Property Checking via Structural Analysis

Jason Baumgartner[1], Andreas Kuehlmann[2], and Jacob Abraham[3]

[1] IBM Enterprise Systems Group, Austin, TX 78758
[2] Cadence Berkeley Labs, Berkeley, CA 94704
[3] The University of Texas, Austin, TX 78712

Abstract. This paper describes a structurally-guided framework for the decomposition of a verification task into subtasks, each solved by a specialized algorithm for overall efficiency. Our contributions include the following: (1) a structural algorithm for computing a bound of a state-transition diagram's diameter which, for several classes of netlists, is sufficiently small to guarantee completeness of a bounded property check; (2) a robust backward unfolding technique for structural target enlargement: from the target states, we perform a series of *compose-based* pre-image computations, truncating the search if resource limitations are exceeded; (3) similar to frontier simplification in symbolic reachability analysis, we use induction via *don't cares* for enhancing the presented target enlargement. In many practical cases, the verification problem can be discharged by the enlargement process; otherwise, it is passed in simplified form to an arbitrary subsequent solution approach. The presented techniques are embedded in a flexible verification framework, allowing arbitrary combinations with other techniques. Extensive experimental results demonstrate the effectiveness of the described methods at solving and simplifying practical verification problems.

1 Introduction

Due to the complexity of modern hardware designs, formal verification methods are finding increasing utilization to augment the coverage shortcomings of test-based validation approaches. There are two primary methodologies for the verification of safety properties. Inductive methods use a provided or computed invariant to prove that no initial state can reach a state that violates a property (a *target state*). State traversal techniques employ exact or approximate search to find a trajectory from an initial state to a target state; unreachability is proven if a search exhausts without finding such a trajectory. Because of their exponential complexity, exact state traversal techniques – whether symbolic or explicit – are applicable only to small or modestly-sized designs.

Numerous approximate techniques have been proposed to address the capacity limitations of exact state traversal. Overapproximating the set of reachable states is useful to prove a target unreachable if all target states remain outside the overapproximation, though cannot readily demonstrate reachability otherwise. For example, design partitioning [3] can be applied to overapproximate the set of reachable states by exploring components whose sizes are tractable for exact traversal. Similarly, the concept of a *free fence* [11] is suggested for proving the correctness of a property by *localization*.

Conversely, underapproximate techniques are useful to demonstrate reachability of targets, but are generally incapable of proving their unreachability. For example, bounded model checking (BMC) [1] is based upon a satisfiability check of a finite k-step unfolding of the target. If it can be proven that the diameter of the design is smaller or equal to k, BMC becomes complete and can thereby also prove unreachability. A similar underapproximate method is based upon a bounded backward unfolding of the design starting from the target. The unfolded structure comprises an enlarged target which can be used to either directly discharge the verification problem or to produce a new, simplified problem to be solved by a subsequent verification flow.

An inductive proof requires an invariant that implies the property. The base step of a k-step inductive proof checks that the invariant holds during the first k time-steps. The inductive step must then show that asserting the invariant during time-steps $i, \ldots, (i + k - 1)$ implies that it continues to hold at time-step $(i + k)$. The typical drawback of inductive schemes is the intrinsic difficulty in determining a powerful enough invariant that is inductive and also implies correctness of the property. However, for many practical problems, backward unfolding results in an inductive invariant after several steps.

In this paper we show how the above-mentioned techniques can be synergistically combined to exploit their individual strengths. We first calculate an overapproximation of the diameter of the design using a novel structural algorithm. This bound has shown significant practical utility in precluding "redundantly large" unfoldings, and obviating more costly unbounded proof techniques. We next alternate between SAT-based forward unfolding and inductive backward BDD-based unfolding to attempt to solve the property. The forward analysis is useful for quickly hitting shallow targets, and also efficiently discharges our induction hypothesis for the backward analysis. The backward analysis is useful for proving unreachability, and renders a simpler problem – the enlarged target – otherwise. This iteration terminates if the unfolding depth surpasses the estimated diameter or if some resource limitations are exceeded. In the first case unreachability is proven; in the latter the enlarged target is passed to a subsequent solution approach. As demonstrated by our experiments, our technique efficiently solves many practical targets, and otherwise offers a significant reduction capability, hence the enlarged target is often much easier to discharge than the original.

2 Netlists: Syntax and Semantics

Definition 1. A *netlist* is a tuple $N = \langle\langle V, E\rangle, G, T, Z\rangle$ comprising a directed graph with vertices V and edges $E \subseteq V \times V$, a semantic mapping from vertices to gate types $G : V \mapsto \mathit{types}$, and a set of *targets* $T \subseteq V$ correlating to a set of properties $AG(\neg t), \forall t \in T$. The function $Z : V \mapsto V$ is the initial value mapping.

Our gate *types* define a set of constants, primary inputs (nondeterministic bits), registers, and combinational gates with various functions, whose semantics are provided in Definition 2. The type *register* is our only sequential gate type; all others are combinational. The type of a gate may place constraints upon its incoming edge count – e.g., registers and inverters have an indegree of one. We assume that every directed cycle comprises one or more registers – i.e., there are no combinational cycles.

Definition 2. The *semantics of a netlist* N are defined in terms of semantic traces: $0, 1$ valuations to gates over time. We denote the set of all legal traces associated with a netlist by $P \subseteq [V \times \mathbb{N} \mapsto \{0,1\}]$, defining P as the subset of all possible functions from $V \times \mathbb{N}$ to $\{0,1\}$ which are consistent with the following rule. The value of gate v at time i in trace p is denoted by $p(v,i)$.

$$p(v,i) = \begin{cases} c & : v \text{is a constant vertex with value } c \in \{0,1\} \\ s^i_{v_p} & : v \text{is a primary input with sampled value } s^i_{v_p} \\ G_v\big(p(u_1,i),...,p(u_n,i)\big) & : v \text{is a combinational gate and } G_v = G(v) \\ p(u_1, i-1) & : v \text{is a register and } i > 0 \\ p\big(Z(v),0\big) & : v \text{is a register and } i = 0 \end{cases}$$

Term u_j denotes the source vertex of the j-th incoming edge to v, implying $(u_j, v) \in E$.

The initial values of a netlist represent the values that registers can take at time 0; note that this function is ignored for non-register types. For a set of vertices $U \subseteq V$, let $\mathit{regs}(U) = \{v \subseteq U : G(v) = \mathit{register}\}$, and let $\mathit{coi}(U)$ be the set of vertices in the cone of influence of U. We assume that $\mathit{coi}\big(Z(\mathit{regs}(V))\big)$ contains no registers. Furthermore, we do not allow combinational cycles, which makes Definition 2 well-formed. We say that target t is *hit* in trace p at time i if $p(t,i) = 1$. A *state* is a mapping from registers to $0, 1$ values. We refer to the set of states for which there exists a primary input mapping that hits $t \in T$ as the set of *target states*.

Definition 3. The *diameter*[1] of $U \subseteq V$, denoted by $d(U)$, is the minimum $d \in \mathbb{N}$ such that for any trace p and $\forall i, j \geq 0$, there exists a trace p' such that $p\big(\mathit{regs}(\mathit{coi}(U)), i\big) = p'\big(\mathit{regs}(\mathit{coi}(U)), i\big)$ and $p(U, i+j+d) = p'(U, i+k)$ for some $0 \leq k < d$.

In other words, if any state s' is reachable from state s, then s' is reachable in less than d steps from s. By this definition, the diameter of a purely combinational cone is 1; the diameter of mod c-counter is c. Clearly $AG(\neg t)$ iff $\forall p \in P. \bigwedge_{i=0}^{d(t)-1} \big(p(t,i) = 0\big)$.

[1] Our definition of diameter is generally one greater than the standard definition for graphs.

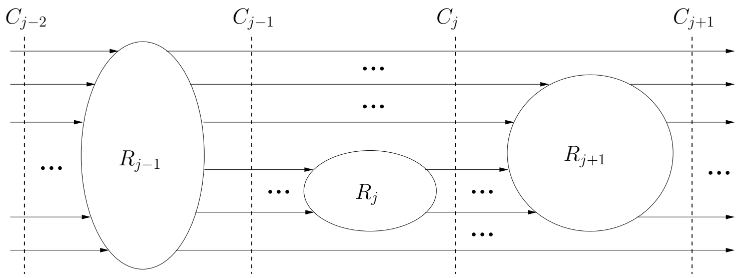

Fig. 1. Slice of TSAP structure

3 Diameter Approximation

In this section, we discuss a structural algorithm for computing an upper bound $\hat{d}(t) \geq d(t)$ on the diameter of target t. The goal is to find a practically tight overapproximation such that bounded search can be applied in a complete manner. A simple bound is $2^{|regs(coi(t))|}$, though this bound is often too weak to be of any practical value. Semantic approaches at obtaining a bound [1,13] are of formidable complexity – often greater than a BMC of the target itself.

There are two characteristics of practical netlists which may be exploited to structurally compute a tighter diameter bound. First, netlists seldom represent monolithic strongly connected graphs. Instead, they often comprise multiple maximal strongly connected components (SCCs); an approximation of diameter can thus be derived from an estimation of the individual SCC diameters. Second, although the overapproximate diameter of a component is generally exponential in its register count, several commonly occurring structures have much tighter bounds. For example, as proven in Theorem 1, the diameter of a single memory *row* comprising n registers is 2 instead of 2^n; acyclic registers only cause a linear, rather than multiplicative increase in diameter.

Definition 4. A *topologically sorted acyclic partitioning (TSAP)* of V into n components is a labeling $comp : V \mapsto \{1, 2, \ldots, n\}$ such that $\forall u, v \in V.\big((u, v) \in E \Rightarrow comp(u) \leq comp(v)\big)$.

Let $R_i = \{v : comp(v) = i\}$ denote the i-th component of a TSAP. Let $C_i = R_i \cup \{u : \exists v.\big((u, v) \in E \wedge u \in \bigcup_{j=1}^{i-1} R_j \wedge v \in \bigcup_{j=i+1}^{n} R_j\big)\}$. Set C_i adds to R_i the elements of lower-numbered components which fan out to higher-numbered components. For example, in Figure 1, some elements of component R_{j-1} are included in C_j and C_{j+1}, though no elements of R_j are included in C_{j+1} since no outgoing edges from R_j have sinks beyond R_{j+1}. We distinguish between the following specific types of components of a TSAP. Let x_i be a register vertex and y_i be the source of the incoming edge to x_i, denoting the present-state and next-state function of the corresponding register, respectively.

– A *combinational component* (*CC*) contains only non-register vertices.

- A *constant register component* (*NC*) contains only register vertices whose incoming edges are sourced by their outputs; i.e. $y_i = x_i$.
- An *acyclic register component* (*AC*) contains only register vertices whose incoming edges are inputs to the component.
- A *memory register component* (*MC*) is composed solely of a set of $r \times c$ registers and combinational gates, for $r \geq 1$ and $c \geq 1$. The next-state functions of the registers have the form: $y_{i,j} = (x_{i,j} \wedge \bigwedge_{k=1}^{w} \neg load_{i,k}) \vee \bigvee_{k=1}^{w} (data_{i,j,k} \wedge load_{i,k})$, for $1 \leq i \leq r$ and $1 \leq j \leq c$, where $data_{i,j,k}$ and $load_{i,k}$ are inputs to the component. Let $rows(R_i) = r$ for $MC\ R_i$.
- A *queue register component* (*QC*) is composed solely of a set of $r \times c$ registers and combinational gates, for $r > 1$ and $c \geq 1$. The next-state functions of the registers have the form: $y_{1,j} = (x_{1,j} \wedge \bigwedge_{k=1}^{w} \neg load_k) \vee \bigvee_{k=1}^{w} (data_{j,k} \wedge load_k)$; $y_{i,j} = (x_{i,j} \wedge \bigwedge_{k=1}^{w} \neg load_k) \vee (x_{i-1,j} \wedge \bigvee_{k=1}^{w} load_k)$, for $1 < i \leq r$ and $1 \leq j \leq c$, where $data_{j,k}$ and $load_k$ are inputs to the component. Let $rows(R_i) = r$ for $QC\ R_i$.
- All remaining components are *general components* (*GC*). Let $|regs(R_i)|$ denote the number of registers in $GC\ R_i$, which must be greater than 0. If there exists a combinational path from an input of R_i to any combinational gate $u \in R_i$, and $\exists v.\big((u,v) \in E \wedge v \notin R_i\big)$, we say that the *GC* is *Mealy*.

Note that *MC*s and *QC*s are generalized for w write ports. Further generalizations are possible, though we have found these adequate for commonly-occurring structures. *NC*s may have constant initial values (in which case they should be simplified by constant propagations) or symbolic initial values (e.g., implementing *forall* variables).

Our approximation of the diameter of a target t is based upon a TSAP of its *coi*. We prefer TSAPs with maximally-sized *AC*s, *NC*s, *MC*s, and *QC*s to ensure tighter approximation, which are obtained by first partitioning $coi(t)$ into maximal SCCs, then selectively clustering components so as not to introduce cycles.

Let $\epsilon(i) = 0$ if $i = 0$, or if $(C_{i-1} \cap C_i \neq \emptyset)$, or if R_i is a *Mealy GC*; otherwise $\epsilon(i) = 1$. Further, let $\delta(i) = 1 - \epsilon(i)$. Term $\epsilon(i)$ denotes whether R_i constitutes a cut between components $R_1 \ldots R_{i-1}$ and components $R_{i+1} \ldots R_n$, and $\delta(i) = \neg\epsilon(i)$. In Figure 1, only R_{j-1} constitutes a cut (provided that it is not a *Mealy GC*), hence $\epsilon(j-1) = 1$, whereas $\epsilon(j) = \epsilon(j+1) = 0$. The following formulas for computing $D(i)$ and $S(i)$ provide the key to our diameter estimation for R_i. Let $Q : R_i \mapsto \{CC, NC, AC, MC, QC, GC\}$ denote the type of component R_i.

$$D(i) = \begin{cases} 1 & : \ i = 0 \\ D(i-1) & : \ i > 0 \ \wedge \ Q(R_i) \in \{CC, NC, AC\} \\ D(i-1) \cdot \big(rows(R_i) + \delta(i)\big) & : \ i > 0 \ \wedge \ Q(R_i) \in \{MC, QC\} \\ D(i-1) \cdot \big(D_i - \epsilon(i)\big) + \epsilon(i) & : \ i > 0 \ \wedge \ Q(R_i) = GC \end{cases}$$

$$S(i) = \begin{cases} 0 & : \ i = 0 \\ S(i-1) & : \ i > 0 \ \wedge \ Q(R_i) \in \{CC, NC, GC\} \\ S(i-1) + \epsilon(i) & : \ i > 0 \ \wedge \ Q(R_i) \in \{MC, QC\} \\ S(i-1) + 1 & : \ i > 0 \ \wedge \ Q(R_i) = AC \end{cases}$$

Term D_i represents an upper-bound on the diameter $GC\, R_i$; clearly $2^{|regs(R_i)|}$ is conservative. We may use other estimation techniques, such as approximate reachability analysis [3], to find tighter bounds on the diameter of a GC. Note that the recurrence diameter [1] must be used with such an approach due to possible input constraints to the GC in the composed netlist.

Theorem 1. *The value $D(i) + S(i)$ is an upper-bound on the diameter of component R_i. Further, term $\hat{d}(t) = D(i) + S(i)$ is an upper-bound on the diameter of target t for $comp(t) = i$.*

Proof. Our proof is based upon the hypothesis that for cut C_i, any arbitrarily-ordered succession of c reachable valuations is producible within $c \cdot D(i) + S(i)$ time-steps. The theorem follows from assigning $c = 1$. We will prove this hypothesis by induction on i. The intuition behind this hypothesis is that component R_{i+1} may transition from each state only upon a distinct valuation of C_i. Therefore, in order to ensure that we attain an upper bound on the diameter of R_{i+1}, we generally must wait for a succession of $c = D(i+1)$ valuations to C_i.

Our base case has $i = 1$. If $Q(R_1) \in \{NC, CC\}$, we obtain $D(1) = 1$ and $S(1) = 0$. This result is correct, since any valuation producible by C_1 is producible every time-step due to its lack of sequential behavior. $Q(R_1)$ cannot be MC, QC, or AC since those types require other components to drive their inputs. Finally, if $Q(R_1) = GC$, then $D(1) = D_1$ which is an upper bound on the diameter of C_1 by definition, hence our proof obligation is satisfied.

We next proceed to the inductive step. If $Q(R_{i+1}) = NC$, then our result is correct by the base case analysis since such components have no inputs sourced by other components. If $Q(R_{i+1}) = CC$, then our result is correct by hypothesis, noting that R_{i+1} is a purely combinational function of C_i and primary inputs. If $Q(R_{i+1}) = AC$, then $D(i+1) = D(i)$ and $S(i+1) = S(i) + 1$. This result is correct since the initial values of an AC have semantic importance only at time 0, and since an AC merely delays some values of C_i by one time-step. If $Q(R_{i+1}) \in \{MC, QC\}$, then we obtain $D(i+1) = D(i) \cdot \big(rows(R_{i+1}) + \delta(i+1)\big)$ and $S(i+1) = S(i) + \epsilon(i+1)$. This result is correct by noting that it can take at most $c \cdot D(i) + S(i)$ time-steps to reach any possible succession of c valuations to C_i by hypothesis. If $\delta(i+1) = 1$, then C_i fans out to C_{i+2}, meaning that we generally must wait for $c = \big(rows(R_{i+1}) + 1\big)$ valuations to C_i to be sure that we have an upper bound on the diameter of C_{i+1}. If $\delta(i+1) = 0$, then we need only wait for $c = rows(R_{i+1})$ valuations to C_i, plus one extra time-step for the *load* to take effect upon C_{i+1}. Lastly, if $Q(R_{i+1}) = GC$, then $D(i+1) = D(i) \cdot \big(D_{i+1} - \epsilon(i+1)\big) + \epsilon(i+1)$ and $S(i+1) = S(i)$, where D_{i+1} is defined as an upper-bound on the diameter of R_{i+1}. For $\epsilon(i+1) = 0$ this result is obvious. Otherwise, note that any trace segment begins in one state of R_{i+1}, and $c = (D_{i+1} - 1)$ transitions, which must initiate within $c \cdot D(i) + S(i)$ time-steps, plus one for the final transition to complete, is sufficient to put R_{i+1} into any of its subsequently-reachable states. Hence $D(i+1) = D(i) \cdot (D_{i+1} - 1) + 1$ time-steps satisfies our obligation. □

We lastly observe that localization [11] has shown significant practical utility in quickly proving unreachability, and possibly enhancing reachability analysis. Localization consists of using a cut frontier (the *free fence*) to isolate an overapproximate localized *coi* for target t, yielding localized target t'. Term $\hat{d}(t')$ may be useful to enable bounded analysis, rather than often-more-expensive unbounded analysis, to solve t' since localized cones may often have structurally-calculatable shallow diameters. This technique may be useful not only to analyze certain localized cones more efficiently, but also to select better localized cones. For example, one may incrementally add gates to a localized cone to make it as large as possible – hence the overapproximation becomes tighter, as long as the incremental additions preserve a shallow diameter.

4 Target Enlargement

A k-step enlargement of target t incrementally computes the set of states that can reach t in k transitions. If an initial state becomes part of the enlarged target during this process, the target is proven reachable. Otherwise, if during the current enlargement step no new states are enumerated that have not been encountered in "shallower" steps, the target is proven unreachable. If $k \geq d(t)$ steps are performed without reaching an initial state, unreachability can be inferred. If at any step the computing resources exceed a given limit, the enlargement process is truncated and the verification problem is reformulated based upon the states enumerated during shallower steps (refer to Figures 2 and 3).

Target enlargement is based upon pre-image computation, for which there are three primary techniques: (1) transition-relation based methods [2,16,15,5], (2) transition-function based methods using the *constrain* operator [4], and (3) transition-function based methods using the *compose* operator [6]. We utilize the latter, since the set of registers in the support of each iteration of a target enlargement is often a small subset of those in the entire *coi* of the target. This precludes entailing unnecessary computational complexity, and well-suits our goal of rendering a simpler problem with as few registers as possible – the enlarged target – if the target is not solved during enlargement.

Figure 2 shows the pseudocode for our target enlargement algorithm. In step 1, $BSAT(N, t, i)$ denotes a SAT-based bounded check of target t using unfolding depth i. We use $BSAT$ to attempt to hit the target as well as to discharge our induction hypothesis for the the subsequent backward analysis. We use SAT rather than BDD-based analysis since the former is often more efficient for bounded analysis. If the overapproximate diameter is surpassed in the bounded search, we discharge the target in step 1b.

If $BSAT$ is inconclusive, we perform *compose-based* pre-image computations[2]. We apply early quantification of primary input variables to keep the intermediate BDD size small; as soon as the last composition which has a given primary

[2] We may alternatively iterate between $BSAT$ and pre-image computation with resource bounds.

```
Algorithm Enlarge($N, t, k, \hat{d}(t)$)
 1. for ( $i = 0$; $i < k$; $i$++ )
    (a) Apply $BSAT(N, t, i)$. If $t$ is reachable, then return REACHABLE with $BSAT$
        trace.
    (b) If $i = \hat{d}(t)$, then return UNREACHABLE.
 2. Build $BDD_0$ for $t$ over the primary inputs and registers in its transitive fanin.
 3. Existentially quantify primary inputs from $BDD_0$.
 4. for ( $i = 1$; $i \leq k$; $i$++ )
    (a) Compute MLP [12] schedule $S = (v_1, \ldots, v_s)$ for registers supporting $BDD_{i-1}$.
    (b) Rename all variables $v$ in $BDD_{i-1}$ to $v'$, forming $BDD_i$.
    (c) for ( $j = 1$; $j \leq |S|$; $j$++ )
          i. $BDD_i$ = bdd_compose($BDD_i, v'_j, f_{v_j}$), which substitutes $f_{v_j}$ in place of
             variable $v'_j$ in $BDD_i$, where $f_{v_j}$ is the BDD for the next-state function
             of $v_j$.
         ii. Perform early quantification of primary inputs from $BDD_i$.
        iii. Minimize $BDD_i$ with bdd_compact [8] using $BDD_0 \ldots BDD_{i-1}$ as don't
             cares.
         iv. If $BDD_i$ is too large, assign $k = i - 1$ and return $BDD_{i-1}$.
    (d) If $BDD_i$ is 0, then return UNREACHABLE.
 5. return $BDD_k$.
```

Fig. 2. Algorithm **Enlarge** for target enlargement

input i in its support is performed, we may quantify i. We utilize a modified MLP algorithm [12] for our quantification and composition scheduling. At each MLP scheduling step, we either schedule a composition, or "activate" an input to simplify future scheduling decisions – initially, all primary inputs are "inactive." Our goal is to minimize the lifetime of input variables, from activation until quantification, and to delay the introduction of registers. Each composition step eliminates one next-state register variable v', and introduces zero or more present-state register variables v and primary inputs. The following modifications of the MLP algorithm have proven to be the most useful:

– At each scheduling step, we schedule compositions of all registers with no inactive primary inputs in their support which introduce at most one register not already in the BDD support. Each such composition eliminates the corresponding v' variable from the BDD support, and adds at most one v variable to the support, which is typically beneficial for minimizing peak BDD size. We next schedule compositions of all registers with zero inactive, and nonzero active, inputs in their support, regardless of their register support, to force input quantification.
– If no register satisfies the above criteria, we instead activate an input. When choosing which input i to activate, we select one which is in the support of an unscheduled register with the fewest, though non-zero, inactive inputs in its support. Ties are broken to minimize the total number of registers not already in the BDD support which would need to be introduced before i could be quantified.

After each quantification, the intermediate BDD_i is simplified by the bdd_compact operator [8], using the BDDs of previous iterations as *don't cares*[3]. Effectively, this simplification attempts to prove inductiveness of the target; the corresponding hypothesis was previously discharged by $BSAT$. The resulting simplified BDD'_i satisfies the relation $BDD_i \setminus \bigcup_{j=0}^{i-1} BDD_j \subseteq BDD'_i \subseteq \bigcup_{j=0}^{i} BDD_j$ and $size(BDD'_i) \leq size(BDD_i)$, where $size(BDD_i)$ represents the node count of BDD_i. The bdd_compact operation cannot introduce new variables into the support of a BDD, but may eliminate some. Hence it is well-suited for our goal of minimizing the support of each pre-image step and thereby of the enlarged target. It is also this goal that prompts us to keep each BDD_i distinct. Using don't cares instead of constraints weakens our unreachability analysis, thus a fixed-point may never be reached. However, as demonstrated by our experimental results, many targets can be solved or significantly simplified which affirms the chosen trade-off of precision versus computational efficiency.

If the BDD size at any step exceeds a given limit, the enlargement process is truncated and the BDD of the previous iteration is returned. This prevents exceedingly large enlarged targets which could harm the subsequent verification flow. We have found that this approach – from structure to BDDs back to structure – is more effective in a flexible toolset than enlargement by purely structural transformation [14]. The latter does tend to yield large, redundant structures which may seriously hinder subsequent BDD- or simulation-based analysis methods. In contrast, our enlargement approach often reduces the size of the target cone and thus enhances any subsequent verification approach.

Using SAT rather than BDDs for an inductive proof may occasionally be more efficient. However, if unsuccessful, our BDD-based result can be reused to directly represent the *simplified* function of the k-step enlarged target. A similar reuse is not possible for a SAT-based method. In [7] it is proposed to apply cubes obtained during an inductive SAT call as "lighthouses" to enhance the ability to subsequently hit targets; such an incomplete approach, however, precludes the structural reductions of our technique.

5 Top-Level Algorithm

In this section we discuss the overall flow of our decomposition algorithm **Verify** which is illustrated in Figure 3. We first determine a limit on the number of enlargement steps and then call the algorithm **Enlarge** on target t. If **Enlarge** returns a *reachable* or *unreachable* solution, this result is returned. Otherwise a structure representing the enlarged target is added to N. This is performed by creating a new netlist N' which encodes the function of the BDD of the enlarged target. The output gate of N', denoted as t', is a combinational function over the

[3] A similar reachability-based approach would exploit states that can hit t within k time-steps as don't cares when assessing reachability of a state that can hit t in exactly k steps. With this observation, we may use these don't cares also to simplify the next-state functions of registers, which may further reduce the complexity for a subsequent verification flow.

registers in N. The composition of N and N', denoted as $N \parallel N'$, is then passed to a subsequent verification flow to attempt to solve t'. For example, we may next apply retiming and combinational optimizations [9] which have the potential to further reduce the netlist size before another application of **Verify** is attempted. This effectively applies the presented approach as a reentrant engine in a general *transformation-based verification* [9] flow. If a subsequent engine demonstrates unreachability of t', then t is also unreachable. If the subsequent verification flow hits t', we use simulation and another *BSAT* in step 6 to undo the effects of the enlargement on the trace. Because of its speed, a simulation preprocessing for eliminating easy-to-hit targets as step 0 may be useful in a robust toolset.

```
Algorithm Verify (N, t)
1. Determine a limit on number of enlargement steps
   $k = \min(\textit{user\_specified\_limit}, \hat{d}(t))$.
2. Invoke algorithm Enlarge($N, t, k, \hat{d}(t)$) to enlarge the target. If the problem is
   solved ($\textit{result} \in$ {REACHABLE, UNREACHABLE}) return result with any applicable trace.
3. Synthesize the BDD for the enlarged target from step 2 into netlist $N'$, compose $N'$
   onto $N$, and declare new target $t' \in V(N')$. This results in a new problem $(N \parallel N', t')$.
4. Utilize an arbitrary verification flow to attempt to solve $(N \parallel N', t')$.
5. If step 4 yields UNREACHABLE return UNREACHABLE.
6. If step 4 yields REACHABLE with a trace $p$, undo the effects of the enlargement
   by the following steps:
   (a) Complete $p$ over $N$ with simulation up to the first hit of $t'$ to obtain $p'$. This
       is needed because the cone-of-influence of $t$ and $t'$ may differ and $p$ may be
       only partial.
   (b) Cast $BSAT(N, t, k)$ from the last state of $p'$, which must be satisfiable, to
       obtain $p''$.
   (c) Concatenate $p''$ onto $p'$, overwriting the last time-step of $p'$ with the first
       time-step of $p''$, to obtain $p'''$.
   (d) return REACHABLE with $p'''$.
```

Fig. 3. Top-level algorithm **Verify**

Theorem 2. Algorithm **Verify** (N, t) is sound and complete.

The proof of Theorem 2 is straight-forward, and omitted due to space limitations.

6 Experimental Results

In this section we provide experimental results. All experiments were run on an IBM ThinkPad model T21 running RedHat Linux 6.2, with an 800 MHz Pentium III and 256 MB main memory. We set the peak BDD size to 2^{17} nodes, and capped *BSAT* (using a structural SAT solver [10]) to 10 seconds per target with an upper-bound of fifty steps.

Our first set of experiments were performed on the ISCAS89 benchmark netlists. The results are provided in Table 6. Since these benchmarks have no specified properties, we labeled each primary output of these netlists as a target. Column 1 gives the name of the benchmark and Column 2 provides the number of registers that are included in the various component types: constants (*NC*s), acyclic (*AC*s), table cells (*MC*s or *QC*s), and complex (*GC*s). Column 3 lists the average diameter $\hat{d}$ of all targets t with $\hat{d}(t) \leq 20$ and their count. These are candidates to be discharged with BMC. The bound of 20 was chosen arbitrarily as being typically efficient for bounded search. The next columns provide results for two distinct runs: first a standard run using the techniques as described in the previous sections, and second a "reduction-only" run which does not apply *BSAT* to solve the problem. Instead, if *BSAT* would solve the target in i steps, our enlargement is performed to depth $j = i - 1$; if $j < 1$, we only build BDD_0 in **Enlarge**. For the standard run in Column 4 we report the number of targets in the netlist, the number of targets which are hit, and the number of targets that are proven unreachable. The number of unreachable results proven with BDDs is provided in parenthesis. In Column 5 we report the accumulated size of the *coi*'s of unsolved targets in terms of the number of registers and primary inputs, and the number eliminated in the corresponding enlarged cones. Column 6 reports the average number of seconds spent per target, and the peak memory usage. For the reduction-only run we report *coi* sizes and reduction results (similar to Column 5) in Column 7.

There are several noteworthy points in Table 6. Our techniques solve most targets whether reachable or unreachable, regardless of netlist size – 1575 of 1615 targets are solved. Though the "difficulty" of these targets is unknown, this is an indication of the robustness of our approach. Many registers are non-complex: 26% are acyclic registers, and 6% are table cells. For netlists with unsolved targets, we achieve an average reduction per netlists of 5.3% in register count and 5.0% in primary input count, and a cumulative reduction of 12.2% for registers and 10.3% for primary inputs. A total of 381 targets have a diameter of less than 20. Our reduction-only run yields an average reduction per netlist of 13.9% in registers and 13.0% in primary inputs.

In Table 6 we provide a similar analysis for randomly-selected targets from the IBM Gigahertz Processor (GP). Most targets, 254 out of 284, are solved. A large fraction of the registers is non-complex: 1% are constants, 57% are acyclic, and 10% are table cells. A total of 91 targets have a diameter of less than 20. We achieve an average reduction per netlist of 12.1% in registers and 11.1% in primary inputs. The reduction-only run yields an average reduction per netlist of 54.9% in registers and 54.8% in primary inputs, and a cumulative reduction of 70.6% of registers and 69.5% of primary inputs.

We now discuss several netlists in more detail. Netlist I_IBBQn is a large table-based netlist. Forward reachability analysis of the optimized [10] cone of a single unreachable target with a diameter of three (comprising 442 registers and 134 primary inputs) requires 172.3 seconds and 25 MB with a MLP [12] algorithm, with sift variable ordering enabled and a random initial ordering. Our

Table 1. Experimental results for the ISCAS89 benchmarks

Model	*regs* in: *NC*;*AC*; *MC*+*QC*; *GC*	Avg. $\hat{d}$ < 20; Count	Standard Run			Reduction-Only Run
			|*T*|;Rch; Unrch (BDDs)	*regs* (inputs) Eliminated ; Sum	Time/|*T*| (*s*); Mem (*MB*)	*regs* (inputs) Eliminated ; Sum
PRO-LOG	0 ; 107 ; 1 ; 28	1.5 ; 6	73 ; 69 ; 4 (0)	0 (0); 0 (0)	0.07 ; 15	146 (126) ; 2044 (1438)
S1196	0 ; 18 ; 0 ; 0	3.3 ; 14	14 ; 14 ; 0 (0)	0 (0); 0 (0)	0.08 ; 12	24 (56); 88 (196)
S1238	0 ; 18 ; 0 ; 0	3.3 ; 14	14 ; 14 ; 0 (0)	0 (0); 0 (0)	0.08 ; 12	24 (56); 88 (196)
S1269	0 ; 9 ; 17 ; 11	4.0 ; 2	10 ; 10 ; 0 (0)	0 (0); 0 (0)	0.10 ; 15	289 (145); 296 (152)
S13207_1	0 ; 314 ; 128 ; 196	1.7 ; 51	152 ; 131 ; 12 (9)	26 (0); 527 (18)	1.02 ; 107	3155 (302); 24244 (2172)
S1423	0 ; 3 ; 16 ; 55	1.0 ; 1	5 ; 5 ; 0 (0)	0 (0); 0 (0)	0.13 ; 15	2 (0); 278 (69)
S1488	0 ; 0 ; 0 ; 6	0.0 ; 0	19 ; 18 ; 0 (0)	0 (0); 6 (8)	0.74 ; 23	0 (0); 114 (152)
S1494	0 ; 0 ; 0 ; 6	0.0 ; 0	19 ; 18 ; 0 (0)	0 (0); 6 (8)	0.89 ; 23	0 (0); 114 (152)
S1512	0 ; 0 ; 1 ; 56	0.0 ; 0	21 ; 10 ; 0 (0)	8 (8); 437 (283)	8.22 ; 24	135 (93); 837 (543)
S15850_1	0 ; 99 ; 124 ; 311	2.3 ; 112	150 ; 135 ; 8 (1)	451 (54); 1450 (174)	0.68 ; 63	2425 (321); 9683 (1301)
S208_1	0 ; 0 ; 0 ; 8	0.0 ; 0	1 ; 1 ; 0 (0)	0 (0); 0 (0)	0.27 ; 15	0 (0); 8 (10)
S27	0 ; 1 ; 2 ; 0	2.0 ; 1	1 ; 1 ; 0 (0)	0 (0); 0 (0)	0.23 ; 12	0 (0); 3 (4)
S298	0 ; 0 ; 1 ; 13	0.0 ; 0	6 ; 6 ; 0 (0)	0 (0); 0 (0)	0.12 ; 15	22 (6); 54 (18)
S3271	0 ; 6 ; 0 ; 110	7.0 ; 1	14 ; 14 ; 0 (0)	0 (0); 0 (0)	0.11 ; 15	0 (0); 1248 (339)
S3330	0 ; 103 ; 1 ; 28	1.0 ; 6	73 ; 73 ; 0 (0)	0 (0); 0 (0)	0.08 ; 15	146 (125); 2044 (1442)
S3384	0 ; 111 ; 0 ; 72	7.8 ; 4	26 ; 26 ; 0 (0)	0 (0); 0 (0)	0.09 ; 15	26 (25); 2587 (425)
S344	0 ; 0 ; 4 ; 11	3.3 ; 3	11 ; 10 ; 1 (0)	0 (0); 0 (0)	0.09 ; 15	6 (2); 129 (75)
S349	0 ; 0 ; 4 ; 11	3.0 ; 3	11 ; 10 ; 1 (0)	0 (0); 0 (0)	0.09 ; 15	3 (1); 126 (74)
S35932	0 ; 0 ; 0 ; 1728	0.0 ; 0	320 ; 320 ; 0 (0)	0 (0); 0 (0)	2.01 ; 105	0 (0); 331776 (11200)
S382	0 ; 6 ; 0 ; 15	0.0 ; 0	6 ; 6 ; 0 (0)	0 (0); 0 (0)	1.71 ; 15	34 (6); 96 (18)
S38584_1	0 ; 47 ; 4 ; 1375	1.0 ; 38	304 ; 301 ; 1 (0)	0 (0); 1377 (24)	1.54 ; 88	17925 (458); 105273 (2564)
S386	0 ; 0 ; 0 ; 6	0.0 ; 0	7 ; 7 ; 0 (0)	0 (0); 0 (0)	0.07 ; 14	0 (1); 42 (43)
S400	0 ; 6 ; 0 ; 15	0.0 ; 0	6 ; 6 ; 0 (0)	0 (0); 0 (0)	1.72 ; 15	34 (6); 96 (18)
S420_1	0 ; 0 ; 0 ; 16	0.0 ; 0	1 ; 1 ; 0 (0)	0 (0); 0 (0)	0.25 ; 15	0 (0); 16 (18)
S444	0 ; 6 ; 0 ; 15	0.0 ; 0	6 ; 6 ; 0 (0)	0 (0); 0 (0)	1.80 ; 15	34 (6); 96 (18)
S4863	0 ; 62 ; 0 ; 42	0.0 ; 0	16 ; 16 ; 0 (0)	0 (0); 0 (0)	0.09 ; 15	0 (0); 1664 (784)
S499	0 ; 0 ; 0 ; 22	0.0 ; 0	22 ; 22 ; 0 (0)	0 (0); 0 (0)	0.09 ; 16	0 (0); 484 (22)
S510	0 ; 0 ; 0 ; 6	0.0 ; 0	7 ; 4 ; 0 (0)	0 (0); 18 (57)	6.64 ; 25	0 (0); 42 (133)
S526N	0 ; 0 ; 1 ; 20	0.0 ; 0	6 ; 2 ; 0 (0)	8 (2); 64 (12)	10.44 ; 27	10 (2); 96 (18)
S5378	0 ; 115 ; 0 ; 64	2.0 ; 2	49 ; 47 ; 1 (1)	4 (0); 164 (33)	0.59 ; 26	165 (37); 7087 (1456)
S635	0 ; 0 ; 0 ; 32	0.0 ; 0	1 ; 0 ; 0 (0)	0 (0); 32 (2)	18.23 ; 15	0 (0); 32 (2)
S641	0 ; 7 ; 0 ; 12	1.0 ; 3	24 ; 23 ; 1 (1)	0 (0); 0 (0)	0.09 ; 15	64 (64); 319 (338)
S6669	0 ; 181 ; 0 ; 58	3.0 ; 37	55 ; 55 ; 0 (0)	0 (0); 0 (0)	0.08 ; 15	16 (0); 3061 (1466)
S713	0 ; 7 ; 0 ; 12	1.0 ; 3	23 ; 22 ; 1 (1)	0 (0); 0 (0)	0.10 ; 15	64 (64); 304 (323)
S820	0 ; 0 ; 0 ; 5	5.3 ; 18	19 ; 19 ; 0 (0)	0 (0); 0 (0)	0.23 ; 13	0 (0); 90 (324)
S832	0 ; 0 ; 0 ; 5	5.3 ; 18	19 ; 19 ; 0 (0)	0 (0); 0 (0)	0.25 ; 13	0 (0); 90 (324)
S838_1	0 ; 0 ; 0 ; 32	0.0 ; 0	1 ; 1 ; 0 (0)	0 (0); 0 (0)	0.29 ; 15	0 (0); 32 (34)
S9234_1	0 ; 45 ; 9 ; 157	1.2 ; 21	39 ; 37 ; 2 (0)	0 (0); 0 (0)	0.06 ; 16	146 (24); 1786 (317)
S938	0 ; 0 ; 0 ; 32	0.0 ; 0	1 ; 1 ; 0 (0)	0 (0); 0 (0)	0.33 ; 15	0 (0); 32 (34)
S953	0 ; 23 ; 0 ; 6	2.0 ; 3	23 ; 23 ; 0 (0)	0 (0); 0 (0)	0.13 ; 15	23 (8); 143 (288)
S967	0 ; 23 ; 0 ; 6	2.0 ; 3	23 ; 23 ; 0 (0)	0 (0); 0 (0)	0.14 ; 15	23 (8); 143 (288)
S991	0 ; 0 ; 0 ; 19	3.9 ; 17	17 ; 17 ; 0 (0)	0 (0); 0 (0)	0.08 ; 13	64 (564); 67 (629)

compose-based search requires 34.7 seconds and 16 MB for the same BDD conditions. However, because of its small diameter, the presented approach can solve the target using SAT in 0.46 seconds and 16 MB. After one step of enlargement, the cone drops to 380 registers and 132 primary inputs; the second step solves the target.

L_FLUSHn is a largely feed-forward netlist. For one target with 38 registers and 47 primary inputs, reachability analysis of the optimized target with MLP requires 1.20 seconds and 11 MB. Optimization [10] plus retiming [9] with MLP solves the target in 0.60 seconds with 13 MB. *Compose-based* search requires 0.50 seconds and 9 MB. Due to a shallow diameter of three, our techniques solve the target using SAT in 0.19 seconds with 9 MB. The first two steps of enlargement of this target reduce it to 4 then 2 registers, and 3 then 2 primary inputs, respectively. The third step hits the target.

Table 2. Experimental results for GP netlists

Model	*regs*: *NC*;*AC*; *MC*+*QC*; *GC*	Avg. $\hat{d}$ < 20; Count	Standard Run			Reduction-Only Run
			$\|T\|$;Rch; Unrch (BDDs)	*regs* (inputs) Eliminated ; Sum	Time/$\|T\|$ (s); Mem (MB)	*regs* (inputs) Eliminated ; Sum
CP_RAS	0 ; 279 ; 66 ; 315	0.0 ; 0	2 ; 2 ; 0 (0)	0 (0); 0 (0)	0.61 ; 19	1 (0); 554 (131)
CLB_CNTL	0 ; 29 ; 2 ; 19	0.0 ; 0	2 ; 2 ; 0 (0)	0 (0); 0 (0)	0.24 ; 15	0 (0); 84 (12)
CR_RAS	0 ; 96 ; 6 ; 329	0.0 ; 0	1 ; 0 ; 0 (0)	0 (0); 401 (99)	3.55 ; 24	0 (0); 401 (99)
D_DASA	0 ; 16 ; 81 ; 18	0.0 ; 0	2 ; 2 ; 0 (0)	0 (0); 0 (0)	0.24 ; 15	11 (17); 20 (25)
D_DCLA	0 ; 382 ; 1 ; 754	0.0 ; 0	2 ; 1 ; 1 (1)	0 (0); 0 (0)	7.65 ; 44	273 (67); 469 (133)
D_DUDD	0 ; 30 ; 28 ; 71	4.4 ; 5	22 ; 14 ; 8 (6)	0 (0); 0 (0)	1.15 ; 25	491 (353); 1009 (725)
I_IBBQn	0 ; 623 ; 1488 ; 0	2.9 ; 15	15 ; 8 ; 7 (0)	0 (0); 0 (0)	0.28 ; 60	190 (30); 2169 (437)
I_IFAR	0 ; 303 ; 11 ; 99	0.0 ; 0	2 ; 2 ; 0 (0)	0 (0); 0 (0)	0.31 ; 16	8 (0); 101 (35)
I_IFPF	11 ; 893 ; 44 ; 598	0.0 ; 0	1 ; 1 ; 0 (0)	0 (0); 0 (0)	2.72 ; 40	745 (152); 746 (154)
L3_SNP1	25 ; 529 ; 39 ; 82	0.0 ; 0	5 ; 4 ; 1 (0)	0 (0); 0 (0)	1.21 ; 22	7 (0); 595 (164)
L_EMQn	5 ; 146 ; 6 ; 66	0.0 ; 0	1 ; 0 ; 1 (1)	0 (0); 0 (0)	11.57 ; 18	127 (89); 127 (89)
L_EXEC	12 ; 421 ; 0 ; 102	0.0 ; 0	2 ; 2 ; 0 (0)	0 (0); 0 (0)	0.47 ; 18	433 (200); 433 (200)
L_FLUSHn	6 ; 198 ; 0 ; 4	3.7 ; 7	7 ; 6 ; 1 (0)	0 (0); 0 (0)	0.11 ; 12	128 (170); 165 (222)
L_INTRo	14 ; 143 ; 12 ; 5	2.9 ; 29	30 ; 24 ; 6 (0)	0 (0); 0 (0)	0.06 ; 12	750 (626); 830 (672)
L_LMQo	28 ; 690 4 ; 133	0.0 ; 0 0.0 ; 0	16 ; 0 ; 8 (8)	0 (0); 2592 (1512)	14.01 ; 39 14.01 ; 39	2568 (1512) 5160 (3024)
L_LRU	0 ; 142 ; 20 ; 75	0.0 ; 0	12 ; 5 ; 7 (7)	0 (0); 0 (0)	6.27 ; 19	721 (192); 721 (192)
L_PFQo	13 ; 1936 ; 18 ; 84	1.0 ; 1 1.0 ; 1	67 ; 0 ; 67 (66)	0 (0); 0 (0) 0 (0); 0 (0)	10.99 ; 77 10.99 ; 77	10318 (3036); 10318 (3036)
L_PNTRn	3 ; 228 ; 10 ; 11	2.0 ; 23	31 ; 0 ; 31 (8)	0 (0); 0 (0)	2.92 ; 19	1057 (1023); 1057 (1023)
L_PRQn	34 ; 366 ; 106 ; 265	2.0 ; 8	10 ; 0 ; 8 (2)	24 (8); 36 (12)	0.30 ; 19	42 (16); 54 (20)
L_SLB	3 ; 135 ; 6 ; 27	1.0 ; 1	3 ; 1 ; 2 (0)	0 (0); 0 (0)	0.16 ; 15	1 (1); 61 (29)
L_TBWKn	0 ; 202 ; 117 ; 14	0.0 ; 0	21 ; 1 ; 3 (3)	2 (0); 291 (238)	17.07 ; 26	36 (28); 342 (280)
M_CIU	0 ; 343 ; 10 ; 424	0.0 ; 0	6 ; 1 ; 5 (0)	0 (0); 0 (0)	0.24 ; 18	775 (60); 775 (60)
SIDECAR	3 ; 109 ; 32 ; 455	0.0 ; 0	1 ; 0 ; 0 (0)	1 (0); 137 (13)	18.64 ; 27	1 (0); 137 (13)
S_SCU1	1 ; 232 ; 4 ; 136	0.0 ; 0	3 ; 2 ; 1 (1)	0 (0); 0 (0)	0.66 ; 24	386 (142); 579 (213)
V_CACH	5 ; 94 ; 15 ; 59	0.0 ; 0	1 ; 0 ; 1 (1)	0 (0); 0 (0)	0.61 ; 16	86 (21); 86 (21)
V_DIR	6 ; 91 ; 13 ; 68	0.0 ; 0	2 ; 2 ; 0 (0)	0 (0); 0 (0)	0.20 ; 15	33 (16); 33 (16)
V_SNPM	65 ; 846 ; 134 ; 376	2.0 ; 1	2 ; 1 ; 1 (0)	0 (0); 0 (0)	1.27 ; 32	905 (266); 905 (266)
W_GAR	0 ; 159 ; 0 ; 83	1.0 ; 1	7 ; 6 ; 0 (0)	4 (0); 86 (37)	2.43 ; 20	4 (0); 500 (224)
W_SFA	0 ; 22 ; 0 ; 42	0.0 ; 0	8 ; 8 ; 0 (0)	0 (0); 0 (0)	0.08 ; 15	42 (21); 112 (56)

One target of netlist S15850_1 comprises 476 registers, primarily complex, and 55 primary inputs. MLP-based analysis is infeasible on this cone, even after optimization [10] plus retiming [9] which yields 397 registers. However, the first five steps of structural enlargement of this target reduce it to 475, 38, 36, 35, and finally 24 registers, and to 55, 55, 14, 13, and 13 primary inputs, respectively. MLP-based forward reachability hits the 5-step-enlarged target in 10 iterations with a combined effort of 2.5 seconds and 23MB. The only other approach that is able to hit this target is a 15-step *BSAT* which requires 7.3 seconds and 14MB. For an unreachable target BMC would not have been applicable. Traditional approaches of target enlargement would be ineffective on this netlist since they do not offer any reduction capability, without which the enlarged target remains infeasibly complex.

7 Conclusion

We have presented an efficient framework for decomposing a verification task into multiple subtasks. Our techniques are capable of solving or simplifying most problems, and comprise a useful component of a more general *transformation-based verification* toolset. We first calculate the number of time-steps k to use for bounded search via a novel structural algorithm for diameter overapproximation. For many practical netlists, this analysis yields a sufficiently small bound on the diameter to allow bounded model checking to discharge the proof. Otherwise, we iteratively perform SAT-based forward search and inductive *compose-based* backward search for *structural target enlargement*. If the property is not solved by the above, we construct a simpler enlarged target and pass it to a subsequent verification approach. While inherently performing a *temporal* decomposition of the verification task, our approach is capable of *spatially* reducing the target size and complexity, thus the enlarged target is often significantly simpler to discharge than the original. We use simulation and BMC to *complete* any traces obtained on the target-enlarged netlist for undoing the effects of the transformation. Extensive experimental results demonstrate the effectiveness of the proposed techniques in solving and simplifying problems.

References

1. Armin Biere, Alessandro Cimatti, Edmund M. Clarke, and Yunshan Zhu. Symbolic model checking without BDDs. In *5th International Conference on Tools and Algorithms for Construction and Analysis of Systems*, March 1999. 152, 154, 156
2. 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), April 1994. 157
3. H. Cho, G. Hachtel, E. Macii, B. Pleisser, and F. Somenzi. Algorithms for approximate FSM traversal based on state space decomposition. *IEEE Transactions on Computer-Aided Design*, 15(12), Dec. 1996. 151, 156
4. O. Coudert, C. Berthet, and J. C. Madre. Verification of sequential machines using Boolean functional vectors. In *IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, Nov. 1989. 157
5. Luca de Alfaro, Thomas A. Henzinger, and Freddy Y. C. Mang. Detecting errors before reaching them. In *Computer-Aided Verification*, July 2000. 157
6. Thomas Filkorn. Functional extensions of symbolic model checking. In *Computer-Aided Verification*, June 1991. 157
7. Malay K. Ganai. *Algorithms for Efficient State Space Search*. PhD thesis, University of Texas at Austin, May 2001. 159
8. Youpyo Hong, Peter A. Beerel, Jerry R. Burch, and Kenneth L. McMillan. Safe BDD minimization using don't cares. In *Proc. 34th ACM/IEEE Design Automation Conference*, June 1997. 158, 159
9. Andreas Kuehlmann and Jason Baumgartner. Transformation-based verification using generalized retiming. In *Computer-Aided Verification*, July 2001. 160, 162, 163
10. Andreas Kuehlmann, Malay K. Ganai, and Viresh Paruthi. Circuit-based Boolean reasoning. In *Proc. 38th ACM/IEEE Design Automation Conference*, June 2001. 160, 161, 162, 163
11. Robert P. Kurshan. *Computer-Aided Verification of Coordinating Processes*. Princeton University Press, 1994. 152, 157
12. In-Ho Moon, Gary D. Hachtel, and Fabio Somenzi. Border-block triangular form and conjunction schedule in image computation. In *Formal Methods in Computer-Aided Design*, Nov. 2000. 158, 161
13. Mary Sheeran, Satnam Singh, and Gunnar Stalmarck. Checking safety properties using induction and a SAT-solver. In *Formal Methods in Computer-Aided Design*, Nov. 2000. 154

14. Poul F. Williams, Armin Biere, Edmund M. Clarke, and Anubhav Gupta. Combining decision diagrams and SAT procedures for efficient symbolic model checking. In *Computer-Aided Verification*, July 2000. 159
15. C. Han Yang and David L. Dill. Validation with guided search of the state space. In *Proc. 35th ACM/IEEE Design Automation Conference*, June 1998. 157
16. Jun Yuan, Jian Shen, Jacob Abraham, and Adnan Aziz. On combining formal and informal verification. In *Computer-Aided Verification*, June 1997. 157