# Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits

K. L. McMillan

School of Computer Science
Carnegie Mellon University

## 1 Introduction

A number of researchers have observed that the arbitrary interleaving of concurrent actions is a major contributor to the state explosion problem, and that substantial efficiencies could be obtained if the enumeration of all possible interleavings could be avoided. As a result, several have proposed verification algorithms based on partial orders [Val89, Val90, God90, GW91, PL89, PL90, PL91, YTK91]. The method presented in this paper is based on unfolding a Petri net into an acyclic structure called an *occurrence net*. The notion of unfolding was introduced by Nielsen, Plotkin and Winskel as a means for giving a concurrent semantics to nets, but in this case the goal is to avoid the state explosion problem. An algorithm is introduced for constructing the unfolding of a net, which terminates when the unfolded net represents all of the reachable states of the original net. The unfolding is adequate for testing reachability (to be more precise, *coverability*) and deadlock properties. Reachability testing can be used to prove safety properties of finite state systems, for example in Dill's trace theory for asynchronous circuits [Dil88]. It is shown using an asynchronous circuit example that the unfolding can be polynomial in the circuit size while the state space is exponential. In contrast, the stubborn sets method of Valmari [Val89, Val90] and trace automaton method of Godefroid [God90, GW91] are ineffective in reducing the state explosion problem for asynchronous circuit models, because of the ubiquity of confusion in such models. In addition, because the unfolding method is fully automatic, it has a certain advantage over behavior machines method of Probst [PL89, PL90, PL91], which requires a pomset grammar describing the circuit's behavior to be constructed by hand.

## 2 The unfolding operation

Briefly, an occurrence net is a Petri net without backward conflict (two transitions outputting to the same place), and without cycles. Such a net can be obtained from an ordinary place/transition net by an unfolding process. Figure 1 shows an example of a net and part of its unfolding. Since the occurrence net it is acyclic and rooted, there is a natural well founded (partial) order on the transitions and places of the net. This order is called the dependency order. It is impossible for a transition of the occurrence net to fire unless all of its predecessors in the dependency order have fired.

The most important theoretical notion regarding occurrence nets is that of a *configuration*. A configuration represents a possible partial run of the net – it is any set of transitions that satisfies the following conditions:

1. If any transition is in the configuration, then so are all of its predecessors in the dependency order (a configuration is *downward closed*).

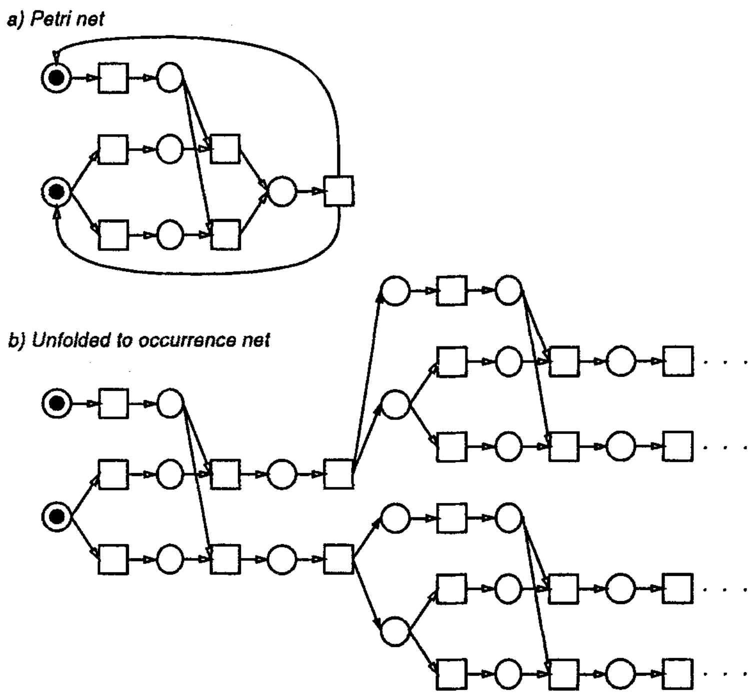

Fig. 1. Unfolding example.

2. A configuration cannot contain two transitions in *conflict*, meaning that both input from the same place.

An example of a configuration is shown in figure 2, with elements of the configuration filled in black. Two transitions in the figure are hatched in. Either of these transitions can be added to the black set to form a new configuration. Adding any other transition would be illegal, however, since it would either violate downward closure or conflict-freeness.

In an unfolding, each transition corresponds to a transition of the original net, and each place corresponds to a place of the original net. We can associate each configuration of the unfolding with a state (marking) of the original net by simply identifying those places whose tokens are produced but not consumed by the transitions in the configuration. This set is marked with black dots in figure 2. Mapping this set back onto the original net, we obtain the *final state* of the configuration.

The final theoretical notion we need regarding unfoldings is that of a *local configuration*. The local configuration associated with any transition consists of that transition and all of its predecessors in

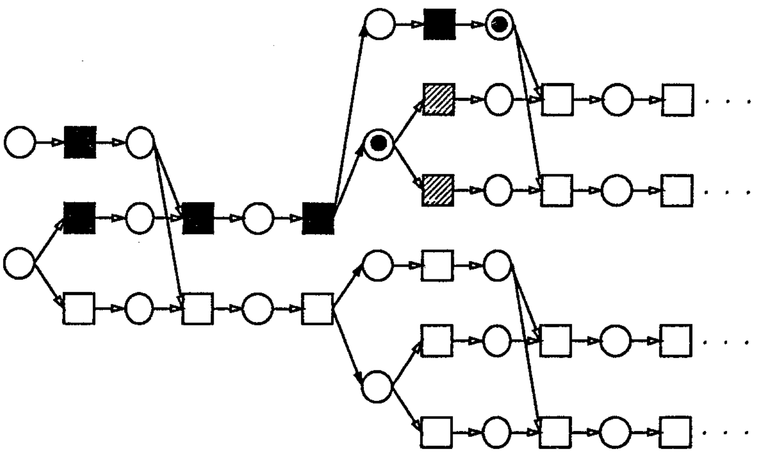

Fig. 2. Configuration

the dependency order (that is, the downward closure of the transition as a singleton). This is the set of transitions which necessarily are contained in any configuration containing the given transition. Note that a local configuration may not exist if this set contains two transitions in conflict.

We are now ready to consider the problem of building a fragment of the unfolding which is large enough to represent all of the reachable markings of the original net. Building the unfolding itself is straightforward. The process starts with a set of places corresponding to the initial marking of the original net. The unfolding is grown by finding a set of places in the unfolding which correspond to the inputs (preset) of a transition in the original net, then adding a new instance of that transition to the unfolding, as well as a new set of places corresponding to its outputs (postset). If the new transition has no conflicts in its local configuration (more precisely, if it *has* a local configuration) it is kept, otherwise it is discarded. This is because the existence of a conflict means that the new transition can occur in no configurations of the unfolding.

The key to termination of the unfolding is to identify a set of transitions of the unfolding to act as *cutoff points*. This set must have the following property: any configuration containing a cutoff point must be equivalent (in terms of final state) to some configuration containing no cutoff points. From this definition, it follows that any successor of a cutoff point can be safely omitted from the unfolding, without sacrificing any reachable markings of the original net. To see this, suppose we have built the unfolding only up to the cutoff points, in the sense that any new transition we can add must have a cutoff point as a predecessor. From this point on, any transition we add must be descended from some cutoff point. Thus, any configuration we might add to the unfolding must have the same final state as some configuration already present.

A sufficient condition for a transition to be a cutoff point is the following: the final state of its local configuration is the same as that of some other transition whose local configuration is smaller. The proof of this statement is as follows: suppose there are two transitions $t_1$ and $t_2$, whose local configurations have the same final state, with that of $t_2$ being smaller. Now imagine a configuration $C_1$ (local or otherwise) containing $t_1$. We can obtain $C_1$ from the local configuration of $t_1$ by adding the transitions in the difference one at a time, in an order consistent with the dependency relation. According to our construction, at each step of this process, there is a corresponding transition we can add to the local configuration of $t_2$ leading to the same final state. Hence, we can build a configuration $C_2$ containing $t_2$ which has the same final state, *but is at least one transition smaller than* $C_1$, since we started from a smaller set. Thus if any configuration contains a cutoff point, it is equivalent to a smaller configuration. Configurations cannot be made arbitrarily small, however, so any configuration containing a cutoff point must be equivalent to a configuration not containing a cutoff point. Since all the reachable states are represented by configurations containing no cutoff points, it is unnecessary to build the unfolding beyond any cutoff point.

We can find the cutoff points by simply keeping a hash table of all transitions, indexed by the final state of the local configuration. If when generating a transition, we find in the table a transition with equivalent but smaller local configuration, we discard the new transition. We can show, as follows, that this process is guaranteed to terminate if the original net is bounded and finite. First, the depth of the unfolding must be bounded by the number of number of reachable markings. The depth of a given transition in the unfolding is the longest chain of predecessors of that transition. Each transition in this chain has a local configuration, and these local configurations form a chain of increasing size. If the depth of the given transition is greater than the number of reachable markings of the original net, then by the pidgeon-hole principle, two of these local configurations must have the same final state. This cannot be, however, since in this case one of the transitions in the chain would have been determined to be a cutoff point. If the original net is bounded, it has a finite number of reachable markings, hence the depth of the unfolding is bounded. If the original net is finite, we can show by induction that the number of transitions at any given depth in the unfolding is finite. Hence the total number of transitions generated by the unfolding process is finite.

As an example of termination, consider the net of figure 3, which represents the dining philosophers paradigm. 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. We observe that the size of the unfolding is not only bounded, but is linear in the number of philosophers, while the number of states is exponential as shown in table 1.

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

Table 1. Unfolding size and number of states for Dining Philosophers

[t]

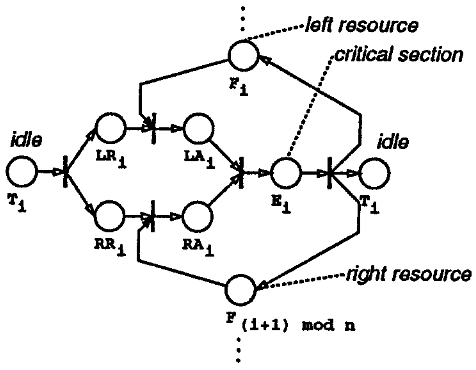

**Fig. 3.** Dining philosophers net.

Recall that in growing the unfolding, it is necessary to enumerate all of the subsets of places which correspond to the inputs of transitions. The complexity of this is $O(\binom{n}{i})$, where $n$ is the size of the unfolding, and $i$ is the largest number of inputs of any transition. This is,of course, bounded by $n^i$, which is polynomial given a fixed value of $i$. In practice, however, the number of subsets which are considered can be reduced quite effectively, using the following two techniques. First, suppose we are enumerating the subsets: we need not add any place to the set if the result would not be contained in the set of inputs of any transition. Second whenever a place is added to the set, we can immediately eliminate from consideration all of the places which have a predecessor in conflict with a predecessor of the new element, since any transition with both places as inputs would be discarded. We add transitions to the net in order increasing size of the local configuration, so that we can use a hash table to determine whether or not each transition is a cutoff point. Thus, whenever a candidate for a transition in the unfolding is generated, it is placed in a queue ordered by increasing local configuration size. The places of the net are enumerated by pulling the first element $t'$ from this queue, testing whether it is a cutoff point, and if not, generating places for its outputs. The procedure terminates when the queue of candidate transitions becomes empty. Figures 5 and 6 show a pseudo-code implementation of this procedure. The pseudo-code is written somewhat inefficiently in places for simplicity.

In function Unfold, the arguments $P$, $T$ and $M_0$ are the places, transitions and initial marking of the original net. Each place in the unfolding is represented by a pair $(place, preds)$, where $place$ is the corresponding place in the original net, and $preds$ is the set of immediate predecessor transitions in the unfolding (note that since there is no backward conflict, the size of this set is at most one). Each transition in the unfolding is represented by a pair $(trans, preds)$, where $trans$ is the corresponding transition in the original net, and $preds$ is the set of immediate predecessor places in the unfolding. The function returns $P'$ and $T'$, the set of places and transitions, respectively, of the unfolding. There is also a queue $Q'$ of transitions to be expanded, and a hash table (HashTable) used for identifying

[t]

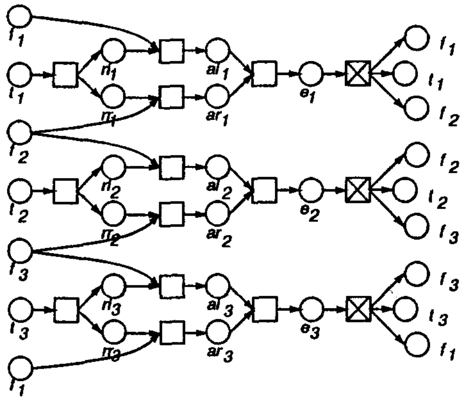

**Fig. 4.** Unfolding of the dining philosophers net.

cutoff points.

Coverability problems can be solved using the unfolding in the following way. Imagine we have a set of places in the original net, and we wish to determined whether this set can every be simultaneously marked. We simply add a new transition to the net, whose inputs are the given set, and then construct the unfolding. If the unfolding contains any instance of this new transition, the set is coverable, and otherwise not.

## 3 Application example

We now consider a more realistic example than the dining philosophers – a speed-independent [Sei80] circuit designed to implement a distributed mutual exclusion (DME) protocol. The circuit was designed by Alain Martin [Mar85] and has been analyzed using an abstracted trace theoretic model by Dill [Dil88].

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 network of gates as a Petri net, we introduce two places for each input of each gate. One represents the 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 7 shows the net fragment representing an AND gate. 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.

```
global $P'$,$T'$,$Q'$,HashTable[]
function Unfold($P$,$T$,$M_0$)
    $P' = T' = Q' = \emptyset$; clear HashTable
    for each $p \in M_0$ do
        add $p' = (p, \emptyset)$ to $P'$
        GenTrans($\{p'\}, T$)
    end for
    while the queue $Q'$ is not empty do
        pull the first $t'$ off of $Q'$
        if not IsCutoffPoint?($t'$) do
            for each $p$ in outputs of $trans(t')$ do
                add $p' = (p, \{t'\})$ to $P'$
                GenTrans($\{p'\}, T$)
            end for
        end if
    end while
    return($P'$,$T'$)
end function

procedure GenTrans($S', T$)
    if not exists $t \in T$ such that $place(S') \subseteq$ inputs of $t$ then return
    if Predecessors($S'$) has forward conflict then return
    forall $t \in T$ do if $place(S') =$ inputs of $t$ then
        add $t' = (t, S')$ to set $T'$
        insert $t'$ in $Q'$ in order of |LocalConfig($t'$)|
    end for
    for all $p' \in P$ where $p'$ older than any member of $S'$ do
        GenTrans($S' \cup p', T$)
end procedure
```

**Fig. 5.** Pseudo-code implementation of unfolding procedure

A dynamic hazard occurs, for example, if the AND gate's output is enabled to rise while one of the inputs is enabled to fall. The problem of whether or not a dynamic hazard can occur can thus be posed as a coverability problem. Alternatively, since dynamic hazards correspond to dynamic conflicts in the unfolding, the problem can be solved by constructing the unfolding and examining it for dynamic conflicts, *i.e.*, two transitions which are in conflict, and which may be simultaneously enabled. 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 9 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

```
function IsCutoffPoint?($t_1'$)
    $C_1' = $ LocalConfig($t_1'$)
    $S_1' = $ FinalState($C_1'$)
    $L' = $ HashTable[HashFun($S_1'$)]
    forall $t_2'$ in $L'$ do
        $C_2' = $ LocalConfig($t_2'$)
        if $S_1' = $ FinalState($C_2'$) and Size($C_2'$) $<$ Size($C_1'$) then return(1)
    end for
    add $t_1'$ to HashTable[HashFun($S_1'$)]
    return(0)
end function

function LocalConfig($t'$)
    return(Predecessors($\{t'\}$) $\cap T'$)
end

function Predecessors($S'$)
    do
        $S' = S' \cup$ preds($S'$)
    until $S'$ unchanged
end function

function FinalState($C'$)
    let $S'$ be the set of all $p' \in P'$ such that preds($p'$) $\subseteq C'$
    return(place($S'$ $-$ preds($C'$)))
end function
```

**Fig. 6.** Pseudo-code, continued.

token, in the worst case. At the same time, the number of cells which are requesting 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 workstatation) appears to increase quartically, as shown in part (b) of the figure. Finally, as we increase the number of cells in the ring, the number of reachable global markings increases exponentially, as shown in part (c) of the figure (on a logarithmic scale).[1] The number of states increases asymptotically by slightly less than a factor ten for each added cell.

How do these results compare to other methods for avoiding the state explosion problem? The trace theory approach of Dill [Dil88] required an abstract model of the arbiter cell to be created by hand. This reduces the state explosion problem, but does not entirely solve it, since even with the reduced model, the number of states still increases exponentially with the number of components. Probst [PL91] reports a method which requires quadratic space and time in the number of cells, but also is not fully automatic. The methods of Valmari [Val89, Val90] and Godefroid [God90, GW91]

[1] The number of reachable states was established using the symbolic model checking technique [BCM$^+$90]

[t]

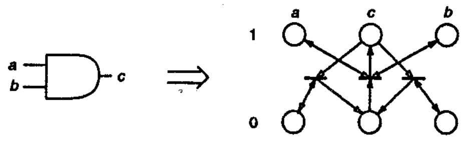

Fig. 7. Translation from circuit to net

and Yoneda [YTK91] cannot be effectively applied to this example or to other speed independent circuits, because in all states, all enabled transitions are in conflict with some disabled transition. Thus no transition can be statically guaranteed to be persistent. Experiments by Holger Schlingloff[2] have confirmed this to be the case. It is possible, perhaps, that some more clever static analysis technique could be used to show that some transitions are persistent, in which case these methods could be applied to some effect.

Finally, we consider the symbolic model checking technique [BCM+90]. For DME circuit, the basic symbolic model checking algorithm requires cubic time and linear space (in the number of cells). Burch and Long[3] have obtained $O(n^{2.5})$ time for the DME using symbolic model checking with a modified search order [BCL]. 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.

## 4 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 reachable terminal marking of an occurrence net is NP-complete. This is easily shown by reduction from 3-SAT.[4] 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

[2] Personal communication
[3] Personal communication
[4] Satisfiability of a Boolean formula in conjunctive normal form, with three literals in each conjunct.

[t]

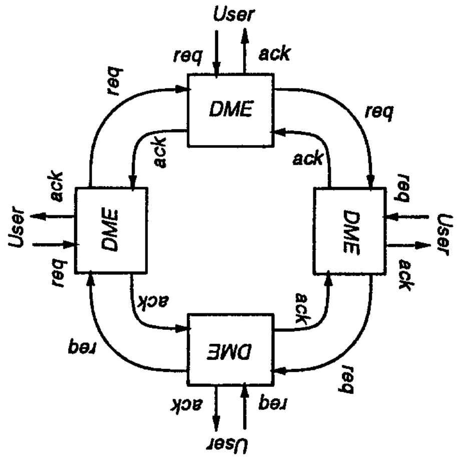

Fig. 8. Distributed mutual exclusion circuit

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 10 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 the 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 P-time reducible to reachability of a terminal marking of an unfolding. Since the configuration representing the terminal marking can be guessed in P-time in the size of the unfolding, and also tested in P-time, it follows that the problem is in NP, and hence 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 all configurations the unfolding can reach some configuration containing a cutoff point. This is simply because if there is no terminal marking, then all configurations can reach a configuration which is arbitrarily large. A configuration $C'$ can reach 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 some transition in the local configuration of $t'$. Such a transition $t'_1$ will be called a *spoiler* of $t'$.

There exists a configuration in conflict with all of the 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 algorithm of figure 11 uses branch and bound techniques to find such a set $T_s$ if one exists.

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. Of course, when backtracking occurs, the the net is also 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 path leads to a terminal marking of the original net $N$. This makes it straightforward to extract a path leading to a terminal marking.

Obviously, 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 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.

## 5 Evaluation

When is unfolding a suitable strategy for problems in automatic verification? The most promising application is hazard checking for asynchronous control circuits. In these circuits, the state explosion seems to derive almost entirely from arbitrary interleavings of concurrent transitions. In such cases, the unfolding method can have a considerable advantage over methods that search the entire state space. Note, however, that other methods based on partial orders are not necessarily effective in reducing the state explosion for these circuits, because of the aforementioned problem of determining when transitions of the net are persistent.

In general, any problem which can be posed in terms of coverability or deadlock in a Petri net model is a possible application of the unfolding method. In addition, it is possible that heuristically efficient procedures can be found for deciding the existence of an infinite firing path in some $\omega$-regular set, given an unfolding. In this case, specifications framed as linear time temporal logic formulas, or $\omega$-automata could be evaluated.

## References

[BCL] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic model checking with partitioned transition relations. To appear in the Proceedings of VLSI'91.

[BCM+90] 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.

[Dil88] D. Dill. Trace theory for automatic hierarchical verification of speed-independent circuits. Technical Report 88-119, Carnegie Mellon University, Computer Science Dept, 1988.

[God90] P. Godefroid. Using partial orders to improve automatic verification methods. In *Workshop on Computer Aided Verification*, 1990.

[GW91] P. Godefroid and P. Wolper. A partial approach to model checking. In *LICS*, 1991.

[Mar85] A. J. Martin. The design of a self-timed circuit for distributed mutual exclusion. In Henry Fuchs, editor, *1985 Chapel Hill Conference on VLSI*, pages 245–260. Computer Science Press, 1985.

[PL89] 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.

[PL90] D. K. Probst and H. F. Li. Using partial order semantics to avoid the state explosion problem in asynchronous systems. In *Workshop on Computer Aided Verification*, 1990.

[PL91] D. K. Probst and H. F. Li. Partial-order model checking: A guide for the perplexed. In *Third Workshop on Computer-aided Verification*, pages 405–416, July 1991.

[Sei80] C. L. Seitz. System timing. In Carver Mead and Lynn Conway, editors, *Introduction to VLSI Systems*, pages 218–262. Addison-Wesley, 1980.

[Val89] A. Valmari. Stubborn sets for reduced state space generation. In *10th Int. Conf. on Application and Theory of Petri Nets*, 1989.

[Val90] A. Valmari. A stubborn attack on the state explosion problem. In *Workshop on Computer Aided Verification*, 1990.

[YTK91] 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.

[t]

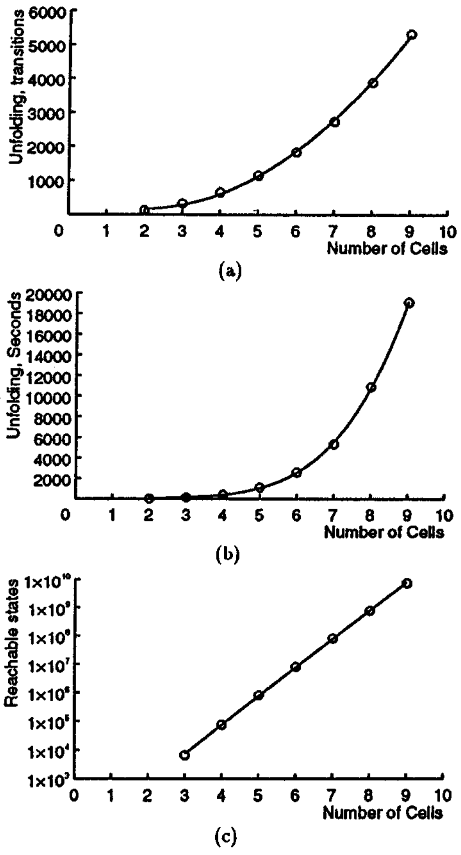

**Fig. 9.** Performance of unfolding method on hazard-detection problem for the distributed mutual exclusion circuit

[t]

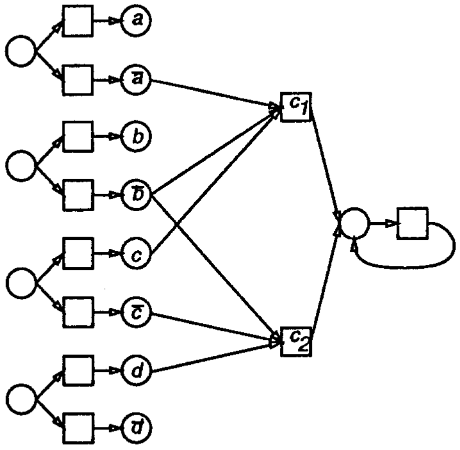

**Fig. 10.** Reduction from 3-SAT problem to a terminal marking problem.

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

**Fig. 11.** Procedure to detect terminal marking.