Formal Methods in System Design, 20, 285–310, 2002
© 2002 Kluwer Academic Publishers. Manufactured in The Netherlands.

An Improvement of McMillan's Unfolding Algorithm

JAVIER ESPARZA esparza@in.tum.de
STEFAN RÖMER* 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

Received June 2, 1998; Accepted September 15, 2000

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), 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

*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 "Halbordnungstesten").

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. Sections 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 $(S \times T) \cup (T \times S) \to \{0, 1\}$. 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 ${}^\bullet x$, is the set $\{y \in S \cup T \mid W(y, x) = 1\}$. The *postset* of x, denoted by $x^\bullet$, is the set $\{y \in S \cup T \mid W(x, y) = 1\}$.

A *marking* of a net (S, T, W) is a mapping $M : S \to \mathbb{N}$ (where $\mathbb{N}$ 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 Σ).

2.1. 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 poset; 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

$$M_0 \xrightarrow{t_1} M_1 \xrightarrow{t_2} \ldots M_{n-1} \xrightarrow{t_n} M_n$$

M_n is the marking reached by the occurrence of σ, also denoted by $M_0 \xrightarrow{\sigma} M_n$. M is a *reachable marking* if there exists an occurrence sequence σ 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 Σ is n-safe if all its reachable markings are n-safe, and *safe* if it is n-safe for some number n.

2.2. *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 $x \; co \; y$, 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.

$Min(O)$ 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 $Min(O)$ 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 ${}^\bullet e$ is a bijection between ${}^\bullet e$ (in Σ) and ${}^\bullet p(e)$ (in β), and similarly for $e^\bullet$ and $p(e)^\bullet$ (p preserves the environments of transitions);
(iii) the restriction of p to $Min(O)$ is a bijection between $Min(O)$ and M_0 (β "starts" at M_0);
(iv) for every $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $p(e_1) = p(e_2)$ then $e_1 = e_2$ (β does not duplicate the transitions of Σ).

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

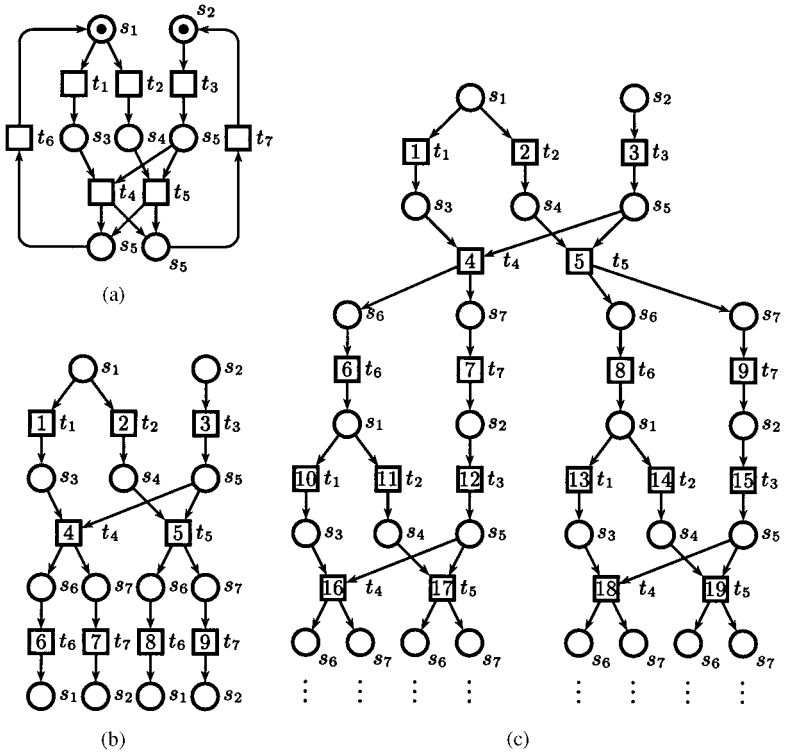

Figure 1. A net system (a) and two of its branching processes (b, c).

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. β' is a *prefix* of β if O' is a subnet of O satisfying

- $Min(O)$ belongs to O';
- if a condition b belongs to O', then its input event $e \in {}^\bullet b$ in O also belongs to O' (if it exists); and
- if an event e belongs to O', then its input and output conditions ${}^\bullet e \cup e^\bullet$ 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 Σ be a net system, and let β be its unfolding. The reachability graphs of Σ and β have isomorphic unfoldings (as graphs as described above). More in detail, the reachable markings of Σ are those $p(M)$ where M is a reachable marking of β; for a reachable marking M of β and a marking M'' and a transition t of Σ, there are M' and e with $p(M') = M''$, $p(e) = t$ and $M \xrightarrow{e} M'$ in β if and only if $p(M) \xrightarrow{t} M''$ in Σ.

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 Σ is *represented* in a branching process $\beta = (O, p)$ of Σ if β contains a cut c such that, for each place s of Σ, 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 $Cut(C)$, defined below, is a cut:

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

In particular, given a configuration C the set of places $Cut(C)$ represents a reachable marking, which we denote by $Mark(C)$. Loosely speaking, $Mark(C)$ is the marking we reach by firing

the configuration C. In the branching process of figure 1(b) we have $Mark(\{1, 3, 4, 6\}) = \{s_1, s_7\}$.

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 Σ 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 Σ, 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 Σ, 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 Σ 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 β.

The following characterisation follows easily from the definitions:

Proposition 4.1. *Let β be a branching process of a net system Σ. The possible extensions of β are the pairs (t, X), where X is a co-set of conditions of β and t is a transition of Σ such that*
- *$p(X) = {}^\bullet t$, and*
- *(t, X) does not already belong to β.*

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 Σ.
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

The algorithm does not necessarily terminate. In fact, it terminates if and only if the input system Σ 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]).

4.2. Constructing a finite complete prefix

We say that a branching process β of a net system Σ is *complete* if for every reachable marking M there exists a configuration C in β such that:

- $Mark(C) = M$ (i.e., M is represented in β), 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.

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

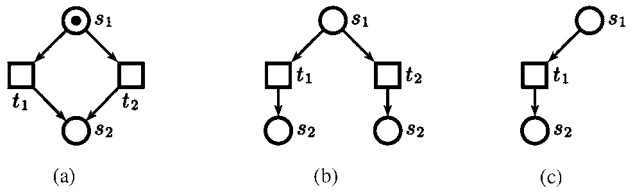

Figure 2. A 1-safe net system (a), its unfolding (b), and a prefix (c).

$\{x \mid x \notin C \cup {}^\bullet C \wedge \forall y \in C: \neg(x \# y)\}$ and p' is the restriction of p to the nodes of O'. Loosely speaking, $\Uparrow C$ is the part of β lying "after" C.

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

Proposition 4.3. *If β is a branching process of (N, M_0) and C is a configuration of β, then $\Uparrow C$ is a branching process of $(N, Mark(C))$. Moreover, if β is the unfolding of (N, M_0), then $\Uparrow C$ is the unfolding of $(N, Mark(C))$ (up to isomorphism).*

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_1^2 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 $C_2 \oplus I_1^2(E)$.

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 behing them.

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

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 $Mark([e]) = Mark([e'])$. 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 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:

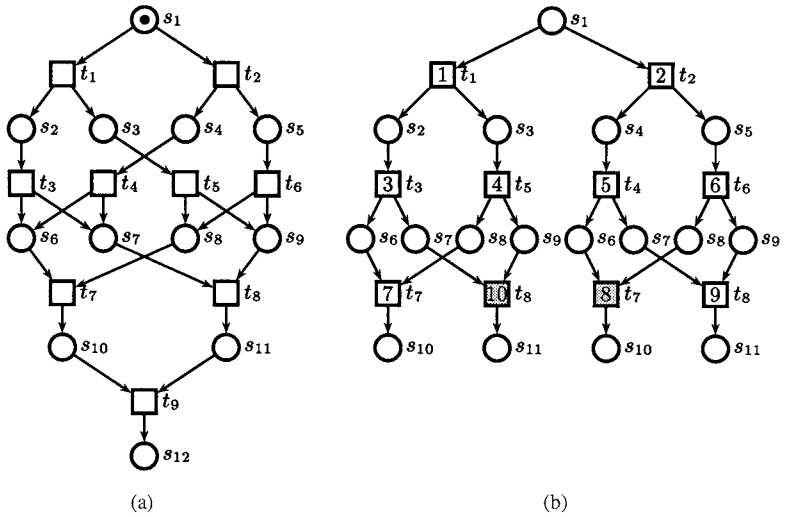

Figure 3. A 1-safe net system (a) and a prefix of its unfolding (b).

- $\prec$ 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 $Mark(C_1) = Mark(C_2)$, 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.

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 β be a prefix of the unfolding containing an event e. The event e is a *cut-off event* of β (with respect to $\prec$) if β contains a local configuration $[e']$ such that

(a) $Mark([e]) = Mark([e'])$, and
(b) $[e'] \prec [e]$.

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*

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

output: A complete finite prefix *Fin* of *Unf*.
begin
Fin := $(s_1, \emptyset), \ldots, (s_k, \emptyset)$;
pe := *PE*(*Fin*);
cut-off := $\emptyset$;
while *pe* $\neq \emptyset$ **do**
  choose an event $e = (t, X)$ in *pe* such that $[e]$ is minimal
    with respect to $\prec$;
  if $[e] \cap$ *cut-off* $= \emptyset$ **then**
    append to *Fin* the event e and a condition (s, e)
      for every output place s of t;
    pe := *PE*(*Fin*);
    if e is a cut-off event of *Fin* **then**
      cut-off := *cut-off* $\cup \{e\}$
    endif
  else *pe* := *pe* $\setminus \{e\}$
  endif
endwhile
end

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

Proposition 4.8. *Fin is finite.*

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

(1) For every event e of *Fin*, $d(e) \leq n + 1$, where n is the number of reachable markings of Σ.

  Since cuts correspond to reachable markings, every chain of events $e_1 < e_2 < \cdots < e_n < e_{n+1}$ of *Unf* contains two events e_i, e_j, $i < j$, such that $Mark([e_i]) = Mark([e_j])$. Since $[e_i] \subset [e_j]$, we have $[e_i] \prec [e_j]$, and therefore $[e_j]$ is a cut-off event of *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 *Fin*, too.

(2) For every event e of *Fin*, the sets ${}^\bullet e$ and $e^\bullet$ are finite.

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

(3) For every $k \geq 0$, *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, $E_k^\bullet$ is finite. Since ${}^\bullet E_{k+1} \subseteq E_k^\bullet \cup Min(Fin)$, 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 *Fin* only contains finitely many events. By (2) it contains only finitely many conditions. □

Proposition 4.9. *Fin is complete.*

Proof:

(a) Every reachable marking of Σ is represented in *Fin*.

Let M be an arbitrary reachable marking of Σ. There exists a configuration C_1 of *Unf* such that $Mark(C_1) = M$. If C_1 is not a configuration of *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 $Mark([e_2]) = Mark([e_1])$.

Consider the configuration $C_2 = [e_2] \oplus I_1^2(E_1)$. Since $\prec$ is preserved by finite extensions, we have $C_2 \prec C_1$. Morever, $Mark(C_2) = M$. If C_2 is not a configuration of *Fin*, then we can iterate the procedure and find a configuration C_3 such that $C_3 \prec C_2$ and $Mark(C_3) = M$. The procedure cannot be iterated infinitely often because $\prec$ is well-founded. Therefore, it terminates in a configuration of *Fin*.

(b) If a transition t can occur in Σ, then *Fin* contains an event labelled by t.

If t occurs in Σ, then some reachable marking M enables t. The marking M is represented in *Fin*. Let C be a minimal configuration with respect to $\prec$ such that $Mark(C) = M$. If C contains some cut-off event, then we can apply the arguments of (a) to conclude that *Fin* contains a configuration $C' \prec C$ such that $Mark(C') = M$. This contradicts the minimality of C. So C contains no cut-off events, and therefore *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 $Mark([e]) = M$ have the same size, and therefore there exist no cut-off events with respect to McMillan's order.[2]

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,

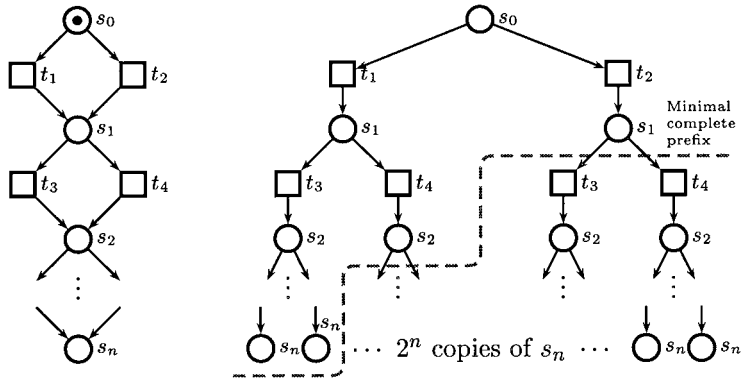

Figure 4. A Petri net and its unfolding.

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 Σ. 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 $\varphi(E) = t_1 t_2 t_2 t_3$. ($\varphi$ 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 $\prec_E$). Let C_1 and C_2 be two configurations of the unfolding of a net system. $C_1 \prec_E C_2$ holds if

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

Theorem 5.2. *Let β be the unfolding of a net system. $\prec_E$ is an adequate order on the finite configurations of β.*

Proof: It is easy to show that $\prec_E$ is a well-founded partial order implied by inclusion. To show that $\prec_E$ is preserved by finite extensions, assume $C_1 \prec_E C_2$. For every finite extension $C_1 \oplus E$ of C_1 we have $|E| = |I_1^2(E)|$, since I_1^2 is a bijection, and $\varphi(E) = \varphi(I_1^2(E))$, since I_1^2 preserves the labelling of events. If $|C_1| < |C_2|$, then $|C_1 \oplus E| < |C_2 \oplus I_{C_1}^{C_2}(E)|$. If $\varphi(C_1) \ll \varphi(C_2)$, then by the properties of the lexicographic order $\varphi(C_1 \oplus E) \ll \varphi(C_2 \oplus I_1^2(E))$. □

If we take $\prec_E$ 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

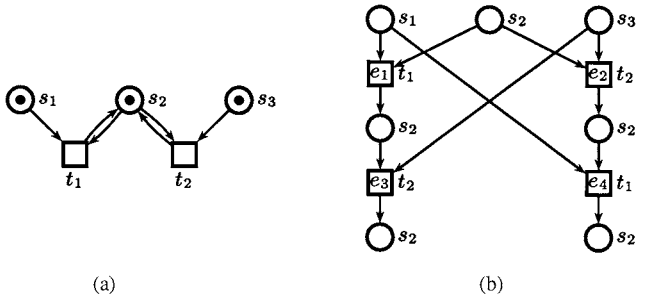

Figure 5. A 1-safe net system (a) and its unfolding (b).

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 $Mark([e]) = Mark([e'])$, 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 Fin be the output of Algorithm* 4.7 *on input* Σ. *The number of non-cut-off events of Fin does not exceed the number of reachable markings of* Σ.

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 Σ. 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 $Min(C)$ to FC;
$C := C \setminus Min(C)$
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 $FC_1 = C_{11} \ldots C_{1n_1}$ 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 $\prec_F$). Let C_1 and C_2 be two configurations of the unfolding of a net system. $C_1 \prec_F C_2$ 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 $C_1 \prec_F C_2$ 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 $\prec_F$ is a refinement of $\prec_E$. We now prove that $\prec_F$ 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 $\beta = (O, p)$ be the unfolding of a* 1*-safe net system. $\prec_F$ is an adequate total order on the configurations of β.*

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 $e_1 \in Min(C_1) \backslash Min(C_2)$. 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 ${}^\bullet Min(C_1)$ and ${}^\bullet Min(C_2)$ are subsets of $Min(O)$, and all the conditions of $Min(O)$ carry different labels by Proposition 6.3, we have ${}^\bullet e_1 = {}^\bullet e_2$. This contradicts condition (iv) of the definition of branching process.

Since $Min(C_1) = Min(C_2)$, both $C_1 \backslash Min(C_1)$ and $C_2 \backslash Min(C_2)$ 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 $C_1 \backslash Min(C_1)$ and $C_2 \backslash Min(C_2)$ is strictly smaller than k, we can apply the induction hypothesis and conclude $C_1 = C_2$.

(d) $\prec_F$ is preserved by finite extensions.

Take $C_1 \prec_F C_2$ with $Mark(C_1) = Mark(C_2)$. 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 $Mark(C_1) = Mark(C_2)$ implies $Mark(C_1 \oplus \{e\}) = Mark(C_2 \oplus I_1^2(\{e\}))$.) 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 ${}^\bullet C_1 \cup C_1^\bullet$ 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 $Cut(C_2)$ with label in ${}^\bullet p(e)$ are minimal conditions of β, 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

$$C_1' \backslash Min(C_1') = C_1 \backslash Min(C_1) \prec_F C_2 \backslash Min(C_2) = C_2' \backslash Min(C_2')$$

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, $C_1 \backslash Min(C_1)$ and $C_2 \backslash Min(C_2)$ are configurations of the branching process $\Uparrow Min(C_1)$ of $(N, Mark(Min(C_1)))$; with an inductive argument we get $C_1' \backslash Min(C_1') \prec_F C_2' \backslash Min(C_2')$ and are also done in this case. □

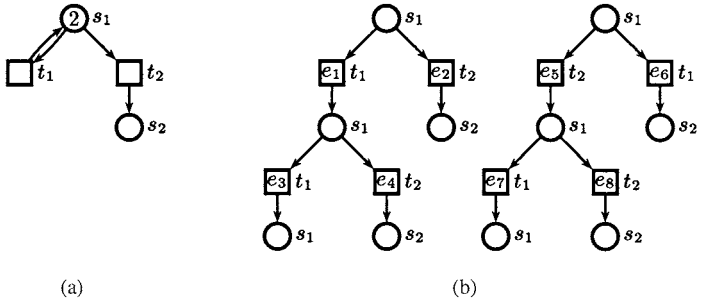

Figure 6. A 2-safe system (a) and one of its branching processes (b).

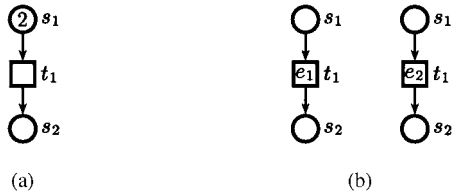

Figure 7. A 2-safe net system (a) and its unfolding (b).

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 $C_1 = \{e_1, e_3\}$ 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 $Mark(C_1) = \{s_1, s_1\} = Mark(C_2)$. There are two possible isomorphisms I_1^2; the first one satisfies $I_1^2(e_6) = e_8$, and the second $I_1^2(e_6) = e_4$. 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 devoloped by Haar in [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 Σ; 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 Σ and Σ_1 are isomorphic. We then define the new unfolding of Σ as the old unfolding of Σ_1.

The system Σ_1 has infinitely many places and transitions; later we show how to deal with this problem. A completely formal definition of Σ_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 $[s, 0]$, $[s, 1]$, $[s, 2]$, ... 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

  $$m\colon ({}^\bullet t \cup t^\bullet) \to \mathbb{N}$$

  a *firing mode of* t if $m(s) > 0$ for every place $s \in {}^\bullet t$. 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 $s \in {}^\bullet t \cup t^\bullet$ and $m(s) = k$, then $[s, k] \in {}^\bullet[t, m]$ and $[s, k - W(s, t) + W(t, s)] \in [t, m]^\bullet$.
- In order to determine the initial marking of Σ_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 Σ, as the initial marking of Σ_1.

It is proved in [19] that Σ_1 is indeed 1-safe, and that the reachability graphs of Σ and Σ_1 are isomorphic up to the projection of the labels $[t, m]$ of the transitions of Σ_1 onto their first components. These results are immediate consequences of the following fact

For every transition t of Σ, 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 Σ_1 instead of the complete prefix of Σ.

If the system Σ is n-safe then the places of Σ_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.

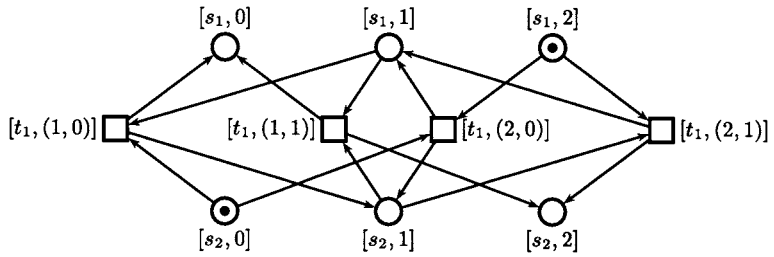

Figure 8. 1-safe system equivalent to the 2-safe system of figure 7.

All these places and transitions can be removed from Σ_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 Σ_1 is infinite, we cannot compute the new complete prefix of Σ by first constructing Σ_1, and then computing the old complete prefix of Σ_1. Fortunately, the old complete prefix of Σ_1 can be directly computed from Σ. It suffices to slightly modify Algorithm 4.7. Given a branching process β of Σ_1, let $PE_1(\beta)$ denote the possible extensions of β as a branching process of Σ_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 Σ_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 $\prec$;
  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 $pe := pe \backslash \{e\}$
  endif
endwhile
end

We still have to show how to compute $PE_1(\beta)$ for a branching process β of Σ_1. We consider each transition t of Σ in turn, and look in Fin_1 for co-sets of conditions X such that

– $p(X) = {}^\bullet[t, m]$ for some firing mode m of t, and
– Fin_1 contains no event e satisfying $p(e) = [t, m]$ and ${}^\bullet e = X$.

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

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

We have the following result:

Theorem 7.2. *Let Fin_1 be the complete prefix of Σ_1 generated by Algorithm* 7.1 *on input Σ with $\prec_F$ as adequate order. The number of non-cut-off events of Fin_1 does not exceed the number of reachable markings of Σ.*

Proof: By Theorem 5.3, the number of non-cut-off events of Fin_1 does not exceed the number of reachable markings of Σ_1. Since the reachability graphs of Σ and Σ_1 are isomorphic, it does not exceed the number of reachable markings of Σ either. □

It follows from this theorem that the algorithm terminates whenever the system Σ 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 Σ 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,

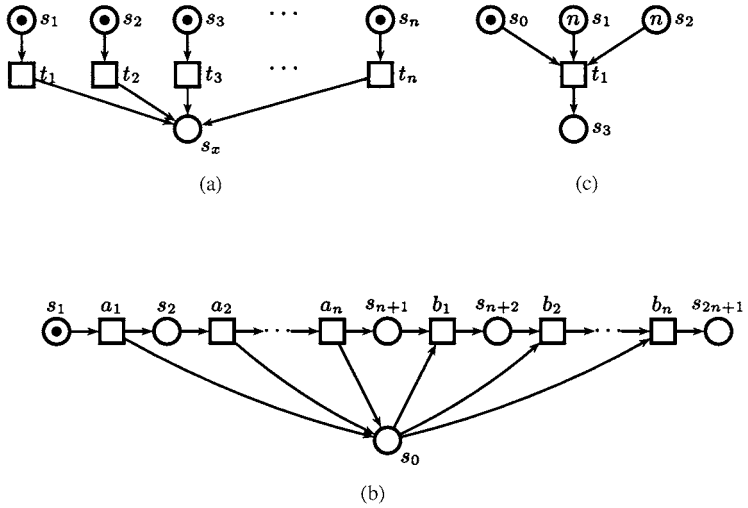

Figure 9. Examples for the n-safe algorithm (a) Example 1; (b) Example 2; (c) Example 3.

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 Examples 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

8.1. 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 $\prec_F$ 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 e and e' satisfy $Mark([e]) = Mark([e'])$ and $|[e]| = |[e']|$. But this is precisely the case in which the algorithm behaves better, because it always identifies either e or e' 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 $p(B) = {}^\bullet t$ for some transition t. We have implemented several improvements in this combinatorial determination, which have significant influence on the performance of the algorithm.

8.2. *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 ouput conditions of a cut-off event cannot belong to a possible extension; that is why we have excluded them from the set B.) If ζ is the size of the preset of t, the algorithm may have to examine $(\frac{|B|}{\zeta})^\zeta$ subsets.

Let ξ denote the maximum size of the presets or postsets of the transitions in the original net system Σ. If $B \gg \xi$, then $(\frac{|B|}{\zeta})^\zeta \leq (\frac{|B|}{\xi})^\xi$, 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 Σ. Since each non-cut-off event contributes at most ξ 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-off

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

8.3. 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 1 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 (Table 2).

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.

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.

Table 1. Results of the distributed mutual exclusion (DME) example.

	Original net			Unfolding			t [s][a]	
n	$\|S\|$	$\|T\|$	R[b]	$\|B\|$	$\|E\|$	c[b]	McMillan	New algorithm
2	135	98	$>10^2$	487	122	4	0.07	0.08
3	202	147	$>10^3$	1210	321	9	0.27	0.25
4	269	196	$>10^4$	2381	652	16	1.23	1.26
5	336	245	$>10^5$	4096	1145	25	3.92	3.88
6	403	294	$>10^6$	6451	1830	36	10.37	10.40
7	470	343	$>10^7$	9542	2737	49	28.45	29.08
8	537	392	$>10^8$	13465	3896	64	68.16	69.21
9	604	441	$>10^9$	18316	5337	81	131.88	130.59
10	671	490	$>10^{10}$	24191	7090	100	240.57	243.11
11	738	539	$>10^{11}$	31186	9185	121	420.12	418.02

[a]All the times (t) have been measured on a SPARCstation 20 with 48 MB main memory.
[b]In all tables, R indicates the number of reachable states and c the number of cut-off events.

Table 2. Results of the slotted ring protocol example.

	Original net		McMillan's algorithm			New algorithm		
n	$\|T\|$	R	$\|E\|$	c	t [s]	$\|E\|$	c	t [s]
1	10	1.2×10^1	12	3	0.00	12	3	0.00
2	20	2.1×10^2	68	12	0.00	62	14	0.00
3	30	4.0×10^3	288	60	0.13	186	42	0.05
4	40	8.2×10^4	1248	296	1.72	528	128	0.38
5	50	1.7×10^6	6240	1630	45.31	1280	300	1.58
6	60	3.7×10^7	31104	8508	1829.48	3216	792	11.08
7	70	8.0×10^8			—[c]	7224	1708	79.08
8	80	1.7×10^{10}			—[c]	17216	4256	563.69
9	90	3.8×10^{11}			—[c]	37224	8820	2850.89
10	100	8.1×10^{12}			—[c]	86160	21320	15547.67

[c]These times could not be calculated; for $n = 7$ we interrupted the computation after more than 12 hours.

Table 3. Results of Milner's cyclic scheduler.

	Original net			McMillan's algorithm				New algorithm			
n	$\|S\|$	$\|T\|$	R	$\|B\|$	$\|E\|$	c	t [s]	$\|B\|$	$\|E\|$	c	t [s]
3	23	17	43	94	44	8	0.02	52	23	4	0.00
6	47	35	639	734	361	64	0.48	112	50	7	0.02
9	71	53	7423	5686	2834	512	22.90	172	77	10	0.05
12	95	71	74264	45134	22555	4096	1471.16	232	104	13	0.13

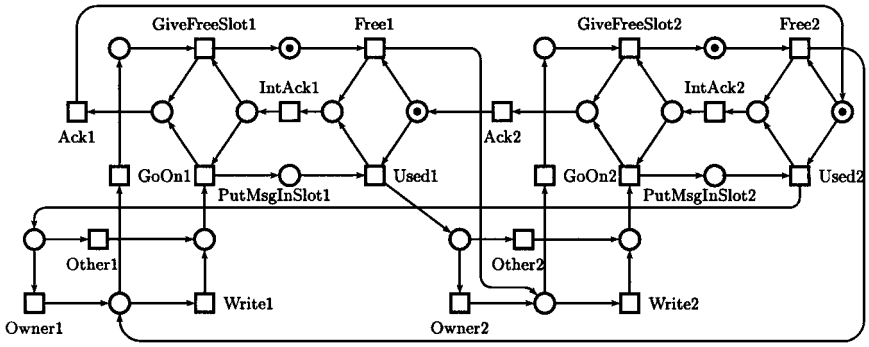

Figure 10. Slotted ring protocol for $n = 2$.

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 ξ 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., $\xi = max_{t \in T}\{|{}^\bullet t \cup t^\bullet|\}$.

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].

Acknowledgments

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.

Notes

1. It is immediate to prove that $[e]$ is a configuration.
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.

References

1. E. Best and C. Fernández, "Nonsequential processes—A Petri net view," *EATCS Monographs on Theoretical Computer Science*, Vol. 13, 1988.
2. J.C. Corbett, "Evaluating deadlock detection methods for concurrent software," in *Proceedings of the 1994 International Symposium on Software Testing and Analysis, ISSTA '94*, ACM-Press, New York, 1994, pp. 204–215.
3. V. Diekert, *Combinatorics on Traces*, LNCS, Vol. 454, 1990.
4. J. Engelfriet, "Branching processes of Petri nets," *Acta Informatica*, Vol. 28, pp. 575–591, 1991.
5. J. Esparza, "Model checking using net unfoldings," *Science of Computer Programming*, Vol. 23, pp. 151–195, 1994.
6. J. Esparza, S. Römer, and W. Vogler, "An improvement of McMillan's unfolding algorithm," in *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems, TACAS '96,* LNCS, Vol. 1055, 1996, pp. 87–106.
7. S. Haar, "Branching processes of general S/T-systems. Workshop Concurrency, Specification and Programming," Humboldt-Universität Berlin, Informatik-Bericht, Vol. 10, 1998, pp. 88–97.
8. M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky, *Concurrent Hardware: The Theory and Practice of Self-Timed Design*, Wiley, New York, 1993.
9. A. Kondratyev and A. Taubin, "Verification of speed-independent circuits by STG unfoldings," in *Proceedings of the Symposium on Advanced Research in Asynchronous Circuits and Systems*, Utah, 1994.
10. A.J. Martin, "The design of a self-timed circuit of distributed mutual exclusion," in Henry Fuchs (ed.), *Chapel Hill Conference on VLSI*, Computer Science Press, 1985, pp. 245–260.

11. K.L. McMillan, "Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits," in *Proceedings of the 4th Workshop on Computer Aided Verification*, Montreal, LNCS, Vol. 663, 1992, pp. 164–174.
12. K.L. McMillan, "A technique of state space search based on unfolding," *Formal Methods in System Design*, Vol. 6, No. 1, pp. 45–65, 1995.
13. K.L. McMillan, "Trace theoretic verification of asynchronous circuits using unfoldings," in *Proceedings of the 7th Workshop on Computer Aided Verification*, Liege, LNCS, Vol. 939, 1995, pp. 180–195.
14. T. Miyamoto and S. Kumagai, "Calculating place capacity for Petri nets using unfoldings," in *Proceedings of the 1998 International Conference on Application of Concurrency to System Design*, Japan, IEEE Computer Society, PR08350, 1998, pp. 143–151.
15. M. Nielsen, G. Plotkin, and G. Winskel, "Petri nets, event structures and domains," *Theoretical Computer Science*, Vol. 13, No. 1, pp. 85–108, 1980.
16. E. Pastor, O. Roig, J. Cortadella, and R.M. Badia, "Petri net analysis using Boolean manipulation," in *Proceedings of Application and Theory of Petri Nets '94*, LNCS, Vol. 815, 1994, pp. 416–435.
17. S. Römer, "Entwicklung und Implementierung von Verifikationstechniken auf der Basis von Netzentfaltungen," Dissertation (in German). Technische Universität München, 2000.
18. G. Ullrich, "Der Entwurf von Steuerstrukturen für parallele Abläufe mit Hilfe von Petri-Netzen," Universität Hamburg, Inst. für Informatik, IFI-HH-B-36/77, 1976.
19. W. Vogler, "Executions: A new partial-order semantics of Petri nets," *Theoretical Computer Science*, Vol. 91, pp. 205–238, 1991.