	2.17	3.79
DPD(5)	45	45	1582	790	211	<1	102	100	<1	2.22	7.90
DPD(6)	54	54	3786	1892	499	<1	123	122	<1	2.26	15.51
DPD(7)	63	63	8630	4314	1129	<1	144	144	<1	2.29	29.96
DPFM(2)	7	5	12	5	2	<1	10	5	<1	1.00	1.00
DPFM(5)	27	41	67	31	20	<1	31	31	<1	0.76	1.00
DPFM(8)	87	321	426	209	162	<1	89	209	<1	0.65	1.00
DPFM(11)	1047	5633	2433	1211	1012	<1	313	1211	<1	0.21	1.00
DPH(4)	39	46	680	336	117	<1	87	108	<1	2.35	3.11
DPH(5)	48	67	2712	1351	547	<1	129	293	<1	4.37	4.61
DPH(6)	57	92	14590	7289	3407	<1	198	904	28	9.83	8.06
DPH(7)	66	121	74558	37272	19207	<1	277	2773	>10hrs	22.92	13.44
FURN(1)	27	37	535	326	189	<1	70	98	<1	2.65	3.33
FURN(2)	40	65	4573	2767	1750	<1	121	432	<1	6.65	6.41
FURN(3)	53	99	30820	18563	12207	<1	180	1224	<1	12.36	15.17
GASNQ(2)	71	85	338	169	46	<1	87	103	<1	1.21	1.64
GASNQ(3)	143	223	2409	1205	401	<1	173	325	<1	1.46	3.71
GASNQ(4)	258	465	15928	7965	2876	3	308	748	14	1.61	10.65
GASNQ(5)	428	841	100527	50265	18751	188	505	1449	1673	1.72	34.69
GASQ(1)	28	21	43	21	4	<1	35	21	<1	1.00	1.00
GASQ(2)	78	97	346	173	54	<1	96	111	<1	1.14	1.56
GASQ(3)	284	475	2593	1297	490	<1	316	509	<1	1.07	2.55
GASQ(4)	1428	2705	19864	9933	4060	4	1540	3004	20	1.11	3.31
OVER(2)	33	32	83	41	10	<1	51	39	<1	1.22	1.05
OVER(3)	52	53	369	187	53	<1	89	97	<1	1.83	1.93
OVER(4)	71	74	1536	783	237	<1	138	217	<1	2.93	3.61
OVER(5)	90	95	7266	3697	1232	<1	186	375	<1	3.95	9.86
RING(3)	39	33	97	47	11	<1	58	40	<1	1.21	1.18
RING(5)	65	55	339	167	37	<1	110	97	<1	1.76	1.72
RING(7)	91	77	813	403	79	<1	160	146	<1	1.90	2.76
RING(9)	117	99	1599	795	137	<1	210	194	<1	1.96	4.10
RW(6)	33	85	806	397	327	<1	51	85	<1	1.00	4.67
RW(9)	48	181	9272	4627	4106	<1	75	181	<1	1.00	25.56
RW(12)	63	313	98378	49177	45069	<1	99	313	<1	1.00	157.12

Table 2 Experimental results for deadlock-free benchmarks.

erties are known to be $\mathcal{PSPACE}$-complete for safe Petri nets [6], whereas in Section 4 we develop a non-deterministic polynomial-time algorithm for checking reachability-like properties of merged processes, i.e., many behaviourial properties of merged processes are in $\mathcal{NP}$. Since many such properties are known to be $\mathcal{NP}$-complete already for unfolding prefixes, the complexity class is not worsened if one uses merged processes rather than unfolding prefixes.

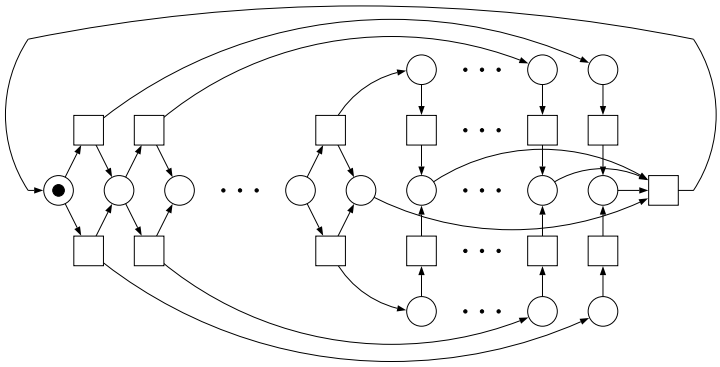


Fig. 6 An $LSFC^2$ Petri net whose unfolding prefix is exponential in its size.

Since merged processes are inherently more compact than unfolding prefixes, it would be natural to seek sharper upper bounds than the trivial ones given by Proposition 2. In particular, it would be interesting to identify subclasses of Petri nets whose unfolding prefixes can be exponential in the size of the original Petri net, but whose merged prefixes are guaranteed to be only polynomial. Below, we present two such results.

Proposition 3 (unravelling of acyclic Petri nets). *If Σ is an acyclic Petri net then its unravelling is isomorphic to the Petri net obtained from Σ by removing all its dead transitions and unreachable places.*

The above result easily follows from the fact that no token in an acyclic Petri net can 'visit' a place more than once, and thus the occurrence-depth of every condition in the unfolding of Σ is 1. On the other hand, unfolding prefixes of even safe acyclic Petri nets can be exponential in the size of the original nets, e.g., this is the case for the acyclic Petri net in Figure 1(a) with the dashed part taken into account.

Let us denote by $LSFC^k$ the class of live and safe free-choice Petri nets [3] whose transitions' postsets have cardinality at most $k \in \mathbb{N} \cup \{\infty\}$ (hence $LSFC^\infty$ denotes the whole class of live and safe free-choice Petri nets). It turns out that if $k \neq \infty$ then the marking-complete merged processes for the nets in $LSFC^k$ are polynomial in the size of the original nets, even though their unfolding prefixes can be exponential; e.g., one can make the Petri net in Figure 1(a) (with the dashed part taken into account) live by adding a subnet 'gathering' tokens at the end of the execution and returning a token

to the initial place, as shown in Figure 6. This net is in $LSFC^2$ and all its complete prefixes are exponential in its size.

Proposition 4 (merged processes of $LSFC^k$ nets). *For every $k \in \mathbb{N}$, there exists a polynomial P_k such that for every Petri net $\Sigma = (N, M_\Sigma)$ in $LSFC^k$ there exists a marking-complete merged process of Σ whose size is bounded by $P_k(|N|)$.*

Proof. Every marking of a safe free-choice Petri net can be reached in a number of steps that is polynomial in the net size [3]; let this number be m for a given $LSFC^k$ net Σ. Consider the branching process β obtained by truncating β_Σ in such a way that it contains exactly those events e which have at most $m-1$ causal predecessors (i.e., events preceding e in the partial order induced by $<$). Then, β is marking-complete, and the occurrence-depth of every condition in β is polynomial, and so there are only polynomially many mp-conditions in the corresponding merged process μ.

In the unfolding of a safe free-choice Petri net, the conditions in the preset of any event have the same occurrence-depth, since the token coming to such a condition cannot be consumed until the other conditions in this preset became marked. Hence there are only polynomially many presets of mp-events in μ.

Moreover, since the postsets of all the mp-events have the cardinality bounded by $k \in \mathbb{N}$ and there are only polynomially many mp-conditions, there are only polynomially many postsets of mp-events in μ. Hence, by Definition 2, there are only polynomially many mp-events in μ. □

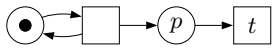
Note that the above proposition does not apply to $LSFC^\infty$ nets. It is still an open question whether one can always build marking-complete merged processes of polynomial size for such nets (the authors believe this is not always possible).

However, one should note that the expressive power of $LSFC^k$ for $k \geq 2$ is comparable with that of $LSFC^\infty$, since every transition of an $LSFC^\infty$ net with the postset of cardinality greater than k can be replaced by a tree of transitions with postsets of cardinality not exceeding k, and the resulting Petri net will be in $LSFC^k$.

Note that the above results are given for the sake of showing that merged processes are inherently more compact than unfolding prefixes rather than for practical model checking based on merged processes, since both acyclic Petri nets and $LSFC$ nets can be efficiently model checked by specialised algorithms [3, 19].

Finiteness of a merged process

In view of Proposition 2, μ is finite if β is. However, it is not obvious that the reverse holds, since, in general, infinitely many nodes of β can correspond to a single node of μ; for example, this is the case for the p-labelled conditions and t-labelled events of the full unfolding of the net below:



However, the analog of König's lemma for branching processes [11, 13] states that if β is infinite then there exists an infinite path in β. Since the number of places in Σ is finite, some place $p \in P$ is repeated infinitely many times along this path, and so the occurrence-depth of its instances grows unboundedly in β. Thus there are infinitely many instances of p after fusion, and the following holds:

Proposition 5 (finiteness). *μ is finite iff β is finite.*

Again, this result allows one to import into the new framework all the finiteness results proved for unfolding prefixes [7, 11, 13].

4 Model checking based on merged processes

In this section, we first describe a model checking approach based on unfolding prefixes. Then it is generalised to merged processes.

Model checking based on unfolding prefixes

Model checking algorithms [9, 11, 14–17] working on complete prefixes of Petri net unfoldings are usually based on the following nondeterministic algorithm:

choose a configuration $C \subseteq E \setminus E_{cut}$
if C violates the property /* e.g., deadlock-freeness */
then accept /* C is a certificate convertible to a witness trace */
else reject

Various kinds of solvers have been employed to implement it, e.g., ones based on mixed-integer programming [17], stable models of logic programs [9], integer programming [11], and Boolean satisfiability (SAT) [14]. For the last one, a system of simultaneous constraints having for each non-cut-off event e of the prefix a variable conf_e is

built (it might also contain other variables), and for every satisfying assignment A, the set of events $C \stackrel{\text{df}}{=} \{e \mid A(\mathsf{conf}_e) = 1\}$ is a configuration such that $\mathit{Mark}(C)$ violates the property being checked. This system of constraints usually has the form $\mathcal{CONF}^\beta \wedge \mathcal{VIOL}$. The role of the *configuration constraint,* $\mathcal{CONF}^\beta$, is to ensure that C is a configuration of the prefix β (not just an arbitrary set of events), and the role of the *violation constraint,* $\mathcal{VIOL}$, is to capture the property violation condition for a configuration C, so that if a configuration C satisfying this constraint is found then the property (e.g., deadlock-freeness) does not hold, and any ordering of events in C consistent with the causal order on the events of the prefix is a violation trace.

Given a set of events C, let $\mathcal{G}(C)$ denote the graph induced by the events of C together with their adjacent conditions and the causally minimal conditions of β. In order to encode $\mathcal{CONF}^\beta$ as a Boolean constraint, in addition to the variables conf_e, we introduce for each non-post-cut-off condition c a Boolean variable conf_c, conveying that c is a vertex of $\mathcal{G}(C)$. The variables conf_e and conf_c are related by the following constraints:

$$\mathsf{conf}_e \implies \bigwedge_{c \in {}^\bullet e} \mathsf{conf}_c \,, \qquad \text{for all } e \in E \setminus E_{cut},$$

$$\mathsf{conf}_c \iff \begin{cases} 1 & \text{if } {}^\bullet c = \emptyset \\ \bigvee_{e \in {}^\bullet c} \mathsf{conf}_e & \text{otherwise} \end{cases}, \qquad \text{for all } c \in B \setminus E^\bullet_{cut}.$$

Intuitively, the former formula ensures that if an event is in $\mathcal{G}(C)$ then all the conditions in its preset are also in $\mathcal{G}(C)$, and the latter one conveys that a condition is in $\mathcal{G}(C)$ iff either it is causally minimal or it has been 'produced' by some event in $\mathcal{G}(C)$. The size of both these formulae is linear in the size of the branching process. Note that since ${}^\bullet c$ is either an empty set or a singleton, the latter formula equates conf_c to either a constant or some conf_e, and so the variables conf_c can be completely eliminated. However, this form of the constraints is chosen for 'compatibility' with model checking based on merged processes, developed later in this section.

In order to ensure that C is a configuration of a branching process, in addition to the constraints above one has to require that no two events in C are in structural conflict, which can be achieved by the following pseudo-Boolean constraint:

$$\sum_{e \in c^\bullet \setminus E_{cut}} \mathsf{conf}_e \leq 1 \,, \quad \text{for all } c \in B \setminus E^\bullet_{cut} \text{ such that } |c^\bullet \setminus E_{cut}| > 1.$$

The size of this constraint is linear in the size of the branching process; however, an equivalent Boolean constraint

$$\bigwedge_{\substack{e,f \in c^\bullet \setminus E_{cut} \\ e \neq f}} \neg(\mathsf{conf}_e \wedge \mathsf{conf}_f)$$

is, in general, quadratic. A linear translation is possible (by introducing auxiliary variables), but it is more complicated and not discussed here, even though it was implemented in our tool.

Note that the built constraint $\mathcal{CONF}^\beta$ is essentially the marking equation (2) for β expressed as a Boolean formula rather than integer linear constraints.

We already mentioned that the role of the *violation constraint,* $\mathcal{VIOL}$, is to express the property violation condition for a configuration C. Suppose that the model checking problem at hand is to check if there is a reachable marking M such that the property $\mathcal{P}(M)$ holds. For example, for deadlock detection $\mathcal{P}(M)$ states that M does not enable any transition of the Petri net. Below we show how to build the $\mathcal{VIOL}$ constraint given $\mathcal{P}(M)$ (see also [9,11,14,17]).

We introduce for each non-post-cut-off condition c a Boolean variable cut_c, conveying that c belongs to $\mathit{Cut}(C)$. These variables are related to conf_* (i.e., the indexed conf-variables) by the following constraints:

$$\mathsf{cut}_c \iff \mathsf{conf}_c \wedge \bigwedge_{e \in c^\bullet \setminus E_{cut}} \neg \mathsf{conf}_e \ , \qquad \text{for all } c \in B \setminus E^\bullet_{cut}.$$

Intuitively, a condition c belongs to $\mathit{Cut}(C)$ iff it has been produced by some event in C (i.e., if c is a vertex of $\mathcal{G}(C)$) and it has not been consumed by any event of C. Furthermore, we introduce for each place p of the Petri net a Boolean variable mark_p, conveying that p belongs to $\mathit{Mark}(C)$. These variables are related to cut_* as follows:

$$\mathsf{mark}_p \iff \bigvee_{\substack{c \in B \setminus E^\bullet_{cut} \\ h(c) = p}} \mathsf{cut}_c \ , \qquad \text{for all } p \in P.$$

Intuitively, a place belongs to $\mathit{Mark}(C)$ iff some condition labelled by this place belongs to $\mathit{Cut}(C)$.

Now $\mathcal{VIOL}$ can be built simply by rewriting the constraint $\mathcal{P}(M)$ using the variables mark_*. For example, $\mathcal{VIOL}$ for deadlock checking has the form

$$\bigvee_{p \in {}^\bullet t} \neg \mathsf{mark}_p \ , \qquad \text{for all } t \in T.$$

Intuitively, this formula requires that no transition t is enabled by $Mark(C)$, i.e., at least one place in the preset of any transition t is not in $Mark(C)$. Marking reachability and coverability, mutual exclusion of places and many other properties can also be checked by modifying the latter formula.

Note that the size of all these formulae is linear in the size of the branching process. Moreover, neither cut-off events nor post-cut-off conditions are used in these formulae, and so the described approach can be used for any marking-complete branching process (i.e., the preservation of firings is not required, and if there are cut-off events in the branching process, they can be safely removed together with the conditions in their postsets before model checking starts).

Model checking based on merged processes

The purpose of the remaining part of this section is to generalise the described approach to merged processes. However, when doing this, one should bear in mind the following complications:

- An mp-configuration is generally a multiset (rather than a set) of mp-events. Though this is not a major problem, it does hamper verification employing Boolean solvers, as associating a single Boolean variable with each mp-event is no longer sufficient for representing an mp-configuration. But if the original Petri net is safe, the mp-configurations of its merged processes are sets.
- An easily testable characterisation of an mp-configuration is necessary (our 'indirect' definition of an mp-configuration as an $\hbar$-image of some configuration of the unfolding is not of much use for model checking). In what follows, we develop such a characterisation for mp-configurations of merged processes of safe Petri nets. Some issues make it non-trivial to develop such a characterisation:

  Spurious solutions of the marking equation Algorithms for model checking working on unfolding prefixes [9,11,14,17] are often based on the marking equation (perhaps expressed not as integer linear constraints but in some other form, e.g., as a Boolean formula) and the fact that for acyclic Petri nets it cannot have spurious solutions [19, Theorem 16]. Since merged processes are not generally acyclic, the marking equation *can* have spurious solutions.

  For example, the associated marking equation for the unravelling shown in Figure 3(d) (with the places and transitions enumerated in the left-to-right top-to-bottom fashion) has a spu-

rious solution corresponding to the unreachable marking $\{p_2^1\}$:

$$\begin{pmatrix}1\\1\\0\\0\\1\\0\\0\end{pmatrix} + \begin{pmatrix}-1&-1&0&0&0&0\\0&0&-1&-1&0&0\\1&0&-1&-1&1&0\\0&1&0&1&-1&-1\\0&0&0&0&-1&-1\\0&0&1&0&0&0\\0&0&0&0&0&1\end{pmatrix} \cdot \begin{pmatrix}0\\0\\0\\1\\1\\0\end{pmatrix} = \begin{pmatrix}1\\0\\0\\0\\0\\0\\0\end{pmatrix}$$

Intuitively, if one 'borrows' a token in p_4^1 then the t_3- and t_4-labelled mp-events forming a cycle can be executed, returning the borrowed token to p_4^1 and leading to the spurious marking $\{p_2^1\}$. Hence the constraint $\mathcal{CONF}^\beta$ described above, which is equivalent to the marking equation (2), cannot be directly re-used.

Spurious runs The correspondence between the runs and mp-configurations of μ is not very straightforward: some of its runs (e.g., the run comprised of the instance of t_1 followed by the left instance of t_3 in Figure 3(d)) do not form mp-configurations.

Below we solve these problems for merged processes of safe Petri nets.

The case of safe Petri nets

To capture the notion of an mp-configuration in the case when the original Petri net Σ is safe, we proceed as follows. Let C be a configuration of β, and $\widehat{C}$ be a set of mp-events of μ. Similarly to $\mathcal{G}(C)$, we define $\mathcal{G}(\widehat{C})$ as the graph induced by the mp-events of $\widehat{C}$ together with their adjacent mp-conditions and the initially marked mp-conditions of μ. We say that $\widehat{C}$ satisfies:

$\mathcal{ME}$ if $\widehat{C}$ is a solution of the marking equation (2) for μ;
$\mathcal{ACYCLIC}$ if $\mathcal{G}(\widehat{C})$ is acyclic;
$\mathcal{NG}$ (no-gap) if, for all $k > 1$ and all places p of Σ, the following holds: if p^k is a node in $\mathcal{G}(\widehat{C})$ then p^{k-1} is also a node in $\mathcal{G}(\widehat{C})$, and there is a directed path from p^{k-1} to p^k in $\mathcal{G}(\widehat{C})$.[5]

Note that if $\widehat{C} = \hbar(C)$ then:

(*) $\hbar\langle C\rangle = \hbar(C)$, and $\mathcal{G}(C)$ is isomorphic to $\mathcal{G}(\widehat{C})$ (including the labelling in terms of places and transitions).

[5] The latter part of this condition was missed in [12] even though it was tacitly assumed in the proof of Proposition 6.

Indeed, since Σ is safe and all conditions in $\mathcal{G}(C)$ are not in conflict, no two conditions in $\mathcal{G}(C)$ were fused together (see Remark 1(2)), and so no two events in $\mathcal{G}(C)$ were fused together either.

(**) $\mathcal{NG}$ holds and, if p^k is in $\mathcal{G}(\widehat{C})$ then there is a simple directed path in $\mathcal{G}(\widehat{C})$ going through $p^1, \ldots, p^k$.
Follows from (*) and the fact that all conditions in $\mathcal{G}(C)$ labelled by p are totally ordered, since C, being a configuration, contained no conflicts and no two concurrent conditions in a branching process of a safe Petri net can have the same label.

The next result gives a *direct* characterisation of mp-configurations and is crucial for model checking:

Proposition 6 (mp-configurations in the safe case). *A set of mp-events $\widehat{C}$ is an mp-configuration iff $\mathcal{ME} \wedge \mathcal{ACYCLIC} \wedge \mathcal{NG}$ holds for $\widehat{C}$.*

Proof. ($\Longrightarrow$) $\mathcal{ME}$ and $\mathcal{ACYCLIC}$ follow from (*), and $\mathcal{NG}$ from (**).
($\Longleftarrow$) We proceed by induction on the number of mp-events in $\widehat{C}$.
Base case: $\hbar(\varnothing) = \varnothing$.
Inductive step: Due to $\mathcal{ACYCLIC}$ there is a maximal mp-event $\widehat{e} \in \widehat{C}$. Let $\widehat{C}' \stackrel{\text{df}}{=} \widehat{C} \setminus \{\widehat{e}\}$. We first observe that $\widehat{C}'$ satisfies $\mathcal{ME}$ and $\mathcal{ACYCLIC}$ (because $\widehat{e}$ was maximal) and $\mathcal{NG}$ (because $\widehat{e}$ was maximal and (**) holds). Hence, by the induction hypothesis, $\widehat{C}'$ is an mp-configuration, i.e., $\widehat{C}' = \hbar(C')$ for some configuration C' of β_Σ, and so, by (*), $\mathcal{G}(\widehat{C}')$ and $\mathcal{G}(C')$ are isomorphic graphs.

By $\mathcal{ME}$, $\widehat{e}$ is enabled by $Cut(\widehat{C}')$, and by graph isomorphism, an $\widehat{h}(\widehat{e})$-labelled event e is enabled by C' and $\hbar({}^\bullet e) = {}^\bullet\widehat{e}$. For each condition $c \in e^\bullet$, there exists an mp-condition $\widehat{c} \in \widehat{e}^\bullet$ with the same label p. Let the occurrence-depth of c be k. Then $\hbar(c) = p^k$. By $\mathcal{NG}$, $\mathcal{ACYCLIC}$ and (**), $p^k \in \widehat{e}^\bullet$.

Thus $\widehat{h}(\widehat{e}) = h(e)$, $\hbar({}^\bullet e) = {}^\bullet\widehat{e}$ and $\hbar(e^\bullet) = \widehat{e}^\bullet$, and so $\hbar(e) = \widehat{e}$. Therefore, $\hbar(C' \cup \{e\}) = \widehat{C}' \cup \{\widehat{e}\} = \widehat{C}$, i.e., $\widehat{C}$ is an $\hbar$-image of a configuration and thus an mp-configuration. □

Hence it is enough for model checking to take

$$\mathcal{CONF}^\mu \stackrel{\text{df}}{=} \mathcal{ME} \wedge \mathcal{ACYCLIC} \wedge \mathcal{NG}$$

and apply an algorithm similar to that described at the beginning of this section for unfolding prefixes.

The implementation of the $\mathcal{ME}$ constraint as a Boolean formula is very similar to that for $\mathcal{CONF}^\beta$, and the implementation of $\mathcal{VIOL}$

is essentially the same as for branching processes. However, when removing cut-off mp-events from the merged process before model checking, one has to bear in mind that not all post-cut-off mp-conditions can now be removed (see Remark 1(5)). An mp-condition can be removed only if *all* the mp-events in its preset are cut-off mp-events.

To implement the $\mathcal{ACYCLIC}$ constraint, we re-formulate the problem as follows: given a digraph $G = (V, E)$ (representing μ) with a Boolean variable conf_v associated with each vertex $v \in V$, construct a Boolean formula $\mathcal{ACYCLIC}$ (depending on the variables conf_* and, perhaps, new auxiliary variables) such that, given an assignment to variables conf_*, the formula obtained from $\mathcal{ACYCLIC}$ by substituting the variables conf_* by their values is satisfiable iff the subgraph of G induced by the vertices whose corresponding variables were assigned to 1 is acyclic. (Note that $\mathcal{ME}$, $\mathcal{NG}$ and $\mathcal{VIOL}$ also contain the variables conf_*.)

Since each cycle is contained in some strongly connected component of G, one can partition G into its strongly connected components, generate such a constraint for each of them separately, and form $\mathcal{ACYCLIC}$ as their conjunction. For each strongly connected component $G_k = (V_k, E_k)$ of $G = (V, E)$, let $>_k$ be a total order on V_k. A vertex $v \in V_k$ is a *feedback vertex w.r.t.* $>_k$ if there exists $w \in V_k$ such that $(w, v) \in E_k$ and $w >_k v$. (In practice, the orderings $>_k$ are chosen with the view to heuristically minimise the number of feedback vertices.) Then for each such a feedback vertex $v \in V_k$ the following formula is generated (reach_* are auxiliary variables created separately for each such v):

$$(\mathsf{conf}_v \Rightarrow \mathsf{reach}_v) \wedge \bigwedge_{\substack{(x,y)\in E_k \\ x \geq_k v \wedge y >_k v}} ((\mathsf{reach}_x \wedge \mathsf{conf}_y) \Rightarrow \mathsf{reach}_y) \wedge \bigwedge_{\substack{(w,v)\in E_k \\ w >_k v}} \neg \mathsf{reach}_w \ .$$

If the subgraph induced by conf_* in G_k has a cycle, then the minimal vertex of this cycle is a feedback vertex. Hence, the idea behind this formula is to perform a reachability analysis in G_k starting from v and ignoring all the vertices which precede v in the chosen order or are not selected (i.e., the associated variables conf_* have the value 0). Note that if the values of the variables conf_* are fixed then this formula is unsatisfiable iff at least one of the sources of the feedback arcs ending at v is reachable from v (and hence there is a cycle); moreover, the unsatisfiability can be proven by unit resolution alone, i.e., one can setup the solver not to branch on the variables reach_*.

The $\mathcal{NG}$ constraint has been implemented as a conjunction of implications of the form $\mathsf{conf}_{p^k} \Rightarrow \mathsf{conf}_{p^{k-1}}$, for all mp-conditions p^k such that $k > 1$. (Intuitively, $\mathsf{conf}_{p^k} = 1$ conveys that p^k is in $\mathcal{G}(\widehat{C})$;

similarly, $\mathsf{conf}_{\widehat{e}} = 1$ conveys that $\widehat{e}$ is in $\mathcal{G}(\widehat{C})$, for each non-cut-off mp-event $\widehat{e}$ of μ.) Moreover, in order to express that $p^1, \ldots, p^k$ should be visited in the right order, one can modify the $\mathcal{ACYCLIC}$ constraint by augmenting the digraph G with the additional arcs (p^{k-1}, p^k), for all mp-conditions p^k such that $k > 1$. (This amends the algorithm of [12].)

Note that such an approach guarantees that: (i) there is no directed path from p^k to p^{k-1} in $\mathcal{G}(\widehat{C})$, since such a path, together with the arc (p^{k-1}, p^k) would result in a cycle, violating thus $\mathcal{ACYCLIC}$; and (ii) there is a directed path from p^{k-1} to p^k in $\mathcal{G}(\widehat{C})$. Indeed, to the contrary, suppose that there is no such a path. Then the removal of all the successors of p^{k-1} and p^k (together with their incident arcs) in the acyclic graph $\mathcal{G}(\widehat{C})$ results in a graph still containing p^{k-1} and p^k and satisfying $\mathcal{ACYCLIC} \wedge \mathcal{ME}$. In particular, this acyclic graph defines a partial order on its vertices that correspond to mp-events. Increasing this partial order to a total one results in a run $\widehat{\sigma}$ of μ leading to a marking of μ putting tokens inside both p^{k-1} and p^k (note that since all the successors of p^{k-1} and p^k have been removed, these tokens can never be consumed). Since $\widehat{h}$ is a net homomorphism, $\widehat{h}(\widehat{\sigma})$ is a run of the original Petri net putting at least two tokens within p, which contradicts the assumption that the original Petri net is safe.

One can show that the sizes of $\mathcal{ACYCLIC}$ and $\mathcal{NG}$, and hence the size of $\mathcal{CONF}^\mu \wedge \mathcal{VIOL}$, are polynomial in the size of the merged process.

We implemented a deadlock checking algorithm based on merged processes using zChaff [18] as the underlying SAT solver. All the experiments were conducted on a PC with a Pentium^{TM} IV/3.2GHz processor and 2G RAM.[6]

The experimental results in Tables 1 and 2 show that the developed model checking algorithm is quite practical; in fact it was as fast as the unfolding based model checking on most of the benchmarks. On the other hand, its performance deteriorated on the Dph and Gasnq series. We reckon that this is due to our inefficient implementation of the $\mathcal{ACYCLIC}$ constraint, and that this can be significantly improved.

The point we are making with these results is that: merged processes are a more compact behaviour representation than unfolding prefixes, but still allow model checking of reachability-like properties

[6] The PC used was different from that in [12], which explains the differences between the runtimes there and in this paper. Moreover, as explained above, the $\mathcal{NG}$ constraint has been amended, which resulted in improvement of some of the runtimes.

in at least comparable time. Since space considerations are of utmost importance in model checking, we regard this as very promising — although, to make merged processes practical, one still has to develop an *unravelling algorithm* that builds them directly from Petri nets instead of deriving them from unfolding prefixes.

5 Conclusions and future work

We proposed the notion of a merged process — a new condensed representation of a Petri net's behaviour allowing one to contain state space explosion arising not only from concurrency, but also from a sequence of choices, and from non-safeness of the Petri net. Experimental results show that merged processes can be smaller by orders of magnitude than the corresponding unfolding prefixes, and are in many cases not much bigger than the original Petri nets. Many results developed for Petri net unfoldings (related to canonicity, finiteness, completeness and size) have been transferred to the new framework. Moreover, we proved sharper upper bounds for some of the net subclasses, and directly characterised the mp-configurations of merged processes of safe Petri nets, which allowed us to develop a model checking algorithm.

In order to stimulate further research in this area, we present below a few possible directions for future study.

Direct characterisation of merged processes

Currently, a merged process is defined as the result of applying the $\mathfrak{Merge}$ transformation to a branching process. It would be interesting to find a direct structural characterisation of merged processes (cf. the characterisation of branching processes by occurrence nets).

Direct characterisation of (general) mp-configurations

A direct characterisation of mp-configurations of merged processes of non-safe Petri nets is still an open problem. Here we briefly summarise our current understanding. First of all, the marking equation must hold for mp-configurations. The natural generalisation of the no-gap constraint requiring that the number of tokens which 'visited' p^{k+1} should not exceed that for p^k in the merged process is, unfortunately, incorrect, as shown in Figure 7. The acyclicity constraint is also not applicable any more — the mp-events in an mp-configuration can, in

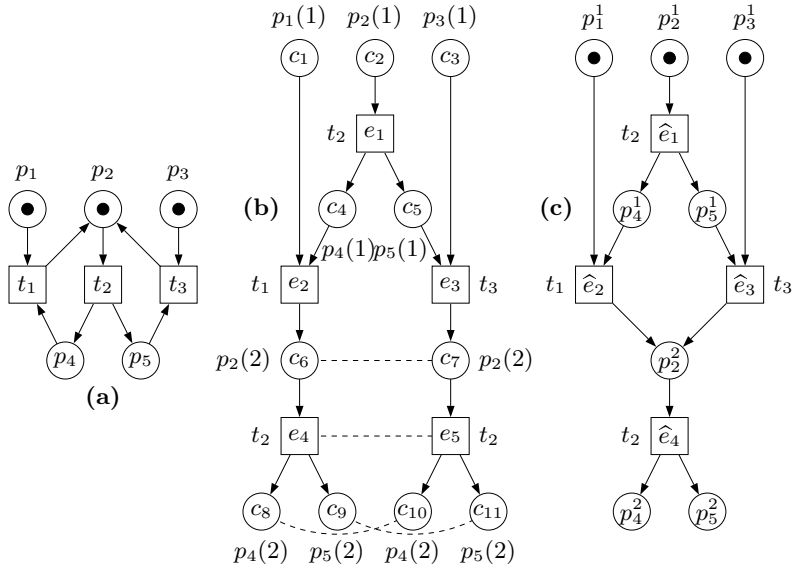


Fig. 7 A non-safe Petri net **(a)**; its unfolding with nodes to be fused connected by dashed lines **(b)**; and its unravelling **(c)** showing that the generalised no-gap constraint fails for the mp-configuration $\{\widehat{e}_1, \widehat{e}_2, \widehat{e}_3\}$ and mp-conditions p_2^1 and p_2^2.

general, induce cycles. Thus it should be generalised along the following lines. One should restrict the runs in such a way that each token in the merged process visits the same mp-condition at most once, and visits the mp-conditions labelled by the same place but having different occurrence-depths in the right order and without skipping any of them. Hence the 'life-history' of each individual token should be acyclic, and the cycles in an mp-configuration are an artefact of overlapping of several independent acyclic life-histories. Intuitively, an mp-configuration should be decomposable into several independent acyclic 'waves' of tokens.

More efficient model checking

It would be important for applications to improve the efficiency of the model checking algorithm. It should be noted that a decade of research on unfolding-based model checking brought about algorithms which are by a few orders of magnitude faster than the original ones [9,11,15–17]. We envisage that there is a scope for significant improvement of the proposed unravelling-based algorithm as well. In particular, our implementation of the acyclicity constraint was quite basic, and so the following lines of research look promising:

- It might be possible to replace the general acyclicity constraint by an equivalent problem-specific one. However, we would need a better understanding of the structure of mp-configurations to do that.
- Some state-of-the-art SAT solvers, e.g., MiniSat [4], allow one to use non-clausal constraints. This may be very helpful for efficiently expressing $\mathcal{ACYCLIC}$.

Unravelling algorithm

In this paper, we transform unfolding prefixes into merged processes using the algorithm derived from Definition 2. Of course, this is not practical, since one has to build the unfolding prefix first. To make merged processes practical, one has to develop an algorithm building them directly from Petri nets.

The main idea behind such an algorithm is to repeatedly solve the problem of finding a new mp-event which can be used as a possible extension of the built part of the unravelling. It is easily reducible to a model checking problem and can be solved using the SAT technique developed in this paper. Moreover, since the SAT instances to be solved are related, the efficiency can be improved by running the SAT solver in the *incremental mode,* i.e., on the subsequent runs the solver can use some of the useful information (e.g., learnt clauses, see [18]) collected so far.

There exists a working prototype of such an unravelling algorithm, and a paper describing it is currently being prepared for publication.

Acknowledgements

The authors would like to thank Keijo Heljanko for a helpful discussion about expressing $\mathcal{ACYCLIC}$ and Javier Esparza for sharing his expertise on *LSFC* nets.

References

1. E. M. Clarke, O. Grumberg and D. Peled: *Model Checking*. MIT Press (1999).
2. J. C. Corbett: Evaluating Deadlock Detection Methods for Concurrent Software. *IEEE Transactions on Software Engineering* 22 (1996) 161–180.
3. J. Desel and J. Esparza: *Free Choice Petri Nets*. Cambridge Tracts in Theoretical Computer Science 40, Cambridge University Press (1995).
4. N. Eén and N. Sörensson: An Extensible SAT-solver. Proc. of *SAT'03*, LNCS 2919 (2003) 502–518.

5. J. Engelfriet: Branching Processes of Petri Nets. *Acta Informatica* 28 (1991) 575–591.
6. J. Esparza: Decidability and Complexity of Petri Net Problems — an Introduction. In: *Lectures on Petri Nets I: Basic Models*, LNCS 1491 (1998) 374–428.
7. J. Esparza, S. Römer and W. Vogler: An Improvement of McMillan's Unfolding Algorithm. *Formal Methods in System Design* 20 (2002) 285–310.
8. D. Harel, H. Lachover, A. Naamad, A. Pnueli, et. al.: STATEMATE: a Working Environment for the Development of Complex Reactive Systems. *IEEE Transactions on Software Engineering* 16 (1990) 403–414.
9. K. Heljanko: Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets. *Fundamenta Informatica* 37 (1999) 247–268.
10. G. Kahn: The Semantics of a Simple Language for Parallel Programming. Proc. of *IFIP Congress*, North Holland (1974) 471–475.
11. V. Khomenko: *Model Checking Based on Prefixes of Petri Net Unfoldings*. PhD Thesis, School of Computing Science, University of Newcastle (2003).
12. V. Khomenko, A. Kondratyev, M. Koutny and V. Vogler: Merged Processes — a New Condensed Representation of Petri Net Behaviour. Proc. of *CONCUR'05*, LNCS 3653 (2005) 338–352.
13. V. Khomenko, M. Koutny and V. Vogler: Canonical Prefixes of Petri Net Unfoldings. *Acta Informatica* 40 (2003) 95–118.
14. V. Khomenko, M. Koutny and A. Yakovlev: Detecting State Coding Conflicts in STG Unfoldings Using SAT. *Fundamenta Informatica* 62 (2004) 221–241.
15. K. L. McMillan: Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. Proc. of *CAV'92*, LNCS 663 (1992) 164–174.
16. K. L. McMillan: *Symbolic Model Checking: an Approach to the State Explosion Problem*. PhD thesis, CMU-CS-92-131 (1992).
17. S. Melzer and S. Römer: Deadlock Checking Using Net Unfoldings. Proc. of *CAV'97*, LNCS 1254 (1997) 352–363.
18. S. Moskewicz, C. Madigan, Y. Zhao, L. Zhang and S. Malik: CHAFF: Engineering an Efficient SAT Solver. Proc. of *DAC'01*, ASME Tech. Publ. (2001) 530–535.
19. T. Murata: Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE* 77 (1989) 541–580.
20. A. Valmari: The State Explosion Problem. In: *Lectures on Petri Nets I: Basic Models*, LNCS 1491 (1998) 429–528.