TECHNISCHE UNIVERSITÄT MÜNCHEN

INSTITUT FÜR INFORMATIK

Sonderforschungsbereich 342:
Methoden und Werkzeuge für die Nutzung
paralleler Rechnerarchitekturen

An Improvement of McMillan's Unfolding Algorithm

Javier Esparza, Stefan Römer, Walter Vogler

TUM-I9525
SFB-Bericht Nr.342/12/95 A
August 1995

TUM–INFO–08-95-I25-350/1.–FI

Anforderungen an: Prof. Dr. A. Bode
Sprecher SFB 342
Institut für Informatik
Technische Universität München
Arcisstr. 21 / Postfach 20 24 20
D-80290 München, Germany

Druck: Fakultät für Informatik der
Technischen Universität München

An Improvement of McMillan's Unfolding Algorithm

Javier Esparza, Stefan Römer*
Institut für Informatik, Technische Universität München

Walter Vogler
Institut für Mathematik, Universität Augsburg

1 Introduction

In a seminal paper [8], 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 [10], and later described in more detail in [3] 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 been later applied to other verification problems. In [6, 7] it is used to check relevant properties of speed independent circuits. In [4], an unfolding-based model checking algorithm for a simple branching time logic is proposed. Recently, the technique has also been used for the verification of timed systems [9] .

Although McMillan's algorithm is simple and elegant, it sometimes generates 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.

*Partially supported by the Teilproject A3 SAM of the Sonderforschungsbereich 342 " Werkzeuge und Methoden für die Nutzung paralleler Rechnerarchitekturen".

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 $S \to I\!N$. 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, ..., 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 $M_0 \xrightarrow{\sigma} M$.
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 Branching processes

Branching processes are "unfoldings" of net systems containing information about both concurrency and conflicts. They were introduced by Engelfriet in [3]. We quickly review the main definitions and results of [3].

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 $x \# y$.

Branching processes. 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 $h: S_1 \cup T_1 \rightarrow S_2 \cup T_2$ such that:

- $h(S_1) \subseteq S_2$ 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 [3] 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

- $Min(N)$ belongs to N',
- 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'.

[1] In [3], homomorphisms are defined between net systems, instead of between nets, but this is only a small technical difference without any severe consequence.

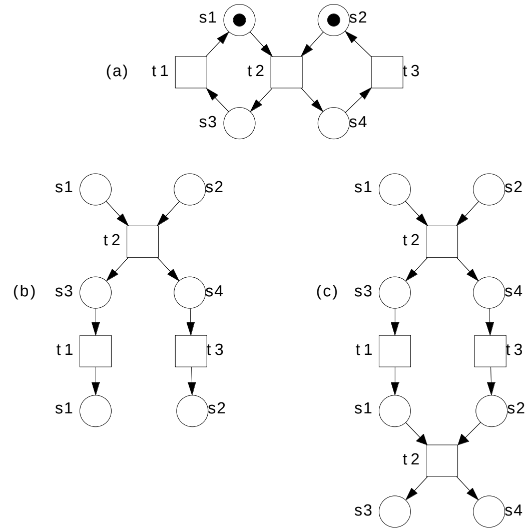


Figure 1: A net system and two of its branching processes

2.3 Configurations and cuts

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

- $e \in C \;\Rightarrow\; \forall e' \leq e \colon e' \in C$ (C is causally closed).
- $\forall e, e' \in C \colon \neg(e \# e')$ (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*.
A marking M of a system Σ is *represented* in a branching process $\beta = (N, p)$ of Σ if β contains a cut c such that, for each place s of Σ, c contains exactly $M(s)$ conditions b with $p(b) = s$. It is easy to prove using results of [1, 3] that every marking represented in a branching process is reachable, and that every reachable marking is represented in the unfolding of the net system.
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 configuration C the set of places $p(Cut(C))$ is a reachable marking, which we denote by $Mark(C)$.
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 $p(x_1) \neq p(x_2)$. ■ 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 $\{x \mid (\exists y \in c : x \geq y) \wedge \forall y \in c : \neg(x \# y)\}$ and p' is the restriction of p to the nodes of N. 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

Algorithm 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

■ 3.2

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 [3]).

Constructing a finite complete prefix

We say that a branching process β of a net system Σ is *complete* if the following two conditions hold:

- every reachable marking of Σ is represented in β, and
- if a transition t can occur in Σ, then β contains an event labelled by t.

The unfolding of a net system is always complete. It is also easy to see that a complete prefix contains as much information as the unfolding.
Since an n-safe net system has only finitely many reachable markings, its unfolding contains at least one complete finite prefix. We show how to transform the algorithm above into a new one whose output is a finite complete 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* of 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

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

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-off $:= \emptyset$;
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 [8] 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

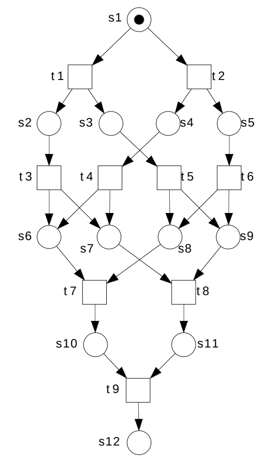


Figure 2: A 1–safe net system

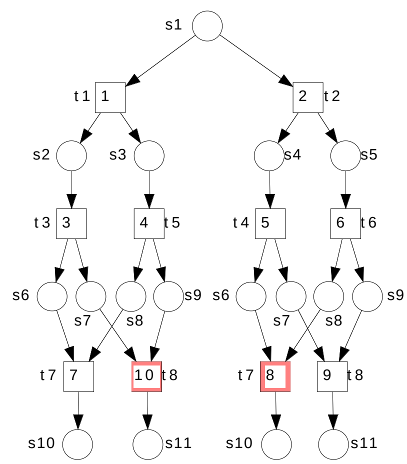


Figure 3: A prefix of the net system of Figure 2

(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 $\mathit{Mark}([e]) = \mathit{Mark}([e'])$, 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 2.

The marking $\{s_{12}\}$ is reachable. Without condition (b) we can generate the prefix of Figure 3.

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.

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 two 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 $\prec$ refines $\subset$, 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.

(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 (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: (a) 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

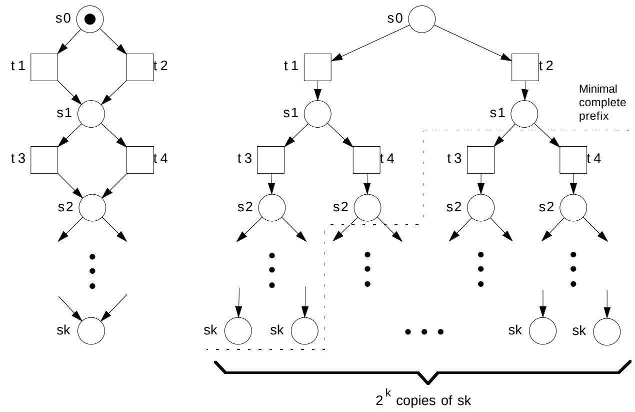


Figure 4: A Petri net and its unfolding

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

(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 $Mark(Cut(C)) = M$. 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 $Mark(Cut(C')) = M$. 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. ■ 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 4.

While a minimal complete prefix has size $O(n)$ in the size of the system (see the dotted line in Figure 4), 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.

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 rechable 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: for such a set E, 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; $\varphi(E)$ is something like the Parikh-vector of E. 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 $\varphi(E_1) = \varphi(E_2)$ and, in particular, $|E_1| = |E_2|$.
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 $E_1 \setminus Min(E_1) \prec_r E_2 \setminus Min(E_2)$.

■ 4.1

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. [2].

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 $\subset$.

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 $C_1 \oplus E \prec_r C_2 \oplus I_{C_1}^{C_2}(E)$, 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 $I_{C_1}^{C_2}(e)$ is minimal in $C_2' = C_2 \cup \{I_{C_1}^{C_2}(e)\}$.

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 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 $I_{C_1}^{C_2}(e) = e$ 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 $I_{C_1}^{C_2}(e)$ 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.

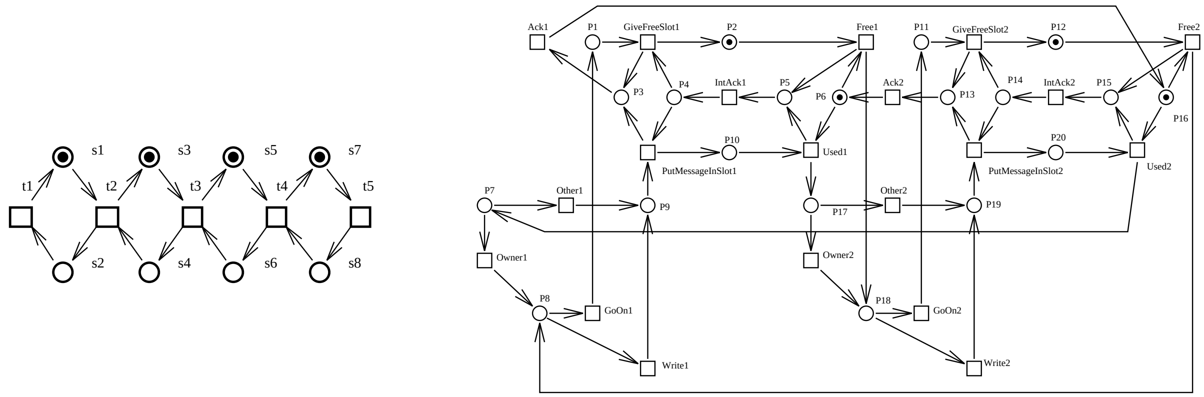
However, the prefixes computed by the algorithm are minimal in another sense. The algorithm stores only the reachable markings corresponding to local configurations, which for the purpose of this discussion we call *local markings*. This is the feature which makes the algorithm interesting for concurrent 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 [4], 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 [12]. 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 [12].



(a) n–buffer for $n = 4$.

(b) Slotted ring protocol for $n = 2$.

Figure 5: Two scalable nets

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 [8], 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 performing unneccessary comparisons. The solution is to sort the events according to the size of their local configuration, as in [8], 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 finite amount of information pro event.

Finally, we present some experimental results on two 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(a)). 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).

	Original net			Unfolding			time [s]	
n	$\lvert S\rvert$	$\lvert T\rvert$	$\lvert [M_0\rangle\rvert$	$\lvert B\rvert$	$\lvert E\rvert$	$\lvert cutoffs\rvert$	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].

	Original net			McMillan's algorithm				New algorithm			
n	$\lvert S\rvert$	$\lvert T\rvert$	$\lvert [M_0\rangle\rvert$	$\lvert B\rvert$	$\lvert E\rvert$	$\lvert cutoffs\rvert$	time [s]	$\lvert B\rvert$	$\lvert E\rvert$	$\lvert cutoffs\rvert$	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.

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(b), is a model of a slotted ring protocol taken from [11]. 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.

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 significantly larger than the reachability graph, what did not hold for the algorithm of [8].

Acknowledgements

We thank Michael Kishinevsky, Alexander Taubin and Alex Yakovlev for drawing our attention to this problem, and Burkhard Graves for detecting some mistakes.

[3]All the times have been measured on a SPARCstation 20 with 48 MB main memory.

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

References

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

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

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

[4] J. Esparza: Model Checking Using Net Unfoldings. Science of Computer Programming (1993).

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

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

[7] 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).

[8] K.L. McMillan: Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. Proceedings of the 4th Workshop on Computer Aided Verification, Montreal, pp. 164–174 (1992).

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

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

[11] 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).

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

SFB 342: Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

bisher erschienen :

Reihe A

342/1/90 A	Robert Gold, Walter Vogler: Quality Criteria for Partial Order Semantics of Place/Transition-Nets, Januar 1990
342/2/90 A	Reinhard Fößmeier: Die Rolle der Lastverteilung bei der numerischen Parallelprogrammierung, Februar 1990
342/3/90 A	Klaus-Jörn Lange, Peter Rossmanith: Two Results on Unambiguous Circuits, Februar 1990
342/4/90 A	Michael Griebel: Zur Lösung von Finite-Differenzen- und Finite-Element-Gleichungen mittels der Hierarchischen Transformations-Mehrgitter-Methode
342/5/90 A	Reinhold Letz, Johann Schumann, Stephan Bayerl, Wolfgang Bibel: SETHEO: A High-Performance Theorem Prover
342/6/90 A	Johann Schumann, Reinhold Letz: PARTHEO: A High Performance Parallel Theorem Prover
342/7/90 A	Johann Schumann, Norbert Trapp, Martin van der Koelen: SETHEO/PARTHEO Users Manual
342/8/90 A	Christian Suttner, Wolfgang Ertel: Using Connectionist Networks for Guiding the Search of a Theorem Prover
342/9/90 A	Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Jaroslav Kremenek, Robert Lindhof, Thomas Ludwig, Peter Luksch, Thomas Treml: TOPSYS, Tools for Parallel Systems (Artikelsammlung)
342/10/90 A	Walter Vogler: Bisimulation and Action Refinement
342/11/90 A	Jörg Desel, Javier Esparza: Reachability in Reversible Free- Choice Systems
342/12/90 A	Rob van Glabbeek, Ursula Goltz: Equivalences and Refinement
342/13/90 A	Rob van Glabbeek: The Linear Time - Branching Time Spectrum
342/14/90 A	Johannes Bauer, Thomas Bemmerl, Thomas Treml: Leistungsanalyse von verteilten Beobachtungs- und Bewertungswerkzeugen
342/15/90 A	Peter Rossmanith: The Owner Concept for PRAMs
342/16/90 A	G. Böckle, S. Trosch: A Simulator for VLIW-Architectures
342/17/90 A	P. Slavkovsky, U. Rüde: Schnellere Berechnung klassischer Matrix-Multiplikationen

342/18/90 A	Christoph Zenger: SPARSE GRIDS
342/19/90 A	Michael Griebel, Michael Schneider, Christoph Zenger: A combination technique for the solution of sparse grid problems
342/20/90 A	Michael Griebel: A Parallelizable and Vectorizable Multi- Level-Algorithm on Sparse Grids
342/21/90 A	V. Diekert, E. Ochmanski, K. Reinhardt: On confluent semi-commutations-decidability and complexity results
342/22/90 A	Manfred Broy, Claus Dendorfer: Functional Modelling of Operating System Structures by Timed Higher Order Stream Processing Functions
342/23/90 A	Rob van Glabbeek, Ursula Goltz: A Deadlock-sensitive Congruence for Action Refinement
342/24/90 A	Manfred Broy: On the Design and Verification of a Simple Distributed Spanning Tree Algorithm
342/25/90 A	Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Peter Luksch, Roland Wismüller: TOPSYS - Tools for Parallel Systems (User's Overview and User's Manuals)
342/26/90 A	Thomas Bemmerl, Arndt Bode, Thomas Ludwig, Stefan Tritscher: MMK - Multiprocessor Multitasking Kernel (User's Guide and User's Reference Manual)
342/27/90 A	Wolfgang Ertel: Random Competition: A Simple, but Efficient Method for Parallelizing Inference Systems
342/28/90 A	Rob van Glabbeek, Frits Vaandrager: Modular Specification of Process Algebras
342/29/90 A	Rob van Glabbeek, Peter Weijland: Branching Time and Abstraction in Bisimulation Semantics
342/30/90 A	Michael Griebel: Parallel Multigrid Methods on Sparse Grids
342/31/90 A	Rolf Niedermeier, Peter Rossmanith: Unambiguous Simulations of Auxiliary Pushdown Automata and Circuits
342/32/90 A	Inga Niepel, Peter Rossmanith: Uniform Circuits and Exclusive Read PRAMs
342/33/90 A	Dr. Hermann Hellwagner: A Survey of Virtually Shared Memory Schemes
342/1/91 A	Walter Vogler: Is Partial Order Semantics Necessary for Action Refinement?
342/2/91 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Rainer Weber: Characterizing the Behaviour of Reactive Systems by Trace Sets
342/3/91 A	Ulrich Furbach, Christian Suttner, Bertram Fronhöfer: Massively Parallel Inference Systems

342/4/91 A Rudolf Bayer: Non-deterministic Computing, Transactions and Recursive Atomicity
342/5/91 A Robert Gold: Dataflow semantics for Petri nets
342/6/91 A A. Heise; C. Dimitrovici: Transformation und Komposition von P/T-Netzen unter Erhaltung wesentlicher Eigenschaften
342/7/91 A Walter Vogler: Asynchronous Communication of Petri Nets and the Refinement of Transitions
342/8/91 A Walter Vogler: Generalized OM-Bisimulation
342/9/91 A Christoph Zenger, Klaus Hallatschek: Fouriertransformation auf dünnen Gittern mit hierarchischen Basen
342/10/91 A Erwin Loibl, Hans Obermaier, Markus Pawlowski: Towards Parallelism in a Relational Database System
342/11/91 A Michael Werner: Implementierung von Algorithmen zur Kompaktifizierung von Programmen für VLIW-Architekturen
342/12/91 A Reiner Müller: Implementierung von Algorithmen zur Optimierung von Schleifen mit Hilfe von Software-Pipelining Techniken
342/13/91 A Sally Baker, Hans-Jörg Beier, Thomas Bemmerl, Arndt Bode, Hubert Ertl, Udo Graf, Olav Hansen, Josef Haunerdinger, Paul Hofstetter, Rainer Knödlseder, Jaroslav Kremenek, Siegfried Langenbuch, Robert Lindhof, Thomas Ludwig, Peter Luksch, Roy Milner, Bernhard Ries, Thomas Treml: TOPSYS - Tools for Parallel Systems (Artikelsammlung); 2., erweiterte Auflage
342/14/91 A Michael Griebel: The combination technique for the sparse grid solution of PDE's on multiprocessor machines
342/15/91 A Thomas F. Gritzner, Manfred Broy: A Link Between Process Algebras and Abstract Relation Algebras?
342/16/91 A Thomas Bemmerl, Arndt Bode, Peter Braun, Olav Hansen, Thomas Treml, Roland Wismüller: The Design and Implementation of TOPSYS
342/17/91 A Ulrich Furbach: Answers for disjunctive logic programs
342/18/91 A Ulrich Furbach: Splitting as a source of parallelism in disjunctive logic programs
342/19/91 A Gerhard W. Zumbusch: Adaptive parallele Multilevel-Methoden zur Lösung elliptischer Randwertprobleme
342/20/91 A M. Jobmann, J. Schumann: Modelling and Performance Analysis of a Parallel Theorem Prover
342/21/91 A Hans-Joachim Bungartz: An Adaptive Poisson Solver Using Hierarchical Bases and Sparse Grids

342/22/91 A	Wolfgang Ertel, Theodor Gemenis, Johann M. Ph. Schumann, Christian B. Suttner, Rainer Weber, Zongyan Qiu: Formalisms and Languages for Specifying Parallel Inference Systems
342/23/91 A	Astrid Kiehn: Local and Global Causes
342/24/91 A	Johann M.Ph. Schumann: Parallelization of Inference Systems by using an Abstract Machine
342/25/91 A	Eike Jessen: Speedup Analysis by Hierarchical Load Decomposition
342/26/91 A	Thomas F. Gritzner: A Simple Toy Example of a Distributed System: On the Design of a Connecting Switch
342/27/91 A	Thomas Schnekenburger, Andreas Weininger, Michael Friedrich: Introduction to the Parallel and Distributed Programming Language ParMod-C
342/28/91 A	Claus Dendorfer: Funktionale Modellierung eines Postsystems
342/29/91 A	Michael Griebel: Multilevel algorithms considered as iterative methods on indefinite systems
342/30/91 A	W. Reisig: Parallel Composition of Liveness
342/31/91 A	Thomas Bemmerl, Christian Kasperbauer, Martin Mairandres, Bernhard Ries: Programming Tools for Distributed Multiprocessor Computing Environments
342/32/91 A	Frank Leßke: On constructive specifications of abstract data types using temporal logic
342/1/92 A	L. Kanal, C.B. Suttner (Editors): Informal Proceedings of the Workshop on Parallel Processing for AI
342/2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS
342/2-2/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: The Design of Distributed Systems - An Introduction to FOCUS - Revised Version (erschienen im Januar 1993)
342/3/92 A	Manfred Broy, Frank Dederichs, Claus Dendorfer, Max Fuchs, Thomas F. Gritzner, Rainer Weber: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/4/92 A	Claus Dendorfer, Rainer Weber: Development and Implementation of a Communication Protocol - An Exercise in FOCUS
342/5/92 A	Michael Friedrich: Sprachmittel und Werkzeuge zur Unterstüt- zung paralleler und verteilter Programmierung
342/6/92 A	Thomas F. Gritzner: The Action Graph Model as a Link between Abstract Relation Algebras and Process-Algebraic Specifications

342/7/92 A	Sergei Gorlatch: Parallel Program Development for a Recursive Numerical Algorithm: a Case Study
342/8/92 A	Henning Spruth, Georg Sigl, Frank Johannes: Parallel Algorithms for Slicing Based Final Placement
342/9/92 A	Herbert Bauer, Christian Sporrer, Thomas Krodel: On Distributed Logic Simulation Using Time Warp
342/10/92 A	H. Bungartz, M. Griebel, U. Rüde: Extrapolation, Combination and Sparse Grid Techniques for Elliptic Boundary Value Problems
342/11/92 A	M. Griebel, W. Huber, U. Rüde, T. Störtkuhl: The Combination Technique for Parallel Sparse-Grid-Preconditioning and -Solution of PDEs on Multiprocessor Machines and Workstation Networks
342/12/92 A	Rolf Niedermeier, Peter Rossmanith: Optimal Parallel Algorithms for Computing Recursively Defined Functions
342/13/92 A	Rainer Weber: Eine Methodik für die formale Anforderungsspezifkation verteilter Systeme
342/14/92 A	Michael Griebel: Grid– and point–oriented multilevel algorithms
342/15/92 A	M. Griebel, C. Zenger, S. Zimmer: Improved multilevel algorithms for full and sparse grid problems
342/16/92 A	J. Desel, D. Gomm, E. Kindler, B. Paech, R. Walter: Bausteine eines kompositionalen Beweiskalküls für netzmodellierte Systeme
342/17/92 A	Frank Dederichs: Transformation verteilter Systeme: Von applikativen zu prozeduralen Darstellungen
342/18/92 A	Andreas Listl, Markus Pawlowski: Parallel Cache Management of a RDBMS
342/19/92 A	Erwin Loibl, Markus Pawlowski, Christian Roth: PART: A Parallel Relational Toolbox as Basis for the Optimization and Interpretation of Parallel Queries
342/20/92 A	Jörg Desel, Wolfgang Reisig: The Synthesis Problem of Petri Nets
342/21/92 A	Robert Balder, Christoph Zenger: The d-dimensional Helmholtz equation on sparse Grids
342/22/92 A	Ilko Michler: Neuronale Netzwerk-Paradigmen zum Erlernen von Heuristiken
342/23/92 A	Wolfgang Reisig: Elements of a Temporal Logic. Coping with Concurrency
342/24/92 A	T. Störtkuhl, Chr. Zenger, S. Zimmer: An asymptotic solution for the singularity at the angular point of the lid driven cavity
342/25/92 A	Ekkart Kindler: Invariants, Compositionality and Substitution
342/26/92 A	Thomas Bonk, Ulrich Rüde: Performance Analysis and Optimization of Numerically Intensive Programs

342/1/93 A	M. Griebel, V. Thurner: The Efficient Solution of Fluid Dynamics Problems by the Combination Technique
342/2/93 A	Ketil Stølen, Frank Dederichs, Rainer Weber: Assumption / Commitment Rules for Networks of Asynchronously Communicating Agents
342/3/93 A	Thomas Schnekenburger: A Definition of Efficiency of Parallel Programs in Multi-Tasking Environments
342/4/93 A	Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: A Proof of Convergence for the Combination Technique for the Laplace Equation Using Tools of Symbolic Computation
342/5/93 A	Manfred Kunde, Rolf Niedermeier, Peter Rossmanith: Faster Sorting and Routing on Grids with Diagonals
342/6/93 A	Michael Griebel, Peter Oswald: Remarks on the Abstract Theory of Additive and Multiplicative Schwarz Algorithms
342/7/93 A	Christian Sporrer, Herbert Bauer: Corolla Partitioning for Distributed Logic Simulation of VLSI Circuits
342/8/93 A	Herbert Bauer, Christian Sporrer: Reducing Rollback Overhead in Time-Warp Based Distributed Simulation with Optimized Incremental State Saving
342/9/93 A	Peter Slavkovsky: The Visibility Problem for Single-Valued Surface (z = f(x,y)): The Analysis and the Parallelization of Algorithms
342/10/93 A	Ulrich Rüde: Multilevel, Extrapolation, and Sparse Grid Methods
342/11/93 A	Hans Regler, Ulrich Rüde: Layout Optimization with Algebraic Multigrid Methods
342/12/93 A	Dieter Barnard, Angelika Mader: Model Checking for the Modal Mu-Calculus using Gauß Elimination
342/13/93 A	Christoph Pflaum, Ulrich Rüde: Gauß' Adaptive Relaxation for the Multilevel Solution of Partial Differential Equations on Sparse Grids
342/14/93 A	Christoph Pflaum: Convergence of the Combination Technique for the Finite Element Solution of Poisson's Equation
342/15/93 A	Michael Luby, Wolfgang Ertel: Optimal Parallelization of Las Vegas Algorithms
342/16/93 A	Hans-Joachim Bungartz, Michael Griebel, Dierk Röschke, Christoph Zenger: Pointwise Convergence of the Combination Technique for Laplace's Equation
342/17/93 A	Georg Stellner, Matthias Schumann, Stefan Lamberts, Thomas Ludwig, Arndt Bode, Martin Kiehl und Rainer Mehlhorn: Developing Multicomputer Applications on Networks of Workstations Using NXLib

342/18/93 A	Max Fuchs, Ketil Stølen: Development of a Distributed Min/Max Component
342/19/93 A	Johann K. Obermaier: Recovery and Transaction Management in Write-optimized Database Systems
342/20/93 A	Sergej Gorlatch: Deriving Efficient Parallel Programs by Systemating Coarsing Specification Parallelism
342/01/94 A	Reiner Hüttl, Michael Schneider: Parallel Adaptive Numerical Simulation
342/02/94 A	Henning Spruth, Frank Johannes: Parallel Routing of VLSI Circuits Based on Net Independency
342/03/94 A	Henning Spruth, Frank Johannes, Kurt Antreich: PHIroute: A Parallel Hierarchical Sea-of-Gates Router
342/04/94 A	Martin Kiehl, Rainer Mehlhorn, Matthias Schumann: Parallel Multiple Shooting for Optimal Control Problems Under NX/2
342/05/94 A	Christian Suttner, Christoph Goller, Peter Krauss, Klaus-Jörn Lange, Ludwig Thomas, Thomas Schnekenburger: Heuristic Optimization of Parallel Computations
342/06/94 A	Andreas Listl: Using Subpages for Cache Coherency Control in Parallel Database Systems
342/07/94 A	Manfred Broy, Ketil Stølen: Specification and Refinement of Finite Dataflow Networks - a Relational Approach
342/08/94 A	Katharina Spies: Funktionale Spezifikation eines Kommunikationsprotokolls
342/09/94 A	Peter A. Krauss: Applying a New Search Space Partitioning Method to Parallel Test Generation for Sequential Circuits
342/10/94 A	Manfred Broy: A Functional Rephrasing of the Assumption/Commitment Specification Style
342/11/94 A	Eckhardt Holz, Ketil Stølen: An Attempt to Embed a Restricted Version of SDL as a Target Language in Focus
342/12/94 A	Christoph Pflaum: A Multi-Level-Algorithm for the Finite-Element-Solution of General Second Order Elliptic Differential Equations on Adaptive Sparse Grids
342/13/94 A	Manfred Broy, Max Fuchs, Thomas F. Gritzner, Bernhard Schätz, Katharina Spies, Ketil Stølen: Summary of Case Studies in FOCUS - a Design Method for Distributed Systems
342/14/94 A	Maximilian Fuchs: Technologieabhängigkeit von Spezifikationen digitaler Hardware
342/15/94 A	M. Griebel, P. Oswald: Tensor Product Type Subspace Splittings And Multilevel Iterative Methods For Anisotropic Problems

342/16/94 A Gheorghe Ştefănescu: Algebra of Flownomials
342/17/94 A Ketil Stølen: A Refinement Relation Supporting the Transition from Unbounded to Bounded Communication Buffers
342/18/94 A Michael Griebel, Tilman Neuhoeffer: A Domain-Oriented Multilevel Algorithm-Implementation and Parallelization
342/19/94 A Michael Griebel, Walter Huber: Turbulence Simulation on Sparse Grids Using the Combination Method
342/20/94 A Johann Schumann: Using the Theorem Prover SETHEO for verifying the development of a Communication Protocol in FOCUS - A Case Study -
342/01/95 A Hans-Joachim Bungartz: Higher Order Finite Elements on Sparse Grids
342/02/95 A Tao Zhang, Seonglim Kang, Lester R. Lipsky: The Performance of Parallel Computers: Order Statistics and Amdahl's Law
342/03/95 A Lester R. Lipsky, Appie van de Liefvoort: Transformation of the Kronecker Product of Identical Servers to a Reduced Product Space
342/04/95 A Pierre Fiorini, Lester R. Lipsky, Wen-Jung Hsin, Appie van de Liefvoort: Auto-Correlation of Lag-k For Customers Departing From Semi-Markov Processes
342/05/95 A Sascha Hilgenfeldt, Robert Balder, Christoph Zenger: Sparse Grids: Applications to Multi-dimensional Schrödinger Problems
342/06/95 A Maximilian Fuchs: Formal Design of a Model-N Counter
342/07/95 A Hans-Joachim Bungartz, Stefan Schulte: Coupled Problems in Microsystem Technology
342/08/95 A Alexander Pfaffinger: Parallel Communication on Workstation Networks with Complex Topologies
342/09/95 A Ketil Stølen: Assumption/Commitment Rules for Data-flow Networks - with an Emphasis on Completeness
342/10/95 A Ketil Stølen, Max Fuchs: A Formal Method for Hardware/Software Co-Design
342/11/95 A Thomas Schnekenburger: The ALDY Load Distribution System
342/12/95 A Javier Esparza, Stefan Römer, Walter Vogler: An Improvement of McMillan's Unfolding Algorithm

SFB 342 : Methoden und Werkzeuge für die Nutzung paralleler Rechnerarchitekturen

Reihe B

342/1/90 B	Wolfgang Reisig: Petri Nets and Algebraic Specifications
342/2/90 B	Jörg Desel: On Abstraction of Nets
342/3/90 B	Jörg Desel: Reduction and Design of Well-behaved Free-choice Systems
342/4/90 B	Franz Abstreiter, Michael Friedrich, Hans-Jürgen Plewan: Das Werkzeug runtime zur Beobachtung verteilter und paralleler Programme
342/1/91 B	Barbara Paech1: Concurrency as a Modality
342/2/91 B	Birgit Kandler, Markus Pawlowski: SAM: Eine Sortier- Toolbox -Anwenderbeschreibung
342/3/91 B	Erwin Loibl, Hans Obermaier, Markus Pawlowski: 2. Workshop über Parallelisierung von Datenbanksystemen
342/4/91 B	Werner Pohlmann: A Limitation of Distributed Simulation Methods
342/5/91 B	Dominik Gomm, Ekkart Kindler: A Weakly Coherent Virtually Shared Memory Scheme: Formal Specification and Analysis
342/6/91 B	Dominik Gomm, Ekkart Kindler: Causality Based Specification and Correctness Proof of a Virtually Shared Memory Scheme
342/7/91 B	W. Reisig: Concurrent Temporal Logic
342/1/92 B	Malte Grosse, Christian B. Suttner: A Parallel Algorithm for Set-of-Support Christian B. Suttner: Parallel Computation of Multiple Sets-of-Support
342/2/92 B	Arndt Bode, Hartmut Wedekind: Parallelrechner: Theorie, Hardware, Software, Anwendungen
342/1/93 B	Max Fuchs: Funktionale Spezifikation einer Geschwindigkeitsregelung
342/2/93 B	Ekkart Kindler: Sicherheits- und Lebendigkeitseigenschaften: Ein Literaturüberblick
342/1/94 B	Andreas Listl; Thomas Schnekenburger; Michael Friedrich: Zum Entwurf eines Prototypen für MIDAS