On Unfolding Unbounded Petri Nets

Parosh Aziz Abdulla
Dept. of Computer Systems
P.O. Box 325
Uppsala University
S-751 05 Uppsala, Sweden
EMail: parosh@docs.uu.se

S. Purushothaman Iyer*
Dept of Computer Science
NC State University
Raleigh, NC 27695-7534
EMail: purush@csc.ncsu.edu

Aletta Nylén
Dept. of Computer Systems
P.O. Box 325
Uppsala University
S-751 05 Uppsala, Sweden
EMail: aletta@docs.uu.se

July 5, 2001

Abstract

Net unfoldings have attracted great attention as a powerful technique for combating state space explosion in model checking, and have been applied to verification of finite state systems including 1-safe (finite) Petri nets and synchronous products of finite transition systems. Given that net unfoldings represent the state space in a distributed, implicit manner the verification algorithm is necessarily a *global* algorithm, where an unfolding is first generated followed by a reasoning step; this contrasts with traditional reachability-based approaches which can be *on-the-fly*. In his seminal work McMillan [McM93] showed that deadlock detection on unfoldings of 1-safe Petri nets is NP-complete. Since the deadlock problem on Petri nets is PSPACE-hard it is generally accepted that the two step process will yield savings (in time and space) provided the unfoldings are small.

In this paper we show how unfoldings can be extended to the context of infinite-state systems. More precisely, we show how unfoldings can be constructed to represent sets of backward reachable states of unbounded Petri nets in a symbolic fashion. Furthermore, based on unfoldings, we show how to solve the coverability problem for unbounded Petri

*Supported in part by ARO under grant ...

nets using a SAT-solver. Our experiments show that the use of unfoldings, in spite of the two-step process for solving coverability, has better time and space characteristics compared to a traditional on-the-fly, reachability based implementation that considers all interleavings for solving the coverability problem.

1 Introduction

Model Checking has had a great impact as an efficient method for algorithmic verification of finite-state systems. A limiting factor in its application is the *state space explosion* problem, which occurs since the number of states grows exponentially with the number of components inside the system. Therefore, much effort has been spent on developing techniques for reducing the effect of state space explosion in practical applications. One such technique is that of *partial orders* which is based on the observation that not all interleavings of a given set of independent actions need to be explored during model checking. Several criteria for independency has been given, e.g., *stubborn sets* [Val90], *persistent sets* [GW93] or *ample sets* [Pel93]. A method which has drawn considerable attention recently is that of *unfoldings* [McM95, ERV96, ER99]. Unfoldings are *occurrence nets*: unrollings of Petri nets that preserve their semantics. Although unfoldings are usually infinite, it is observed in [McM95] that we can always construct a finite initial prefix of the unfolding which captures its entire behavior, and which in many cases is much smaller than the state space of the system. Unfoldings have been applied to n-safe (i.e., finite-state) Petri nets, and more recently to other classes of finite-state systems such as synchronous products of finite transition systems [LB99, ER99]

In a parallel development, there has been numerous efforts, to extend the applicability of model checking to the domain of infinite-state systems. This has resulted in several highly nontrivial algorithms for verification of timed automata, lossy channel systems, (unbounded) Petri nets, broadcast protocols, relational automata, parametrized systems, etc. These methods operate on symbolic representations, called *constraints*, each of which may represent an infinite set of states. However, in a manner similar to finite-state verification, many of these algorithms suffer from a *constraint explosion* problem limiting their efficiency in practical applications. As the interest in the area of infinite-state systems increases, it will be important to design tools which limit the impact of constraint explosion. With this in mind, we have considered [AKP97, AJKP98] a refinement of the ample set construction and applied it to infinite-state systems such as Petri nets and lossy channel systems.

In this paper, we show how the unfolding technique can be made to work in the context of infinite state systems. More precisely, we present an unfolding algorithm for symbolic verification of unbounded Petri nets. We adapt an algorithm described in [AČJYK96] for backward reachability analysis which can be used to verify general classes of safety properties. Instead of working on individual markings (configurations) of the net (as is the case with the previous approaches [McM95, ERV96, ER99, LB99]) we let our unfolding algorithm operate on constraints each of which may represent an (infinite) upward closed set of markings. We start from a constraint describing a set of "final" markings, typically undesirable configurations which we do not wish to occur during the execution of the net. From the set of final markings we unroll the net backwards, generating a *Reverse Occurrence Net (RON).* In order to achieve termination we present an algorithm to compute a postfix of the RON, which gives a complete characterization of the set of markings from which we can reach a final marking. Using concepts from the theory of well quasi-orderings we show that the postfix is always finite. In fact, our method offers the same advantages over the algorithm in [AČJYK96], as those offered by the algorithms of [McM95, ERV96] in the context of finite-state systems.

Given that net unfoldings represent the state space in a distributed, implicit manner the verification process is a two step process: generation of the unfolding and reasoning about the unfolding. This contrasts with traditional approaches where the verification problem is done in a single step. In his seminal work McMillan [McM93] showed that deadlock detection on complete prefix of a 1-safe Petri net is NP-complete. Since the deadlock problem on petri nets is PSPACE-hard it is generally expected that the two step process will yield savings (in time and space) provided the unfoldings are small.

We show how to reason with unfoldings of unbounded Petri nets by reducing the problem of deciding whether a final marking is coverable from some initial marking to satisfiability of a propositional formula. We have implemented the two step process – that of the generating a postfix and the use of PROVER, a satisfiability checker based on the Stålmark Method [**?**], to verify safety properties for a number of examples. We offer an end-to-end comparison of the time and space required for the coverability problem. The comparison pits on one hand a combination of the unfolding construction and the use of PROVER for reasoning and on the other a backward reachability algorithm, which considers all interleavings. Our experiments lead us to conclude that the space and time required for reasoning based on unfoldings is significantly lesser than the space and time required for reasoning based on consideration of all interleavings.

Outline In the next section we give some preliminaries on Petri nets. In Section 3 we introduce Reverse Occurrence Nets (RONs). In Section 4 we describe the unfolding algorithm. In Section 5 we describe how to compute a finite postfix of the unfolding. In Section 6 we explain how we use the PROVER to solve the coverability problem. In Section **??** we discuss issues in implementing the abstract algorithms discussed in this paper. In Section 8 we report on some experimental results. Finally, in Section **??** we present some conclusions and discuss directions for future research.

2 Preliminaries

Let $\mathbb{N}$ be the set of natural of numbers. For $a, b \in \mathbb{N}$, we define $a \ominus b$ to be equal to $a - b$ if $a \geq b$, and equal to 0 otherwise. A *bag* over a set A is a mapping from A to $\mathbb{N}$. Relations and operations on bags such as $\leq$, $+$, $-$, $\ominus$, etc, are defined as usual. Sometimes, we write bags as tuples, so (a, b, a) represents a bag B with $B(a) = 2$, $B(b) = 1$, and $B(d) = 0$ if $d \neq a, b$. A set A of bags is said to be *upward closed* if $B_1 \in A$ and $B_1 \leq B_2$ then $B_2 \in A$.

The *upward closure* of a bag A is the set $\{A' \mid A \leq A'\}$. We may interpret a set S as a bag with $S(a) = 1$ if $a \in S$ and $S(a) = 0$ if $a \notin S$. We use $|S|$ to denote the size of the set S. In this paper, we shall use the terminology of [ERV96] as much as possible.

A *net* is a triple (P, T, F) where P is a finite set of *places*, T is a finite set of *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. By a *node* we mean a place or a transition. The preset ${}^\bullet x$ of a node x is the set $\{y \mid (y, x) \in F\}$. The postset $x^\bullet$ is similarly defined. A *marking* M is a bag over P. We say that a transition t is *enabled* in a marking M if ${}^\bullet t \leq M$. We define a transition relation on the set of markings, where $M_1 \longrightarrow M_2$ if there is $t \in T$ which is enabled in M_1 and $M_2 = M_1 - {}^\bullet t + t^\bullet$. We let $\stackrel{*}{\longrightarrow}$ denote the reflexive transitive closure of $\longrightarrow$. We say that a marking M_2 is *coverable* from a marking M_1 if $M_1 \stackrel{*}{\longrightarrow} M_2'$, for some $M_2' \geq M_2$. A *net system* is a tuple $N = (P, T, F, M_{init}, M_{fin})$, where (P, T, F) is a net and M_{init}, M_{fin} are markings, called the *initial* and the *final* marking of N respectively. In this paper, we consider the *coverability problem* defined as follows.

Instance: A net system $(P, T, F, M_{init}, M_{fin})$.

Question: Is M_{fin} coverable from M_{init}?

Note that using standard methods [VW86, GW93], we can reduce the problem of checking safety properties for Petri nets to the coverability problem.

To solve the coverability problem, we perform a backward reachability analysis. We define a *backward* transition relation [AČJYK96], such that, for markings M_1 and M_2 and a transition t, we have $M_2 \rightsquigarrow_t M_1$ if $M_1 = (M_2 \ominus t^\bullet) + {}^\bullet t$. We let $\rightsquigarrow = \cup_{t \in T} \rightsquigarrow_t$, and let $M \stackrel{k}{\rightsquigarrow} M'$ denote that $M = M_0 \rightsquigarrow M_1 \rightsquigarrow \cdots M_k = M'$, for markings $M_0, \ldots, M_k$. We define $\stackrel{*}{\rightsquigarrow}$ to be the reflexive transitive closure of $\rightsquigarrow$. Observe that, for each marking M_2 and transition t, there is a marking M_1 with $M_2 \rightsquigarrow_t M_1$, i.e., transitions are always enabled with respect to $\rightsquigarrow$. The following lemma relates the forward and backward transition relations.

Lemma 2.1

1. *If $M_1 \longrightarrow M_2$ and $M_2' \leq M_2$ then there is $M_1' \leq M_1$ such that $M_2' \rightsquigarrow M_1'$.*
2. *If $M_2 \rightsquigarrow M_1$ and $M_1' \geq M_1$ then there is M_2' such that $M_2' \geq M_2$ and $M_1' \longrightarrow M_2'$.*

3 Reverse Occurrence Nets

In this section we introduce *Reverse Occurrence Nets (RONs)*. A RON corresponds to "unrolling" a net *backwards*. Formally, a *RON R* is a net $(\mathsf{C}, \mathsf{E}, \mathsf{F})$ satisfying the following three conditions

(i) $|c^\bullet| \leq 1$ for each $c \in \mathsf{C}$.

(ii) there is no infinite sequence of the form $c_1 \mathsf{F} e_1 \mathsf{F} c_2 \mathsf{F} \cdots$. This condition implies that there are no cycles in the RON, and that there is a set $\max(\mathsf{F})$ of nodes which are maximal with respect to F.

(iii) $\max(\mathsf{F}) \subseteq \mathsf{C}$.

In a RON, the places and transitions are usually called *conditions* and *events* respectively. A set of events $E \subseteq \mathsf{E}$ is considered to be a *configuration* if $e \in E$ and $e\mathsf{F}^*e'$ then $e' \in E$.

Remark In [McM95, ERV96], a configuration E is upward closed in the sense that if an event e belongs to E then all events above e (with respect to F) also belong to E.

In our case, configurations are downward closed. Furthermore, in [McM95, ERV96], configurations are required to be *conflict free*, i.e., for all events $e_1, e_2 \in E$ we have ${}^\bullet e_1 \cap {}^\bullet e_2 = \emptyset$. Notice that this property is always satisfied by our configurations, since we demand that $|c^\bullet| \leq 1$ for each condition. ■

Consider a net system $N = (P, T, F, M_{init}, M_{fin})$ and a RON $(\mathsf{C}, \mathsf{E}, \mathsf{F})$, and let $\mu : \mathsf{C} \cup \mathsf{E} \to P \cup T$ such that $\mu(c) \in P$ if $c \in \mathsf{C}$ and $\mu(e) \in T$ if $e \in \mathsf{E}$. For $C \subseteq \mathsf{C}$, we define $\#C$ to be a marking such that, for each place p, the value of $\#C(p)$ is equal to the size of the set $\{c \in C \mid \mu(c) = p\}$. In other words $\#C(p)$ is the number of conditions in C labeled with p. We say that $(\mathsf{C}, \mathsf{E}, \mathsf{F}, \mu)$ is a *(backward) unfolding* of N if the following two conditions are satisfied:

(i) $\#\max(\mathsf{F}) = M_{fin}$, i.e., the set of conditions which are maximal with respect to F correspond to the final marking; and

(ii) μ preserves F, viz., if $(x, y) \in \mathsf{F}$ then $(\mu(x), \mu(y)) \in F$.

For a configuration E, we define $Cut(E) = (\{{}^\bullet e \mid e \in E\} \cup \max(\mathsf{F})) - \{e^\bullet \mid e \in E\}$. We define the marking $mark(E) = \#(Cut(E))$.

In Figure 1, we show a net system N with seven places, $p_1, \ldots, p_7$, and four transitions, $t_1, \ldots, t_4$. We also show an unfolding[1] U of N, assuming a final marking (p_1, p_7). Examples of configurations in U are $E_1 = \{e_2, e_4\}$ with $mark(E_1) = (p_1, p_2, p_3)$, and $E_2 = \{e_1, e_2, e_3, e_4\}$ with $mark(E_2) = (p_1, p_2, p_2, p_3)$.

4 An Unfolding Algorithm

We present an algorithm (Figure 2) which, for a given net system $N = (P, T, F, M_{init}, M_{fin})$, generates an unfolding of N in an incremental manner. In a manner similar to [ERV96], an unfolding $U = (\mathsf{C}, \mathsf{E}, \mathsf{F}, \mu)$ is represented as a list of objects corresponding to conditions and events in the underlying RON. An event e is represented as an object (C, t) where t is the label $\mu(e)$ of e and C is its set $e^\bullet$ of post-conditions. A condition c is represented by an object (e, p) where p is the label $\mu(c)$ of c, and e is its (single) post-event $c^\bullet$. We observe that the flow relation F and the labeling function μ are included in the encoding.

[1] To increase readability, we show both names and labels of events in the figure, while we omit name of conditions.

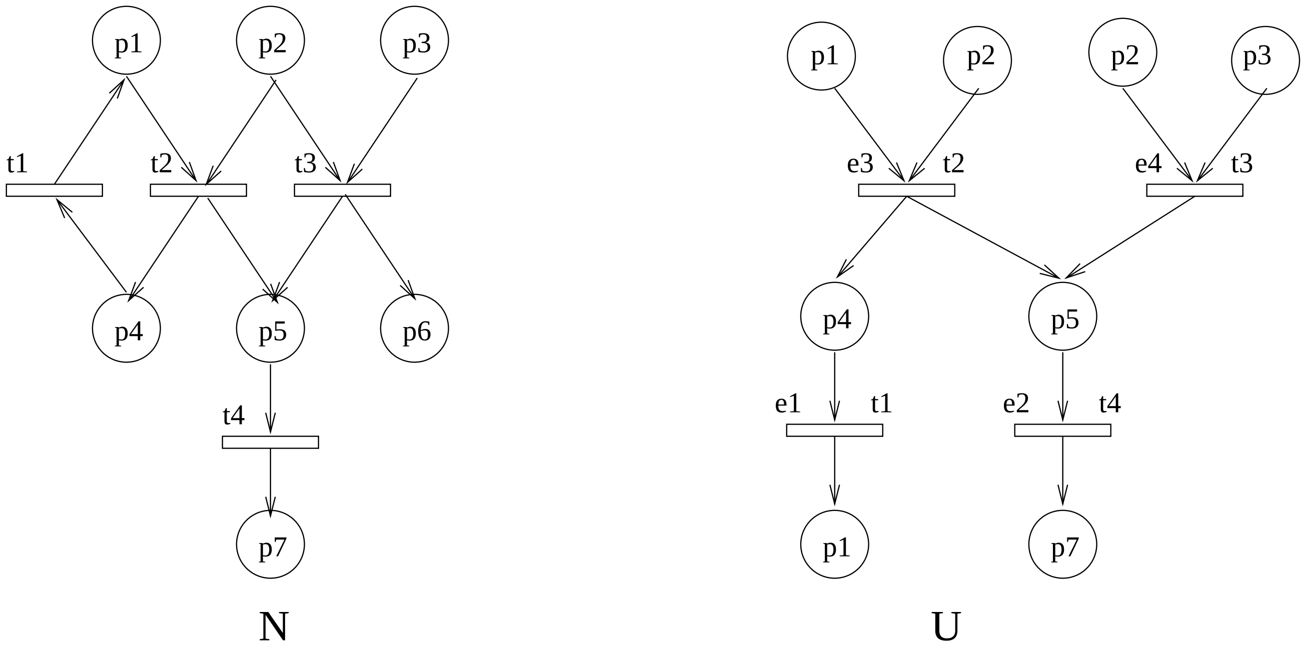


Figure 1: A net system and one of its unfoldings

Consider a set of conditions C of U to be t-enabled provided there exists a configuration E such that $C \subseteq Cut(E)$ and $0 < \#C \leq t^{\bullet}$, i.e., there is a configuration E such that $C \subseteq Cut(E)$ and all the conditions in C are in the postset of t. Furthermore, consider C to be maximally t-enabled provided there is no other set C' such that $C \subset C' \subseteq Cut(E)$ and C' is t-enabled. We will write $\mathsf{ME}_t(C)$ to denote that C is maximally t-enabled. We define $Xtnd(U)$ to be the set of events by which U can be extended and is formally defined as follows:

$$Xtnd(U) = \{(C,t) \mid \mathsf{ME}_t(C) \text{ and } (C,t) \notin U\}$$

Observe that the definition implies that there are no redundancies in the unfolding. In other words we will not have two different events both having the same label and the same postcondition.

The unfolding algorithm is shown in Figure 2. It maintains two variables, namely the current unfolding U (initialized to the final marking M_{fin}), and a set X of events by which the unfolding can be extended. The algorithm proceeds by considering the events in X in turn (this procedure is fair in the sense that each event added to X will eventually be considered). At each iteration an event in X is picked and moved to U. Furthermore, the possible extensions of the new unfolding are computed, using the function $Xtnd$, and added to X. Notice that the algorithm does not necessarily terminate.

The unfolding algorithm gives a symbolic representation of upward closed sets from which M_{fin} is coverable. More precisely (Theorem 4.1), the upward closure of the markings appearing in U gives exactly the set of markings from which M_{fin} is coverable. Notice that each event in the unfolding corresponds to a step in the backward unrolling of the net. The efficiency we gain through applying unfoldings on upward closed sets, as compared to the standard symbolic algorithm based on the backward transition relation $\rightsquigarrow$, can be explained in a manner similar to the finite state case [McM95, ERV96]; namely the addition of a set of concurrent events to the unfolding corresponds to an exponential number of applications of the $\rightsquigarrow$ relation.

Input: net system $N = (P, T, F, M_{init}, M_{fin})$, where $M_{fin} = (p_1, \ldots, p_n)$.
var U: unfolding of N; X: set of events.
begin
  $U := (p_1, \emptyset), \ldots, (p_m, \emptyset)$
  $X := Xtnd(U)$
  while $(X \neq \emptyset)$ **do**
    Pick and delete $e = (C, t)$ from X
    Add (C, t) to U and also add $\forall p \in {}^\bullet t$ a new condition (p, e) to U
    $X := X \cup Xtnd(U)$
end

Figure 2: Unfolding Algorithm

In the sequel we let U^i denote the value of the variable U after i iterations of the loop. The following lemmas relate unfoldings with the backward transition relation $\rightsquigarrow$.

Lemma 4.1 *If* $M_{fin} \overset{k}{\rightsquigarrow} M$ *then there is an* ℓ *and a configuration* E *in* U^ℓ *such that* $mark(E) \leq M$

Proof By induction on k.
Base case: Follows from the first step of the algorithm (initializing the value of U).

Induction case: Suppose that $M_{fin} \overset{k}{\rightsquigarrow} M' \rightsquigarrow_t M$. By the induction hypothesis there is an ℓ' and a configuration E' in $U^{\ell'}$ such that $mark(E') \leq M'$. Let C_1 be a maximal subset of $Cut(E')$ such that $\#C_1 \leq t^\bullet$.

There are two cases.

1. If C_1 is empty. We define $E = E'$. We show that $mark(E)(p) \leq M(p)$ for each place p. We have two subcases.
   (a) If $p \notin t^\bullet$, then $mark(E)(p) = mark(E')(p) \leq mark(E')(p) + {}^\bullet t(p) \leq M'(p) + {}^\bullet t(p) = M(p)$.
   (b) If $p \in t^\bullet$ then we know by maximality of C_1 that $mark(E')(p) = 0$, and hence $mark(E)(p) = mark(E')(p) \leq M(p)$.
2. If C_1 is not empty. Notice that $\mathsf{ME}_t(C_1)$ holds. Given that our algorithm is fair in selecting transitions that are backward firable, an event $e = (C_1, t)$ will be chosen and added at some point ℓ. We define $E = E' \cup \{e\}$. Clearly, E is a configuration in U^ℓ. Observe that $mark(E) = (mark(E') \ominus t^\bullet) + {}^\bullet t$. This means that $mark(E) = (mark(E') \ominus t^\bullet) + {}^\bullet t \leq (M'(p) \ominus t^\bullet) + {}^\bullet t = M$.

■

We now present the lemma in the other direction which shows that the marking associated with every configuration in an unfolding is backwards reachable.

Lemma 4.2 *For each ℓ and configuration E in U^ℓ, there is a marking M such that $M \leq mark(E)$ and $M_{\mathit{fin}} \overset{*}{\rightsquigarrow} M$.*

Proof By induction on ℓ.
Base case: Follows from the first step of the algorithm (initializing the value of U).

Induction case: Suppose that in step $\ell+1$ we add an event $e = (C, t)$ (together with its preset) to U^ℓ. Take any configuration E in $U^{\ell+1}$. If $e \notin E$ then we know that E is also a configuration in U^ℓ implying the result by induction hypothesis. Hence, we can assume that $E = E' \cup \{e\}$ where E' is a configuration in U^ℓ. By the induction hypothesis we know that there is a marking M' such that $M' \leq mark(E')$ and $M_{\mathit{fin}} \overset{*}{\rightsquigarrow} M'$. We define $M = (M' \ominus t^\bullet) + {}^\bullet t$. Clearly $M' \rightsquigarrow M$ and hence $M_{\mathit{fin}} \overset{*}{\rightsquigarrow} M$. Observe that $mark(E) = (mark(E') \ominus t^\bullet) + {}^\bullet t$. This means that $mark(E) = (mark(E') \ominus t^\bullet) + {}^\bullet t \geq (M'(p) \ominus t^\bullet) + {}^\bullet t = M$.
■

From Lemma 2.1, Lemma 4.1, and Lemma 4.2 we get the following theorem.

Theorem 4.1 *M_{fin} is coverable from a marking M if and only if there is an ℓ and a configuration E in U^ℓ such that $mark(E) \leq M$.*

Notice that as a special case we can take M in Theorem 4.1 to be equal to M_{init}.

5 Termination

In this section we show how to compute finite postfixes of unfoldings. We define special types of events which we call *cut-off points*. In Theorem 5.1 we show that cut-off points do not add any markings to the upward closed sets characterized by the unfolding. This means that in the unfolding algorithm (Figure 2) we can safely discard all cut-off points, without ever adding them to the unfolding U. Furthermore, we use concepts from the theory of well quasi-orderings (Theorem 5.1) to show that if all cut-off points are discarded then the variable X in the unfolding algorithm eventually becomes empty implying termination of the algorithm. We start with some definitions and auxiliary lemmas.

We assume a net system N and an unfolding U of N. For an event, we use $e{\downarrow}$ to denote the configuration $\{e' \mid e \mathsf{F}^* e'\}$. For configurations E_1 and E_2, we use $E_1 \prec E_2$ to denote that $|E_1| < |E_2|$ and $mark(E_1) \leq mark(E_2)$. For an event e, we say that e is a *cutoff point* in U if there is a configuration E in U such that $E \prec e{\downarrow}$.

We recall from the previous section that U^i denotes the value of the variable U after i iterations of the loop in the unfolding algorithm. In order to prove the cutoff theorem we need the following lemma:

Lemma 5.1 *Consider configurations E_1, E_2, and E_2' in U^k where $E_1 \prec E_2$ and $E_2 \subseteq E_2'$. There is an ℓ and a configuration E_1' in U^ℓ such that $E_1' \prec E_2'$.*

Proof We show the claim for a configuration $E_2' = E_2 \cup \{e_2\}$ where $e_2 \notin E_2$. The result follows using induction on $|E_2'| - |E_2|$. Let e_2 be of the form (C_2, t). We know that $mark(E_2') = (mark(E_2) \ominus t^\bullet) + {}^\bullet t$. We define C_1 to be a maximal subset of $Cut(E_1)$ such that $\#C_1 \leq t^\bullet$.

There are two cases:

1. If C_1 is empty we define $E_1' = E_1$. We have $|E_1'| = |E_1| < |E_2| < |E_2'|$. We show that $mark(E_1')(p) \leq mark(E_2')(p)$ for each place p. There are two subcases.
   (a) If $p \notin t^\bullet$. In this subcase we have $mark(E_2')(p) = mark(E_2)(p) + {}^\bullet t(p) \geq mark(E_1)(p) + {}^\bullet t(p) \geq mark(E_1)(p) = mark(E_1')(p)$.
   (b) If $p \in t^\bullet$. By maximality of C_1 we know that $mark(E_1)(p) = 0$. This means that $mark(E_2')(p) = (mark(E_2)(p) \ominus {}^\bullet t(p)) + {}^\bullet t(p) \geq {}^\bullet t(p) \geq mark(E_1)(p) = mark(E_1')(p)$.
2. If C_1 is not empty then, by fairness of the algorithm in selecting transitions that are backward firable, an event $e_1 = (C_1, t)$ will be chosen and added at some point ℓ. We define $E_1' = E_1 \cup \{e_1\}$. It is clear that E_1' is a configuration in U^ℓ. We have $|E_1'| = |E_1| + 1 < |E_2| + 1 \leq |E_2'|$. We know that $mark(E_1') = (mark(E_1) \ominus t^\bullet) + {}^\bullet t$. This means that $mark(E_2') = (mark(E_2) \ominus {}^\bullet t) + {}^\bullet t \geq (mark(E_1) \ominus {}^\bullet t) + {}^\bullet t = mark(E_1')(p)$.

■

Now we are ready to show in the following theorem that cutoff points can be discarded safely.

Theorem 5.1 *For each k and configuration E_2 in U^k, there is an ℓ and configuration E_1 in U^ℓ where $mark(E_1) \leq mark(E_2)$ and E_1 does not contain any cutoff points.*

Proof We use induction on $|E_2|$. The base case is trivial. If E_2 does not contain any cutoff points, then the proof is trivial. Otherwise let e_2 be a cutoff point in E_2. Clearly, $e_2{\downarrow} \subseteq E_2$. Since e_2 is a cut-off point, we know that there is a configuration E in U^k such that $E \prec e_2{\downarrow}$. By Lemma 5.1 there is an ℓ and a configuration E_1 in U^ℓ such that $E_1 \prec E_2$, i.e., $|E_1| < |E_2|$ and $mark(E_1) \leq mark(E_2)$. The claim follows by induction hypothesis. ■

To prove termination of the unfolding algorithm, we use the fact that markings are well quasi-ordered (consequence of Dickson's lemma [Dic13]), i.e., for any infinite sequence $M_0, M_1, \ldots$ of markings, there are i and j with $i < j$ and $M_i \leq M_j$.

Theorem 5.2 *The unfolding algorithm terminates if all cut-off points are discarded.*

Proof Suppose that the algorithm does not terminate. Since all nodes are finitely branching we have an infinite sequence $e_0, e_1, e_2, \ldots,$ of events where $e_{i+1}\mathsf{F}c_i\mathsf{F}e_i$, for some condition c_i. Notice that $|e_j{\downarrow}\,| > |e_i{\downarrow}\,|$, whenever $j > i$. By Dickson's lemma, it follows that there are i and j with $i < j$ and $\mathit{mark}(e_i{\downarrow}) \leq \mathit{mark}(e_j{\downarrow})$. This implies that e_j is a cut-off point, which is a contradiction. ∎

Remark Theorem 4.1, Theorem 5.1, and Theorem 5.2 give a complete terminating procedure for checking coverability in unbounded Petri nets: use the unfolding algorithm discarding all cutoff points. The final marking M_{fin} is coverable from the initial marking M_{init} iff a configuration E appears in the unfolding with $\mathit{mark}(E) \leq M_{\mathit{init}}$. ∎

6 Checking Coverability using SAT-solvers

The problem of checking coverability from a marking M once an unfolding U has been generated is, as stated above, the problem of finding a configuration E such that $\mathit{mark}(E) \leq M$. Since the number of configurations of U could be very large ($|\mathcal{P}(\mathsf{E})|$ in the worst case) a brute force method based on computing the set of all configurations is not practical. We, consequently, propose to use a SAT-solver, in our case PROVER, to carry out the task. This depends upon encoding the coverability problem as one of checking satisfiability of a propositional formula.

By the definition of configurations we know that for each configuration E the following holds:

(i) if an event $e \in E$ then for all events e' s.t. $e\mathsf{F}c\mathsf{F}e'$ we have $e' \in E$.

(ii) a condition $c \in \mathit{Cut}(E)$ if and only if $c^{\bullet} \subseteq E$ and for all events $e \in {}^{\bullet}c$, $e \notin E$.

Finally, a configuration E satisfies $\mathit{mark}(E) \leq M$, for a marking M, provided for each place p we have

$$\#\mathit{Cut}(E)(p) \leq M(p)$$

Given a net system $N = (P, T, F, M_{\mathit{init}}, M_{\mathit{fin}})$ and an unfolding $U = (\mathsf{C}, \mathsf{E}, \mathsf{F}, \mu)$ of N we can construct a formula $\mathcal{F}$, where each node x in U is represented by a variable v_x, according to the following

1. For each event $e \in \mathsf{E}$, add a conjunct $v_e \Rightarrow v_{e_1} \wedge \cdots \wedge v_{e_n}$ where $\{e_1, \ldots, e_n\}$ is the set $\{e_i \mid \exists c \in \mathsf{C} : e\mathsf{F}c\mathsf{F}e_i\}$

2. For each condition $c \in \mathsf{C}$, add a conjunct $v_c \Leftrightarrow v_{e_p} \wedge \neg v_{e_1} \wedge \cdots \wedge \neg v_{e_n}$ where $\{e_p\} = c^{\bullet}$ and $\{e_1, \ldots, e_n\}$ is the set $\{e_i \mid e_i\mathsf{F}c\}$

3. For each place $p \in P$, add a conjunct $LTEk(k, [v_{c_1}, \ldots, v_{c_n}])$ where $k = M_{init}(p)$, and $\{c_1, \ldots, c_n\}$ is the set $\{c_i \mid \mu(c_i) = p\}$. Furthermore, the predicate $LTEk(k, [v_1, \ldots, v_n])$ is true exactly when the number of propositions in the set $\{v_1, \ldots, v_n\}$ that are assigned true is lesser than or equal to k.

Now a model of $\mathcal{F}$ is an assignment of variables corresponding to a configuration E in the following way

- For each event e, $e \in E$ iff v_e
- For each condition c, $c \in Cut(E)$ iff v_c
- $mark(E) \leq M_{init}$

The problem of checking coverability has now been reduced to the satisfiability of the propositional formula $\mathcal{F}$, i.e., $\mathcal{F}$ is satisfiable if and only if M_{fin} is coverable from M_{init}.

The formula $\mathcal{F}$ is, of course, a simple propositional formula except for the third set of conjuncts, those containing $LTEk$. These formulas can be encoded in propositional logic; however, the naive translation would involve an exponential blow up. The SAT-solver used in this work, Prover, has support for such predicates, which comes to our aid.

6.1 Prover

To determine satisfiability we use a commercial tool, Prover, a proof procedure for propositional logic augmented with finite domain integer arithmetic and enumerated types. The theorem prover underlying Prover is an implementation of the Stålmarck Method [**?**] which is based on a system for natural deduction. The proof procedure has been know to be versatile and has been used with propositional formula containing as much as 350,000 connectives. Prover deals with formulas involving all the usual logical connectives, conjunction, disjunction, implication, equivalence and negation. In addition, it also provides a number of predicates of the form LTEk$(k, [f_1, \ldots, f_n])$ which are statements about how many of the n formulas $f_1, \ldots, f_n$ are true, in this case the number of true formulas is less than or equal to k. These predicates can be defined using the basic connectives, but the naive way to do so results in a number of connectives which grows as a k-order polynomial in n, whereas the calls in Prover use approximatively $2 * k * (n - k)$ connectives. Finally, Prover also deals with the arithmetic connectives, addition, subtraction, multiplication, division, remainder and negation.

7 Implementation

In our experimentation, we have compared two implementations of the unfolding algorithm and an implementation of a backwards reachability algorithm.

```
Input: net system N = (P, T, F, M_init, M_fin).
var Min : Set of minimal markings, Q : Queue of markings to be considered
begin
    Min := ∅;
    Q := {M_fin};
    while (Q ≠ ∅) do
        Pick and delete a marking M from Q
        If M ≤ M_init then return "yes"
        If ∃M' ∈ Min such that M' ≤ M then continue;
        Add M to Min while removing any M' ∈ Min such that M ≤ M'
        Add to Q all M' such that M ⇝ M'.
    end
    Return "No";
end
```

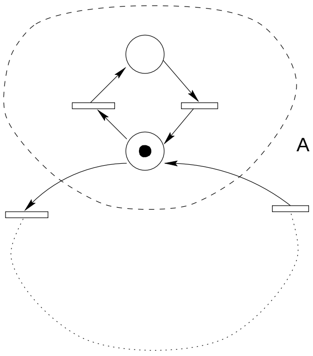
Figure 3: Backward Reachability Algorithm

The implementation of the backwards reachability algorithm is a straightforward rendition of the abstract algorithm in [AČJYK96] and is as given in Figure 3. Note that this algorithm does not make use of partial-order techniques, and, therefore, considers all possible interleavings.

A technical point to note in the algorithm above is that the set Min, at the end of the k^{th} iteration, maintains the minimal elements of the set $\{M | M_{fin} \rightsquigarrow^k M\}$. Given that these minimal elements denote upward closed sets of markings, we are, again, guaranteed termination by Dickson's Lemma [Dic13]. More importantly, we wish to point out that if a marking that is smaller than M_{init} is generated then the algorithm terminates immediately without generating the entire set of minimal elements of the backward reachable set. This contrasts with our unfolding algorithm where we have to build the whole prefix before checking whether M_{init} is represented in the unfolding.

Issues in the unfolding algorithm: The implementation of the unfolding algorithm is a straight-forward rendition of the abstract algorithm given in Section 3. There were, however, two issues that need explanation; computation of *Xtnd* and checking of termination. In the computation of *Xtnd* we maintain a queue of sets of conditions, where each set denotes a set of conditions that could hold concurrently and can, consequently, be the postset of an event. As new conditions are generated we check whether a new condition can be added to a set of conditions that is already under consideration. It is in this way that a seemingly combinatorial problem is converted to one of carrying out depth-first searches.

The two implementations that we will report upon, Unfolding 1 and Unfolding 2, use the same abstract program and the same strategy for calculating *Xtnd*. They, however, differ, on how the termination conditions are checked. In Unfolding 1 when a new event e is generated we compute $mark(e'{\downarrow})$ and $|e'{\downarrow}|$ for all events e' in the current unfolding. Clearly, there is a



# processes	$\|P\|$	$\|T\|$	Unfolding 1		Unfolding 2		Backward	
			Time $\times 10^{-2}$ sec	Space	Time $\times 10^{-2}$ sec	Space	Time $\times 10^{-2}$ sec	Space
2	4	6	3	14	1	20	1	37
4	8	12	14	30	7	44	10	147
8	16	24	79	62	26	92	284	550
16	32	48	596	126	134	188	12124	2006

Note: The number representing space is the number of markings that need to be stored.

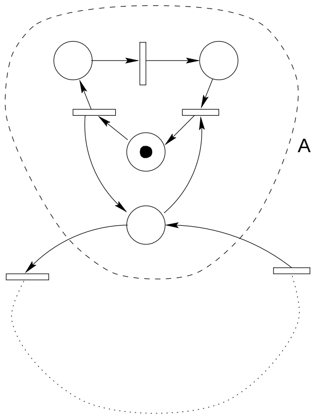
Figure 4: A Simple Token Ring Network

lot of wasted time with this design choice but it does save on space. With Unfolding 2 with each event e' we maintain both $mark(e\downarrow)$ and $|e'\downarrow|$.

8 Experimental Results

The thesis of our experimental work is that although analysis with unfoldings is is done in two steps, constructing the unfolding and reasoning about it, the time and space required is less than for an interleaving based backward analysis for the coverability problem of unbounded Petri nets.

We will now show three examples which illustrate convincingly that unfoldings provide significant savings in space and time. In each case, we give the size of the Petri net in terms of the number of places, $|P|$, and the number of transitions, $|T|$. We compute (a) the maximum number of markings that need to be maintained for the traditional backward analysis [AJ01] and (b) the total number of nodes generated by the unfolding algorithm. Given that the storage requirements of a node is bounded by the storage required for a marking, comparing the number of markings from backwards analysis against the total number of nodes in an unfolding is appropriate. Furthermore, we also report on the time taken for using both the prover and the generation of the unfolding.



# tokens & processes	$\|P\|$	$\|T\|$	Unfolding 1		Unfolding 2		Backward	
			Time $\times 10^{-2}$ sec	Space	Time $\times 10^{-2}$ sec	Space	Time $\times 10^{-2}$ sec	Space
2	8	8	18	30	7	42	20	197
4	16	16	56	54	18	78	3053	1700
8	32	32	313	102	70	150	553324	14637
16	64	64	2611	198	409	294	*	*

Note: * implies non-termination after a reasonable amount of time.

Figure 5: A more complicated token ring network

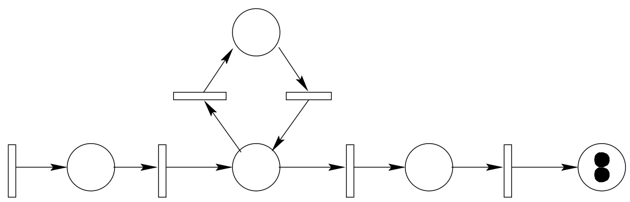
Example 1: The first example we consider, presented in Figure 4, is that of a simple token ring consisting of a number of processes. As we increase the number of processes the number of places and transitions increase too. Note, however, that M_{init} and M_{fin} were so chosen that they have the same total number of tokens in each case, and, furthermore, that M_{fin} is not coverable from M_{init}. Consequently, the backwards analysis algorithm would have to compute the basis for the entire backward reachability set.

Example 2: This is a slightly more complicated version of Example 1 designed with the aim of introducing more branching (see Figure 5). In this case, though the number of processes was changed (and thus the size of the petri net changed) the M_{fin} was kept the same. However, the initial marking was changed with every instance. Finally, in all of these cases too M_{fin} was not coverable from the M_{init}.

Example 3: In this example, of a buffer process, from Figure 6, the original Petri net was kept the same while the number of tokens in M_{fin} was changed with each instance while the M_{init} was kept the same. Furthermore, in all of the instances M_{fin} is coverable from M_{init}.

There are several conclusions that can be drawn from these experiments:

- Unfolding 2 is a better algorithm than Unfolding 1. While it uses approximately 1.5



# tokens	Unfolding 1		Unfolding 2		Backward	
	Time $\times 10^{-2}$ sec	Space	Time $\times 10^{-2}$ sec	Space	Time $\times 10^{-2}$ sec	Space
2	4	20	3	30	2	36
4	18	40	9	60	14	130
8	105	80	26	120	526	723
16	708	160	95	240	78733	6068

Figure 6: A buffer process

times the memory that Unfolding 1 uses, the savings in time is indeed significant. The time falls by much more than a factor of 1.5.

- Both Unfolding 1 and Unfolding 2 are better than a traditional backward analysis, which uses interleaving semantics.
- The cost of using PROVER to reason about the unfolding is insignificant compared to the time, and space, needed to compute the unfolding.
- Unfoldings do offer great savings in time and space.

References

[AČJYK96] Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. LICS' 96* 11^{th} *IEEE Int. Symp. on Logic in Computer Science*, pages 313–321, 1996.

[AJ01] Parosh Aziz Abdulla and Bengt Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems. *Theoretical Computer Science*, 256:145–167, 2001.

[AJKP98] Parosh Aziz Abdulla, Bengt Jonsson, Mats Kindahl, and Doron Peled. A general approach to partial order reductions in symbolic verification. In *Proc.* 10^{th} *Int. Conf. on Computer Aided Verification*, volume 1427 of *Lecture Notes in Computer Science*, pages 379–390, 1998.

[AKP97] Parosh Aziz Abdulla, Mats Kindahl, and Doron Peled. An improved search strategy for Lossy Channel Systems. In Tadanori Mizuno, Nori Shiratori, Teruo Hegashino, and Atsushi Togashi, editors, *FORTE X / PSTV XVII '97*, pages 251–264. Chapman and Hall, 1997.

[Dic13] L. E. Dickson. Finiteness of the odd perfect and primitive abundant numbers with n distinct prime factors. *Amer. J. Math.*, 35:413–422, 1913.

[ER99] J. Esparza and S. Römer. An unfolding algorithm for synchronous products of transition systems. In *Proc. CONCUR '99,* 10^{th} *Int. Conf. on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 2–20. Springer Verlag, 1999.

[ERV96] J. Esparza, S. Römer, and W. Vogler. An improvement of McMillan's unfolding algorithm. In *Proc. TACAS '96,* 2^{th} *Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, volume 1055 of *Lecture Notes in Computer Science*, pages 87–106. Springer Verlag, 1996.

[GW93] P. Godefroid and P. Wolper. Using partial orders for the efficient verification of deadlock freedom and safety properties. *Formal Methods in System Design*, 2(2):149–164, 1993.

[LB99] R. Langerak and E. Brinksma. A complete finite prefix for process algebra. In *Proc.* 11^{th} *Int. Conf. on Computer Aided Verification*, volume 1633 of *Lecture Notes in Computer Science*, pages 184–195. Springer Verlag, 1999.

[McM93] K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[McM95] K.L. McMillan. A technique of a state space search based on unfolding. *Formal Methods in System Design*, 6(1):45–65, 1995.

[Pel93] D. Peled. All from one, one for all, on model-checking using representatives. In *Proc.* 5^{th} *Int. Conf. on Computer Aided Verification*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer-Verlag, 1993.

[Val90] A. Valmari. Stubborn sets for reduced state space generation. In *Advances in Petri Nets*, volume 483 of *Lecture Notes in Computer Science*, pages 491–515. Springer-Verlag, 1990.

[VW86] M. Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proc. LICS '86,* 1^{st} *IEEE Int. Symp. on Logic in Computer Science*, pages 332–344, June 1986.