Model Checking Based on Sequential ATPG

Vamsi Boppana, Sreeranga P. Rajan, Koichiro Takayama, and Masahiro Fujita

Fujitsu Laboratories of America, Inc., 595 Lawrence Expressway, Sunnyvale, CA 94086,

{vboppana,sree,ktakayam,fujita}@fla.fujitsu.com

Abstract. State-space explosion remains to be a significant challenge for Finite State Machine (FSM) exploration techniques in model checking and sequential verification. In this work, we study the use of sequential ATPG (Automatic Test-Pattern Generation) as a solution to overcome the problem for a useful class of temporal logic properties. We also develop techniques to exploit the existence of synchronizing sequences to reduce some temporal logic properties to simpler properties that can be efficiently checked using an ATPG algorithm . We show that the method has the potential to scale up to large, industrial-strength, hardware designs for which current model checking techniques fail.

1 Introduction

The state-space explosion problem that challenges Finite State Machine (FSM) exploration techniques such as CTL temporal logic model checking [McM93] for automatic formal verification has been intensively studied from various angles. There have been numerous efforts to tackle the state-space explosion problem [CGL94]. Techniques such as compact data structures to represent the state-space [Bry95], on-the-fly model checking [Pel96], state-space reduction techniques such as localization reduction [Kur94], and navigated model checking [TSNH98] have improved the applicability of model checking towards increasingly large designs.

However, past efforts in alleviating the state-space explosion problem fall short of making model checking scale up for efficient automatic verification of current, industrial, hardware designs. Current model checking techniques could fail in several ways including failure to extract state-transition relation information from the design structure and requiring excessive storage for functional representations of the state-space during computation.

In this work, we study the use of sequential ATPG (Automatic Test-Pattern Generation) algorithms [ABF90] for model checking a simple class of CTL formulae. The approach involves the construction, based on the CTL formula, of a new circuit structure from the circuit to be verified. Model checking is then cast into *detecting* a *stuck-at-fault* on the output line of the constructed circuit. The method avoids building elaborate

N. Halbwachs and D. Peled (Eds.): CAV'99, LNCS 1633, pp. 418–430, 1999.
© Springer-Verlag Berlin Heidelberg 1999

functional information such as a complete state transition relation and benefits from a directed, on-the-fly, structural exploration of the design under verification. Experiments are performed on benchmark circuits with simple formulae of the form **AG EF** P to study the efficiency of state space exploration with a state-of-the-art sequential ATPG. Furthermore, we show reductions of the form **AG EF** P to **EF** P based on verifying the existence of *synchronizing* sequences [Koh78,Hen68], whose application causes an FSM to reach a specific state regardless of the starting state. The motivation for these reductions stems from the fact that most machines are synchronizable (atleast partially/weakly) with designer-supplied/ATPG-generated sequences and because they enable efficient ATPG problem formulations. The reduction results have been verified and incorporated into the PVS [ORR$^+$96] proof checker.

The rest of the paper is organized as follows: Section 2 discusses the reduction of simple CTL formulae for synchronizable machines. A discussion on sequential ATPG algorithms and their application in state-space exploration is given in Section 3. The method of transforming model checking simple CTL formulae to stuck-at-fault testing is explained in Section 4. Experimental results are discussed in Section 5. Finally, conclusions are summarized in Section 6.

2 Formula Reduction for Synchronizable FSMs

Synchronizability of a FSM is used to reduce CTL formulae of the form **AG EF** P to **EF** P. A formal definition of synchronizable machines is given in Section 2.1 followed by an example illustrating the reduction in Section 2.2. The formalization of the reduction of CTL formulae and their proofs in the PVS proof checker are explained in Section 2.3 and Section 2.4 respectively.

2.1 Synchronizable FSMs

Definition 1 (Synchronizability [Koh78,Hen68]) *A machine M is synchronizable, if there exists an input sequence Y, that takes M to a specified final state, regardless of the output or the initial state.*

Definition 2 (Initializability [CA89,CA93]) *A machine M is initializable with three-valued logic simulation if there exists an input sequence Y, such that the resulting state of M (evaluated by three-valued simulation) is fully specified on the application of Y, when the initial state is fully unspecified (consisting of all Xs and corresponding to the entire state space). Initializability, is thus synchronizability subject to three-valued logic simulation.*

It is important to note that verifying that a given sequence of test vectors is an initializing sequence is a much simpler task than verifying if the sequence is a synchronizing sequence. The reason is that while checking for initializing sequences can be done on the structure of the circuit (using 3-valued logic simulation on the netlist), checking for synchronizing sequences may often require some form of knowledge and representation of the state space. For large, industrial designs, the only feasible checks possible are based on initializing sequences.

Table 1. Example FSM

PS ↓	NS x=0	NS x=1
A	B	D
B	A	B
C	D	A
D	D	C

2.2 Basic Idea and Example

Consider the FSM shown in Table 1 [Koh78]. The machine has four states A, B, C and D, and one input x. The first column in the table represents the present state, the next two columns represent the next states reached up on the application of input x.

It is clear that the machine has a synchronizing sequence 01010; this sequence, when applied to the FSM, synchronizes the machine to state D, regardless of the output or the initial state. Consider the property $D \models AGEF(C)$. This property can now be reduced to $D \models EF(C)$ based on verifying that there is a synchronizing sequence to the state D (in this example, the sequence 01010 achieves the objective).

An intuitive explanation of the savings possible from this method is presented below. Consider any arbitrary state that is reachable from D, say B for illustration. Transferring the machine from state B to state C can be performed in the two distinct steps of transferring the machine from state B to the synchronized state D followed by transferring the machine from state D to state C. Symbolically, this can be represented by the following:

$$B \stackrel{EF}{\longrightarrow} C \Leftarrow (B \stackrel{Syn.seq.}{\longrightarrow} D : D \stackrel{EF}{\longrightarrow} C).$$

The key is to note that checking for the validity of a given initializing sequence (for example, from designers or from ATPG vectors), incurs only the cost of logic simulation. We also note the important difference between the use of a synchronizing sequence and a reset sequence that is potentially derived from a reset signal in the circuit. While the use of reset signals is equally applicable for our result, it may be entirely uninteresting to apply a reduction in the formula based on the use of such signals. However, synchronizing sequences (which are more general than reset sequences), when available, can be used to simplify the properties as illustrated. Further, checking of multiple properties of the kind illustrated can benefit from a single check for the validity of a synchronizing sequence.

2.3 Reduction of CTL Formulae: Formalization

Synchronizability can be formally expressed as a CTL formula: **AG EF** (s_0), where s_0 is the specific state to which the FSM is synchronizable. Using the above CTL formula to represent synchronizability, we state the following result that if there exists

a synchronizing sequence for the FSM, then checking for the existence of at least one path from the synchronized state on which a property holds eventually is equivalent to checking for the property to hold eventually along at least one path from every state. Formally,

Result 1 *AG-EF Reduction*
$\texttt{AG EF}(s = s_0) \wedge \texttt{EF}_{s_0}(\texttt{p}) \implies \texttt{AG EF(p)}$

where s_0 is the specific state to which the FSM is synchronizable, p is the predicate to be checked. The reduction of the formula into the form **EF**(p) is critically helpful in embedding the predicate p directly into the state justification engine of the sequential ATPG algorithm. A brief description of the main phases in a sequential ATPG algorithm and the proposed transformation procedure for obtaining a sequential ATPG problem are discussed later in the paper.

2.4 Proof Mechanization of the CTL Formula Reduction

In this section, we verify and incorporate the model checking reduction results stated in Section 2.3 into a mechanical theorem prover PVS [ORR+96]. PVS provides an integrated environment for the development and analysis of formal specifications and has a powerful theorem prover with a high-degree of automation together with a Binary Decision Diagram (BDD)-based model checker. The verification of properties is performed by invoking appropriate built-in proof strategies. The proof strategies consist of a combination of induction, rewriting, and special purpose decision procedures such as for linear arithmetic and model checking using BDDs.

The proof of Result 1 proceeds by first lifting the CTL operator AG to a general universal quantification on states in PVS and then expanding the EF operator first into a mu-calculus formula which is then expanded into least/greatest fixpoints definitions. Using definitions and theorems of least/greatest fixpoints theory the proof is completed. The proof is fully automatic using PVS after lifting the AG operator to a general universal quantification on states. It takes a few seconds on SPARCstation20 with 32M.

3 Sequential ATPG Algorithms and State Space Exploration

The objective of sequential ATPG algorithms is to generate test sequences that detect all the detectable stuck-at faults in a sequential circuit [ABF90]. There is a large body of literature available in the area of algorithms to solve the sequential ATPG problem. A brief summary of the main steps in a typical sequential ATPG algorithm is now presented. A detailed discussion of these algorithms is beyond the scope of this paper.

A *stuck-at-0(1)* fault refers to the value of a line in the circuit being held to a constant 0(1) value. A sequential ATPG algorithm attempts to detect every such fault (two faults per each line in the sequential circuit) in the circuit. A fault is said to be detected if there exists a sequence that produces different responses on the good machine and faulty

machines. Varying requirements on the start states sequential circuit are possible (the machine may be assumed to start either from a completely unknown state or a from a given start state).

Typical sequential ATPG algorithms achieve this objective of detecting a fault by solving three sub-problems, *excitation, propagation, and (state) justification*. *Excitation* refers to the process of identifying an input vector (on a single time-frame) that can either produce a difference between the two circuits at a primary output or a flip-flop. The objective of *propagation* is to then produce a test sequence that can take a difference value (good/faulty equal to 1/0 or 0/1) from a flip-flop and propagate it to a primary output. *(State) justification* is the process of taking any requirements at the state lines (flip-flops) that were produced at the excitation phase and justify them either to the starting state or the all-unknowns state. Efficient methods to solve each of these problems are available in literature. Current methods are capable of handling designs with thousands of latches.

The main benefits of using a sequential ATPG algorithm are that there is no explicit storage of states required at each time-frame; time-frame expansion is on-the-fly and is restricted only to those parts of the design that truly need to expanded (this is a way to perform on-the-fly abstraction). The algorithms overcome the need to store all the states at each step of navigation through the state space by using decision trees that keep track of variables being assigned at specific stages in the program (for example, a latch may be given a value of 1 at a time-frame and if that value assignment results in no solution to the problem, a value of 0 is reached by backtracking). Hence, this method of searching for a requirement in the state space achieves a balance between a purely breadth-first state exploration method (as in conventional model checkers) and a purely depth-first exploration method (not efficient to explore large state spaces). The state-tuples that the method explores at each time frame are usually decided based on effective heuristics to determine easily *controllable* or *observable* state elements.

3.1 Distinguishing Sequence Generation

Results demonstrating the ability of state-of-the-art sequential ATPG algorithms to justify and efficiently handle large state spaces (over 1700 latches) have been reported in academic literature. In addition, commercial sequential ATPG tools are frequently applied to designs consisting of several thousand latches. We first present the results from a state-of-the-art, deterministic, test generation algorithm [NP91] (capable of proving the indistinguishability of the faulty machine from the good machine) and then present results obtained by using a genetic algorithm-based sequential ATPG [HRP97] that is *extremely efficient for obtaining distinguishing sequences but is incapable of proving undetectable faults*. The fault detection results reported in that work achieved the highest known detection coverages at that time.

These results are presented in Tables 2 and 3. For the data in Table 2, the time limit and backtrack limit for each fault were set to be 20 seconds and 100,000 respectively in each of the circuits except for s35932. because of the large number of faults in it. For this circuit, a two second time limit and a backtrack limit of 10,000 were placed. The columns in the table indicate the circuit name, the number of detected faults, the

number of faults proven to be undetectable by the test generator, the number of aborted faults, the time taken in seconds, and the number of vectors respectively. For the data in Table 3, the columns represent the circuit, checkpoint, number of detected faults, number of vectors produced and the time taken, respectively. The checkpoints refer to varying stages during execution of the genetic algorithm where computation could be stopped. The entries in bold represented the highest reported fault coverages at the time. It is clear from the data in these two tables that sequential ATPG algorithms have the ability to navigate through large state spaces efficiently to achieve the desired objectives.

Table 2. Sequential ATPG results

Circuit	Detected	Undetectable	Aborted	Time (sec.)	Vectors
s298	265	26	17	389	306
s344	314	8	20	489	117
s400	336	9	81	1888	1644
s420	28	152	275	6235	16
s526	51	17	487	10883	34
s641	404	61	2	89	219
s713	476	105	0	23	177
s820	812	31	7	433	928
s832	816	46	8	500	967
s953	89	990	0	147	14
s1238	1283	72	0	14	478
s1423	555	11	949	20359	88
s1488	1439	27	20	1238	1124
s1494	1439	27	20	1238	1124
s5378	3152	148	1303	27078	949
s35932	34719	3856	519	7172	317

4 Model Checking Using Sequential ATPG

The transformation of model checking to stuck-at-fault detection can be performed based on an automata-theoretic approach as illustrated in Figure 1. Given a temporal logic formula, the transformation constructs monitor automata and a test network realizing a function that evaluates to “1” iff the monitor automaton/automata reaches a *bad* state/states. After generating the network, a sequential ATPG algorithm can be invoked on the new circuit with the stuck-at fault to be tested as its objective. Note that transforming model checking to stuck-at-fault detection in this manner may not be the most efficient. It is usually more efficient to build the model-checking objectives such as checking for the reachability of a bad state in the monitor automaton into the implementation of the

Table 3. Sequential ATPG results

Circuit	Ckpt	Det	Vec	Time	Circuit	Ckpt	Det	Vec	Time
s382	1	361	601	1.07 min	s1423	1	**1410**	2065	13.2 min
	2	362	1285	5.9 min		2	**1410**	2965	40.1 min
	3	**364**	1486	8.1 min		3	**1414**	3943	1.27 hr
s444	1	408	354	38.5 sec	s1494	1	1393	295	5.34 min
	2	**420**	753	2.3 min		2	**1453**	540	7.50 min
	3	**424**	1945	20.1 min		3	**1453**	540	7.60 min
s526	1	**431**	486	1.37 sec	s5378	1	**3562**	2175	4.60 hr
	2	**442**	1098	8.3 min		2	**3607**	4461	25.1 hr
	3	**454**	2642	54.5 min		3	**3639**	11571	37.8 hr
s713	1	475	157	1.1 min	s35932	1	**35100**	257	2.1 hr
	2	**476**	176	1.30 min		2	**35100**	257	10.2 hr
	3	**476**	176	1.31 min		3	**35100**	257	10.9 hr
s820	1	812	572	3.07 min	am2910	1	**2190**	953	6.25 min
	2	**814**	590	3.60 min		2	**2197**	1761	13.5 min
	3	**814**	590	3.63 min		3	**2198**	2509	29.4 min
s1196	1	1235	521	1.12 min	div16	1	1727	352	32.0 min
	2	1237	536	1.21 min		2	**1810**	1168	2.62 hr
	3	**1239**	574	1.49 min		3	**1814**	3476	8.1 hr

ATPG algorithm as state justification objectives. We note again that reducing formulae to forms that permit passing of objectives directly to the state-justification engine is critically helpful in the efficiency of this procedure.

Any property for which a monitor automaton can be constructed to result in a test network of manageable size can be checked by such a transformation. Intuitively, this approach seems ideally suited for checking *safety* properties i.e., those properties every violation of which occurs after a finite execution of the system. A theoretical characterization of the exact class of properties that can be transformed effectively into sequential ATPG problems was not attempted in this paper. Our paper is targeted at studying the efficiency of sequential ATPG algorithms for state space exploration. Specifically, we have restricted the properties to be of the form **EF** P. The general reduction approach would be based on techniques for constructing monitor automata for more general properties [Wol82,FTMo83,FTMo85,NFKT87] and may be able to exploit recent results on constructing smaller automata based on a classification of safety properties [KV99].

4.1 Example

The transformation of a property of the form **EF** P, where P is a conjunction of value assignments to some signals in the circuit is shown in Figure 2. The property checked in the example is **EF (y1 = 1 AND y2 = 1)**. In the example, an AND gate tying the

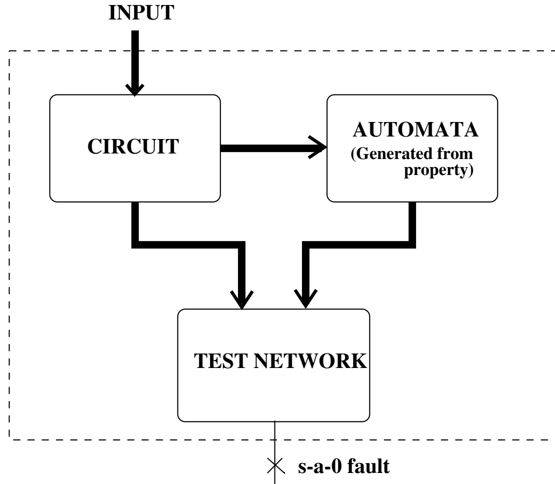

Fig. 1. Transformation of model checking to sequential ATPG: resulting circuit structure

signals **y1** and **y2** is added to the original circuit under verification. Note that no monitor automata are needed to be constructed for this example.

4.2 Three-Valued Testability and Overspecification

It is important to note that various definitions of untestability have been discussed in literature [PR92,PR93,CM93,Bop97]. A detailed discussion of these definitions is beyond the scope of this paper. However, two of the most important issues involved in the definition of untestability are briefly discussed.

First, we consider the notion of three-valued testability. A fault is three-valued testable iff there exists a test sequence that can produce a difference (0/1 or 1/0) at a primary output when the good and the faulty machines are started from the all-unknowns (all Xs) states and three-valued logic simulation is used to evaluate the output responses. This is the notion of untestability used by most practical gate-level ATPG algorithms operating with three-valued logic. The set of three-valued testable faults was shown to be a subset of all testable faults [PR92,PR93]..

Secondly, we consider the problem of overspecification [CM92,CM93] present in **some** sequential ATPG algorithms. The problem occurs because most gate-level test generation algorithms for sequential circuits are based on the use of the time frame expansion technique [ABF90] and the use of combinational test generation algorithms such as PODEM [Goe90] within each time frame. Some underlying combinational test generation algorithms, unfortunately, may overspecify the requirements at present state

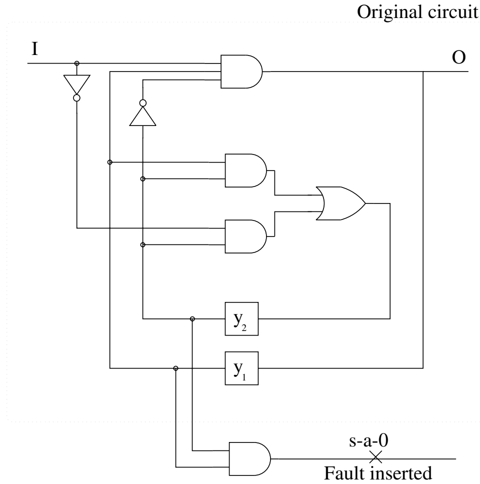

Fig. 2. Transformation of model checking to sequential ATPG: example

lines while processing a time frame (PODEM, for instance, may overspecify the requirements). This, of course, does not create a problem for combinational circuits, because overspecifying primary inputs does not affect the applicability of a test vector to the circuit. However, for sequential circuits, whenever this occurs, the objectives on the previous time frame may be more specified than necessary and may result in an incorrect claim by the test generation algorithm regarding the three-valued testability of the fault.

While the loss of accuracy caused by the use of three-valued logic and overspecification have not been much of a concern to the test generation problem itself (because it was shown that they cause only a small loss of fault coverage in the test generation process), it has potentially serious implications as far as using some ATPG algorithms for verification is concerned. Untestability characterization and several techniques for improving the accuracy of the test generation process (for example, based on verifying the existence of initializing sequences) have been presented earlier [PR93,CM93, Bop97]. The design verification application must be carefully analyzed before choosing the appropriate sequential ATPG algorithm.

5 Experimental Results

Results on state justification experiments for some *hard-to-test* ISCAS circuits

Four circuits have been chosen for our experiments on property checking because of the difficulty posed by them to sequential ATPG algorithms. Each of these circuits has

Table 4. Number of properties successfully checked using VIS and ATPG

Circuit	Number of signal assignments per property (5 properties for each case)					
	2		3		5	
	VIS	ATPG	VIS	ATPG	VIS	ATPG
s526	5	5	5	4	5	5 (4+1)
s1423	0	4	0	0	0	0
s5378	0	5 (2+3)	0	4	0	0
s35932	0	4	0	1	0	1

been checked to be initializable using ATPG-generated test sequences. The sequential ATPG algorithm being used in our experiments is HITEC [NP91]. Our experiments on the ISCAS circuits were run on a SPARCstation 20 with 64MB of memory. A time limit of 20 seconds and a backtrack limit of 100,000 where set in the ATPG algorithm for each of the formulae checked.

Table 4 shows the experimental results comparing the performance of the ATPG-based approach with VIS [Gro96]. For each circuit, fifteen properties of the form **AG EF**(P_i) were generated and verified against the circuit. For each formula, P_i was generated by choosing a specific number of internal signals (five properties generated for each case with two, three and five signals), specifying Boolean values for them randomly, and **AND**ing them together. For example, a case with two signal assignments could consist of a P_i with (a=1 AND b=0). The table lists the number of cases out of the five chosen cases that the formula was successfully proven/disproved. For entries where numbers are provided in parantheses, the first number in the paranthesis indicates the number of cases for which vectors were obtained and the second number indicates the number of cases for which no test sequence was available (proven untestable). As can be clearly seen from the Table, the ATPG-based approach is capable of providing results for certain formulae even for very large circuits.

It is also interesting to note the differences in the state space exploration strategy in the two approaches. Results are shown on the small circuit s526 for which VIS could successfully complete the model-checking experiment to compare it with the state space exploration strategy in the ATPG-based approach. These results are presented in Table 5. Five signals were chosen from the circuit, random Boolean values were assigned to these signals and they were tied together by an **AND** as before. The numbers of vectors produced to achieve the required assignment of internal signals and the times required for generating these sequences are shown. The number of vectors produced by the ATPG-based approach indicates the performance of the structural search (somewhere between a DFS and a BFS) as opposed to the BFS-like search (for these types of formulae) involved in VIS. We emphasize again, of course, that the ATPG-based approach does not need to build the state transition relation and extensive functional representations and hence is memory efficient. Even the largest benchmark circuit tried (with 1728 flip-flops) required less than 20MB of memory.

Table 5. Differences in state space exploration strategy

Circuit	VIS		ATPG	
	Vectors	Time (sec.)	Vectors	Time (sec.)
s526	21	0.9	132	880.8
s526	48	1.8	240	41.4
s526	1	0.6	3	0.03
s526	8	0.8	50	3.6
s526	1	0.8	3	0.02

Results on property checking experiments on an industrial circuit

Experiments were also performed on a large industrial circuit to verify the effectiveness of the proposed sequential ATPG-based property checking system. The design used was that of an IO controller consisting of five modules: ADDRESS_DECODER, OUT_CONTROL, READ_CONTROL, IRQ_CONTROL and REG_BANK. The circuit consisted of 148 flip-flops, 51 primary inputs, 51 primary outputs and 1753 basic cells.

Experiments were performed to identify load sequences for obtaining specific values at registers embedded deep in the design. The ATPG approach was compared against a state-of-the-art, model checking tool BINGO [INH96,IN97]. The ATPG approach was successful in obtaining a sequence for *every* register tried while BINGO could not produce any sequence more than 6 vectors long. BINGO was terminated in each of these cases because the memory requirement exceeded 500 MB. The ATPG approach required no more than 20 MB for each of the cases and produced vector sequences of length upto 22.

6 Conclusions

In this paper we have given an efficient method based on stuck-at-fault testing techniques for automatic verification for a useful subclass of properties of synchronizable FSMs, typical of hardware designs. We have presented reduction of CTL formulae of the form **AG EF** P to **EF** P based on the existence of synchronization sequences and proven the reductions in the PVS proof-checker. Model checking the reduced formulae is transformed to stuck-at-fault testing and solved by sequential ATPG.

We have shown that the method has the potential to scale up to large hardware designs for which current model checking methods fail. The reason our method scales up is because it does not involve extracting and computing expensive functional information such as the complete state-transition relation. Instead, the approach relies on efficient fault-testing that exploits the circuit structure of the hardware design to be verified. As part of future work, we plan to characterize and experiment with more general properties that can be reduced to the stuck-at-fault testing problem and to investigate methods to incorporate the advantages of BDD-based model checking and the ATPG-based approach into a unified framework.

References

[ABF90] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital System Testing and Testable Design*. New York, NY: Computer Science Press, 1990.

[AH96] R. Alur and T. A. Henzinger, editors. *Computer-Aided Verification, CAV '96*, volume 1102 of *Lecture Notes in Computer Science*, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[Bop97] V. Boppana. State information-based solutions for sequential circuit diagnosis and testing. Technical Report CRHC-97-20, Ph.D. thesis, Center for Reliable and High-Performance Computing, University of Illinois at Urbana-Champaign, July 1997.

[Bry95] R.E. Bryant. Binary decision diagrams and beyond: Enabling technologies for formal verification. In *Proceedings of the International Conference on Computer-Aided Design*, pages 236–243, November 1995.

[CA89] K. T. Cheng and V. Agrawal. State Assignment for Initialilzable Synthesis. In *Proc. Intl. Conf. Computer-Aided Design*, pages 212–215, November 1989.

[CA93] K. T. Cheng and V. Agrawal. Initializability consideration in sequential machine synthesis. *IEEE Trans. Computers*, 41(3):374–379, March 1993.

[CGL94] E. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *A Decade of concurrency–Reflections and Perspectives*, volume 803 of *Lecture Notes in Computer Science*. Springer-Verlag, 1994.

[CM92] K. T. Cheng and H. K. T. Ma. On the over-specification problem in sequential ATPG algorithms. In *Proc. Design Automation Conf.*, pages 16–21, June 1992.

[CM93] K. T. Cheng and H. K. T. Ma. On the over-specification problem in sequential ATPG algorithms. *IEEE Trans. Computer-Aided Design*, 12(10):1599–1604, October 1993.

[FTMo83] M. Fujita, H. Tanaka, and T. Moto-oka. Verification with prolog and temporal logic. In *Proc. of IFIP WG10.2 International Conference on Hardware Description Languages and their Applications*, May 1983.

[FTMo85] M. Fujita, H. Tanaka, and T. Moto-oka. Logic design assistance with temporal logic. In *Proc. of IFIP WG10.2 International Conference on Hardware Description Languages and their Applications*, Aug. 1985.

[Goe90] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Trans. Computers*, C-30(3):215–222, March 1990.

[Gro96] The VIS Group. VIS: A system for verification and synthesis. In Alur and Henzinger [AH96], pages 428–432.

[Hen68] F. C. Hennie. *Finite-State Models for Logical Machines*. New York, NY: John Wiley & Sons, Inc., 1968.

[HRP97] M. S. Hsiao, E. M. Rudnick, and J. H. Patel. Sequential circuit test generation using dynamic state traversal. In *Proc. European Design and Test Conf.*, pages 22–28, March 1997.

[IN97] H. Iwashita and T. Nakata. Forward model checking techniques oriented to buggy designs. In *Proc. Intl. Conf. Computer-Aided Design*, pages 400–404, November 1997.

[INH96] H. Iwashita, T. Nakata, and F. Hirose. Ctl model checking based on forward state traversal. In *Proc. Intl. Conf. Computer-Aided Design*, pages 82–87, November 1996.

[Koh78] Z. Kohavi. *Switching and Finite Automata Theory*. New York, NY: McGraw-Hill, 1978.

[Kur94] R. P. Kurshan. *Computer-Aided Verification of Coordinating Processes—The Automata-Theoretic Approach*. Princeton University Press, Princeton, NJ, 1994.

[KV99] O. Kupferman and Moshe Y. Vardi. Model checking of safety properties. In *CAV99*, 1999.

[McM93] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Pub., Boston, MA, 1993.

[NFKT87] H. Nakamura, M. Fujita, S. Kono, and H. Tanaka. Temporal logic based fast verification systems using cover expressions. In *Proc. of IFIP WG10.5 International Conference on VLSI*, Aug. 1987.

[NP91] T. Niermann and J. H. Patel. HITEC: A test generation package for sequential circuits. In *Proc. European Design Automation Conf.*, pages 214–218, February 1991.

[ORR+96] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS: Combining specification, proof checking, and model checking. In Alur and Henzinger [AH96], pages 411–414.

[Pel96] Doron Peled. Combining partial order reductions with on-the-fly model-checking. *Formal Methods in System Design*, 8(1):39–64, 1996.

[PR92] I. Pomeranz and S. M. Reddy. The multiple observation time test strategy. *IEEE Trans. Computer-Aided Design*, 40(5):627–637, May 1992.

[PR93] I. Pomeranz and S. M. Reddy. Classification of faults in synchronous sequential circuits. *IEEE Trans. Computers*, 42(9):1066–1077, September 1993.

[TSNH98] K. Takayama, T. Satoh, T. Nakata, and F. Hirose. An approach to verify a large scale system-on-a-chip using symbolic model checking. In *Proceedings of the International Conference on Computer Design*, pages 308–313, October 1998.

[Wol82] P. Wolper. “synthesis of communicating processes from temporal logic specifications”. Technical Report STAN-CS-82-925, Dept. of Computer Science, Stanford University, 1982.