An Improvement of McMillan's Unfolding Algorithm

Javier Esparza*
Stefan Römer*
Walter Vogler†

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.

1 Introduction

In a seminal paper [10], 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 unfolding, a well known partial order semantics of Petri nets introduced in [12], 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 an initial part satisfying this property (in fact slightly stronger one) a *finite complete prefix*. He then shows how to use these prefixes for deadlock detection.

The unfolding technique has been later applied to other verification problems. In [7, 8, 11] 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 gen-

---

erates prefixes much larger than necessary. In some cases a minimal complete prefix has $O(n)$ in 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 sense to be defined). The prefix is always smaller than or as large as the prefix generated with the old algorithm.

The paper is organised as follows. Section 2 contains basic definitions about Petri nets and branching processes. In Section 3 we show that McMillan's algorithm is just an element of a whole family of algorithms for the construction of finite complete prefixes. In Section 4 we select an element of this family, and show that it generates minimal prefixes in a certain sense. Finally, in Section 5 we present experimental results.

2 Basic Definitions

2.1 Petri Nets

A triple (S, T, F) is a *net* if $S \cap T = \emptyset$ and $F \subseteq (S \times T) \cup (T \times S)$. The elements of S are called *places*, and the elements of T *transitions*. Places and transitions are generically called *nodes*. We identify F with its characteristic function on the set $(S \times T) \cup (T \times S)$. The *preset* of a node x, denoted by ${}^\bullet x$, is the set $\{y \in S \cup T \mid F(y, x) = 1\}$. The *postset* of x, denoted by $x^\bullet$, is the set $\{y \in S \cup T \mid F(x, y) = 1\}$.

A *marking* of a net (S, T, F) is a mapping $S \to I\!N$. We identify a marking M with the multiset containing $M(s)$ copies of s for every $s \in S$. A 4-tuple $\Sigma = (S, T, F, M_0)$ is a *net system* if (S, T, F) is a net and M_0 is a marking of (S, T, F) (called the *initial marking* of Σ). A marking M *enables* a transition t if $\forall s \in S\colon F(s, t) \leq M(s)$. If t is enabled at M, then it can *occur*, and its occurrence leads to a new marking M' (denoted $M \xrightarrow{t} M'$), defined by $M'(s) = M(s) - F(s, t) + F(t, s)$ for every place s. 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 Σ is a labelled graph having the set of reachable markings of Σ as nodes and the relations $\xrightarrow{t}$ between markings as edges.

A marking M of a net is *n-safe* if $M(s) \leq n$ for every place s. We identify 1-safe markings with the set of places s such that $M(s) = 1$. A net system Σ is n-safe if all its reachable markings are n-safe.

In this paper we consider only net systems satisfying the following two additional properties:

- The number of places and transitions is finite.

- Every transition of T has a nonempty preset *and* a nonempty postset.

2.2 Occurrence Nets

Let (S, T, F) be a net and let x_1, $x_2 \in S \cup T$. The nodes x_1 and x_2 are in *conflict*, denoted by $x_1 \# x_2$, if there exist distinct transitions $t_1, t_2 \in T$ such that $^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, and (t_1, x_1), (t_2, x_2) belong to the reflexive and transitive closure of F. In other words, x_1 and x_2 are in conflict if there exist 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). For $x \in S \cup T$, x is in *self-conflict* if $x \# x$.

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

- for every $b \in B$, $|^\bullet b| \leq 1$,

- F is acyclic, i.e. the (irreflexive) transitive closure of F is a partial order,

- N 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) belongs to the transitive closure of F is finite, and

- no event $e \in E$ is in self-conflict.

The elements of B and E are called *conditions* and *events*, respectively. $Min(N)$ denotes the set of minimal elements of $B \cup E$ with respect to the transitive closure of F.

The (irreflexive) transitive closure of F is called the *causal relation*, and denoted by $<$. The symbol $\leq$ denotes the reflexive and transitive closure of F. Given two nodes $x, y \in B \cup E$, we say x *co* y if neither $x < y$ nor $y < x$ nor $x \# y$.

2.3 Branching Processes

Branching processes are "unfoldings" of net systems containing information about both concurrency and conflicts. They were introduced by Engelfriet in [4]. We quickly review the main definitions and results of [4].

Let $N_1 = (S_1, T_1, F_1)$ and $N_2 = (S_2, T_2, F_2)$ be two nets. A *homomorphism* from N_1 to N_2[1] is a mapping $h: S_1 \cup T_1 \to S_2 \cup T_2$ such that:

---

[1] In [4], homomorphisms are defined between net systems, instead of between nets, but this is only a small technical difference without any severe consequence.
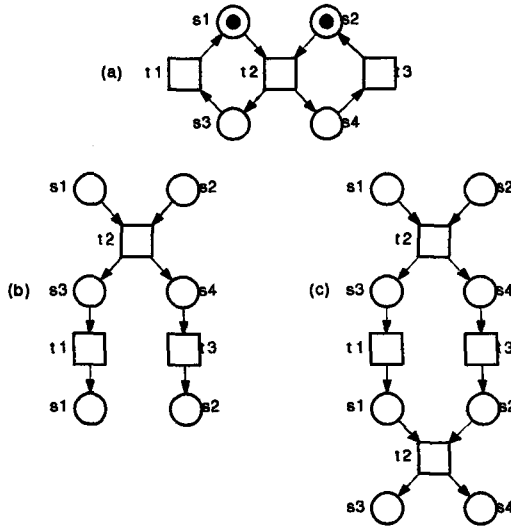
FIGURE 1. A net system and two of its branching processes

- $h(S_1) \subseteq S_2$ and $h(T_1) \subseteq T_2$, and

- for every $t \in T_1$, the restriction of h to ${}^\bullet t$ is a bijection between ${}^\bullet t$ (in N_1) and ${}^\bullet h(t)$ (in N_2), and similarly for $t^\bullet$ and $h(t)^\bullet$.

In other words, a homomorphism is a mapping that preserves the nature of nodes and the environment of transitions.

A *branching process* of a net system $\Sigma = (N, M_0)$ is a pair $\beta = (N', p)$ where $N' = (B, E, F)$ is an occurrence net, and p is a homomorphism from N' to N such that

(i) The restriction of p to $Min(N')$ is a bijection between $Min(N')$ and M_0,

(ii) for every $e_1, e_2 \in E$, if ${}^\bullet e_1 = {}^\bullet e_2$ and $p(e_1) = p(e_2)$ then $e_1 = e_2$.

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

Two branching processes $\beta_1 = (N_1, p_1)$ and $\beta_2 = (N_2, p_2)$ of a net system are *isomorphic* if there is a bijective homomorphism h from N_1 to N_2 such that $p_2 \circ h = p_1$. Intuitively, two isomorphic branching processes differ only in the names of conditions and events.

It is shown in [4] that a net system has a unique maximal branching process up to isomorphism. We call it the *unfolding* of the system. The unfolding of the 1-safe system of Figure 1 is infinite.

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

- if a condition belongs to N', then its input event in N also belongs to N', and

- if an event belongs to N', then its input and output conditions in N also belong to N'.

and p' is the restriction of p to N'.

2.4   Configurations and Cuts

A *configuration* C of an occurrence net is a set of events satisfying the following two conditions:

- $e \in C \Rightarrow \forall e' \le e : e' \in C$ (C is causally closed).

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

A set B' of conditions of an occurrence net is a *co-set* if its elements are pairwise in *co* relation. A maximal co-set B' with respect to set inclusion is called a *cut*.

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

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

In particular, given a finite configuration C the set of places $p(Cut(C))$ is a reachable marking, which we denote by $Mark(C)$.

A marking M of a system Σ is *represented* in a branching process β of Σ if β contains a finite configuration C such that $Mark(C) = M$. 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.

For 1–safe systems, we have the following result, which will be later used in Section 4:

Proposition 2.1

> Let x_1 and x_2 be two nodes of a branching process of a 1-safe net system. If x_1 co x_2, then $p(x_1) \ne p(x_2)$. ∎ 2.1

Given a cut c of a branching process $\beta = (N, p)$, we define $\Uparrow c$ as the pair (N', p'), where N' is the unique subnet of N whose set of nodes is $\{x \mid (\exists y \in c : x \ge y) \land \forall y \in c : \neg(x \# y)\}$ and p' is the restriction of p to the nodes of N'. Further, we define $p(c)$ as the multiset containing an instance of the place $p(b)$ for every $b \in c$. The following result will also be used later:

Proposition 2.2

> If β is a branching process of (N, M_0) and c is a cut of β, then $\Uparrow c$ is a branching process of $(N, p(c))$. ∎ 2.2

3 An Algorithm for the Construction of a Complete Finite Prefix

3.1 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 list $n_1, \ldots, n_k$ of nodes. A node is either a condition or an event. A condition is a pair (s, e), where s is a place of Σ and e the input event. An event is a pair (t, B), where t is a transition of Σ, and B is the set of input conditions. Notice that the flow relation and the labelling function of a branching process are already encoded in its list of nodes. How to express the notions of causal relation, configuration or cut in terms of this data structure is left to the reader.

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. Events are added one at a time together with their output conditions.

We need the notion of "events that can be added to a given branching process".

Definition 3.1

> Let $\beta = n_1, \ldots, n_k$ be a branching process of a net system Σ. The *possible extensions* of β are the pairs (t, B), where B is a co-set of conditions of β and t is a transition of Σ such that
>
> - $p(B) = {}^\bullet t$, and
> - β contains no event e satisfying $p(e) = t$ and ${}^\bullet e = B$
>
> $PE(\beta)$ denotes the set of possible extensions of β.   ■ 3.1

Procedure 3.2 *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**
>   append to Unf an event $e = (t, B)$ of pe and a
>   condition (s, e) for every output place s of t;
> $pe := PE(Unf)$
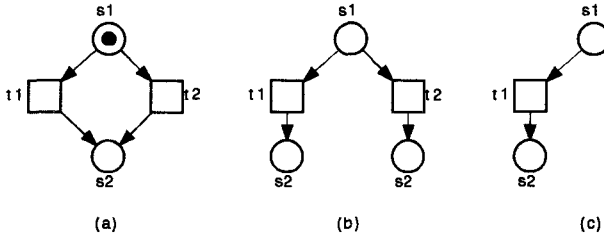> **endwhile**
> **end**

<div align="right">■ 3.2</div>

FIGURE 2. A 1–safe net system, its unfolding, and a prefix

The procedure does not necessarily terminate. In fact, it terminates if and only if the input system Σ does not have any infinite occurrence sequence. It will eventually produce any reachable marking 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 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.

We need some preliminary notations and definitions:

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* to C. Obviously, if $C \subset C'$ then there is a suffix E of C such that $C \oplus E = C'$.

Let C_1 and C_2 be two finite configurations such that $Mark(C_1) = Mark(C_2)$. It follows easily from the definitions that $\Uparrow Cut(C_i)$ is isomorphic to the

unfolding of $\Sigma' = (N, Mark(C_i))$, $i = 1, 2$; hence, $\Uparrow Cut(C_1)$ and $\Uparrow Cut(C_2)$ are isomorphic. Moreover, there is an isomorphism $I_{C_1}^{C_2}$ from $\Uparrow Cut(C_1)$ to $\Uparrow Cut(C_2)$. This isomorphism induces a mapping from the finite extensions of C_1 onto the extensions of C_2: it maps $C_1 \oplus E$ onto $C_2 \oplus I_{C_1}^{C_2}(E)$.

We can now introduce the three basic notions of the algorithm:

Definition 3.3

A partial order $\prec$ on the finite configurations of a branching process is an *adequate order* if:

- $\prec$ is well-founded,

- $\prec$ refines $\subset$, i.e. $C_1 \subset C_2$ implies $C_1 \prec C_2$, and

- $\prec$ is preserved by finite extensions, meaning that if $C_1 \prec C_2$ and $Mark(C_1) = Mark(C_2)$, then $C_1 \oplus E \prec C_2 \oplus I_{C_1}^{C_2}(E)$.

■ 3.3

Definition 3.4 *Local configuration*

The local configuration $[e]$ of an event of a branching process is the set of events e' such that $e' \leq e$.[2]   ■ 3.4

Definition 3.5 *Cut-off event*

Let β be a branching process and let $\prec$ be an adequate partial order on the configurations of β. An event e is a *cut-off event* (with respect to $\prec$) if β contains a local configuration $[e']$ such that

(a) $Mark([e]) = Mark([e'])$, and

(b) $[e'] \prec [e]$.

■ 3.5

The new algorithm has as parameter an adequate order $\prec$, i.e. every different adequate order leads to a different algorithm.

Algorithm 3.6 *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
$Fin := (s_1, \emptyset), \ldots, (s_k, \emptyset)$;
$pe := PE(Fin)$;
$cut\text{-}off := \emptyset$;

---

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

while $pe \neq \emptyset$ **do**
    choose an event $e = (t, B)$ 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

                                                  ■ 3.6

McMillan's algorithm in [10] corresponds to the order

$$C_1 \prec_m C_2 :\Leftrightarrow |C_1| < |C_2|.$$

It is easy to see that $\prec_m$ is adequate.

The reason of condition (a) in the definition of cut-off event is intuitively clear in the light of this algorithm. Since $Mark([e']) = Mark([e])$, the continuations of Unf from $Cut([e])$ and $Cut([e'])$ are isomorphic. Then, loosely speaking, all the reachable markings that we find in the continuation of Unf from $Cut([e])$ are already present in the continuation from $Cut([e'])$, and so there is no need to have the former in *Fin*. The rôle of condition (b) is more technical. In fact, when McMillan's algorithm is applied to "ordinary" small examples, condition (b) seems to be superfluous, and the following strategy seems to work: if an event e is added and *Fin* already contains a local configuration $[e']$ such that $Mark([e]) = Mark([e'])$, then mark e as cut-off event. The following example (also independently found by K. McMillan) shows that this strategy is incorrect. Consider the 1-safe net system of Figure 3.

The marking $\{s_{12}\}$ is reachable. Without condition (b) we can generate the prefix of Figure 4.

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 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. This prefix is not complete, because $\{s_{12}\}$ is not represented in it.

Observe that *Fin* contains all the events of the set *cut-off*. We could modify the algorithm to remove all these events, and the prefix so obtained would still enjoy the property that every reachable marking is represented in it. However, the prefix would not be necessarily complete. Consider for
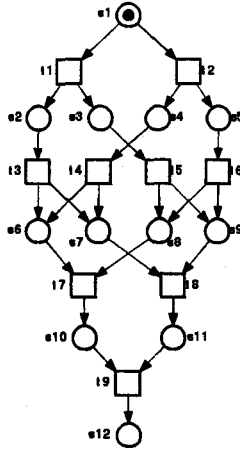
FIGURE 3. A 1–safe net system

example the net system of Figure 2(a). Algorithm 3.6 generates the branching process of Figure 2(b), and one of the two events of the process (the maximal one with respect to ≺) is a cut-off event. If this event is removed, we obtain an incomplete prefix.

We now prove the correctness of Algorithm 3.6.

Proposition 3.7

Fin is finite.

Proof: Given an event e of *Fin*, define the *depth* of e as the length of a longest chain of events
$e_1 < e_2 < \ldots < e$; 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 $e_1 < e_2 < \ldots < e_n < e_{n+1}$ of *Unf* contains two events e_i, e_j, $i < j$, such that $Mark([e_i]) = Mark([e_j])$. Since $[e_i] \subset [e_j]$ and ≺ refines ⊂, 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 homomorphism, there is a bijection between $p(e)^\bullet$ and $p(e^\bullet)$, where p denotes the homomorphism of *Fin*, and similarly for ${}^\bullet p(e)$ and $p({}^\bullet e)$. The result follows from the finiteness of N.
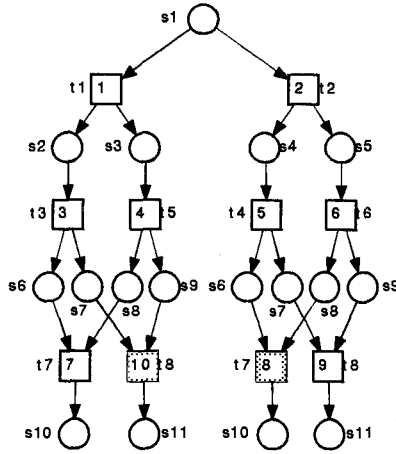
FIGURE 4. A prefix of the net system of Figure 3

(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. Define $E_k^\bullet = \{b | e \in E_k, b \in e^\bullet\}$. 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 (ii) 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.   ■ 3.7

Proposition 3.8

Fin is complete.

Proof: We first prove that every reachable marking of Σ is represented in *Fin*.

Let M be an arbitrary reachable marking of Σ. There exists a configuration C of *Unf* such that $Mark(C) = M$. If C is not a configuration of *Fin*, then it contains some cut-off event e, and so $C = [e] \oplus E$ for some set of events E. By the definition of a cut-off event, there exists a local configuration $[e']$ such that $[e'] \prec [e]$ and $Mark([e']) = Mark([e])$.

Consider the configuration $C' = [e'] \oplus I_{[e]}^{[e']}(E)$. Since $\prec$ is preserved by finite extensions, we have $C' \prec C$. Morever, $Mark(C') = M$. If C' is not a configuration of *Fin*, then we can iterate the procedure

and find a configuration C'' such that $C'' \prec C'$ and $Mark(C'') = M$. The procedure cannot be iterated infinitely often because $\prec$ is well-founded. Therefore, it terminates in a configuration of Fin.

Now we show that Fin is complete. We have to prove that for every reachable marking M there exists a configuration C in β such that:

- $Mark(C) = M$, 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.

Let M be an arbitrary reachable marking of Σ. Since M is represented in Fin, the set of configurations C of Fin satisfying $Mark(C) = M$ is nonempty. By well-foundedness, this set has at least a minimal element C_m with respect to $\prec$. If C_m would contain some cut-off event, then we would find as above another configuration C' satisfying $C' \prec C_m$ and $Mark(C') = M$, which contradicts the minimality of C_m. So C_m contains no cut-off event.

Let t be an arbitrary transition enabled by M. Then there exists a configuration $C_m \cup \{e\}$ of Unf such that $e \notin C_m$ and e is labelled by t. Assume that $C_m \cup \{e\}$ is not a configuration of Fin. Since Fin contains all the events of the set *cut-off* in Algorithm 3.6, it also contains a cut-off event $e' < e$. This implies $e' \in C_m$, which contradicts that C_m contains no cut-off event. So $C_m \cup \{e\}$ is a configuration of Fin. ■ 3.8

4 An Adequate Order for the 1–Safe Case

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 5.

While a minimal complete prefix has size $O(n)$ in the size of the system (see the dotted line in Figure 5), 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 $\prec_m$.

Our parametric presentation of Algorithm 3.6 suggests how to improve this: it suffices to find a new adequate order $\prec_r$ that refines McMillan's order $\prec_m$. Such an order induces a weaker notion of cut-off event; more precisely, every cut-off event with respect to $\prec_m$ is also a cut-off event with respect to $\prec_r$, but maybe not the other way round. Therefore, the instance of Algorithm 3.6 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 3.6 generates a smaller prefix.
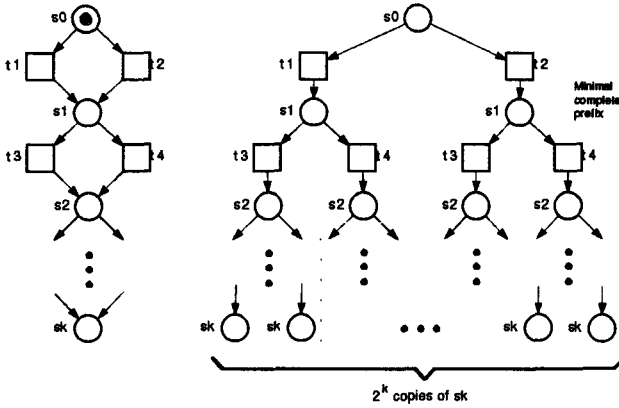
FIGURE 5. A Petri net and its unfolding

The order $\prec_r$ is particularly good if in addition it is *total*. In this case, whenever an event e is generated after some other event e' such that $Mark([e]) = Mark([e'])$, we have $[e'] \prec_r [e]$ (because events are generated in accordance with the total order $\prec_r$), and so e is marked as a cut-off event. So we have the following two properties:

- the guard "e is a cut-off event of *Fin*" in the inner **if** instruction of Algorithm 3.6 can be replaced by "*Fin* contains a local configuration $[e']$ such that $Mark([e]) = Mark([e'])$", and

- the number of events of the complete prefix which are not cut-off events cannot exceed the number of reachable markings.

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 extend $\ll$ to a partial order on sets of events of a branching process as follows: for 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. Now we say that $E_1 \ll E_2$ if $\varphi(E_1)$ is shorter than $\varphi(E_2)$, or if they have the same length but $\varphi(E_1)$ is lexicographically smaller than $\varphi(E_2)$. Note that E_1 and E_2 are incomparable with respect to $\ll$ iff $\varphi(E_1) = \varphi(E_2)$. In particular, if E_1 and E_2 are incomparable with respect to $\ll$, then $|E_1| = |E_2|$.

We now define $\prec_r$ more generally on suffixes of configurations of a branching process (recall that a set of events E is a suffix of a configuration if there exists a configuration C such that $C \oplus E$).

Definition 4.1 *Total order* $\prec_r$

Let E_1 and E_2 be two suffixes of configurations of a branching process β and let $Min(E_1)$ and $Min(E_2)$ denote the sets of minimal elements of E_1 and E_2 with respect to the causal relation. We say $E_1 \prec_r E_2$ if:

- $E_1 \ll E_2$, or
- $\varphi(E_1) = \varphi(E_2)$ and
  - $Min(E_1) \ll Min(E_2)$, or
  - $\varphi(Min(E_1)) = \varphi(Min(E_2))$ and $E_1 \setminus Min(E_1) \prec_r E_2 \setminus Min(E_2)$.

<div style="text-align: right">■ 4.1</div>

Notice that this definition would not be correct for configurations only, because $E \setminus Min(E)$ need not be a configuration even if E is one.

The second condition of this definition could be expressed as: the Foata-Normal-Form of E_1 is smaller than that of E_2 with respect to $\ll$, cf. e.g. [3].

Theorem 4.2

Let β be a branching process of a 1-safe net system. $\prec_r$ is an adequate total order on the configurations of β.

Proof: a) $\prec_r$ is a partial order.

It is easy to see by induction on $|E|$ that $\prec_r$ is irreflexive. Now assume $E_1 \prec_r E_2 \prec_r E_3$. Clearly, $E_1 \prec_r E_3$ unless $\varphi(E_1) = \varphi(E_2) = \varphi(E_3)$, which in particular implies $|E_1| = |E_2| = |E_3|$. For such triples with these equalities we apply induction on the size: if $Min(E_1) \ll Min(E_2)$ or $Min(E_2) \ll Min(E_3)$, we conclude $E_1 \prec_r E_3$, and otherwise we apply induction to $E_i \setminus Min(E_i)$, $i = 1, 2, 3$, which are also suffixes of configurations.

b) $\prec_r$ is total on configurations.

Assume that C_1 and C_2 are two incomparable configurations, i.e. $|C_1| = |C_2|$, $\varphi(C_1) = \varphi(C_2)$, and $\varphi(Min(C_1)) = \varphi(Min(C_2))$. We prove $C_1 = C_2$ by induction on $|C_1| = |C_2|$.

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

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 $^\bullet Min(C_1)$ and $^\bullet Min(C_2)$ are subsets of $Min(N)$, and all the conditions of $Min(N)$ carry different labels by Proposition 2.1, we have $^\bullet e_1 = {}^\bullet e_2$. This contradicts condition (ii) 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 $\Uparrow Cut(Min(C_1))$ of $(N, Mark(Min(C_1)))$ (Proposition 2.2); by induction we conclude $C_1 = C_2$.

c) $\prec_r$ is well-founded.

In a sequence $C_1 \succ_r C_2 \succ_r \ldots$ the size of the C_i cannot decrease infinitely often; also, for configurations of the same size,

C_i cannot decrease infinitely often with respect to $\ll$, since the sequences $\varphi(C_i)$ are drawn from a finite set; an analogous statement holds for $Min(C_i)$. Hence, we assume that all $|C_i|$, all $\varphi(C_i)$ and all $\varphi(Min(C_i))$ are equal and apply induction on the common size. For $|C_i| = 0$, an infinite decreasing sequence is impossible. Otherwise, we conclude as in case b) that we would have $C_1 \setminus Min(C_1) \succ_r C_2 \setminus Min(C_2) \succ_r \ldots$ in $\Uparrow Cut(Min(C_1))$, which is impossible by induction.

d) $\prec_r$ refines $\sqsubseteq$.

Obvious.

e) $\prec_r$ is preserved by finite extensions.

This is the most intricate part of the proof, and here all the complications in Definition 4.1 come into play. Take $C_1 \prec_r C_2$ with $Mark(C_1) = Mark(C_2)$. We have to show that $C_1 \oplus E \prec_r C_2 \oplus I^{C_2}_{C_1}(E)$, and we can assume that $E = \{e\}$ and apply induction afterwards. The case $C_1 \ll C_2$ is easy, hence assume $\varphi(C_1) = \varphi(C_2)$, and in particular $|C_1| = |C_2|$. We show first that e is minimal in $C_1' = C_1 \cup \{e\}$ if and only if $I^{C_2}_{C_1}(e)$ is minimal in $C_2' = C_2 \cup \{I^{C_2}_{C_1}(e)\}$.

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 2.1. 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 $I^{C_2}_{C_1}(e) = e$ 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 $I^{C_2}_{C_1}(e)$ in C_2', we proceed as follows. If $Min(C_1) \ll Min(C_2)$, then we now see that $Min(C_1') \ll Min(C_2')$, so we are done. If $\varphi(Min(C_1)) = \varphi(Min(C_2))$ and $e \in Min(C_1')$, then

$$C_1' \setminus Min(C_1') = C_1 \setminus Min(C_1) \prec_r C_2 \setminus Min(C_2) = C_2' \setminus Min(C_2')$$

hence $C_1' \prec_r C_2'$. 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 $\Uparrow Cut(Min(C_1))$ of $(N, Mark(Min(C_1)))$; with an inductive argument we get $C_1' \setminus Min(C_1') \prec_r C_2' \setminus Min(C_2')$ and are also done in this case. ■ 4.2

We close this section with a remark on the minimality of the prefixes generated by the new algorithm, i.e. by Algorithm 3.6 with $\prec_r$ as adequate order. Figure 1(b) and (c) are a minimal complete prefix and the prefix

generated by the new algorithm for the 1-safe system of Figure 1(a), respectively. It follows that the new algorithm does not always compute a minimal complete prefix.

However, the prefixes computed by the algorithm are minimal in another sense. The algorithm stores only the reachable markings corresponding to local configurations, which for the purpose of this discussion we call *local markings*. This is the feature which makes the algorithm interesting for concurrent systems: the local markings can be a very small subset of the reachable markings, and therefore the storage of the unfolding may require much less memory than the storage of the state space. In order to find out that the prefix of Figure 1(b) is complete, we also need to know that the initial marking $\{s_1, s_2\}$ appears again in the prefix as a non-local marking. If we only store information about local markings, then the prefix of Figure 1(c) is minimal, as well as all the prefixes generated by the new algorithm. The reason is the observation made above: all the local configurations of *Fin* which are not induced by cut-off events correspond to different markings; therefore, in a prefix smaller than *Fin* we lose information about the reachability of some marking.

5   Implementation Issues and Experimental Results

The implementation of the Algorithm 3.6 has been carried out in the context of the model checker described in [5], which allows to efficiently verify formulae expressed in a simple branching time temporal logic.

For the storage of Petri nets and branching processes we have developed an efficient, universal data structure that allows fast access to single nodes [14]. 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 to support fast insertion and deletion of single nodes.

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. The interested reader is referred to [6].

Algorithm 3.6 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 [10], in the implementation we use a queue to store the set PE of possible extensions. The new events of *Fin* are extracted from the head of this list, and, when an event is added, the new possible extensions it generates are appended to its tail.

The simplest way to organize the list would be to sort its events according to the total order $\prec_r$. However, this is again inefficient, because it involves
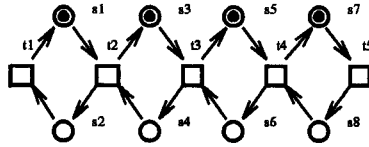
FIGURE 6. n–buffer for $n = 4$.

| | Original net | | | Unfolding | | | time [s] | |
n	$|S|$	$|T|$	$|[M_0\rangle|$	$|B|$	$|E|$	$|cutoffs|$	McMillan	New algorithm
20	40	21	2^{20}	421	211	1	0.22	0.20
40	80	41	2^{40}	1641	821	1	2.40	2.50
60	120	61	2^{60}	3661	1831	1	17.45	18.08
80	160	81	2^{80}	6481	3241	1	66.70	67.85
100	200	101	2^{100}	10101	5051	1	191.58	197.34
120	240	121	2^{120}	14521	7261	1	444.60	437.30
140	280	141	2^{140}	19741	9871	1	871.93	869.50
160	320	161	2^{160}	25761	12881	1	1569.90	1563.74
180	360	181	2^{180}	32581	16291	1	2592.93	2597.86

TABLE 1. Results of the n buffer example[3].

performing unneccessary comparisons. The solution is to sort the events according to the size of their local configuration, as in [10], and compare events with respect to $\prec_r$ only when it is really needed.

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 running time of the new algorithm is $O((\frac{|B|}{\xi})^\xi)$, where B is the set of conditions of the unfolding, and ξ denotes the maximal size of the presets of the transitions in the original net (notice that this is not a measure in the size of the input). The dominating factor in the time complexity is the computation of the possible extensions. The space required is linear in the size of the unfolding because we store a fixed amount of information per event.

Finally, we present some experimental results on three scalable 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 a concurrent n–buffer (see Figure 5). The net has $2n$ places and $n + 1$ transitions, where n is the buffer's capacity. While the number of reachable markings is 2^n, Fin has size $O(n^2)$ and contains one single cut–off event (see Table 1). 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 5, is a model of a slotted ring protocol taken from [13]. Here the size of the prefix produced by the new algorithm grows

---

[3] All the times have been measured on a SPARCstation 20 with 48 MB main memory.
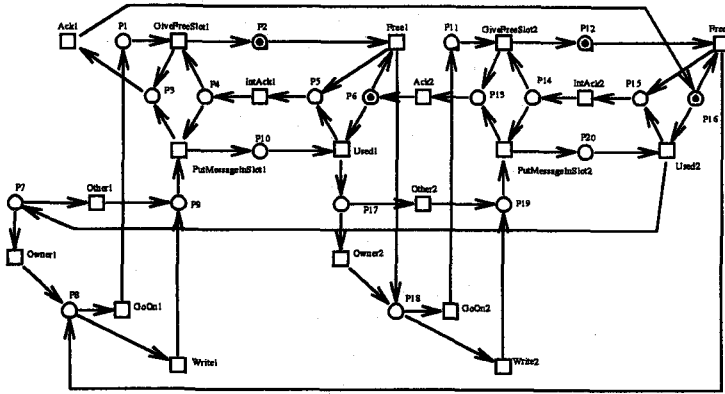
FIGURE 7. Slotted ring protocol for $n = 2$.

	Original net			McMillan's algorithm				New algorithm			
n	$|S|$	$|T|$	$|[M_0\rangle|$	$|B|$	$|E|$	c	time [s]	$|B|$	$|E|$	c	time [s]
1	10	10	$1.2 \cdot 10^1$	18	12	3	0.00	18	12	3	0.00
2	20	20	$2.1 \cdot 10^2$	100	68	12	0.00	90	62	14	0.00
3	30	30	$4.0 \cdot 10^3$	414	288	60	0.13	267	186	42	0.05
4	40	40	$8.2 \cdot 10^4$	1812	1248	296	1.72	740	528	128	0.38
5	50	50	$1.7 \cdot 10^6$	8925	6240	1630	45.31	1805	1280	300	1.58
6	60	60	$3.7 \cdot 10^7$	45846	31104	8508	1829.48	4470	3216	792	11.08
7	70	70	$8.0 \cdot 10^8$				—[4]	10143	7224	1708	79.08
8	80	80	$1.7 \cdot 10^{10}$				—[4]	23880	17216	4256	563.69
9	90	90	$3.8 \cdot 10^{11}$				—[4]	52209	37224	8820	2850.89
10	100	100	$8.1 \cdot 10^{12}$				—[4]	119450	86160	21320	15547.67

TABLE 2. Results of the slotted ring protocol example; $c = |cutoffs|$.

more slowly than in the case of McMillan's algorithm. For $n = 6$ the output is already one order of magnitude smaller. The slow growth in size can cause an even more dramatic reduction in the running time.

	Original net			McMillan's algorithm				New algorithm			
n	$|S|$	$|T|$	$|[M_0\rangle|$	$|B|$	$|E|$	c	time [s]	$|B|$	$|E|$	c	time [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

TABLE 3. Results of Milner's cyclic scheduler; $c = |cutoffs|$.

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.

---

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

6 Conclusions

We have presented an algorithm for the computation of a complete finite prefix of an unfolding. We have used 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 larger than the reachability graph, which does not hold for the algorithm of [10].

Recently, Kondratyev *et al.* have independently found another partial order between events which permits to obtain reduced unfoldings [9]. Their technique works for bounded nets. However, the partial order is not total, and so the upper bound on the size of the unfolding cannot be derived.

Acknowledgements

7 REFERENCES

[1] E. Best and C. Fernández: Nonsequential Processes – A Petri Net View. EATCS Monographs on Theoretical Computer Science 13 (1988).

[2] James C. Corbett: Evaluating Deadlock Detection Methods. University of Hawaii at Manoa (1994).

[3] V. Diekert: Combinatorics on Traces. LNCS 454 (1990).

[4] J. Engelfriet: Branching processes of Petri nets. Acta Informatica 28, pp. 575–591 (1991).

[5] J. Esparza: Model Checking Using Net Unfoldings. Science of Computer Programming 23, pp. 151–195 (1994).

[6] J. Esparza, S. Römer and W. Vogler: An improvement of McMillan's unfolding algorithm. Informatik Bericht, TU München, in preparation.

[7] M. Kishinevsky, A. Kondratyev, A. Taubin, and V. Varshavsky: Concurrent Hardware: The Theory and Practice of Self-Timed Design, Wiley (1993).

[8] A. Kondratyev and A. Taubin: Verification of speed-independent circuits by STG unfoldings. Proceedings of the Symposium on Advanced Research in Asynchronous Circuits and Systems, Utah (1994).

[9] A. Kondratyev, A. Taubin, M. Kishinevsky and S. Ten: Analysis of Petri Nets by Ordering Relations. Technical Report TR:95-2-002, University of Aizu (1995).

[10] K.L. McMillan: A Technique of a State Space Search Based on Unfolding. Formal Methods in System Design 6(1), pp. 45–65 (1995)

[11] K.L. McMillan: Trace theoretic verification of asynchronous circuits using unfoldings. Proceedings of the 7th Workshop on Computer Aided Verification, Liege (1995).

[12] M. Nielsen, G. Plotkin and G. Winskel: Petri Nets, Event Structures and Domains. Theoretical Computer Science 13(1), pp. 85–108 (1980).

[13] E. Pastor, O. Roig, J. Cortadella and R.M. Badia: Petri Net Analysis Using Boolean Manipulation. Proc. Application and Theory of Petri Nets '94, LNCS 815, pp. 416–435 (1994).

[14] S. Römer: Implementation of a Compositional Partial Order Semantics of Petri Boxes. Diploma Thesis (in German). Universität Hildesheim (1993).