URL: http://www.elsevier.nl/locate/entcs/volume23.html 10 pages

Concurrent Symbolic Verification of Liveness Properties for Interleaved Models

Felice Balarin

Cadence Berkeley Laboratories, Berkeley, CA, USA

Abstract

Interleaved models of computations limit the number of system components that can change states simultaneously. This *interleaving constraint* often decreases efficiency of symbolic verification methods. It was shown previously that the constraint can be (possibly partially) removed while still preserving safety properties of systems. We propose two extensions of this approach to liveness properties. The first one does not require changes to existing verification algorithms, while the second (which is usually more efficient) does. Our experiments indicate that both approaches can drastically reduce verification time.

1 Introduction

Finite-state models can be divided into two classes: *interleaved* and *concurrent*. In interleaved models (sometimes also called asynchronous, or disjunctive), a transition of the system corresponds to a transition of a single, or a small number of related components [5,11,9]. On the other hand, in concurrent models (sometimes also called synchronous, or conjunctive, or simultaneous), a transition of the system corresponds to transitions of all, or an arbitrary number of components [7,12,3].

The two classes differ in approaches to their verification. Formal verification tools based on interleaved models typically employ explicit state space search [5,9], while those based on concurrent models often use symbolic search [12,3]. While not fundamental, this distinction has some practical justification. On one side, interleaved models are better for explicit search because they require enumerating enabled transitions, while concurrent models require enumerating all subsets of enabled transitions. On the other side, the *interleaving constraint*, i.e. the requirement that only a single component executes at a time, may cause artificial correlation between components, which may decrease the efficiency of the symbolic search [4,1].

Recently, there have been several attempts to improve symbolic search for interleaved models. Most of these improvements are either based on extending partial-order methods to symbolic search [1,10], or on changing the search

order to avoid the problem of artificial correlation [4]. In [2] we have analyzed a different approach, where a system in an interleaved model is analyzed as if its components were composed in a concurrent model. We have shown that safety properties of systems can be verified even if the interleaving constraint is (sometimes only partially) removed. We have also experimentally confirmed that removing the interleaving constraint increases the efficiency of the symbolic search.

In this paper, we extend the approach of [2] to liveness properties. We show that it is possible to partially remove the interleaving constraint while checking liveness properties with a standard verification algorithm. We demonstrate that this partial removal can significantly improve the verification efficiency. We also show that further improvements are possible if the verification algorithm is modified so that the interleaving constraint is removed only at certain steps of the algorithm.

In the rest of this paper, we first review relevant existing concepts and results in Section 2. We propose two approach to verification of liveness properties in Section 3. Some experimental results are presented in Section 4, and final comments are given in Section 5.

2 Preliminaries

2.1 Notation

Formal verification algorithms require manipulating sets of states and relations among states. In this paper, sets and relations are represented with *characteristic predicates* over state variables. We use bold-face capitals (e.g. $\mathbf{I}, \mathbf{R}, \mathbf{T} \ldots$) to denote predicates and lower-case bold-face letters to denote variables. Also, we use $\{\mathbf{S}\}$ to denote the set that some predicate $\mathbf{S}$ characterizes, i.e.:

$$\{\mathbf{S}\} \stackrel{def}{=} \{s \mid s \text{ satisfies } \mathbf{S}\} \ .$$

2.2 Model

Next, we present an extension to infinite executions of the interleaved model of computation proposed in [2]. In this formalism, a *system* is a finite collection of *processes*. Each process P_i is given by:

- the *present state variable* $\mathbf{p}_i$ and the *next state variable* $\mathbf{n}_i$ both ranging over some finite set of *local states*; we use $\mathbf{p}$ ($\mathbf{n}$) to denote the vector containing all present (next) state variables, and refer to their valuations as global states,
- the *initial states* predicate $\mathbf{I}_i$ in variable $\mathbf{p}_i$,
- the *transition relation* predicate $\mathbf{T}_i$ in variables $\mathbf{p}$ and $\mathbf{n}_i$.

Note that $\mathbf{T}_i$ can depend on all present state variables in $\mathbf{p}$, i.e. the processes communicate by observing states of other processes. Consequently,

elements of $\{\mathbf{T}_i\}$ are pairs (p, n), where p is a global state (itself a vector with one component for each process), and n is a local state of P_i.

An *execution* of the system is any infinite sequence $s_0, s_1, \ldots$ of global states[1] such that $s_{0,i} \in \{\mathbf{I}_i\}$ for all processes P_i, and for every $j > 0$ there exists a process P_i such that all components of s_{j-1} and s_j except the i-th are the same, and $(s_{j-1}, s_{j,i}) \in \{\mathbf{T}_i\}$.

Intuitively, an execution of the system proceeds in a series of steps, such that in every step a single process makes a transition. If s_{j-1} and s_j differ in the i-th component, then P_i is the process that makes the transition in the j-th step, and the change in the corresponding local state must be consistent with its transition relation.

With every system we associate set $\mathcal{F}$ of *fairness constraints*. Each fairness constraint is a pair of predicates over $\mathbf{p}$. An execution $s_0, s_1, \ldots$ is said to be *fair* if for every $(\mathbf{U}, \mathbf{V}) \in \mathcal{F}$ the set of states appearing in the execution infinitely often is either contained in $\{\mathbf{U}\}$ or intersects $\{\mathbf{V}\}$.

For example, if $\{\mathbf{U}\}$ contains all states in which some service is *not* requested, and $\{\mathbf{V}\}$ contains states in which that service is granted, the constraint $(\mathbf{U}, \mathbf{V})$ would require that the service is granted infinitely often if it is requested infinitely often.

Given a system consisting of processes P_i, we say that a *composition* of P_i's is any process that has vector $\mathbf{p}$ of the present state variables of P_i's as its present state variable, vector $\mathbf{n}$ as its next state variable, and the initial states predicate $\bigwedge_i \mathbf{I}_i$. In particular, we define the *interleaved composition* $P_{\|}$ to be the composition with the transition relation predicate:

$$\mathbf{T}_{\|} \;=\; \bigvee_i \left(\mathbf{T}_i \wedge \bigwedge_{j \neq i} (\mathbf{p}_j = \mathbf{n}_j) \right) \quad , \tag{1}$$

and the *concurrent composition* $P_{\times}$ to be the composition with the transition relation predicate:

$$\mathbf{T}_{\times} \;=\; \bigwedge_i \; (\mathbf{T}_i \vee (\mathbf{p}_i = \mathbf{n}_i)) \quad . \tag{2}$$

Intuitively, while $\mathbf{T}_{\|}$ requires that at every step exactly one process executes, $\mathbf{T}_{\times}$ allows any number of processes (including zero) to execute in any step. It is not hard to see that a collection of processes has the same executions as their interleaved composition.

2.3 Verification of liveness properties

First we need some definitions. Given some predicates $\mathbf{S}$ in variable $\mathbf{p}$ and $\mathbf{T}$ in variables $\mathbf{p}$ and $\mathbf{n}$, we use $Next(\mathbf{S}, \mathbf{T})$ to denote the predicate (in $\mathbf{p}$) that

[1] We use $s_{j,i}$ to denote the component of global state s_j corresponding to process P_i, i.e, we assume that $s_j = (s_{j,1}, s_{j,2}, \ldots, s_{j,n})$, where n is the number of processes in the system.

characterizes the set of states *reachable in one step by* $\mathbf{T}$ *from* $\mathbf{S}$. Formally:

$$\{Next(\mathbf{S}, \mathbf{T})\} \;=\; \{s \,|\, \exists q \in \{\mathbf{S}\} : (q, s) \in \{\mathbf{T}\}\} \;.$$

Similarly, we use $Reach(\mathbf{S}, \mathbf{T})$ to denote the predicate that characterizes the set of states *reachable by* $\mathbf{T}$ *from* $\mathbf{S}$ in any number of steps. Formally, $\{Reach(\mathbf{S}, \mathbf{T})\}$ is the smallest set satisfying:

$$\{Reach(\mathbf{S}, \mathbf{T})\} \;=\; \{\mathbf{S}\} \cup \{Next(Reach(\mathbf{S}, \mathbf{T}), \mathbf{T})\} \;.$$

Computing these predicates is a standard and crucial part of symbolic approaches to formal verification [12].

Now, we are ready to introduce a verification algorithm. Verification of many liveness properties can be reduced to checking for the existence of a fair execution. Probably the simplest algorithm for this check is as follows [8,6]:

Algorithm FAIR
1. $\mathbf{F} := Reach(\bigwedge_i \mathbf{I}_i, \mathbf{T}_{\|})$;
   repeat
2. $\mathbf{F} := \mathbf{F} \wedge Next(\mathbf{F}, \mathbf{T}_{\|})$;
   for each $(\mathbf{U}, \mathbf{V}) \in \mathcal{F}$
3. $\mathbf{F} := \mathbf{F} \wedge (\mathbf{U} \vee Reach(\mathbf{V} \wedge \mathbf{F}, \mathbf{T}_{\|}))$;
   rof
   until convergence

The set $\{\mathbf{F}\}$ is an increasingly better over-approximation of the set of *fair states*. We say that a state is fair if it appears infinitely often in some fair execution. In step 2 we remove from $\{\mathbf{F}\}$ all the states that cannot be reached from any other state in $\{\mathbf{F}\}$. Since every fair state must belong to a cycle of fair states, no fair states are eliminated in this step. In step 3 we remove from $\{\mathbf{F}\}$ states which are neither in $\{\mathbf{U}\}$ nor can be reached from any fair state in $\{\mathbf{V}\}$. It is not hard to check that no such state is fair. At the end, $\{\mathbf{F}\}$ is empty if and only if the set of fair states is.

2.4 Concurrent approach to verification of safety properties

Verifying safety properties of systems usually reduces to reachability analysis. It was observed in [4] that it is often cheaper to compute $Reach(\mathbf{S}, \mathbf{T}_{\times})$ than $Reach(\mathbf{S}, \mathbf{T}_{\|})$. However, only:

$$Reach(\mathbf{S}, \mathbf{T}_{\|}) \subseteq Reach(\mathbf{S}, \mathbf{T}_{\times})$$

holds in general, so replacing $\mathbf{T}_{\|}$ with $\mathbf{T}_{\times}$ may lead to incorrect verification results. In [2], we have shown that $Reach(\mathbf{S}, \mathbf{T}_{\|})$ is equal to $Reach(\mathbf{S}, \mathbf{T}_{\times})$ if $\mathbf{T}_{\times}$ is *serializable*, i.e. if for every transition (s, q) appearing in some execution of $P_{\times}$, there exists a sequence of states $s_0, \ldots, s_N$ such that:

- $s_0 = s$, $s_N = q$, $N \;=\; |\{P_i \,|\, s_i \neq q_i\}|$,
- there exists an ordering $P_{i_1}, \ldots, P_{i_N}$ of processes in $\{P_i \,|\, s_i \neq q_i\}$ such that for every $j = 1, \ldots, N$ all components of s_{j-1} and s_j except i_j-th are equal,

and $(s_{j-1}, s_{j,i_j}) \in \{\mathbf{T}_{i_j}\}$.

We say that $s_0, \ldots, s_N$ is a *serialization* of (s, q). Note that if some state v appears in a serialization of (s, q) then v must be *between* s and q, i.e. either $v_i = s_i$ or $v_i = q_i$ must hold for every i.

Even if $\mathbf{T}_\times$ is not serializable, it is always possible to modify it, yielding $\mathbf{T}'_\times$ such that $\mathbf{T}'_\times$ is serializable and $\{\mathbf{T}_\parallel\} \subseteq \{\mathbf{T}'_\times\} \subseteq \{\mathbf{T}_\times\}$. For such a modification, $Reach(\mathbf{S}, \mathbf{T}_\parallel) = Reach(\mathbf{S}, \mathbf{T}'_\times)$ holds [2]. Therefore, without a loss of generality, in the rest of this paper we will assume that $\mathbf{T}_\times$ is serializable.

3 Concurrent Approach to Verification of Liveness Properties

In this section we propose two approaches to verification of liveness properties for interleaved models. Both approaches are based on relaxing the interleaving constraint. The first approach (called *external*) does not require modifying existing verification algorithms, but it may require that the interleaving constraint is only partially relaxed. The second approach (called *internal*) is a modification of algorithm FAIR. In the proposed modification the interleaving constraint is fully removed but only in some steps of FAIR. In general, the internal approach is more efficient, but the external approach may be appropriate if an existing verification tool is available, but impossible (or hard) to modify.

3.1 External Approach

Ideally, $P_\times$ has fair executions if and only if $P_\parallel$ does. It is not hard to see that every (fair) execution of $P_\parallel$ is also a (fair) execution $P_\times$ [2]. For the desired relation to hold, we need to show that for every fair execution of $P_\times$ there exists a related fair execution of $P_\parallel$. Unfortunately, this is not generally true, even if $\mathbf{T}_\times$ is serializable.

Following the approach taken in [2] for the finite case, from a given fair execution r of $P_\times$ we may try to construct a fair execution r' of $P_\parallel$ by:

(i) removing from r all states s_j such that $(s_j, s_{j+1,i}) \notin \{\mathbf{T}_i\}$ for any process P_i, (Such cases are possible because $\mathbf{T}_\times$ allows that in a given step none of the processes make a transition. Obviously, $s_j = s_{j+1}$ must hold in this case, so these transitions are called *artificial self-loops*.)

(ii) replacing every remaining transition with its serialization.

This approach may not work for two reasons:

- after the first step there may be only finitely many states left (recall that we require all executions to be infinite),
- expansion in the second step may cause the execution to violate some fairness constraints.

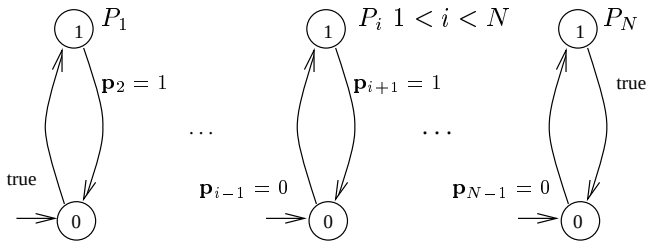

Fig. 1. A system with N processes.

Let us now examine the problem of artificial self-loops more closely. Consider the system shown in Figure 1. A typical process P_i starts in the initial state 0, moves to 1 if its left neighbor is in state 0, and then moves back to 0 if its right neighbor is in state 1. Boundary processes P_1 and P_N are similar, except that their state changes are unconditional, if their neighbors do not exist. Assume that the system in Figure 1 has a single fairness constraint with $\mathbf{U} \equiv false$ and:

$$\{\mathbf{V}\} \;=\; \{(1,0,1,0,1,0,\ldots)\} \;. \tag{3}$$

In other words, an execution is fair if it visits infinitely often the state in which all even numbered processes are in state 0 and all odd numbered processes are in state 1. Such a state is reachable in $P_{\|}$. However, if N is even, then that state is a deadlock state: odd numbered processes cannot move from 1 because their right neighbors are not in 1, and even numbered processes cannot move from 0 because their left neighbors are not in 0. Therefore, $P_{\|}$ has no fair executions. On the other hand, $P_{\times}$ does have a fair execution. It consists of a path to the state in $\{\mathbf{V}\}$ and then looping in it forever.

To solve this problem, we need to eliminate all artificial self-loops from $\{\mathbf{T}_{\times}\}$. This can be accomplished by replacing $\mathbf{T}_{\times}$ with:

$$\mathbf{T}_{\times} \wedge \bigvee_i \mathbf{T}_i \;. \tag{4}$$

The modified $\mathbf{T}_{\times}$ requires that at least one process makes a transition in any given step. In the special case when P_i's do not have any self-loops (which occurs often for interleaved models) (4) is equivalent to:

$$\mathbf{T}_{\times} \wedge \bigvee_i (\mathbf{p}_i \neq \mathbf{n}_i) \;. \tag{5}$$

Even though (4) and (5) are equivalent (assuming P_i's have no self-loops), computing (5) typically requires simpler intermediate results.

Let us now focus on the problem of fairness constraint violation. Again, consider the example in Figure 1, but this time assume that the single fairness constraint has $\mathbf{V} = false$ and:

$$\{\mathbf{U}\} \;=\; \{(s_1,\ldots,s_n) \,|\, \sum_{i=1}^{N} s_i \text{ is even } \} \;. \tag{6}$$

In other words, an execution is fair if the number of processes in state 1 is odd only finitely many times. Since in every step of $P_{\|}$ exactly one process moves

from or to state 1, it is obvious that $P_{\|}$ has no fair executions. However, $P_{\times}$ does have fair executions (even if $\mathbf{T}_{\times}$ is modified to eliminate artificial self-loops). For example, an infinite loop consisting of states (0,0,0,...) and (1,1,1,...) is fair. This discrepancy occurs because states occuring infinitely often in the execution of $P_{\times}$ are all contained in $\{\mathbf{U}\}$, but in every serialization of that execution some state outside of $\{\mathbf{U}\}$ appears infinitely often.

We can solve this problem by eliminating from $\{\mathbf{T}_{\times}\}$ all transitions (s,q) such that for some fairness constraint $(\mathbf{U},\mathbf{V})$ both s and q are in $\{\mathbf{U}\}$, but every serialization of (s,q) visits some state not in $\{\mathbf{U}\}$. This can be accomplished by replacing $\mathbf{T}_{\times}$ with:

$$\mathbf{T}_{\times} \wedge \bigwedge_{(\mathbf{U},\mathbf{V})\in\mathcal{F}} \neg\left(\mathbf{U} \wedge \mathbf{U_n} \wedge \exists \mathbf{q} : (\mathbf{B} \wedge \neg \mathbf{U_q})\right) \quad , \tag{7}$$

where $\mathbf{U_n}$ and $\mathbf{U_q}$ denote the predicates obtained from $\mathbf{U}$ by replacing every occurrence of $\mathbf{p}$ with $\mathbf{n}$ and $\mathbf{q}$ respectively, and $\{\mathbf{B}\}$ contains all states between $\mathbf{p}$ and $\mathbf{n}$. More precisely:

$$\mathbf{B} \equiv \bigwedge_i \left((\mathbf{q}_i = \mathbf{p}_i) \vee (\mathbf{q}_i = \mathbf{n}_i)\right) \quad .$$

Note that (7) eliminates more transitions than it is necessary. It eliminates all transitions (s,q) such that for some fairness constraint $(\mathbf{U},\mathbf{V})$ both s and q are in $\{\mathbf{U}\}$, but not *all* the states that *could* appear in a serialization of (s,q) are in $\{\mathbf{U}\}$. Eliminating more transitions may diminish the advantages of the concurrent approach. Still, we have chosen this approximation of the desired set of transitions, because computing better approximations appears to be significantly more expensive. Also note that even though (7) eliminates more transitions than it is necessary, it is still correct, i.e. it never eliminates any transition from $\{\mathbf{T}_{\|}\}$.

An important special case arises when all the predicates in fairness constraints depend on at most one variable. It is not hard to check that in that case (7) leaves $\mathbf{T}_{\times}$ unchanged. This special case is quite common: fairness constraints are typically just a union of component constraints, which depend only on local states.

If $\mathbf{T}_{\times}$ is modified as in (4) and (7), then $P_{\times}$ has fair executions if and only if $P_{\|}$ does. In most symbolic formal verification systems (4) and (7) can be implemented as additional "monitor" processes and do not require changing the internals of the program.

3.2 Internal approach

In this section we propose to remove the interleaving constraint in some steps of algorithm FAIR. Steps 1 and 3 of FAIR require reachability computation. Since $Reach(\mathbf{S},\mathbf{T}_{\times}) = Reach(\mathbf{S},\mathbf{T}_{\|})$ for any $\mathbf{S}$ (assuming $\mathbf{T}_{\times}$ is serializable), we can safely replace $\mathbf{T}_{\|}$ with $\mathbf{T}_{\times}$ in steps 1 and 3 of FAIR. We expect this modification to improve efficiency for the reasons which are explained in details

Table 1
Experimental results

	FAIR		external		internal	
N	size	time	size	time	size	time
100	890	1.46s	3307	0.14s	496	0.02s
200	1790	9.29s	6707	0.38s	996	0.02s
300	2690	29.26s	10107	0.84s	1496	0.03s
400	3590	70.02s	13507	1.50s	1996	0.04s
500	4490	156.78s	16907	2.21s	2496	0.04s
1000	8990	$\gg$	33907	9.01s	4996	1.37s
2000	17990	$\gg$	67907	40.56s	9996	5.52s

in [2]. In Section 4 we will show that these expectations are experimentally confirmed.

Replacing $\mathbf{T}_{\|}$ with $\mathbf{T}_{\times}$ in step 2 of algorithm FAIR may lead to incorrect verification results. Moreover, it is not likely that such a modification would result in any improvement. To reduce the number of iterations, it is useful that as many states as possible are eliminated from $\{\mathbf{F}\}$ in step 2. Since $\{\mathbf{T}_{\|}\} \subseteq \{\mathbf{T}_{\times}\}$, using $\mathbf{T}_{\times}$ instead of $\mathbf{T}_{\|}$ in step 2 may only result in fewer states being eliminated, and increasing number of iterations.

Finally, let us note that even though we consider only algorithm FAIR here, a similar analysis can be applied to other algorithms for checking for the existence of a fair execution.

4 Experiments

We have tested the proposed approaches on the example from Figure 1 with a single fairness constraint defined by (6) and (3). The results are summarized in Table 1. The first column shows the number of processes. Next two columns correspond to algorithm FAIR, followed by two columns corresponding to the external approach described in Section 3.1, and finally two columns corresponding to the internal approach described in Section 3.2. For each approach we give the run time and the size (in terms of the number of BDD nodes) of the relevant transition relation: $\mathbf{T}_{\|}$ for FAIR, $\mathbf{T}_{\times}$ for the internal approach, and $\mathbf{T}_{\times}$ modified according to (5) and (7) for the external approach. All experiments were performed using CUDD package by Somenzi [13].

The first thing to notice in Table 1 is that $\mathbf{T}_{\times}$ is smaller than $\mathbf{T}_{\|}$. However, if $\mathbf{T}_{\times}$ is modified as required by the external approach, then it becomes larger than $\mathbf{T}_{\|}$. Overall, these differences in sizes are relatively small, and all three transition relations grow only linearly with the number of processes.

The difference in run times is much more dramatic. Both the internal and the external approach outperform FAIR by several orders of magnitude, and the difference seems to be growing with the number of processes. Table 1 also shows that the internal approach significantly outperforms the external approach, but the precise relation is hard to determine, because for all but largest experiments the run times of the internal approach are too small to measure precisely.

5 Conclusions and Future Work

We have shown that relaxing the interleaving constraint may improve efficiency of symbolic verification of liveness properties. We have proposed two approaches to this end. One appears to be more efficient but the other has an advantage that it can be used with existing verification systems, without any internal modifications.

Future work should include more experiments to better understand when the proposed approaches are beneficial, and what are relative merits of each. Another interesting problem is extending these approaches to model checking for branching-time temporal logics.

References

[1] Rajeev Alur, R. K. Brayton, T. A. Henzinger, S. Qadeer, et al. Partial-order reduction in symbolic state space exploration. In O. Grumberg, editor, *Proceedings of Computer Aided Verification: 9th International Conference, CAV'97, Haifa, Israel, 22-25 June*. Springer, 1997. LNCS vol. 1254.

[2] Felice Balarin. Correctness of the concurrent approach to symbolic verification of interleaved models. In Alan Hu and Moshe Vardi, editors, *Proceedings of Computer Aided Verification: 10th International Conference, CAV'98, Vancouver, BC, Canada, June-July 1998*. Springer, 1998.

[3] R.K. Brayton, A. Sangiovanni-Vincentelli, G.D. Hachtel, F. Somenzi, A. Aziz, S.-T. Cheng, S. Edwards, S. Khatri, Y. Kukimoto, S. Qadeer, R.K. Ranjan, T.R. Shiple, G. Swamy, T. Villa, A. Pardo, and S. Sarwary. VIS: A system for verification and synthesis. In Rajeev Alur and Thomas A. Henzinger, editors, *Proceedings of Computer Aided Verification: 8th International Conference, CAV'96, Rutgers, NJ, July, 1996*. Springer, 1996. LNCS vol. 1102.

[4] J. R. Burch, Edmund M. Clarke, David E. Long, Ken L. McMillan, and David L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer–Aided Design of Integrated Circuits and Systems*, 13(4):401–24, April 1994.

[5] D. L. Dill, A. J. Drexler, A. J. Hu, and C. H. Yang. Protocol Verification as a Hardware Design Aid. In *Proceedings of ICCD*, pages 522–525, October 1992.

[6] E. Allan Emerson and C. L. Lei. Efficient model checking in fragments of the propositional mu-calculus (extended abstract). In *Proceedings, Symposium on Logic in Computer Science*, pages 267–278, Cambridge, Massachusetts, 16–18 June 1986. IEEE Computer Society.

[7] Z. Har'El and R. P. Kurshan. Software for analysis of coordination. In *Proceedings of the International Conference on System Science*, pages 382–385, 1988.

[8] Ramin Hojati and Robert K. Brayton. An environment for formal verification based on symbolic computation. *Formal Methods in System Design: An International Journal*, 6(2):191–216, March 1995.

[9] Gerard J. Holzmann. *Design and validation of computer protocols.* Englewood Cliffs, N.J. : Prentice Hall, 1991.

[10] R. P. Kurshan, V. Levin, M. Minea, D. Peled, et al. Verifying hardware in its software context. In *Digest of Technical Papers of the 1997 IEEE International Conference on CAD*, pages 742–9, November 1997.

[11] Zohar Manna and Anir Pnueli. Verification of concurrent programs: The temporal framework. In R. Boyer and J. Moore, editors, *Correctness Problem in Computer Science*, pages 215–273. Academic Press, 1981.

[12] Kenneth L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.

[13] F. Somenzi. CUDD : CU Decision Diagram Package, June 1996. User's Manual.