Branching Processes of High-Level Petri Nets

Victor Khomenko and Maciej Koutny

Department of Computing Science, University of Newcastle Newcastle upon Tyne NE1 7RU, U.K. {Victor.Khomenko, Maciej.Koutny}@ncl.ac.uk

Abstract. In this paper, we define branching processes and unfoldings of high-level Petri nets and propose an algorithm which builds finite and complete prefixes of such unfoldings. The advantage of our method is that it avoids a potentially expensive translation of a high-level Petri net into a low-level one. The approach is conservative in the sense that all the verification tools employing the traditional unfoldings can be reused with prefixes derived directly from high-level nets. Our experiments demonstrate that this method is often superior to the usual one, involving the explicit construction of the intermediate low-level net.

Keywords: verification, model checking, high-level Petri nets, unfolding, canonical prefix.

1 Introduction

A distinctive characteristic of reactive concurrent systems is that their sets of local states have descriptions which are both short and manageable, and the complexity of their behaviour comes from highly complicated interactions with the external environment rather than from complicated data structures and manipulations thereon. One way of coping with this complexity problem is to use formal methods and, especially, computer aided verification tools implementing model checking (see, e.g., [5]) — 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. That is, even a relatively small system specification can (and often does) yield a very large state space. To help in coping with this, a number of techniques have been proposed, which can roughly be classified as aiming at an implicit compact representation of the full state space of a reactive concurrent system, or at an explicit generation of its reduced (though sufficient for a given verification task) representation.

McMillan's (finite prefixes of) Petri Net unfoldings (see, e.g., [24, 25]) rely on the partial order view of concurrent computation, and represent system's actions and local states implicitly, using an acyclic net. In view of the development of fast model checking algorithms employing unfoldings ([12, 14, 18]), the problem of efficiently building them is becoming increasingly important. Recently, [8, 9, 13, 15, 19, 27] addressed this issue — considerably improving the original McMillan's technique — but we feel that generating net unfoldings deserves further investigation. In particular, it is highly desirable to generalize this technique to more expressive formalisms, such as high-level (or 'coloured') Petri nets. This formalism allows one to model in quite a natural way many constructs of high-level specification languages used to describe concurrent systems [2, 10, 11]. Though it is possible to translate a high-level net into a low-level one and then unfold the latter, it is often the case that the intermediate low-level net is much larger than the resulting prefix.

In this paper, we propose an approach which allows one to build the prefix directly from a high-level net, thus avoiding potentially expensive translation into a low-level net. Our experiments demonstrate that this method is often superior to the traditional one, involving the explicit construction of an intermediate low-level net. We show that it is possible to generate exactly the same prefix which would have been generated by the traditional approach, and so

2 V. Khomenko, M. Koutny

all the verification tools employing unfoldings can be reused with prefixes generated by the method proposed in this paper.

While writing up this paper, we found out that a related work was reported in [22]. Therefore, we highlight here the main differences between our approach and that of [22]. We establish an important relation between the branching processes of a high-level net and those of its lowlevel counterpart. This allows us to import results proven for branching processes of low-level nets rather than re-prove them. Among such results are the canonicity of the prefix for different cutting contexts, the usability of the total adequate order proposed in [9], and the parallel unfolding algorithm proposed in [15] (neither of these were proved in [22]). Moreover, we adopt a different way of introducing branching processes of high-level nets, which results in a neat and relatively easy-to-comprehend presentation. In particular, we do not use algorithm-dependent proofs, and we tried to make the definitions for high-level nets as similar to the corresponding definitions for low-level nets as possible. Finally, we do not restrict ourselves to finite sets of colours, and fix a subtle mistake of [22] in the definition of cut-off events and the related prefix (see Remark 1).

2 Notation

Throughout the paper, we use the following mathematical notation for multisets.

A multiset over a set X is a function $\mu : X \to \mathbb{N} \stackrel{\text{df}}{=} \{0, 1, 2, \ldots\}$. Note that any subset of X may be viewed (through its characteristic function) as a multiset over X. We denote $x \in \mu$ if $\mu(x) \geq 1$, and for two multisets over X, μ and μ' , we write $\mu \leq \mu'$ if $\mu(x) \leq \mu'(x)$ for all $x \in X$. We will use \emptyset to denote the *empty multiset* defined by $\emptyset(x) \stackrel{\text{df}}{=} 0$, for all $x \in X$. Moreover, a finite multiset may be represented by explicitly listing its elements between the $\{1, \ldots\}$ brackets. For example $\{y, y, z\}$ denotes the multiset μ such that $\mu(y) = 2$, $\mu(z) = 1$ and $\mu(x) = 0$, for $x \in X \setminus \{y, z\}$.

The sum of two multisets μ and μ' over X is given by $(\mu + \mu')(x) \stackrel{\text{df}}{=} \mu(x) + \mu'(x)$, the difference by $(\mu - \mu')(x) \stackrel{\text{df}}{=} \max\{0, \mu(x) - \mu'(x)\}$, and the intersection by $(\mu \cap \mu')(x) \stackrel{\text{df}}{=} \min\{\mu(x), \mu'(x)\}$, for all $x \in X$. A multiset μ is finite if there are finitely many $x \in X$ such that $\mu(x) \ge 1$. In such a case, the cardinality of μ is defined as $|\mu| \stackrel{\text{df}}{=} \sum_{x \in X} \mu(x)$.

The notation $\{P(x) \mid x \in \mu\}$, where μ is a multiset and P(x) is an object constructed from $x \in X$, will be used to denote the multiset μ' such that

$$\mu'(y) \stackrel{\text{df}}{=} \sum_{x \in X \land P(x) = y} \mu(x) \cdot y ,$$

where $\mu(x) \cdot y$ is the multiset consisting of exactly $\mu(x)$ copies of y. Furthermore, for a mapping $h : X \to Y$ and a multiset μ over X, we denote $h\{\mu\} \stackrel{\text{df}}{=} \{h(x) \mid x \in \mu\}$. For example, $\{x^2+1 \mid x \in \{-1, 0, 0, 1\}\} = \{1, 1, 2, 2\}$.

3 Low-level Petri nets

In this section, we first present basic definitions concerning Petri nets, and then recall (see also [7, 9, 20]) notions related to net unfoldings.

A net (with weighted arcs) is a triple $N \stackrel{\text{df}}{=} (P, T, W)$ such that P and T are disjoint sets of respectively places and transitions, and W is a multiset over $(P \times T) \cup (T \times P)$ called the weight function. The net N is called ordinary if W is a set; in such a case, W can be considered as a flow relation on $(P \times T) \cup (T \times P)$. A marking of N is a multiset M over P, and the set of all markings of N will be denoted by $\mathcal{M}(N)$. (Note that M is finite whenever P is.) We adopt the standard rules about drawing nets, viz. places are represented as circles, transitions as boxes, the weight function by arcs with the indicated weight (we do not draw arcs whose weight is 0, and we do not indicate the weight if it is 1), and markings are shown by placing tokens within circles. The multisets $\bullet z \stackrel{\text{df}}{=} \{ y \mid (y, z) \in W \}$ and $z^{\bullet} \stackrel{\text{df}}{=} \{ y \mid (z, y) \in W \}$, denote the *pre-* and *postset* of $z \in P \cup T$. (Note that for an ordinary net, both $\bullet z$ and z^{\bullet} are sets.) We will assume that $\bullet t \neq \emptyset \neq t^{\bullet}$, for every $t \in T$.

A net system is a pair $\Sigma \stackrel{\text{df}}{=} (N, M_0)$ comprising a finite net N = (P, T, W) and an *initial* marking M_0 . A transition $t \in T$ is *enabled* at a marking M if ${}^{\bullet}t \leq M$. Such a transition can be *fired*, leading to the marking $M' \stackrel{\text{df}}{=} M - {}^{\bullet}t + t^{\bullet}$. We denote this by $M[t\rangle M'$. The set of *reachable* markings of Σ is the smallest (w.r.t. set inclusion) set $\mathcal{RM}(\Sigma)$ containing M_0 and such that if $M \in \mathcal{RM}(\Sigma)$ and $M[t\rangle M'$, for some $t \in T$, then $M' \in \mathcal{RM}(\Sigma)$.

 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 the set $\mathcal{RM}(\Sigma)$ is finite iff Σ is bounded.

Note that if a safe net system Σ has arcs of weight more than 1, the transitions incident to them can never become enabled, and so can be removed (together with their incoming and outgoing arcs) without changing the behaviour of Σ . Thus, one can assume that the underlying nets of safe net systems are ordinary.

Places p_1, \ldots, p_k of a net system Σ are *mutually exclusive* if no reachable marking puts tokens on more than one of them, i.e., for every $M \in \mathcal{RM}(\Sigma)$, $M(p_i) \ge 1$ implies $M(p_j) = 0$, for all $j \in \{1, \ldots, k\} \setminus \{i\}$.

Low-level branching processes Two nodes (places or transitions), y and y', of an ordinary net N = (P, T, W) are *in conflict*, denoted by y # y', if there are distinct transitions $t, t' \in T$ such that ${}^{\bullet}t \cap {}^{\bullet}t' \neq \emptyset$ and (t, y) and (t', y') are in the reflexive transitive closure of the flow relation W, denoted by \preceq . A node y is in *self-conflict* if y # y.

An occurrence net is an ordinary net $ON \stackrel{\text{df}}{=} (B, E, G)$, where B is the set of conditions (places), E is the set of events (transitions) and G is a flow relation, satisfying the following: ON is acyclic (i.e., \leq is a partial order); for every $b \in B$, $|\bullet b| \leq 1$; for every $y \in B \cup E$, $\neg(y \# y)$ and there are finitely many y' such that $y' \prec y$, where \prec denotes the transitive closure of G. Min(ON) will denote the set of minimal (w.r.t. \prec) elements of $B \cup E$. The relation \prec is the causality relation. Two nodes are concurrent, denoted y co y', if neither y # y' nor $y \leq y'$ nor $y' \leq y$.

Definition 1. A homomorphism from an occurrence net ON = (B, E, G) to a net system Σ is a mapping $h: B \cup E \to P \cup T$ such that

 $-h(B) \subseteq P$ and $h(E) \subseteq T$ (conditions are mapped to places, and events to transitions).

- For each $e \in E$, $h\{\!\{\bullet e\}\!\} = \bullet h(e)$ and $h\{\!\{e^\bullet\}\!\} = h(e)^\bullet$ (transition environments are preserved).

- $-h\{Min(ON)\} = M_0$ (minimal conditions are mapped to the initial marking).
- For all $e, f \in E$, if $\bullet e = \bullet f$ and h(e) = h(f) then e = f (there is no redundancy).

A branching process of Σ is a pair $\pi \stackrel{\text{df}}{=} (ON, h)$ such that ON is an occurrence net and h is a homomorphism from ON to Σ .

If an event e is such that h(e) = t then we will often refer to it as being t-labelled. A branching process $\pi' = (ON', h')$ of Σ is a prefix of a branching process $\pi = (ON, h)$, denoted by $\pi' \sqsubseteq \pi$, if ON' = (B', E', G') is a subnet of ON = (B, E, G) containing all minimal elements and such that: if $e \in E'$ and $(b, e) \in G$ or $(e, b) \in G$ then $b \in B'$; if $b \in B'$ and $(e, b) \in G$ then $e \in E'$; and h' is the restriction of h to $B' \cup E'$. For each Σ there exists a unique (up to isomorphism) maximal (w.r.t. \Box) branching process Unf_{Σ}^{max} , called the unfolding of Σ (see [7]).

Sometimes it is convenient to start a branching process with a (virtual) initial event \perp , which has the postset Min(ON), empty preset, and no label; we will henceforth use such an event, without drawing it in figures or treating it explicitly in algorithms.

An example of a safe net system and two of its branching prefixes is shown in Figure 1, where the homomorphism h is indicated by the labels of the nodes. The process in Figure 1(b) is a prefix of that in Figure 1(c).



Fig. 1. A net system (a) and two of its branching processes (b,c).

Configurations and cuts A configuration of an occurrence net ON is a set of events C such that for all $e, f \in C, \neg(e \# f)$ and, for every $e \in C, f \prec e$ implies $f \in C$; since we assume the initial event \bot , we additionally require that $\bot \in C$. For every event $e \in E$, the configuration $[e] \stackrel{\text{df}}{=} \{f \mid f \preceq e\}$ is called the *local configuration* of e, and $\langle e \rangle \stackrel{\text{df}}{=} [e] \setminus \{e\}$ denotes the set of causal predecessors of e. Moreover, for a set of events E' we denote by $C \oplus E'$ the fact that $C \cup E'$ is a configuration and $C \cap E' = \emptyset$. Such an E' is a suffix of C, and $C \oplus E'$ is an extension of C.

The set of all finite (resp. local) configurations of a branching process π will be denoted by C_{fin}^{π} (resp. C_{loc}^{π}), and we will drop the superscript π in the case $\pi = Unf_{\Sigma}^{max}$.

A set of events E' is *downward-closed* if all causal predecessors of the events in E' also belong to E'. Such a set *induces* a unique branching process π whose events are exactly the events in E', and whose conditions are the conditions adjacent to the events in E' (including \perp).

A set of conditions B' such that for all distinct $b, b' \in B'$, $b \ co \ b'$, is called a *co-set*. A *cut* is a maximal (w.r.t. set inclusion) co-set. Every marking reachable from Min(ON) is a cut.

Let C be a finite configuration of a branching process π . Then the set

$$Cut(C) \stackrel{\text{\tiny df}}{=} \left(Min(ON) \cup \bigcup_{c \in C} c^{\bullet} \right) \setminus \left(\bigcup_{c \in C} \bullet c \right)$$

is a cut; moreover, the multiset of places $Mark(C) \stackrel{\text{df}}{=} h\{\!\!\{Cut(C)\}\!\!\}$ is a reachable marking of Σ , called the *final marking* of C. A marking M of Σ is *represented* in π if there is $C \in C_{fin}^{\pi}$ such that M = Mark(C). Every marking represented in π is reachable in the original net system Σ , and every reachable marking of Σ is represented in Unf_{Σ}^{max} .

Complete prefixes of Petri net unfoldings Though unfoldings are infinite whenever the original net systems have infinite runs, it turns out that often they can be truncated in such a way that the resulting prefixes, though finite, contain enough information to decide a certain behavioural property, e.g., deadlock freeness. We then say that the prefixes are *complete* for this property.

There exist several different methods of truncating Petri net unfoldings. The differences are related to the kind of information about the original unfolding one wants to preserve in the prefix, as well as to the choice between using only local configurations (which can improve the running time of an algorithm), or all finite configurations (which can result in a smaller prefix), to cut the unfolding. In [20], a uniform approach to truncating unfoldings, based on *cutting contexts*, was proposed.

Cutting contexts For greater flexibility, the approach proposed in [20] is parametric. The first parameter determines the information to be preserved in a complete prefix (in the standard case, the set of reachable markings). The main idea there was to shift the emphasis from the reachable markings of Σ to the finite configurations of Unf_{Σ}^{max} . Formally, the information to be preserved in the prefix corresponds to the equivalence classes of some equivalence relation \approx on C_{fin} . The other two parameters are more technical: they specify under which circumstances an event can be designated as a cut-off event (intuitively, this means that all its causal successors in the full unfolding can be removed).

Definition 2. A cutting context is a triple $\Theta \stackrel{\text{\tiny df}}{=} \left(\approx, \triangleleft, \left\{\mathcal{C}_e\right\}_{e \in E}\right)$, where:

- 1. \approx is an equivalence relation on C_{fin} .
- 2. \triangleleft , called an *adequate* order, is a strict well-founded partial order on \mathcal{C}_{fin} refining \subset , i.e., $C' \subset C''$ implies $C' \triangleleft C''$.

$\mathbf{6}$ V. Khomenko, M. Koutny

3. \approx and \triangleleft are preserved by finite extensions, i.e., for every pair of configurations $C' \approx C''$, and for every suffix E' of C', there exists¹ a finite suffix E'' of C'' such that:

 \diamond

 \diamond

- (a) $C'' \oplus E'' \approx C' \oplus E'$, and (b) if $C'' \triangleleft C'$ then $C'' \oplus E'' \triangleleft C' \oplus E'$. 4. $\{\mathcal{C}_e\}_{e \in E}$ is a family of subsets of \mathcal{C}_{fin} , i.e., $\mathcal{C}_e \subseteq \mathcal{C}_{fin}$, for all $e \in E$.

The main idea behind the adequate order is to specify which configurations will be preserved in the complete prefix; it turns out that all *I*-minimal configurations in each equivalence class of \approx will be preserved. The last parameter is needed to specify the set of configurations used later to decide whether an event can be designated as a cut-off event. For example, C_e may contain all finite configurations of Unf_{Σ}^{max} , or, as it is usually the case in practice, only the local ones. We will say that a cutting context Θ is *dense (saturated)* if $C_e \supseteq C_{loc}$ (resp. $C_e = C_{fin}$), for all $e \in E$.

In practice, Θ is usually dense (or even saturated, see [13]), and at least the following three kinds of the equivalence \approx have been used:

- $-C' \approx_{mar} C''$ if Mark(C') = Mark(C''). This is the most widely used equivalence (see [9, 13, 15,24). Note that the equivalence classes of \approx_{max} correspond to the reachable markings of Σ .
- $-C' \approx_{code} C''$ if Mark(C') = Mark(C'') and Code(C') = Code(C''), where Code(C) is the signal coding function. Such an equivalence is used in [28] for unfolding Signal Transition Graphs (STGs) specifying asynchronous circuits.
- $-C' \approx_{sum} C''$ if Mark(C') and Mark(C'') are symmetric (equivalent) markings. This equivalence is the basis of the approach exploiting symmetries to reduce the size of the prefix, described in [6].

We will write $e \triangleleft f$ whenever $[e] \triangleleft [f]$. Clearly, \triangleleft is a well-founded partial order on the set of events refining \prec . Hence, one can use the Noetherian induction for definitions and proofs, i.e., it suffices to define or prove something for an event under the assumption that it has already been defined or proven for all its ⊲-predecessors. In the rest of this section, we assume that the cutting context Θ is fixed.

Completeness of branching processes We now recall (see [20]) the notion of completeness for branching processes.

Definition 3. A branching process π of Σ is complete w.r.t. a set E_{cut} of events of Unf_{Σ}^{max} if the following hold:

- 1. If $C \in \mathcal{C}_{fin}$, then there is $C' \in \mathcal{C}_{fin}^{\pi}$ such that $C' \cap E_{cut} = \emptyset$ and $C \approx C'$. 2. If $C \in \mathcal{C}_{fin}^{\pi}$ is such that $C \cap E_{cut} = \emptyset$, and e is an event such that $C \oplus \{e\} \in \mathcal{C}_{fin}$, then $C \oplus \{e\} \in \mathcal{C}_{fin}^{\pi}.$

A branching process π is *complete* if it is complete w.r.t. some set E_{cut} .

Note that π remains complete following the removal of all events e for which $\langle e \rangle \cap E_{cut} \neq \emptyset$, after which the events from E_{cut} (usually referred to as *cut-off* events) will be either maximal events of the prefix or not in the prefix at all. Note also that the last definition depends only on the equivalence \approx , and not on the other components of the cutting context.

For the relation \approx_{mar} , each reachable marking is represented by a configuration in C_{fin} and, hence, also by a configuration in C_{fin}^{π} , provided that π is complete. This is what is usually expected from a correct prefix. Moreover, Definition 3(2) implies that all firings enabled by the configurations from C_{fin}^{π} containing no events from E_{cut} are preserved (see [20] for the explanation why this property is desirable).

Unlike [9], we do not require that $E'' = I_1^2(E')$, where I_1^2 is the 'natural' isomorphism between the finite extensions of C' and C''. That isomorphism may be undefined if $Mark(C') \neq Mark(C'')$, and thus cannot be used in our generalised setting.

7

Static cut-off events Here we recall (see [20]) the definition of static cut-off events. They are defined w.r.t. the whole unfolding, so that they are independent on an algorithm (hence the term 'static'), together with *feasible* events, which are precisely those events whose causal predecessors are not cut-off events, and as such must be included in the prefix determined by the static cut-off events.

Definition 4. The sets of *feasible* events, denoted by $fsble_{\Theta}$, and *static cut-off* events, denoted by cut_{Θ} , of Unf_{Σ}^{max} are defined thus:

- 1. An event e is a feasible event if $\langle e \rangle \cap cut_{\Theta} = \emptyset$.
- 2. An event e is a static cut-off event if it is feasible, and there is a configuration $C \in C_e$ such that $C \subseteq fsble_{\Theta} \setminus cut_{\Theta}, C \approx [e]$, and $C \triangleleft [e]$. Any C satisfying these conditions will be called a *corresponding* configuration of e.

It turns out that, due to the well-foundedness of \triangleleft , $fsble_{\Theta}$ and cut_{Θ} are well-defined sets (see [20]). Since $\langle \perp \rangle = \emptyset$, $\perp \in fsble_{\Theta}$ by the above definition. Furthermore, $\perp \notin cut_{\Theta}$, since \perp cannot have a corresponding configuration. Indeed, $[\perp] = \{\perp\}$ is the smallest (w.r.t. set inclusion) configuration, and so \triangleleft -minimal by Definition 2(2).

Remark 1. A naïve attempt to define an algorithm-independent notion of a cut-off event as an event e for which there is a configuration $C \in C_e$ such that $C \approx [e]$ and $C \triangleleft [e]$ generally fails for non-saturated cutting contexts, e.g., when (as it is often the case in practice) only local configurations can be used as cut-off correspondents. Indeed, a corresponding local configuration C of a cut-off event e defined in this way may contain another cut-off event. Though in this case Unf_{Σ}^{max} contains another corresponding configuration $C' \approx C$ with no cut-off events such that $C' \triangleleft C$, C' is not necessarily local.

Our approach, though slightly more complicated, allows one to deal uniformly with arbitrary cutting contexts. Moreover, it coincides with the described naïve approach when Θ is saturated. \diamond

Canonical prefix Once the feasible events are defined, the following notion arises quite naturally. The *canonical* prefix of Unf_{Σ}^{max} is the branching process Unf_{Σ}^{Θ} induced by $fsble_{\Theta}$. Thus Unf_{Σ}^{Θ} is uniquely determined by the cutting context Θ . The paper [20] proves that the canonical prefix is complete (in the sense of Definition 3), and also investigates the conditions which guarantee its finiteness. Further in this paper we will show that all these results can be imported to the theory of branching processes of high-level Petri nets.

Algorithms for generating canonical prefixes It turns out that canonical prefixes can be constructed by straightforward generalizations of the existing unfolding algorithms (see, e.g., [9, 15]). The *slicing* algorithm from [15], parameterised by a cutting context Θ , is shown in Figure 2. (The algorithm proposed in [9] is its special case.) It is assumed that the function POTEXT(π) finds the set of *possible extensions* of a branching process π , according to the following definition.

Definition 5. For a branching process π of Σ , a *possible extension* is a pair (D, t), where D is a co-set in π and t is a transition of Σ , such that $h\{D\} = {}^{\bullet}t$ and π contains no t-labelled event with preset D.

We will take the pair (D, t) as a t-labelled event having D as its preset.

Compared to the standard unfolding algorithm in [9], the slicing algorithm has the following modifications in its main loop. A set of events Sl, called a *slice*, is chosen on each iteration and processed as a whole, without taking or adding any events from or to *pe*. A slice must satisfy the following conditions:

-Sl is a non-empty subset of the current set of possible extensions pe.

8 V. Khomenko, M. Koutny

input : $\Sigma = (N, M_0)$ — a net system **output** : $Pref_{\Sigma}$ — the canonical prefix of Σ 's unfolding (if it is finite) $Pref_{\Sigma} \leftarrow$ the empty branching process add instances of the places from M_0 to $Pref_{\Sigma}$ $pe \leftarrow \text{POTExt}(Pref_{\Sigma})$ $cut_off \leftarrow \emptyset$ while $pe \neq \emptyset$ do choose $Sl \in SLICES(pe)$ if $\exists e \in Sl : [e] \cap cut_off = \emptyset$ then for all $e \in Sl$ in any order refining \triangleleft do if $[e] \cap cut_off = \emptyset$ then adde together with its postset to $\textit{Pref}_{\varSigma}$ **if** e is a cut-off event of $Pref_{\Sigma}$ **then** $cut_off \leftarrow cut_off \cup \{e\}$ $pe \leftarrow \text{POTEXT}(Pref_{\Sigma})$ else $pe \leftarrow pe \setminus Sl$

Note: e is a cut-off event of $Pref_{\Sigma}$ if there is $C \in C_e$ such that the events of C belong to $Pref_{\Sigma}$ but not to *cut_off*, $C \approx [e]$, and $C \triangleleft [e]$.

Fig. 2. Unfolding algorithm with slices.

- For every $e \in Sl$ and every event $f \triangleleft e$ of Unf_{Σ}^{max} , $f \notin pe \setminus Sl$ and $pe \cap \langle f \rangle = \emptyset$.

In particular, if $f \in pe$ and $f \triangleleft e$ for some $e \in Sl$, then $f \in Sl$. The set SLICES(*pe*) is chosen so that it is non-empty whenever *pe* is non-empty. Note that this algorithm, in general, exhibits more non-determinism than the one from [9]: it may be non-deterministic even if the order \triangleleft is total. Since the events in the current slice can be processed independently, the slicing algorithm admits efficient parallelization (along the lines proposed in [15]). A crucial property of the slicing unfolding algorithm is that it generates the canonical prefix.

Theorem 6 ([15, 20]). If Unf_{Σ}^{Θ} is finite then the slicing algorithm generates Unf_{Σ}^{Θ} in a finite number of steps.

4 High-level Petri nets

In this paper we use M-nets (see [1]) as the main high-level Petri net model, as we believe that it is general enough to cover many other existing relevant formalisms. The full definition of M-nets can be found in [1]. Here, in order to match the presentation of low-level nets as closely as possible, we give a suitably adapted short description omitting those details which are not directly related to our purposes. In particular, [1] devotes a lot of attention to the composition rules for M-nets, which are relevant only at the construction stage of an M-net, but not for model checking of an already constructed one.

M-nets

It is assumed that there exists a (finite or infinite) set Tok of elements (or 'colours') and a set VAR of variable names, such that $Tok \cap VAR = \emptyset$. An *M*-net *N* is a quadruple $N \stackrel{\text{df}}{=} (P, T, W, \iota)$ such that *P* and *T* are disjoint sets of respectively places and transitions, *W* is a multiset over $(P \times VAR \times T) \cup (T \times VAR \times P)$ of arcs, and ι is an inscription function with the domain

9

 $P \cup T$. It is assumed that, for every place $p \in P$, $\iota(p) \subseteq Tok$ is the *type* of p and, for every transition $t \in T$, $\iota(t)$ is a boolean expression over $Tok \cup VAR$, called the *guard* of t. We assume that the types of all places are finite.² In what follows, we assume that $N = (P, T, W, \iota)$ is a fixed M-net.

For a transition $t \in T$, let $\bullet t \stackrel{\text{df}}{=} \{p^v \mid (p, v, t) \in W\}, t^{\bullet} \stackrel{\text{df}}{=} \{p^v \mid (t, v, p) \in W\}$, and $VAR(t) \stackrel{\text{df}}{=} \{v \mid (p, v, t) \in W\} \cup VAR(\iota(t))$, where $VAR(\iota(t))$ is the set of variables appearing in $\iota(t)$. A firing mode of t is a mapping $\sigma : VAR(t) \to Tok$ such that $\sigma(v) \in \iota(p)$, for all p^v in $\bullet t + t^{\bullet}$, and $\iota(t)$ evaluates to **true** under the substitution given by σ . (The notation p^v , similarly as p^x and t^{σ} used later on, is a shorthand for the pair (p, v).)

We define the set of *legal place instances* as $\mathcal{P} \stackrel{\text{df}}{=} \{p^x \mid p \in P \land x \in \iota(p)\}$ and the set of *legal firings* as $\mathcal{T} \stackrel{\text{df}}{=} \{t^{\sigma} \mid t \in T \text{ and } \sigma \text{ is a firing mode of } t\}$. For every $t^{\sigma} \in \mathcal{T}$, we will also denote $\bullet t^{\sigma} \stackrel{\text{df}}{=} \{p^{\sigma(v)} \mid p^v \in \bullet t\}$ and $t^{\sigma \bullet} \stackrel{\text{df}}{=} \{p^{\sigma(v)} \mid p^v \in t^{\bullet}\}$. According to the definitions given below, all valid markings of an M-net will be composed of legal place instances, and its firing sequences will be composed of legal firings. Furthermore, the sets \mathcal{P} and \mathcal{T} will provide the basis for the construction of the low-level net corresponding to a high-level one.

A marking M of N is a multiset over \mathcal{P} . We will denote the set of all such markings by $\mathcal{M}(N)$. (Traditionally, a marking is a mapping which, to every place $p \in P$, associates a multiset over $\iota(p)$. Clearly, such a representation is equivalent to that we chose to use.)

The transition relation is a ternary relation on $\mathcal{M}(N) \times \mathcal{T} \times \mathcal{M}(N)$ such that a triple (M, t^{σ}, M') belongs to it (denoted $M[t^{\sigma} \rangle M')$ if ${}^{\bullet}t^{\sigma} \leq M$ and $M' = M - {}^{\bullet}t^{\sigma} + t^{\sigma}{}^{\bullet}$. Note that σ is a firing mode of t, which guarantees that M' is a valid marking of N.

M-net systems

An *M*-net system is a pair $\Upsilon \stackrel{\text{df}}{=} (N, M_0)$ comprising a finite M-net N and an *initial* marking M_0 . The set of *reachable* markings of an M-net system Υ is the smallest (w.r.t. set inclusion) set $\mathcal{RM}(\Upsilon)$ containing M_0 and such that if $M \in \mathcal{RM}(\Upsilon)$ and $M[t^{\sigma}\rangle M'$ in N, for some $t^{\sigma} \in \mathcal{T}$, then $M' \in \mathcal{RM}(\Upsilon)$.

An M-net net system Υ is k-bounded if, for every marking $M \in \mathcal{RM}(\Upsilon)$ and every $p^x \in \mathcal{P}$, $M(p^x) \leq k$; safe if it is 1-bounded; and bounded if it is k-bounded for some $k \in \mathbb{N}$. Moreover, Υ is strictly k-bounded if, for every marking $M \in \mathcal{RM}(\Upsilon)$ and every place $p \in P$, $|\{x \mid p^x \in M\}| \leq k$, and strictly safe if it is strictly 1-bounded. One can show that strictly k-bounded M-net systems are k-bounded, strictly safe ones are safe, and the set $\mathcal{RM}(\Upsilon)$ is finite iff Υ is bounded. Note that according to the above definitions, a safe M-net system can have a reachable marking which places several tokens on the same place, provided that their 'colours' are all distinct. The rational behind our choice of the definition is that the low-level expansion (defined below) of an M-net system is safe iff the original M-net system is safe, and so the total adequate order proposed in [9] for safe net systems can be re-used (see the end of Section 5).

As an example, consider the M-net system shown in Figure 3(a). At the initial marking, t_1 can fire with the firing mode $\sigma \stackrel{\text{df}}{=} \{v_1 \mapsto 1, v_2 \mapsto 2, v_3 \mapsto 1\}$ or $\sigma' \stackrel{\text{df}}{=} \{v_1 \mapsto 1, v_2 \mapsto 2, v_3 \mapsto 2\}$, consuming the tokens from p_1 and p_2 and producing respectively the token 1 or 2 on p_3 . Formally, we have $\{p_1^1, p_2^2\}[t_1^{\sigma} \rangle \{p_3^1\}$ and $\{p_1^1, p_2^2\}[t_1^{\sigma'} \rangle \{p_3^2\}$.

Transforming M-net systems into low-level nets For each M-net it is possible to build an 'equivalent' low-level one. Such a transformation is called 'unfolding' in [1], but since this term is used in this paper with a different meaning (see Section 3), we will use the term 'expansion' instead.

 $^{^{2}}$ In general, allowing infinite types yields a Turing-powerful model. Nevertheless, this restriction can be omitted in certain important cases (see Section 6).



Fig. 3. An M-net system (a), its expansion (b), and its unfolding (c). Note that a firing mode σ of t is represented as a three-element string $\sigma(v_1)\sigma(v_2)\sigma(v_3)$.

The expansion $\mathcal{E}(N)$ of an M-net $N = (P, T, W, \iota)$ is a low-level net $\mathcal{E}(N) \stackrel{\text{df}}{=} (\mathcal{P}, \mathcal{T}, W')$ where

$$W' \stackrel{\mathrm{df}}{=} \sum_{t^{\sigma} \in \mathcal{T}} \left(\left\{ \left(p^{\sigma(v)}, t^{\sigma} \right) \mid (p, v, t) \in W \right\} + \left\{ \left(t^{\sigma}, p^{\sigma(v)} \right) \mid (t, v, p) \in W \right\} \right)$$

Moreover, the expansion $\mathcal{E}(M)$ of a marking M of N is M itself, i.e., $\mathcal{E}(M) \stackrel{\text{df}}{=} M$ (this is possible since we chose our definitions so that there is no difference between the markings of $\mathcal{E}(N)$ and N.) Finally, the expansion of an M-net system $\Upsilon = (N, M_0)$ is defined as $\mathcal{E}(\Upsilon) \stackrel{\text{df}}{=} (\mathcal{E}(N), \mathcal{E}(M_0))$. Figure 3(a,b) illustrates the last definition.

One can show that the following hold.

Proposition 1 ([1]). Let N be an M-net, and $M', M'' \in \mathcal{M}(N)$. Then $M'[t^{\sigma} \rangle M''$ in Υ iff $M'[t^{\sigma} \rangle M''$ in $\mathcal{E}(\Upsilon)$.

Proposition 2. Let $\Upsilon = (N, M_0)$ be an M-net system.

- For every $k \in \mathbb{N}$, $\mathcal{E}(\Upsilon)$ is k-bounded iff Υ is k-bounded.
- $\mathcal{E}(\Upsilon)$ is safe iff Υ is safe.
- If Υ is strictly safe and p is a place of Υ , then the places p^x , $x \in \iota(p)$, are mutually exclusive in $\mathcal{E}(\Upsilon)$.

Proof. Follows directly from the definitions.

Though, according to Proposition 1, the expansion of an M-net system faithfully models the original system, the disadvantage of this transformation is that it usually yields a very large net. Moreover, the resulting net system is usually *unnecessarily* large, in the sense that it contains many places which cannot be marked and many dead transitions. This is so because the place types are usually overapproximations, and the transitions of the original M-net system may have many firing modes, only few of which are realized when executing the net from the initial marking. For example, only two out of eight transitions of the expansion of the M-net system in Figure 3(a), shown in Figure 3(b), can actually fire. Therefore, though the M-net expansion is a neat theoretical construction, it is often impractical.

5 Branching processes of high-level nets

In this section we develop the main results of this paper, namely the notions of a branching process of an M-net system, the associated unfolding, and its canonical prefix. We also show that there is a strong correspondence between the branching processes of an M-net system and those of its expansion. This allows for importing many results from the theory of branching processes of low-level Petri nets.

Definition 7. A homomorphism from an occurrence net ON = (B, E, G) to an M-net system Υ is a mapping $h: B \cup E \to \mathcal{P} \cup \mathcal{T}$ such that

- $(-h(B) \subseteq \mathcal{P} \text{ and } h(E) \subseteq \mathcal{T} \text{ (conditions are mapped to legal place instances, and events to legal firings).}$
- For every $e \in E$, $h\{\!\!\{\bullet e\}\!\!\} = \bullet h(e)$ and $h\{\!\!\{e^\bullet\}\!\!\} = h(e)^\bullet$ (the environments of legal firings are preserved).
- $-h\{Min(ON)\} = M_0$ (minimal conditions are mapped to the initial marking).
- For all $e, f \in E$, if $\bullet e = \bullet f$ and h(e) = h(f), then e = f (there is no redundancy).

A branching process of Υ is a pair $\pi \stackrel{\text{df}}{=} (ON, h)$ such that ON is an occurrence net and h is a homomorphism from ON to Υ .

12 V. Khomenko, M. Koutny

The above definition is illustrated in Figure 3.

Definition 7 closely follows the definition of a (low-level) branching process of $\mathcal{E}(\Upsilon)$. Thus most of the definitions for branching processes of low-level net systems can now be lifted to branching processes of M-net systems. In particular, this is the case for the notions of a *configuration*, *cut*, *final marking*, prefix relation \sqsubseteq , *cutting context*, and the notion of *completeness* of a prefix. Similarly, most of the results proven for branching processes of low-level Petri nets can also be lifted to branching processes of M-net systems. In particular, for each M-net system Υ there exist a unique (up to isomorphism) maximal w.r.t. \sqsubseteq branching process Unf_{Υ}^{max} of Υ , called the *unfolding* of Υ . Moreover, for any cutting context Θ there exists unique *canonical* prefix Unf_{Υ}^{Θ} (coinciding with $Unf_{\mathcal{E}(\Upsilon)}^{\Theta}$) of Unf_{Υ}^{max} , and the theory of *canonical* prefixes (see [20]) can be transferred without any changes.

Remark 2. One should be careful when dealing with adequate orders: though they are defined on the configurations of the unfolding, in practice the node labels are often employed in order to compute it. In particular, \triangleleft is often parameterized by some order \ll on the set of transitions of the low-level Petri net (see [8, 9]). Hence, in order to unfold a high-level net Υ one has to define such an order on \mathcal{T} rather than on T.

It is straightforward to give an upper bound on the size of Unf_{Υ}^{Θ} , since the results of [9, 20] regarding the size of the canonical prefix are still applicable. In particular, if the cutting context $\Theta = \left(\approx, \triangleleft, \{\mathcal{C}_e\}_{e \in E}\right)$ is dense, \triangleleft is total, and $C' \approx C'' \Leftrightarrow Mark(C') = Mark(C'')$, then the number of non-cut-off events in Unf_{Υ}^{Θ} does not exceed $|\mathcal{RM}(\Upsilon)|$.

6 M-net unfolding algorithm

Thanks to the results developed in the previous section, it is now possible to suggest a suitable modification of the standard unfolding algorithms, e.g., that in Figure 2, which is capable of building canonical prefixes of M-net unfoldings. It turns out that the only thing which has to be changed is the notion of a possible extension (so all the modifications are inside the POTEXT function and thus are not visible in the top-level description of the algorithm).

Definition 8. For a branching process π of an M-net system Υ , a *possible extension* is a pair (D, t^{σ}) , where D is a co-set in π and t^{σ} is a legal firing, such that $h\{D\} = {}^{\bullet}t$ and π contains no t^{σ} -labelled event with preset D.

Similarly as in the low-level case, we will take the pair (D, t^{σ}) as a new event of the prefix, with the preset D. After it is inserted into the prefix, its postset D' consisting of new conditions such that $h\{D'\} = t^{\sigma \bullet}$ is also inserted.

It is worth noting that most of the existing heuristics aiming at speeding up the prefix generation can be applied. In particular, the total adequate order for safe net systems proposed in [9] can be used to unfold safe M-net systems. It is still adequate, since Unf_{Υ}^{max} coincides with $Unf_{\mathcal{E}(\Upsilon)}^{max}$ and the expansion of a safe M-net system is safe. Moreover, the concurrency relation (see [8, 27]) can also be employed, even for non-safe systems. As for the preset trees (see [19]), they can be used without any modifications to unfold strictly safe M-net systems (and we work now on generalizing them to a wider class of M-net systems).

It turns out that direct unfolding a high-level net not only avoids the generation of its (potentially, very large) expansion, but often is also more efficient than unfolding its expansion. Indeed, the most time-consuming part of the algorithm is computing the possible extensions.³ Since one high-level transition usually corresponds to several low-level ones, less transitions have to be tried each time possible extensions are computed, which may lead to considerable savings in the running time.

³ Checking whether the set of possible extensions is empty is, in fact, an NP-complete problem (see [12]).

It is often the case that the information about the firing mode of an event needs not be explicitly stored. Indeed, this information almost always can be discarded, since one is usually not interested what was the precise firing mode of a transition, as long as the consumed and produced tokens are the same.

An important extension of our approach allows for M-nets with places having infinite types. For example, it is often convenient to assign to a place the type N rather than $\{1, \ldots, n\}$, since n might be not known in advance. Even when the set of reachable markings of such an M-net system is finite, its expansion is infinite and so of little use for model checking, whereas with our direct approach we still can build the canonical prefix and complete the verification. The only thing which needs to be ensured is that at any stage of prefix construction only a finite number of legal firings needs to be considered. This will be the case if, for every transition t and every finite multiset Z over \mathcal{P} , the set of all firing modes σ of t such that $\bullet t^{\sigma} \leq Z$ is both finite and computable.

Having built a canonical prefix, one can easily construct the refined version of the lowlevel expansion of the original M-net system, with unreachable places and dead transitions removed. This may be important, e.g., for directly mapping a Petri net to a circuit simulating its behaviour.

Finally, it is worth mentioning that since our method constructs exactly the same prefix which would have been generated from the corresponding expansion of the M-net system, all the existing model checkers employing unfolding prefixes derived from low-level nets can be used without any changes when dealing with prefixes generated directly from M-net systems.

7 Case studies

In this section, we compare our approach with the traditional one, viz. the unfolding of M-net expansions. We used the unfolding engine described in [15, 19] which after suitable modifications was able to unfold both low-level and high-level nets. For building M-net expansions, we used the h1211 utility from the PEP tool (see [3, 4]). The experiments were conducted on a PC with a *PentiumTM* III/500MHz processor and 128M RAM.

The meaning of the columns in the tables is as follows (from left to right): the size of the problem; the number of places and transitions in the original M-net system; the number of places and transitions in the corresponding expansion, together with the time required by the h1211 utility to build the expansion; the number of conditions, feasible events, and cut-off events in the canonical prefix; the times (in seconds) required to unfold the M-net system and its expansion, respectively.

The first example is data-intensive, and so the traditional (via low-level nets) approach is extremely inefficient, whereas we expected our algorithm to perform well. The second example is control-intensive, so the M-net expansions are just slightly larger that the original M-nets. It was chosen to test the worst-case performance of our method relatively to the unfolding of the low-level expansion.

7.1 Greatest common divisor

An M-net simulating the Euclid's algorithm for computing the greatest common divisor of two non-negative integers, together with its unfolding, is shown in Figure 4. In our experiments, for each N we computed the greatest common divisor of F_N and F_{N-1} , where F_i denotes the *i*-th Fibonacci's number (such numbers are known to produce the longest sequences of computational steps for the Euclid's algorithm). The results of our experiments are summarized in Table 1. From the structure of the M-net it is easy to calculate that its expansion contains $3(F_N + 1)$ places and $(F_N + 1)^2$ transitions. These values are reported in the corresponding columns of the table, even though hl2ll failed to produce the expansions when they became large.



Fig. 4. An M-net system modelling the Euclid's algorithm for computing the greatest common divisor of two non-negative integers (a), and its unfolding (b). Firing modes are not shown in the unfolding.

The experimental results show that for this example the high-level unfolding is clearly superior. Though the M-net expansion grows very quickly, the resulting prefix has only 2N - 1 conditions and N - 1 events. Therefore, our algorithm was able to build it for relatively large N (we had to stop the experiments after N = 45 since F_{50} overflows 4-bytes integer, but it is a limitation of the current implementation rather than of the method itself).

	M-net		Expansion			Unfolding			Time[s]	
Ν	P	T	P	T	$\operatorname{Time}[\mathbf{s}]$	B	E	$ E_{cut} $	LL	HL
5	3	2	18	36	<1	9	4	0	$<\!\!1$	<1
10	3	2	168	3136	1	19	9	0	6	$<\!\!1$
15	3	2	1833	$>10^{5}$		29	14	0		$<\!\!1$
20	3	2	$>10^{4}$	$>10^{7}$		39	19	0		$<\!\!1$
25	3	2	$>10^{5}$	$>10^{9}$		49	24	0		$<\!\!1$
30	3	2	$>10^{6}$	$>10^{11}$		59	29	0		$<\!\!1$
35	3	2	$>10^{7}$	$>10^{13}$		69	34	0		$<\!\!1$
40	3	2	$>10^{8}$	$> 10^{16}$		79	39	0	—	$<\!\!1$
45	3	2	$> 10^{9}$	$> 10^{18}$		89	44	0		$<\!\!1$

Table 1. Experimental results for the M-net system simulating the Euclid's algorithm.

7.2 Mutual exclusion algorithm

The previous example was rather favourable for our algorithm, since the expansions of the M-net systems were very large. We therefore checked the performance of our approach in a totally opposite case, when the expansion of an M-net is relatively small. This happens when the transitions of the M-net are connected to few places and the cardinality of most place types

```
start:
      \langle b_i \leftarrow \mathbf{true} \rangle;
      \langle x \leftarrow i \rangle;
      if < y \neq 0 >
      then
            < b_i \leftarrow \mathbf{false} >;
            await < y = 0 >;
            goto start;
      \langle y \leftarrow i \rangle;
      if \langle x \neq i \rangle
      then
            \langle b_i \leftarrow \mathbf{false} \rangle:
            for j \leftarrow 1 to N do
                        await \langle b_i = false \rangle;
            if \langle y \neq i \rangle
            \mathbf{then}
                        await < y = 0 >;
                        goto start;
      critical section;
      \langle y \leftarrow 0 \rangle;
```

 $< b_i \leftarrow$ false >;

Fig. 5. The pseudocode of the *i*-th process in Lamport's mutual exclusion algorithm.

is 1. Such an M-net arises when modelling Lamport's mutual exclusion algorithm (see [16, 23]), which employs 'very small' atomic actions. The pseudo-code of this algorithm is shown in Figure 5, where N is the number of processes trying to access the critical section. We encoded it in the $B(PN)^2$ language supported by the PEP tool, as shown in Figure 6. Note that we had to replicate parts of the code since $B(PN)^2$ currently does not support the **goto** operator. The type of the places corresponding to the variables x and y is $\{0, \ldots, N\}$, the type of places corresponding to b_i 's is {**false, true**}, and all the other places have the type { \bullet } and thus are not replicated in the expansion. Every transition has not more than 2 incoming and 2 outgoing arcs, and is connected to at least two places of type { \bullet }; moreover, in all assignments and conditions, one of the operands is always a constant. Therefore, the number of transition replicas in the expansion is relatively small.

The experimental results for the mutual exclusion algorithm are shown in Table 2. As one can see, our algorithm performs almost as well as the algorithm for low-level nets. Though there is some overhead when computing transition guards and more complicated final states, it is relatively small, because the most time-consuming operation is computing the possible extensions of a current prefix. Moreover, this overhead becomes relatively smaller as the size of the prefix grows (it is just 0.5% for the last example in the table).

	M-net		Expansion		J	Infoldin	Time[s]			
Ν	P	T	P	T	$\operatorname{Time}[s]$	B	E	$ E_{cut} $	LL	HL
2	52	50	58	88	<1	711	368	102	<1	<1
3	77	76	86	154	<1	23424	12026	4562	29	30
4	104	104	116	236	<1	736507	375983	167780	28772	28917

Table 2. Experimental results for the M-net system simulating the mutual exclusion algorithm.

begin

```
VAR \ x : \{0..N\} init 0;
VAR \ y : \{0..N\} init 0;
VAR \ b1, \ldots, bN : \{ false, true \} init false; 
proc PROCESS(const i : \{1..N\}, ref b : \{false, true\}) _max_ N
begin
    do
        < b' = {f true}>; \ < x' = i>;
        do
            < y \# 0 >; < b' = false >; < y = 0 >;
            \mathbf{exit}
        \Box \ < y = 0 >; \ < y' = i >;
            do
                 \langle x \# i \rangle; \langle b' = false\rangle; \langle b1 = false\rangle; \dots; \langle bN = false\rangle;
                 do
                     < y \# i >; < y = 0 >;
                     \mathbf{exit}
                 \Box \quad < y = i >;
                     critical section
                     < y' = 0 >; < b' = false >;
                     \mathbf{exit}
                 od;
                 \mathbf{exit}
            \Box \quad < x = i >;
                 critical section
                 < y' = 0 >; < b' = false >;
                 \mathbf{exit}
            od;
            \mathbf{exit}
        od ;
        repeat
    \mathbf{od}
end;
PROCESS(1, b1) \parallel \dots \parallel PROCESS(N, bN)
```

 \mathbf{end}

Fig. 6. The $B(PN)^2$ code for Lamport's mutual exclusion algorithm.

After the prefixes had been build, we verified using the efficient model checker described in [17, 18] that the M-net system is deadlock free, and that the places corresponding to the critical sections of the processes are mutually exclusive. This was done without recompiling the model checker, since our unfolding algorithm generates prefixes which are indistinguishable from those generated by a low-level net unfolder from the corresponding expansions of the Mnets.

It is worth noting that in this example partial-order methods have advantage over the state-space ones. In [16], this mutual exclusion algorithm was verified for N = 3 by building a reachability graph of the Petri net model and for N = 4 by applying symmetry reductions. We managed to verify the case N = 4 without applying symmetry reductions, using a PC with smaller memory (128M rather than 256M), for a net which was generated from a relatively high-level description $(B(PN)^2$ language) rather than built by hand. Moreover, as it was already mentioned, our specification was not optimal since we had to replicate parts of the code. In principle, it is also possible to apply partial-order methods together with symmetry reductions (see [6, 20]) to achieve even better results, but we have not implemented the combined method yet.

8 Conclusions

In this paper, we have defined branching processes and unfoldings of high-level Petri nets and proposed an algorithm which builds finite and complete prefixes of such unfoldings. We establish an important relation between the branching processes of a high-level net and those of its low-level expansion. This allows us to import results proven for branching processes of low-level nets rather than re-prove them. Among such results are the canonicity of the prefix for different cutting contexts, the usability of the total adequate order proposed in [9], and the parallel unfolding algorithm proposed in [15]. Our approach is conservative in the sense that all the verification tools employing the traditional unfoldings can be reused with such prefixes. The conducted experiments demonstrated that it is, on one hand, superior to the traditional approach on data-intensive application, and, on the other hand, has the same performance on control-intensive ones.

Acknowledgements

This research was supported by an ORS Awards Scheme grant ORS/C20/4, and by EPSRC grants GR/M99293 and GR/M94366 (MOVIE).

References

- E. Best, H. Fleischhack, W. Fraczak, R. Hopkins, H. Klaudel, and E. Pelz: A Class of Composable High Level Petri Nets. Proc. of *ATPN'1995*. G. DeMichelis and M. Diaz (Eds.). Springer LNCS 935 (1995) 103–120.
- E. Best, H. Fleischhack, W. Fraczak, R. Hopkins, H. Klaudel, and E. Pelz: An M-net Semantics of B(PN²). Proc. of International Workshop on Structures in Concurrency Theory (STRICT'95). J. Desel (Ed.). Berlin (1995) 85–100.
- 3. E. Best and B. Grahlmann: PEP. Documentation and User Guide. Version 1.4. Manual (1995).
- E. Best and B. Grahlmann: PEP more than a Petri Net Tool. Proc. of TACAS'96: Tools and Algorithms for the Construction and Analysis of Systems. Margaria T., Steffen B. (Eds.). Springer LNCS 1055 (1996) 397–401.
- 5. E. M. Clarke, O. Grumberg, and D. Peled: Model Checking. MIT Press (1999).
- J.-M. Couvreur, S. Grivet, and Denis Poitrenaud: Unfolding of Products of Symmetrical Petri Nets. Proc. of *ICATPN'2001*. J.-M. Colom and M. Koutny (Eds.). Springer LNCS 2075 (2001) 121–143.
- 7. J. Engelfriet: Branching processes of Petri Nets. Acta Inf. 28 (1991) 575-591.

- J. Esparza and S. Römer: An Unfolding Algorithm for Synchronous Products of Transition Systems. Proc. of CONCUR'99, Invited paper, LNCS 1664 (1999) 2–20.
- J. Esparza, S. Römer and W. Vogler: An Improvement of McMillan's Unfolding Algorithm. Proc. of TACAS'96. Margaria T., Steffen B. (Eds.). Springer LNCS 1055 (1996) 87–106. Full version to appear in Formal Methods in System Design.
- 10. H. Fleischhack, B. Grahlmann: A Petri Net Semantics for $B(PN^2)$ with Procedures which Allows Verification. Techn. Rep. 21, Universität Hildesheim (1996).
- H. Fleischhack, B. Grahlmann: A Petri Net Semantics for B(PN²) with Procedures. Proc. of Parallel and Distributed Software Engineering (PDSE'97). IEEE Computer Society Press. Boston (1997) 15–27.
- K. Heljanko: Deadlock and Reachability Checking with Finite Complete Prefixes. Techn. Rep. A56, Laboratory for Theoretical Computer Science, HUT, Espoo, Finland (1999).
- K. Heljanko: Minimizing Finite Complete Prefixes. Proc. of CS&P'99, Workshop Concurrency, Specification and Programming (1999) 83–95.
- K. Heljanko: Using Logic Programs with Stable Model Semantics to Solve Deadlock and Reachability Problems for 1-Safe Petri Nets. Fund. Inf. 37(3) (1999) 247–268.
- K. Heljanko, V. Khomenko and M. Koutny: Parallelisation of the Petri Net Unfolding Algorithm. Proc. of *TACAS'02*, Springer LNCS 2280 (2002) 371–385. Full version: Techn. Rep. CS-TR-733, Dept. of Comp. Sci., Univ. of Newcastle (2001).
- 16. K. Jensen: Colored Petri Nets. Basic Concepts, Analysis Methods and Practical Use. EATCS Monographs on Theoretical Computer Science (1992).
- V. Khomenko and M. Koutny: LP Deadlock Checking Using Partial Order Dependencies. Proc. of *CONCUR*'2000. C. Palamidessi (Ed.). Springer LNCS 1877 (2000) 410–425. Full version: Techn. Rep. CS-TR-711, Dept. of Comp. Sci., Univ. of Newcastle (2000).
- V. Khomenko and M. Koutny: Verification of Bounded Petri Nets Using Integer Programming. Techn. Rep. CS-TR-711, Department of Computing Science, University of Newcastle (2000).
- V. Khomenko and M. Koutny: Towards An Efficient Algorithm for Unfolding Petri Nets. Proc. of CONCUR'2001. P. G. Larsen and M. Nielsen (Eds.). Springer LNCS 2154 (2001) 366–380. Full version: Techn. Rep. CS-TR-726, Dept. of Comp. Sci., Univ. of Newcastle (2001).
- 20. V. Khomenko, M. Koutny, and V. Vogler: Canonical Prefixes of Petri Net Unfoldings. Proc. of CAV'02, LNCS (2002) to appear. Full version: Techn. Rep. CS-TR-741, Dept. of Comp. Sci., Univ. of Newcastle (2001). URL: http://www.cs.ncl.ac.uk/research/trs/lists/2001.html
- V. Khomenko, M. Koutny and A. Yakovlev: Detecting State Coding Conflicts in STGs Using Integer Programming. Proc. of *DATE'02*. C. D. Kloos and J. Franca (Eds.). IEEE Computer Society Press (2002) 338–345. Full version: Techn. Rep. CS-TR-736, Dept. of Comp. Sci., Univ. of Newcastle (2001).
- 22. V. E. Kozura: Unfolding of Colored Petri Nets. Techn. Rep. 80, A. P. Ershov Institute of Informatics Systems (2000).
- L. Lamport: A Fast Mutual Exclusion Algorithm. ACM Transactions on Computer Systems 5(1) (1987) 1–11.
- K. L. McMillan: Using Unfoldings to Avoid State Explosion Problem in the Verification of Asynchronous Circuits. Proc. of 4th CAV. G. von Bochmann and D. K. Probst (Eds.). LNCS 663 (1992) 164–174.
- 25. K. L. McMillan: Symbolic Model Checking. PhD thesis, CMU-CS-92-131 (1992).
- S. Melzer and S. Römer: Deadlock Checking Using Net Unfoldings. Proc. of Computer Aided Verification (CAV'97). O. Grumberg (Ed.). Springer LNCS 1254 (1997) 352–363.
- 27. S. Römer: Entwicklung und Implementierung von Verifikationstechniken auf der Basis von Netzentfaltungen. PhD thesis, Technische Universitat Munchen (2000).
- A. Semenov: Verification and Synthesis of Asynchronous Control Circuits Using Petri Net Unfolding. PhD Thesis, University of Newcastle upon Tyne (1997).

¹⁸ V. Khomenko, M. Koutny