Symbolic Model Checking with Partitioned Transition Relations

J. R. Burch E. M. Clarke D. E. Long

October 1991
CMU-CS-91-195

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Winner of the Sidney Michaelson Best Paper Award at VLSI 91, Edinburgh, Scotland.

This research was sponsored in part by the Avionics Laboratory, Wright Research and Development Center, Aeronautical Systems Division (AFSC), U.S. Air Force, Wright-Patterson AFB, Ohio 45433-6543 under Contract F33615-90-C-1465, ARPA Order No. 7597 and in part by the National Science Foundation under Contract No. CCR-9005992.

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the NSF or the U.S. government.

Keywords: Logic design verification, Binary decision diagrams, Partitioned transition relations, Asynchronous circuits, Breadth first search

Abstract

We significantly reduce the complexity of BDD-based symbolic verification by using *partitioned transition relations* to represent state transition graphs. This method can be applied to both synchronous and asynchronous circuits. The times necessary to verify a synchronous pipeline and an asynchronous stack are both bounded by a low polynomial in the size of the circuit. We were able to handle stacks with over 10^{50} reachable states and pipelines with over 10^{120} reachable states.

1 Introduction

Although methods for verifying sequential circuits by searching their state transition graphs have been investigated for many years, it is only recently that such methods have begun to seem practical. Before, the largest circuits that could be verified had about 10^6 states. Now it is easy to check circuits that have many orders of magnitude more states [3, 5, 6, 7]. The reason for the dramatic increase is the use of special data structures such as binary decision diagrams (BDDs) [2] for encoding the state transition graphs of such systems.

In this paper, we show how to process state transition graphs more efficiently than in our previous work [5, 6]. Our new approach involves using multiple BDDs, which are implicitly conjuncted or disjuncted, to represent the graphs. We call this kind of representation a *partitioned transition relation*. The BDDs that make up the partitioned transition relation are derived in a natural way from the structure of the circuit being verified. We illustrate the power of the technique by verifying an asynchronous stack [10] and a synchronous pipeline circuit [5]. Using a partitioned transition relation, we were able to verify a stack 32 bits wide and 2 cells deep. For comparison, we were unable to verify a stack only 1 bit wide and 1 cell deep when using a single BDD to represent the transition relation because the transition relation required more than 350,000 BDD nodes. For a pipeline with 4 registers, each 32 bits wide, the partitioned transition relation required less than 2,500 BDD nodes, while using a single BDD required nearly 340,000 nodes, a savings of nearly a factor of 140. On a Sun 4, the verification time improved from approximately 14,000 seconds (projected) to 995 seconds, a factor of about 14. We were also able to handle example pipelines with over 10^{120} reachable states.

There are several other methods that use BDDs in the verification of sequential circuits. Bryant and Seger [3] use a symbolic switch-level simulator to check pre- and post-conditions specified in a restricted form of temporal logic. The logic allows boolean conjunction and the next time modality ($\mathbf{X}$). Coudert, Berthet, and Madre describe a system for showing equivalence between deterministic finite automata [7]. Their system performs a symbolic breadth-first search of the state space reachable by the product of the two automata. None of these methods can easily handle nondeterministic systems. With transition relations, it is very natural to model examples like

the cache coherency protocol for the Encore Gigamax, which McMillan has recently investigated [11]. A major feature of the Gigamax architecture is an asynchronous, and hence nondeterministic, interconnection network. The use of abstraction to hide certain details of the cache replacement policy also gives rise to nondeterminism in this example.

2 Symbolic verification

Given a circuit, let V be its set of boolean state variables. We identify a boolean formula over V with the set of valuations which make the formula true. A valuation of the variables corresponds in a natural way to a state of the circuit; hence the formula may be thought of as representing a set of circuit states. The BDD for the formula is in practice a concise representation for this set of states. In the remainder of the paper, we will denote sets of states using S and T. We denote the BDD representing the set S by $S(V)$, where V is the set of variables that the BDD depends on. In addition to representing sets of states of a circuit, we must represent the transitions that the circuit can make. To do this, we use a second set of variables V'. A valuation for the variables in V and V' can be viewed as designating a pair of states in the circuit, and we can represent sets of pairs using BDDs as above. We will refer to sets of pairs of states as transition relations. If N is a transition relation, then we write $N(V, V')$ to denote the BDD that represents it.

There are many finite state verification methods that can make effective use of this representation [5, 7]. For our purposes, the important property of these algorithms is that the basic step is performing computations of the following form:

$$S'(V') = \exists_{v \in V} \left[S(V) \wedge N(V, V') \right].$$

(The notation above indicates a series of nested existential quantifications, one for each variable in V.) This expression, called a relational product, gives the set of states S' reachable in one step from the set of states S in a circuit with transition relation N. It is crucial to be able to do this computation efficiently. A special algorithm is typically used to do this operation in one pass over the BDDs $S(V)$ and $N(V, V')$. By using such an algorithm, it is possible to avoid building the BDD for $S(V) \wedge N(V, V')$, which would often

be impractically large. Unfortunately, the BDD $N(V, V')$ itself is often very big. Up to this point, being forced to construct this BDD has been the major stumbling block in trying to verify complex circuits. In the following sections, we describe how to overcome this problem by using a partitioned transition relation to represent N.

3 Deriving transition relations

The first step in verifying a circuit is to derive its transition relation. Our goal is to reflect the structure of the circuit in the structure of the transition relation, so that the transition relation can be stored and manipulated more efficiently.

For a synchronous circuit with n state variables, we let $V = \{v_0, \ldots, v_{n-1}\}$ and $V' = \{v'_0, \ldots, v'_{n-1}\}$. For each state variable v_i, there is a piece of combinational logic which determines how it is updated. Let f_i be the function computed by this logic. Then the value of v_i in the next state is given by

$$v'_i = f_i(V).$$

These equations are used to define the relations

$$N_i(V, V') = (v'_i \Leftrightarrow f_i(V)).$$

In a legal transition of the circuit, each N_i must be true; hence the transition relation for the circuit is

$$N(V, V') = N_0(V, V') \wedge \cdots \wedge N_{n-1}(V, V').$$

Thus, the transition relation for a synchronous circuit can be expressed as a conjunction of relations.

In practice, each N_i can often be represented by a small BDD (typically fewer than 100 nodes). However, the size of the BDD representing the entire transition relation may grow as the product of the sizes of the individual parts, and thus may be prohibitively large. In the past, this has been the major limitation of symbolic model checking. For our new method, we instead represent the transition relation by a list of the parts, which are implicitly conjuncted. We call this representation a conjunctive partitioned transition relation.

Asynchronous circuits can be modeled with a conjunctive partitioned transition relation, like synchronous circuits, and can also be represented by a disjunctive partioned transition relation. To simplify the description of how these forms of transition relation are computed, we assume that all the components of the circuit have exactly one output, and have no internal state variables. It is straightforward to generalize the method to handle cases where this assumption does not hold.

In asynchronous circuits, there can be an arbitrary delay between when a transition is enabled and when it actually occurs. We can model this by allowing each component to nondeterministically choose whether to transition its output, resulting in a conjunctive partitioned relation with n parts, all of the form

$$N_i(V, V') = (v_i' \Leftrightarrow f_i(V)) \vee (v_i' \Leftrightarrow v_i).$$

For some components, such as C-elements and flip-flops, the function $f_i(V)$ may depend on the current value of the output of the component, as well as the inputs.

The above model for asynchronous circuits allows wires to transition concurrently. We can also use an interleaving model, which allows only one wire to transition at a time. This idea can be used to construct a disjunctive partitioned transition relation, as follows. First, apply distributivity to the conjunction of the R_i, giving a disjunction of 2^n terms. Each of these terms corresponds to the simultaneous transitioning of some subset of the n wires in the circuit. Second, keep only those terms that correspond to exactly one wire transitioning. This results in a disjunction of the form

$$N(V, V') = N_0(V, V') \vee \cdots \vee N_{n-1}(V, V')$$

where

$$N_i(V, V') = (v_i' \Leftrightarrow f_i(V)) \wedge \bigwedge_{j \neq i} (v_j' \Leftrightarrow v_j).$$

We represent the full transition relation as a list of the $N_i(V, V')$, which are implicitly disjuncted.

4 Computing relational products

As noted earlier, computing relational products is a fundamental operation in many symbolic verification methods. This section describes how relational

products can be computed using the representations described in the previous section. These techniques significantly increase the size of circuits that can be verified compared to previous methods.

For a disjunctive partitioned transition relation, the relational product computed is of the form

$$S'(V') = \exists_{v \in V} \Big[S(V) \wedge (N_0(V, V') \vee \cdots \vee N_{n-1}(V, V')) \Big].$$

This relational product can be computed without ever constructing the BDD for the full transition relation by rewriting $S'(V')$,

$$S'(V') = \exists_{v \in V} \Big[S(V) \wedge N_0(V, V') \Big] \vee \cdots \vee \exists_{v \in V} \Big[S(V) \wedge N_{n-1}(V, V') \Big].$$

Thus, we are able to reduce the problem of computing $S'(V')$ to one of computing a series of relational products involving relatively small BDDs. This technique was used previously for verifying asynchronous circuits [5]. Much larger asynchronous circuits could be verified using this method than with a monolithic transition relation.

For a conjunctive partitioned transition relation, the relational product computed is of the form

$$S'(V') = \exists_{v \in V} \Big[S(V) \wedge (N_0(V, V') \wedge \cdots \wedge N_{n-1}(V, V')) \Big]. \tag{1}$$

The main difficulty in computing $S'(V')$ without building the conjunction is that conjunction does not distribute over existential quantification. The method given below overcomes this difficulty.

Our new technique is based on two observations. First, circuits exhibit locality, so many of the $N_i(V, V')$ will depend on only a small number of the variables in V and V'. Second, although conjunction does not distribute over existential quantification, subformulas can be moved out of the scope of existential quantification if they do not depend on any of the variables being quantified. We will take advantage of these observations by conjuncting the $N_i(V, V')$ with $S(V)$ one at a time and quantifying out each variable v when none of the remaining $N_i(V, V')$ depend on v. More formally, the user must choose a permutation ρ of $\{0, \ldots, n-1\}$. This permutation determines the

order in which the $N_i(V, V')$ are conjuncted. For each i, let D_i be the set of variables in V that $N_i(V, V')$ depends on. Also, let

$$E_i = D_{\rho(i)} - \bigcup_{k=i+1}^{n-1} D_{\rho(k)}.$$

Thus, E_i is the set of variables contained in $D_{\rho(i)}$ that are not contained in $D_{\rho(k)}$ for any k larger than i. The E_i are pairwise disjoint and their union is equal to V. The relational product in equation 1 can be computed as

$$\begin{aligned}
S_1(V, V') &= \exists_{v \in E_0} \left[S(V) \wedge N_{\rho(0)}(V, V') \right] \\
S_2(V, V') &= \exists_{v \in E_1} \left[S_1(V, V') \wedge N_{\rho(1)}(V, V') \right] \\
&\vdots \\
S'(V') &= \exists_{v \in E_{n-1}} \left[S_{n-1}(V, V') \wedge N_{\rho(n-1)}(V, V') \right].
\end{aligned}$$

The ordering ρ has a significant impact on how early in the computation state variables can be quantified out. This affects the size of the BDDs constructed and the efficiency of the verification procedure. Thus, it is important to choose ρ carefully, just as with the BDD variable ordering. In practice, we have found it fairly easy to come up with orderings which give good results.

In the previous section, we described how a circuit could be represented by a set of $N_i(V, V')$, each depending on exactly one variable in V'. While this is almost always more efficient than constructing the full transition relation, it may not be the best choice. As long as the BDDs do not get too large, it is better to combine several of the $N_i(V, V')$ into one BDD by forming their disjunction or conjunction.

5 Verifying asynchronous circuits

Asynchronous circuits can be verified in two steps. First, compute the set of states the circuit, composed with an environment, can reach from a given set of initial states. Then check that no hazard can occur in any of the reachable states. Finding the reachable states is the most computationally

expensive of these two steps. In practice, checking for hazards is usually done as the reachable states are computed. This is similar to Dill's [9] method for verifying safety properties of asynchronous circuits.

The set of reachable states is found by computing the least fixed point S of

$$S(V') = S_0(V') \vee \exists_{v \in V} \left[S(V) \wedge N(V, V') \right],$$

where S_0 is the initial set of states and N is the transition relation of the circuit. We use frontier set simplification to speed up the computation of this fixed point [5, 7]

There are significant differences in the complexity of doing reachability analysis using conjunctive and disjunctive partitioned transition relations. Consider two uncoupled systems M' and M'' with disjoint sets of state variables V' and V''. Let M be the composition of these two systems. This is an unrealistic example, but it helps illustrate what happens when computing the reachable states of loosely coupled systems. The BDD $S(V)$ representing the set of reachable states of M is equal to $S'(V') \wedge S''(V'')$, where $S'(V')$ ($S''(V'')$) is the BDD for the reachable states of M' (M''), and $V = V' \cup V''$. An efficient way to order the BDD variables of the combined system in this case is to have all the variables of one component (say M') before any of the variables in the other component. Then the number of BDD nodes in $S(V)$ is equal to the sum of the nodes in $S'(V')$ and $S''(V'')$, independent of whether conjunctive or disjunctive partitioning is used. However, the sizes of the BDDs representing the intermediate state sets are potentially different for the two methods.

Let $S_i(V)$, $S'_i(V')$ and $S''_i(V'')$ be the BDDs representing the states reachable in i steps by M, M' and M'', respectively, using non-interleaved semantics. Similarly, let $T_i(V)$, $T'_i(V')$ and $T''_i(V'')$ be the BDDs representing the states reachable in i steps by M, M' and M'', respectively, using interleaved semantics. In the conjunctive case, $S_i(V) = S'_i(V') \wedge S''_i(V'')$, so the size of each $S_i(V)$ is equal to the sum of the sizes of $S'_i(V')$ and $S''_i(V'')$, just as for the set of reachable states. However, for the disjunctive case,

$$T_i(V) = \bigvee_{k=0}^{i} T'_k(V') \wedge T''_{i-k}(V'').$$

Thus, interleaving semantics introduces an artificial correlation between the local states of M' and M'' in the $T_i(V)$. The $T_i(V)$ are generally much larger

than the $S_i(V)$, since each $T_i(V)$ must contain $T''_k(V)$ for all $k \leq i$. Because of this effect, reachability analysis with disjunctive partitioning is less efficient than with conjunctive partitioning.

We can make disjunctive partitioning more efficient by modifying the breadth first search used for reachability analysis. To search the reachable states of M, first compute states reachable by transitions of wires in M'. Then compute the states reachable from that set by transitioning on wires in M''. This is equal to the global reachable state set, since M' and M'' are uncoupled. Separately computing local fixed points for the two parts of the system in this way removes the artificial correlation described above. In general, for a circuit C divided into loosely coupled subcircuits C_j, we compute the reachable states of C by repeatedly computing local fixed points for each C_j until a global fixed point is reached. This idea can be extended to a hierarchy with any number of levels.

6 An asynchronous stack

In this section, we compare conjunctive and disjunctive partitioned transition relations for verifying asynchronous circuits by considering an asynchronous lazy stack due to Martin [10]. To determine the asymptotic performance of the various methods discussed earlier, we performed a reachability analysis for stacks with varying depth d and word width w. This is sufficient to determine the asymptotic complexity of verification, even though we did not check for hazards. Hazard checking increases the times by a constant factor.

The stack consists of an array of d cells, each cell consisting of a control part, a data part and a completion tree. The data part of each cell consists of w storage elements. The completion trees signal when all the storage elements in a cell have completed the current data transfer.

The verification system that we use is written in a combination of C and LISP. The BDD package is written in C and is roughly comparable in performance to the package described by Brace, Rudell and Bryant [1].

We studied how verification time varied with w for four different methods:

1. Disjunctive partitioning using standard breadth first search. We combined the transition relations for the gates making up each individual control part, each of the individual storage elements, and each completion tree.

2. Disjunctive partitioning using modified breadth first search and the same partitioning of the transition relation as above. At the top level, the hierarchy used for local fixed point computation consisted of the environment and each cell as a unit. Each cell was broken into the control part, the completion tree and the data part. The data part was further subdivided into $\lceil \lg(w) \rceil$ levels, each of two parts.

3. Conjunctive partitioning using the same partitioning of the transition relation as above. We used the following ordering ρ of the parts of transition relation: the environment at the top of the stack; the control part and data parts of each cell, ordered from the top of the stack to the bottom; the completion trees, also ordered from the top of the stack to the bottom; and the environment at the bottom of the stack.

4. Conjunctive partitioning using the same partitioning as above, but with the control and data parts within each cell combined into one BDD. The ρ used above is modified in the obvious way.

In all cases, we used an initial state set in which each cell could be full or empty and the data in each cell was arbitrary. Using a more restricted set of initial states, such as having all cells initially empty, can increase the verification time by as much as a factor of d.

A graph of the search times versus stack width for the various methods is shown in figure 1. We found that disjunctive partitioning with breadth first search were feasible only for small examples. Disjunctive partitioning with modified breadth first search and conjunctive partitioning were all much more efficient. Search times using methods 2 and 3 grew slightly faster than quadratically. Method 4 gave a growth rate of roughly $w^{1.5}$. Using this method, we were able to find the reachable states of a 32 bit wide, depth 2 stack in under an hour of CPU time on a Sun 4. This circuit had over 989 boolean state variables and 10^{50} reachable states.

The BDDs in the transition relation are all of constant size, except for those representing the completion trees. These BDDs are growing as $w^{\lg 3}$, but for the values of w we considered, they are still quite small. For larger w, it might be necessary to split the completion trees into more than one BDD.

We also explored how the search time varied with the depth of the stack, using conjunctive partitioning. The number of steps needed to compute the reachable states grows quadratically in d. The states which require the largest

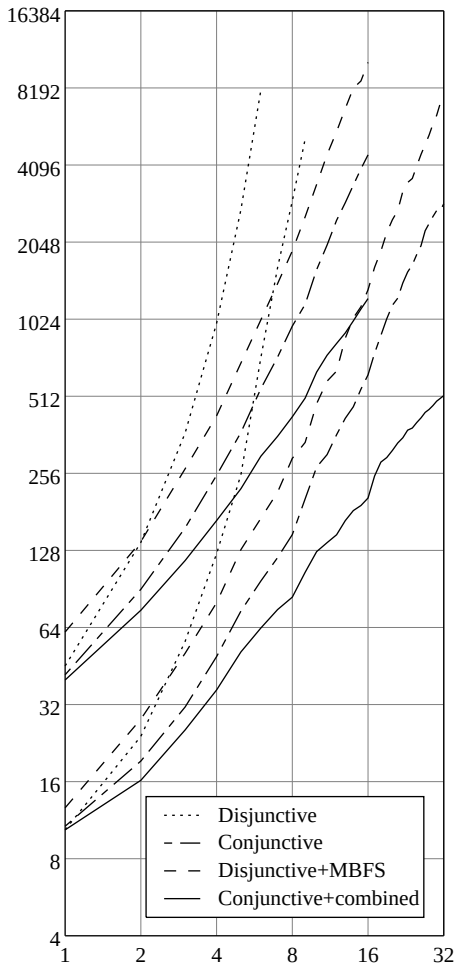

Figure 1: Search times in seconds for stacks of various widths, with $d = 1, 2$

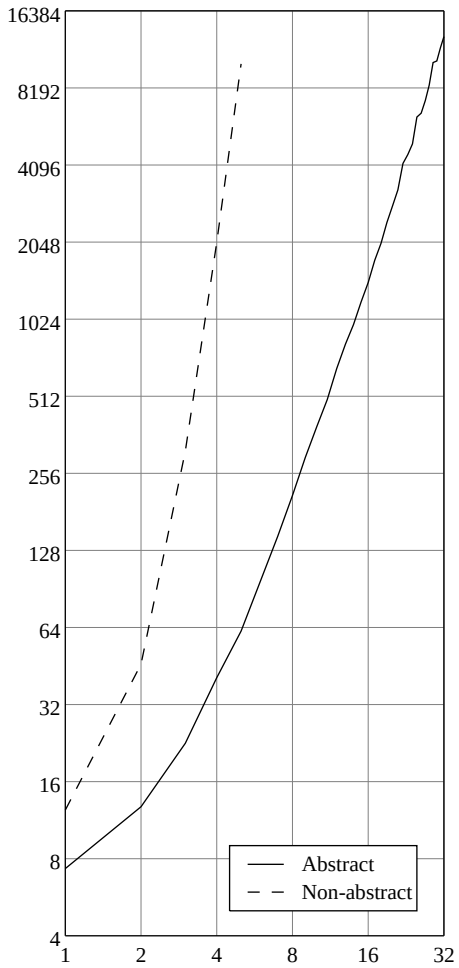

Figure 2: Search times in seconds for stacks of various depths, with $w = 1$

number of steps to reach are states in which internal signals within the stack control are not stable. Thus, we were able to avoid the quadratic search depth by replacing the control part of each cell by an abstract model having only external signals. We separately verified that the abstract model correctly describes the external behavior of the control part. With this abstraction, the number of steps needed to find all reachable states is linear in d, and the search time is cubic in d (see figure 2).

Although this kind of abstraction can greatly improve the efficiency of verifiers that explicitly enumerate states, it is usually not nearly as helpful when used with symbolic verifiers. For example, the search times for stacks of depth one improve only about 20 percent when the abstract model of the control part is used. The effect that abstraction has on the search depth, as described above, is an exception to this rule.

7 A synchronous pipeline

We also considered the verification of a synchronous pipeline circuit. This circuit, described in an earlier paper [5], performs three-address arithmetic and logical operations on operands stored in a register file. We experimented with a number of versions of the pipeline with varying numbers of registers, register widths, numbers of pipe stages, and numbers of operations. The verification times grow as low polynomials in all dimensions. We also ran several more realistic examples. The largest of these was a pipeline with 8 registers, each 32 bits wide, 2 pipe registers, and one operation. This example had 406 state variables resulting in more than 10^{120} reachable states, and the verification took 4 hours and 20 minutes of CPU time on a Sun 4. Details of the verification of the pipeline can be found in [4].

8 Discussion and future research

Using partitioned transition relations significantly improves the efficiency of symbolic verification. We verified a stack with over 950 state variables and more than 10^{50} reachable states and a pipeline with more than 400 state variables and over 10^{120} reachable states. We also studied the asymptotic performance of our verification methods. This kind of asymptotic analysis is an important way to compare different techniques.

For deterministic systems, a transition function vector can be used to represent how a circuit transitions from one state to another. In this method, a separate BDD is used for each state holding node of the system. This BDD represents the function computed by the combinational logic driving the associated node. Coudert *et al.* [7, 8] describe a number of algorithms for manipulating transition functions. They note that the monolithic transition relation can require many more BDD nodes than the corresponding transition function vector [8]. However, they report that computations with transition relations are faster than those using transition functions. Partitioned transition relations provide the speed of transition relations and the memory efficiency of transition functions.

Touati *et al.* [12] proposed another method for representing transition relations as implicit conjunctions. They use the *constrain* operator of Coudert *et al.* [7] to eliminate the state set $S(V)$ in equation 1. Then they compute the resulting conjunction as a balanced binary tree, quantifying out each variable in V when all the BDDs depending on that variable have been combined. We believe that this method is inferior to the one proposed here because the constrain operator may introduce dependencies on any of the variables in $S(V)$. This makes it impossible to compute in advance a schedule for quantifying out the variables in V, which in turn reduces the practicality of caching results between relational product computations. In addition, if $S(V)$ depends on most of the variables in V, it may not be possible to quantify out many variables before performing the final conjunction. They also suggest having one transition relation per state variable. In our experience, it is often better to combine parts of the transition relations to reduce overhead; this idea is also applicable to their method. We implemented their method and tested it on some of the examples in section 7. For a pipeline with four 8 bit registers, one pipe register and one operation, our method was more than five times faster. In addition, for some of the relational product computations, the intermediate BDDs using their method were more than an order of magnitude larger than the final result.

References

[1] K. S. Brace, R. L. Rudell, and R. E. Bryant. Efficient implementation of a BDD package. In *27th ACM/IEEE Design Automation Conference*, 1990.

[2] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comput.*, C-35(8), 1986.

[3] R. E. Bryant and C.-J. Seger. Formal verification of digital circuits using symbolic ternary system models. In R. Kurshan and E. M. Clarke, editors, *Workshop on Computer-Aided Verification*. DIMACS, 1990.

[4] J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in symbolic model checking. In *28th ACM/IEEE Design Automation Conference*, 1991.

[5] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. L. Dill. Sequential circuit verification using symbolic model checking. In *27th ACM/IEEE Design Automation Conference*, 1990.

[6] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *LICS*, 1990.

[7] O. Coudert, C. Berthet, and J. C. Madre. Verification of synchronous sequential machines based on symbolic execution. In J. Sifakis, editor, *Automatic Verification Methods for Finite State Systems*, volume 407 of *LNCS*. Springer-Verlag, 1989.

[8] O. Coudert, J. C. Madre, and C. Berthet. Verifying temporal properties of sequential machines without building their state diagrams. In R. Kurshan and E. M. Clarke, editors, *Workshop on Computer-Aided Verification*. DIMACS, 1990.

[9] D. L. Dill. *Trace Theory for Automatic Hierarchical Verification of Speed-Independent Circuits*. ACM Distinguished Dissertations. MIT Press, 1989.

[10] A. J. Martin. A synthesis method for self-timed VLSI circuits. In *Proceedings: IEEE International Conference on Computer Design*, 1987.

[11] K. L. McMillan and J. Schwalbe. Formal verification of the Gigamax cache consistency protocol. In *International Symposium on Shared Memory Multiprocessing*, 1991.

[12] H. J. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit state enumeration of finite state machines using BDD's. In *ICCAD*, 1990.