Maximal Input Reduction of Sequential Netlists via Synergistic Reparameterization and Localization Strategies

Jason Baumgartner and Hari Mony

IBM Systems & Technology Group, Austin TX 78758

Abstract. Automatic formal verification techniques generally require exponential resources with respect to the number of primary inputs of a netlist. In this paper, we present several fully-automated techniques to enable maximal input reductions of sequential netlists. First, we present a novel min-cut based localization refinement scheme for yielding a *safely* overapproximated netlist with minimal input count. Second, we present a novel form of reparameterization: as a trace-equivalence preserving structural abstraction, which provably renders a netlist with input count at most a constant factor of register count. In contrast to prior research in reparameterization to offset input growth during symbolic simulation, we are the first to explore this technique as a structural transformation for sequential netlists, enabling its benefits to general verification flows. In particular, we detail the synergy between these input-reducing abstractions, and with other transformations such as retiming which – as with traditional localization approaches – risks substantially increasing input count as a byproduct of its register reductions. Experiments confirm that the complementary reduction strategy enabled by our techniques is necessary for iteratively reducing large problems while keeping both proof-fatal design size metrics – register count and input count – within reasonable limits, ultimately enabling an efficient automated solution.

1 Introduction

Automatic formal verification techniques generally require exponential resources with respect to the number of primary inputs of a netlist. For example, the size of a transition relation may grow exponentially with respect to the number of inputs, in addition to state elements. The initial state encoding of a netlist may also grow exponentially complex with respect to the number of inputs used to encode that relation. Symbolic simulation – used for bounded model checking and induction – may require exponential resources with respect to the number of inputs multiplied by the unfolding depth. A large input count may thus render proof as well as falsification efforts inconclusive, and may arise through being inherent in the design under verification, or as the byproduct of a particular verification strategy – e.g., as the result of a register-reducing transformation such as localization or retiming, which are often critical to ensure that a large *register count* is not the fatal verification bottleneck on larger industrial designs.

Several techniques have been proposed to reduce the number of primary inputs of a netlist for specific verification algorithms. For example, the approach of enhancing

D. Borrione and W. Paul (Eds.): CHARME 2005, LNCS 3725, pp. 222–237, 2005.

symbolic simulation through altering the parametric representation of subsets of the input space via manual case splitting strategies was proposed in [1,2]. The approach of automatically reparameterizing unfolded variables during symbolic simulation to offset their increase over unfolding depth has also been explored, e.g., in [3,4,5]. Various approaches for reducing variable count in symbolic reachability analysis have been proposed, e.g., through early quantification of inputs from the transition relation [6], enhanced by partitioning [7] or overapproximation [8]. In this paper, we propose a novel set of fully-automated techniques to enable maximal input reductions of sequential netlists for *arbitrary* verification frameworks.

First, we present a novel form of reparameterization: as a sound and complete structural abstraction. Unlike prior research in reparameterization which focused upon its enhancement to symbolic simulation [3–5], we are the first to explore the use of this technique as a structural transformation for sequential netlists, enabling it to benefit general verification flows. We prove that this technique renders a netlist with input count at most a constant factor of register count, and discuss how it heuristically reduces register count and correlation. These reductions may thereby enhance the application of a variety of verification and falsification algorithms, including semi-formal search, reachability analysis, and emulation. Algorithm-specific reparameterization techniques may be complementarily applied to the resulting abstracted netlist, and are likely to benefit from its reduction. For example, in our experience, it is almost always worth performing aggressive reductions on the sequential netlist prior to unfolding to achieve a *simplify once, unfold many* optimization to bounded analysis, in this case reducing the amount of costly reparameterization needed over unfolding depth. More significantly, our structural reparameterization enables synergistic application with various other transformations such as retiming [9] and localization [10], which overall are capable of yielding dramatic iterative netlist reductions.

Second, we present a novel min-cut based localization refinement scheme tuned for yielding an overapproximated netlist with minimal input count. Unlike traditional localization approaches which refine entire next-state functions or individual gates, ours augments gate-based refinement by adding gates within a min-cut over the combinational logic driving the localized cone to minimize localized input count. A related approach was proposed in [10,11], where register-based localization is followed by the insertion of cut-points to a combinational min-cut between the localized inputs and sequentially-driven logic. Our approach improves upon this work as follows. **(1)** Whereas their approach *eliminates* gates from the logic deemed necessary by the refinement process, hence is prone to introducing spurious counterexamples to the already-overapproximated netlist, ours *adds* gates to the chosen refinement hence avoids this secondary overapproximation risk. **(2)** Their approach resolves spurious counterexamples caused by the secondary cut-point insertion by adding registers to the localized logic, whereas ours performs refinement at the level of individual gates, avoiding the addition of unnecessary registers while preserving minimal input count. **(3)** The ability of our technique to safely inject cut-points to sequentially-driven localized logic theoretically and practically improves upon the input reductions possible with their localization approach. Additionally, our approach is the first to address the use of localization to simplify initial value cones. Complex initial value cones arise in

a variety of applications such as retiming, and may otherwise be fatal to proof analysis. Localization refinement algorithms may be used to reduce the input count of these cones, effectively attempting to overapproximate the *initial states* of the design in a property-preserving manner.

Third, we detail the synergy that these reparameterization and localization transformations have with each other, and also with other transformations such as retiming and redundancy removal [12–14]. For example, the former approaches break interconnections in the design and reduce correlation among its registers, enabling greater register reductions through subsequent retiming and localization. Retiming and localization eliminate registers which constitute bottlenecks to the reduction potential of reparameterization, enabling greater input reductions through subsequent reparameterization. Retiming and localization are powerful techniques for reducing register count, which is indeed a critical step in enabling automated proofs on larger netlists. However, these techniques often entail a dramatic proof-fatal increase in input count as a byproduct of their register reductions, which has been our primary motivation for the development of the techniques presented in this paper. We have often found in practice that the iterative application of such register-reducing and input-reducing transformations constitutes a necessary strategy to enable automated proofs on complex industrial designs.

The rest of this paper is organized as follows. In Section 2, we introduce various formalisms used throughout the paper; the reader well-versed in such notation may wish to skip this section. In Section 3, we discuss our structural parametric abstraction. In Section 4, we present our min-cut based localization refinement scheme. In Section 5, we detail synergies between these two transformations and various others. In Section 6, we present experimental results to illustrate the power and synergy of these techniques in reducing netlist size. In Section 7, we conclude this work.

2 Formalisms

Definition 1. A *netlist* is a tuple $N = \langle\langle V, E\rangle, G, T, Z\rangle$ comprising a finite directed graph with vertices V and edges $E \subseteq V \times V$, a semantic mapping from vertices to gate types $G : V \mapsto types$, 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 verification problem is represented entirely as a netlist, comprising the *design under verification*, its *environment*, and its *property automata*. Our gate $types$ define a set of primary inputs, registers (our only sequential gate type), and combinational gates with various functions, including constants. The type of a gate may place constraints upon its incoming edge count – e.g., each register has an indegree of one (whose source gate is referred to as its *next-state function*); primary inputs and constants have an indegree of zero. We denote the set of inputs as $I \subseteq V$, and the set of registers as $R \subset V$.

Definition 2. The *combinational fanin* of gate set U is defined as $\bigcup_{u \in U}$ *cfi*(u), where *cfi*(u) is defined as u if $u \in R$, else $u \cup$ *combinational fanin*$\big(\{v : (v, u) \in E\}\big)$.

Definition 3. The *semantics of a netlist* N are defined in terms of semantic traces. 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 functions from $V \times \mathbb{N}$ to $\{0, 1\}$ which are consistent with

the following rule. Term u_j denotes the source vertex of the j-th incoming edge to v, implying that $(u_j, v) \in E$. The value of gate v at time i in trace p is denoted by $p(v, i)$.

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

The initial values of a netlist represent the values that registers can take at time 0. We disallow registers from appearing in the combinational fanin of any initial value cones. We additionally disallow combinational cycles, which makes Definition 3 well-formed.

Definition 4. Gate sets $A \subseteq V$ and $A' \subseteq V'$ of netlists N and N', respectively, are said to be *trace equivalent* iff there exists a bijective mapping $\psi : A \mapsto A'$ such that:

- $\forall p \in P. \exists p' \in P'. \forall i \in \mathbb{N}. \forall a \in A.\ p(a,i) = p'\big(\psi(a), i\big)$
- $\forall p' \in P'. \exists p \in P. \forall i \in \mathbb{N}. \forall a \in A.\ p(a,i) = p'\big(\psi(a), i\big)$

Definition 5. A *cut of a netlist* is a partition of V into two sets: $\mathcal{C}$ and $\overline{\mathcal{C}} = V \setminus \mathcal{C}$. A cut induces two sets of *cut gates* $V_{\mathcal{C}} = \{u \in \mathcal{C} : \exists v \in \overline{\mathcal{C}}.\big(((u,v) \in E) \vee (v \in R \wedge u = Z(v))\big)\}$, and $V_{\overline{\mathcal{C}}} = \{u \in \overline{\mathcal{C}} : \exists v \in \mathcal{C}.\big(((u,v) \in E) \vee (v \in R \wedge u = Z(v))\big)\}$.

One may visualize a cut of netlist N as the composition [15] of netlists $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$, with $V_{\mathcal{C}}$ denoting inputs to $N_{\overline{\mathcal{C}}}$ which are closed under the composition, and with $V_{\overline{\mathcal{C}}}$ denoting inputs to $N_{\mathcal{C}}$ which are closed under the composition.

Definition 6. An *s-t cut* is a cut seeded with vertex sets $s \subseteq \mathcal{C}$ and $t \subseteq \overline{\mathcal{C}}$. An *s-t min-cut* refers to an *s-t cut* where $V_{\mathcal{C}}$ is of minimal cardinality.

Algorithmically, when computing an *s-t min-cut*, sets s and t will be selected according to some application-specific criteria, and provided as constraints to the min-cut solver. The structural reparameterization technique that we will introduce in Section 3 and the min-cut based localization technique that we will introduce in Section 4 both utilize an *s-t min-cut* algorithm for optimality. However, they use the result for different purposes, hence have different criteria for selecting s and t as will be discussed in the respective sections. Numerous algorithms have been proposed for the efficient computation of *s-t min-cuts*, for example, the *augmenting path* algorithm [16]. It is noteworthy that the optimality of our techniques is independent of the chosen algorithm, and of the chosen min-cut if multiple cuts of minimal cardinality exist.

3 Structural Parametric Abstraction

In this section we discuss our structural reparameterization technique. We prove the correctness and optimality of this fully-automated abstraction, and discuss the algorithms used for performing the abstraction as well as for lifting abstract traces to ones consistent with the original netlist.

Definition 7. Consider a cut $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ of netlist N where $N_{\mathcal{C}}$ comprises inputs and combinational logic but no registers or target gates. A *structural reparameterization* of N is a netlist $N' = N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ such that $V_{\mathcal{C}}$ of N is trace-equivalent to $V'_{\mathcal{C}}$ of N' under the bijective mapping implied by the composition onto $N_{\overline{\mathcal{C}}}$.

1. Compute a cut $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ of N using an *s-t min-cut* algorithm, specifying the inputs as s, and the initial value gates, next-state function gates, registers, and target gates as t.
2. Compute the range of the cut as the set of minterms producible at $V_{\mathcal{C}}$ as a function of the registers in its combinational fanin.
3. Synthesize the range via the algorithm of Figure 2. The resulting netlist $N'_{\mathcal{C}}$ is combinational, and includes $V'_{\mathcal{C}}$ which is trace-equivalent to $V_{\mathcal{C}}$ under composition with $N_{\overline{\mathcal{C}}}$.
4. Replace each $v \in V_{\mathcal{C}}$ by its correspondent in $V'_{\mathcal{C}}$, yielding abstract netlist $N' = N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$.

Fig. 1. Structural parametric abstraction algorithm

Theorem 1. Let $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ be a cut of netlist N, and $N' = N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ be a structural reparameterization of N. The gates of $N_{\overline{\mathcal{C}}}$ in composition $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ are trace-equivalent to those in $N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ under the reflexive bijective mapping.

Proof. By Definition 5, any gate $u \in N_{\mathcal{C}}$ which sources an edge whose sink is in $N_{\overline{\mathcal{C}}}$, or is the initial value of a register in $N_{\overline{\mathcal{C}}}$, is an element of $V_{\mathcal{C}}$. Definition 3 thus implies that we may evaluate $N_{\overline{\mathcal{C}}}$ of N from valuations to $V_{\mathcal{C}}$ independently of valuations to gates in $N_{\mathcal{C}} \setminus V_{\mathcal{C}}$; similarly for N' and $V'_{\mathcal{C}}$. Since we compose each gate of $V_{\mathcal{C}}$ onto a trace-equivalent gate of $V'_{\mathcal{C}}$, this implies that $N_{\overline{\mathcal{C}}}$ of N is trace-equivalent to $N_{\overline{\mathcal{C}}}$ of N'. □

Theorem 1 is related to the result that simulation precedence is preserved under Moore composition [15]. This theorem establishes the soundness and completeness of our structural parametric abstraction: we wish to replace $N_{\mathcal{C}}$ by a simpler netlist which preserves trace-equivalence, while ensuring that every target is in $\overline{\mathcal{C}}$ and thereby preserving property checking. Numerous aggressive state-minimization techniques have been proposed for such purposes such as bisimulation minimization; however, such approaches tend to outweigh the cost of invariant checking [17]. Structural reparameterization is a more restrictive type of abstraction, though one which requires only lower-cost combinational analysis and is nonetheless capable of offering dramatic enhancements to the overall verification process.

We use the algorithm depicted in Figure 1 to perform the parametric abstraction. In Step 1, we compute an *s-t min-cut* of N. In Step 2, we compute the range of the cut using well-known algorithms as follows. For each $c_i \in V_{\mathcal{C}}$, we introduce a distinct parametric variable p_{c_i}, and we denote the function of c_i – over registers and primary inputs in its combinational fanin – as $f(c_i)$. The range of the cut is $\exists I. \bigwedge_{i=1}^{|V_{\mathcal{C}}|} \left(p_{c_i} \equiv f(c_i)\right)$. In Step 3, we compute the replacement logic for $N_{\mathcal{C}}$ from the range. The replacement gate r_{c_i} for c_i may be computed using the algorithm of Figure 2, assuming that the range is represented as a BDD.[1] Note that the approach of [18] may also be used for this synthesis; the algorithm of Figure 2 is merely an alternative included herein for completeness, implemented using common algorithms and applicable to BDDs with inverted edges. When completed, each produced gate r_{c_i} is trace-equivalent to c_i.

Figure 3a illustrates an example netlist, where we wish to reparameterize a cut at gates g_1 and g_2. Gate g_1 has function $i_1 \not\equiv r_1$, and g_2 has function $i_2 \vee (i_3 \wedge r_2)$, where

[1] In [5], it is proposed to perform the range computation for symbolic simulation using SAT; their technique is also applicable in our framework for structural reparameterization.

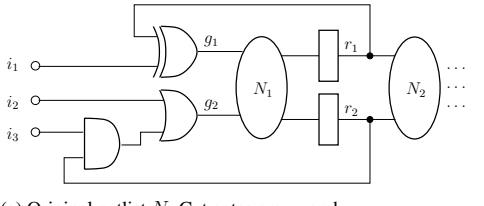
for ($i = 1, \ldots, |V_C|$) { // Process i in rank order of variables p_{c_i} in BDD *range*
$b_i = \exists p_{c_{i+1}}, \ldots, p_{c_n}.range$;
forced_0$_i$ = $\neg b_i|_{p_{c_i}=1}$; forced_1$_i$ = $\neg b_i|_{p_{c_i}=0}$;

// SYNTH creates logic gates from BDDs. It creates a distinct primary input to synthesize
// each p_{c_i}. It processes "forced" terms using a standard multiplexor-based synthesis,
// using $r_{c_1}, \ldots, r_{c_{i-1}}$ as selectors for nodes over $p_{c_1}, \ldots, p_{c_{i-1}}$ variables,
// and using registers as selectors for nodes over their corresponding variables.
// OR, AND, NOT create the corresponding gate types.
r_{c_i} = OR(SYNTH(forced_1$_i$), AND(SYNTH(p_{c_i})), NOT(SYNTH(forced_0$_i$)))); }

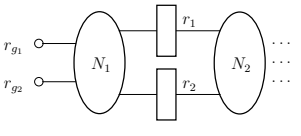
Fig. 2. Range synthesis algorithm

$i_1, i_2, i_3 \in I$ and $r_1, r_2 \in R$. The range of this cut is $\exists i_1, i_2, i_3.\big((p_{g_1} \equiv (i_1 \not\equiv r_1)) \wedge (p_{g_2} \equiv (i_2 \vee (i_3 \wedge r_2)))\big)$ which simplifies to $\top$. Replacement gates r_{g_1} and r_{g_2} are thus parametric inputs p_{g_1} and p_{g_2}, respectively, and r_1 and r_2 are eliminated from the support of V_C' as illustrated in Figure 3b. While this abstraction is primarily intended for input elimination, this example illustrates its heuristic ability to reduce *correlation* between registers, here breaking any correlation through N_1 between the next-state functions of r_1 and r_2 and their respective present-state values. Additionally, note that if N_2 does not depend upon either of these registers (say r_2), that register will be eliminated from the abstracted netlist by reparameterization alone, illustrating the heuristic register elimination capability of this technique. This correlation reduction synergistically enables greater structural reductions through other transformation techniques such as retiming, as will be discussed in Section 5.

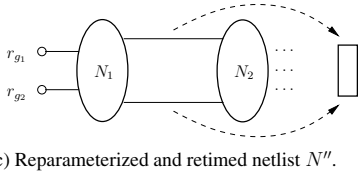
Theorem 2. The maximum number of primary inputs of the abstracted netlist N' generated by the algorithm of Figure 1 is $|T| + 2 \times |R|$.



(a) Original netlist N. Cut gates are g_1 and g_2.



(b) Reparameterized netlist N'.



(c) Reparameterized and retimed netlist N''.

Fig. 3. Reparameterization example

Proof. An *s-t min-cut* algorithm may be guaranteed to return a netlist cut with $|V_{\mathcal{C}}| \leq \min(|s|, |t|)$, as follows from the following analysis. The bound of $|s|$ follows from the existence of a cut where $\mathcal{C} = s$. Noting that the min-cut is seeded with $s = I$, this guarantees that our algorithm cannot increase input count. The bound of $|t|$ follows by automatically preprocessing the netlist to ensure that each element of t has indegree of one,[2] and selecting $\overline{\mathcal{C}} = t$. The seeded set t comprises the target gates, as well as the registers' initial value and next-state function gates – a set of cardinality $|T| + 2 \times |R|$. The resulting cut $V_{\mathcal{C}}$ may thus be upper-bounded in cardinality by $\min(|I|, |T| + 2 \times |R|)$. At most one input is required per element of $V'_{\mathcal{C}}$ in N', used in the synthesis of the parametric variable for that cut gate. The structural reparameterization thus replaces the $|I|$ inputs of $N_{\mathcal{C}}$ with the $|V'_{\mathcal{C}}|$ inputs of $N'_{\mathcal{C}}$.

Though we also add R to t, this does not alter the above bound because the only gates sourcing an edge into the registers – their next-state functions – are seeded into $\overline{\mathcal{C}}$. This inclusion serves only to facilitate compositional reasoning, in that registers in the support of the synthesized range will appear in N' – whereas $N_{\mathcal{C}}$ and $N'_{\mathcal{C}}$ are disjoint.

Let U represent the set of gates which contain an input in their combinational fanin. Straight-forward analysis will demonstrate that N' will have at most $(|T \cap U| + |\{r \in R : Z(r) \in U\}| + |\{r \in R : \exists u_1 \in U . (u_1, r) \in E\}|)$ primary inputs, which often yields a significantly tighter bound in practice. □

There are several noteworthy points relating to Theorem 2. First, note that at most one parametric input may be required per register for abstract initial values. This illustrates the duality between structural initial values and reachable state data, which is often represented with one variable per register. Certain techniques have been proposed which lock reachability data into structural initial values. For example, retiming [9] uses symbolic simulation to compute retimed initial values. If an input is retimed by k time-steps, there may be k unfolded copies of that input in the retimed initial values. Our parametric abstraction offsets this input amplification within the initial value data, similarly to how reparameterizing symbolic simulators operate [4,5]. As another example, one may underapproximate the reachable states (e.g., via symbolic simulation), then form a new netlist by altering the initial values of the original netlist to reflect the resulting state set [19,5]. Second, aside from initial values, note that at most one parametric input per register is necessary for abstract next-state functions. This bound has significant potential for enhancing a variety of verification paradigms, especially when coupled with synergistic register-reduction techniques (e.g., localization and retiming).

Because our abstraction preserves trace-equivalence of all targets in $N_{\overline{\mathcal{C}}}$, demonstrating that a target cannot be asserted within a bounded or unbounded time-frame on the abstracted netlist implies the same result on the original netlist. However, if a trace is obtained asserting a target in the abstracted netlist, that trace must be *lifted* to indicate an assertion of the corresponding target in the original netlist. Our algorithm for trace lifting is provided in Figure 4. In Step 1, we simulate the abstracted trace to ensure that

[2] This preprocessing entails "splitting" a gate v into gates v_1 and v_2. Gate v_1 has input connectivity and type identical to that of v, and fans out exclusively a new *buffer* gate v_2, which in turn inherits all fanout references of v (including fanout edges, as well as target and initial value references). A similar approach is used to ensure that $s \cap t = \emptyset$ in Step 1 of the algorithm of Figure 1, e.g., in case a next-state function is also an input.

1. Given partial trace p' of N', fully populate that trace up to the necessary length to assert the abstracted target, using binary simulation as per Definition 3 and injecting arbitrary values to any *don't cares* (unassigned values) of any primary inputs.
2. Cast a satisfiability check over $V_{\mathcal{C}}$ to obtain the same sequence of valuations as witnessed to $V'_{\mathcal{C}}$ in the populated trace p'. This check must be satisfiable since $V'_{\mathcal{C}}$ is trace-equivalent to $V_{\mathcal{C}}$ under composition with $N_{\overline{\mathcal{C}}}$, and yields trace p''.
3. Return trace p produced by composing values to $N_{\overline{\mathcal{C}}}$ from p' with values to $N_{\mathcal{C}}$ from p''.

Fig. 4. Parametric abstraction trace lifting algorithm

we have adequate deterministic valuations to $V'_{\mathcal{C}}$ and R' to enable the lifting. This is necessary because many verification algorithms produce *partial traces*, where certain valuations may be omitted for certain gates at certain time-steps. For example, in Figure 3b, parametric input r_{g_1} replaced gate g_1 of function $i_1 \not\equiv r_1$, eliminating r_1 from the support of $V'_{\mathcal{C}}$. The abstracted trace p' is thus less likely to include valuations to r_1. In order to lift p', and thereby provide the proper sequence of valuations to i_1 to yield an identical sequence of valuations to $V_{\mathcal{C}}$, the trace-lifting process must be aware of the valuations to r_1. After simulation populates the necessary valuations to p', a bounded satisfiability check in Step 2 will yield a trace p'' over $N_{\mathcal{C}}$ which provides the identical sequence of valuations to $V_{\mathcal{C}}$. This check tends to require only modest resources regardless of netlist size, since register valuations in p' effectively break the k-step bounded analysis into k one-step satisfiability checks, each injecting the netlist into the state reflected in the corresponding time-step of the trace. Step 3 splices p' and p'' together, producing a consistent trace over the original netlist asserting the original target. This algorithm is similar to those for lifting traces over localized netlists (e.g., [20]); its primary difference is the binary simulation step, which reduces satisfiability resources and is enabled due to the soundness *and completeness* of our abstraction as per Theorem 1.

Related Work. The approach of automatically reparameterizing unfolded variables during symbolic simulation to offset their increase over unfolding depth has been explored in prior work, e.g., in [3,4,5]. Overall, our technique is complementary to this prior work: by transforming the sequential netlist prior to unfolding, we enable a *simplify once, unfold many* optimization to bounded analysis reducing the amount of costly reparameterization needed over unfolding depth. Nonetheless, input growth over unfolding depth is inevitable; while our technique reduces this growth, a reparameterizing symbolic simulator may nonetheless be beneficial for analysis of the abstracted netlist.

Our approach is most similar to that of [4], which computes a cut of a logic cone, then parametrically replaces that cut by a simpler representation which preserves trace-equivalence. Unlike [4], which seeks to improve the efficiency of BDD-based combinational analysis hence retains all computations as BDDs, ours converts the reparameterized representation to gates. We are the first to propose the use of reparameterization as a structural reduction for sequential netlists, enabling its benefits to arbitrary verification and falsification algorithms, in addition to enabling dramatic iterative reductions with synergistic transformations as will be discussed in Section 5. Our approach also enables an efficient trace lifting procedure, unlike the approach of [4].

1. Begin with an initial abstraction $\mathcal{A}$ of N such that $T \subseteq \overline{\mathcal{C}}$.
2. Attempt to prove or falsify each target in $\mathcal{A}$.
3. If the target is proven unreachable, this result is valid for N; return this result.
4. If a trace is obtained asserting the target in $\mathcal{A}$, search for a corresponding trace in N. If one is found, return this result.
5. Otherwise, the trace over $\mathcal{A}$ is spurious. Identify a refinement of $\mathcal{A}$ – i.e., a set of gates to move from $\mathcal{C}$ to $\overline{\mathcal{C}}$ – to eliminate the spurious trace. Repeat Step 2 with the refinement.

Fig. 5. Localization refinement algorithm

Optimality. Note that the algorithm of Figure 2 uses a single parametric input per cut gate. One may instead attempt a more aggressive synthesis of the range, using $\lceil \log_2 m \rceil$ variables to directly select among the m possible minterms on a per-state basis (for maximal m), similarly to the approach proposed in [1]. While this may yield heuristically lesser input count, we have found this approach to be inferior in practice since $\lceil \log_2 m \rceil$ is often nearly equivalent to the cut-width due to the density of the range, and since the resulting encoding tends to be of significantly greater combinational complexity resulting in an increase in the analysis resources needed by virtually all algorithms, including simulation, satisfiability, and BDD-based algorithms (the latter was also noted in [2]).

We may readily eliminate the $|T|$ contribution of the bound proven in Theorem 2 by using the structural target enlargement technique of [21]. In particular, we may replace each target $t_i \in T$ by the synthesis of the characteristics function of the set of states for which there exists an input valuation which asserts that target, i.e., by $\exists I.f(t_i)$.

We utilize an *s-t min-cut* algorithm to ensure maximal input reductions as per Theorem 2. However, the range computation of the resulting cut may in cases be prohibitively expensive. It therefore may be desired to choose a cut with larger cardinality, weakening reduction potential in favor of computational efficiency – though iterative abstractions may be performed to ultimately converge upon the min-cut with lesser resources. In [4] it is proposed to reparameterize a *group* U of a candidate cut $V_{\mathcal{C}}$ to eliminate inputs I_U which are in the combinational fanin of U but not $V_{\mathcal{C}} \setminus U$. This reduction may be accomplished in our framework by selecting a cut of $V_{\mathcal{C}} = U \cup (I \setminus I_U)$, noting that any inputs in $V_{\mathcal{C}}$ will merely be replaced by other inputs, hence may effectively be treated as non-quantifiable variables when computing the range (similarly to registers in $N_{\overline{\mathcal{C}}}$). We have found that an efficient way to select suboptimal cuts for incremental abstraction is to compute min-cuts over increasing subsets of the desired cut, enabling the earlier abstractions to simplify later abstractions by iteratively decreasing $|I|$.

4 Min-cut Based Localization

Definition 8. A *localization* $\mathcal{A}$ of N is a netlist obtained by computing a cut of N such that $T \subseteq \overline{\mathcal{C}}$, and by replacing $V_{\mathcal{C}}$ by a set of primary inputs $V'_{\mathcal{C}}$ of netlist $N'_{\mathcal{C}}$, resulting in $\mathcal{A} = N'_{\mathcal{C}} \,\|\, N_{\overline{\mathcal{C}}}$. This replacement is referred to as *injecting cut-points to* $V_{\mathcal{C}}$.

Localization differs from the parametric abstraction of Section 3 since it renders an *overapproximated* netlist which can simulate the original, though the converse may not be true. Because the overapproximation may result in a spurious assertion of a target,

refinement is often used to tighten the overapproximation by increasing the size of $\overline{\mathcal{C}}$, e.g., using the algorithm of Figure 5. For larger netlists, the localization may contain many thousands of inputs when using traditional approaches of selecting $V_{\mathcal{C}}$ to comprise only registers and inputs (e.g., [10,22]), or of refining individual gates. This large input count tends to render the BDD-based reachability analysis which is commonly used for the proof analysis in Step 2 infeasible. In [10,11], this problem is addressed by further overapproximating the localization by computing an *s-t min-cut* between its inputs and sequentially-driven gates (i.e., gates which have a register in their combinational fanin), and injecting cut-points to the resulting cut gates to significantly reduce localized input count. When a trace is obtained on the post-processed localization, an attempt is made to map that trace to the original localization. If the mapping fails, in [11] various heuristics are proposed to select registers to add for the next localization refinement phase, instead of directly addressing the causal post-process cut-point injection.

The min-cut based localization refinement scheme we have developed to minimize input growth is depicted in Figure 6. In Step 1, a new localization $\mathcal{A}'$ is created from $\mathcal{A}$ by adding a set of refinement gates, which may be selected using any of the numerous proposed refinement schemes (e.g., [11,20]). For optimality, however, we have found that the refinement should be at the granularity of individual gates vs. entire next-state functions to avoid locking unnecessary complex logic into the localization. In Step 2, an *s-t min-cut* $\langle \mathcal{C}_1, \overline{\mathcal{C}}_1 \rangle$ is computed over N. In Step 3, the gates of $\overline{\mathcal{C}}_1$ are added to $\mathcal{A}'$ to ensure that $\mathcal{A}'$ has as few inputs as possible while containing the original refinement of Step 1. Note that the newly-added gates are all combinational because all registers not already in $\mathcal{A}'$ are seeded into s, hence cannot be in $\overline{\mathcal{C}}_1$ which is the set added to $\mathcal{A}'$.

Unlike the approach of [10,11], which *eliminates* gates from the logic deemed necessary by the refinement process hence is prone to introducing spurious counterexamples, our min-cut based localization *adds* combinational logic to the refinement to avoid this risk while ensuring minimal input count. While the overapproximate nature of localization may nonetheless result in spurious counterexamples, our approach avoids the secondary overapproximation of theirs which is done without refinement analysis to heuristically justify its validity. Our more general approach also avoids adding unnecessary registers during refinement, since it has the flexibility to select which combinational logic to include. In our experience, many refinements may be addressed solely by altering the placement of the cut within the combinational logic. Additionally, our approach is often able to yield a localization with *lesser* input count due to its ability to safely inject cut-points at gates which are sequentially-driven by registers included in the localization, which their register-based localization does not support and their combinational cut-point insertion disallows to minimize its introduction of spurious counterexamples. Finally, our approach enables localization to simplify complex initial value cones, as the inclusion of register r does not imply the inclusion of its initial value cone. Only the subset of that cone deemed necessary to prevent spurious counterexamples will be added during refinement. This initial-value refinement capability has not been addressed by prior research, despite its utility – e.g., when coupled with techniques which lock reachability data into initial values such as retiming [9].

In a transformation-based verification framework [9,23], one could attempt to reduce the input count of an arbitrarily-localized netlist by using the parametric abstrac-

1. Select a set of gates to add to the refinement $\mathcal{A}'$ of $\mathcal{A}$ using an arbitrary algorithm. Let $\langle \mathcal{C}', \overline{\mathcal{C}}' \rangle$ be the cut of N corresponding to $\mathcal{A}'$.
2. Compute an *s-t min-cut* $\langle \mathcal{C}_1, \overline{\mathcal{C}}_1 \rangle$ over N, with all gates in $\overline{\mathcal{C}}'$ as t, and $I \cup (R \cap \mathcal{C}')$ as s.
3. Add $\overline{\mathcal{C}}_1$ to the refinement $\mathcal{A}'$.

Fig. 6. Min-cut based abstraction refinement algorithm

tion of Section 3 instead of using a min-cut based localization refinement scheme, or of overapproximately injecting cut-points to a combinational min-cut thereof as proposed in [10]. As per Theorem 2, this synergistic strategy is theoretically able to reduce input count to within a factor of two of register count. This bound is only possible due to the ability of reparameterization to abstract sequentially-driven logic. In contrast, the min-cut approach of [10] is taken with t being the set of all sequentially-driven gates, which is often much larger than the set of registers – hence input count may remain arbitrarily larger than register count with their approach. Reparameterization is thus a superior input-elimination strategy compared to the cut-point insertion of [10], and has the additional benefit of retaining soundness and completeness. Nevertheless, the dramatic input growth which may occur during traditional localization approaches often entails exorbitant resources for reparameterization to overcome on large netlists. We have therefore found that an input-minimizing localization scheme such as ours is necessary to safely minimize input growth *during* localization, to in turn enable the optimal input elimination of reparameterization with minimal resources.

5 Transformation Synergies

In a transformation-based verification (TBV) framework [9], various algorithms are encapsulated as *engines* which each receive a netlist, perform some processing on that netlist, then transmit a new, simpler netlist to a child engine. If a verification result (e.g., a proof or counterexample) is obtained by a given engine from a child engine, that engine must map that result to one consistent with the netlist it received before propagating that result to its parent – or suppress it if no such mapping is possible. Synergistic transformation sequences often yield dramatic iterative reductions – possibly several orders of magnitude compared to a single application of the individual techniques [23]. In this section we detail some of the synergies enabled and exploited by our techniques.

Theorem 2 illustrates that all register-reducing transformations (e.g., retiming [9], localization [10], redundancy removal [12,13,14], and structural target enlargement [21]) synergistically enable greater input reductions through structural reparameterization. For example, retiming finds a minimal-cardinality register placement to eliminate reparameterization bottlenecks caused by their arbitrary initial placement. Localization injects cut-points to the netlist, which when reparameterized enable reductions even at *deep* gates which previously had no inputs in their combinational fanin. Redundancy removal may enable *s-t min-cut* algorithms to identify smaller-cardinality netlist cuts.

In addition to its input reductions, structural reparameterization reduces register correlation as per Figure 3b. As with redundancy removal, this often enables subsequent localization to yield greater reductions, since the heuristic abstraction algorithms are less likely to identify unnecessary registers as being required to prevent spurious counterexamples. We have found iterative localization and reparameterization strategies to be critical to yield adequate simplifications to enable a proof *or* a counterexample result on many complex industrial verification problems. The concept of iterative localization strategies was also proposed in [22], leveraging the heuristics inherent in the SAT algorithms used for the abstraction to identify different subsets of the netlist as being necessary across the nested localizations, in turn enabling iterative reductions. Our TBV approach enables significantly greater reduction potential, since it not only allows the use of differing abstraction heuristics across nested localizations, but also allows arbitrary transformations to iteratively simplify the netlist between localizations to *algorithmically – not merely heuristically* – enable greater localization reductions. In cases, the result enabled through our iterative reductions was a *spurious* localization counterexample which could be effectively used by the causal prior localization engine for refinement. This illustrates the utility of our synergistic transformation framework for the generation of complex counterexamples for abstraction refinement, enabling a more general refinement paradigm than that of prior work, e.g., [10,11,22].

Retiming is limited in its reduction potential due to its inability to alter the register count of any directed cycle in the netlist graph, and its inability to remove all registers along *critical paths* of differing register count between pairs of gates [24]. Both reparameterization and localization are capable of eliminating such paths, enabling greater register reductions through retiming. This is illustrated in Figure 3b, where reparameterization eliminates the directed cycles comprising r_1 and r_2, enabling a subsequent retiming to eliminate those registers in Figure 3c. Retiming has the drawback of increasing input count due to the symbolic simulation used to calculate retimed initial values [9]. Both reparameterization and our min-cut based localization are capable of offsetting this input growth, enabling retiming to be more aggressively applied without risking a proof-fatal input growth, as we have otherwise witnessed in practice.

6 Experimental Results

In this section we provide experimental results illustrating the reduction potential of the techniques presented in this paper. All experiments were run on a 2GHz Pentium 4, using the IBM internal transformation-based verification tool *SixthSense*. The engines used in the experiments are as follows; each performs a cone-of-influence reduction.

- **COM**: a BDD- and SAT-based combinational redundancy removal engine [13].
- **RET**: a min-area retiming engine [9].
- **CUT**: a structural reparameterization engine as per Section 3.
- **LOC**: a min-cut based localization engine as per Section 4.

We present several sets of experiments in Table 1 to illustrate the power of and synergy between these engines. The first column indicates the name of the benchmark and the size metric being tracked in the corresponding row. The second reflects the size of

Table 1. Synergistic transformation experiments

S4863 [12]	**Initial**	**COM**	**RET**	**COM**	**CUT**		**Initial**	**COM**	**CUT**	**RET**					**Resources**
Registers	101	101	37	37	21		101	101	34	0					**1 sec**
Inputs	49	49	190	190	37		49	49	21	21					**34 MB**
S6669 [12]	**Initial**	**COM**	**RET**	**COM**	**CUT**		**Initial**	**COM**	**CUT**	**RET**					
Registers	303	186	49	49	0		303	186	138	0					**1 sec**
Inputs	80	61	106	81	40		80	61	40	40					**35 MB**
SMM	**Initial**	**COM**	**LOC**	**CUT**	**LOC**	**CUT**	**LOC**	**CUT**							
Registers	36359	33044	760	758	464	167	130	129							**229 sec**
Inputs	261	71	2054	666	366	109	135	60							**291 MB**
MMU	**Initial**	**COM**	**LOC**	**CUT**	**LOC**	**CUT**	**RET**	**COM**	**CUT**						
Registers	124297	67117	698	661	499	499	133	131	125						**1038 sec**
Inputs	1377	162	1883	809	472	337	1004	287	54						**386 MB**
RING	**Initial**	**COM**	**LOC**	**CUT**	**RET**	**COM**	**CUT**	**LOC**	**CUT**	**LOC**	**CUT**	**LOC**	**CUT**		
Registers	20692	19557	266	262	106	106	106	65	65	49	48	47	35		**745 sec**
Inputs	2507	2507	568	280	726	587	480	452	376	330	263	259	64		**240 MB**
BYPASS	**Initial**	**COM**	**LOC**	**CUT**	**LOC**	**CUT**	**LOC**	**CUT**	**LOC**	**CUT**	**LOC**	**CUT**	**LOC**	**CUT**	
Registers	11621	11587	311	306	265	265	216	212	164	154	127	124	101	95	**240 sec**
Inputs	432	410	501	350	333	254	248	216	203	156	154	123	110	79	**175 MB**

the original netlist; phase abstraction [25] was used to preprocess the industrial examples. The successive columns indicate the size of the problem *after* the corresponding transformation engine (indicated in the row labeled with the benchmark name) was run.

The first two examples in Table 1 are sequential equivalence checking proof obligations of SIS-optimized ISCAS89 benchmarks from [12]. The first presented flow demonstrates how **CUT** offsets the increase in input count caused by **RET**, and also the register reduction potential of **CUT** itself. The second flow additionally illustrates how reparameterization enhances the register-reduction ability of **RET**, enabling retiming to eliminate *all* registers from both benchmarks. **CUT** was able to eliminate significant register correlation – and thereby critical paths – in these benchmarks due to logic of the form $(i_1 \not\equiv r_1)$ and $i_2 \vee (i_3 \wedge r_2)$ as illustrated in Figure 3.

The remaining four examples are difficult industrial invariant checking problems. SMM and MMU are two different memory management units. RING validates the prioritization scheme of a network interface unit. BYPASS is an instruction decoding and dispatch unit. These results illustrate the synergistic power of iterative reparameterization and localization strategies, coupled with retiming, to yield dramatic incremental netlist reductions. The resulting abstracted netlists were easily discharged with reachability analysis, though otherwise were too complex to solve with reachability or induction. In SMM, the first **LOC** reduces register count by a factor of 43, though increases input count by a factor of 29 to 2054. Without our min-cut based localization, this input growth is even more pronounced. Refining entire next-state functions as per [10] yields 29221 inputs; their combinational cut-point injection may only eliminate 54 of these, as most of the logic is sequentially driven. **CUT** could eliminate 28514 of these, modulo resource limitations. If refining individual gates, we obtain 2755 inputs. In practice, we often witness an even more pronounced input growth through gate-based refinement (e.g., 3109 vs. 1883 inputs for MMU). In MMU, **LOC** and **CUT** enable a powerful **RET** reduction with input growth which is readily contained by a subsequent **CUT**. RING is a difficult example which **LOC** and **CUT** alone were unable to adequately reduce to enable reachability. **RET** brought register count down to an adequate level,

Table 2. Input counts with and without structural reparameterization prior to unfolding

Benchmark	$\|R\|$	$\|I\|$ Orig.	$\|I'\|$ Reparam.	$\|R\| \leq \|I\|$ Unfold Depth	$\|R'\| \leq \|I'\|$ Unfold Depth	$\|I\|$ Unfold Depth 25	$\|I'\|$ Unfold Depth 25	$\|I\|$ Unfold Depth 100	$\|I'\|$ Unfold Depth 100
LMQ	345	189	135 (29%)	6	8	3884	2735 (30%)	17309	12111 (30%)
DA_FPU	6348	534	240 (57%)	24	39	7038	3120 (56%)	47088	21120 (55%)
SQMW	13583	1271	421 (67%)	23	47	16356	4538 (72%)	111681	36113 (68%)

though increased input count substantially due to complex retimed initial values. A single **CUT** was unable to contain that input growth with reasonable resources, though the ability to safely overapproximate the initial value cones with **LOC** iteratively and synergistically enabled **CUT** to eliminate all but a single input per initial value cone.

Table 2 illustrates the utility of structural reparameterization prior to unfolding. Column 2 and 3 illustrate the register and input count of the corresponding redundancy-removed [13] netlists. Column 4 provides the input count of the reparameterized netlist; the numbers in parentheses illustrate percent reductions. Columns 5 and 6 illustrate the unfolding depth at which input count exceeds register count with and without reparameterization. This is the unfolding depth at which one may wish to use reparameterization within the symbolic simulator to *guarantee* a reduction in variable count [5]. Note that this depth is significantly greater for the abstracted than the original netlist. Practically, a bug may be exposed by the symbolic simulator *between* these depths, hence our approach may preclude the need for reparameterization on the unfolded instance. More generally, the *simplify once, unfold many* optimization enabled by our abstraction reduces the amount of costly reparameterization necessary over greater unfolding depths, and enables shallower depths to be reached more efficiently due to lesser variable count. Another noteworthy point is that register count is significantly greater than input count in these netlists (as is common with industrial designs). Reparameterization within symbolic simulators operates on parametric variables for the *registers*, and on the *unfolded* inputs which become comparable in cardinality to the registers. In contrast, our structural reparameterization operates solely upon parametric variables for the *cut gates* (bounded in cardinality by the abstracted input count, in turn bounded by the original input count as per the proof of Theorem 2), and on the *original* inputs: a set of significantly lesser cardinality, implying significantly lesser resource requirements.

Note also that we did not perform more aggressive transformations such as localization and retiming on the examples of Table 2. As illustrated by Table 1, doing such is clearly a beneficial strategy in our synergistic transformation framework. However, the purpose of this table is to demonstrate how our structural reparameterization alone benefits symbolic simulation. The final columns of this table indicate input count with and without reparameterization for unfolding depths of 25 and 100.

7 Conclusion

We have presented several fully-automated techniques for maximal input reduction of sequential netlists for arbitrary verification flows. **(1)** We introduced a structural reparameterization technique, which provably reduces input count to a constant factor of register count. This technique also heuristically reduces register count and correlation. **(2)** We introduced a min-cut based localization refinement scheme for *safely* overap-

proximating a netlist through minimal cut-point insertion. We also detailed the synergy between these two abstractions, along with other transformations such as retiming.

Overall, the transformation synergy enabled by our techniques comprise their greatest benefit, capable of yielding dramatic iterative reductions unachievable by any standalone approach. For example, a single reparameterization application is able to reduce our RING benchmark from 2507 to 2109 inputs. A single application of our min-cut-based localization is able to reduce RING to 568 inputs (and prior localization approaches substantially increase its input count). Our iterative transformations, however, bring RING down to 64 inputs, ultimately enabling efficient reachability analysis. Such a profound reduction is obviously capable of yielding dramatic improvements to virtually all search algorithms, including reparameterizing symbolic simulators. We have made extensive use of these reduction strategies in a variety of complex industrial verification tasks, both for proofs and falsification, in many cases obtaining a conclusive result that was otherwise unattainable. For example, with many larger netlists, we have found that traditional localization and retiming strategies alone may ultimately reduce register count to a reasonable level, though result in an abstracted netlist with far too many inputs for an automated proof. The techniques presented in this paper were largely motivated by such complications, and have to a large extent solved these problems.

Acknowledgments. The authors wish to thank Geert Janssen, Viresh Paruthi, Robert Kanzelman, Jessie Xu, and Mark Williams for their contributions to the TBV system used in our experiments, and Koen van Eijk for providing the benchmarks of [12].

References

1. P. Jain and G. Gopalakrishnan, "Efficient symbolic simulation-based verification using the parametric form of Boolean expressions," *IEEE Transactions on CAD*, April 1994.
2. M. D. Aagaard, R. B. Jones, and C.-J. H. Seger, "Formal verification using parametric representations of Boolean constraints," in *Design Automation Conference*, June 1999.
3. V. Bertacco and K. Olukotun, "Efficient state representation for symbolic simulation," in *Design Automation Conference*, June 2002.
4. I.-H. Moon, H. H. Kwak, J. Kukula, T. Shiple, and C. Pixley, "Simplifying circuits for formal verification using parametric representation," in *FMCAD*, Nov. 2002.
5. P. Chauhan, E. M. Clarke, and D. Kroening, "A SAT-based algorithm for reparameterization in symbolic simulation," in *Design Automation Conference*, June 2004.
6. S. Mador-Haim and L. Fix, "Input elimination and abstraction in model checking," in *FMCAD*, Nov. 1998.
7. H. Jin, A. Kuehlmann, and F. Somenzi, "Fine-grain conjunction scheduling for symbolic reachability analysis," in *Tools and Algos. Construction and Analysis of Systems*, April 2002.
8. P. Chauhan, E. Clarke, J. Kukula, S. Sapra, H. Veith, and D. Wang, "Automated abstraction refinement for model checking large state spaces using SAT based conflict analysis," in *FMCAD*, November 2002.
9. A. Kuehlmann and J. Baumgartner, "Transformation-based verification using generalized retiming," in *Computer-Aided Verification*, July 2001.
10. D. Wang, P.-H. Ho, J. Long, J. H. Kukula, Y. Zhu, H.-K. T. Ma, and R. F. Damiano, "Formal property verification by abstraction refinement with formal, simulation and hybrid engines," in *Design Automation Conference*, June 2001.

11. D. Wang, *SAT based Abstraction Refinement for Hardware Verification*. PhD thesis, Carnegie Mellon University, May 2003.
12. C. A. J. van Eijk, "Sequential equivalence checking without state space traversal," in *Design, Automation, and Test in Europe*, March 1998.
13. A. Kuehlmann, V. Paruthi, F. Krohm, and M. Ganai, "Robust Boolean reasoning for equivalence checking and functional property verification," *IEEE Transactions on CAD*, Dec. 2002.
14. H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman, "Exploiting suspected redundancy without proving it," in *Design Automation Conference*, June 2005.
15. O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Transactions on Programming Languages and System*, vol. 16, no. 3, 1994.
16. L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, 1956.
17. K. Fisler and M. Vardi, "Bisimulation and model checking," in *CHARME*, Sept. 1999.
18. J. H. Kukula and T. R. Shiple, "Building circuits from relations," in *CAV*, July 2000.
19. M. Awedh and F. Somenzi, "Increasing the robustness of bounded model checking by computing lower bounds on the reachable states," in *FMCAD*, Nov. 2004.
20. E. Clarke, A. Gupta, J. Kukula, and O. Strichman, "SAT based abstraction-refinement using ILP and machine learning techniques," in *Computer-Aided Verification*, July 2002.
21. J. Baumgartner, A. Kuehlmann, and J. Abraham, "Property checking via structural analysis," in *Computer-Aided Verification*, July 2002.
22. A. Gupta, M. Ganai, Z. Yang, and P. Ashar, "Iterative abstraction using SAT-based BMC with proof analysis," in *Int'l Conference on Computer-Aided Design*, Nov. 2003.
23. H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman, and A. Kuehlmann, "Scalable automated verification via expert-system guided transformations," in *FMCAD*, Nov. 2004.
24. C. Leiserson and J. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, 1991.
25. J. Baumgartner, T. Heyman, V. Singhal, and A. Aziz, "An abstraction algorithm for the verification of level-sensitive latch-based netlists," *Formal Methods in System Design*, (23) 2003.