Optimal Constraint-Preserving Netlist Simplification

Jason Baumgartner[1] Hari Mony[1,2] Adnan Aziz[2]
[1]IBM Systems & Technology Group, Austin, TX [2]The University of Texas at Austin

***Abstract*— We consider the problem of optimal netlist simplification in the presence of *constraints*. Because constraints restrict the reachable states of a netlist, they may enhance logic minimization techniques such as redundant gate elimination which generally benefit from unreachability invariants. However, optimizing the logic appearing in a constraint definition may weaken its state-restriction capability, hence prior solutions have resorted to suboptimally neglecting certain valid optimization opportunities. We develop the theoretical foundation, and corresponding efficient implementation, to enable the optimal simplification of netlists with constraints. Experiments confirm that our techniques enable a significantly greater degree of redundant gate elimination than prior approaches (often greater than $2\times$), which has been key to the automated solution of various difficult verification problems.**

I. Introduction

Verification testbenches often require the specification of environmental assumptions to prevent uninteresting failures due to illegal input scenarios. For example, a testbench for an instruction buffer may require adherence to assumptions which model the behavior of the instruction fetch unit, such as requiring that instructions are not transferred into the buffer if doing so would cause overflow. Most testbenches require a substantial number of assumptions, many of which involve temporal handshaking with the outputs of the design.

There are two fundamental approaches to modeling environmental assumptions. First, one may utilize an imperative generator-style paradigm, where (possibly sequential) *filter* logic is used to convert nondeterministic data streams into legal input sequences. This filter logic is in turn composed with the design under verification [1]. Second, one may utilize a declarative constraint-based approach, wherein illegal scenarios are enumerated using specific language constructs, and the verification toolset must ensure that these scenarios are not violated in any reported counterexample [2].

Constraint-based testbenches have numerous advantages over generator-based approaches. Most notably, the *checker-assumption duality* paradigm allows an assumption on the inputs of one design component to be directly reused as a checker on the outputs of an adjacent design component. This duality enables the guarantee of compositional correctness by cross-validating assumptions across adjacent components [3]. Constraints may also be used to implement case-splitting strategies to decompose complex verification tasks for computational efficiency [4]. Due to their benefits, constraints have gained wide-spread acceptance, and most verification languages provide constructs to specify constraints – e.g., through the *assume* keyword of SystemVerilog [5].

Despite their prevalence, constraints pose challenges to numerous verification algorithms. For example, consider redundancy removal frameworks, wherein gates which are functionally equivalent in all reachable states may be merged to reduce design size. Because constraints restrict the set of reachable states, they may enhance reduction potential by enabling a pair of gates to be merged which are equivalent within the constrained state space, but may not be equivalent otherwise. Viewed another way, the constraints imply a *don't care* condition which may be exploited when attempting to merge gates, similar to the exploitation of *observability don't cares* (ODCs) for enhanced reduction [6]. However, as discussed in [7], such merging risks weakening the evaluation of the constraints, resulting in spurious property violations.

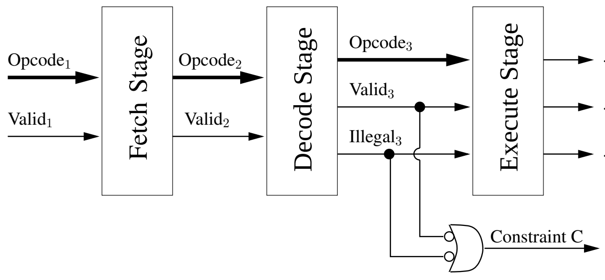

Fig. 1: Constraint weakening through merging

To illustrate how constraints may be weakened through merging, consider the example in Figure 1 which includes the Fetch-Decode-Execute component of a processor. Assume that the testbench for this component must prevent the Execute stage from encountering *valid* instructions with *illegal* opcodes. This may easily be achieved by constraining the output of the Decode stage to be *invalid* if its opcode is *illegal*. Next assume that the logic in the Fetch and Decode stages ensures that if the instruction in the Decode stage is invalid, its opcode is illegal. Coupled with the constraint, which enforces (Valid$_3$, Illegal$_3$) $\neq$ (1,1), this *satisfiability don't care* condition that (Valid$_3$, Illegal$_3$) $\neq$ (0,0) ensures that instructions in the Decode stage are invalid if and only if they are illegal, within all reachable states. This fact may be used to simplify logic in the Execute stage.

However, this fact will also entail redundancies in the fanin of the constraint, e.g., that the Valid$_3$ and Illegal$_3$ gates are antivalent—hence the redundancy removal process may wish to perform the corresponding merges. While the simplification entailed by merging is often advantageous in reducing subsequent verification resources, constraint-enabled merges within the fanin of constraints may lead to an overapproximation in property checking, in that the resulting constraints may lose their ability to prevent invalid counterexamples. In this case, such merging would syntactically simplify the constraint gate to *constant one*, precluding its ability to prevent valid yet illegal instructions from propagating into the Execute stage.

We address the simplification of netlists in the presence of constraints through redundancy removal. In particular, we provide the theoretical foundations as well as an efficient implementation of an *assume-then-prove* sequential redundancy removal procedure. Our specific contributions include:

1. an efficient input stimulus generation algorithm that is robust against *dead-end states* (Section III);
2. a sequential redundancy identification framework which allows the identification of all equivalent gates in the presence of constraints (Section IV); and
3. an abstraction-refinement algorithm to optimally leverage the identified redundancy for netlist simplification (Sections V and VI).

Experiments are presented in Section VII to demonstrate that our solution enables a significantly greater degree of redundant gate elimination than prior techniques, which has been key to the automated solution of numerous difficult industrial verification problems. We discuss related work in Section VIII, and conclude this work in Section IX.

II. Preliminaries

Definition 1: A *netlist* is a tuple $\langle\langle V, E\rangle, G, T, C\rangle$ comprising a finite directed graph with vertices V and edges $E \subseteq V \times V$. Function $G : V \mapsto types$ represents a mapping from vertices to *gate types*, including primary inputs, registers, and combinational gates with various functions. The register is the only sequential gate type, which has a designated *initial value* (which specifies its value at time 0) as well as *next-state function* (which defines its time $i+1$ behavior). To ensure consistent semantics, it is required that a netlist contain no *combinational cycles*: directed cycles in $\langle V, E\rangle$ comprising no registers. Set $T \subseteq V$ represents the *targets*, corresponding to properties to be checked. Set $C \subseteq V$ represents the *constraints*, the significance of which will be described below.

Definition 2: A *trace* is a temporal sequence of Boolean valuations to vertices in the netlist which is consistent with G, beginning at time 0 which is consistent with initial values. A trace is hereafter understood to be restricted to its valid prefix wherein all constraint gates evaluate to 1. Practically, this prefix is often understood to be significantly shorter, e.g., merely long enough to demonstrate the assertion of a target. The *length* of a trace is 0 if it has no valid time-frames in its prefix, else one plus the last prefix time-frame.

The *verification goal* associated with a netlist is to obtain a trace illustrating an assertion of a target within its valid prefix (such a trace is hereafter referred to as a *counterexample*), or to prove that no such trace exists.

Definition 3: The *fanin* of gate g is the union of g and all gates h for which there exists a directed path from h to g in $\langle V, E\rangle$. The *combinational fanin* of g is merely g if g is of type *register*, else the union of g and all gates h for which there exists a directed path from h to g which includes zero registers, aside possibly from h itself. The *fanout* of g is the set of gates which include g in their fanin. The *combinational fanout* of g is the set of gates which include g in their combinational fanin.

1. Use an arbitrary set of algorithms to compute the *redundancy candidates* of N: sets of equivalence classes of gates, where every gate g in equivalence class $Q(g)$ is suspected to be equivalent to every other gate in the same equivalence class, along every trace.
2. Select a *representative gate* $R\big(Q(g)\big)$ from each equivalence class $Q(g)$.
3. Construct the *speculatively-reduced netlist* N' from N by replacing the source gate g of every edge $(g, h) \in E$ by $R\big(Q(g)\big)$. Additionally, for each gate g, add a *miter* T_g, which is a target representing function $g \not\equiv R\big(Q(g)\big)$.
4. Attempt to prove that each of the miters in N' is semantically equivalent to 0.
5. If any miters cannot be proven equivalent to 0 due to obtaining a trace illustrating their assertion or due to an inconclusive result from the proof algorithm, refine the equivalence classes to separate the corresponding gates and goto Step 2; else proceed to Step 6.
6. All miters have been proven equivalent to 0; return the accurate equivalence classes.

Fig. 2: Sequential redundancy identification framework

Definition 4: A *state* is a valuation to the registers of a netlist. A *reachable state* is one which may occur in a trace, and an *initial state* is a reachable state which may occur at time 0. A *dead-end state* is a state for which no valuation to the primary inputs will satisfy the constraints, hence is unreachable.

Definition 5: Given two netlists N and N', we say that N' *trace-contains* N if every trace of N is valid for N'. If the converse is also true, we say that N and N' are *trace-equivalent*.

Note that if N is trace-equivalent to N', verifying N' in place of N is *sound* and *complete*: a proof or counterexample obtained on N' may be reused as a result for the corresponding target of N. If N is trace-contained by N', verifying N' in place of N is *sound* but *incomplete*: a proof obtained on N' may be reused as a result for the corresponding target of N, though a counterexample on N' may not be valid for N. We refer to such a trace, which illustrates a target assertion on N' but not on N, as a *spurious counterexample*.

A. Redundancy Removal

Definition 6: A *merge* from gate g_1 onto gate g_2 consists of replacing every fanout edge $(g_1, g_3) \in E$ with (g_2, g_3).

To facilitate subsequent reasoning (e.g., trace analysis), after merging, g_1 is made a *buffer* – a single-input gate whose *type* is the identity function. This is done by deleting its fanin edges, and adding edge (g_2, g_1). Without loss of generality we assume that $g_1 \not\equiv g_2$, and that merges will yield valid netlists – i.e., may be initiated only if no combinational cycles will result.

Sequential redundancy removal frameworks attempt to identify functionally redundant gates in a netlist, which may be merged as a trace-equivalence preserving transformation. We refer to redundancy identification as *sound* if the identified gates are truly equivalent in all reachable states, and *complete* if all functionally equivalent gates are identified. Sequential redundancy identification frameworks, e.g., [8], [9], [10], operate as shown in Figure 2. Step 1 of this algorithm typically uses random simulation to compute candidates for equivalence. We address the impact of constraints on this process in Section III.

The speculative merging performed in Step 3 is necessary for scalability, since sequential redundancy identification is PSPACE-complete [11]. To ensure soundness of redundancy

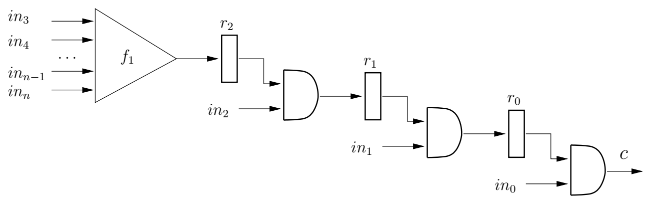

Fig. 3: Sequential constraint example

identification when using speculative reduction, it is required that the redundancy candidates computed in Step 1 be validated as being accurate in the initial states [10]. The selection of representatives must furthermore be performed such that N' contains no combinational cycles. This step requires additional consideration in the presence of constraints, since if the redundancy candidates are not truly equivalent, the speculative merge may alter constraint evaluation resulting in a procedure that is neither sound nor complete. We address this problem in Section IV.

When this algorithm terminates, it will reflect *all* gates which are equivalent across all reachable states. However, in the presence of constraints, additional analysis is necessary before they may be merged to avoid violating netlist semantics. We address this topic in Sections V and VI.

III. Constraint-Preserving Simulation

The use of random simulation is often critical to the effective computation of redundancy candidates. By comparing gate valuations from simulation traces, many inequivalences may be identified with modest runtime [8], [12].

Constraint-preserving input stimulus generation has been widely researched by the simulation community, e.g., in [13], [14]. However, the majority of prior work does not address dead-end states. While some discourage the use of constraints which entail dead-end states [13], such constraints may be readily expressed in various specification languages. We have often seen such constraints in practice due to the manual effort involved in mapping all constraints directly to inputs; e.g., to convert the single-predicate constraint that illegal Decode-stage instructions imply invalidity in Figure 1 into a complex constraint over instructions entering the Fetch stage. When a dead-end state is encountered, the simulation process must halt and begin anew from a shallower state. While the overhead of checkpointing and restoring states is somewhat undesirable, a more significant problem is that dead-end states may outright preclude the ability of simulation to reach deeper states in the design. In a sequential redundancy identification framework, this may entail highly inaccurate initial equivalence classes for gates which may only be differentiated by deep traces.

To illustrate the challenge of stimulus generation in the presence of sequentially-driven constraints, consider the example of Figure 3, where gate c is a constraint and r_0, r_1, r_2 are registers. The stimulus generator must assign values to inputs in_0, in_1, in_2 which prevent c from evaluating to 0 for the desired duration of the simulation run. While the stimulus generator could assign 0 to in_i (for $i \in \{0,1,2\}$) at any time-frame j, doing so would result in violating c at time $j+i$. The stimulus generator thus must perform assignments at time j which preclude constraint violations at later time-frames.

Our solution for stimulus generation is based upon k-step satisfiability (SAT) solving. At each time-frame j, we cast a satisfiability check from the current state of the netlist to identify a set of input valuations at time j which may be extended to satisfy each constraint for times j through $j+k$ using a *Sliding Window* approach. For scalability, it is important to keep k as small as possible because satisfiability checking is an NP-complete problem [15]. Our solution, as follows, thus performs design-specific analysis to determine a minimal value of k which avoids dead-end states.

Definition 7: Given a simple directed path in $\langle V, E \rangle$, the *delay* of that path is the number of gates of type *register*.[1] The *minimum input delay* of a gate g is defined as the minimum delay along any simple path from any primary input to g. The *maximum input delay* of a gate g is the minimum delay relative to the deepest primary input: the maximum value among any primary input for the minimum delay along any simple path from that primary input to g.

Empirically, we have observed that setting k to the *maximum input delay* of a corresponding constraint gate is adequate to prevent simulation from encountering dead-end states. The use of maximum input depth for k is intuitive; it represents the minimum delay at which some *furthest* input may affect the evaluation of a given constraint. Depending upon the nature of the design, an input may affect the constraint much later than the input delay, e.g., if the design has a counter. However, we have not encountered any instances for which this depth is inadequate, given a large set of dozens of industrial designs.

A related concern is whether k is larger than necessary, since the performance of the SAT solver is highly sensitive to this value. For example, if the AND gates in Figure 3 are converted to OR gates, the *minimum input delay* of 0 is adequate to avoid dead-end states by assigning in_0 to 1. In practice, the computation of minimum adequate window depth is prohibitively expensive, requiring the solution of a quantified Boolean formula checking whether for every state, there exists an input sequence of a particular depth which satisfies that constraint. Our practical solution to this problem is to break the overall simulation run into multiple phases, using a $\log_2$ search procedure within a minimum and maximum range (initialized as the *minimum* and *maximum input delay*, respectively) to identify a minimal adequate window depth to enable the desired simulation run length. At each phase, we generate stimuli using the minimum range value as the window depth. If a dead-end state is encountered, we update the minimum range to the unsuccessful value plus one and repeat the phase using a median depth between the minimum and maximum range. Otherwise, we update the maximum range to the successful value and proceed to the next phase.

IV. Optimal Redundancy Identification under Constraints

Speculative merging in the presence of constraints raises two concerns:

[1]We assume that oscillating clocks have been factored out of the netlist diameter, e.g. through *phase abstraction* [16]. Otherwise, the *delays* introduced in this section should be multiplied by the periodicity of the clocks.

1. Even if the speculative merges are for truly equivalent gates, the merges may weaken the constraints as per the example of Figure 1. This may entail incomplete redundancy identification, since unreachable states may become reachable under the speculative merges.
2. At the time of speculative reduction, the validity of the redundancy candidates has not been demonstrated. Thus, the speculative merges risk arbitrarily altering constraint evaluation, which may weaken *or strengthen* their evaluation, causing unsound redundancy identification since reachable states may become unreachable.

The following theorem establishes the correctness of a slightly modified form of speculative reduction in netlists with constraints, which ensures that miters for inequivalent redundancy candidates will remain assertable, and accurate miters will remain unassertable.

Theorem 1: Given a netlist N, consider netlist N' formed by adding to N a replication of the combinational fanin of each constraint, and re-labeling the replicated counterpart of each constraint gate as the constraint. Consider netlist N'', derived by speculative reduction of N', though restricting merging to the original gates of N. Using N'' as the basis of sequential redundancy identification is sound and complete in identifying redundant gates in N.

Proof: Consider any trace p over N, and the corresponding trace p'' obtained by simulating the input sequence of p on N''. To demonstrate soundness and completeness of redundancy identification, we prove a stronger condition: that either p illustrates no mismatches within any equivalence class *and* no miters are asserted in p'' (completeness), or that a nonempty subset of earliest-occurring mismatches illustrated in p has a corresponding subset of earliest-occurring miter assertions in p'' (soundness). We prove this condition by induction on this earliest time-frame.

Base Case: Time 0. As discussed in Section II-A, to ensure the soundness of speculative reduction, all equivalence classes must be validated as accurate at time 0. Thus no mismatches may be illustrated in p. Additionally, if p is valid at time 0, the speculative merging cannot alter any valuations in p vs. p'' at time 0 hence there can be no miter assertions at time 0. Furthermore, since there are no speculative merges in the combinational fanin of the constraints, their evaluation and hence their constraining power over registers and inputs cannot be altered, and thus p is valid at time 0 if and only if p'' is valid at time 0

Inductive Case: Time $i + 1$. By the induction hypothesis, assume that no mismatches nor miter assertions occurred at times $0, \ldots, i$. We prove that this condition holds at time $i+1$. If no mismatches in p nor miter assertions in p'' manifest at time $i+1$, our proof obligation is trivially satisfied. Otherwise, first consider the case that there is a nonempty set of gates A which differ from their representatives at time $i + 1$ in p. Consider the maximal subset $B \subseteq A$ such that no element in $b \in B$ has any element of $A \setminus b$ in the combinational fanin of b or $R(b)$, which is the representative gate from the equivalence class containing b. Clearly B is non-empty since N'' is a netlist hence free of combinational cycles. Note that any speculative merging in the fanin of b or $R(b)$ cannot alter their valuation at time $i + 1$, because that merging did not alter the valuation of any gate at times $0, \ldots, i$ by the induction hypothesis, nor at time $i+1$ by the construction of B. Thus, the miters correlating to every gate in B must assert at time $i + 1$ in p''. This same argument may be used to demonstrate that a nonempty set B of miter assertions at time $i+1$ in p'' must have a corresponding set of mismatches in p.

We finally must demonstrate that p is valid at time $i + 1$ if and only if p'' is valid at time $i + 1$. Note that the state of the registers in the combinational fanin of the constraints must be identical across p vs. p'' at time $i + 1$, since their next-state functions evaluated the same at times $0, \ldots, i$ by the induction hypothesis. Since there are no speculative merges in the combinational fanin of the constraints, any mismatches at time $i + 1$ cannot affect their *valuation* at that time-frame, nor their *evaluation* hence their constraining power over registers and inputs cannot be altered. Thus p is valid at time $i + 1$ if and only if p'' is valid at time $i + 1$.

We thus conclude that either no mismatches occur in p nor miter assertions in p'' at time $i + 1$, or a nonempty set of speculatively merged gates B will illustrate miter assertions in p'' and mismatches in p at time $i + 1$. ■

A. Refinement of Equivalence Classes

Because speculative merging of incorrect redundancy candidates may alter the evaluation of gates in their fanout, this may ambiguate the exact set of speculative merges which must be refined given a trace asserting a miter. Such inaccurate merging may cause miters in the fanout of B, the set in the proof of Theorem 1, to assert even though the corresponding redundancy candidates are truly equivalent, which if refined would result in suboptimal redundancy identification. This may also cause such miters to become unassertable even though the corresponding redundancy candidates would be differentiated during resimulation of the mismatch trace on the original netlist, which if not refined would require the computationally redundant step of generating an equivalent miter assertion trace at a future refinement iteration. Thus in practice, miter assertion traces should be resimulated on the original netlist to assess the exact set of gates to refine [10].

V. Redundancy Removal under Constraints

Using the framework we have developed in Section IV, we may compute the exact set of gates which are equivalent in the constrained reachable state space. In theory, this framework is capable of solving every unassertable target, since they correlate to gates which are semantically equivalent to 0. However, either due to computational resource limitations which result in suboptimal redundancy identification, or due to targets which are truly assertable, some targets may remain unsolved after the redundancy identification process.

If any targets remain unsolved after the redundancy identification process, it is generally desirable to leverage the identified redundancy to simplify the netlist so that a subsequent verification strategy may benefit from that simplification [10], [12], [17]. Examples of known algorithmic synergies which benefit from redundancy elimination include: faster and deeper exhaustive bounded search using SAT; greater reduction potential through transformations and abstractions such as combinational rewriting, retiming, and localization reduction; and enhanced inductiveness [10], [17]. In this section, we discuss how to optimally leverage this identified redundancy.

It was demonstrated in [7] that a merge of gate g_1 onto g_2 is guaranteed to preserve property checking as long as no constraint c_1 in the combinational *or* sequential fanout of g_1 was used to constrain the state space during the proof of $g_1 \equiv g_2$. This implies that certain identified equivalent gates may be safely merged. However, this result is suboptimal since certain merges which do not adhere to this criterion may nonetheless be performed while preserving property checking. For example, industrial verification testbenches often arise which incorporate a large number of constraints and targets, even though some of the constraints may be unnecessary to establish the correctness of some of the targets. If the targets under verification are already unreachable (possibly due to a set of existing constraints $c_1, \ldots, c_i$), adding a constraint c_{i+1} need not preclude merges regardless of fanout connectivity since weakening c_{i+1} cannot cause spurious counterexamples. In practice, it is difficult to assess which of the identified redundancies is conditional upon which subsets of the constraints (e.g., vs. holding due to satisfiability don't cares alone), without performing numerous redundancy identification analyses with different subsets of constraints or per-miter minimal-proof analysis. Such processes are computationally expensive, motivating our technique to optimally leverage the redundancy identified using *all* constraints within a single efficient run of the algorithm of Figure 2.

The following theorem establishes that we may safely perform a greater degree of merging than enabled by [7]

Theorem 2: Consider any set of gate pairs which have been proven equivalent across all reachable states in netlist N, and whose merged *from* gates (vs. merged *onto* gates) do not lie in the combinational fanin of any constraint. Performing the corresponding merges to yield netlist N' will ensure that N' is trace-equivalent to N.

Proof: Similarly to Theorem 1, since no gates within the combinational fanin of constraints are merged onto others, their evaluation and hence their constraining power over registers and inputs cannot be altered. The initial states of N and N' are thus identical; any trace of length 1 in N is valid in N' and vice-versa. Furthermore, since the merges have been verified to reflect equivalence across the constrained reachable states, they cannot alter next-state function valuations. Thus, by simple inductive reasoning, any trace prefix of length $i+1$ in N' is also valid for N and vice-versa. ■

Theorem 2 enables us to perform a significantly greater degree of merging than enabled by the results of [7], particularly for highly cyclic netlists where the fanin of the constraints includes almost all gates. However, we note that this result is also suboptimal, since the combinational fanin of the constraints may be arbitrarily large, and there may still be verification-preserving merging possible therein. We now demonstrate that general constraint-enhanced merging is sound but incomplete.

```
Given: Netlist $N'$ formed from $N$ by merging equivalent gate pairs
    $G_f \mapsto G_t$, and traces $p'_1, \ldots, p'_i$ over $N'$
1. Simulate each $p'_i$ on $N$ to obtain trace $p_i$
2. If $p_i$ asserts any target $t$ in $N$, report that result as a valid counter-
   example and eliminate $t$ from $T$, the targets of $N$
3. If $T$ is empty, exit
4. Identify the set of merged nodes $D_1 \subseteq G_f$ of $N'$ which differ in
   valuation across any $p_i$ and $p'_i$ within the prefix of $p'_i$, ignoring
   constraint violations in $p_i$
5. Construct netlist $N''$ from $N$, performing conditional merges for set
   $D_1$, and conjuncting each constraint gate with each target in $N''$
6. Cast a SAT problem conjuncting over each $p'_i$, checking for the lack
   of an assertion of targets in $N''$ under the input sequence of $p'_i$
7. Define causal merge set $D_2$ as those whose selector is assigned to 0
   from the SAT solution
8. Form refined netlist $N'''$ from $N$ by merging $G_f \setminus D_2$
```

Fig. 4: Trace refinement algorithm

Theorem 3: Consider any set of gate pairs which have been proven equivalent across all reachable states in netlist N. Performing the corresponding merges to yield netlist N' will ensure that N' trace-contains N.

Proof: The fact that the gate pairs have been proven equivalent across all reachable states ensures that, within all trace prefixes which do not violate constraints in N, these merges do not alter the valuation of any gate whatsoever in N'. Thus, every constraint-satisfying trace prefix in N is also valid in N'. However, the fact that merging may be performed in the combinational fanin of the constraints means that their evaluation relative to registers and inputs may be altered. In particular, for extensions to valid trace prefixes in N – i.e., for time-frames $i + j$ (for non-negative j) where some constraint in N is violated at time i – the merges may alter the evaluation of gates in their fanout. In doing so, these merges may cause constraints in their fanout to evaluate to 1 in N' vs. to 0 in N. Thus, constraint-satisfying trace prefixes in N' may violate constraints in N. Netlist N' thus trace-contains N, but generally may not be trace-equivalent to N. ■

Theorem 3 implies that, if a target t' is proven as unassertable in N', the corresponding target t must be unassertable in N – though counterexamples from N' may be invalid on N. This theorem motivates an abstraction-refinement framework conceptually similar to [18], which retains those merges which do not entail spurious counterexamples for efficiency of a subsequent verification strategy, while discarding the others.

A. Abstraction-Refinement Framework

Definition 8: A *conditional merge* from gate g_1 onto gate g_2 consists of replacing g_1 by a multiplexor whose selector is driven by a newly-created nondeterministic constant[2] gate i_{g_1,g_2}. If the selector evaluates to 1, the value of g_2 is driven at the output of the multiplexor. Otherwise, the original value of g_1 is driven at the output of the multiplexor.

We present an algorithm in Figure 4 for identifying a set of merges (hereafter referred to as *causal merges*) to discard in response to a set of spurious counterexamples. Note that this algorithm is similar to those for automated

[2]A nondeterministic constant may be represented in a netlist by a register whose next-state function is itself (hence it never toggles), and whose initial value is a primary input.

design debugging [19]. The abstracted netlist N' is formed by merging each $g_f \in G_f$ onto the corresponding $g_t \in G_t$ as per the surjective mapping $G_f \mapsto G_t$. Step 1 of this algorithm maps each trace p'_i obtained from N' to trace p_i over N, from which we may assess the behavior of N under the input sequence demonstrated in p'_i. Step 2 checks if any resulting trace constitutes a valid assertion of any target in N. If so, that trace is reported as a counterexample and the corresponding target is eliminated from the set of unsolved targets T. If T becomes empty, then the verification problem has been solved hence the algorithm exits in Step 3. Otherwise, the algorithm proceeds to identify a set of merges which were responsible for the spurious counterexamples. Step 4 discerns an overapproximation D_1 of the set of causal merges, identifying those merged gates whose valuations differ between any p_i and p'_i during the trace-asserting prefix of p'_i. Set D_1 is next minimized in Steps 5-7 by casting a SAT problem which seeks to avoid target assertions under the input sequence of each p'_i within netlist N'' formed from N by performing *conditional merges* for the potentially causal merges. Note that the constraints are conjuncted with each target in N'' vs. being retained natively, so that the resulting SAT instance will be satisfiable and reflect the ability to preclude a target assertion due to constraint violations in N. Step 8 constructs a refined netlist, eliminating those merges which were determined to cause the spurious counterexamples in N'.

Theorem 4: Given trace p'_i which is a counterexample to target t'_i of netlist N', the algorithm of Figure 4 will either yield a valid counterexample p_i to target t_i of N, or produce a refined netlist N''' which will not exhibit a spurious assertion against the input stimuli of p'_i.

Proof: This theorem is trivially true if the algorithm produces a counterexample on N. Otherwise, we note that:

- When constructing D_1, gate inequivalence is checked for all time-frames in the prefix of p'_i, which illustrates the spurious assertion of t'_i. This check ignores constraint-based prefixing within p_i to enable detecting *which* merges in N' weakened the constraints in N. Thus D_1 will include all merged gates whose behavior was altered in p'_i vs. p_i.
- The SAT problem is satisfiable, since if the selector of each conditional merge construct is set to 0, the SAT solution will be equivalent to p_i which has been demonstrated not to assert t_i in N.
- Set D_2 by construction enumerates the subset of merges which, if eliminated, will prevent the assertion of t''', the counterpart of t' in N'''. ■

As per Theorem 4, the refinement process of Figure 4 is adequate to ensure that the resulting netlist will not exhibit a spurious assertion against any counterexample used as the basis of the refinement. This result allows us to develop an abstraction-refinement framework which is guaranteed to converge as presented in Figure 5.

Theorem 5: The algorithm of Figure 5 (run in *standard abstraction-refinement* mode) will converge upon a correct verification result for the original netlist N.

Proof: We consider the individual steps of this algorithm.

```
Given: Netlist $N$; Initialize $i = 1$
1. Compute equivalence classes in $N$ using a variant of the algorithm
   of Figure 2 as per Theorem 1, yielding desired merges $G_f \mapsto G_t$
2. Form $N'$ by performing the subset of merges $G'_f \mapsto G_t$ which adhere
   to Theorem 2
3. Form $N_1$ from $N'$ by performing the remaining merges $G''_f \mapsto G_t$
4. Use an arbitrary set of algorithms to attempt to prove or falsify the
   targets in $N_i$
5. If any targets were proven unassertable on $N_i$, report the corresponding
   targets unassertable for $N$
6. If a nonempty set of counterexamples $P_i$ were obtained on $N_i$, use
   Figure 4 on netlist $N^*$ vs. $N$ and trace set $P^*$ to obtain a valid set of
   counterexamples and / or a refined netlist $N_{i+1}$, else exit
7. Report any valid counterexamples that were obtained for $N$
8. If unsolved targets remain, increment $i$ and goto Step 4, else exit
```

Fig. 5: Abstraction-refinement framework. $N^* = N_i$ and $P^* = P_i$ in Step 4 implies *standard abstraction-refinement*; $N^* = N_1$ and $P^* = \bigcup_{j \in \{1,\ldots,i\}} P_j$ in Step 4 implies *optimal abstraction-refinement*

- Step 1 computes the equivalence classes of gates, which are correct as per Theorem 1. We may select an arbitrary set of desired merges consistent with these equivalence classes, reflected by surjective mapping $G_f \mapsto G_t$.
- Step 2 performs those subset of merges (denoted as $G'_f \mapsto G_t$) which are guaranteed to preserve trace-equivalence as per Theorem 2. Thus, if verification were performed directly upon N', all results would be correct with no need for refinement.
- Step 3 performs the remaining merges, denoted as $G''_f \mapsto G_t$ where $G''_f = G_f \setminus G'_f$, to yield abstract netlist N_1.
- Because N_i trace-contains N as per Theorem 3, any target proven unassertable in N_i implies a corresponding unassertable target in N, hence any results reported in Step 5 are correct.
- If counterexamples were obtained for N_i, they may be valid for N or they may be spurious. The algorithm of Figure 4 is used to differentiate these cases in Step 6. The result of this algorithm is a set of valid counterexamples for N and / or a refined netlist N_{i+1}.
- By Theorem 4, any counterexamples reported in Step 7 will be valid.
- By Theorem 4, if any targets remain unsolved, refined netlist N_{i+1} will not exhibit a spurious assertion against traces P_i. Convergence of the refinement loop is guaranteed noting that netlists are finite as per Definition 1, hence $|G''_f|$ is finite and each refinement iteratively eliminates one or more elements from G''_f. ■

VI. Optimality of Reductions

Theorem 5 ensures the correctness and convergence of the overall abstraction-refinement procedure. However, there are several points to consider regarding the *optimality* of the resulting refined netlist, which may have a significant impact on the resources required to compute counterexamples, as well as to prove targets unassertable.

i) While the SAT solution obtained from Step 6 of the algorithm of Figure 4 identifies an adequate set of causal merges for refinement, it does not directly attempt to obtain a solution with a *minimal* set of causal merges, as would be necessary for optimality of the refined netlist.

ii) Compatibility issues with don't-care enabled merging entail that two sets of merges may be independently but not jointly valid or vice-versa [20], [6], [27]. The optimal selection of causal merges may thus entail cumulative suboptimalities across refinement iterations, even if each individual iteration is optimal.

Regarding the first issue: for optimal reductions, one would wish to eliminate as few merges as possible during each refinement. A precise solution to this problem may be attained by solving a max-sat problem [21] in Step 6 of the algorithm of Figure 4, constructed from the original SAT instance augmented with an additional clause for each conditional merge construct representing the selection of the merged value. For enhanced runtime, we have found that a near-optimal initial bound to the max-sat solution may be obtained using a standard SAT solver augmented with a decision procedure which heuristically assigns 1 to primary inputs before assigning 0. Due to enforcing the input sequence of p_i', note that primary inputs within the resulting SAT instance need only occur within conditional merge instances.

Regarding the second issue: to circumvent the risk that choices at a given refinement iteration will entail cumulative suboptimality across multiple iterations, it is necessary to re-compute refinements relative to the maximally-merged abstraction N_1. This is illustrated by the *optimal abstraction-refinement* mode of the algorithm of Figure 5, where at each refinement iteration i, *all* prior counterexamples $\bigcup_{j\in\{1,\ldots,i\}} P_j$ are used to refine relative to N_1, instead of merely using the final set P_i to refine N_i. The following theorems demonstrate the correctness and optimality of this flavor of the abstraction-refinement process.

Theorem 6: The algorithm of Figure 5 (run in *optimal abstraction-refinement* mode) will converge upon a correct verification result for the original netlist N.

Proof: The correctness of verification results follows from the proof of Theorem 5. To guarantee convergence, we note that we cannot have two identical refined netlists $N_i = N_j$ for $i > j$. This follows by Theorem 4 noting that N_i is formed by refining against *all* prior traces, including P_j which comprises one or more spurious counterexamples on N_j. Additionally, since netlists and hence G_f'' are finite, there are a finite number of possible distinct refined netlists. ■

Theorem 7: Assuming that a max-sat procedure is used in the refinement algorithm of Figure 4, the algorithm of Figure 5 (run in *optimal abstraction-refinement* mode) will at every iteration yield an optimal refined netlist which retains as many of the desired merges as possible while not exhibiting a spurious assertion against any counterexample obtained prior to that iteration.

Proof: This proof follows trivially by Theorem 6 and by the construction of the max-sat formulation. ■

The validity of don't-care enabled merges is generally neither symmetric nor transitive [6]. Thus, the selection of desired merges from the computed equivalence classes in Step 1 of the algorithm of Figure 5 may impact the size of the refined netlist. Clearly optimality would be achieved if the algorithm of Figure 5 were run upon every permutation of desired merges consistent with the computed equivalence classes, and selecting the result which retains the most merges. However, doing so would be computationally intractable. This computational expense may be minimized as follows: while *causal* merges must be eliminated upon a refinement, we may attempt to introduce *alternate* merges from within the equivalence classes in their place. Such a framework may be used to generate a refinement retaining an optimal number of merges, provided that the procedure exhaustively attempt alternate merges before outright discarding any set of causal merges. Convergence of such an alternate-merge introduction framework may be guaranteed simply by ensuring that no refinement N_{i+j} repeat the same set of merges reflected in an earlier refinement N_i. While computationally superior to the naïve approach of exhaustive enumeration, since alternative merges need only be explored *on demand*, this approach is also prone to be computationally intractable in practice due to requiring the exploration of all cross-products of alternate merges for the identified causal merges. We thus introduce an efficient technique to generate a nearly-optimal set of desired merges from the equivalence classes in Section VI-A.

A. Incremental Elimination of Constraint-Weakening Merges

SAT-based analysis is often used to search for counterexamples, iteratively checking for failures until computational limits are exceeded. For efficiency, it is desirable to leverage *incremental SAT* in this process, first creating a SAT instance to check for a failure at time 0, then unfolding an additional time-frame onto the existing instance to check for a failure at time 1, etc. This incrementality enables the reuse of learned clauses from earlier time-frames to speed up the SAT solution for later time-frames [22].

In an abstraction-refinement framework, incrementality is more difficult to achieve across refinements. However, if using the *conditional merge* construct instead of outright performing the desired merges in Step 3 of the algorithm of Figure 5, incrementality may be achieved across refinements by merely constraining the causal conditional merge selectors to 0 as also noted in [23]. Such constraints effectively eliminate the causal merges within the SAT instance and allow additional counterexamples to be identified therein. We have found that devoting a small amount of computational resources to such an incremental SAT-based procedure, and using the resulting counterexamples to jump-start the optimal abstraction-refinement process, tends to substantially reduce the resources necessary to arrive at an optimal refined abstraction which is not prone to spurious counterexamples.

Furthermore, one mechanism that we have found to quickly converge upon a nearly-optimal set of compatible merges from within the equivalence classes (Step 1 of the algorithm of Figure 5) is to use the preprocessing mechanism discussed in the prior paragraph with a variant of the conditional merge which enables the selection among *all* gates within an equivalence class. As spurious counterexamples are obtained, we may disable the causal merges and preserve selection among the rest. The desired merges may then be chosen from those which remain at the termination of this preprocessing step, heuristically helping to ensure near-optimal compatibility.

Benchmark	Design Info	SimGen [13]		Sliding Window			SAT-only	
	Gates	Valid Time-Steps	Time (s)	Input Depth: Min; Max; Algo	Valid Time-Steps	Time (s)	Valid Time-Steps	Time (s)
FXU	32903	1	0.13	1; 2; 1	1000	2.6	165	1800
FPU	115037	5	0.39	5; 14; 10	1000	228	902	1800
SCNTL	51504	53	2.94	7; 14; 8	1000	87	72	1800
IBUFF	19230	57	1.05	5; 13; 6	1000	7.4	134	1800
AXU	345518	1	33.24	1; 2; 1	1000	44.15	1000	60.68
IBM_FV_11	4799	2	0.17	4; 8: 6	1000	2.6	337	1800
IBM_FV_24	13391	2	0.59	4; 19; 4	1000	3.2	252	1800

TABLE I: Constraint-preserving simulation results

Benchmark	Design Info		Constraint-Safe Merging [7]			Constraint-Enhanced Merging						
	Gates	Targets	Gates Merged	Unsolved Targets	Resources (s; MB)	Gates Merged	Refined Merges	Refinement # CEXs	CEX Max Depth	Improvement in # Merges	Unsolved Targets	Resources (s; MB)
FXU	32903	8	2218	0	450; 146	2482	62	7	8	9.1%	0	318; 195
FPU	115037	1	2022	1	5465; 690	4928	0	0	0	143.7%	0	1140; 384
SCNTL	51504	551	6638	24	342; 133	6962	4	1	19	4.8%	0	162; 383
IBUFF	19230	303	222	14	77; 91	831	0	0	0	274.3%	0	78; 160
AXU	345518	1	734	1	956; 479	4828	343	6	16	511.0%	1[4]	616; 540
IBM_FV_11	4799	1	228	0	16; 64	747	0	0	0	227.6%	0	34; 69
IBM_FV_24	13391	1	313	0	70; 119	793	13	1	22	149.2%	0	58; 137

TABLE II: Sequential redundancy removal results

VII. Experimental Results

We now provide experiments to illustrate the verification enhancements enabled by our techniques. All experiments were run on a 2.1GHz POWER5 processor, using *Sixth-Sense* [17]. We applied our techniques on a variety of designs, including the subset of the IBM FV Benchmarks [24] which have interestingly large constraint cones, and five diverse and difficult industrial testbenches. FXU is a testbench used to verify the control path of a fixed-point unit. FPU verifies the correctness of bypassing logic in a floating-point unit. SCNTL is a testbench used to verify the control of an instruction-dispatch unit. IBUFF is an instruction buffer. AXU checks the datapath correctness of an arithmetic unit. Each of these has constraints which entail dead-end states.

The experiments of Table I illustrate the power of the Sliding Window algorithm presented in Section III. The first two columns indicate the name of the benchmark and the size of the original netlist. We compare the results of: **(1)** our implementation of SimGen [13] (Columns 3-4) which performs purely combinational constraint solving; **(2)** our Sliding Window approach (Columns 5-7); and **(3)** purely formal analysis using SAT to solve the constraints for the entire duration of the simulation run, using random stimulus generation for unassigned inputs (Columns 8-9). Our goal is to simulate the designs without encountering dead-end states for 1000 time-steps, within a time-limit of 1800 seconds. While fast, SimGen [13] results in constraint violations within 57 time-steps for each design, and often substantially lesser. The SAT-only approach times-out for every design aside from AXU, often completing substantially lesser than 1000 time-steps. In contrast, our Sliding Window approach is able to complete the desired simulation for every design. The window depth used varies across the designs; in Column 5, we report the minimum and maximum input depth, followed by the depth algorithmically converged upon by our $\log_2$ range analysis.

The enhanced redundancy removal enabled by our approach is illustrated in Table II. The first 3 columns indicate the name of the benchmark, the size of the original netlist and the number of targets in the original netlist. We first compare the redundancy removal possible using prior techniques [7] with that of our approach (Columns 4 vs. 7), and then illustrate the impact of this additional reduction on the verification process by using k-induction [25] to attempt to verify the targets on the optimized designs (Columns 5 vs. 12). The resources reported in Columns 6 vs. 13 refer to the combined process of redundancy removal and induction. The induction process was limited to 30 seconds and $k \leq 10$ to avoid significant skew of runtime for cases where targets were left unsolved.

As illustrated in Table II, our approach identifies significantly more redundancy resulting in an increased number of non-refined merges, often more than $2\times$ (indicated by a number greater than 100% in Column 11). The average increase in non-refined merges across all of these designs is 187.8%. This reduction was essential to proving a number of these targets, which otherwise were not inductive and extremely computationally expensive to solve with alternate algorithms.[3] We report the number of merges which were refined during the algorithm of Figure 5 in Column 8 (which is a subset of Column 7), along with the number of counterexamples (CEXs) used during that refinement in Column 9 and the maximum depth at which a refinement occurred in Column 10. Note that only a small percentage of the additional merges enabled by our techniques (4.8%) must actually be refined; nonetheless, in 4 of 7 examples, spurious property failures would have occurred without our refinement process. Our approach in only two cases entails moderate additional run-time due to the larger set of equivalence candidates during the redundancy identification process, though in most cases, particularly the FPU, this results in significantly lesser runtime.

VIII. Related Work

There are similarities between aspects of our theory and prior work on proof decomposition in assume-guarantee reasoning, e.g. [3]. For example, our requirement that the speculatively-merged netlist be acyclic, and our combinational constraint-cone replication of Theorem 1, are closely related to

[3]The AXU target required additional transformations before it became tractable for proof analysis; these details are omitted due to lack of space.

the requirement that combinationally-dependent assumptions vs. properties be used only under a strict dependency relation. However, such prior work in assume-guarantee reasoning has not addressed the problem of automated derivation of equivalent-gate conditions, nor addressed the impact of using such conditions for direct netlist simplification.

The work of [7] discusses redundancy removal in the presence of constraints, allowing gate merges in the fanin of a constraint provided that the proof of the corresponding gate equivalence does not require the state-pruning power of that constraint. Our approach eliminates this suboptimality.

In [26], the authors propose to enhance an inductive SAT solver by performing *all* merges enabled by the constraints representing the induction hypothesis. To compensate for the resulting constraint weakening, they add additional constraints within the SAT solver. Unlike our approach, they do not address constraint-enhanced reduction of sequential netlists; theirs is effectively a run-time optimization to the inductive SAT solver, which our approach may complementarily use if relying upon induction in Step 4 of the algorithm of Figure 2.

There are similarities between our approach and those that optimize relative to other types of don't cares such as ODCs. While most scalable ODC-based techniques rely upon suboptimal local analysis for efficiency (e.g., [6]), the SAT-based technique of [27] uses induction to validate sets of ODC-enhanced merges, enabling unreachability invariants to enhance reduction potential. However, the equivalence boundaries against which the ODC conditions are validated are purely combinational, limiting optimality. The approach of [23] enhances interpolation by optimizing logic resulting from interpolant synthesis relative to don't cares implied by already-reached states. They use incremental SAT to justify that a set of gates may be safely merged to constants, using a construct similar to the conditional merge of Definition 8 so that invalid merges may be disabled within the SAT instance. Neither of these works (nor any others that we are aware of) address efficient yet globally optimal reduction of sequential netlists, leveraging constraints for increased reduction potential while preserving constraint evaluation. Though overall, ODC-based optimization and constraint-enhanced redundancy removal are complementary approaches, and it is a promising area of future research to pursue constraint-enhanced yet constraint-preserving extensions to such complementary techniques.

Constraint-satisfying stimulus generation has been extensively studied, e.g., [13], [14], though little focus has been given to dead-end states. The approach of [28] does address dead-end states, using a BDD-based framework to manipulate a synthesized constraint automaton to avoid dead-end states before they are reached. While demonstrated to be effective, this approach is only applicable if the constraints are specified using temporal logic to reason solely about primary inputs. If arbitrary gates are referenced by the constraints, as in our practical experience they often are, their approach becomes underapproximate, precluding the exploration of arbitrary reachable states. Our techniques do not suffer this limitation.

IX. Conclusion

We have developed a theoretical framework and an efficient implementation for netlist simplification in the presence of constraints. Our solution includes a robust and efficient algorithm for constraint-preserving random stimulus generation, a sound and complete extension to scalable *assume-then-prove* redundancy identification frameworks, and an efficient abstraction-refinement framework to optimally eliminate the identified redundancy for enhanced property checking.

References

[1] Y. Zhu and J. Kukula, "Generator-based verification," in *ICCAD*, Nov. 2003.

[2] C. Pixley, "Integrating model checking into the semiconductor design flow," in *Electronic Systems Technology & Design*, 1999.

[3] T. A. Henzinger, S. Qadeer, and S. K. Rajamani, "Decomposing refinement proofs using assume-guarantee reasoning," in *ICCAD*, Nov. 2000.

[4] C. Jacobi, K. Weber, V. Paruthi, and J. Baumgartner, "Automatic formal verification of fused-multiply-add FPUs," in *DATE*, March 2005.

[5] Accelera, *SystemVerilog Language Reference Manual*. http://www.systemverilog.org/.

[6] Q. Zhu, N. Kitchen, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "SAT sweeping with local observability don't-cares," in *DAC*, July 2006.

[7] H. Mony, J. Baumgartner, and A. Aziz, "Exploiting constraints in transformation-based verification," in *CHARME*, Oct. 2005.

[8] C. A. J. van Eijk, "Sequential equivalence checking without state space traversal," in *DATE*, Feb. 1998.

[9] P. Bjesse and K. Claessen, "SAT-based verification without state space traversal," in *FMCAD*, Nov. 2000.

[10] H. Mony, J. Baumgartner, V. Paruthi, and R. Kanzelman, "Exploiting suspected redundancy without proving it," in *DAC*, June 2005.

[11] J.-H. Jiang and R. Brayton, "Retiming and resynthesis: A complexity perspective," in *TCAD*, vol. 25, Dec. 2006.

[12] A. Kuehlmann, "Dynamic transition relation simplification for bounded property checking," in *ICCAD*, Nov. 2004.

[13] J. Yuan, K. Shultz, C. Pixley, H. Miller, and A. Aziz, "Modeling design constraints and biasing in simulation using BDDs," in *ICCAD*, Nov. 1999.

[14] J. Yuan, K. Albin, A. Aziz, and C. Pixley, "Constraint synthesis for environment modeling in functional verification," in *DAC*, June 2003.

[15] S. A. Cook, "The complexity of theorem-proving procedures," in *ACM Symposium on the Theory of Computing*, May 1971.

[16] P. Bjesse and J. Kukula, "Automatic generalized phase abstraction for formal verification," in *ICCAD*, Nov. 2005.

[17] H. Mony, J. Baumgartner, V. Paruthi, R. Kanzelman, and A. Kuehlmann, "Scalable automated verification via expert-system guided transformations," in *FMCAD*, Nov. 2004.

[18] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith, "Counterexample-guided abstraction refinement," in *CAV*, July 2000.

[19] M. F. Ali, A. Veneris, S. Safarpour, R. Dreshler, A. Smith, and M. Abadir, "Debugging sequential circuits using Boolean satisfiability," in *ICCAD*, Nov. 2004.

[20] A. Mishchenko and R. Brayton, "SAT-based complete don't-care computation for network optimization," in *DATE*, March 2005.

[21] D. S. Johnson, "Approximation algorithms for combinatorial problems," *Journal of Computer and System Sciences*, vol. 9, 1974.

[22] O. Shtrichman, "Pruning techniques for the SAT-based bounded model checking problem," in *CHARME*, Sept. 2001.

[23] G. Cabodi, M. Murciano, S. Nocco, and S. Quer, "Stepping forward with interpolants in unbounded model checking," in *ICCAD*, Nov. 2006.

[24] IBM Formal Verification Benchmark Library. http://www.haifa.il.ibm.com/projects/verification/RB_Homepage/fvbenchmarks.html.

[25] N. Eén and N. Sörennson, "Temporal induction by incremental SAT solving," in *Workshop on Bounded Model Checking*, 2003.

[26] F. Lu and K.-T. Cheng, "IChecker: An efficient checker for inductive invariants," in *HLDVT*, Nov. 2006.

[27] M. Case, V. Kravets, A. Mishchenko, and R. Brayton, "Merging nodes under sequential observability," in *DAC*, June 2008.

[28] E. Cerny, A. Dsouza, K. Harer, P.-H. Ho, and H.-K. T. Ma, "Supporting sequential assumptions in hybrid verification," in *ASPDAC*, Jan. 2005.