Formal Methods in System Design, 6, 45–65 (1995)

A Technique of State Space Search Based on Unfolding

K.L. MCMILLAN MCMILLAN@RESEARCH.ATT.COM
School of Computer Science, Carnegie Mellon University

Received May 1, 1991

Editor: D.K. Probst

Abstract. Unfoldings of Petri nets provide a method of searching the state space of concurrent systems without considering all possible interleavings of concurrent events. A procedure is given for constructing the unfolding of a Petri net, terminating the construction when it is sufficient to represent all reachable markings. This procedure is applied to hazard and deadlock detection in asynchronous circuits. Examples are given of scalable systems with exponential size state spaces, but polynomial size unfoldings, including a distributed mutual exclusion ring circuit.

Keywords: verification, Petri nets, unfolding, hazard detection, deadlock detection, mutual exclusion, partial orders, state explosion problem

1. Introduction

Consider the following simple game: the playing board consists of a single row of spaces; the game begins with a number of playing pieces on space X, and the object is to move all of the pieces to space Y, by moving each piece one space at a time. This is hardly a game at all, since the playing pieces can be moved from X to Y in any order, without affecting the outcome. However, consider writing a computer program to play this game. The standard approach would be to use a depth first search—a program that enumerates all possible combinations with one move, then two moves, then three, etc., until a solution is found. This approach is hopeless, since the number of combinations of this simple game is exponential in the number of playing pieces. One might imagine improving the depth first strategy by keeping a list of all visited board configurations to avoid visiting them again, but this stratagem is also hopeless.

In fact, our simple game is quite analogous to the problem of analyzing the behavior of concurrent finite state systems, as might arise in distributed protocols, parallel computations, or asynchronous digital circuits. Such systems are generally made of a number of relatively independent processes (i.e., the playing pieces) that synchronize or communicate only occasionally (such as in the final state of our game). Any attempt to analyze such systems by exhaustively enumerating the behaviors or reachable states must be limited to very small systems, because of the intractable combinatorics. However, in our simple game, ignoring the (irrelevant) ordering of moves leads to a much simpler (in fact trivial) analysis. Might such be the case with concurrent systems?

This question has been approached before [14], [15], [5], [6], [16]. Valmari and Godefroid have (independently) observed that the notion of *persistence* is an important one in determining when move ordering can be ignored. A persistent move is one which remains possible (is not blocked) regardless of any other moves which might be taken in the future. In our simple game, all moves are persistent, since we allow any number of playing pieces on a given space. Now, suppose that at a given point in our search, a move cannot be blocked, and cannot block any other move we might want to make in the future. Then we may as well make that move now as later, and we have no need to consider other branches of the search tree—all lead to the same possible outcomes.[1] The notion of persistence can be generalized to persistent sets of moves, in the sense that one move of the set must eventually be taken. In our simple game, since no move can result in the blocking of any other move, a search in the style Valmari or Godefroid can lead directly to the solution, without any branching in the search.

So far, so good, but life is not always so simple. In some cases, a straightforward analysis of the rules of play may determine that certain moves are persistent. In other cases, this information may be more difficult to obtain. For example, let us modify our simple game so that no space between X and Y may contain more than three pieces (for example, each piece might represent a process, which at each step requires a resource, of which only three are available). It is still true that the game is won regardless of the order in which legal moves are made. Further, all moves are persistent (and non-blocking) in the weak sense that if we keep playing the game, *eventually* any given piece will be able to move. Unfortunately, obtaining the latter information seems to be as difficult as the analysis of the game itself. In fact, this property is not an artificial one. In this paper, we will be chiefly concerned with analyzing the behavior of asynchronous digital circuits. In our models of these circuits, the moves correspond to signal transitions; the nature of these rules is such that no transition or small set of transitions is inherently persistent—we must refer to the global behavior of the circuit to make this determination. Since this appears to be difficult, let us take a step back, and consider what other forms of analysis, besides a search tree, might be suitable to the problem.

At each step in constructing a search tree, we are forced to choose a move to take *next*. In other words, we are constructing *totally ordered* sequences of moves. If this choice may result in an increase in information regarding the outcome (i.e., reduce the set of possible outcomes), we are forced to create a bifurcation in the tree, since we wish to consider all possible outcomes. This in turn leads to an increase in the number of combinations or states that we must search. We should note, however, that this choice of a total ordering is mandated by the structure of the search tree, and not by the rules of the game itself. These latter may allow us to choose a partial rather than a total order on moves, and thereby avoid unnecessary bifurcation in the search. Such an analysis requires a richer structure than a search tree—one that represents both a partial order on moves or transitions, and choice, or bifurcation in the search. One such structure, called a *pomtree*, has been invented by Probst and Li [10], [11], [12], and has been used in the verification of asynchronous circuits.[2]

Another framework for a partially ordered analysis has been provided by Nielsen, Plotkin and Winskel [9]. They describe the unfolding of a model called a Petri net into an infinite net of a restricted form called an occurrence net. From the occurrence net, one can derive a partial order (one might say a causal order) on events, and a conflict relation between events which corresponds to bifurcation or choice. Taken together, these two relations form a prime event structure. Nielsen, Plotkin and Winskel describe the relation between event structures and domains of information as defined by Scott. Here, however, we are concerned with more mundane matters. Petri nets, while somewhat restricted in their expressiveness, can be used to model the behavior of asynchronous circuits in such a way as to capture the causal relation between signal transitions. We will discuss an algorithm that unfolds a sufficient fragment of the corresponding occurrence net to reveal important properties of the circuit, such as potential hazards and deadlocks, if such exist. We will also see that the unfolding approach can in some cases avoid the exponential explosion of states that results from the total ordering of transitions in standard search algorithms.

2. Unfolding

Let us begin with an informal introduction to Petri nets and unfolding. A Petri net is a game board of sorts. The spaces on the board are drawn as circles and called *places*. These may hold any number of game pieces, which are drawn as dots and called *tokens*. The rules for moving the tokens are given by a set of *transitions*, which are drawn as lines or boxes. An arrow from a place to a transition indicates that the transition removes a token from the place, while an arrow in the opposite direction indicates that the transition places a new token on that place. Removal occurs before placement, so a transition cannot be used until and unless there is one token at the tail of each arrow entering the transition. Later, we will see how this token game can be used to model an asynchronous circuit, using two places to model each wire. A token on one represents a logical zero on the wire, while a token on the other represents a logical one, and transitions moving tokens from one to the other correspond to rising and falling transitions on the wire.

Unfolding a Petri net is a game that can easily be played on a piece of paper, though it quickly becomes tedious. First, place some initial tokens on your Petri net. For each of these tokens, make a copy of the place on which it resides in the occurrence net (label the copies appropriately). Now, repeat the following process *ad infinitum*:

1. Choose a transition from your Petri net and call it t.

2. The set of places in your Petri net which have arrows going to t is called its *preset*. For each place in the preset of t, find a copy in the occurrence net and mark it with a token (if you can't find a copy, go back to step 1). For a given t, do not choose the same subset of places in the occurrence net twice.

3. Two places x and y in the occurrence net are said to be *concurrent* if

   - You cannot follow any path of arrows from x to y or *vice versa.*
   - There is no third place z from which you can reach both x and y, exiting z by different arrows (this is called *conflict*).

   If any of the places you marked are *not* concurrent, go to step 1.

4. Make a copy of t in the occurrence net. Call it t'. Draw an arrow from every place you marked in the occurrence net to t'. Erase the tokens.

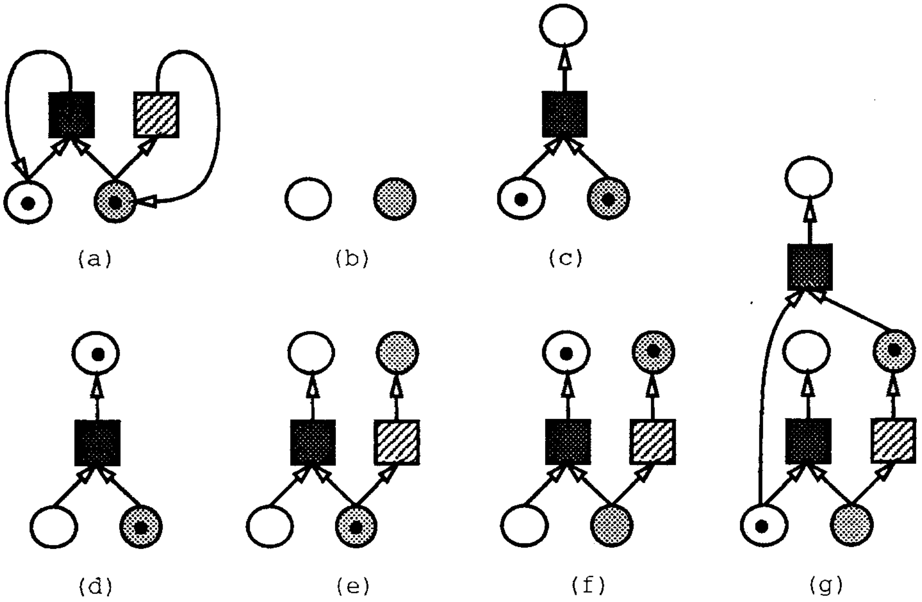


Figure 1. Unfolding a net: (a) the original net, with marking, (b) starting point of the occurrence net, (c) after one complete iteration, (d) chosen places are not concurrent, (e) after a second complete iteration, (f) chosen places are in conflict, (g) after a third complete iteration.

5. The set of places in your Petri net which are at the head of arrows originating at t is called its *postset.* For each place in the postset of t, make a copy in the occurrence net (label it, of course) and draw an arrow from t' to it.

If you make your choices fairly and have infinite time, you will build the unfolding of your Petri net (see figure 1). In the process, you will probably notice that in step 2, as soon as you choose a copy of a given place in the preset of t, it helps to black out (temporarily) any places in the occurrence net which are not concurrent with your choice. In this way you rapidly narrow down the choices, and avoid having to go back to the beginning in step 3. To see the advantage of this algorithm over an exhaustive state space search, the reader might want to make a Petri net model of the simple game from the introduction, and produce the unfolding (hint: use a special transition for the end game that removes all the tokens from Y—multiple arrows from a place to a transition are allowed).

There are several facts that are worth observing about the unfolding. The first is that the unfolding is an acyclic graph, defining a partial order on its nodes. We might think of this as providing a causal order on events (copies of the transitions). Second, bifurcations occur naturally in the structure where actual choice occurs (as opposed to the superfluous choice of the total order of unrelated transitions). This is the notion of conflict alluded to in step 3. Because of it, we are not put in the position of determining *a priori* which transitions are impersistent, and hence where bifurcations must occur. A final unfortunate fact, however, is that the unfolding may be infinite. If we wish to use it to infer properties of the original net, such as whether a given place may eventually receive a token or not, we will have to make do with a finite fragment. Such a finite fragment can be constructed, but we will require a certain amount of formalism to prove the adequacy of this fragment for our puroposes. In particular, we need to precisely define the relationship between the unfolding and the behaviors of the net.

We begin with the standard definition of a Petri net (note—there is a glossary of terms and symbols at the end of the paper). It simplifies matters if we assume that there can be only one arrow from a given place to a given transition, though this assumption is not necessary.

DEFINITION 1 *A (Petri) net is a triple $N = (P, T, F)$, where*

1. *P is a set of places,*
2. *T is a set of transitions (disjoint from P),*
3. *$F \subseteq (P \times T) \cup (T \times P)$.*

The preset of transition t, denoted ${}^\bullet t$ is the set of places p such that $p\ F\ t$. The postset of transition t, denoted $t^\bullet$ is the set of places p such that $t\ F\ p$. The notation $t_1{}^\bullet t_2$ means that $t_1{}^\bullet$ and ${}^\bullet t_2$ are incident.

An arrangement of tokens on a Petri net (a state of the token game) is usually called a *marking*. Since there can be any number of tokens on a given place, and tokens are indistinguishable, we will think of a marking mathematically as a multiset of places, or a map from the set of places to the natural numbers.

DEFINITION 2 *A marking of a net (P, T, F) is a multiset on P. A marked net has an associated initial marking I.*

We now define recursively the legal sequences of moves in the token game. The postset of a sequence of transitions (with respect to the initial marking) is the marking that remains when the sequence of transitions is fired. One property of a net that we will be interested in is the set of reachable markings. A marking is reachable if it is the postset of some legal sequence of transitions. In particular, we would like to infer from the unfolding certain properties of the reachable markings of the original net.

DEFINITION 3 *The firing sequences of a marked net N, and their postsets, are defined recursively as follows:*

1. *ϵ (the null sequence) is a firing sequence, and $\epsilon^\bullet = I$ (the initial marking).*
2. *if π is a firing sequence, σ is a transition, and $(\pi^\bullet) \supseteq ({}^\bullet\sigma)$, then $\pi\sigma$ is a firing sequence, and $\pi\sigma^\bullet = (\pi^\bullet) - ({}^\bullet\sigma) + (\sigma^\bullet)$.*

A marking M is reachable exactly when there is a firing sequence π such that $M = \pi^\bullet$.

When we unfold a Petri net, we build a structure which will be called a labeled occurrence net. This is a Petri net in which every place and transition is labeled with a corresponding place or transition in the original net. The occurrence net is a specialized form of net which must satisfy certain restrictions. First, it must be well founded, meaning that the arrows cannot be followed backward infinitely from any point. Second, it must have no forward conflict, meaning that two arrows may not converge on the same place. Third, no event may be in conflict with itself, and fourth, no two events with the same label may have the same preset. The reader might want to verify that the informal unfolding procedure above in fact constructs an occurrence net.

DEFINITION 4 *A (P, T) labeled occurrence net N' consists of a Petri net (P', T', F'), and a labeling function L' which maps P' onto a set P and T' onto a set T. The net must have the following properties:*

1. *Well foundedness: every subset of T' must contain a minimal element with respect to F'^*.*
2. *No forward conflict: for all places $p \in P'$, $p \in t_1{}^\bullet$ and $p \in t_2{}^\bullet$ implies $t_1 = t_2$.*
3. *No self-conflict: for all $t_1, t_2, t_3 \in T'$, $t_1 F'^* t_3$ and $t_2 F'^* t_3$ and ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ implies $t_1 = t_2$.*
4. *No redundancy: for all $t_1, t_2 \in T'$, $L'(t_1) = L'(t_2)$ and ${}^\bullet t_1 = {}^\bullet t_2$ implies $t_1 = t_2$.*

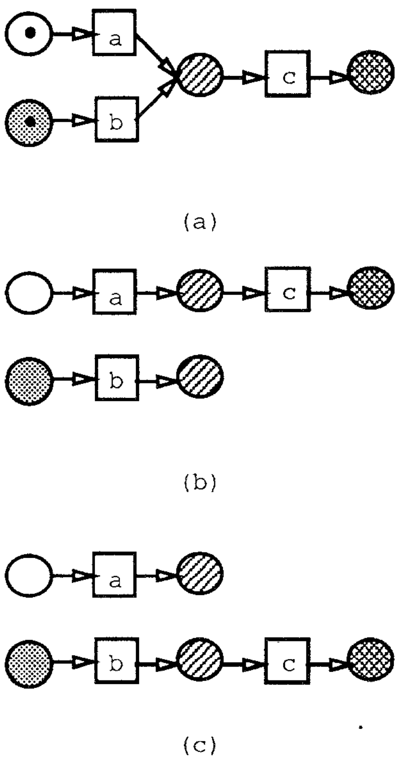
The occurrence net derived by unfolding a fiven Petri net can be characterized as follows:

DEFINITION 5 *If N is a marked net, then the unfolding of N is the maximal (P, T) labeled occurrence net (up to isomorphism) satisfying the following:*

1. *for all $t' \in T$, $L'({}^\bullet t') = {}^\bullet L'(t')$ and $L'(t'^\bullet) = L'(t')^\bullet$, and*
2. $L'(P' - \text{codomain}(F')) = I$.

The first part of the definition says that the labels of the preset and postset of any transition in the unfolding match the preset and postset of the corresponding transition in the original net. The second part says that the labels the of places in the unfolding with no predecessors match the initial marking of the original net. We will not concern ourselves here with proving that the procedure described above actually builds this net. It will be of interest, however, to prove some results concerning the relation of unfoldings and firing sequences. To do this, we introduce the notion of a *configuration*. A *configuration* of an occurrence net is a set of events representing a possible partially ordered run of the net. There is a requirement of causality for configurations: if we start with an event that is in the configuration, and trace a path backward along the arrows, all events we encounter must also be in the configuration. We call this requirement backward closure. A configuration must also be free of conflict: two events in the configuration may not be at the heads of arrows originating at the same place.

DEFINITION 6 *Let N' be a labeled occurrence net. A subset S of T' is a configuration of N' exactly when*



(a)

(b)

(c)

Figure 2. (a) A Petri net, (b) and (c) two configurations corresponding to the firing sequence *abc*.

1. *It is backward closed: if $t_1{}^\bullet t_2$, then $t_2 \in S$ implies $t_1 \in S$.*
2. *It is conflict free: for all distinct $t_1 \in S$, $t_2 \in S$, ${}^\bullet t_1$ and ${}^\bullet t_2$ are disjoint.*

A configuration describes a partially ordered multiset (pomset) of transitions of the original net. A pomset on T can be represented, for example, as a triple $(S, \leq, L)$ where S is some set, $<$ is a transitive irreflexive relation, and L is a function mapping S into T. In this case, the pomset we are interested in is

$$\mathcal{P}(S) = (S, F'^* \cap S^2, L|_S).$$

where S is a configuration. That is, it is the pomset defined by the transitive closure of F' over the transitions in the configuration. We can think of this pomset as a partially ordered run of the original net N. It can be represented equally well by the set of its linearizations, which are ordinary firing sequences of N. Unfortunately, the configurations do not partition the firing sequences into equivalence classes. That is, a single firing sequence can be contained in (the pomset of) more than one configuration, if the net is not safe[3] (see figure 2). This matter complicates somewhat the relation between configurations and firing sequences for unsafe nets, but not sufficiently to justify restricting our consideration to safe nets. The reason is that we are mainly concerned with the relationship between configurations and reachable markings, rather than firing sequences. As it turns out, each configuration can be associated with a unique reachable marking.

Note that regardless of the linearization of the events in a configuration, the same number of tokens are removed from and placed on each place by the firing sequence. Thus a configuration has a well defined final state. This final state is determined by the postset of the configuration: those places on the frontier between events in the configuration and events not in the configuration. The multiset of their labels is a marking on the original net, which we call the *final state* of the configuration.

DEFINITION 7 *Let S be a configuration of N'. The postset $S^\bullet$ of S is the set of all places $p \in P'$ such that*

1. *for all $t \in S$, $p \notin {}^\bullet t$*

2. *for all $t \in T' - S$, $p \notin t^\bullet$*

The final state of S, denoted $\mathcal{F}(S)$, is $L'(S^\bullet)$ (the multiset of labels appearing in $S^\bullet$).

Our first theorem will show that the final states of all the configurations are exactly the reachable markings of the original net.

LEMMA 1 *Given a labeled occurrence net N', if C and $C' = C \cup \{t'\}$ are configurations of N', then $C'^\bullet = C^\bullet - {}^\bullet t' + t'^\bullet$.*

Proof: Backward closure (definition 6) implies that ${}^\bullet t' \subseteq C^\bullet$. Lemma follows from definition 7. ■

LEMMA 2 *Given a labeled occurrence net N', if C and $C' = C \cup \{t'\}$ are configurations of N', then $\mathcal{F}(C') = \mathcal{F}(C) - {}^\bullet L(t') + L(t')^\bullet$.*

Proof: From definition 5 (part 1), definition of $\mathcal{F}$ and previous lemma. ■

LEMMA 3 *Given N', the unfolding of a net N, s a firing sequence of N, $t \in T$ such that st is a firing sequence of N, and C a configuration of N' such that $\mathcal{F}(C) = s^\bullet$, there exists $t' \in T'$ such that $C' = C \cup \{t'\}$ is a configuration of N', and $\mathcal{F}(C') = st^\bullet$.*

Proof: Suppose no such t' exists. Construct a new net N'' by adding event t', satisfying definition 5 part 1, such that ${}^\bullet(t') \subseteq C^\bullet$. Since C is conflict free, t' is non-self-conflicting. Also, t' is non-redundant, since by the above lemma, any transition redundant with t' must prove this lemma, and we have assumed no such t' exists. Therefore, N'' is an occurrence net, satisfying points 1 and 2 of definition 5, thus contradicting the maximality of N'. ■

LEMMA 4 *Given N', the unfolding of a net N, s a firing sequence of N, C a configuration of N' such that $\mathcal{F}(C) = s^\bullet$, and $t' \in T'$ an event such that $C' = C \cup \{t'\}$ is a configuration of N', it follows that $sL(t')$ is a firing sequence, and $\mathcal{F}(C') = (sL(t))^\bullet$.*

Proof: From lemma 2 and definition 3. ■

THEOREM 1 *Let N' be the unfolding of N. M is a reachable marking of N if and only if M is the final state of some finite configuration of N'.*

Proof: By induction on the length of firing sequences. The basis case derives from the firing sequence ϵ and the configuration $\emptyset$, using part 2 of definition 5. The induction step for the forward direction derives from lemma 3 and in the reverse direction from lemma 4. ■

3. Truncated unfoldings

Having established that the configurations of the infinite unfolding represent exactly the set of reachable markings of the original net, we are now in a position to consider whether a finite fragment of the unfolding might be constructed which is sufficient to represent all of the reachable markings. On the surface, this is a reasonable proposition, since a bounded net[4] has a finite number of reachable markings. A construction yielding such a finite fragment could be used to answer such questions as whether a given set of places in the Petri net is coverable, meaning a state can be reached with at least one token on every place in the set. This information can be used to verify that an asynchronous circuit is hazard free. The same fragment of the unfolding can also be used to determine if the token game can ever reach a deadlock–a state in which no transitions are enabled, as we will see later.

The algorithm is based on the notion of a *local configuration*—any configuration which has a unique maximal element. The backward closure of any event t' (i.e., $F'^{*-1}(t') \cap T'$) is a configuration (owing to non-self-conflict of events) and is called the local configuration of t'.

DEFINITION 8 *Let N' be a labeled occurrence net, and let $t' \in T'$. The local configuration of t', denoted $\lceil t' \rceil$ is the least backward closed subset of T', with respect to F', containing t'.*

A local configuration has, of course, a final state. We will call an event a *cutoff point* if there exists another event whose local configuration is smaller, but has the same final state. The intuition behind this definition is that any configuration containing the first event must be equivalent to a smaller configuration containing the second. Any configuration containing a cutoff point therefore adds no new reachable markings to the unfolding, thus a cutoff point may be excluded from the unfolding. This argument will be examined in detail shortly, but first let us modify the unfolding procedure, by adding one more step to the loop, between steps 3 and 4:

3.5. If there is any event t'' in the occurrence net such that $L'(\lceil t''\rceil^\bullet) = L'(\lceil t'\rceil^\bullet)$ and $|\lceil t''\rceil| < |\lceil t'\rceil|$, go to step 1.

Clearly, the test in this step can be done more quickly using a hash table to store the final states of the existing local configurations in association with their size. The algorithm terminates when we run out of choices in steps 1 and 2. It always terminates for bounded nets, and at termination, the configurations of the occurrence net represent exactly the reachable markings of the original net by their final states. The argument for this proposition runs roughly as follows:

DEFINITION 9 *Let N' be an unfolding of a Petri net N. A transition $t' \in T'$ is a cutoff point of N' exactly when there exists $t'' \in T'$ such that*

1. $\mathcal{F}\lceil t'\rceil = \mathcal{F}\lceil t''\rceil$, *and*

2. $|\lceil t''\rceil| < |\lceil t'\rceil|$

The truncation of N' is the greatest backward closed subnet of N' containing no cutoff points.

LEMMA 5 *Let N' be an unfolding of N, and let C be a configuration of N'. There exists a configuration S of N' such that C and S have the same final state, and S does not contain a cutoff point.*

Proof: If C contains no cutoff point, lemma is trivial. Else, let t be a cutoff point contained in C. By definition, there is a transition t' such that $\lceil t\rceil$ and $\lceil t'\rceil$ have the same final state and $|\lceil t'\rceil| < |\lceil t\rceil|$. We can show by induction on $|C| - |\lceil t\rceil|$ that there exists a configuration C' containing t' such that the final states of C and C' are equal, and $|C| - |\lceil t\rceil| = |C'| - |\lceil t'\rceil|$. This implies that $|C'| < |C|$. Therefore, by induction there exists a configuration S containing no cutoff points having the same final state as C', and hence the same final state as C. ■

THEOREM 2 *Let N' be the unfolding of N. M is a reachable marking of N if and only if M is the final state of some finite configuration of the truncation of N'.*

Proof: Forward direction. By theorem 1, if M is reachable, then there exists C configuration of N' such that $\mathcal{F}(C) = M$. By the above lemma, there exists configuration S such that $\mathcal{F}(C) = M$, containing no cutoff points. The truncation must contain S, since otherwise there would be a greater subnet of N' containing no cutoff points. Reverse direction follows from theorem 1. ■

THEOREM 3 *Let N' be an unfolding of N. If N is bounded, then the truncation of N' is finite.*

Proof: Given a transition t in the truncation, let π be a longest chain of events ending in t such $t_1{}^\bullet t_2$ holds for each successive pair (t_1, t_2) in the chain. For each such pair,

$|[t_1]| < |[t_2]|$. The length of π cannot be greater than the number of reachable markings of N, since otherwise the local configurations of two elements of the chain would have the same final state, hence the latter of the two would be a cutoff point. We can show by induction that the number of transitions for which π has a given length is finite, since if transition t has length n all the predecessors of t have length $n - 1$ or fewer, leaving us a finite number of combinations for the preset of t. Therefore the number of transitions in the truncation is finite. ■

As an example of termination of the algorithm, consider the net of figure 3, which represents the dining philosophers. In this scenario, there are n concurrent processes (philosophers), each of which must acquire the use of two shared resources (forks) in order to execute its critical section (eating spaghetti). The processes are organized in a ring, with each neighboring pair sharing one resource. Figure 4 shows the completed unfolding for the case of three philosophers ($n = 3$). The cutoff points are marked with an X. The local configuration of each of these transitions is equivalent to the empty configuration (initial marking). The size of the unfolding is not only bounded, but is linear in the number of philosophers. The number of states (reachable markings) is exponential however, as shown below:

n	unfolding size (transitions)	reachable states
2	9	22
3	13	100
4	17	466
5	21	2164

One of the questions we can easily answer using the truncated unfolding is whether a given place is *coverable*, meaning that at least one token can be put on that place by means of some firing sequence.

COROLLARY 1 *Let N' be an unfolding of N. A place p in N is coverable iff there is some place p' in the truncation of N' labeled with p.*

Proof: By theorem 2. ■

Since we can test coverability of a single place in the net, we can easily test it for a subset of places by simply adding a transition to the original net whose preset is the set of places in question, and whose postset is a single place not otherwise used in the net. Coverability of this latter place is equivalent to coverability of the set.

4. Application example

We now consider a more realistic example than the dining philosophers–a speed-independent [13] circuit designed to implement a distributed mutual exclusion (DME) protocol.

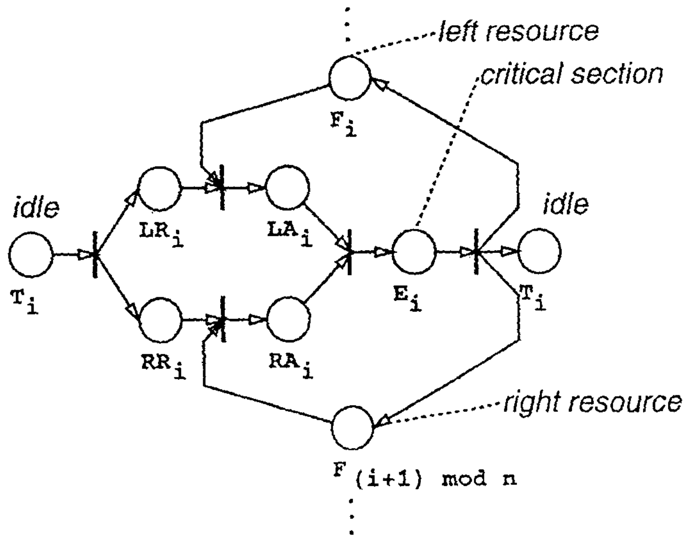


Figure 3. Dining philosophers net.

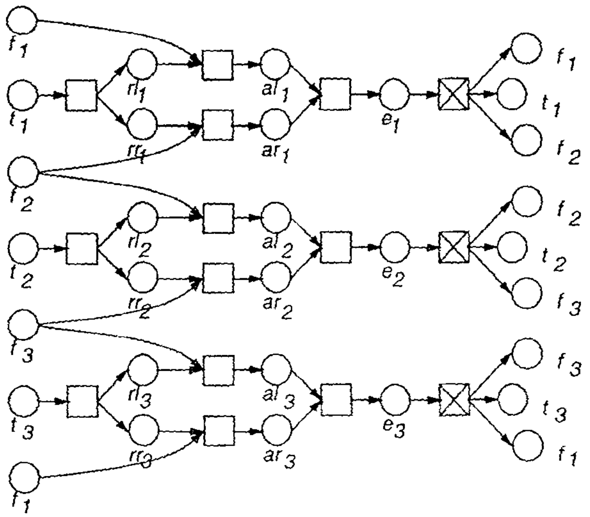


Figure 4. Unfolding of the dining philosophers net.

The circuit was designed by Alain Martin [8] and has been analyzed using an abstracted trace theoretic model by Dill [4].

Networks of logic gates in speed-independent circuits are readily modeled by Petri nets. A network of n gates can be modeled by a Petri net of $O(n)$ places. When we model a

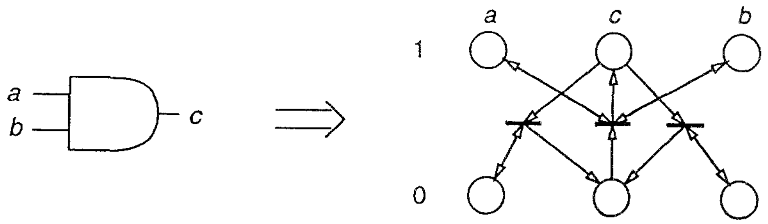


Figure 5. Translation from circuit to net.

network of gates as a Petri net, we introduce two places for each input of each gate. One represents the input in a logic low state, while the other represents the input in a logic high state. Transitions in the Petri net correspond to rising or falling transitions of gate outputs. A rising transition of a gate output removes all the logic low tokens from the inputs to which it is connected, and places tokens on the corresponding logic high places.

As an example, figure 5 shows a net fragment representing an AND gate with one fanout. When both inputs of the gate are at the logic high state, we can move a token from the place representing logic low at the output to the place representing logic high. Similarly, if either input is at the logic low state, we can move a token from the place representing logic high at the output to the place representing logic low.

It is important in designing asynchronous circuits to know that their behavior is free of hazards. A hazard occurs, for example, if the AND gate's output is enabled to rise while one of its inputs is enabled to fall. Such a situation can produce a pulse on the gate output which is not a legitimate digital signal, and thus violates the assumptions under which the gate is modeled as a digital device. The problem of whether or not this hazard can occur can be posed as a coverability problem, of that set of places enabling both transitions. Alternatively, since hazards correspond to conflicts in the unfolding, the problem can be solved by constructing the unfolding and examining it for conflicts. In particular, we search for two transitions in the unfolding which are in conflict (presets are incident), and which may be simultaneously enabled (union of presets is concurrent). The DME circuit also uses special two-way mutual exclusion elements as components, which are immune to certain hazards. In checking the DME ring for hazards, we ignore conflicts between rising transitions of a mutual exclusion element's acknowledge outputs.

Figure 7 shows the results of the occurrence net unfolding procedure for the Petri net model of the DME circuit, for rings with one to nine cells. The depth of the occurrence net unfolding for the case of 5 cells was 141 transitions. The number of transitions in the unfolding, shown in part (a) of the figure, increases quadratically in the number of cells. This is because as the number of cells in the ring increases, a request must be relayed through a greater number of stages in order to obtain the token, in the worst case. At the same time, the number of cells which may request also increases. The occurrence net therefore grows in both width and depth in proportion to the number of cells. The time to construct the unfolding (running a LISP implementation on a Sun3 workstation) appears to increase quartically, as shown in part (b) of the figure. Finally, as we increase the number of cells in

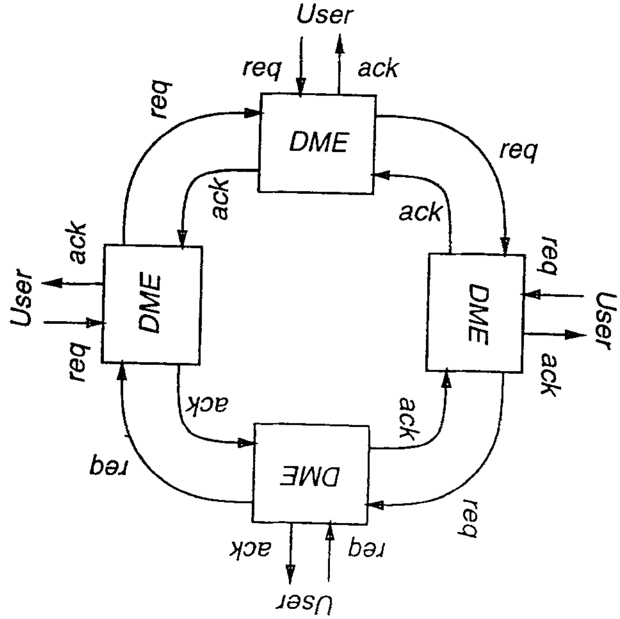


Figure 6. Distributed mutual exclusion circuit.

the ring, the number of reachable global markings increases exponentially, as shown in part (c) of the figure (on a logarithmic scale).[5] The number of states increases asymptotically by slightly less than a factor ten for each added cell. The fact that the truncated unfolding increases quadratically in size implies that this exponential increase in states is due only to the arbitrary ordering of independent transitions.

This is an interesting result, because it highlights a distinction between analysis using unfolding, and methods of Valmari [14], [15], Godefroid [5], [6] and Yoneda [16], which perform state space search using the notion of persistence to limit the search. Recall that a persistent transition is one which remains enabled regardless of the firing of other transitions. In the case of gates, any output transition that is enabled may become disabled by a transition on one of the inputs. In general, determining whether such a disabling transition can occur before the output transition occurs is a difficult problem, for which one must consider the global structure of the circuit. Experiments by Holger Schlingloff[6] have confirmed that the method of Yoneda provides little or no improvement over an ordinary search strategy.

There are, however, other methods of analysis which can avoid large state space searches, some of which do not directly exploit partial ordering of events. For example, the trace theory approach of Dill [4] has been applied to the DME circuit. This requires an abstract model of the arbiter cell to be created by hand. The abstraction reduces the state explosion problem, but does not entirely solve it, since even with the reduced model, the number of states increases exponentially with the number of components. Probst [12] reports a method which requires quadratic space and time in the number of cells, though it requires

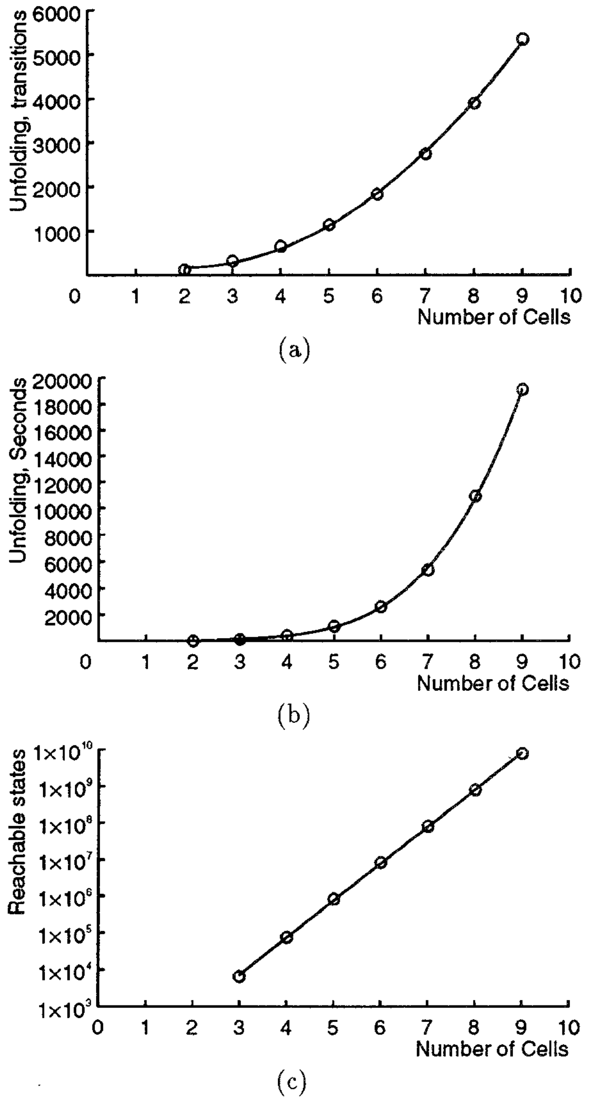


Figure 7. Performance of unfolding method on hazard-detection problem for the distributed mutual exclusion circuit.

a pomtree model of the circuit to be constructed by hand. Finally, the symbolic model checking technique [2] has also been applied to the DME circuit. The basic symbolic model checking algorithm requires cubic time and linear space (in the number of cells). Burch and Long[7] have obtained $O(n^{2.5})$ time for the DME using symbolic model checking with a modified search order [1]. This method requires some hand optimization, however. In any event, it appears that the symbolic model checking method yields somewhat better asymptotic performance for the DME circuit, though both methods effectively solve the state explosion problem. The unfolding method has an advantage over the symbolic model checking method in that no variable ordering or other heuristic information is required. It is not difficult to construct a variation on the dining philosophers for which there is no good variable ordering for symbolic model checking, but for which the unfolding is still linear space (in the number of philosophers). However, the author is presently unaware of any practical circuits for which this is the case.

5. Deadlock and occurrence nets

Besides coverability, another interesting problem for Petri nets is the question of deadlock. A *terminal marking* of a Petri net is one in which no transitions are enabled. Reachability of a terminal (or deadlocked) state cannot be framed in terms of the coverability problem. However, since the unfolding represents all reachable markings, a net has a reachable terminal marking if and only if its unfolding has a reachable terminal marking. The problem of existence of a terminal marking in an occurrence net is $\mathcal{NP}$-complete. This is easily shown by reduction from 3-SAT.[8] To see this, consider the formula $(x_1 + y_1 + z_1)\ (x_2 + y_2 + z_2) \cdots (x_n + y_n + z_n)$ where each x_i, y_i and z_i is a positive or negative literal. Assume the formula has m variables. Let the positive literals be $l_1, \ldots, l_m$, and the negative literals be $\bar{l}_1, \ldots, \bar{l}_m$. In polynomial time, we can construct a net which has a terminal marking if and only if the formula is satisfiable. The initial marking of the net is a set of places $\{v_1, \ldots, v_m\}$. There is a place representing each positive literal $l_1, \ldots, l_m$ and each negative literal $\bar{l}_1, \ldots, \bar{l}_m$. For each variable v_i, there is a transition from v_i to l_i and from v_i to $\bar{l}_i$. For each conjunct $(x_i + y_i + z_i)$, there is a transition c_i, whose preset is $\{\bar{x}_i, \bar{y}_i, \bar{z}_i\}$. In other words, the transition c_i is enabled to fire if and only if $(x_i + y_i + z_i)$ is false. Thus, some transition c_i is enabled to fire if and only if the whole formula is false. The postset of each transition c_i is the single place $\{q\}$, and there is a transition from $\{q\}$ to $\{q\}$. Thus, if any c_i fires, the net may never reach a terminal marking. As a result, there is a terminal marking of the net if and only if the formula is satisfiable. For example, figure 8 shows the net constructed for the formula $(a + b + \bar{c})(b + c + \bar{d})$.

The reader may easily verify that the size of the unfolding of such a net (up to the cutoff points) is linear in the size of the original net. In fact, it is essentially the same net, except that the place q occurs n times in the unfolding. Since all reachable markings of the original net occur as configurations of the unfolding, the unfolding has a terminal marking if and only if the formula is satisfiable. Hence 3-SAT is $\mathcal{P}$-time reducible to reachability of a terminal marking of an unfolding. Since the configuration representing the terminal marking can be guessed in $\mathcal{P}$-time in the size of the unfolding, and also tested in $\mathcal{P}$-time, it follows that the problem is in $\mathcal{NP}$, and hence $\mathcal{NP}$-complete.

Interestingly, however, the problem is readily solved in practice even for very large unfoldings, using an algorithm based on techniques of constraint satisfaction search. The key observation which leads to this algorithm is that there is no terminal marking exactly when every configuration of the unfolding is a subset of some configuration containing a cutoff point. This is simply because if there is no terminal marking, then every configuration can be extended to a configuration which is arbitrarily large. A configuration C' can be

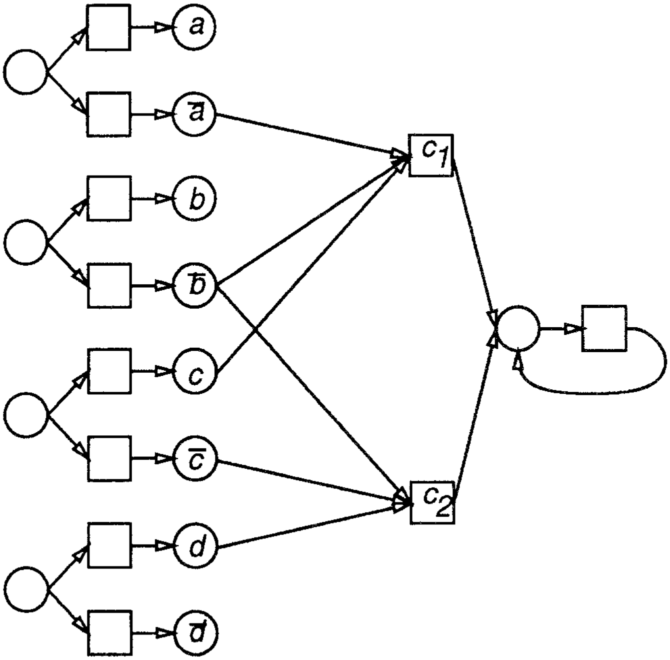


Figure 8. Reduction from 3-SAT problem to a terminal marking problem.

extended to a configuration containing transition t' if and only if the union of C' and the local configuration of t' is a configuration. If it is not, then no set containing C' and t' is a configuration. If the union is not a configuration, we will say that C' and t' are in conflict. Hence, there is a terminal marking if and only if there is a configuration which is in conflict with every cutoff point. The search for such a configuration can be carried out using branch and bound techniques. For example, if a configuration C' is in conflict with a cutoff point t', there must be a transition $t'_1 \in C'$ which is in conflict with t'. Such a transition t'_1 will be called a *spoiler* of t'.

There exists a configuration in conflict with all of the cutoff points (equivalently, there exists a terminal marking) if and only if there exists a configuration containing a spoiler for every cutoff point. The set of spoilers contained in this configuration will be called T_s. The following algorithm uses branch and bound techniques to find such a set T_s if one exists:

```
let $B$ be the set of the cutoff points, $T_s = \emptyset$
while $B$ is not empty do
    let $t$ be the element of $B$ with the fewest spoilers
    if $t$ has no spoilers, then backtrack
    choose an element $t'$ from the spoilers of $t$
    add $t'$ to $T_s$
    delete all transitions in conflict with $T_s$
end do
```

Note that in line 3 of the procedure, the cutoff point with the smallest number of spoilers is chosen so that the number of choices in line 5 is minimized. Whenever a spoiler for a given cutoff point is chosen to belong to T_s in line 5, everything in conflict with T_s is eliminated from future consideration in line 7. Note that the cutoff points in conflict with T_s are also eliminated, which cuts down on the amount of future branching. Whenever there is a cutoff point with no remaining spoilers, the procedure backtracks, from line 4 to the most recent occurrence of line 5 where there are remaining choices. If there are no remaining choices, the procedure fails. When backtracking occurs, the net is returned to the state it was in at the point where execution is being resumed. This backtracking is easily implemented by keeping a stack of the remaining choices for t' in each iteration of the loop, and marking each transition in the net with the level of the stack at the time it was "removed". Interestingly, if the procedure terminates successfully, the remaining net has the property that every maximal configuration corresponds to a class of terminal firing sequences. This makes it straightforward to extract such a sequence, by building a maximal configuration, and choosing any linearization of that configuration.

Because of the backtracking, this procedure is exponential (as it must be, if $\mathcal{P} \neq \mathcal{NP}$). However, this is only the worst case. The dining philosophers serve as an example of a case in which the exponential complexity is avoided. In fact, the procedure finds the terminal marking in time which is *linear* in the number of philosophers. This is easily seen by examining the unfolding of the Dining Philosophers net in figure 4. There is one cutoff point in this net for each process. Initially, each of these transitions has two spoilers, which correspond to the two resources required to enter the critical region being granted to the two neighboring processes. Regardless of which cutoff point is used first, the symmetry is then broken as the part of the net in conflict with one of the two spoilers is removed. This removes, in particular, the transition which granted one of the resources to the first philosopher, hence one of its neighbors now has only one spoiler, so there is only one choice available the next time line 5 is reached. After this spoiler is added to T_s, the remaining neighbor of the second philosopher now has only one spoiler. This process continues without backtracking until it has come full circle and the terminal marking is found. Note that if the cutoff point with the fewest spoilers were not chosen in line 3, the procedure might have examined an exponential number of candidates for T_s before a valid one was found.

For the DME circuit example, we find that the run time of the deadlock algorithm is 218 seconds for a ring of five cells, and 6600 seconds for a ring of 9 cells. Hence, even though the algorithm is exponential in the worst case, in this case it runs in reasonable time for an unfolding of over 5000 transitions. It is clear that the branch and bound technique quickly narrows down the number of choices for this example.

6. Conclusion

We have observed that much of the complexity in the analysis of concurrent finite state systems may be avoided by considering events to be partially rather than totally ordered. Occurrence nets, as described by Nielsen, Plotkin and Winskel have proved to be a natural structure for such an analysis, provided the infinite unfolding of a net can be truncated in a suitable manner. Here, we have extended the work of Nielsen et al. by providing such a method of truncation.

We have also observed that Petri nets can provide a model of asynchronous circuits that makes the partial ordering of events explicit. This made it possible to apply unfolding methods to hazard analysis. For a commonly studied example circuit (really a class of circuits of increasing size), the unfolding method reduced the cost of the analysis from exponential in the size parameter to polynomial. This demonstrates a possible advantage of unfoldings over state space search methods using partial orders, which in general are not effective for asynchronous circuits.

The problem of deadlock analysis has been shown to be hard, given the truncated unfolding, but solvable in practice using a branch and bound technique.

This paper has not dealt with verification in the sense of comparing an implementation and specification. Rather, we have treated only the problem of searching the state space for certain conditions such as hazards and deadlock. However, it is plausible that unfoldings could be used for various verification problems that involve state space search, such as conformation checking [4], temporal logic model checking [3], or language containment of ω-automata [7].

Conformation checking seems to be the most natural candidate, since it deals only with finite sequences. In such a program, trace structures might be modeled with Petri nets. Verification of trace conformance involves forming the composition of an implementation with the "mirror" of its specification, and testing this composition for "chokes" that occur when one process is enabled to produce output that the receiving process is not enabled to receive. Given the unfolding of the composition, the problem of testing for chokes is $\mathcal{NP}$-complete (like the deadlock problem). However, there are indications that this problem can be solved efficiently in the special case where there is no "autoconcurrency" (concurrently enabled transitions on the same wire). In any event, this is an application of unfolding that perhaps deserves exploration.

Building unfolding into the model checking or ω-automata frameworks is a more challenging problem, since these are concerned with properties of infinite runs, hence the notions of coverability and deadlock are not applicable. The expressive power of Petri nets may also prove to be a limitation. Extracting an unfolding from richer process models may pose interesting problems. In all events, there is much work left to be done.

Glossary

$\bullet$	(see *postset* and *preset*)
R^*	transitive closure of relation R
$L\|_S$	function L with domain restricted to set S
$\lceil t' \rceil$	local configuration of t'
$\|S\|$	cardinality of set S
F	set of edges of N
F'	set of edges of N'
$\mathcal{F}$	final state of... (see definition 6)
I	initial marking of N (see definition 2)

L'	labeling of N' (see definition 4)
N	original net (see definition 1)
N'	unfolding of N
P	set of places of N
P'	set of places of N'
T	set of transitions of N
T'	set of transitions of N'
backward closed	closed under predecessor relation (see definition 6)
backward conflict	distinct transitions sharing input
bounded	allowing only a bounded number of tokens on each place
configuration	(see definition 6)
conflict	backward conflict
coverable	may be marked by some firing sequence
cutoff point	(see definition 9)
event	transition in unfolding
final state	(of S) = marking resulting from configuration S (see definition 6)
firing sequence	(see definition 3)
forward conflict	distinct transitions sharing output (see definition 4)
hazard	condition resulting in non-digital behavior
labeling	map from unfolding to original net (see definition 4)
local configuration	(of t') = configuration with unique maximal element t' (see definition 8)
marking	multiset of tokens (see definition 2)
multiset	map from some set to the natural numbers
occurrence net	(see definition 4)
Petri net	(see definition 1)
place	holds tokens (see definition 1)
postset	outputs of a transition (see definition 1) ... of a firing sequence (see definition 3) ... of a configuration (see definition 7)
preset	inputs of a transition (see definition 1)
reachable	(see definition 3)
spoiler	(of t') = transition in conflict with cutoff point t'
terminal marking	marking which enables no transitions (deadlock)
transition	moves tokens (see definition 1)
truncation	(of unfolding N') = N' up to cutoff points (see definition 9)
unfolding	(see definition 5)
well founded	every subset contains a minimal element (see definition 4)

Notes

[1] Actually, if the game contains cycles, such an approach might lead us to become stuck interminably in a cycle, missing a possible solution. The solution to this problem can be found in [5].

[2] A pomtree is a tree whose edges are labeled with partially ordered multisets of events.

[3] An unsafe net is one where more than one token may appear on a given place.

[4] A bounded net generates no more than a fixed number of tokens on each place, for any firing sequence.

[5] The number of reachable states was established using the symbolic model checking technique [2].

[6] Personal communication.

[7] Personal communication.

[8] 3-SAT is satisfiability of a Boolean formula in conjunctive normal form, with three literals in each conjunct.

References

1. J.R. Burch, E.M. Clarke, and D.E. Long, Symbolic model checking with partitioned transition relations. In the Proceedings of VLSI'91.
2. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and J. Hwang, Symbolic model checking: 10^{20} states and beyond. In *Proceedings of the Fifth Annual Symposium on Logic in Computer Science*, June 1990.
3. E.M. Clarke and E.A. Emerson, Synthesis of synchronization skeletons for branching time temporal logic. In Dexter Kozen, editor, *Logic of Programs: Workshop,* volume 131 of *Lecture Notes in Computer Science,* Yorktown Heights, New York, May 1981. Springer-Verlag.
4. D. Dill, Trace theory for automatic hierarchical verification of speed-independent circuits. Technical Report 88–119, Carnegie Mellon University, Computer Science Dept., 1988.
5. P, Godefroid, Using partial orders to improve automatic verification methods. In *Workshop on Computer Aided Verification,* 1990.
6. P. Godefroid and P. Wolper, A partial approach to model checking. In *LICS,* 1991.
7. R.P. Kurshan, Testing containment of ω-regular languages. Technical Report 1121-861010-33-TM, Bell Laboratories, 1986.
8. A.J. Martin, The design of a self-timed circuit for distributed mutual exclusion. In Henry Fuchs, editor, *1985 Chapel Hill Conference on VLSI,* pp. 245–260. Computer Science Press, 1985.
9. M. Nielsen, G. Plotkin, and G. Winskel, Petri nets, event structures and domains, part I. *Theoretical Computer Science,* 13:85–108, 1981.
10. D.K. Probst and H.F. Li, Abstract specification, composition, and proof of correctness of delay-insensitive circuits and systems. Technical Report, Concordia University, Dept. of Computer Science, 1989.
11. D.K. Probst and H.F. Li, Using partial order semantics to avoid the state explosion problem in asynchronous systems. In *Second Workshop on Computer Aided Verification,* June 1990. Also LNCS 531, pp. 15–24.
12. D.K. Probst and H.F. Li, Partial order model checking: A guide for the perplexed. In *Third Workshop on Computer Aided Verification*, pp. 405–416, July 1991. Also LNCS 575, pp. 322–331.
13. C.L. Seitz, System timing. In Carver Mead and Lynn Conway, editors, *Introduction to VLSI Systems,* pp. 218–262. Addison-Wesley, 1980.
14. A. Valmari, Stubborn sets for reduced state space generation. In *10th Int. Conf. on Application and Theory of Petri Nets,* 1989.
15. A. Valmari, A stubborn attack on the state explosion problem. In *Workshop on Computer Aided Verification,* 1990.
16. Tomohiro Yoneda, Yoshihiro Tohma, and Yutaka Kondo, Acceleration of timing verification method based on time Petri Nets. *Systems and Computers in Japan,* 22(12):37–52, 1991.