Direct Construction of Complete Merged Processes[☆]

V. Khomenko[*], A. Mokhov

School of Computing Science, Newcastle University, Newcastle upon Tyne, NE1 7RU, United Kingdom

Abstract

Merged process is a recently proposed condense representation of a Petri net's behaviour similar to a branching process (unfolding), which copes well not only with concurrency, but also with other sources of state space explosion like sequences of choices. They are by orders of magnitude more condense than traditional unfoldings, and yet can be used for efficient model checking.

However, constructing complete merged processes is difficult, and the only known algorithm is based on building a (potentially much larger) complete unfolding prefix of a Petri net, whose nodes are then merged. Obviously, this significantly reduces their appeal as a representation that can be used for practical model checking.

In this paper we develop an algorithm that avoids constructing the intermediate unfolding prefix, and builds a complete merged process directly. In particular, a challenging problem of truncating a merged process is solved.

Keywords: merged process, unravelling, Petri net unfolding, model checking, SAT, 2QBF.

1. Introduction

Formal verification, and in particular model checking of concurrent systems is an important and practical way of ensuring their correctness. However, the main drawback of model checking is that it suffers from the *state space explosion* problem [1]. That is, even a relatively small system specification can (and often does) yield a very large state space. To alleviate this problem, many model checking techniques use a condense representation of the full state space of the system. Among them, a prominent technique is McMillan's (finite prefixes of) Petri net unfoldings (see, e.g. [2–4]). 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.

[☆]This research was supported by the EPSRC grant EP/G037809/1 (VERDAD).

[*]Corresponding author

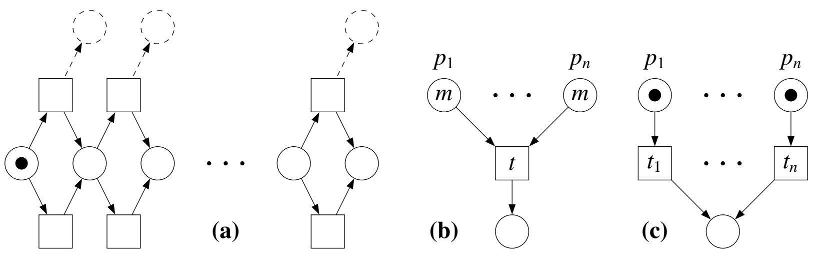

Figure 1: Examples of Petri nets.

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 [5].

First, consider Fig. 1(a) with the dashed part not taken into account. The cut-off condition proposed in [2] copes well with this Petri net (since the marking reached after either choice on each stage is the same — in fact, there are 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 added, the smallest complete prefix is exponential, since no event can be declared a cut-off (intuitively, each reachable marking 'remembers' its past). Thus sequences of choices leading to different markings often yield exponential prefixes.

Another problem arises when one tries to unfold non-safe Petri nets, e.g. one in Fig. 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 [2]. 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 Fig. 1(c), it yields an exponential prefix, while the traditional unfolding technique would yield a linear one [2].

The problems with unfolding prefixes described above should be viewed in the light of the fact that all these examples have a very simple structure — viz. they are all acyclic, and thus many model checking techniques, in particular those based on the *marking equation* [3, 6, 7], could be applied *directly to the original Petri nets*. And so it may happen that an exponential prefix is built for a relatively simple problem!

In [5], a new condense representation of a Petri net's behaviour called *merged processes (MPs)* was proposed, which remedies the problems outlined above. It copes well not only with concurrency, but also with the other mentioned sources of state space explosion, 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 this representation is to fuse some nodes in the complete prefix, and use the resulting net as the basis for verification. For example, the unfolding of the net shown in Fig. 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.

It turns out that for a safe Petri net model checking of a reachability-like property (i.e. the existence of a reachable state satisfying a predicate given by a Boolean expression) can be efficiently performed on its MP, and [5] provides a method for reducing this problem to SAT (with the size of the SAT instance being polynomial in the sizes of the MP and the property). Moreover, the experimental results in [5] indicate that this method is quite practical.

However, constructing complete MPs is difficult, and the only known algorithm is based on building a (potentially much larger) complete unfolding prefix, whose nodes are then merged [5]. Obviously, this significantly reduces their appeal as a condense representation that can be used for practical model checking.

In this paper we develop an algorithm (referred subsequently as the *unravelling algorithm*) that avoids constructing the intermediate unfolding prefix, and builds a complete MP directly. In particular, a challenging problem of truncating an MP is solved, by reducing it to 2QBF, i.e. to satisfiability of a fully quantified Boolean formula of the form $\forall X \exists Y\ \varphi$, where φ is a Boolean formula in conjunctive normal form (CNF) and X and Y are disjoint sets of Boolean variables such that $X \cup Y$ are exactly the variables occurring in φ.

This paper is an extended and more developed version of [8] and the corresponding technical report [9]. While the previous version of the unravelling algorithm was just a proof of concept, and could not compete with unfolders, the improvements described in this paper made it much more competitive. In particular, the following advancements have been made:

- A new total adequate order that refines the size order and can be easily encoded in a SAT instance has been developed, see Sect. 3.2. This had a positive impact on the performance of the unravelling algorithm and the size of the computed MP.
- The comparison with the unfolding algorithm is now more fair and consistent:
  - In [8, 9] the unfolding algorithm used in the cut-off check only local configurations whereas the unravelling algorithm used all configurations; in this paper both algorithms use all configurations in the cut-off check.
  - Both algorithms in [8, 9] used the size-lexicographical adequate order (which is not total), which is consistent, but detrimental for the unfolding algorithm as it works much better with a total adequate order. However, the ERV adequate order [2] that is commonly used by unfolders has a very poor encoding into SAT, and so would be detrimental for the unravelling algorithm. In this paper both algorithms use the newly developed total adequate order (see Sect. 3.2), that is good for both of them.
- A number of optimisations have been introduced into the implementation of the unravelling algorithm, in particular a better use of incremental SAT.

2. Basic notions

In this section we recall the basic notions concerning SAT, QBF, Petri nets, their unfolding prefixes and merged processes (see also [2, 4, 6, 7, 10–13]).

2.1. Boolean satisfiability (SAT)

The *Boolean Satisfiability Problem (SAT)* consists in finding a *satisfying assignment*, i.e. a mapping $A : Var_\varphi \to \{0, 1\}$ defined on the set of variables Var_φ occurring in a given Boolean expression φ such that $\varphi|_A$ evaluates to 1. This expression is often assumed to be given in the *conjunctive normal form (CNF)* $\varphi = \bigwedge_{i=1}^{n} \bigvee_{l \in L_i} l$, i.e. it is represented as a conjunction of *clauses*, which are disjunctions of *literals*, each literal l being either a variable or the negation of a variable. It is assumed that no two literals in the same clause correspond to the same variable.

SAT is a canonical NP-complete problem. In order to solve it, SAT solvers perform exhaustive search assigning the values 0 or 1 to the variables, using heuristics to reduce the search space [13]. Many leading SAT solvers, can be used in the *incremental mode,* i.e. after solving a particular SAT instance the user can slightly change it (e.g. by adding and/or removing a small number of clauses) and execute the solver again. This is often much more efficient than solving these related instances as independent problems, because on the subsequent runs the solver can use some of the useful information (e.g. learnt clauses [13]) collected so far.

2.2. *Quantified Boolean formulae (QBF)*

A *Quantified Boolean Formula* is a formula of the form $Q_1 x_1 \dots Q_n x_n\ \varphi$, where each Q_i is either $\exists$ or $\forall$, each x_i is a Boolean variable, and φ is a Boolean expression. It is *fully quantified* if all the variables occurring in φ are quantified. For convenience, adjacent quantifiers of the same type are often grouped together, so the formula can be re-written as $Q_1 X_1 \dots Q_k X_k\ \varphi$, where X_is are pairwise disjoint non-empty sets of Boolean variables, and the quantifiers' types alternate. Furthermore, φ is often assumed to be given in CNF.

A fully quantified Boolean formula is equal to either 0 or 1, under the natural semantics. The problem of determining if such a formula is equal to 1 is PSPACE-complete. However, if the number of alternations of the quantifiers is a constant k, the complexity is the kth level of Stockmeyer's polynomial hierarchy PH [12]. PH is included in PSPACE, and it is conjectured that PH does not collapse (i.e. the inclusion between its levels is strict); this would imply that the inclusion of PH into PSPACE is also strict.

One can see that checking if a fully quantified QBF instance of the form $\exists X\ \varphi$ is equal to 1 amounts to SAT. Also, QBF instances of the form

$$\forall X \exists Y\ \varphi \tag{1}$$

are of particular interest for this paper. W.l.o.g., one can assume that φ is in CNF, as any instance of (1) can be converted into this form without changing the number of alternation of quantifiers and by only linearly increasing the size of the formula. The problem of checking if (1) equals to 1 is called 2QBF [11]. This problem, though more complicated then SAT (unless PH collapses), is nevertheless much simpler than general QBF and other PSPACE-complete problems like model checking (as these problems are separated by infinitely many levels of PH, unless PH collapses). Furthermore, though general QBF solvers have not reached maturity level of contemporary SAT solvers, 2QBF admits specialised methods, e.g. based on a pair of communicating SAT solvers [11].

2.3. *Petri nets*

A *net* is a triple $N \overset{\text{df}}{=} (P, T, F)$ 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*. The net is *finite* if 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, ${}^\bullet z \overset{\text{df}}{=} \{y \mid (y, z) \in F\}$ and $z^\bullet \overset{\text{df}}{=} \{y \mid (z, y) \in F\}$ denote the *preset* and *postset* of $z \in P \cup T$. In this paper, the presets of transitions are restricted to be non-empty, i.e. ${}^\bullet t \neq \emptyset$ for every $t \in T$. For a finite net N, we define the *size* of N as $|N| \overset{\text{df}}{=} |P| + |T| + |F|$.

A *Petri net (PN)* is a pair $\Sigma \overset{\text{df}}{=} (N, M_\Sigma)$ comprising a finite net $N = (P, T, F)$ and an (initial) marking M_Σ. A transition $t \in T$ 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 $M' \overset{\text{df}}{=}$

$M - {}^\bullet t + t^\bullet$. 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* of Σ.

A marking is *deadlocked* if it does not enable any transitions. A PN Σ is *deadlock-free* if none of its reachable marking is deadlocked. It 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.

A powerful tool in analysis of PNs is the so called *marking equation* [7], which states that the final number of tokens in any place p of a PN Σ can be calculated as $M_\Sigma(p)$ plus the number of tokens brought to p minus the number of tokens taken from p. The feasibility of this equation is a necessary condition for a marking to be reachable from M_Σ. However, the marking equation can have *spurious* solutions which do not correspond to any execution of Σ, i.e. it is not a sufficient condition, and yields an over-approximation of $[M_\Sigma\rangle$. However, for some PN classes, in particular for acyclic PNs (which include branching processes defined below), this equation gives an *exact* characterisation of $[M_\Sigma\rangle$ [7, Th. 16] — providing the basis for model checking algorithms based on unfolding prefixes [3, 6]. Moreover, exact characterisations of $[M_\Sigma\rangle$ can be obtained for some further PN classes by augmenting the marking equation with additional constraints; in particular, such a characterisation was obtained in [5] for MPs, and it will be essential for the proposed unravelling algorithm.

2.4. *Branching processes*

A *branching process* β of a PN 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 β_Σ of Σ called the *unfolding* of Σ [2, 14]. For example, the unfolding of the PN in Fig. 2(a) is shown in part (b) of this figure.

The unfolding β_Σ is infinite whenever Σ has executions of unbounded length; however, if Σ has finitely many reachable states then the unfolding eventually starts to repeat itself and thus 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. Note that when talking about an *execution* of β, we will mean any execution from its implicit initial marking M_β that comprises the initial conditions.

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 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 ${}^\bullet e \cap {}^\bullet f \neq \emptyset$ and $e \leq x$ and $f \leq y$. Intuitively, no execution of β can contain two events in

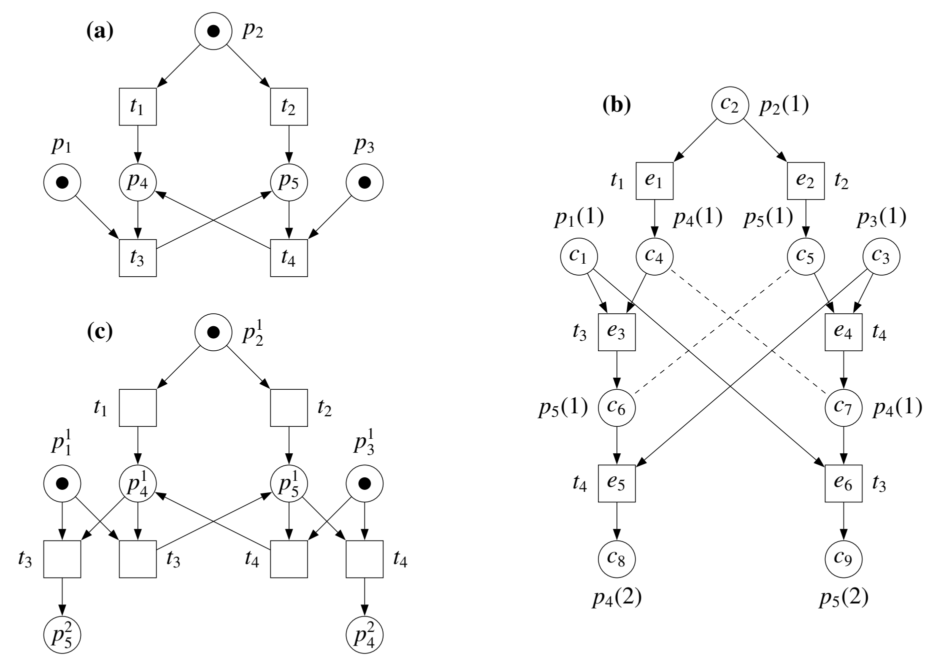

Figure 2: A Petri net **(a)**; its unfolding with the occurrence-depths of conditions shown in brackets and the conditions to be fused connected by dashed lines **(b)**; and its unravelling **(c)**.

conflict. Two nodes $x, y \in B \cup E$ are *concurrent*, denoted $x \parallel y$, if neither $x \# y$ nor $x \le y$ nor $y \le x$. 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 Fig. 2(b) the following relationships hold: $e_1 < e_5$, $e_3 \# e_4$ and $c_1 \parallel c_4$.

Due to structural properties of branching processes, 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 (i) for all $e, f \in C$, $\neg(e \# f)$; and (ii) for every $e \in C$, $f < e$ implies $f \in C$. For example, in the branching process shown in Fig. 2(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 partially ordered execution, i.e. an execution where the order of firing of some of its events (viz. concurrent ones) is not important. For every event e of β_Σ, $[e] \overset{\text{df}}{=} \{f \mid f \text{ is an event of } \beta_\Sigma \text{ and } f \le e\}$ is called the *local configuration* of e. Intuitively, it comprises e and all its causal predecessors.

After starting β from the implicit initial marking M_β and executing all the events in C, one reaches the marking (of β) denoted by $Cut(C)$. $Mark(C) \overset{\text{df}}{=} h(Cut(C))$ denotes the corresponding marking of Σ, reached by firing a transition sequence corresponding to the events in C. Let E_{cut} be a set of events of β. 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 $C \cap E_{cut} = \emptyset$ and $Mark(C) = M$. Moreover, β is *complete* if it is marking-complete and, for each configuration C of β such that $C \cap E_{cut} = \emptyset$ and for each event $e \notin C$ of β_Σ such that $C \cup \{e\}$ is a configuration of β_Σ, e is in β. This *preservation* *of firings* property is useful for deadlock detection.

Complete branching processes are often called complete (unfolding) *prefixes*. One can build 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 [2, 3]. The unfolding algorithms described there declare an event e cut-off if there is a smaller (w.r.t. some *adequate order* $\lhd$) *corresponding configuration* C in the already built part of the prefix containing no cut-off events and such that $Mark([e]) = Mark(C)$; here an *adequate order* is a strict well-founded partial order $\lhd$ on configurations of the unfolding that refines the set inclusion and is *preserved by finite extensions,* i.e. whenever $C \lhd C'$ and C and C' have the same final marking, $C \oplus I(E) \lhd C' \oplus E$ holds, where E is any finite extension of C' and $I(E)$ denotes an extension of C that is isomorphic to E. With some natural conditions on the way this cutting is performed, the resulting prefix is unique, even though the unfolding algorithm may be non-deterministic. This unique prefix is called *canonical,* and it can be defined in an algorithm-independent way [10].

As already mentioned, in acyclic nets like β, a marking is reachable iff the corresponding marking equation has a solution; since there is a correspondence between the reachable markings of Σ and those of any of its marking-complete branching processes β, the latter can be used for efficient model checking [3, 4, 6, 15].

2.5. Merged processes

Let β be a branching process of a PN 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 an initial condition and terminating at x in the acyclic digraph representing β. The occurrence-depth is well-defined since there is always at least one such a path, and the number of all such paths is finite. In Fig. 2(b) the occurrence-depths of conditions are shown in brackets.

Definition 1 (merged process). Given a branching process β, the corresponding *merged process* $\mu = \mathfrak{Merge}(\beta)$ is a PN 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 the initial 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 $\underline{\text{all}}$ the events merged into it were cut-off events in β.

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

Fig. 2(b,c) illustrates this definition, which also yields an algorithm for computing $\mathfrak{Merge}$. In the sequel, $\hbar$ will denote the mapping of the nodes of β to the corresponding nodes of μ, and we will use 'hats' and 'mp-' to distinguish the elements of μ from those of β, in particular, $\widehat{E}$, $\widehat{B}$, $\widehat{G}$, $\widehat{M_\mu}$, $\widehat{E_{cut}}$ will denote the set of mp-events, the set of mp-conditions, the flow relation, the initial marking and the set of cut-off mp-events of μ, and $\widehat{h}$ will denote the mapping of the nodes of μ to the corresponding nodes of Σ.

Note that in general, μ is not acyclic (cycles can arise due to criss-cross fusions of conditions, as illustrated in Fig. 2(b,c)). This, in turn, leads to complications for model checking, as the

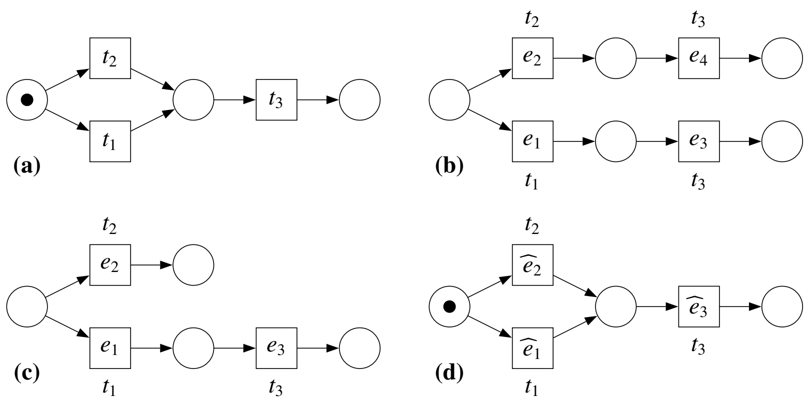

Figure 3: A Petri net **(a)**; its unfolding **(b)**; one of its unfolding prefixes **(c)**; and the merged process corresponding to both the unfolding and the depicted prefix **(d)**.

marking equation for μ can have spurious solutions not corresponding to any reachable marking, and thus has to be augmented with additional constraints, see Sect. 2.6.

A multiset $\widehat{C}$ of mp-events is an *mp-configuration* of μ if $\widehat{C} = \hbar(C)$ 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 configuration 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 PN in Fig. 3(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) MP 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[1] the events of $\widehat{C}$ starting from the initial marking $\widehat{M_\mu}$. Moreover, $Mark(\widehat{C}) \overset{\text{df}}{=} \widehat{h}(Cut(\widehat{C}))$. Note that if $\widehat{C} = \hbar(C)$ then $Mark(\widehat{C}) = Mark(C)$.

MPs of safe PNs have a number of special properties. First of all, their mp-configurations are sets (rather than multisets) of mp-events. Moreover, there is a one-to-one correspondence between the mp-configurations of the unravelling and the configurations of the unfolding, such that each mp-configuration $\widehat{C}$ is isomorphic to the corresponding configuration C, in the sense that the $\widehat{h}$-labelled digraph induced by the nodes in $\widehat{C} \cup \widehat{C}^\bullet \cup \widehat{M_\mu}$ in the unravelling is isomorphic to the h-labelled digraph induced by the nodes in $C \cup C^\bullet \cup M_\beta$ in the unfolding. Furthermore, one can partially transfer the notion of an event's local configuration from branching processes to MPs. Though the notion of *the* local configuration of an mp-event $\widehat{e}$ does not make sense, as $\widehat{e}$ can have many of them, one can specify some additional information to make this local configuration unique, viz. an mp-configuration $\widehat{C}$ such that $\widehat{e} \in \widehat{C}$ within which the local configuration of $\widehat{e}$ must be contained; it will be denoted by $[\widehat{e}]_{\widehat{C}}$. (Note that the uniqueness is still not guaranteed for MPs of unsafe PNs.)

[1]I.e., each mp-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 [7].

2.5.1. *Canonical merged processes*

Since $\mathfrak{Merge}$ is a deterministic transformation, one can define the *canonical* MP as $\mathfrak{Merge}(\beta)$, where β is the canonical unfolding prefix of [10]. This allows for an easy import of the results of [10] related to the canonicity. Furthermore, it simplifies the proof of correctness of the algorithm proposed in this paper, as it is enough to prove that it constructs $\mathfrak{Merge}(\beta')$, where β' is obtained by removing the cut-off events from the canonical prefix.

However, $\mathfrak{Merge}$ has problems with the cut-offs, as when merging events, it loses the information about their cut-off status, see Def. 1. For example, in Fig. 3(b) e_4 can be declared a cut-off, but this information is lost when it is merged with a non-cut-off event e_3, resulting in the MP shown in part (d) of the figure. This causes problems with the notion of completeness, which cannot be easily fixed; hence [5] suggests to be content with marking-completeness, which is sufficient for model checking.

The MPs produced by the unravelling algorithm proposed in this paper do not have this problem, and can be easily made complete. (And they coincide with the MPs produced by $\mathfrak{Merge}$ on non-cut-off mp-events.) Therefore, they may be considered as better candidates for being called canonical.

2.5.2. *Finiteness of merged processes*

It is easy to show that $\mathfrak{Merge}(\beta)$ is finite iff β is finite [5]. Again, this allows one to import all the finiteness results proved for unfolding prefixes [2, 10].

2.5.3. *Completeness of merged processes*

Marking-completeness of MPs is defined similarly to that of branching processes. An MP 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 $\widehat{C} \cap \widehat{E_{cut}} = \emptyset$ and $Mark(\widehat{C}) = M$. Moreover, μ is *complete* if it is marking-complete and, for each mp-configuration $\widehat{C}$ of μ such that $\widehat{C} \cap E_{cut} = \emptyset$ and each mp-event $\widehat{e} \notin \widehat{C}$ of μ_Σ such that $\widehat{C} \cup \{\widehat{e}\}$ is an mp-configuration of μ_Σ, $\widehat{e}$ is in μ (i.e. the firings are preserved).

Let C be a configuration of β and $\widehat{C} = \hbar(C)$ be the corresponding mp-configuration of μ. One can easily show that if C contains no cut-off events then $\widehat{C}$ contains no cut-off mp-events, and that $Mark(C) = Mark(\widehat{C})$. Hence, if β is a marking-complete branching process then $\mathfrak{Merge}(\beta)$ is a marking-complete MP [5].

Unfortunately, no such result holds for full completeness: [5] provides an example where $\mathfrak{Merge}(\beta)$ contains a false deadlock, even though β is a complete prefix of a deadlock-free PN. Hence, model checking algorithms developed for unfolding prefixes relying on the preservation of firings (e.g. some of the deadlock checking algorithms in [3, 4, 6, 15, 16]) cannot be easily transferred to MPs. 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 PN. (The model checking approach of [5], recalled in in Sect. 2.6, does not rely on preservation of firings.)

In contrast, the algorithm proposed in this paper allows one to build a complete MP (i.e. to preserve firings), and hence import the model checking algorithms making use of cut-off events, in particular those for deadlock detection.

2.5.4. *The size of a merged process*

The fusion of conditions in Def. 1 can only decrease the number of conditions, without affecting the number of events or arcs; moreover, merging events can only decrease the number of

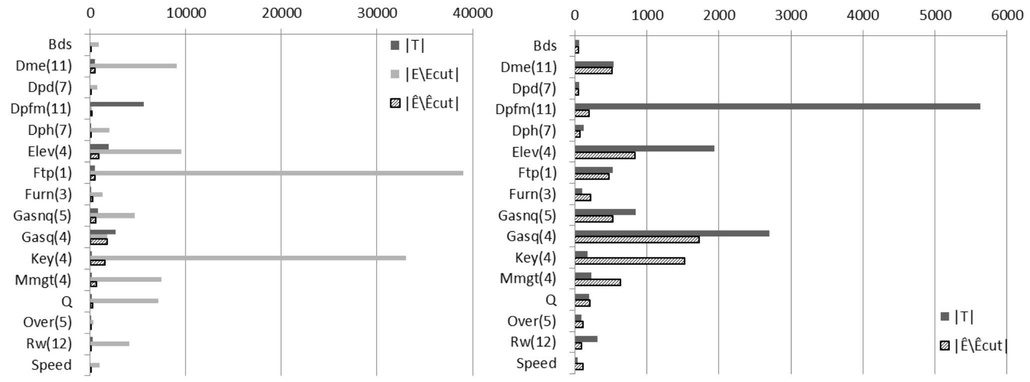

Figure 4: The first chart compares the numbers of transitions in the original PN, events in its complete unfolding prefix, and mp-events in the corresponding MP; the size of unfolding dominates in all but few benchmarks (viz. those with a large number of dead transitions, where the size of PN dominates). The second chart is obtained from the first one by removing the unfolding prefixes data and re-scaling; on all these benchmarks the size of the MP is either comparable with the size of PN or even much smaller on benchmarks with many dead transitions.

events and arcs, without affecting the number of conditions. Hence, $|\mathfrak{Merge}(\beta)| \leq |\beta|$, i.e. MPs are no less condense than branching processes (but can be exponentially smaller, see the examples in Sect. 1). This allows one to import all the upper bounds proved for unfolding prefixes [2, 10]. In particular, since for every safe PN 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 MP has the same upper bound on the number of its mp-events.

However, the upper bound given by the size of the unfolding prefix is rather pessimistic; in practice, MPs turn out to be much more compact than the corresponding unfolding prefixes. Fig. 4 compares the numbers of transitions in the original PNs, the numbers of non-cut-off events in their complete unfolding prefixes, and the numbers of non-cut-off mp-events in the corresponding MPs on some of the benchmarks collected by J.C. Corbett [17]. The chart on the left shows that MPs are usually much smaller than unfoldings. The picture on the right does not give the sizes of the unfolding prefixes in order to improve the scale; it shows that in most cases the sizes of MPs are close to those of the original PNs (and even smaller in some cases due to the presence of dead nodes in the original PNs).

Since MPs are inherently more compact than unfolding prefixes, it would be natural to seek sharper upper bounds than the trivial one given by the size of the unfolding prefix. In particular, [5] identifies two subclasses of PNs whose smallest complete unfolding prefixes can be exponential in the size of the original PN, but whose complete MPs are only polynomial:

- Acyclic PNs (the unravelling of such a net Σ coincides with the net obtained from Σ by removing its dead transitions and unreachable places).

- Live and safe free-choice PNs with transitions' postsets of bounded size (the degree of the polynomial depends on the bound on transitions' postsets sizes). Note that the expressive power of this class of PNs is comparable with that of the full class of live and safe free-choice PNs, since every transition with a large postset can be replaced by a tree of transitions with postsets of bounded size, so that the behaviour of the PN is preserved.

2.6. Model checking based on merged processes

We briefly recall the main ideas of the reduction of a reachability-like property R to SAT using MPs (see [5] for more details). Given a marking-complete MP μ of a safe PN, we associate with each non-cut-off mp-event $\widehat{e}$ of μ a Boolean variable $\mathsf{conf}_{\widehat{e}}$. Hence, every assignment A to these variables corresponds to the set $\widehat{C} \overset{\mathrm{df}}{=} \{\widehat{e} \mid A(\mathsf{conf}_{\widehat{e}}) = 1\}$ of mp-events of μ. The SAT instance has the form $\mathcal{CONF} \wedge \mathcal{VIOL}$, where the role of the *configuration constraint, $\mathcal{CONF}$*, is to ensure that $\widehat{C}$ is an mp-configuration of μ (not just an arbitrary set of mp-events), and the role of the *violation constraint, $\mathcal{VIOL}$*, is to express that the property R holds for the marking $Mark(\widehat{C})$.

$\mathcal{CONF}$ has the form $\mathcal{ME} \wedge \mathcal{ACYCLIC} \wedge \mathcal{NG}$, where $\mathcal{ME}$ expresses that $\widehat{C}$ is a solution of the marking equation [7] (i.e. the mp-events in $\widehat{C}$ provide a valid token flow), $\mathcal{ACYCLIC}$ conveys that the digraph induced by $\widehat{M_\mu} \cup \widehat{C} \cup \widehat{C}^\bullet$ is acyclic, and $\mathcal{NG}$ (no-gap) expresses that the mp-conditions labelled by the same place p are visited in an order consistent with their occurrence-depths, without gaps.

To construct $\mathcal{VIOL}$ from R, one can first build a Boolean formula 'computing' $Mark(\widehat{C})$, i.e. relating the variables conf_* with new variables mark_p, for each place p of the PN, tracing whether $p \in Mark(\widehat{C})$. Intuitively, $\mathsf{mark}_p = 1$ iff some mp-condition $\widehat{c}$ labelled by p is in $Cut(\widehat{C})$, i.e. $\mathsf{conf}_{\widehat{e}} = 1$ for some $\widehat{e} \in {}^\bullet\widehat{c}$ and $\mathsf{conf}_{\widehat{e}} = 0$ for all $\widehat{e} \in \widehat{c}^\bullet$. Then one can simply substitute all the references to places in R by the corresponding variables mark_*, which yields the $\mathcal{VIOL}$ constraint.

The existence of a reduction to SAT means that MPs are much more amenable to model checking than general safe PNs — e.g. most of 'interesting' behaviourial properties are known to be PSPACE-complete for safe PNs [18], whereas checking reachability-like properties on a marking-complete MP is only NP-complete. Since many such properties are known to be NP-complete already for unfolding prefixes, the theoretical complexity is not worsened if one uses MPs instead of unfolding prefixes.

On the practical side, the SAT instances for MPs are more complicated than those for unfolding prefixes. However, as MPs are usually much smaller than unfolding prefixes, the model checking runtimes are quite similar according to the experiments in [5]. Since space considerations are of utmost importance in model checking, these results can be regarded as positive. Furthermore, the encoding used in [5] for the $\mathcal{ACYCLIC}$ constraint was rather inefficient, and there is a chance that it can be significantly improved, speeding up model checking.

3. Unravelling algorithm

In this section we present the main contribution of this paper, viz. an algorithm for constructing a complete MP of a safe PN, that avoids building a complete unfolding prefix, see Fig. 5. We also formally prove the correctness of this unravelling algorithm.

The unravelling algorithm constructs the MP by starting from the initial mp-conditions (they exactly correspond to the initially marked places of the PN), and repeatedly adding *possible extensions* to it, until a certain termination condition is fulfilled.

Definition 2 (Possible extension). An mp-event $\widehat{e}$ of μ_Σ is a *possible extension* of an MP μ of Σ if $\widehat{e}$ is not in μ and adding $\widehat{e}$ (together with the mp-conditions in $\widehat{e}^\bullet$ that are not in μ yet) to μ results in an MP of Σ.

```
input : $\Sigma$ — a safe PN
output : $\mu$ — a marking-complete or complete MP of $\Sigma$

$\mu \leftarrow$ the branching process comprised of the initial mp-conditions
$conf\_sz \leftarrow 0$ /* current configuration size */
repeat
    $conf\_sz \leftarrow conf\_sz + 1$
    $pe \leftarrow$ {possible extensions of $\mu$} /* SAT */
    $cand \leftarrow \{\widehat{e} \in pe \mid \widehat{e}$ has a local configuration of size $conf\_sz$ in $\mu\}$ /* SAT */
    /* filter out potential cut-offs */
    $slice \leftarrow \{\widehat{e} \in cand \mid \neg\textsc{MaybeCutOff}(\mu \oplus cand, \widehat{e}, conf\_sz)\}$
    $\mu \leftarrow \mu \oplus slice$
    /* Invariant: $\mu = \mathfrak{Merge}(\beta_{\lceil conf\_sz \rceil})$ */
until $slice = \emptyset \wedge$
    $\neg\exists\widehat{e} \in pe : \widehat{e}$ has a local mp-configuration of size $> conf\_sz$ in $\mu \oplus pe$ /* SAT */
/* optionally assign $\mu \leftarrow \mu \oplus pe$ and mark the mp-events in $pe$ as cut-offs */

/* Check if each local mp-configuration of $\widehat{e}$ of size $conf\_sz$ in $\overline{\mu}$ contains a cut-off */
$\textsc{MaybeCutOff}(\overline{\mu}, \widehat{e}, conf\_sz) \equiv$
    /* 2QBF */
    $\forall$ local mp-configurations $\widehat{C}$ of $\widehat{e}$ in $\overline{\mu}$ such that $|\widehat{C}| = conf\_sz$:
        $\exists\widehat{f} \in \widehat{C} : \exists$ mp-configuration $\widehat{C'}$ in $\overline{\mu} : Mark([\widehat{f}]_{\widehat{C}}) = Mark(\widehat{C'}) \wedge [\widehat{f}]_{\widehat{C}} \lhd \widehat{C'}$
```

Figure 5: An unravelling algorithm that avoids constructing a complete unfolding prefix. It assumes that the adequate order $\lhd$ refines the size order.

One can compute the set of possible extensions of μ as follows. For each transition t of Σ, one looks for an mp-configuration $\widehat{C}$ of μ which can be extended by an instance $\widehat{e}$ of t that is not in μ yet, such that $\widehat{C} \cup \{\widehat{e}\}$ is an mp-configuration of μ_Σ. This can be formulated as a model checking problem and reduced to SAT as explained in Sect. 2.6. Once a possible extension is computed, a constraint preventing its re-computation is added to the SAT instance, and the problem is solved again, until it becomes unsatisfiable. The process terminates when all possible extensions corresponding to every transition t are computed. Note that the SAT instances to be solved are very similar, and incremental SAT can be used to optimise the whole process. Given an MP μ and a set S of possible extensions of μ, we denote by $\mu \oplus S$ the MP obtained by adding the mp-events in S to μ, together with the mp-conditions in their postsets that are not in μ yet.

The cut-off check is implemented using the MaybeCutOff predicate. It naturally translates to a 2QBF instance, and a 2QBF solver is used to check if it holds. Note that until the algorithm terminates, it is not possible to designate an mp-event $\widehat{e}$ as a cut-off, as $\widehat{e}$ can have many local mp-configurations not all of which are known yet (as adding other mp-events to μ can result in $\widehat{e}$ acquiring new local mp-configurations). However, all the local mp-configurations of $\widehat{e}$ of size up to $conf\_sz$ are guaranteed to be in μ (except ones containing a cut-off), and if the adequate order[2]

[2]Since for a safe PN Σ configurations of β_Σ are isomorphic to the corresponding mp-configurations of μ_Σ, the adequate orders used for truncating the unfolding, see e.g. [2], can be re-used in the context of MPs.

$\lhd$ refines the size order, it is guaranteed that if MAYBECUTOFF does not hold for $\widehat{e}$ (and so $\widehat{e}$ is added to μ), it will never hold for $\widehat{e}$ in future, i.e. an added mp-event will not suddenly become cut-off after further mp-events are added to μ.

To summarise, the algorithm adds possible extensions to the MP being constructed in rounds, where in round k a possible extension is considered for addition if it has at least one local configuration of size k that contains no cut-offs. In this way, in k th round the algorithm can already perform a one-sided check of the MAYBECUTOFF condition for local configurations of size k, and if it does not hold then it *definitely* knows that the mp-event cannot be a cut-off[3] and so can be added to the MP. Hence the same mp-event can be considered for addition several times (in different rounds), and each time more of its local configurations are taken into account. Eventually the termination criterion becomes satisfied, which ensures that all the possible extensions remaining in *pe* can be declared cut-off, and the algorithm stops.

In general, if β is a complete branching prefix, $\mathfrak{Merge}(\beta)$ can still be incomplete (only marking-completeness is guaranteed). In contrast, the proposed algorithm can construct a complete MP (i.e. preserve firings in addition to marking-completeness). This is achieved by adding the possible extensions that are still in *pe* to μ and marking them as cut-offs in the optional operator following the main loop. This incurs almost no overhead, but allows one to employ more efficient model checking algorithms, in particular for deadlock detection.

3.1. Correctness of the algorithm

We make two important assumptions about the parameters of the unravelling algorithm, that are essential for the correctness proofs below:

A1. the adequate order $\lhd$ refines the size order on mp-configurations, i.e. $|\widehat{C}| < |\widehat{C'}|$ implies $\widehat{C} \lhd \widehat{C'}$;

A2. any configuration (not only a local one) can be used as a correspondent of a cut-off event.

We will denote by $\beta^{\lhd}$ the unfolding prefix obtained by removing the cut-off events from the canonical unfolding prefix of Σ that was built using $\lhd$ as the adequate order, with all configurations (not only the local ones, as often the case in unfoldings) being allowed as cut-off correspondents. Furthermore, $\beta^{\lhd}_{\lceil n \rceil}$ will denote the prefix obtained from $\beta^{\lhd}$ by removing all the events whose local configurations contain more than n events.

The Lemma below is needed to show that if an event f can be declared cut-off due to some corresponding configuration in the full unfolding, then it has a corresponding configuration already in $\beta^{\lhd}_{\lceil |[f]| \rceil}$; in particular, this means that $\beta^{\lhd}_{\lceil n \rceil}$ is a prefix of $\beta^{\lhd}_{\lceil n+1 \rceil}$, i.e. an event e cannot disappear from the prefix due to some $f \in [e]$ being a 'delayed' cut-off (i.e. suddenly becoming a cut-off as n increases). This result relies on Assumption A1 above; moreover, though the lemma itself does not depend on Assumption A2 (as it does not explicitly mention cut-off events at all), this assumption is needed to be able to declare f a cut-off event of $\beta^{\lhd}_{\lceil |[f]| \rceil}$ due to the corresponding configuration C', which can be non-local even if f has a local cut-off correspondent C in a larger prefix (here f, C and C' are as in the statement of the lemma).

Lemma 3 (No delayed cut-offs). *Let f be an event of β_Σ and C be a configuration of β_Σ such that $C \lhd [f]$ and $Mark(C) = Mark([f])$, where $\lhd$ refines the size order. Then $\beta^{\lhd}_{\lceil |[f]| \rceil}$ contains a configuration C' such that $C' \lhd [f]$ and $Mark(C') = Mark([f])$.*

[3]The opposite is not true, i.e. if MAYBECUTOFF predicate holds for some mp-event, it cannot be immediately declared cut-off, as it might acquire new mp-configurations in future.

Proof. Let C' be a minimal w.r.t. $\lhd$ configuration satisfying $C' \lhd [f]$ and $Mark(C') = Mark([f])$. Such a configuration must exist, as the set of configurations satisfying this condition is non-empty (as it contains C) and $\lhd$ is well-founded by definition of an adequate order. C' cannot contain a cut-off event, as otherwise there would be a smaller w.r.t. $\lhd$ configuration satisfying this condition (cf. the completeness proof in [2]), contradicting the choice of C'. Moreover, as $\lhd$ refines the size order, $|C'| \leq \|[f]\|$, and so C' is a configuration of $\beta^{\lhd}_{\lceil \|[f]\| \rceil}$. $\square$

The result below proves the crucial invariant of the unravelling algorithm.

Lemma 4 (Invariant). *After each execution of the body of the main loop the invariant* $\mu = \mathfrak{Merge}(\beta^{\lhd}_{\lceil conf\_sz \rceil})$ *holds.*

Proof. The invariant trivially holds before the loop is executed first time, so it remains to show that it is preserved by executing the body of the loop.

First we show that every mp-event $\widehat{e}$ of $\mathfrak{Merge}(\beta^{\lhd}_{\lceil conf\_sz \rceil})$ that has a local mp-configuration of size $conf\_sz$ will be in μ after the loop's body is executed, i.e. for every event e of $\beta^{\lhd}_{\lceil conf\_sz \rceil}$ with $\|[e]\| = conf\_sz$ (after $conf\_sz$ has been incremented), $\widehat{e} \overset{\text{df}}{=} \hbar(e)$ will be in μ.

If $\widehat{e} \overset{\text{df}}{=} \hbar(e)$ is in μ (e.g. because it has a local configuration of size smaller than $conf\_sz$) then the desired conclusion vacuously holds by the induction hypothesis. Hence it only remains to consider the case when $\widehat{e}$ is not in μ. By the induction hypothesis all the mp-events in $\hbar([e] \setminus \{e\})$ are in μ, and so $\widehat{e} \in pe$. The configuration $[e]$ of β_Σ and $\widehat{C} \overset{\text{df}}{=} \hbar([e])$ of μ_Σ are isomorphic due to Σ being a safe PN, in particular $\widehat{C}$ is a local configuration of $\widehat{e}$ in μ_Σ and $|\widehat{C}| = conf\_sz$. Hence, $\widehat{e} \in cand$ holds.

Since e is in $\beta^{\lhd}_{\lceil conf\_sz \rceil}$, $[e]$ contains no cut-off events, and so the MaybeCutOff condition cannot hold for $\widehat{e}$. Indeed, in the opposite case one can find a configuration C that is a cut-off correspondent of some $f \in [e]$ in β_Σ, and Lemma 3 guarantees the existence of a cut-off correspondent C' of f already in $\beta^{\lhd}_{\lceil [f] \rceil}$; since $\|[f]\| \leq \|[e]\| = conf\_sz$, C' is a configuration of $\beta^{\lhd}_{\lceil conf\_sz \rceil}$, which contradicts e being in $\beta^{\lhd}_{\lceil conf\_sz \rceil}$. Therefore, $\widehat{e} \in slice$, and so $\widehat{e}$ is added to μ.

Now we show that if an mp-event $\widehat{e}$ is added to μ in the loop's body then $\widehat{e}$ is also in $\mathfrak{Merge}(\beta^{\lhd}_{\lceil conf\_sz \rceil})$, i.e. there is an event e in $\beta^{\lhd}_{\lceil conf\_sz \rceil}$ such that $\widehat{e} = \hbar(e)$. Since $\widehat{e}$ is added in the loops body, $\widehat{e}$ is a possible extension of μ that has a local mp-configuration $\widehat{C}$ of size $conf\_sz$ (after $conf\_sz$ has been updated in the loop's body) that does not contain a cut-off mp-event in $\mu \oplus cand$. There is a unique configuration C of size $conf\_sz$ in β_Σ such that $\widehat{C} = \hbar(C)$ due to Σ being a safe PN, which is local due to $\widehat{C}$ being local, i.e. $C = [e]$ for some event e. For the sake of contradiction, suppose that e is not in $\beta^{\lhd}_{\lceil conf\_sz \rceil}$. Then there is an event $f \in [e]$ that is cut-off due to some configuration C_f of $\beta^{\lhd}_{\lceil conf\_sz \rceil}$. Since $\lhd$ refines the size ordering and $C_f \lhd [f] \subseteq [e]$, $|C_f| \leq \|[f]\| \leq \|[e]\| = conf\_sz$, i.e. $|C_f| \leq conf\_sz$. By the induction hypothesis, the $\hbar$-image of any strict sub-configuration of C_f is in μ, and so $\hbar(C_f)$ is an mp-configuration of $\mu \oplus cand$. Hence, $\hbar(f) \in \widehat{C}$ would be a cut-off event due to $\hbar(C_f)$, a contradiction. $\square$

The soundness of the algorithm trivially follows from the invariant of Lemma 4, and so does not require a separate proof.

Proposition 5 (Soundness). *If the algorithm adds an mp-event* $\widehat{e}$ *to the merged process being constructed then there is an event* e *in* $\beta^{\lhd}$ *such that* $\hbar(e) = \widehat{e}$.

Proof. Trivially follows from Lemma 4. $\square$

The following results state that the generated merged process is marking-complete (and even complete if the optional part of the algorithm is performed), and that the algorithm terminates.

Proposition 6 (Completeness). *Upon termination of the algorithm the resulting merged process μ is marking-complete. Moreover, it is complete if the optional statement after the main loop is executed.*

Proof. First, we show that upon termination of the main loop, μ is marking-complete. Since the invariant of Lemma 4 holds upon termination of the loop and due to marking-completeness of $\mathfrak{Merge}(\beta^{\lhd})$ [5], it only remains to show that the main loop does not terminate prematurely, i.e. its exit condition is not satisfied before μ becomes $\mathfrak{Merge}(\beta^{\lhd})$. For the latter to hold, $\beta^{\lhd}$ must contain an event e such that $\widehat{e} \overset{\mathrm{df}}{=} \hbar(e)$ is not in μ, and w.l.o.g., we can assume that e is a causally minimal such event. Hence, due to the invariant of Lemma 4, e is a possible extension of $\beta^{\lhd}_{\lceil conf\_sz \rceil}$, and so, since $\widehat{e}$ is not in μ yet, $\widehat{e}$ is a possible extension of μ, i.e. $\widehat{e} \in pe$. Moreover, as $|[e]| > conf\_sz$ due to e being a possible extension of $\beta^{\lhd}_{\lceil conf\_sz \rceil}$, and all the mp-events in $\hbar([e] \setminus \{e\})$ being in μ, $\widehat{e}$ has a local mp-configuration $\widehat{C}$ in $\mu \oplus pe$ satisfying $|\widehat{C}| > conf\_sz$, and so the loop's termination condition does not hold.

Therefore, upon termination of the algorithm μ is marking-complete, as the optional assignment $\mu \leftarrow \mu \oplus pe$, if present, will not eliminate any reachable markings from μ and thus will not affect its marking-completeness. Moreover, this optional assignment will turn a marking-complete MP μ into a complete process. Indeed, $slice = \emptyset$ upon termination of the loop, and so no mp-events have been added to μ on the last iteration of the loop, which means that pe contains exactly the possible extensions of μ, i.e. the firings are preserved. □

Proposition 7 (Termination). *The algorithm terminates.*

Proof. For the sake of contradiction, suppose the algorithm does not terminate, i.e. the loop's exit condition is never satisfied. Since $\beta^{\lhd}$ is a finite prefix, $\beta^{\lhd} = \beta^{\lhd}_{\lceil n \rceil}$ for some $n \in \mathbb{N}$. Since $conf\_sz$ is incremented every time the body of the loop is executed, $\mu = \mathfrak{Merge}(\beta^{\lhd})$ when $conf\_sz$ reaches the value n due to the invariant of Lemma 4. Hence for all the iterations when $conf\_sz > n$, $slice = \emptyset$ and μ and pe do not change. Since $\mu \oplus pe$ is finite, it has only finitely many mp-configurations (as the set of its mp-events has only finitely many subsets), and we define

$$n' \overset{\mathrm{df}}{=} \max\{|\widehat{C}| \mid \widehat{e} \in pe \text{ and } \widehat{C} \text{ is a local mp-configuration of } \widehat{e}\}.$$

When $conf\_sz$ reaches the value n', the loop's exit condition becomes satisfied, a contradiction. □

3.2. Total adequate order

One of the important breakthroughs in the unfolding theory was the development of a total adequate order [2]. Unfortunately, the order developed in [2] has a very inefficient Boolean encoding, which is detrimental for the proposed unravelling algorithm, as it is based on SAT and 2QBF.

Fortunately, other adequate orders exist, in particular the order $\prec^d_{sl}$ order [19] has a relatively simple Boolean encoding. However, $\prec^d_{sl}$ does not refine the size order, violating Assumption A1, which is essential for the proof of correctness of the proposed unravelling algorithm to proceed.

Below we prove the result that allowed us to adapt the $\prec^d_{sl}$ order. Besides our application, this result is of independent interest for the unfolding theory. Essentially, it says that any adequate order can be turned into one refining the size order by prefixing it with the size comparison; moreover, the totality of the order is preserved.

Proposition 8 (Turning any adequate order onto one refining the size order). *Let $\lhd$ be an adequate order. Then the relation $\lhd'$ defined by*

$$C \lhd' C' \iff |C| < |C'| \lor (|C| = |C'| \land C \lhd C')$$

is an adequate order refining the size order. Moreover, $\lhd'$ is total if $\lhd$ is total.

Proof. It is trivial to show that $\lhd'$ is a well-founded strict partial order refining the size order (and hence the set inclusion), and that $\lhd'$ is total whenever $\lhd$ is total. Hence, it remains to show that $\lhd'$ is preserved by finite extensions, i.e. the following condition holds:

> If $C \lhd' C'$ then $C \oplus I(E) \lhd' C' \oplus E$ for any configurations C and C' with the same final marking and for any finite extension E of C'; recall that $I(E)$ denotes an extension of C that is isomorphic to E.

In what follows, we take any configurations C and C' with the same final marking and satisfying $C \lhd' C'$, and consider the two possible cases:

(i) Suppose $|C| < |C'|$. Then $|C + I(E)| = |C| + |I(E)| < |C'| + |E| = |C' + E|$, as $|I(E)| = |E|$, and so $C \oplus I(E) \lhd' C' \oplus E$.

(ii) Suppose $|C| = |C'|$ and $C \lhd C'$. Then $|C + I(E)| = |C' + E|$ due to $|I(E)| = |E|$, and $C + I(E) \lhd C' + E$ as $\lhd$ is preserved by finite extensions due to being an adequate order. Hence $C \oplus I(E) \lhd' C' \oplus E$.

In either case $C \oplus I(E) \lhd' C' \oplus E$ holds, and so $\lhd'$ is preserved by finite extensions, i.e. it is an adequate order. □

3.3. Optimisations

The algorithm in Fig. 5 is formulated in such a way that it would be easier to prove its correctness. In practice, however, a number of optimisations can be introduced to improve the performance.

Computing possible extensions

The algorithm does not have to re-compute the set of possible extensions from scratch on every iteration of the main loop. It is sufficient to update the set *pe* left from the previous iteration, by removing the events in *slice* from it (as they have been added to μ) and adding only those possible extensions that are not in μ or *pe* yet.

Cut-off check

If for a possible extension $\widehat{e}$, there is an mp-condition $\widehat{c} \in \widehat{e}^\bullet$ such that there is no mp-condition labelled by $\widehat{h(c)}$ in $\mu \oplus (cand \setminus \{\widehat{e}\})$, the predicate MaybeCutOff($\mu \oplus cand, \widehat{e}, conf\_sz$) cannot hold, and so this check becomes unnecessary.

Counterexamples in the 2QBF solver

If the 2QBF solver is implemented by a pair of SAT solvers [11], one can reduce the number of communication rounds by ensuring that the counterexamples generated by the auxiliary solver are mp-configurations of size *conf_sz*. This can be achieved by initialising the auxiliary solver with the formula expressing this condition, rather than with an empty formula.

Exploiting unate constraints

One can observe that the $\mathcal{ACYCLIC}$ predicate is negative unate; moreover, the adequate order $\lhd$, due to being a refinement of the size order, is negative unate in its first parameter and positive unate in its second parameter. This allows to optimise the corresponding Boolean encodings, see [20, Sect. 5.1].

4. Experimental results

We have evaluated the developed unravelling algorithm on a popular set of benchmarks collected mainly by J.C. Corbett [17]. The same benchmarks were used in [8, 9]; however, there are some important differences in our setup, which resulted in different figures in the tables. Moreover, the runtime of the unravelling algorithm has been very significantly improved compared to [8, 9].

The first major difference is that the new total adequate order (see Sect. 3.2) was used for producing cut-offs. For consistency, the same order was used to produce the unfolding prefixes. The second important difference is that the unfolding algorithm was re-implemented to use any configuration in the built part of the prefix as a potential cut-off correspondent, unlike the standard unfolding technique that uses only local ones. This was done to ensure the consistency with the unravelling algorithm, which, due to a subtle point in its correctness proof, currently does not allow one to restrict the cut-off check to local configurations; moreover, it is not clear that this would result in efficiency gains anyway. However, this differs from the conventional unfolding technique, where restricting the cut-off check to local configuration reduces its complexity (at the price of generating larger prefixes). The issue of using local vs. all configurations as cut-off correspondents in the context of unfoldings was investigated in [21]; the main conclusion was that by using all rather than local configurations, smaller prefixes can be produced, but at the expense of increase in runtime; hence, using all configurations usually does not pay off.

Due to the consistency between the unfolding and unravelling algorithms in the setup outlined above, it was possible to verify the correctness of the constructed MPs. Indeed, due to Lemma 4, the built MP should be equal to the one obtained by applying the $\mathfrak{Merge}$ operation to the corresponding unfolding prefix, which is easy to check.

Table 1 shows the results of our experiments, which were conducted on a PC with an Intel i7 2.8GHz CPU, 4GB RAM and 64-bit Windows 7. Both the unfolder and the unraveller were compiled as 32-bit applications, and no parallelism was used, i.e. only one CPU core was utilised. The unfolding prefixes in our experiments were built using the PUNF unfolder [22], and the MINISAT [23] SAT solver was employed by the proposed unravelling algorithm (the 2QBF solver was implemented using a pair of communicating SAT solvers [11]). 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 PN; the number of conditions and events the unfolding prefix stripped of cut-off events, together with the time to construct it; the number of mp-conditions and mp-events in the corresponding MP, together with the time to construct it.

When interpreting the results, one should keep in mind the following. MPs are often more compact than unfoldings, but producing an mp-event in an MP is much more computationally expensive than producing an event in an unfolding prefix, as this usually involves solving one or more 2QBF problems. Hence, if the benchmark is such that its unfolding is small, the unravelling algorithm cannot improve the running time, i.e. the advantages of MPs can only be achieved on complicated benchmarks where unfoldings do not perform well.

Benchmark	Net		Unfolding prefix			Merged process		
	$\lvert P\rvert$	$\lvert T\rvert$	$\lvert B\rvert$	$\lvert E \setminus E_{cut}\rvert$	t[s]	$\lvert\widehat{B}\rvert$	$\lvert\widehat{E} \setminus \widehat{E}_{cut}\rvert$	t[s]
Abp	43	95	221	109	<1	63	56	<1
Bds	53	59	1907	956	<1	78	50	<1
Byz	504	409	40564	13972	24	650	312	14
Dartes-r	331	257	—	—	timeout	535	231	221
Ftp	176	529	78026	39011	258	259	478	166
Q	163	194	14043	7180	7	238	212	6
Speed	33	39	1830	1037	<1	86	118	1
Cyclic(6)	47	35	98	43	<1	83	38	<1
Cyclic(9)	71	53	152	67	<1	128	59	<1
Cyclic(12)	95	71	206	91	<1	173	80	<1
Dac(9)	63	52	79	51	<1	63	44	<1
Dac(12)	84	70	106	69	<1	84	59	<1
Dac(15)	105	88	133	87	<1	105	74	<1
Dp(8)	48	32	256	120	<1	64	32	<1
Dp(10)	60	40	400	190	<1	80	40	<1
Dp(12)	72	48	576	276	<1	96	48	<1
Dpd(5)	45	45	392	195	<1	59	38	<1
Dpd(6)	54	54	766	382	<1	71	46	<1
Dpd(7)	63	63	1484	741	<1	83	54	<1
Dpfm(5)	27	41	27	11	<1	22	11	<1
Dpfm(8)	87	321	102	47	<1	63	47	<1
Dpfm(11)	1047	5633	409	199	5	222	199	4
Dph(5)	48	67	790	390	<1	61	50	<1
Dph(6)	57	92	1862	925	<1	73	63	1
Dph(7)	66	121	4142	2064	1	85	77	4
Elev(2)	146	299	882	477	<1	126	135	1
Elev(3)	327	783	4185	2241	<1	264	346	9
Elev(4)	736	1939	18008	9567	7	557	841	133
Furn(1)	27	37	146	75	<1	47	34	<1
Furn(2)	40	65	662	344	<1	80	99	1
Furn(3)	53	99	2587	1345	1	113	226	12
Gasnq(3)	143	223	619	310	<1	147	151	1
Gasnq(4)	258	465	2440	1221	<1	262	301	3
Gasnq(5)	428	841	9301	4652	3	432	530	17
Gasq(2)	78	97	134	67	<1	84	63	<1
Gasq(3)	284	475	641	321	<1	292	301	1
Gasq(4)	1428	2705	3656	1829	<1	1438	1725	35
Hart(50)	252	152	352	201	<1	302	201	1
Hart(75)	377	227	527	301	<1	452	301	3
Hart(100)	502	302	702	401	<1	602	401	6
Key(2)	94	92	828	412	<1	137	262	4
Key(3)	129	133	7419	3707	4	183	768	1027
Key(4)	164	174	66020	33007	241	229	1526	timeout
Mmgt(2)	86	114	502	250	<1	99	155	1
Mmgt(3)	122	172	2849	1424	<1	141	355	6
Mmgt(4)	158	232	14900	7450	9	183	638	91
Over(3)	52	53	128	62	<1	64	48	<1
Over(4)	71	74	305	150	<1	89	80	<1
Over(5)	90	95	718	356	<1	114	118	1
Ring(5)	65	55	181	88	<1	81	55	<1
Ring(7)	91	77	437	215	<1	119	85	1
Ring(9)	117	99	869	430	<1	157	115	1
Rw(6)	33	85	152	70	<1	33	28	<1
Rw(9)	48	181	1060	521	<1	48	55	10
Rw(12)	63	313	8240	4108	<1	63	91	3107
Sent(50)	179	80	323	159	<1	188	86	<1
Sent(75)	254	105	398	184	<1	263	111	<1
Sent(100)	329	130	473	209	<1	338	136	<1

Table 1: Experimental results.

As one can see, overall the unravelling algorithm still loses to the unfolder, though there are some successes like Byz, Dartes-r and Ftp. These results should be considered in the light of the fact that our implementation of the unravelling algorithm is still in its early days, and it is compared with a fully-fledged unfolder employing some sophisticated techniques that took over a decade to develop. If viewed that way, the results do look very promising: we are confident that our implementation can be considerably improved — in fact, it has already been very considerably improved compared to the one used in the conference version of this paper [8]. We believe that the optimisations outlined in Sect. 5 have the potential to further increase the performance of the proposed algorithm. In particular, currently there are some anomalies in its behaviour, e.g. on the Rw(12) benchmark the algorithm has a rather high runtime, even though the resulting MP is quite small, which indicates the low quality of our home-brewed 2QBF solver used by the unravelling algorithm.

5. Conclusions and future work

In this paper we have proposed an algorithm for constructing complete MPs of safe PNs. This algorithm avoids building an intermediate complete unfolding prefix, which is usually exponentially larger than the final result, overcoming thus the main disadvantage of the previous approach [5]. In particular, the challenging problem of identifying cut-offs in MPs has been solved.

The proposed unravelling algorithm has been evaluated on a number of benchmarks. Though overall it performs worse than an unfolder algorithms, it was very successful on some benchmarks, and generally quite competitive. We consider the results as encouraging, and believe they can be significantly improved. In particular, the following obvious steps can be done:

- Our implementation of the 2QBF solver is very basic, and its performance occasionally deteriorates even on very small benchmarks, e.g. Rw(12). Using a better solver (e.g. implementing non-critical signal reasoning, which can increase the performance by an order of magnitude according to experiments in [11]) is likely to improve the runtime dramatically.
- The proposed algorithm can be easily parallelised, as typically there are several SAT or 2QBF instances that can be solved concurrently. This opportunity is very useful, as contemporary PCs are multi-core. Moreover, the computation can be easily distributed over a network of PCs.

The above improvements seem fairly obvious and just require implementation effort. There are also some research areas, progress in which would improve the proposed algorithm:

- The implemented encoding of the $\mathcal{ACYCLIC}$ constraint [5] seems very inefficient; improving it will have a direct effect on both the unravelling algorithm and the subsequent model checking.
- The proposed algorithm works by considering configurations of larger and larger sizes (cf. the *conf_sz* variable). Hence, each iteration of the main loop repeats much of the work done in the previous ones, and so this might be not the best strategy — perhaps this repeated work can be eliminated or at least reduced. Alternatively, the experience from bounded model checking might be useful to alleviate this problem.

- The last few iterations of the algorithm are often free-running, i.e. no mp-events are added to the MP being built, as the termination criterion is not sharp. These iterations are quite expensive (as the MP has already reached its full size), and so much time is wasted. Hence, developing a sharper termination criterion is likely to significantly improve the runtime.

On a more general note, we believe that the 2QBF based strategy used by the unravelling algorithm for identifying cut-offs is fairly general and can be employed elsewhere. Indeed, it is quite common for an (mp-)event to have multiple local configurations, e.g. this is the case for unfoldings of PNs with read arcs [24, 25], symbolic unfoldings [26] and unfoldings of time nets [27].

Finally, the natural question about generalisation of the proposed algorithm from safe to bounded PNs is more complicated then one might think, for the following reasons. First of all, the mp-configurations become multisets, and so one would have to use Integer Linear Programming (ILP) instead of SAT, and formulate (and develop an efficient algorithm for) an integer analog of 2QBF. Moreover, the relationship between the configurations of the unfolding and mp-configurations of the unravelling is somewhat more complicated, in particular the isomorphism is lost. Furthermore, no easily checkable characterisation of mp-configurations of merged processes of unsafe PNs has been developed yet, which is probably the most serious obstacle (see [5] where this question is expounded, along with some ideas about developing such a characterisation).

Acknowledgements

The authors would like to thank Javier Esparza and Keijo Heljanko for helpful discussions.

References

[1] A. Valmari, Lectures on Petri Nets I: Basic Models, Vol. 1491 of LNCS, Springer-Verlag, 1998, Ch. The State Explosion Problem, pp. 429–528.

[2] J. Esparza, S. Römer, W. Vogler, An improvement of McMillan's unfolding algorithm, FMSD 20 (3) (2002) 285–310.

[3] V. Khomenko, Model checking based on prefixes of Petri net unfoldings, Ph.D. thesis, School of Comp. Sci., Newcastle Univ. (2003).

[4] K. McMillan, Using unfoldings to avoid state explosion problem in the verification of asynchronous circuits, in: Proc. CAV'92, Vol. 663 of LNCS, Springer-Verlag, 1992, pp. 164–174.

[5] V. Khomenko, A. Kondratyev, M. Koutny, W. Vogler, Merged processes — a new condensed representation of Petri net behaviour, Acta Inf. 43 (5) (2006) 307–330.

[6] S. Melzer, S. Römer, Deadlock checking using net unfoldings, in: Proc. CAV'97, Vol. 1254 of LNCS, Springer-Verlag, 1997, pp. 352–363.

[7] T. Murata, Petri nets: Properties, analysis and applications, Proc. of the IEEE 77 (4) (1989) 541–580.

[8] V. Khomenko, A. Mokhov, An algorithm for direct construction of complete merged processes, in: Proc. Petri Nets'11, Vol. 6709 of LNCS, Springer-Verlag, 2011, pp. 89–108.

[9] V. Khomenko, A. Mokhov, An algorithm for direct construction of complete merged processes, Tech. Rep. CS-TR-1231, School of Comp. Sci., Newcastle Univ. (2011).
URL `http://www.cs.ncl.ac.uk/publications/trs/papers/1231.pdf`

[10] V. Khomenko, M. Koutny, V. Vogler, Canonical prefixes of Petri net unfoldings, Acta Inf. 40 (2) (2003) 95–118.

[11] D. Ranjan, D. Tang, S. Malik, A comparative study of 2QBF algorithms, in: Proc. SAT'04, ACM, 2004, pp. 323–328.

[12] L. Stockmeyer, The polynomial-time hierarchy, Theor. Comp. Sci. 3 (1) (1976) 1–22.

[13] L. Zhang, S. Malik, The quest for efficient Boolean satisfiability solvers, in: Proc. CAV'02, Vol. 2404 of LNCS, Springer-Verlag, 2002, pp. 582–595.

[14] J. Engelfriet, Branching processes of Petri nets, Acta Inf. 28 (1991) 575–591.

[15] K. McMillan, Symbolic model checking: an approach to the state explosion problem, Ph.D. thesis, School of Comp. Sci., Carnegie Mellon Univ. (1992).

[16] K. Heljanko, Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets, Fund. Inf. 37 (1999) 247–268.

[17] J. Corbett, Evaluating deadlock detection methods for concurrent software, IEEE Trans. Softw. Eng. 22 (1996) 161–180.

[18] J. Esparza, Lectures on Petri Nets I: Basic Models, Vol. 1491 of LNCS, Springer-Verlag, 1998, Ch. Decidability and Complexity of Petri Net Problems — an Introduction, pp. 374–428.

[19] J. Esparza, K. Heljanko, Unfoldings – A Partial-Order Approach to Model Checking, EATCS Monographs in Theoretical Computer Science, Springer-Verlag, 2008.

[20] N. Eén, N. Sörensson, Translating pseudo-Boolean constraints into SAT, Journal on Satisfiability, Boolean Modeling and Computation 2 (2006) 1–25.

[21] K. Heljanko, Minimizing finite complete prefixes, in: Proc. CS&P'99, 1999, pp. 83–95.

[22] Punf home page.
URL `http://homepages.cs.ncl.ac.uk/victor.khomenko/tools/punf`

[23] MiniSat home page.
URL `http://minisat.se`

[24] P. Baldan, A. Corradini, B. König, S. Schwoon, McMillan's complete prefix for contextual nets, in: Trans. on Petri Nets and Other Models of Concurrency (ToPNoC), Vol. 5100 of LNCS, Springer-Verlag, 2008, pp. 199–220.

[25] W. Vogler, A. Semenov, A. Yakovlev, Unfolding and finite prefix for nets with read arcs, in: Proc. CONCUR'98, Vol. 1466 of LNCS, Springer-Verlag, 1998, pp. 501–516.

[26] Th. Chatain, C. Jard, Symbolic diagnosis of partially observable concurrent systems, in: Proc. FORTE'04, Vol. 3235 of LNCS, Springer-Verlag, 2004, pp. 326–342.

[27] Th. Chatain, C. Jard, Complete finite prefixes of symbolic unfoldings of safe time Petri nets, in: Proc. ATPN'06, Vol. 4024 of LNCS, Springer-Verlag, 2006, pp. 125–145.