

SAT-solving the Coverability Problem for Unbounded Petri Nets

Parosh Aziz Abdulla
Dept of Computer Systems
Uppsala University
Uppsala, Sweden
parosh@docs.uu.se

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

Aletta Nylén
Dept of Computer Systems
Uppsala University
Uppsala, Sweden
aletta@docs.uu.se

Abstract

Verifying systems using net unfoldings to represent the state space is a process containing two steps: generation of the unfolding and reasoning about the unfolding. In a recent paper [AIN00] we generalized the notion of unfoldings to unbounded Petri nets and showed how to capture a symbolic representation of the state space of these nets, thus completing the first of the two steps discussed above. We now continue with our experimentation and show how to reason with the unfolding obtained. In particular, we show how to use a SAT-solver to solve the coverability problem for unbounded Petri nets. The results of our experiments show that the use of unfoldings, in spite of the two-step process, has better time and space characteristics than an implementation that does not use unfoldings. In effect, we provide the first evidence for the conjecture that the two step process based on unfoldings could be better than a single step verification process that considers all interleavings.

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 in the system. Therefore, much effort has been spent on developing techniques for reducing the effect of state space explosion in practical applications. One such a 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].

There has also 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,

*Supported in part by US ARO under grant P-38682-MA and by STINT

lossy channel systems, (unbounded) Petri nets, broadcast protocols, relational automata, parameterized 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 showed how the unfolding technique can be made to work in the context of infinite state systems [AIN00] by adapting an algorithm described in [AČJYK96] for backward reachability analysis which can be used to verify general classes of safety properties. More precisely, we presented an unfolding algorithm for symbolic verification of unbounded Petri nets. Where previous approaches [McM95, ERV96, ER99, LB99] worked on individual markings of the net, we instead let our unfolding algorithm operate on constraints representing (potentially infinite) upward closed sets of markings. We start from a constraint describing a set of “final” markings, usually representing configurations that are undesirable during the execution of the net. From the set of final markings we unroll the net backwards, generating a *Reverse Occurrence Net* (*RON*). The algorithm computes a finite postfix of the RON, which gives a complete characterization of the set of markings from which a final marking is coverable.

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 conjectured that the two step process will yield savings (in time and space) provided the unfoldings are small.

We now 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. Based on the postfix algorithm, we have implemented a prototype, which we have used together with PROVER, a satisfiability checker based on the Stålmark Method [SS98], to verify safety properties for a number of examples. The main contribution of this work is 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 main conclusion is 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) and unfoldings. In Section 4 we describe how the coverability problem can be reduced to a satisfiability problem which can be solved using the SAT-solver PROVER which is described in Section 5. In Section 6 we describe the implementations used in our experimentation and in Section 7 the results are reported. Finally, in Section 8 we give some conclusions and 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. We use $|S|$ to denote the size of the set S .

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, \longrightarrow , on the set of markings where $M_1 \longrightarrow M_2$ if there is a $t \in T$ which is enabled in M_1 and $M_2 = M_1 - \bullet t + t^\bullet$. We let $\xrightarrow{*}$ denote the reflexive transitive closure of \longrightarrow . We say that a marking M_2 is *coverable* from a marking M_1 if

$M_1 \xrightarrow{*} M'_2$, 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} ?

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 $M_1 = (M_2 \ominus t^\bullet) + \bullet t$. 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 result justifies the use of backward reachability:

Lemma 2.1 [AČJYK96]

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 and Unfoldings

A *Reverse Occurrence Net (RON)* corresponds to a backwards “unrolling” of a net. Formally, a *RON* R is a net (C, E, F) satisfying the following conditions

- (i) $|c^\bullet| \leq 1$ for each $c \in C$.
- (ii) there is no infinite sequence of the form $c_1 F e_1 F c_2 F \dots$. This condition implies that there are no cycles in the RON, and that there is a set $\max(F)$ of nodes which are maximal with respect to F .
- (iii) $\max(F) \subseteq C$.

In a RON, the places and transitions are usually called *conditions* and *events* respectively. A set of events $E \subseteq E$ is considered to be a *configuration* if for every event e' , if there is an event $e \in E$ with eF^*e' then $e' \in E$. Intuitively, a configuration captures a set of events that could have been fired.

Consider a net system $N = (P, T, F, M_{init}, M_{fin})$ and a RON (C, E, F) , and let $\mu : C \cup E \rightarrow P \cup T$ such that $\mu(c) \in P$ if $c \in C$ and $\mu(e) \in T$ if $e \in E$. For $C \subseteq C$, 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 (C, E, F, μ) is a (*backward*) *unfolding* of N if the following two conditions are satisfied:

- (i) $\#\max(F) = M_{fin}$, 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 $(x, y) \in F$ then $(\mu(x), \mu(y)) \in F$.

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

In Figure 1, we show a net system N with seven places, p_1, \dots, p_7 , and four transitions, t_1, \dots, t_4 . We also show an unfolding¹ 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 $mark(E_2) = (p_1, p_2, p_2, p_3)$.

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

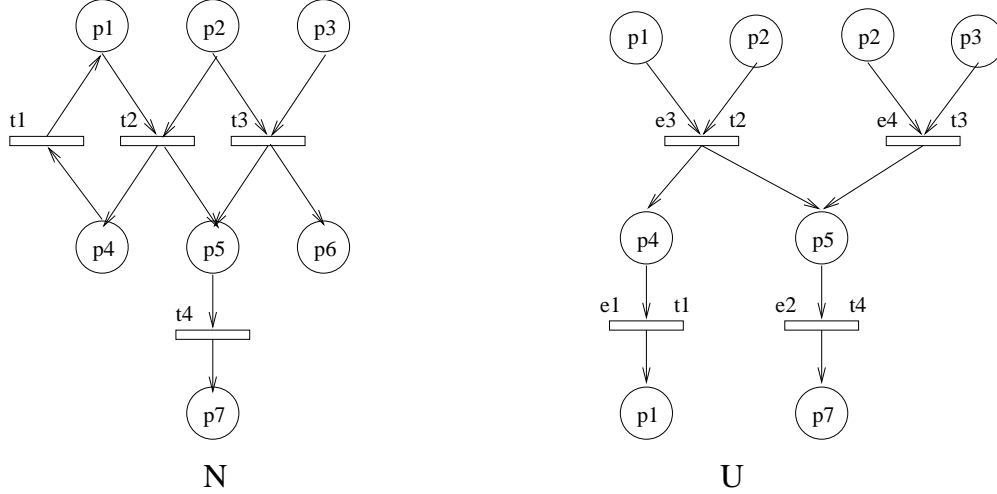


Figure 1: A net system and one of its unfoldings

Remark 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.

3.1 Postfix of an unfolding

Our definition of unfolding does not preclude the fact that it could be an infinite (backward) unrolling. In practice, we wish to construct a finite postfix of the unfolding and reason with it. The construction of the finite postfix critically depends upon certain events that are called *cut-off* events. Intuitively, by backward firing cut-off events we would only be generating states that are already represented in some other part of the net.

For every event e we define its downward closure $e\downarrow$ to be $\{e' \mid e F^* e'\}$. Clearly $e\downarrow$ is a configuration and has a marking associated with it. We say that a configuration E_1 covers a configuration E_2 , written formally as $E_1 \prec E_2$, iff $|E_1| < |E_2|$ and $mark(E_1) \leq mark(E_2)$. From Lemma 2.1 it is easy to see that markings backward reachable from E_2 are also reachable from E_1 . Consequently, it does not make any sense to explore states reachable from E_2 . The importance of the size restriction comes from the fact that there could be multiplicity of tokens in a place (concurrently) and one can not be ignored for the sake of others.

An event e is a cut-off event in U if there is a configuration E in U such that $E \prec e\downarrow$.

In Figure 2, an algorithm which generates an unfolding $U = (C, E, F, \mu)$ of a given net system $N = (P, T, F, M_{init}, M_{fin})$ in an incremental way is presented. In a manner similar to [ERV96], the unfolding 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.

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 . We will write $E_t(C)$ to denote that C is t -enabled. We define $Xtnd(U)$ to be the set of events by which U can be extended, formally defined as follows:

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

Observe that the definition implies that there are no redundancies in the unfolding. In other words we will

```

Input: net system  $N = (P, T, F, M_{init}, M_{fin})$ , where  $M_{fin} = (p_1, \dots, p_n)$ .
var  $U_N$ : unfolding of  $N$ ;  $X$ : set of events.
begin
   $U_N := (p_1, \emptyset), \dots, (p_m, \emptyset)$ 
   $X := X_{tn}d(U_N)$ 
  while  $(X \neq \emptyset)$  do
    Pick and delete  $e = (C, t)$  from  $X$ 
    Add  $e$  to  $U$ 
    If  $\neg cut\text{-}off(e)$  then  $\forall p \in \bullet t$  add a new condition  $(p, e)$  to  $U_N$ 
     $X := X \cup X_{tn}d(U_N)$ 
end

```

Figure 2: Unfolding Algorithm

not have two different events both having the same label and the same postcondition.

If the algorithm, given above, does not terminate then there would be an infinite sequence of events $\{e_i\}_{i \geq 1}$ such that for every pair of distinct events e_i and e_j we would have $mark(e_i \downarrow)$ and $mark(e_j \downarrow)$ are incomparable, which is not possible according to Dickson's Lemma [Dic13]. Consequently, we are assured that the unfolding construction does terminate leaving behind a structure to reason with. Note that M_{fin} is coverable from a marking M_{init} if and only if there is a configuration E in U such that $mark(E) \leq M_{init}$.

4 Checking Coverability using SAT-solvers

The problem of checking coverability from a marking M once an unfolding U has been generated is, as stated in Section 3, the problem of finding a configuration E such that $mark(E) \leq M$. Since the number of configurations of U could be very large ($|\mathcal{P}(E)|$ 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. $eFcFe'$ we have $e' \in E$.
- (ii) a condition $c \in Cut(E)$ if and only if $c^\bullet \subseteq E$ and for all events $e \in \bullet c$, $e \notin E$.

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

$$\# Cut(E)(p) \leq M(p)$$

Given a net system $N = (P, T, F, M_{init}, M_{fin})$ and an unfolding $U = (C, E, F, \mu)$ 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 $e \in E$, add a conjunct $v_e \Rightarrow v_{e_1} \wedge \dots \wedge v_{e_n}$ where $\{e_1, \dots, e_n\}$ is the set $\{e_i \mid \exists c \in C : eFcFe_i\}$
2. For each condition $c \in C$, add a conjunct $v_c \Leftrightarrow v_{e_p} \wedge \neg v_{e_1} \wedge \dots \wedge \neg v_{e_n}$ where $\{e_p\} = c^\bullet$ and $\{e_1, \dots, e_n\}$ is the set $\{e_i \mid e_iFc\}$
3. For each place $p \in P$, add a conjunct $LTEk(k, [v_{c_1}, \dots, v_{c_n}])$ where $k = M_{init}(p)$, and $\{c_1, \dots, c_n\}$ is the set $\{c_i \mid \mu(c_i) = p\}$. Furthermore, the predicate $LTEk(k, [v_1, \dots, v_n])$ is true exactly when the number of propositions in the set $\{v_1, \dots, v_n\}$ 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 , $c \in \text{Cut}(E)$ iff v_c
- $\text{mark}(E) \leq M_{init}$

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.

5 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 [SS98] which is based on a system for natural deduction. The proof procedure has been known 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 $\text{LTEk}(k, [f_1, \dots, f_n])$ which are statements about how many of the n formulas f_1, \dots, f_n that 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.

PROVER also deals with the arithmetic connectives, addition, subtraction, multiplication, division, remainder and negation.

6 Implementation

In our experimentation, we have compared two implementations of the unfolding algorithm and an implementation of a backwards 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 $\{M \mid M_{fin} \rightsquigarrow^k M\}$. 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;

```

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
   $Min := \emptyset$ ;
   $Q := \{M_{fin}\}$ ;
  while ( $Q \neq \emptyset$ ) do
    Pick and delete a marking  $M$  from  $Q$ 
    If  $M \leq M_{init}$  then return “yes”
    If  $\exists M' \in Min$  such that  $M' \leq M$  then continue;
    Add  $M$  to  $Min$  while removing any  $M' \in Min$  such that  $M \leq M'$ 
    Add to  $Q$  all  $M'$  such that  $M \rightsquigarrow M'$ .
  end
  Return “No”;
end

```

Figure 3: Backward Reachability Algorithm

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 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 $|e'\downarrow|$.

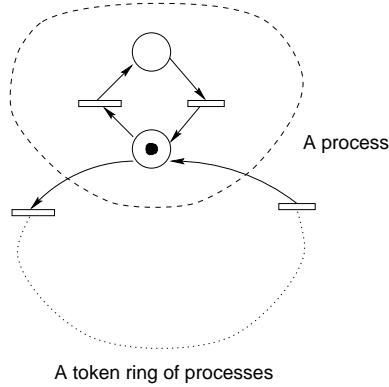
7 Experimental Results

The thesis of our experimental work is that although analysis with unfoldings 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, in spite of being small², illustrates 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 [AJ98] 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.

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 the M_{init} and M_{fin} where so chosen that they have the same total number of tokens in each case, and, furthermore, the 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.

²Unfortunately, no large examples of unbounded petri nets are available. We checked the Petri net list's repository without much luck. Any suggestions are welcome.



# 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

Figure 4: A Simple Token Ring Network

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_{fn} was kept the same. However, the initial marking was changed with every instance. Finally, in all of these cases too M_{fn} was not coverable from the M_{init} .

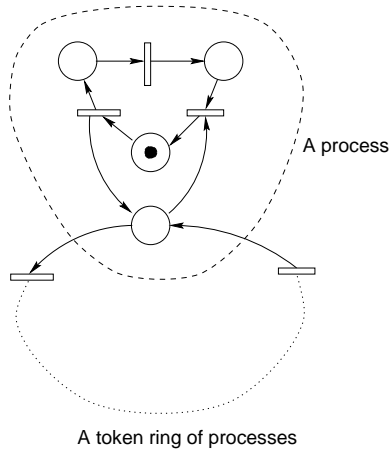
Example 3: In this example, of a buffer process, from Figure 6, the original Petri net was kept the same while the number of tokens in M_{fn} was changed with each instance while the M_{init} was kept the same. Furthermore, in all of the instances M_{fn} is coverable from M_{init} .

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 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.

8 Conclusions and Future Work

We have shown how safety properties can be verified for infinite state systems modeled by unbounded Petri nets by using a combination of the unfolding technique presented in [AIN00] and a SAT-solver,



# 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

PROVER [SS98]. We have compared this two-step process with an implementation of a single step verification that does not use unfoldings and found that it is more efficient both concerning time and space.

Since the cost of using PROVER to reason about unfoldings is insignificant compared to the cost of creating the unfolding an important direction of future research is the design of efficient data structures for implementation of the unfolding algorithm. It is also important to study the performance of the algorithm on more advanced examples.

References

- [AČJYK96] Parosh Aziz Abdulla, Karlis Čerāns, Bengt Jonsson, and Tsay Yih-Kuen. General decidability theorems for infinite-state systems. In *Proc. 11th IEEE Int. Symp. on Logic in Computer Science*, pages 313–321, 1996.
- [AIN00] Parosh Aziz Abdulla, Purushothaman Iyer, and Aletta Nylén. Unfoldings of unbounded petri nets. To appear in *Proc. CAV’2000, 12th Int. Conf. on Computer Aided Verification*, 2000.
- [AJ98] Parosh Aziz Abdulla and Bengt Jonsson. Ensuring completeness of symbolic verification methods for infinite-state systems, 1998. To appear in the journal of *Theoretical Computer Science*.
- [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, 9th Int. Conf. on Concurrency Theory*, volume 1664 of *Lecture Notes in Computer Science*, pages 2–20. Springer Verlag, 1999.

