An Improvement of McMillan’s Unfolding Algorithm

Javier Esparza and Stefan Römer* (esparza|roemer}@in.tum.de)  
Institut für Informatik, Technische Universität München, Arcisstr. 21, D-80333 München, Germany

Walter Vogler† (walter.vogler@informatik.uni-augsburg.de)  
Institut für Informatik, Universität Augsburg, Universitätsstr. 14, D-86159 Augsburg, Germany

Abstract. McMillan has recently proposed a new technique to avoid the state explosion problem in the verification of systems modelled with finite-state Petri nets. The technique requires to construct a finite initial part of the unfolding of the net. McMillan’s algorithm for this task may yield initial parts that are larger than necessary (exponentially larger in the worst case). We present a refinement of the algorithm which overcomes this problem.

Keywords: unfolding, partial-order semantics, Petri nets

1. Introduction

In a seminal paper [12], McMillan has proposed a new technique to avoid the state explosion problem in the verification of systems modelled with finite-state Petri nets. The technique is based on the concept of net unfoldings, a well known partial-order semantics of Petri nets introduced in [15], and later described in more detail in [4] under the name of branching processes. The unfolding of a net is another net, usually infinite but with a simpler structure. McMillan proposes an algorithm for the construction of a finite initial part of the unfolding which contains full information about the reachable states. We call such an initial part a finite complete prefix. He then shows how to use these prefixes for deadlock detection.

The unfolding technique has later been applied to other verification problems. In [8, 9, 13] it is used to check relevant properties of speed independent circuits. In [5], an unfolding-based model checking algorithm for a simple branching time logic is proposed.

Although McMillan’s algorithm is simple and elegant, it sometimes generates prefixes much larger than necessary. In some cases a minimal complete prefix has size \(O(n)\) (where \(n\) is the size of the Petri net),

* Partially supported by the Teilprojekt A3 SAM of the Sonderforschungsbereich 342 “Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen”.
† Work on this paper was partially supported by the DFG (Project “Halbordnungstests”).

while the algorithm generates a prefix of size $O(2^n)$. In this paper we provide an algorithm which generates a minimal complete prefix (in a certain sense to be defined). The prefix is always smaller than or as large as the prefix generated with the old algorithm, and it is never larger (up to small constant) than the state space of the Petri net.

The paper is organised as follows. Section 2 and 3 contain basic definitions about Petri nets and branching processes, respectively. In Section 4 we show that McMillan's algorithm is just an element of a whole family of algorithms for the construction of finite complete prefixes. The algorithms of this family depend on the choice of a so-called adequate order; this is a partial order on the configurations of a branching process. In Section 5 we improve McMillan's algorithm by exhibiting a finer adequate order. In Section 6 we define for 1-safe net systems an adequate order which is total. Section 7 extends this idea to $n$-bounded systems; for the representation of the process net another partial-order semantics is used, so-called executions. Finally, in Section 8 we present aspects of an efficient implementation of the algorithms, accompanied by experimental results.

## 2. Petri nets

A net is a triple $(S, T, W)$, where $S$ and $T$ are disjoint sets of places (Stellen in Petri’s original notation) and transitions, respectively, and $W$ is a function $(S \times T) \cup (T \times S) \rightarrow \{0, 1\}$. Places and transitions are generically called nodes. If $W(x, y) = 1$ then we say that there is an arc from $x$ to $y$. Thus, a net can be considered as a directed graph. A path in such a graph is — as usual — a nonempty sequence of nodes without repetitions such there is an arc from each node to the following (if there is one).

The preset of a node $x$, denoted by $\cdot x$, is the set $\{y \in S \cup T \mid W(y, x) = 1\}$. The postset of $x$, denoted by $x^\cdot$, is the set $\{y \in S \cup T \mid W(x, y) = 1\}$.

A marking of a net $(S, T, W)$ is a mapping $M : S \rightarrow N$ (where $N$ denotes the natural numbers including 0). We identify $M$ with the multiset containing $M(s)$ copies of $s$ for every $s \in S$. For instance, if $S = \{s_1, s_2\}$ and $M(s_1) = 1, M(s_2) = 2$, we write $M = \{s_1, s_2, s_2\}$.

A 4-tuple $\Sigma = (S, T, W, M_0)$ is a net system if $(S, T, W)$ is a net and $M_0$ is a marking of $(S, T, W)$ (called the initial marking of $\Sigma$).

**General assumptions** In this paper we consider only nets in which every transition has a nonempty preset and a nonempty postset. We further assume that all net systems are finite.
A marking $M$ enables a transition $t$ if it marks each place $s \in \bullet t$ with a token, i.e. if $M(s) > 0$ for each $s \in \bullet t$. If $t$ is enabled at $M$, then it can fire or occur, and its occurrence leads to a new marking $M'$, obtained by removing a token from each place in the preset of $t$, and adding a token to each place in its postset; formally, $M'(s) = M(s) - W(s,t) + W(t,s)$ for every place $s$. For each transition $t$ the relation $\xrightarrow{t}$ is defined as follows: $M \xrightarrow{t} M'$ if $t$ is enabled at $M$ and its occurrence leads to $M'$.

A sequence of transitions $\sigma = t_1 t_2 \ldots t_n$ is an occurrence sequence if there exist markings $M_1, M_2, \ldots, M_n$ such that

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots \xrightarrow{t_n} M_n$$

$M_n$ is the marking reached by the occurrence of $\sigma$, also denoted by $M_0 \xrightarrow{\sigma} M_n$. $M$ is a reachable marking if there exists an occurrence sequence $\sigma$ such that $M_0 \xrightarrow{\sigma} M$. The reachability graph of a net system is the labelled graph having the reachable markings of the system as nodes, and the $\xrightarrow{t}$ relations (more precisely, their restriction to the set of reachable markings) as arcs.

A marking $M$ of a net is $n$-safe if $M(s) \leq n$ for every place $s$. A net system $\Sigma$ is $n$-safe if all its reachable markings are $n$-safe, and safe if it is $n$-safe for some number $n$.

**Labelled nets** A labelled net is a pair $(N, l)$ (also represented as a 4-tuple $(S, T, W, l)$), where $N$ is a net and $l$ is a labelling function that assigns to each node $x$ of $N$ a label $l(x)$ taken from some set. Notice that different nodes can carry the same label. For each label $a$ we define the relation $\xrightarrow{a}$ between markings as follows: $M \xrightarrow{a} M'$ if $M \xrightarrow{t} M'$ for some transition $t$ such that $l(t) = a$. The reachability graph of a labelled net system is the labelled graph having the reachable markings of the system as nodes, and the $\xrightarrow{a}$ relations (more precisely, their restriction to the set of reachable markings) as arcs.

### 3. Branching processes

In this section we describe branching processes, a partial-order semantics of Petri nets. Before giving any formal definitions, we give some intuitive ideas.

Consider a directed graph $G$ with a root node. It is well-known that such a graph can be “unfolded” into a labelled tree (whose nodes are the paths in $G$ starting at the root). The nodes of the tree are labelled with the nodes of the graph (i.e. with the last node of the respective path). The unfolding process can be stopped at different
times yielding different trees, but there is a unique labelled tree, usually infinite, obtained by unfolding “as much as possible”. This labelled tree is called the unfolding of the graph.

In the same way, net systems can be “unfolded” into labelled occurrence nets, a subclass of nets with a particularly simple, tree-like structure. The nodes of the occurrence net are labelled with the places and transitions of the net. The labelled occurrence nets obtained through unfolding of a net are called branching processes. The unfolding process can be stopped at different times yielding different branching processes, but there is a unique, usually infinite, branching process obtained by unfolding “as much as possible”. This branching process is called the unfolding of the net system.

In the next two subsections we formally define occurrence nets, branching processes and the unfolding.

3.1. Occurrence nets

First of all, we need to define the causal, conflict, and concurrency relations between nodes of a net.

- Two nodes $x$ and $y$ are in causal relation, denoted by $x < y$, if the net contains a path with at least one arc leading from $x$ to $y$.

- $x$ and $y$ are in conflict relation, or just in conflict, denoted by $x\#y$, if the net contains two paths $st_1 \ldots x_1$ and $st_2 \ldots x_2$ starting at the same place $s$, and such that $t_1 \neq t_2$. In words, $x_1$ and $x_2$ are in conflict if the net contains two paths leading to $x_1$ and $x_2$ which start at the same place and immediately diverge (although later on they can converge again).

- $x$ and $y$ are in concurrency relation, denoted by $x \bowtie y$, if neither $x < y$ nor $y < x$ nor $x\#y$.

An occurrence net is a net $O = (B, E, F)$ such that:

1. $|\bullet b| \leq 1$ for every $b \in B$;
2. $O$ is acyclic, or, equivalently, the causal relation is a partial order;
3. $O$ is finitely preceded, i.e., for every $x \in B \cup E$, the set of elements $y \in B \cup E$ such that $y < x$ is finite;
4. no element is in conflict with itself.

It is easy to see that any two nodes of an occurrence net are either in causal, conflict, or concurrency relation.
The elements of $B$ and $E$ are usually called *conditions* (*Bedingungen* in Petri’s original notation) and *events*, respectively.

$\text{Min}(O)$ denotes the set of minimal elements of $B \cup E$ with respect to the causal relation, i.e., the elements that have an empty preset. Since we only consider nets in which every transition has a nonempty preset, the elements of $\text{Min}(O)$ are conditions.

3.2. **Branching processes**

Those labelled occurrence nets obtained from net systems by “unfolding” are called branching processes, and have the following formal definition:

A branching process of a net system $\Sigma = (S, T, W, M_0)$ is a labelled occurrence net $\beta = (O, p) = (B, E, F, p)$ where the labelling function $p$ satisfies the following properties:

(i) $p(B) \subseteq S$ and $p(E) \subseteq T$

($p$ preserves the nature of nodes);

(ii) for every $e \in E$, the restriction of $p$ to $\cdot e$ is a bijection between $\cdot e$ (in $\Sigma$) and $\cdot p(e)$ (in $\beta$), and similarly for $e^\ast$ and $p(e)^\ast$

($p$ preserves the environments of transitions);

(iii) the restriction of $p$ to $\text{Min}(O)$ is a bijection between $\text{Min}(O)$ and $M_0$

($\beta$ “starts” at $M_0$);

(iv) for every $e_1, e_2 \in E$, if $\cdot e_1 = \cdot e_2$ and $p(e_1) = p(e_2)$ then $e_1 = e_2$

($\beta$ does not duplicate the transitions of $\Sigma$).

Figure 1 shows a 1-safe net system (part (a)), and two of its branching processes (parts (b) and (c)).

Branching processes differ on “how much they unfold”. It is natural to introduce a prefix relation formalising the idea “a branching process unfolds less than another”.

Let $\beta' = (O', p')$ and $\beta = (O, p)$ be two branching processes of a net system. $\beta'$ is a prefix of $\beta$ if $O'$ is a subnet of $O$ satisfying

- $\text{Min}(O)$ belongs to $O'$;
- if a condition $b$ belongs to $O'$, then its input event $e \in \cdot b$ in $O$ also belongs to $O'$ (if it exists); and
- if an event $e$ belongs to $O'$, then its input and output conditions $\cdot e \cup e^\ast$ in $O$ also belong to $O'$,
and $p'$ is the restriction of $p$ to $O'$.

It is shown in [4] that a net system has a unique maximal branching process with respect to the prefix relation. To be precise, this process is unique up to isomorphism, i.e., up to renaming of the conditions and the events. This is the branching process that “unfolds as much as possible”. We call it the unfolding of the system. The unfolding of the 1-safe system of Figure 1 is infinite.

A branching process has a natural initial marking, namely the marking that puts one token in each minimal condition, and no tokens anywhere else. When we talk of the reachable markings and the reachability graph of a branching process, we refer to the natural initial marking.

With this, we can formulate how an unfolding describes the behaviour of a net as follows: Let $\Sigma$ be a net system, and let $\beta$ be its unfolding. The reachability graphs of $\Sigma$ and $\beta$ have isomorphic unfoldings (as graphs as described above). More in detail, the reachable
markings of $\Sigma$ are those $p(M)$ where $M$ is a reachable marking of $\beta$; for a reachable marking $M$ of $\beta$ and a marking $M''$ and a transition $t$ of $\Sigma$, there are $M'$ and $e$ with $p(M') = M''$, $p(e) = t$ and $M \xrightarrow{t} M'$ in $\beta$ if and only if $p(M) \xrightarrow{t} M''$ in $\Sigma$.

3.3. Configurations and cuts

In order to work with branching processes we need the notions of configuration and cut.

A configuration $C$ of a branching process is a set of events satisfying the following two conditions:

- $e \in C \Rightarrow \forall e' \leq e: e' \in C$ (C is causally closed).
- $\forall e, e' \in C: \neg(e\#e')$ (C is conflict-free).

The set of events $\{1,3,4,6\}$ in Figure 1(b) is a configuration, but the sets $\{3,4\}$ (not causally closed) and $\{1,2\}$ (non conflict-free) are not. Intuitively, a configuration is a set of events 'firable' from the natural initial marking, i.e., there is a firing sequence from the natural initial marking in which each event of the set occurs exactly once. For $\{1,3,4,6\}$ we can first fire 1 and 3, then 4 and then 6, but neither $\{3,4\}$ nor $\{1,2\}$ are firable from the natural initial marking.

A set of conditions of a branching process is a co-set if its elements are pairwise in co relation. A maximal co-set with respect to set inclusion is called a cut. In Figure 1(b), the set containing the (unique) output condition of event 6 and the output condition of event 4 labelled by $s_7$ is a cut.

A marking $M$ of a system $\Sigma$ is represented in a branching process $\beta = (O,p)$ of $\Sigma$ if $\beta$ contains a cut $c$ such that, for each place $s$ of $\Sigma$, $c$ contains exactly $M(s)$ conditions $b$ with $p(b) = s$. For instance, the marking $\{s_1, s_7\}$ is represented in the branching process of Figure 1(b) because of the cut mentioned above. It is easy to prove using results of [1, 4] that every marking represented in a branching process is reachable, and that every reachable marking is represented in the unfolding of the net system. Observe in particular that $\{s_1, s_7\}$ is reachable.

Finite configurations and cuts are tightly related. Let $C$ be a finite configuration of a branching process $\beta = (O,p)$. Then the co-set $\text{Cut}(C)$, defined below, is a cut:

$$\text{Cut}(C) = (\text{Min}(O) \cup C^*) \setminus ^* C.$$ 

In particular, given a configuration $C$ the set of places $\text{Cut}(C)$ represents a reachable marking, which we denote by $\text{Mark}(C)$. Loosely speaking, $\text{Mark}(C)$ is the marking we reach by firing the configuration
C. In the branching process of Figure 1(b) we have \( \text{Mark}(\{1, 3, 4, 6\}) = \{s_1, s_7\} \).

4. An algorithm for the construction of a complete finite prefix

4.1. Constructing the unfolding

We give an algorithm for the construction of the unfolding of a net system. First of all, let us describe a suitable data structure for the representation of branching processes.

We implement a branching process of a net system \( \Sigma \) as a set \( \{n_1, \ldots, n_k\} \) of nodes. A node is either a condition or an event. A condition is a record containing two fields: a place of \( \Sigma \), and a pointer to an event (the unique input event of the condition), or to NIL, in case the condition has an empty preset. In the pseudocode description of our algorithms we represent a condition as a pair \((s, e)\) or \((s, \emptyset)\). An event is also a record with two fields: a transition of \( \Sigma \), and a list of pointers to conditions (the input conditions of the event). In pseudocode we represent an event as a pair \((t, X)\).

Notice that the flow relation and the labelling function of a branching process are already encoded in its set of nodes. How to express the notions of causal relation, configuration or cut in terms of this data structure is left to the reader.

We need the notion of “events that can be added to a given branching process”. Let \( t \) be a transition of \( \Sigma \) with output places \( s_1, \ldots, s_n \). Formally, a pair \( e = (t, X) \) is a possible extension of a branching process \( \{n_1, \ldots, n_k\} \) if \( \{n_1, \ldots, n_k, e, (s_1, e), \ldots, (s_n, e)\} \) is also a branching process. \( PE(\beta) \) denotes the set of possible extensions of a branching process \( \beta \).

The following characterisation follows easily from the definitions:

**Proposition 4.1.**

Let \( \beta \) be a branching process of a net system \( \Sigma \). The possible extensions of \( \beta \) are the pairs \((t, X)\), where \( X \) is a co-set of conditions of \( \beta \) and \( t \) is a transition of \( \Sigma \) such that

1. \( p(X) = \bullet t \), and
2. \((t, X)\) does not already belong to \( \beta \).

The algorithm for the construction of the unfolding starts with the branching process having the conditions corresponding to the initial
marking of Σ and no events. New events are added one at a time together with their output conditions. Observe that the initial marking $M_0$ is a multiset, and so a place can appear several times in it. If $M_0(s) = k$, then $Unf$ contains $k$ minimal conditions labelled by $s$, i.e., $k$ elements $(s, \emptyset)$.

**ALGORITHM 4.2. The unfolding algorithm**

**input:** A net system $\Sigma = (N, M_0)$, where $M_0 = \{s_1, \ldots, s_n\}$.

**output:** The unfolding $Unf$ of $\Sigma$.

**begin**

$Unf := \{(s_1, \emptyset), \ldots, (s_n, \emptyset)\}$;

$pe := PE(Unf)$;

**while** $pe \neq \emptyset$ **do**

- add to $Unf$ an event $e = (t, X)$ of $pe$ and a condition $(s, e)$ for every output place $s$ of $t$;

- $pe := PE(Unf)$

**endwhile**

**end**

[4.2]

The algorithm does not necessarily terminate. In fact, it terminates if and only if the input system $\Sigma$ does not have any infinite occurrence sequence. It is correct only under the fairness assumption that every event added to $pe$ is eventually chosen to extend $Unf$ (the correctness proof follows easily from the definitions and from the results of [4]).

**CONSTRUCTING A FINITE COMPLETE PREFIX**

We say that a branching process $\beta$ of a net system $\Sigma$ is complete if for every reachable marking $M$ there exists a configuration $C$ in $\beta$ such that:

- $Mark(C) = M$ (i.e., $M$ is represented in $\beta$), and

- for every transition $t$ enabled by $M$ there exists a configuration $C \cup \{e\}$ such that $e \notin C$ and $e$ is labelled by $t$.

The unfolding of a net system is always complete. A complete prefix contains as much information as the unfolding, in the sense that we can construct the unfolding from it as the least fixpoint of a suitable operation. This property does not hold if we only require every reachable marking to be represented. For instance, the net system of Figure 2(a) has Figure 2(b) as unfolding. Figure 2(c) shows a prefix of the unfolding in which every reachable marking is represented. The prefix has lost the information indicating that $t_2$ can occur from the initial marking. Observe that the prefix is not complete.
Figure 2. A 1-safe net system (a), its unfolding (b), and a prefix (c)

Since an $n$-safe net system has only finitely many reachable markings, its unfolding contains at least one complete finite prefix. We transform the algorithm above into a new one whose output is such a prefix. The key idea (due to McMillan) is to identify certain events, called cut-off events, at which the construction can be stopped without losing information; stopped means that no new events causally related to the cut-off event are added.

We start with some Given a configuration $C$ of a branching process $\beta = (O, p)$, we define $\uparrow C$ as the pair $(O', p')$, where $O'$ is the unique subnet of $O$ whose set of nodes is \{x | x \notin C \cup *C \land \forall y \in C : -(x \# y)\} and $p'$ is the restriction of $p$ to the nodes of $O'$. Loosely speaking, $\uparrow C$ is the part of $\beta$ lying “after” $C$.

The following result can be easily proved, directly from the definitions:

PROPOSITION 4.3

If $\beta$ is a branching process of $(N, M_0)$ and $C$ is a configuration of $\beta$, then $\uparrow C$ is a branching process of $(N, Mark(C))$. Moreover, if $\beta$ is the unfolding of $(N, M_0)$, then $\uparrow C$ is the unfolding of $(N, Mark(C))$ (up to isomorphism). ■ 4.3

Given a configuration $C$, we denote by $C \oplus E$ the fact that $C \cup E$ is a configuration such that $C \cap E = \emptyset$. We say that $C \oplus E$ is an extension of $C$, and that $E$ is a suffix of $C$. Obviously, for a configuration $C'$, if $C \subset C'$ then there is a nonempty suffix $E$ of $C$ such that $C \oplus E = C'$.

Now, let $C_1$ and $C_2$ be two finite configurations leading to the same marking, i.e. $Mark(C_1) = M = Mark(C_2)$. By Proposition 4.3 $\uparrow C_1$ and $\uparrow C_2$ are isomorphic to the unfolding of $(N, M)$, and so they are isomorphic to each other. Let $I^2_1$ be an isomorphism between $\uparrow C_1$ and $\uparrow C_2$. This isomorphism induces a mapping from the finite extensions of $C_1$ onto the finite extensions of $C_2$: it maps $C_1 \oplus E$ onto $C_2 \oplus I^2_1(E)$.
Figure 3. A 1-safe net system (a) and a prefix of its unfolding (b)

We can now introduce the three basic notions needed by the algorithm: adequate order, local configuration, and cut-off event. We present the formal definitions together with the intuition behind them.

McMillan’s idea [12] is to attach to each event $e$ added by the unfolding algorithm a reachable marking of $\Sigma$. For this, we first compute the local configuration $[e]$ of $e$, defined below, and then we associate to $e$ the marking $\text{Mark}([e])$.

**DEFINITION 4.4. Local configuration**

The local configuration $[e]$ of an event $e$ of a branching process is the set of events $e'$ such that $e' \leq e$.\(^1\)

Now, assume that a new event $e$ is added to the current branching process, such that some event $e'$ added before satisfies $\text{Mark}([e]) = \text{Mark}([e'])$. We know that $\uparrow [e]$ and $\uparrow [e']$ are isomorphic, and so it is sufficient to pursue the construction of one of the two. Intuitively, it seems possible to mark $e$ as “cut-off” event, and so stop the construction of $\uparrow [e]$. However, the following example (independently found by McMillan and one of the authors) shows that this strategy is incorrect. Consider the 1-safe net system of Figure 3(a).

The marking $\{s_{12}\}$ is reachable. However, we can generate the prefix of Figure 3(b), in which this marking is not represented. The names

\(^1\) It is immediate to prove that $[e]$ is a configuration.
of the events are numbers which indicate the order in which they are added to the prefix. The events 8 and 10 are marked as "cut-off" events, because their corresponding markings \(\{s_7, s_9, s_{10}\}\) and \(\{s_6, s_8, s_{11}\}\) are also the markings corresponding to the events 7 and 9, respectively. Although no events can be added, the prefix is not complete, because \(\{s_{12}\}\) is not represented in it.

The choice between \([e]\) and \([e']\) is made on the basis of a partial order. We show below that all orders satisfying three properties make the correctness proof work, i.e., lead to finite complete prefixes. We call these orders **adequate**.

**DEFINITION 4.5. Adequate order**

A partial order \(\prec\) on the finite configurations of the unfolding of a net system is an adequate order if:

- \(\prec\) is well-founded,
- \(C_1 \subset C_2\) implies \(C_1 \prec C_2\), and
- \(\prec\) is preserved by finite extensions; if \(C_1 \prec C_2\) and \(\text{Mark}(C_1) = \text{Mark}(C_2)\), then the isomorphism \(I_1^2\) from above satisfies \(C_1 \oplus E \prec C_2 \oplus I_1^2(E)\) for all finite extensions \(C_1 \oplus E\) of \(C_1\). \(\blacksquare\) 4.5

In [12] \([e']\) is smaller than \([e]\) if it contains less events, i.e. if \(|[e']|\) < \(|[e]|\). It is easy to see that this order is adequate. We can now formally define cut-off events with respect to an adequate order.

**DEFINITION 4.6. Cut-off event**

Let \(\prec\) be an adequate order on the configurations of the unfolding of a net system. Let \(\beta\) be a prefix of the unfolding containing an event \(e\). The event \(e\) is a cut-off event of \(\beta\) (with respect to \(\prec\)) if \(\beta\) contains a local configuration \([e']\) such that

- \((a)\) \(\text{Mark}([e]) = \text{Mark}([e'])\), and
- \((b)\) \([e'] \prec [e]\). \(\blacksquare\) 4.6

The new algorithm is in fact a family of algorithms: each adequate order \(\prec\) leads to a different algorithm. Events are respecting the \(\prec\) order, and cut-offs are identified and marked. The algorithm terminates when no event can be added.
ALGORITHM 4.7. *The complete finite prefix algorithm*

\textbf{Input:} An \( n \)-safe net system \( \Sigma = (N, M_0) \),
where \( M_0 = \{ s_1, \ldots, s_k \} \).

\textbf{Output:} A complete finite prefix \( \text{Fin} \) of \( \text{Unf} \).

\textbf{Begin}
\begin{align*}
\text{Fin} & := (s_1, \emptyset), \ldots, (s_k, \emptyset); \\
\text{pe} & := \text{PE}(\text{Fin}); \\
\text{cut-off} & := \emptyset;
\end{align*}
\textbf{While} \( \text{pe} \neq \emptyset \) \textbf{Do}
\begin{align*}
\text{choose an event } e = (t, X) \text{ in } \text{pe} \text{ such that } [e] \text{ is minimal} \\
\text{ with respect to } \prec; \\
\text{if } [e] \cap \text{cut-off} = \emptyset \text{ then} \\
& \text{append to } \text{Fin} \text{ the event } e \text{ and a condition } (s, e) \\
& \text{for every output place } s \text{ of } t; \\
\text{pe} & := \text{PE}(\text{Fin}); \\
\text{if } e \text{ is a cut-off event of } \text{Fin} \text{ then} \\
\text{cut-off} & := \text{cut-off} \cup \{ e \}
\end{align*}
\textbf{End}
\textbf{While}
\textbf{End} \qquad \blacksquare 4.7

The correctness of Algorithm 4.7 is proved in the next two propositions.

**Proposition 4.8.**

\( \text{Fin} \) is finite.

\textbf{Proof}: Given an event \( e \) of \( \text{Fin} \), define the depth of \( e \) as the length of a longest chain of events \( e_1 < e_2 < \ldots < e \); the depth of \( e \) is denoted by \( d(e) \). We prove the following results:

1. For every event \( e \) of \( \text{Fin} \), \( d(e) \leq n + 1 \), where \( n \) is the number of reachable markings of \( \Sigma \).

   Since cuts correspond to reachable markings, every chain of events \( e_1 < e_2 < \ldots < e_n < e_{n+1} \) of \( \text{Unf} \) contains two events \( e_i, e_j, i < j \), such that \( \text{Mark}([e_i]) = \text{Mark}([e_j]) \). Since \( [e_i] \subseteq [e_j] \), we have \( [e_i] \prec [e_j] \), and therefore \( [e_j] \) is a cut-off event of \( \text{Unf} \). Should the finite prefix algorithm generate \( e_j \), then it has generated \( e_i \) before and \( e_j \) is recognized as a cut-off event of \( \text{Fin} \), too.
(2) For every event $e$ of $\text{Fin}$, the sets $e^*$ and $e^p$ are finite.

By the definition of prefix, there is a bijection between $e^*$ and $p(e)^*$, where $p$ denotes the labelling function of $\text{Fin}$, and similarly for $e^*$ and $p(e)^*$. The result follows from the finiteness of $N$.

(3) For every $k \geq 0$, $\text{Fin}$ contains only finitely many events $e$ such that $d(e) \leq k$.

By complete induction on $k$. The base case, $k = 0$, is trivial. Let $E_k$ be the set of events of depth at most $k$. We prove that if $E_k$ is finite then $E_{k+1}$ is finite.

By (2) and the induction hypothesis, $E_k^*$ is finite. Since $e^*E_{k+1} \subseteq E_k^* \cup \text{Min}(\text{Fin})$, we get by property (iv) in the definition of a branching process that $E_{k+1}$ is finite.

It follows from (1) and (3) that $\text{Fin}$ only contains finitely many events. By (2) it contains only finitely many conditions. ■

4.8

PROPOSITION 4.9.

$\text{Fin}$ is complete.

Proof: (a) Every reachable marking of $\Sigma$ is represented in $\text{Fin}$.

Let $M$ be an arbitrary reachable marking of $\Sigma$. There exists a configuration $C_1$ of $\text{Unf}$ such that $\text{Mark}(C_1) = M$. If $C_1$ is not a configuration of $\text{Fin}$, then it contains some cut-off event $e_1$, and so $C_1 = [e_1] \oplus E_1$ for some set of events $E_1$. By the definition of a cut-off event, there exists a local configuration $[e_2]$ such that $[e_2] \prec [e_1]$ and $\text{Mark}([e_2]) = \text{Mark}([e_1])$. Consider the configuration $C_2 = [e_2] \oplus \mathcal{I}_1^2(E_1)$. Since $\prec$ is preserved by finite extensions, we have $C_2 \prec C_1$. Moreover, $\text{Mark}(C_2) = M$. If $C_2$ is not a configuration of $\text{Fin}$, then we can iterate the procedure and find a configuration $C_3$ such that $C_3 \prec C_2$ and $\text{Mark}(C_3) = M$. The procedure cannot be iterated infinitely often because $\prec$ is well-founded. Therefore, it terminates in a configuration of $\text{Fin}$.

(b) If a transition $t$ can occur in $\Sigma$, then $\text{Fin}$ contains an event labelled by $t$.

If $t$ occurs in $\Sigma$, then some reachable marking $M$ enables $t$. The marking $M$ is represented in $\text{Fin}$. Let $C$ be a minimal configuration with respect to $\prec$ such that $\text{Mark}(C) = M$. If
Figure 4. A Petri net and its unfolding

If \( C \) contains some cut-off event, then we can apply the arguments of (a) to conclude that \( \text{Fin} \) contains a configuration \( C' \prec C \) such that \( \text{Mark}(C') = M \). This contradicts the minimality of \( C \). So \( C \) contains no cut-off events, and therefore \( \text{Fin} \) also contains a configuration \( C \oplus \{e\} \) such that \( e \) is labelled by \( t \).

Notice that the adequacy of an order is a sufficient but not necessary condition for the correctness of Algorithm 4.7. For example, a look at the proof of Proposition 4.9 reveals that the preservation of the order by finite extensions is only applied to local configurations. So in the third property of Definition 4.5 \( C_1 \) and \( C_2 \) could be replaced by \([e_1]\) and \([e_2]\).

5. An adequate order for arbitrary net systems

As we mentioned in the introduction, McMillan’s algorithm may be inefficient in some cases. An extreme example due to Kishinevsky and Taubin is the family of systems on the left of Figure 4. While a minimal complete prefix has size \( O(n) \) in the size of the system (see the dashed line on the right of the figure), the branching process generated by McMillan’s algorithm has size \( O(2^n) \). The reason is that for every marking \( M \) all the local configurations \([e]\) satisfying \( \text{Mark}([e]) = M \) have the same size, and therefore there exist no cut-off events with respect to McMillan’s order. \(^2\)

\(^2\) It is not important that the nets in Figure 4 are not simple, i.e. have transitions with the same pre- and postset; we could also replace each transition by a sequence of two transitions to obtain a suitable family of simple nets.
Our parametric presentation of Algorithm 4.7 suggests how to improve this: we find a new adequate order that refines McMillan’s order. Such an order induces a weaker notion of cut-off event. More precisely, every cut-off event with respect to McMillan’s order is also a cut-off event with respect to the new order, but maybe not the other way round. Therefore, the instance of Algorithm 4.7 which uses the new order generates at least as many cut-off events as McMillan’s instance, and maybe more. In the latter case, Algorithm 4.7 generates a smaller prefix.

Let \( \Sigma = (N, M_0) \) be a net system, and let \( \ll \) be an arbitrary total order on the transitions of \( \Sigma \). Given a set \( E \) of events, let \( \varphi(E) \) be that sequence of transitions which is ordered according to \( \ll \), and contains each transition \( t \) as often as there are events in \( E \) with label \( t \). For instance, if we have \( t_1 \ll t_2 \ll t_3 \ll t_4 \), and the set \( E \) contains four events labelled by \( t_1, t_2, t_2, \) and \( t_3 \), then \( \varphi(E) = t_1t_2t_2t_3 \). (\( \varphi \) is somewhat similar to a Parikh-vector.) We say \( \varphi(E_1) \ll \varphi(E_2) \) if \( \varphi(E_1) \) is lexicographically smaller than \( \varphi(E_2) \) with respect to the order \( \ll \).

**DEFINITION 5.1.** Partial order \( \ll_E \)

Let \( C_1 \) and \( C_2 \) be two configurations of the unfolding of a net system. \( C_1 \ll_E C_2 \) holds if

- \( |C_1| < |C_2| \), or
- \( |C_1| = |C_2| \), and \( \varphi(C_1) \ll \varphi(C_2) \).

**THEOREM 5.2.**

Let \( \beta \) be the unfolding of a net system. \( \ll_E \) is an adequate order on the finite configurations of \( \beta \).

**Proof:** It is easy to show that \( \ll_E \) is a well-founded partial order implied by inclusion. To show that \( \ll_E \) is preserved by finite extensions, assume \( C_1 \ll_E C_2 \). For every finite extension \( C_1 \oplus E \) of \( C_1 \) we have \( |E| = |I_E^2(E)| \), since \( I_E^2 \) is a bijection, and \( \varphi(E) = \varphi(I_E^2(E)) \), since \( I_E^2 \) preserves the labelling of events. If \( |C_1| < |C_2| \), then \( |C_1 \oplus E| < |C_2 \oplus I_{C_1}^2(E)| \). If \( \varphi(C_1) \ll \varphi(C_2) \), then by the properties of the lexicographic order \( \varphi(C_1 \oplus E) \ll \varphi(C_2 \oplus I_{C_1}^2(E)) \).

If we take \( \ll_E \) as adequate order, the complete prefix generated by Algorithm 4.7 for the net system of Figure 4 is the minimal one corresponding to the dotted line.
The question is whether there can be other examples in which $\prec_E$ performs poorly. We would like to have an adequate order which guarantees that the complete prefix is at most as large as the reachability graph. A slightly weaker guarantee is provided by total adequate orders. In this case, whenever an event $e$ is generated after some other event $e'$ such that $\text{Mark}([e]) = \text{Mark}([e'])$, we have $[e'] \prec [e]$ (because events are generated in accordance with the total order $\prec$), and so $e$ is marked as a cut-off event. So total adequate orders have the following important property:

**PROPOSITION 5.3.** A property of total adequate orders

Let $\prec$ be an adequate total order, and let $\text{Fin}$ be the output of Algorithm 4.7 on input $\Sigma$. The number of non-cut-off events of $\text{Fin}$ does not exceed the number of reachable markings of $\Sigma$. □ 5.3

Unfortunately, $\prec_E$ is not total. Consider the net system on the left of Figure 5, and its unfolding on the right of the same figure. The configurations $C_1 = \{e_1, e_3\}$ and $C_2 = \{e_2, e_4\}$ have size 2, and we have $\varphi(C_1) = t_1 t_2 = \varphi(C_2)$ (assuming $t_1 \ll t_2$). So neither $C_1 \prec_E C_2$ nor $C_2 \prec_E C_1$.

The existence of a total adequate order for arbitrary net systems is an open problem. However, in the next section we provide a total adequate order $\prec_F$ for 1-safe systems, the most relevant case in practice. Moreover, in Section 7 we show that the unfolding of a net system can also be defined in another way; with this new definition the order $\prec_F$ is total for arbitrary net systems, and Theorem 5.3 holds.
6. A total order for 1-safe systems

In the sequel, let $\Sigma = (N, M_0)$ be a fixed net system, and let $\ll$ be an arbitrary total order on the transitions of $\Sigma$. We first introduce the Foata normal form of a configuration. Given a finite configuration $C$, its Foata normal form $FC$ is the list of sets of events constructed by the following algorithm [3]:

**ALGORITHM 6.1. Foata normal form of a configuration**

- **input:** A configuration $C$ of a branching process
- **output:** The Foata normal form $FC$ of $C$.

**begin**

$FC := \emptyset$;

**while** $C \neq \emptyset$ **do**

- append $\text{Min}(C)$ to $FC$;
- $C := C \setminus \text{Min}(C)$

**endwhile**

**end**  

Loosely speaking, the Foata normal form is obtained by repeatedly slicing out the set of minimal events.

Given two configurations $C_1$ and $C_2$, we can compare their Foata normal forms $FC_1 = C_{11} \ldots C_{1m_1}$ and $FC_2 = C_{21} \ldots C_{2n_2}$ with respect to the order $\ll$: we say $FC_1 \ll FC_2$ if there exists $1 \leq i \leq n_1$ such that

- $\varphi(C_{1j}) = \varphi(C_{2j})$ for every $1 \leq j < i$, and
- $\varphi(C_{1i}) \ll \varphi(C_{2i})$.

Now, we define

**DEFINITION 6.2. Order $\ll_F$**

Let $C_1$ and $C_2$ be two configurations of the unfolding of a net system. $C_1 \ll_F C_2$ holds if

- $|C_1| < |C_2|$, or
- $|C_1| = |C_2|$ and $\varphi(C_1) \ll \varphi(C_2)$, or
- $\varphi(C_1) = \varphi(C_2)$ and $FC_1 \ll FC_2$.  

In other words, in order to decide if $C_1 \ll_F C_2$ we compare first the sizes of $C_1$ and $C_2$; if they are equal, we compare $\varphi(C_1)$ and $\varphi(C_2)$; if they are equal, we compare $FC_1$ and $FC_2$.

Observe that $\ll_F$ is a refinement of $\ll_E$. We now prove that $\ll_F$ is indeed adequate and total. The key property of 1-safe systems that yields to this result is:
PROPOSITION 6.3.

Any two concurrent conditions of the branching process of a 1-safe net system carry different labels.

Proof: Assume that two concurrent conditions $b_1$ and $b_2$ carry the same label $s$. Since $\{b_1, b_2\}$ is a co-set, there is a cut $c$ containing both $b_1$ and $b_2$. This cut corresponds to a reachable marking that puts at least two tokens on the place $s$, which violates 1-safeness. ■

THEOREM 6.4.

Let $\beta = (Q, p)$ be the unfolding of a 1-safe net system. $\prec_F$ is an adequate total order on the configurations of $\beta$.

Proof: a) $\prec_F$ is a well-founded partial order.

This follows immediately from the fact that $\prec_E$ is a well-founded partial order as is the lexicographic order on transition sequences of some fixed length.

b) $C_1 \subset C_2$ implies $C_1 \prec_F C_2$.

This is obvious, since $C_1 \subset C_2$ implies $|C_1| < |C_2|$.

c) $\prec_F$ is total.

Assume that $C_1$ and $C_2$ are two incomparable configurations under $\prec_F$, i.e. $|C_1| = |C_2|$, $\varphi(C_1) = \varphi(C_2)$, and $\varphi(FC_1) = \varphi(FC_2)$. We prove $C_1 = C_2$ by induction on the common size $k = |C_1| = |C_2|$.

The base case $k = 0$ gives $C_1 = C_2 = \emptyset$, so assume $k > 0$.

We first prove $\min(C_1) = \min(C_2)$. Assume without loss of generality that $e_1 \in \min(C_1) \setminus \min(C_2)$. Since $\varphi(\min(C_1)) = \varphi(\min(C_2))$, $\min(C_2)$ contains an event $e_2$ such that $p(e_1) = p(e_2)$. Since $*\min(C_1)$ and $*\min(C_2)$ are subsets of $\min(O)$, and all the conditions of $\min(O)$ carry different labels by Proposition 6.3, we have $*e_1 = *e_2$. This contradicts condition (iv) of the definition of branching process.

Since $\min(C_1) = \min(C_2)$, both $C_1 \setminus \min(C_1)$ and $C_2 \setminus \min(C_2)$ are configurations of the branching process $\uparrow\min(C_1)$ of $(N, \mark(\min(C_1)))$ (Proposition 4.3), and they are incomparable under $\prec_F$ by construction. Since the common size of $C_1 \setminus \min(C_1)$ and $C_2 \setminus \min(C_2)$ is strictly smaller than $k$, we can apply the induction hypothesis and conclude $C_1 = C_2$. 19
d) $\prec_F$ is preserved by finite extensions.

Take $C_1 \prec_F C_2$ with $\text{Mark}(C_1) = \text{Mark}(C_2)$. We have to show that $C_1 \oplus E \prec_F C_2 \oplus I_1^2(E)$. We can assume that $E = \{e\}$ and apply induction afterwards. (Notice that $\text{Mark}(C_1) = \text{Mark}(C_2)$ implies $\text{Mark}(C_1 \oplus \{e\}) = \text{Mark}(C_2 \oplus I_1^2(\{e\})).$ The cases $|C_1| < |C_2|$ and $C_1 \prec_E C_2$ are easy. Hence assume $|C_1| = |C_2|$ and $\varphi(C_1) = \varphi(C_2)$. We show first that $e$ is a minimal event of $C_1' = C_1 \cup \{e\}$ if and only if $I_1^2(e)$ is a minimal event of $C_2' = C_2 \cup \{I_1^2(e)\}$.

So let $e$ be minimal in $C_1'$, i.e. the transition $p(e)$ is enabled under the initial marking. Let $s \in \bullet p(e)$; then no condition in $\bullet C_1 \cup C_1'$ is labelled $s$, since these conditions would be in co relation with the $s$-labelled condition in $\bullet e$, contradicting Proposition 6.3. Thus, $C_1$ contains no event $e'$ with $s \in \bullet p(e')$, and the same holds for $C_2$ since $\varphi(C_1) = \varphi(C_2)$. Therefore, the conditions in $\text{Cut}(C_2)$ with label in $\bullet p(e)$ are minimal conditions of $\beta$, and $I_1^2(e) = e$ is minimal in $C_2'$. The reverse implication holds analogously, since about $C_1$ and $C_2$ we have only used the hypothesis $\varphi(C_1) = \varphi(C_2)$.

With this knowledge about the positions of $e$ in $C_1'$ and $I_1^2(e)$ in $C_2'$, we proceed as follows. If $\min(C_1) \prec_E \min(C_2)$, then we now see that $\min(C_1') \prec_E \min(C_2')$, hence $\varphi(FC_1') \ll \varphi(FC_2')$ and so we are done. If $\varphi(\min(C_1)) = \varphi(\min(C_2))$ and $e \in \min(C_1')$, then

$$C_1' \setminus \min(C_1') = C_1 \setminus \min(C_1) \prec_F C_2 \setminus \min(C_2) = C_2' \setminus \min(C_2')$$

hence $C_1' \prec_F C_2'$. Finally, if $\varphi(\min(C_1)) = \varphi(\min(C_2))$ and $e \notin \min(C_1')$, we again argue that $\min(C_1) = \min(C_2)$ and that, hence, $C_1 \setminus \min(C_1)$ and $C_2 \setminus \min(C_2)$ are configurations of the branching process $\uparrow \min(C_1)$ of (N, Mark(C_1)); with an inductive argument we get $C_1' \setminus \min(C_1') \prec_F C_2' \setminus \min(C_2')$ and are also done in this case.

$\blacksquare$ 6.4

7. The $n$-safe case

In this section we study the problem of computing a complete finite prefix for an $n$-safe but not necessarily 1-safe-system.

In the case of $n$-safe systems the partial order $\prec_F$ is neither adequate nor total. Figure 6 shows a 2-safe system (left) and one of its branching processes (right). Take $t_1 \ll t_2$, and consider the configurations $C_1 = \ldots$
\{e_1, e_3\} and \( C_2 = \{e_1, e_5\} \); since their Foata normal forms are \( \{e_1\}\{e_3\} \) and \( \{e_1, e_5\} \) we have \( C_1 \prec_F C_2 \).

The processes \( \uparrow C_1 \) and \( \uparrow C_2 \) are isomorphic because \( \text{Mark}(C_1) = \{s_1, s_1\} = \text{Mark}(C_2) \). There are two possible isomorphisms \( I^2_1 \); the first one satisfies \( I^2_1(e_6) = e_8 \), and the second \( I^2_1(e_6) = e_4 \). However, we have both \( C_1 \oplus \{e_6\} \succ_F C_2 \oplus \{e_8\} \) and \( C_1 \oplus \{e_6\} \succ_F C_2 \oplus \{e_4\} \). So \( \prec_F \) is not preserved by extensions.

Figure 7 shows a 2-safe net system (a) and its unfolding (b). The configurations \( \{e_1\} \) and \( \{e_2\} \) are not ordered by \( \prec_F \). So \( \prec_F \) is not total.

7.1. A different unfolding and finite prefix

It is not known whether there exists a total adequate order for the \( n \)-safe case. To deal with this case, we propose a different definition for the unfolding of a net system (similar ideas have been independently developed by Haar in \cite{7}). The old and the new definition are essentially equivalent for 1-safe systems. For \( n \)-safe systems, the new definition leads to branching processes with less concurrency. We accept this loss of concurrency for two reasons. First, the new definition allows to make use of our total adequate order \( \prec_F \). Second, as we will see at the end of the section, the branching processes according to the new definition can be more compact – independently of the adequate order used. So the
loss of concurrency does not necessarily lead to a poorer performance, as one might think.

Fix a net system \( \Sigma = (N, M_0) \) for the rest of this section. For the moment we do not impose any condition on \( \Sigma \); it could even be unsafe. Using a well-known folklore construction of net theory (to our knowledge first presented in [18]), we show that there exists a 1-safe system \( \Sigma_1 = (N_1, M_{01}) \) such that the reachability graphs of \( \Sigma \) and \( \Sigma_1 \) are isomorphic. We then define the new unfolding of \( \Sigma \) as the old unfolding of \( \Sigma_1 \).

The system \( \Sigma_1 \) has infinitely many places and transitions; later we show how to deal with this problem. A completely formal definition of \( \Sigma_1 \) can be found in page 229 of [19]. Here we give a less formal, but hopefully precise description.

- For each place \( s \) of \( N \) we add places \( [s, 0], [s, 1], [s, 2], \ldots \) to \( N_1 \). The intended meaning of a token in the place \( [s, i] \) of \( N_1 \) is that the place \( s \) of \( N \) currently contains \( i \) tokens.

- For each transition \( t \) we add to \( N_1 \) as many transitions as there are markings for the input and output places of \( t \). More precisely, given a transition \( t \) of \( N \), we call a function

\[
m: (\cdot t \cup t^*) \to N
\]

a firing mode of \( t \) if \( m(s) > 0 \) for every place \( s \in \cdot t \). The net \( N_1 \) contains a transition \([t, m]\) for each firing mode \( m \) of \( t \). The input and output places of \([t, m]\) are determined by \( m \) in the natural way: if \( s \in \cdot t \cup t^* \) and \( m(s) = k \), then \([s, k] \in [t, m]\) and \([s, k - W(s, t) + W(t, s)] \in [t, m]^*\).

- In order to determine the initial marking of \( \Sigma_1 \), we first associate to each marking \( M \) of \( N \) a 1-safe marking \( M_1 \) of \( N_1 \) in the following way: for each place \( s \) of \( N \), if \( M \) puts \( k \) tokens on \( s \), then \( M_1 \) puts one token on \([s, k]\) and no tokens on any other place \([s, i]\) with \( i \neq k \). We choose \( M_{01} \), the 1-safe marking associated to the initial marking \( M_0 \) of \( \Sigma \), as the initial marking of \( \Sigma_1 \).

It is proved in [19] that \( \Sigma_1 \) is indeed 1-safe, and that the reachability graphs of \( \Sigma \) and \( \Sigma_1 \) are isomorphic up to the projection of the labels \([t, m]\) of the transitions of \( \Sigma_1 \) onto their first components. These results are immediate consequences of the following fact

For every transition \( t \) of \( \Sigma \), we have \( M \xrightarrow{t} M' \) if and only if there exists a firing mode \( m \) of \( t \) such that \( M_1 \xrightarrow{[t, m]} M'_1 \).
which can be easily proved using the definitions. As long as we are only interested in properties that can be decided by inspection of the reachability graph (such as deadlock freedom, reachability of a marking etc.), we can use the complete prefix of $\Sigma_1$ instead of the complete prefix of $\Sigma$.

If the system $\Sigma$ is $n$-safe then the places of $\Sigma_1$ of the form $[s, k]$ with $k > n$ never become marked, and the transitions $[t, m]$ where $m(s) > n$ for some place $s$ never become enabled. All these places and transitions can be removed from $\Sigma_1$ without changing its behaviour in any way, and so we obtain a finite 1-safe system. Figure 8 shows the finite 1-safe system obtained in this way from the 2-safe system of Figure 7.

Since $\Sigma_1$ is infinite, we cannot compute the new complete prefix of $\Sigma$ by first constructing $\Sigma_1$, and then computing the old complete prefix of $\Sigma_1$. Fortunately, the old complete prefix of $\Sigma_1$ can be directly computed from $\Sigma$. It suffices to slightly modify Algorithm 4.7. Given a branching process $\beta$ of $\Sigma_1$, let $PE_1(\beta)$ denote the possible extensions of $\beta$ as a branching process of $\Sigma_1$.

**ALGORITHM 7.1. The complete prefix algorithm (arbitrary systems)**

```
input: An n-safe net system $\Sigma = (N, M_0)$ with $\{s_1, \ldots, s_n\}$ as set of places.
output: A complete finite prefix of $\Sigma_1$'s unfolding.
begin
  $Fin_1 := ([s_1, M_0(s_1)], \emptyset), \ldots, ([s_n, M_0(s_n)], \emptyset);$
  $pe := PE_1(Fin_1);$
  $cut-off := \emptyset;$
  while $pe \neq \emptyset$ do
    choose an event $e = ([t, m], X)$ in $pe$ such that $[e]$ is minimal with respect to $<;$
    if $[e] \cap cut-off = \emptyset$ then
      append to $Fin_1$ the event $e$ and a condition $([s, k], e)$
      for every output place $[s, k]$ of $[t, m];$
      $pe := PE_1(Fin_1);$
      if $e$ is a cut-off event of $Fin_1$ then
        $cut-off := cut-off \cup \{e\}$
      endif
    else $pe := pe \setminus \{e\}$
    endif
  endwhile
end
```

23
We still have to show how to compute $PE_1(\beta)$ for a branching process $\beta$ of $\Sigma_1$. We consider each transition $t$ of $\Sigma$ in turn, and look in $Fin_1$ for co-sets of conditions $X$ such that

- $p(X) = *[t, m]$ for some firing mode $m$ of $t$, and

- $Fin_1$ contains no event $e$ satisfying $p(e) = [t, m]$ and $*e = X$.

Clearly, the nodes $([t, m], X)$ are the possible extensions of $\beta$.

In order to determine the existence of a firing mode $m$ such that $p(X) = *[t, m]$, the only information we need are the sets of input and output places of $t$; this information can be directly retrieved from $\Sigma$.

We have the following result:

**THEOREM 7.2.**

Let $Fin_1$ be the complete prefix of $\Sigma_1$ generated by Algorithm 7.1 on input $\Sigma$ with $<_F$ as adequate order. The number of non-cut-off events of $Fin_1$ does not exceed the number of reachable markings of $\Sigma$.

**Proof:** By Theorem 5.3, the number of non-cut-off events of $Fin_1$ does not exceed the number of reachable markings of $\Sigma_1$. Since the reachability graphs of $\Sigma$ and $\Sigma_1$ are isomorphic, it does not exceed the number of reachable markings of $\Sigma$ either. ■ 7.2

It follows from this theorem that the algorithm terminates whenever the system $\Sigma$ is $n$-safe for some number $n$.

Note that the maximum number of tokens on each place of the bounded net system can easily be obtained after computation of the finite prefix. This can be achieved by linearly searching through the list of conditions, remembering the maximum token values of the respective place nodes. In [14], a graph representing the co-relation of
the underlying McMillan prefix has to be constructed to determine the maximum number of tokens.

7.2. PARTIAL-ORDER SEMANTICS AND COMPARISON

The complete prefixes obtained from the same system $\Sigma$ through the application of Algorithm 4.7 and Algorithm 7.1 generate the same reachability information, as we have seen. However, they can be very different. They correspond to two different semantics of Petri nets, which are usually called the process semantics (Algorithm 4.7) and the execution semantics (Algorithm 7.1). The latter has been defined and compared to the former in [19]. In this paper we are interested in the sizes of the complete prefixes obtained with the two semantics. We have not found any general relationship between the sizes; the following examples show that none of them lead to smaller complete prefixes in all cases.

If we treat Example 1 of Figure 9 with the usual partial-order semantics and McMillan’s cut-off criterion, we get a prefix with $n$ events; executions give $n!$ events with McMillan’s cut-off criterion and $2^n$ non-cut-off events with our improved criterion. This shows that executions can suffer severely from the ‘loss of concurrency’ compared to the usual partial-order semantics.
We again refer to the system shown in Figure 4, but this time with two initial tokens put on place $s_0$. The original net consists of $2n$ transitions, and the usual partial-order semantics leads to a prefix with $2(2^n+1-2)$ events with our improved criterion. Experimentally, we have extrapolated that the prefix using execution semantics contains $8n-4$ events only.

In Example 2 of Figure 9, the transitions $a_i$ produce tokens on $s$ one after the other, the transitions $b_j$ can then take these in any possible order. With the usual partial-order semantics and McMillan's cut-off criterion, this gives a prefix with $n!$ events labelled $b_n$—while executions even without our improvement give $2n$ events altogether. Example 3 of the same figure shows another effect: the usual partial-order semantics has to consider all pairs of tokens from $s_1$ and $s_2$ and leads to $n^2$ events, whereas executions (even without our improvement) lead to only one event.

One could have the feeling that the loss-of-concurrency effect is more common than the effects of Example 2 and 3; if this is so, a good application area will be nets with only a few reachable markings with more than 1 token on a place, where one can hope that the loss-of-concurrency effect will be more than cancelled out by the effects of our improved cut-off criterion.

8. Implementation, complexity, and experimental results

*Implementation.* Algorithm 4.7 requires to store and manipulate Petri nets and branching processes. For the storage we have developed an efficient, universal data structure that allows fast access to single nodes [17]. This data structure is based on the underlying incidence matrix of the net. Places, transitions and arcs are represented by nodes of doubly linked lists. We have developed a library of basic operations on nets supporting in particular fast insertion of single nodes.

Algorithm 4.7 is very simple, and can be easily proved correct, but is not efficient. In particular, it computes the set $PE$ of possible extensions each time a new event is added to $Fin$, which is clearly redundant. Similarly to McMillan's original algorithm [11], in the implementation we use a priority queue to store the set $PE$ of possible extensions. The queue is implemented in a rather naive way, because experiments with more sophisticated implementations show no improvements in average time. The events are sorted according to the size of their local configurations, as in [11], and not according to $\prec_F$, because this leads to many unnecessary comparisons. Events are compared with respect to
only when it is needed, i.e., when there are several events at the head of the queue whose local configurations have the same size.

With this implementation, the new algorithm only computes more than McMillan’s when two events $\epsilon$ and $\epsilon'$ satisfy $\text{Mark}(|\epsilon|) = \text{Mark}(|\epsilon'|)$ and $|\epsilon| = |\epsilon'|$. But this is precisely the case in which the algorithm behaves better, because it always identifies either $\epsilon$ or $\epsilon'$ as a cut-off event. In other words: when the complete prefix computed by McMillan’s algorithm is minimal, our algorithm computes the same result with no time overhead.

The computation of new elements for the set $PE$ involves the combinatorial problem of finding sets of conditions $B$ such that $p(B) = \text{\#}t$ for some transition $t$. We have implemented several improvements in this combinatorial determination, which have significant influence on the performance of the algorithm.

**Complexity.** The exact running time of the algorithm depends on the details of the implementation. Here we only give some information. The dominating factor is the computation of the possible extensions. Let $B$ be the set of conditions of the finite complete prefix after removing all cut-off events and their output conditions. In the worst case, for each transition $t$ the algorithm may have to examine all subsets $B'$ of $B$ such that there is a bijection between $p(B')$ and the preset of $t$, since these are the candidates for a possible extension. (Observe that the output conditions of a cut-off event cannot belong to a possible extension; that is why we have excluded them from the set $B$.) If $\xi$ is the size of the preset of $t$, the algorithm may have to examine $(\frac{|B|}{\xi})^\xi$ subsets.

Let $\xi$ denote the maximum size of the presets or postsets of the transitions in the original net system $\Sigma$. If $B \gg \xi$, then $(\frac{|B|}{\xi})^\xi \leq (\frac{|B|}{\xi})^\xi$, and so the algorithm may have to examine a total of $O(|T| \cdot (\frac{|B|}{\xi})^\xi)$ subsets. In the case of 1-safe Petri nets and a total adequate order, the number of non-cut-off events is at most the number $R$ of reachable markings of $\Sigma$. Since each non-cut-off event contributes at most $\xi$ conditions to the finite complete prefix, we have $|B| \leq R \cdot \xi + |S|$, and so – under the natural assumption $R \geq |S|$ – we obtain an upper bound of $O(|T| \cdot R^\xi)$ subsets. However, the question whether there exists a family of net systems for which the algorithm examines $\Theta(|T| \cdot R^\xi)$ subsets is open.

The space required by the algorithm is linear in the size of the complete prefix. If the cut-off events are not stored (which is not required for many verification algorithms working on the prefix), then for the case of 1-safe Petri nets and total adequate orders the complete prefix contains a total number of $O(\xi \cdot R)$ conditions and events. For the number of cut-
Table I. Results of the distributed mutual exclusion (DME) example

<table>
<thead>
<tr>
<th>n</th>
<th>Original net</th>
<th>Unfolding</th>
<th>t [s](^3)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>McMillan</td>
</tr>
<tr>
<td>2</td>
<td>135 98 &gt; 10(^2)</td>
<td>487 122 4</td>
<td>0.07</td>
</tr>
<tr>
<td>3</td>
<td>202 147 &gt; 10(^3)</td>
<td>1210 321 9</td>
<td>0.27</td>
</tr>
<tr>
<td>4</td>
<td>269 196 &gt; 10(^4)</td>
<td>2381 652 16</td>
<td>1.23</td>
</tr>
<tr>
<td>5</td>
<td>336 245 &gt; 10(^5)</td>
<td>4096 1145 25</td>
<td>3.92</td>
</tr>
<tr>
<td>6</td>
<td>403 294 &gt; 10(^6)</td>
<td>6451 1830 36</td>
<td>10.37</td>
</tr>
<tr>
<td>7</td>
<td>470 343 &gt; 10(^7)</td>
<td>9542 2737 49</td>
<td>28.45</td>
</tr>
<tr>
<td>8</td>
<td>537 392 &gt; 10(^8)</td>
<td>13465 3896 64</td>
<td>68.16</td>
</tr>
<tr>
<td>9</td>
<td>604 441 &gt; 10(^9)</td>
<td>18316 5337 81</td>
<td>131.88</td>
</tr>
<tr>
<td>10</td>
<td>671 490 &gt; 10(^10)</td>
<td>24191 7090 100</td>
<td>240.57</td>
</tr>
<tr>
<td>11</td>
<td>738 539 &gt; 10(^11)</td>
<td>31186 9185 121</td>
<td>420.12</td>
</tr>
</tbody>
</table>

off events we only know a trivial upper bound of \(O(|T| \cdot R^c)\), although in the experiments conducted so far we have always observed \(O(R)\).

Experimental results. We consider three scalable 1-safe net examples. We compare McMillan's algorithm and the new algorithm, both implemented using the universal data structure and the improvements in the combinatorial determination mentioned above.

The first example is a model of an asynchronous circuit for distributed mutual exclusion (DME), proposed in [10] and also used in [11]. McMillan has already shown that the state space grows exponentially in the number of DME-cells while the unfolding increases just quadratically. In Table I we list the experimental results. In this example, the complete prefix computed by McMillan's algorithm is minimal. The new algorithm computes the same prefix without time overhead, as expected.

Our second example, Figure 10, is a model of a slotted ring protocol taken from [16]. Here, the output of the new algorithm grows significantly slower than the output of McMillan's algorithm. For \(n = 6\) the output is already one order of magnitude smaller.

\(^2\) All the times (\(t\)) have been measured on a SPARCstation 20 with 48 MB main memory.

\(^3\) In all Tables, \(R\) indicates the number of reachable states and \(c\) the number of cut-off events.

\(^4\) These times could not be calculated; for \(n = 7\) we interrupted the computation after more than 12 hours.
Figure 10. Slotted ring protocol for $n = 2$

Table II. Results of the slotted ring protocol example

<table>
<thead>
<tr>
<th>$n$</th>
<th>Original net</th>
<th>McMillan’s algorithm</th>
<th>New algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>T</td>
<td>$</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>$1.2 \cdot 10^1$</td>
<td>12</td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td>$2.1 \cdot 10^2$</td>
<td>68</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
<td>$4.0 \cdot 10^3$</td>
<td>288</td>
</tr>
<tr>
<td>4</td>
<td>40</td>
<td>$8.2 \cdot 10^4$</td>
<td>1248</td>
</tr>
<tr>
<td>5</td>
<td>50</td>
<td>$1.7 \cdot 10^5$</td>
<td>6240</td>
</tr>
<tr>
<td>6</td>
<td>60</td>
<td>$3.7 \cdot 10^7$</td>
<td>31104</td>
</tr>
<tr>
<td>7</td>
<td>70</td>
<td>$8.0 \cdot 10^8$</td>
<td>—5</td>
</tr>
<tr>
<td>8</td>
<td>80</td>
<td>$1.7 \cdot 10^9$</td>
<td>—5</td>
</tr>
<tr>
<td>9</td>
<td>90</td>
<td>$3.8 \cdot 10^{10}$</td>
<td>—5</td>
</tr>
<tr>
<td>10</td>
<td>100</td>
<td>$8.1 \cdot 10^{12}$</td>
<td>—5</td>
</tr>
</tbody>
</table>

In Table 3, we give the times for an example taken from [2] that models Milner’s cyclic scheduler for $n$ tasks. While the size of the unfolding produced by the McMillan’s algorithm grows exponentially with the number of tasks, we get linear size using our new one.

Table 3. Results of Milner’s cyclic scheduler

<table>
<thead>
<tr>
<th>$n$</th>
<th>Original net</th>
<th>McMillan’s algorithm</th>
<th>New algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>S</td>
<td>$</td>
</tr>
<tr>
<td>3</td>
<td>23</td>
<td>17</td>
<td>43</td>
</tr>
<tr>
<td>6</td>
<td>47</td>
<td>35</td>
<td>630</td>
</tr>
<tr>
<td>9</td>
<td>71</td>
<td>53</td>
<td>7423</td>
</tr>
<tr>
<td>12</td>
<td>95</td>
<td>71</td>
<td>74264</td>
</tr>
</tbody>
</table>

29
For the implementation of the $n$-safe algorithm 7.1, the underlying data structure for the prefix has been slightly extended to store the additional token information. The representation of the original net remains unchanged; in particular, there is no additional structure combining presets and postsets of transitions.

The computation of possible extensions once more dominates the time complexity; now, there are even more place nodes to investigate for each single event since we have to consider all elements in the preset and postset of a transition. For the $n$-safe case, the term $\xi$ in the above formula describing the time complexity denotes the maximum number of input and output arcs of the transitions of the original net, i.e., $\xi = \max_{t \in T} \{|t \cup t^*|\}$.

9. Conclusions

We have presented an algorithm for the computation of a complete finite prefix of an unfolding using a refinement of McMillan’s basic notion of cut-off event. The prefixes constructed by the algorithm contain at most $n$ non-cut-off events, where $n$ is the number of reachable markings of the net. Therefore, we can guarantee that the prefix is never significantly larger than the reachability graph, which did not hold for the algorithm of [11].

Acknowledgements

We thank Michael Kishinevsky, Alexander Taubin and Alex Yakovlev for drawing our attention to this problem, Burkhard Graves for detecting some mistakes, and Ken McMillan for sending us his LISP sources of the DME generator.

References
