Exploiting State Encoding for Invariant Generation in Induction-based Property Checking

Markus Wedler Dominik Stoffel Wolfgang Kunz
Department of Electrical and Computer Engineering
University of Kaiserslautern
D-67653 Kaiserslautern, Germany
wedler@eit.uni-kl.de stoffel@eit.uni-kl.de kunz@eit.uni-kl.de

Abstract— This paper focuses on checking safety properties for sequential circuits specified on the RT-level. We study how different state encodings can be used to create a gate-level representation of the circuit that facilitates the computation of effective invariants for induction-based property checking. Our experiments show the strong impact of state encoding on the efficiency of the induction process.

I. Introduction

Checking safety properties is an unsolved problem in design automation. Existing methods either perform an FSM traversal [7], [3] or an induction on the property [1] [9]. The first kind of method suffers from the state explosion problem. The second kind of method suffers from the fact that many properties are not strong enough to be proven with a small induction depth. When these methods fail proving a property the user has to strengthen the property by inventing a lemma that is supposed to hold in all reachable states. In [2] the authors generate invariants for induction-based equivalence checking. In this paper, we propose a method to generate this lemma by using state encoding and RT-level information. For large machines that can be independently controlled we suggest a binary encoding and strengthen the property by using the upper bound given by the RT-level description. For coupled machines the proposed method uses *structural FSM traversal* [10], [11], [12], to find an over-approximation of the set of reachable states. This is only effective if an appropriate state encoding is chosen.

The paper is organized as follows: Section 2 gives a brief review on induction-based property checking. Section 3 shows how state encoding can be used to strengthen properties automatically. Section 4 gives an overview over our implementation of the ideas presented in Section 3. Section 5 reports on experimental results.

II. Induction-based Property Checking

In this section we give a brief review of existing induction-based methods for property checking. In [9] and [1] the authors study the use of induction to verify safety properties. The basic model for sequential circuits is a finite state machine M which is a 6-tuple $M = (I, S, \delta, S_0, O, \lambda)$ where I is the input alphabet, S is the set of states, $\delta : S \times I \longrightarrow S$ is the next-state function, S_0 is a set of initial states, O is the output alphabet and $\lambda : S \times I \longrightarrow O$ is the output function. Given such a machine, we would like to check whether a condition P holds for all reachable states. If this is the case we call the machine P-safe. In the following, we identify P with the set of all states $s \in S$ where P holds. We now repeat some sufficient conditions for a machine to be P-safe [9], [1].

Lemma 1. *A finite state machine is P-safe if the following conditions hold:*

- *P holds in all states $s_0 \in S_0$, i.e., $S_0 \subset P$.*
- *If P holds in a state $s \in S$ then it also holds in all next states reachable from s, i.e., $s \in P \Rightarrow \forall i \in I : \delta(s, i) \in P$.*

It is easy to see that this condition is not strong enough to prove P-safeness for realistic designs. If there is an unreachable state $s \in S$ where P holds the second condition of Lemma 1 implies that P holds in all successors of s.

Figure 1 shows the state transition graph of a 3-bit

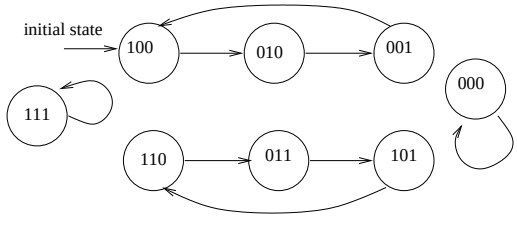

Fig. 1. State transition graph of a 3-bit ring counter

ring counter taken from [9]. We cannot prove that state 101 is unreachable from the initial state using Lemma 1 ($P = \{s \in S | s \neq 101\}$). The reason is that the second condition of the lemma fails. The property holds in state 011, but does not in the next state 101. For this reason, a stronger induction method has to be considered.

Lemma 2. *A finite state machine M is P-safe if there is a $k \geq 0$ such that the following conditions hold:*

- *For all paths $s_0, s_1, ..., s_k$ in the state transition graph of M with $s_0 \in S_0$, P holds for all s_i.*
- *If P holds in the states on some path $s_0, ..., s_k$ [1] in the state transition graph of M then it also holds in all next states reachable from s_k, i.e., $\forall i \in I : \delta(s_k, i) \in P$.*

[1] Note that s_0 does not have to be an initial state.

With Lemma 2 we can prove the unreachability of 101 with $k = 2$. However, this condition is still too weak to prove safety properties for many designs of interest.

Suppose we add an enable signal to the ring counter of Figure 1. If the enable signal is set the new machine has the same behavior as the ring counter otherwise the machine stays in the current state. With this extension the property $P = \{s \in S | s \neq 101\}$ is still valid. However, Lemma 2 is not strong enough to prove it. Again the second condition fails. In state 011 P holds and for all $k \geq 0$ there is the path $s_0, ..., s_k$ in the state transition graph with $s_i = 011$. In the next state 101 P does not hold anymore.

This problem is created by the cycles in the set of unreachable states. We can exclude these cycles by extending the second condition of Lemma 2:

Lemma 3. *A finite state machine M is P-safe if there is a $k \geq 0$ such that the following conditions hold:*

- *For all paths $s_0, s_1, ..., s_k$ in the state transition graph of M with $s_0 \in S_0$, P holds in all s_i.*
- *If P holds in the states on some path $s_0, ..., s_k$ in the state transition graph of M, with pairwise different s_i then it also holds in all next states reachable from s_k, i.e., $\forall i \in I : \delta(s_k, i) \in P$.*

Using Lemma 3 we can also prove the unreachability of 101 in the extended machine with $k = 2$. However for practical use of induction-based methods, we can afford only small values of k. An algorithm that is based on Lemma 3 has to unroll the transition relation $k + 2$ times and solve two satisfiability problems on the resulting structure. If k is unknown, as is the case in most applications, the algorithm has to perform a search for a sufficiently large k. This implies that the algorithm has to solve $2 * (k + 2)$ SAT problems.

In the next section we are going to analyze conditions that help to minimize k in Lemma 3. Before we continue on this topic it should be mentioned that a method based on Lemma 3 is complete.

Lemma 4. *If a finite state machine M is P-safe there is a $k \geq 0$ such that the conditions of Lemma 3 hold.*

III. State Encodings

Standard property checking tools start from an RT-level design specification in VHDL or Verilog that is augmented by some property to be verified. From this specification the front-end of the property checker generates a gate level description of the design and the property. This description is the basis for applying standard boolean proving techniques like SAT solving.

Several degrees of freedom exist when generating the gate level model from the RTL specification. It should be noted that the choice of an appropriate state encoding can help significantly to reduce the proving complexity. For example, in the context of *symbolic FSM traversal* it was shown in [4] that retiming can have a strong impact on the BDD sizes. In this paper, we demonstrate that this freedom can also be used to reduce the induction length k of a prover based on Lemma 3.

We will analyze the induction length k needed to prove that a machine is P-safe. We will present a method to strengthen the property with information from the RT-level and with state space information derived from *synthesis for state space representability* and a *structural FSM traversal* [10], [11], [12].

An algorithm based on Lemma 3, applied on a machine that is not P-safe, will increase k, until a path of length $k + 1$ from an initial state to an unsafe state is found. In the worst case, $k + 1$ will be the diameter of the machine. If the algorithm is applied on a machine that is P-safe, the induction length k needed does not depend on the set of reachable states. The following theorem gives an exact answer to the question, what is the minimum induction length k needed.

Theorem 1. *Let M be a P-safe finite state machine. Let $p_1, ..., p_t$ be a longest path in the transition graph of M such that the following conditions hold:*

- *$p_i \neq p_j$, for $i \neq j$*
- *$p_i \in P$, for $i \in \{1...t\}$*
- *p_t has a next state $p_{t+1} = \delta(i, p_t)$ with $p_{t+1} \notin P$.*

Then $k = t$ is the smallest k such that the conditions of Lemma 3 hold.

Proof: First we show that the conditions of Lemma 3 hold for k. The first condition is clear because M is P-safe. Suppose the second condition is wrong. Then there is a loop-free path $s_0, ..., s_t$ such that P holds for all s_i and there is a next state $s_{t+1} = \delta(s_t, i) \notin P$. Thus $p_1, ..., p_t$ is not the longest path with the above conditions. Therefore, the second condition of Lemma 3 must hold. Next we will show that there is no smaller $k' < k$ that fulfills the conditions of Lemma 3. This results from the fact that $p_{k-k'}, ..., p_t$ is a counterexample for the second condition of Lemma 3. □

Theorem 1 states that the induction length k is determined by the longest loop-free path along P-safe states to a non P-safe state. If the property we want to check is weak, in the sense that there are a lot of unreachable states that have this property, this path and, hence, the induction length can be very large. Therefore, we look for information from the RT-level that can strengthen the property. We can exploit the fact that the front-end of the property checker is free in the choice of a state-encoding. We make the assumption that for all state machines the set of reachable states is a strongly connected component of the state transition graph. This is, e.g., the case if the machine has a synchronous reset input. We first analyze how these facts can be exploited if the design is restricted to a single finite state machine. After that we will look at designs with multiple finite state machines.

A. Binary Encoding

We first assume that the front-end of our property checker has synthesized the RT-level description using a binary state encoding. Each state variable in the design is synthesized into an array of registers $R[0...m]$ and the states are binary numbered from 0 up to $n-1$. The information about the number of states n is available from the RTL description of the design. Instead of proving the property P we prove

$$P' = P \wedge (n > \sum_{i=0}^{m} 2^i R[i]). \quad (1)$$

This can easily be done by adding a binary comparator to the property. If we have a single finite state machine P' can only be valid in the set of reachable states unless the designer has produced unreachable states in his RTL description. However, those unreachable states that result from the fact that state machines do not always have a number of states being a power of 2 will not fulfill the property P'. So in many practically relevant cases Theorem 1 will give us $k = 0$ as induction length unless the designer has created unreachable states in his description himself. Even if this has happened Equation (1) may still reduce the induction length substantially. We now analyze what happens if a design contains more than one state machine. Figure 2 shows two coupled machines. x_1 and x_2 are sets of independent inputs and X is a set of common inputs. Formally, the partially coupled machine is defined as follows.

Definition 1. *Let A, B be two finite state machines with*

- $A = (X_2 \times X, S_A, \delta_A, S_{A_0}, O_A, \lambda_A)$ *and*
- $B = (X_1 \times X, S_B, \delta_B, S_{B_0}, O_B, \lambda_B)$.

The partially coupled machine $M = (I, S, \delta, S_0, O, \lambda)$ of A and B is defined as follows:

- $I = X_1 \times X_2 \times X$,
- $S = S_A \times S_B$,
- $\delta((s_A, s_B), (x_1, x_2, x))$
  $= (\delta_A((x_2, x), s_A), \delta_B((x_1, x), s_B))$
- $S_0 = S_{A_0} \times S_{B_0}$
- $O = O_A \times O_B$
- $\lambda((s_A, s_B), (x_1, x_2, x))$
  $= (\lambda_A((x_2, x), s_A), \lambda_B((x_1, x), s_B))$

Note that this definition covers the following two extreme situations. If X is empty the machines are independent. In this case the set of reachable states $Reach(M)$ of M is just the product of the sets of reachable states of A and B, i.e., $Reach(M) = Reach(A) \times Reach(B)$. Therefore, strengthening the property with all the upper bounds for the state variables like in Equation (1) is helpful in this case.

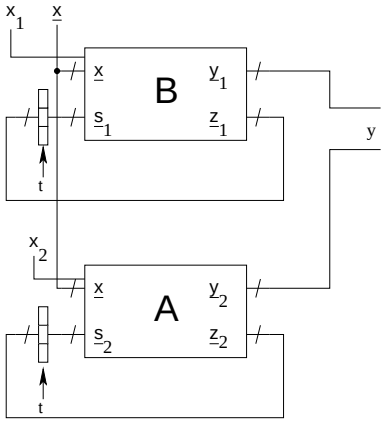

Fig. 2. Coupled machines with two independent inputs

The other extreme is that X_1 and X_2 are empty. This is a product machine of the type occurring in sequential equivalence checking. Whereas in equivalence checking the two machines have a lot of similarities that can be exploited to reduce the proof complexity, we cannot expect this to be true for property checking. But as we will see later, our control over the choice of the state encoding can help us here.

For now we simply state that in all cases where the machines are not independent we only have $Reach(M) \subset Reach(A) \times Reach(B)$. Therefore, strengthening the property with all the upper bounds for the state variables like in Equation (1) is not strong enough to obtain small values for the induction length.

Outputs of one machine can also be used as inputs for another machine. This leads to a more general notion of combined machines. Of course, one has to make sure that no cycles in the combinational logic are created by this combination. For more details see standard textbooks like [6].

B. One-hot Encoding

Especially in telecommunication applications, large state machines are created by combination of many small machines. In this case it is affordable that the property checker chooses a one-hot encoding for the small machines. Also, designers frequently use "flags" in RTL code to store certain information about the current state of the designed system. Note that such flags can be interpreted as 2-state machines and the one-hot encoding is just the flag itself.

Binary state encoding distributes the state information over all registers. For example, if M is a coupled machine of the two machines given by the state diagrams in Figure 3, the state $(2, 2)$ is not reachable from the initial state. This creates the following dependencies between the registers $r_1[i], r_2[i]$ encoding the states:
$r_1[1] \wedge \overline{r_1[0]} \Rightarrow r_2[0] \vee \overline{r_2[1]}$ and $r_2[1] \wedge \overline{r_2[0]} \Rightarrow r_1[0] \vee \overline{r_1[1]}$.
The calculation of these dependencies is computationaly very expensive. With one-hot encoding, certain dependencies between the states in a composed machine can be modeled by implications containing only single literals. Given the registers $r_1[2], r_2[2]$ encoding state 2 of the

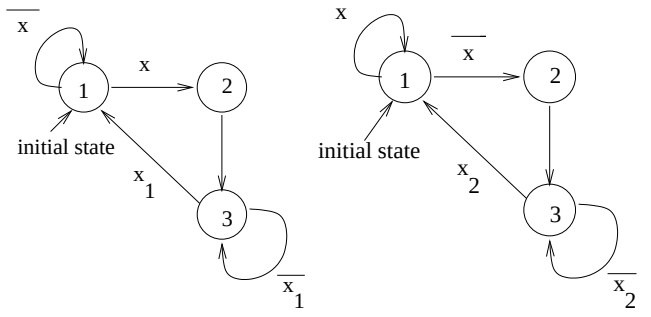

Fig. 3. Example

above machines we have the implications $r_1[2] \Rightarrow \overline{r_2[2]}$ and $r_2[2] \Rightarrow \overline{r_1[2]}$.

These implications can be calculated by performing a structural FSM traversal. For coupled machines having dependencies between two machines only this will give us an exact representation of the set of reachable states. Note that this guarantees an induction length $k = 0$.

Theorem 2. *Let M be a combined machine of the one-hot encoded machines M_1, M_2 and $r_i[1..k_i](i = 1, 2)$ be the registers of M_i. Then the reachable state set of M is characterized exactly by a set I of implications between the registers of M_i, two sets of constant registers C_0, C_1 and the equations $\sum_{l=1...k_1} r_1[l] = 1 = \sum_{l=1...k_2} r_2[l]$.*

Proof: For all states $(s_1, s_2) \in (Reach(M_1) \times Reach(M_2)) \backslash Reach(M)$, we add the implications $r_1 \Rightarrow \overline{r_2}$ and $r_2 \Rightarrow \overline{r_1}$ of the corresponding registers to I. For both machines M_k and all pairs of registers $(r_k[1], r_k[2])$ of M_k we add the implication $r_k[1] \Rightarrow \overline{r_k[2]}$ to I. Let π_k be the projection on $Reach(M_k)$. For all states $s_k \in Reach(M_k) \setminus \pi_k(Reach(M))$ we add the corresponding register r_k to C_0. If $\pi_k(Reach(M)) = \{s_k\}$ we add the corresponding registers r_k to C_1.

It is easy to see that I, C_0, C_1 together with the above equations are an exact characterization of $Reach(M)$. A value assignment to the registers belongs to a reachable state iff it satisfies all implications in I, constants in C_0, C_1 and the above equation. □

In the case that more than two machines are dependent we obtain an over-approximation of the set of reachable states. Our experiments show that this over-approximation is still an invariant that can reduce the induction length dramatically. It works well in cases where the property checks dependencies between pairs of machines like bus conflicts, acknowledge requests, etc.

IV. Implementation of the property checker

In this section we give a brief overview over our implementation based on the ideas in this paper. Figure 4 shows the overall flow.

The RTL-code of the design is augmented by combinational logic that calculates the property. We create a gate level representation of our designs using the front-end of an industrial property checker. We choose one-hot encoding for small machines and binary otherwise. Note again

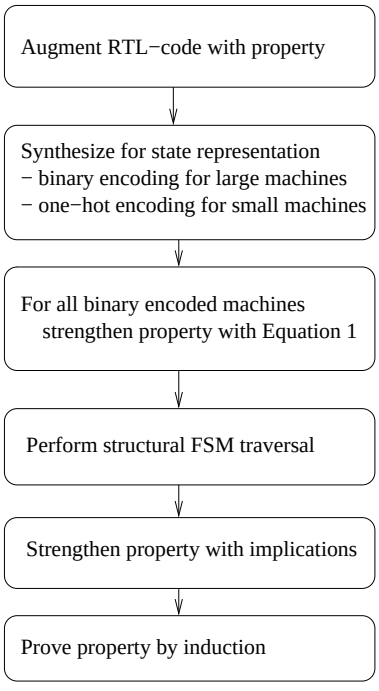

Fig. 4. Overall flow of the proposed method

that the encoding used for verification can be chosen in the RT-to-gate front-end of the verification tool independently of the encoding used for the actual implementation. The property is strengthened with Equation (1) for the binary encoded machines and with the results of a structural FSM traversal. After this we prove the property by induction starting with induction length 0, increasing the induction length until a proof is found, a counterexample is generated or a user-defined upperbound for the induction length is reached.

The proofs for each induction length are translated into SAT problems and given to the well-known SAT-solver CHAFF [8]. During the structural FSM traversal the implications are calculated using *recursive learning* [5].

V. Experimental Results

To examine the benefits and the limitations of our ideas we created designs that are hard to prove for induction-based methods. A hard example with a single machine is a modulo-$2^k + 2^{k-1}$ counter. For this design the synthesis with binary encoding creates $k+1$ registers. The property we verified is that the state $2^{k+1} - 1$ is never reached. The pure induction method requires an induction length of $2^k - 1$ for the design. After strengthening the property with Equation 1 the proof was completed with induction length 0. The results are shown in the Figure 5. P refers to the original property, P' is the property strengthened by Equation (1). The second design we created contained two coupled modulo-$2^k + 1$ counters. One of the counters counts from $2^k + 1$ down to 0, the second counts from 0 up to $2^k + 1$. The property we verified is that the state $(0, 1)$ is never reached. After strengthening the property with Equation 1 the proof was completed with induction length $2^k + 1$, without strengthening it was $2^{k+1} - 1$. This shows

k	induction length		CPU-time (hh:mm:ss)	
	P	P'	P	P'
5	15	0	00:01	00:00
6	31	0	00:07	00:00
7	63	0	01:53	00:00
8	127	0	11:58:33	00:00
15	>1000	0	aborted	00:00
23	>1000	0	aborted	00:01
30	>1000	0	aborted	00:03

Fig. 5. Results for mod-$\mathbf{2^k + 2^{k-1}}$ counters

k	induction length		CPU-time (hh:mm:ss)	
	P	P'	P	P'
5	63	33	0:04:06	0:00:33
6	127	65	1:32:31	0:10:36
7	255	129	aborted	3:33:24
15	>1000	>1000	aborted	aborted

Fig. 6. Two coupled mod-$\mathbf{2^k + 1}$ counters

the limitation of Equation 1 when dealing with coupled designs.

Motivated by industrial designs that contain multiple instances of state machines with a relatively small number of states, the third design we created contained several state machines having the state diagram of Figure 7. This state diagram was also motivated by a true industrial application in telecommunication.

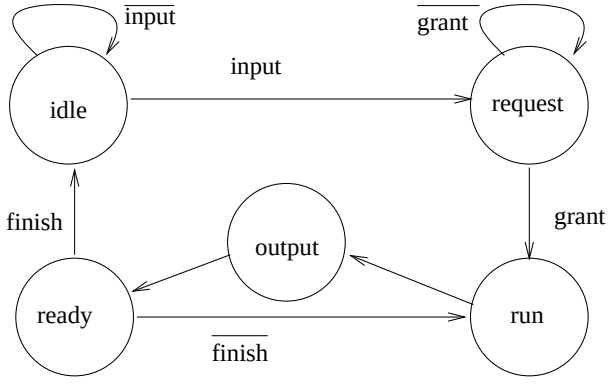

Fig. 7. Example for multiple machines

Each of the machines has its own input line. When an input occurs the machines request for a resource. An arbiter was implemented to receive the request from the machines and to set the grant signal when the resource is free. It also creates a finish signal when another request occurs while a machine is running. Many such machines can be connected to a large controller that handles the requests from many clients for a central resource in some communication network.

In the following, k is the number of machines combined. The property we prove is that two machines will never be in the run state at the same time. So we not only check that the arbiter works correctly, but we also check whether each of the components reacts correctly to the arbiters signals. The models generated by the front-end contained up to 170 registers. Tables 8 and 10 are organized as follows. The first column is the number of clients and the second and the fifth column report the encodings chosen for the components ((B)inary/ (O)ne-hot). Columns three and four and columns six and seven contain the induction length for the proof of the original property and the property strengthened with implications. Tables 9 and 11 are organized in the same way but report the CPU times. The time for invariant generation is included here.

k	induction length					
	enc	P	P∧Impl	enc	P	P∧Impl
2	B	14	14	O	14	**0**
4	B	28	20	O	28	**0**
8	B	>40	>40	O	>40	**0**
16	B	>30	>30	O	>30	**0**

Fig. 8. Induction length for multiple coupled machines with binary dependencies

k	CPU-time (mm:ss)					
	enc	P	P∧Impl	enc	P	P∧Impl
2	B	00:09	0:00:11	O	00:13	**00:01**
4	B	40:05	0:04:53	O	41:57	**00:08**
8	B	abort	abort	O	abort	**01:25**
16	B	abort	abort	O	abort	**06:56**

Fig. 9. CPU-times for multiple coupled machines with binary dependencies

k	induction length					
	enc	P	P∧Impl	enc	P	P∧Impl
4	B	>32	24	O	>32	**3**
5	B	>32	26	O	>32	**3**
6	B	>32	21	O	>32	**3**
7	B	>32	>27	O	>32	**3**

Fig. 10. Induction length for multiple coupled machines with non-binary dependencies

k	CPU-time (hh:mm:ss)					
	enc	P	P∧Impl	enc	P	P∧Impl
4	B	abort	0:56:02	O	abort	**0:00:57**
5	B	abort	2:49:22	O	abort	**0:01:16**
6	B	abort	10:09:31	O	abort	**0:02:15**
7	B	abort	aborted	O	abort	**0:05:59**

Fig. 11. CPU-times for multiple coupled machines with non-binary dependencies

Finally we report the numbers of registers and the numbers of gates for the models in Tables 12 and 13.

k	enc	# Regs	enc	# Regs
2	B	19	O	23
4	B	41	O	49
8	B	78	O	94
16	B	138	O	170

Fig. 12. Size of coupled machines with binary dependencies

k	enc	# Regs	enc	# Regs
4	B	47	O	63
5	B	52	O	74
6	B	63	O	92
7	B	72	O	109

Fig. 13. Size of coupled machines with non-binary dependencies

The last design we created contained the same state machines but we supposed that we have two interchangeable resources. Therefore we replaced the arbiter by a 2-resource arbiter. We proved that 3 machines will never be in the run state at the same time. Note that this property cannot be expressed by implications between the registers of the machine. Nevertheless, *structural FSM traversal* provides an over-approximation of the state space that helps to significantly reduce induction depths.

The results show that structural FSM traversal can generate powerful invariants in the case of one-hot encoding. Sometimes it also reduces induction length in the case of binary encoding. But this reduction is not as powerful as in the case of a one-hot encoding. Instead of minutes it takes hours to finish the proof.

VI. Conclusion

In this paper we exploit that the choice of state encoding during verification is independent from the state encoding of the actual implementation. We show that this can be used to reduce the complexity of induction-based property checking. In particular, we introduce invariants for binary encodings and demonstrate that one-hot encoding together with structural FSM traversal creates useful invariants for many applications.

References

[1] Pasosh Aziz Abdulla, Per Bjesse, and Niklas E'en. Symbolic reachability analysis based on sat solvers. In *Proc. Sixth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS-00)*, pages 411–425, March 2000.

[2] Per Bjesse and Koen Claessen. SAT-based verification without state space traversal. In *Formal Methods in Computer-Aided Design*, pages 372–389, 2000.

[3] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, and D. L. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design*, 13(4):401–424, April 1994.

[4] Andreas Kühlmann and Jason Baumgartner. Transformation-based verification using generalized retiming. In *Proc.Intl. Conf. Computer Aided Verification(CAV-01)*, pages 104–117, July 2001.

[5] Wolfgang Kunz and Dhiraj Pradhan. Recursive learning: A new implication technique for efficient solutions to CAD problems: Test, verification and optimization. *IEEE Transactions on Computer-Aided Design*, 13:1143–1158, Sep. 1994.

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

[7] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Boston, 1993.

[8] M.W. Moskewicz, C.F. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient sat solver. In *Proc. Intl. Design Automation Conference (DAC-01)*, pages 530–535, June 2001.

[9] Mary Sheeran, Satnam Singh, and Gunnar Stalmarck. Checking safety properties using induction and a sat solver. In *Proc. Intl. Conf.Formal Methods in Computer-Aided Design(FMCAD 2000)*, volume 1954 of *Lecture Notes in Computer Science*. Springer, November 2000.

[10] Dominik Stoffel and Wolfgang Kunz. Record & play: A structural fixed point iteration for sequential circuit verification. In *Proc. Intl. Conference on Computer-Aided Design (ICCAD-97)*, pages 394–399, Nov 1997.

[11] C.A.J. van Eijk. Sequential equivalence checking without state space traversal. In *Proc. Conference on Design, Automation and Test in Europe (DATE-98)*, pages 618–623, Paris, France, March 1998.

[12] Markus Wedler, Dominik Stoffel, and Wolfgang Kunz. Improving structural fsm-traversal by constraint-satisfying simulation. In *Proc. IEEE Computer Society Annual Symposium on VLSI 2002 (ISVLSI 2002)*, pages 151–158, April 2002.