Scalable Compositional Minimization via Static Analysis

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

Abstract

State-equivalence based reduction techniques, e.g. bisimulation minimization, can be used to reduce a state transition system to facilitate subsequent verification tasks. However, the complexity of computing the set of equivalent state pairs often exceeds that of performing symbolic property checking on the original system. We introduce a fully-automated efficient compositional minimization approach which requires only static analysis*. Key to our approach is a heuristic algorithm that identifies components with high reduction potential in a bit-level netlist. We next inject combinational logic which restricts the component's inputs to selected representatives of symbolically-computed equivalence classes thereof. Finally, we use existing transformations to synergistically exploit the dramatic netlist reductions enabled by these input filters. Experiments confirm that our technique is able to efficiently yield substantial reductions on large industrial netlists.*

1 Introduction

Automated formal verification techniques generally require exponential resources with respect to the number of state elements of a design. Numerous approaches have thus been proposed to reduce the complexity of a design under verification. For example, bisimulation minimization techniques seek to iteratively collapse equivalent states from the design's state transition representation [1, 2, 3]. While such techniques may yield large reductions in state space, they address a PSPACE-complete problem, hence often outweigh the cost of verification frameworks such as symbolic invariant checking [3].

We present CMSA—a novel fully-automated *compositional minimization* technique which requires only *static analysis*. CMSA is based upon bisimulation, hence results in sound and complete property checking. Unlike traditional approaches which operate on word-level models and/or require manual guidance, CMSA uses a novel linear-time partitioning algorithm to *automatically* isolate a component of a *bit-level* netlist. CMSA preserves the behavior of this component with respect to its outputs and any signals referenced in the property formulae therein. We will hereafter refer to these signals as *predicates*. Once selected, we equivalence-class the states of the component which are bisimilar with respect to the component's predicates, and then equivalence-class its input space under transitions to bisimilar state sets. We next synthesize this input equivalence class into the design, to restrict the component's visibility of the original input space to equivalence class representatives thereof. Finally, transformations such as sequential redundancy removal are used to exploit the reductions enabled by this input restriction.

1.1 Contributions and Related Work

CMSA relies upon identifying bisimilar states, similar to [1, 2, 3, 4]. While any bisimilar-state identification approach may be incorporated into our technique, we have developed a solution which copes with the intractability of traditional bisimulation computation in two ways: **(1)** We use static analysis—a problem in NP—to compute an underapproximation of the set of bisimilar state pairs. As noted in [2], such underapproximation identifies the majority of truly equivalent states in certain netlists. **(2)** We minimize individual *components* of the design to avoid the complexity of analyzing the entire system, similarly to [4, 1].

As we will show in Section 3, the isolation of high-quality components greatly increases the fraction of bisimilar state pairs identifiable via static analysis. This *enables* efficient and near-optimal bisimilar state identification in large industrial netlists for which traditional approaches fail. Note that the complete bisimulation relation can be computed for very small components; however, the potential for their minimization tends to be small because their internal signals are highly observable and controllable.

Predicate abstraction techniques overapproximate a design by reasoning about the set of possible transitions between valuations of selected predicates [5, 6, 7]. This overapproximation often results in spurious property failures, requiring a *refinement* step which increases the number of predicates in the abstracted system to tighten the overapproximation. For larger control-intensive netlists, the number of refinement steps may become exorbitant, and the abstracted system may exceed the complexity of the original netlist [7], often degrading to an inconclusive result. Our technique is related to predicate abstraction in that its reduction potential is dependent upon identifying a high-quality set of predicates against which to minimize. However, unlike predicate abstraction, our approach is sound *and complete* for property checking, hence avoids the need for refinement to cope with spurious failures.

Bit-Level Generality. One of fundamental contributions of CMSA is its ability to automatically process bit-level netlists, with no need for a word-level representation. This approach has several benefits:

1. It enables the selection of predicates which are of *higher* reduction quality than possible if restricted to word-level operations. For example, predicates may be identified which effectively fragment and coalesce multiple bit-level (e.g., *control*) and word-level (e.g., *data*) signals.
2. It enables *iterative reductions* through repeated applications of CMSA, possibly interspersed with arbitrary synergistic bit-level transformations such as retiming [8], redundancy

removal [9, 10], logic rewriting [11], and localization [7] under a transformation-based verification framework [8]. The reduction capability of these transformations synergistically enable the identification of higher-quality predicates by CMSA. Conversely, CMSA offers reductions which cannot be achieved by these techniques, enabling greater reductions through their subsequent application. For example, retiming may find a better register placement, and localization may render a smaller abstraction, after CMSA.

3. We have found this approach necessary to automate optimal reductions of high-performance industrial designs, which often fragment word-level operations *in the RTL model itself* to ensure latch-accuracy with the highest-performance silicon. Requiring a word-level model thus in practice often mandates a *manual rewriting* of the RTL, which limits the practical utility of such techniques.

The only other work we are aware of which combines predicate-based abstraction with other transformations is that of Wang [7]. Because the number of predicates needed over control logic tends to be very large, they propose applying predicate abstraction to simplify portions of a *localization* of the design. While powerful, their integration of these techniques is not as tight as ours because they cannot be *iteratively* applied for synergistic reductions (along with other arbitrary transformations).

Novel Reduction Potential. As we show in Section 3, CMSA enables reductions that go qualitatively beyond related techniques such as *retiming and resynthesis*, *symmetry reductions*, and *datapath reductions*. Our approach injects a combinational *filter* onto a component's inputs to restrict its visibility of the original input space to a selected representative from each of the input equivalence classes thereof. To obtain significant reductions of the resulting modified netlist, we exploit other synergistic transformations such as sequential redundancy removal [9] which can be applied in a resource-constrained manner for efficiency. CMSA thereby performs minimal structural modifications to the original netlist to enable its reductions, avoiding the costly sequential re-encoding techniques used by state minimization approaches for synthesis (e.g., [2, 4]) or bisimulation minimization (e.g., [1, 3]), which in many cases actually *increase* the complexity of the design being re-encoded. Since CMSA merely *restricts* the input space of the netlist, it also thereby trivializes counterexample generation.

Retiming and resynthesis is fundamentally limited in its reduction potential due, for example, to its inability to relocate combinational logic *across* cyclic components [12, 13]; our approach does not share this limitation. *Symmetry reduction* techniques exploit the fact that many systems are comprised of multiple instantiations of identical component types, and that the overall system may behave identically with respect to permutations of the states of those instantiations. Symmetry reduction replaces states encountered during verification with representatives from *orbit relations* of equivalent states. CMSA is capable of yielding symmetry reduction: if applied to a component whose predicates exhibit symmetry, its input filtering will inherently entail state symmetry reduction. Additionally, CMSA may be viewed as a more general paradigm: while symmetry reductions have been extended to special cases of *nearly symmetric* systems [14], CMSA inherently offers reductions to fully- as well as partially-symmetric designs – and even to completely asymmetric designs.

Datapath reductions have been extensively studied over the past decade. For example, *uninterpreted functions* [15, 16] have been proposed, which are sound yet *incomplete* approaches. *Controlled token nets* [17] are of limited expressibility, and like *integer combinational/sequential concurrency models* [18], tend to fail if the datapath fans out to the control logic, and additionally require dedicated verification algorithms. Prior work relies upon word-level design representations (which may not be available for high-performance designs), and often requires manual guidance to the abstraction process or an automated overapproximation refinement scheme [5, 6, 7, 16]. In contrast, the reductions enabled by CMSA are property-preserving, are attainable automatically from bit-level netlists, are structurally reflected hence applicable for arbitrary verification algorithms (and iteratively applicable with other synergistic transformations), and are not limited by fanin or fanout dependencies between control and data logic. CMSA was to a large part motivated by the lack of automated applicability of prior datapath reduction techniques to control-intensive high-performance industrial designs.

2 Preliminaries

In this section we provide formalisms used throughout this paper. A reader well-versed in hardware verification may wish to skip this section, using it only as a reference. Our technique operates on a *netlist* representation of the design.

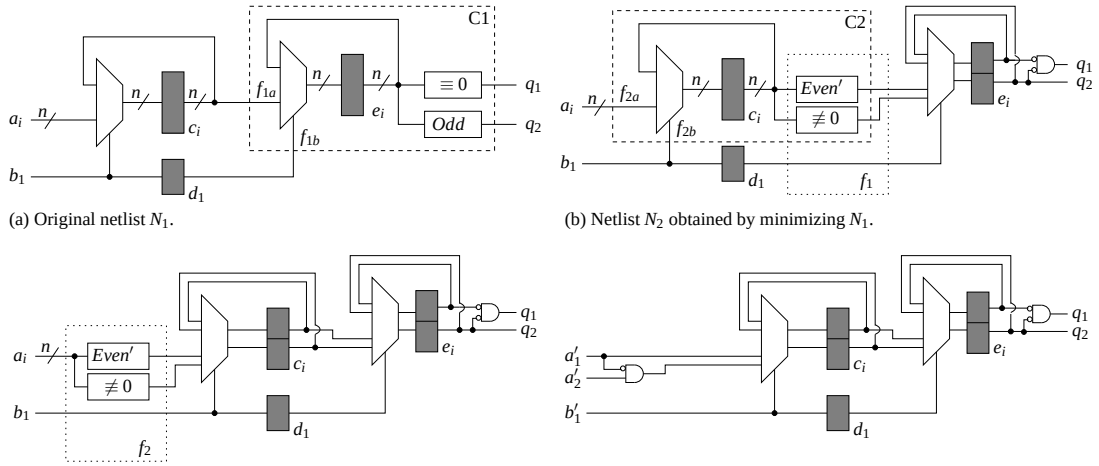
Definition 1. A *netlist* is a tuple $\langle\langle V,E\rangle,G\rangle$ comprising a directed graph with vertices V and edges $E \subseteq V \times V$. Function $G : V \mapsto types$ represents a mapping from vertices to gate *types*, including primary inputs denoted as I, registers denoted as R, and combinational gates with various functions. Registers have designated *initial values*, as well as *next-state functions*. The *semantics of a netlist* are defined in terms of semantic *traces*: $0,1$ valuations to gates over time which are consistent with G.

We will often use $Q \subseteq V$ to refer to a set of gates of interest—e.g., outputs of a component.

Definition 2. A *Moore machine* is a tuple $M = (S, J, O, \Psi, L, S_0)$, where S is a set of states, $S_0 \subseteq S$ is a set of initial states, J is a set of input propositions, O is a set of output propositions, $\Psi \subseteq S \times J \times S$ is a transition relation, and $L : S \mapsto 2^O$ is a labeling function. The Moore machine associated with a netlist is obtained by taking $S = 2^R$, $J = 2^I$, and $O = Q$, and defining S_0, Ψ, and L to be consistent with initial values, next-state functions, and the functions corresponding to gates in Q, respectively.

In the sequel, we will use $\Psi(s,i)$ as short-hand notation for $\{t : (s,i,t) \in \Psi\}$, and $\Psi(s)$ for $\{t : \exists i.(s,i,t) \in \Psi\}$. We will also interchangeably refer to a netlist and its associated machine. We define $\text{image}(\Psi, S_j) = \{t : \exists s \in S_j.\exists i.(s,i,t) \in \Psi\}$, and $\text{preimage}(\Psi, S_j) = \{t : \exists s \in S_j.\exists i.(t,i,s) \in \Psi\}$. States S_i are said to be *output-equivalent* iff $\forall s,t \in S_i.\big(L(s) = L(t)\big)$, and *image-equivalent* iff $\forall s,t \in S_i.\big(\text{image}(\Psi,s) = \text{image}(\Psi,t)\big)$.

Definition 3. Given two machines M and M', the *bisimulation relation*, denoted by $\sim$, is the coarsest relation satisfying the following. For all $s \in S$ and $s' \in S'$, $s \sim s'$ iff:



(a) Original netlist N_1.

(b) Netlist N_2 obtained by minimizing N_1.

(c) Netlist N_3 obtained by minimizing N_2.

(d) Nelist N_4 obtained by reparameterizing inputs of N_3.

Figure 1: **Compositional minimization of a two-stage gated pipeline**

1. $L(s) = L'(s')$,
2. $t \in \text{image}(\Psi, s) \implies \exists t'.\big(t' \in \text{image}(\Psi', s') \wedge t \sim t'\big)$, and
3. $t' \in \text{image}(\Psi', s') \implies \exists t.\big(t \in \text{image}(\Psi, s) \wedge t \sim t'\big)$.

We say that $M \sim M'$ iff $\forall s_0 \in S_0.\exists s'_0 \in S'_0.s_0 \sim s'_0$ and $\forall s'_0 \in S'_0.\exists s_0 \in S_0.s_0 \sim s'_0$.

Theorem 1. If states s and t are *output- and image-equivalent* (OIE), they are also bisimilar.

Proof. Condition 1 of Definition 3 follows from output equivalence. Conditions 2 and 3 follow from image equivalence. □

Definition 4. 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* $V_{\mathcal{C}} = \{u \in \mathcal{C} : \exists v \in \overline{\mathcal{C}}.\big((u,v) \in E\big)\}$ and $V_{\overline{\mathcal{C}}} = \{u \in \overline{\mathcal{C}} : \exists v \in \mathcal{C}.\big((u,v) \in E\big)\}$.

One may visualize a cut of netlist N as the composition [19] of two component netlists $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$, with $V_{\mathcal{C}}$ denoting *pseudo-inputs* to $N_{\overline{\mathcal{C}}}$ (equivalently, outputs of $N_{\mathcal{C}}$) which are closed under the composition, and with $V_{\overline{\mathcal{C}}}$ denoting pseudo-inputs to $N_{\mathcal{C}}$ (equivalently, outputs of $N_{\overline{\mathcal{C}}}$) which are closed under the composition. When minimizing a given component $N_{\mathcal{C}}$ of N, we will take its outputs $V_{\mathcal{C}}$ along with any formula signals in $N_{\mathcal{C}}$ as the predicates Q against which we will perform our abstraction.

Definition 5. An *s-t cut* is a cut seeded with 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.

Theorem 2. Let $N_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$ be a cut of netlist N. Consider the netlist $N' = N'_{\mathcal{C}} \parallel N_{\overline{\mathcal{C}}}$. If $N_{\mathcal{C}}$ is bisimilar to $N'_{\mathcal{C}}$, then N is bisimilar to N'.

Theorem 2 states that simulation precedence is preserved under machine composition, as proven in [19]. This theorem demonstrates that bisimulation minimization may be applied in a compositional fashion; i.e., that minimization of one component preserves bisimulation of the overall composition.

3 CMSA Example

We introduce an example netlist in Figure 1 to illustrate our CMSA technique, which will be formally presented in Section 4. This example has an n-bit input vector $\langle a_1, \ldots, a_n \rangle$, and a single-bit input b_1, corresponding to *data* and *control*, respectively. An n-bit register vector $\langle c_1, \ldots, c_n \rangle$ updates with data a_i when $b_1 = 1$, else holds its current value. The control input is pipelined through register d_1. When $d_1 = 1$, the n-bit register vector $\langle e_1, \ldots, e_n \rangle$ updates with the contents of c_i, else it holds its current value. The contents of e_i drive two output predicate gates q_1 and q_2. Gate q_1 evaluates to 1 iff e_i is all zeros. Gate q_2 evaluates to 1 iff e_i has odd parity. All registers have an initial value of zero.

Assume that we first minimize component C1 of N_1. Its output-equivalent states $\mathcal{E}_q$ consist of four disjoint sets of states: $\{e_i \text{ is even and nonzero}\}$ when $\langle q_1, q_2 \rangle = 00$; $\{e_i \text{ is odd}\}$ when $\langle q_1, q_2 \rangle = 01$; $\{e_i \text{ is zero}\}$ when $\langle q_1, q_2 \rangle = 10$; and $\emptyset$ when $\langle q_1, q_2 \rangle = 11$. The image-equivalent states $\mathcal{E}_T$ are $\{2^{e_i}\}$ since the *pseudo-inputs* f_{1a} and f_{1b} induced by the isolation of C1 enable transitioning e_i to all possible states. The *output- and image-equivalent* (OIE) states $\mathcal{E}_{qT}$ are the pairwise intersection of these two sets, equivalent in this case to $\mathcal{E}_q$. We may use this data to compute an equivalence class on the set of input valuations on a per-state basis. In particular, for every state s, we may compute the set of all input valuations which transition s to each partition of OIE states. More formally, we compute $\mathcal{I}_T(s) = \{i : \exists t \in T.(s,i,t) \in \Psi\}$ for each state s and OIE state set $T \in \mathcal{E}_{qT}$. Since in this example the $\mathcal{I}_T(s)$ sets are disjoint, no further partitioning is needed and we may arbitrarily select a representative i' from each of the equivalence classes in $\mathcal{I}_T$, and form relation $I_T \subseteq 2^R \times 2^I \times 2^I$ where $(s,i,i') \in I_T$ implies that $t = \Psi(s,i)$ and $t' = \Psi(s,i')$ are in the same OIE state set. Finally, we synthesize this relation, and inject the resulting *filter logic* onto the inputs f_{1a} and f_{1b} of this component. This structural transformation restricts the component's visibility of its original inputs to the selected representatives from the equivalence classes thereof, while preserving bisimilarity as per Theorem 1.

When constructing I_T, assume that we select input valuations $f_{1a} = 0^{n-2}11$ to preserve transitions to the OIE set where $\langle q_1, q_2 \rangle = 00$; $f_{1a} = 0^{n-1}1$ to preserve transitions to the OIE set

where $\langle q_1, q_2 \rangle = 01$; and $f_{1a} = 0^n$ to preserve transitions to the OIE set where $\langle q_1, q_2 \rangle = 10$. The resulting minimized netlist is depicted as N_2. Logic f_1 represents the input filter injected by the transformation. Predicate *Even'* refers to a check of nonzero yet even parity. We additionally have deployed redundancy removal on this netlist to propagate the input filtering through C1. Because the filtering mapped all but the two least-significant bits of inputs f_{1a} to 0, we reduce the component from n to 2 registers. We additionally simplified the logic driving q_2. Note that this transformation is similar to an *architectural retiming* of the output predicates across the memory bank. This is a transformation which classical netlist-based retiming and resynthesis cannot achieve due to the placement of the feedback cycles [12], illustrating how CMSA subsumes certain retiming and resynthesis moves.

We next minimize component C2 of N_2. This minimization is carried out similarly to that of C1, which after redundancy removal yields netlist N_3, reducing c_i from n to 2 registers. Finally, we may *reparameterize* the input space of N_3 to reduce a_i from n to 2 inputs, resulting in netlist N_4. While transformation of component inputs is generally performed by insertion of filter logic (which does not reduce input count), in the special case that a component input is a true netlist input versus a pseudo-input induced by the partitioning, we may reparameterize I_T to reduce input count, as will be discussed in Section 4.2. Note that this overall transformation correlates to an optimal datapath reduction.

It is noteworthy that CMSA is capable of yielding reductions to more complex netlists, e.g., with data transformations and more complex control logic, even along the feedback paths. The chosen example is deliberately simple for clarity of exposition. Note also that—without component extraction—the set of image-equivalent states of N_1 would be partitioned to the extent that no data reductions would be possible. However, input b_1 may be tied to a 1. This reduction is also possible for component C2 through an alternate selection of input representatives, though it is not illustrated for simplicity. This demonstrates how CMSA subsumes datapath reductions. Such reduction capability is particularly useful on modern high-performance designs, where *clock-gating* for low-power tends to turn even feed-forward pipelines into structures such as N_1, rendering techniques such as retiming ineffective in their simplification.

4 Compositional Minimization via Static Analysis

In this section we discuss the algorithms for performing our predicate-based minimization, and prove their correctness.

Theorem 3. Let T be a set of pairwise bisimilar states, i.e., $\forall t,t' \in T.t \sim t'$. For any $s \in \text{preimage}(\Psi, T)$, define $\jmath_T(s) = \{i : \exists t \in T.(s,i,t) \in \Psi\}$. For any input $i^a \in \jmath_T(s)$ let $\Psi^a = \{(s,i,t) : t = \Psi(s,i^a)$ *if* $i \in \jmath_T(s), \Psi(s,i)$ *otherwise*$\}$. The two machines $M = (2^R, 2^I, 2^Q, \Psi, L, S_0)$ and $M^a = (2^R, 2^I, 2^Q, \Psi^a, L, S_0)$ are bisimilar.

Proof. For the final two conditions of Definition 3, let $t^a = \Psi(s,i^a)$, and consider any $t \in \Psi(s)$. If $t \in T$, then $t \sim t^a$ by construction. Otherwise $t \in \Psi(s) \setminus T$, hence $t \in \Psi^a(s)$ because Ψ^a only prunes transitions to elements of T. Condition 1 follows since the machines have the same initial states. It follows that $M \sim M^a$. □

$\mathcal{E}_q$	$= \bigwedge_{j=1}^{\lvert Q \rvert}(q_j \equiv l_j)$;	// support: Q,R
$\mathcal{E}_T$	$= \exists I.\bigwedge_{j=1}^{\lvert R \rvert}(\hat{r}_j \equiv f_j)$;	// $\hat{R}$, R
$\mathcal{E}_{qT}$	$= \mathcal{E}_q \wedge \mathcal{E}_T$;	// Q,R,$\hat{R}$
PreImg	$= (\mathcal{E}_{qT}\lvert_{r_i = r_i''})\lvert_{r_i'' = (r_i' \equiv f_i)}$;	// Q,R,R',$\hat{R}$,I
PreImg^a	$= \mathit{project}\big((\hat{R} \cup Q), \text{PreImg}\big)\lvert_{i_j = i_j'}$;	// Q,R,R',$\hat{R}$,I'
I_T	$= \exists Q, \hat{R}, R'.(\text{PreImg} \wedge \text{PreImg}^a)$;	// R,I,I'
return I_T;		

Figure 2: **Algorithm ComputeInputEquivalence**

Theorem 3 demonstrates the correctness of the input-restricting transformation of CMSA. Note that this transformation may violate *image equivalence*: a set of image-equivalent states S_i in M may not remain image-equivalent in M^a, because we may select differing i^a and thereby images for some of them. However, bisimulation is nonetheless preserved by this input restriction.

Corollary 1. Consider machine $M^{a'}$ formed by iteratively applying a sequence of input reductions to M using the procedure of Theorem 3. Machine $M^{a'}$ is bisimilar to M.

Corollary 1 is a consequence of Theorems 2 and 3, illustrating that we may iteratively apply this input-restricting transformation to arbitrary individual components of the netlist and still preserve its compositional bisimilarity.

4.1 Computing the Input Equivalence Relation I_T

As per Theorem 3, the core of CMSA relies upon computing equivalence classes of input valuations that preserve bisimilarity, represented as a relation I_T. To detail our algorithm for computing I_T, we formalize the notation introduced in the example of Section 3. The partition of output-equivalent states is denoted as $\mathcal{E}_q = \bigcup_{p \in 2^Q}\{\{s : L(s) = p\}\}$. We denote the partition of image-equivalent states as $\mathcal{E}_T = \bigcup_{S_i \subseteq S}\{\{s : \text{image}(\Psi, s) = S_i\}\}$. The partition of OIE states is denoted as $\mathcal{E}_{qT} = \bigcup_{S_i \in \mathcal{E}_q,\ S_j \in \mathcal{E}_T}\{\{S_i \cap S_j\}\}$. We wish to compute I_T, which equivalence-classes concrete input valuations i to representative input valuations i' on a per-state basis, such that $(s,i,i') \in I_T$ implies that $t = \Psi(s,i)$ and $t' = \Psi(s,i')$ are both in the same bisimilar state set $S_{qT} \in \mathcal{E}_{qT}$.

The symbolic algorithm presented in Figure 2 demonstrates how we may efficiently compute I_T for a netlist component. We use variables $Q = \langle q_1, \ldots, q_m \rangle$ to capture valuations to our predicates, whose functions over the registers in their combinational fanin is denoted by $L = \langle l_1, \ldots, l_m \rangle$. Similarly, we denote present-state register variables as $R = \langle r_1, \ldots, r_n \rangle$, whose next-state functions are denoted $F = \langle f_1, \ldots, f_n \rangle$. We will use variables $\hat{R}$, correlating to R, to represent equivalence relations of states.

Set $\mathcal{E}_q$ represents the output-equivalent states (over R) as a relation with the predicate variable valuations (over Q). In particular, for any valuation to Q, set $\mathcal{E}_q$ provides exactly those states producing that output valuation. Set $\mathcal{E}_T$ represents image-equivalent states as a relation over variables R and $\hat{R}$. This set is computed by quantifying inputs from the equivalence relation of the next-state functions and their valuations denoted with $\hat{R}$ variables, capturing the relation of present states (over R) which transition to identical next states. $\mathcal{E}_{qT}$ represents the OIE states, and PreImg reflects states and input valuations that transition to these OIE states.

PreImg is computed by substituting temporary R'' variables for R variables, then composing into R'' the equivalence of the next-state functions F with R'. This will symbolically partition input sets to refine each $\jmath_T(s)$ into multiple sets $\jmath'_T(s) = \{i : i \in \jmath_T(s) \wedge \forall T_1 \in E_T . \forall x,y \in T_1 . \exists T_2 \in E_{qT} . \big(\Psi(x,i) \in T_2 \wedge \Psi(y,i) \in T_2\big)\}$. This refinement step, which resembles a check in a fixed-point computation, ensures that the input equivalence, defined over a state s by sets $\jmath_T(s)$, also forms an equivalence over sets of image-equivalent states. This is necessary because we limit ourselves to injecting an input filter onto the component inputs instead of synthesizing the product of the sequential input filter with the transformed machine. If the check fails, each $\jmath_T(s)$ will be fragmented into smaller disjoint sets defined by $\jmath'_T(s)$ to force an input equivalence relation. Intuitively, one input can not represent two state-input pairs where the states happen to be image-equivalent, if the corresponding next-states are not bisimilar. Typically no such refinement is needed in data-oriented designs such as the example of Figure 1. Note also that variables R' are redundant in the relation since they are encoded by the next-state functions over R and I hence can be quantified for an optimal computation.

Function *project* is based upon the *compatible projection* operator [20] that is used to symbolically map equivalence classes to representatives thereof. PreImg^a uses *project* to collapse PreImg down to representatives over I variables, to equivalence-class transitions from present states (over R) and their image-equivalent states (over R') to bisimilar next-state representatives (over $\hat{R}$ and Q). We additionally map input variables i_j to i'_j in PreImg^a, so that the conjunction of PreImg^a and PreImg represents equivalence classes of input valuations (over i) and representatives thereof (over i') between transitions from *present states* (over R) to bisimilar *next-states* (over $\hat{R}$ and Q). Quantifying Q and $\hat{R}$ from the resulting conjunction forms our equivalence relation $I_T \subseteq 2^R \times 2^I \times 2^I$ between original and representative inputs on a per-state basis, where $(s,i,i') \in I_T$ implies that input stimulus i may be mapped to i' in state s while preserving bisimilarity.

We chose Moore machines in formalizing CMSA for simplicity of the exposition. This approach can readily be extended to *Mealy machines* by considering inputs in the combinational fanin of the predicates as predicates themselves. Another useful and straight-forward extension addresses the reduction of symbolic initial states, e.g., if the initial values are non-constant. To briefly sketch this extension: we construct a relation $R_T \subseteq 2^R \times 2^R$ that maps initial states to their bisimilar states. We next project each equivalence class of initial states in S_0 to a selected bisimilar representative s'_0, and substitute initial value functions into register variables appearing in S_0 to reflect the restricted initial value functions for the registers appearing in S'_0.

4.2 Transforming the Netlist under I_T

The algorithm of Figure 2 illustrates how we may symbolically compute I_T, which represents a mapping between original input valuations and a bisimulation-preserving restricted subset on a per-state basis. To form the transformed netlist, we alter the structure of the original netlist to reflect the corresponding restricted input space. In particular, we synthesize the ⟨*present state, original input, restricted input*⟩ relation of I_T to equivalence-class the visibility of the transformed component to its original inputs.

Our algorithm for transforming the netlist utilizes a relation-synthesis approach similar to that of [21]. Their approach converts an input-output relation $T(x,y)$ into a set of combinational *output gates* that preserve valuations of y as the corresponding function of input gates x. In our case, x correlates to the original inputs to the component i and its register variables s, and y correlates to the input equivalence class representatives i'. Note that our output variables y will be a deterministic function of the input variables x. Once the *filter logic* is synthesized, we replace all fanout references of the original inputs by the corresponding output gates to *inject* that filter logic onto the component's inputs.

Note that our transformed netlist retains all of its inputs and registers; the only transformation is the injection of combinational filter logic to restrict the fanout visibility of the transformed netlist to its inputs. We will detail the impact of this restriction in Section 4.3. However, in the special case that an input variable correlates to an input of the original netlist (versus a pseudo-input induced by extracting a component), we may existentially quantify such true input variables from the I_T relation prior to synthesis. This may cause certain output variables to become nondeterministic functions of the remaining variables. The approach of [21, 22] will *parameterize* such output gates using fresh input gates to reflect the nondeterminism. The original quantified input will then be discarded from the netlist. Such a transformation was used to obtain the netlist of Figure 1d from that of Figure 1c.

4.3 Reducing the Transformed Netlist

In cases, $\jmath'_T(s)$ may reflect a different input valuation set for nearly each state s, hence nearly the entire input space may need to be preserved precluding a useful reduction. However, it is often the case that high-quality predicate selection will enable the vast majority of the original input space 2^I to be discarded across all states. More formally, given $I^a = \{i' : \exists s . \exists i . \exists T . (s,i,i') \in I_T(s)\}$, it is often the case that $|I^a| \ll |2^I|$. In this section, we discuss how we construct I_T from $\jmath'_T(s)$.

Our transformation explicitly restricts only the netlist's visibility to its input space through the injection of filter logic. This restriction itself may simplify subsequent analysis of the transformed netlist. For example, if performing symbolic reachability analysis, many state variables may become constant across all reachable states through this transformation, which may reduce peak BDD sizes and variable ordering resources. However, the primary goal of CMSA is to enable significant *structural* reductions of the transformed netlist using low-cost transformations such as redundancy removal [9, 10]. Such techniques are capable of structurally reflecting the fact that CMSA causes internal gates to become constant and/or deterministic functions of other gates – which may generally not have been the situation before the abstraction – by *merging* those newly redundant gates and significantly reducing netlist size. This merging in turn tends to reduce resources for arbitrary subsequent algorithms. In a transformation-based verification framework [8], one may furthermore iteratively apply CMSA interspersed with arbitrary synergistic transformations to yield iterative reductions.

To heuristically maximize the effectiveness of subsequent redundancy removal, our approach attempts to select the elements i' of I_T to assign as many inputs as possible to a constant. For inputs for which a constant value cannot be so assigned, we attempt to define them as a deterministic function of other input variables –

$\hat{A}_0 = \{r\}$; $B_0 = \emptyset$;
for ($j = 0$; $(j \equiv 0) \vee (\hat{A}_j \not\equiv \hat{A}_{j-1})$; j++)
    $\hat{B}_{j+1} = \hat{B}_j \cup \text{fanout}(\hat{A}_j)$;
    $\hat{A}_{j+1} = \hat{A}_j \cup \text{fanin}(\hat{B}_{j+1})$;
$\hat{B} = \hat{B}_j$; $\hat{A} = \hat{A}_j$;
if($\hat{A} \cap \hat{B} \equiv \emptyset$) **return** s-t mincut($\hat{A}, \hat{B}$);
$A_0 = \{r\}$; $B_0 = \emptyset$;
for ($j = 0$; $(j \equiv 0) \vee (A_j \not\equiv A_{j-1})$; j++)
    $B_{j+1} = B_j \cup \{\text{fanout}(A_j) \setminus \text{fanin}(A_j)\}$;
    $A_{j+1} = A_j \cup \{\text{fanin}(B_{j+1}) \setminus \text{fanout}(B_{j+1})\}$;
$A = A_j$; $B = B_j$;
return s-t mincut(A, B);

Figure 3: **Algorithm FindPredicates, relative to seed** r

preferably selecting input i_j to be equivalent (modulo inversion) to some input i_k. Such input restrictions tend to in turn cause more internal gates to become redundant, which may then be eliminated through redundancy removal. Algorithmically, we form I_T symbolically using a modified version of the compatible projection operator [20]. In particular, as we select each i' (which collectively define I_T), we use the heuristic of a minimal Hamming distance (modulo inversion) to already-selected representatives.

4.4 Selecting Components with High-Quality Predicates

The reduction potential of our technique is clearly dependent upon the *quality* of the selected predicates to enable an input restriction where I^a is much smaller in cardinality than 2^I. In this section, we discuss our heuristic algorithm for selecting components with high-quality predicates against which to compute equivalence. Note also that component partitioning is an effective tactic for reducing the expense of the equivalence computation.

Our primary metric for estimating the quality of a component is that its set of outputs (V_C) should be significantly smaller in cardinality than its set of pseudo-inputs ($V_{\overline{C}}$). Such a criterion heuristically increases the probability that the resulting component will have high-cardinality elements of $\jmath_T(s)$, in turn increasing the probability that I^a may be selected so as to enable a significant structural reduction through redundancy removal.

Our algorithm for identifying a high-quality component containing seed register r is provided in Figure 3. Function *fanin*(x) returns the set of registers in the combinational fanin of the next-state function of x, and *fanout*(x) returns the set of registers and formula signals in the combinational fanout of x. The corresponding component is defined by the computation of minimally-sized register boundaries $A \supseteq r$ and B, where A is a superset of all registers in the combinational fanin of the next-state functions of B, and B is a superset of all registers in the combinational fanout of A. We compute a combinational min-cut between A and B; the cut gates V_C represent the outputs of the selected component. Intuitively, the min-cut attempts to find a minimal set of predicates between a component (containing only registers in A) and the logic that this component fans out to (containing registers B). Once computed, if the resulting cut yields $|V_C|$ substantially smaller than $|A|$, we will consider the component as a candidate for minimization.

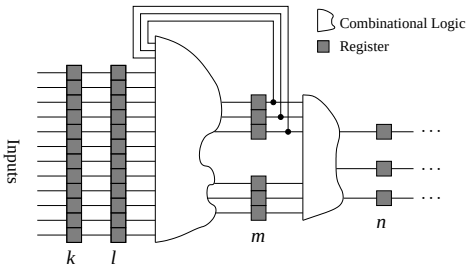


Figure 4: **Predicate selection example**

We present the algorithm of Figure 3 using two fixed-point computations for clarity of exposition; practically, the second subsumes the first. The first fixed-point approximates A and B as sets $\hat{A}$ and $\hat{B}$, respectively. Consider the application of this algorithm on the netlist of Figure 4, relative to a seed l_i from l. Initially, we will obtain $\hat{A}_0 = \{l_i\}$, and $\hat{B}_1 = \{m\}$. On the next iteration, we will obtain $\hat{A}_1 = \{l, m\}$, because both l and m are in the combinational fanin of $\hat{B}_1$. We thus obtain $\hat{B}_2 = \{m, n\}$, due to the addition of m to $\hat{A}_1$. Intuitively, we see that this is likely to be a suboptimal result, because we have grouped l and m into the same component, whereas we would prefer to reduce the component containing only registers l relative to predicates in their combinational fanout. Furthermore, we also could reduce the component containing only registers m relative to predicates in their combinational fanout – which may in turn create even higher-quality predicates against which we may reduce the component containing l. This sub-optimality motivates the second fixed-point.

The cause of the prior sub-optimality was that duplicate elements were obtained between $\hat{A}$ and $\hat{B}$. The second loop enforces disjointness by preventing the addition of registers in self-loops involving A into B, and vice-versa. Going back to the example of Figure 4, this will ensure that $A_1 = \{l\}$. Note that m will not be added to A_1 due to these restrictions, hence the process will terminate with $A = \{l\}$ and $B = \{m\}$, as desired. Since $|B| = 6$, there exists a min-cut of cardinality at most 6, which is significantly smaller than $|A| = 12$, making this a candidate for minimization.

To further illustrate this algorithm, consider other seeds from the example of Figure 4. Each seed k_i of k will have only a single l_i in its combinational fanout – hence A and B will have cardinality 1, as will V_C. We will thus neglect considering these as candidates. Next consider a seed m_i from m. We will have $\hat{A}_0 = \{m_i\}$, and $\hat{B}_1 = \{m, n\}$; then $\hat{A}_1 = \{l, m\}$ and $\hat{B}_2 = \{m, n\}$. The second fixed-point will prevent m from becoming part of $B_1 = \{n\}$, in turn preventing l from becoming part of $A_1 = \{m\}$. Since $|B| = 3$, there exists a min-cut of cardinality at most 3, which is significantly smaller than $|A| = 6$, making this a candidate for minimization.

Consider the application of this algorithm to the netlist of Figure 1a, assuming that $\{q_1, q_2\}$ are formula signals. Relative to a seed in c_i, we will obtain $\hat{A}_0 = \{c_i\}$ and $\hat{B}_1 = \{c_i, e_i\}$, then $\hat{A}_1 = \{c_i, d_i, e_i\}$ and $\hat{B}_2 = \{c_i, e_i, q_1, q_2\}$. The second fixed-point will render $A = \{c_i, d_i\}$ and $B = \{e_i\}$. This is not a very promising candidate for minimization, as the min-cut between these sets is of cardinality n – only one less than that of A. Relative to a seed in e_i, we obtain $\hat{A}_0 = \{e_i\}$ and $\hat{B}_1 = \{e_i, q_1, q_2\}$, then $\hat{A}_1 = \{c_i, d_i, e_i\}$ and $\hat{B}_2 = \{c_i, e_i, q_1, q_2\}$. The second fixed-point will terminate with $A = \{e_i\}$ and $B = \{q_1, q_2\}$. Our min-cut will thus yield $|V_C| = 2 \ll |A| = n$, hence this is a good candidate for minimization. Referring to the reduced netlist of Figure 1b, if we now select

Design Name	Original Size			Reduced Size			Components	Mem / CPU (MB) / (sec)
	Inputs	Regs	ANDs	Inputs	Regs	ANDs		
IFPF [i]	147	623	2741	109	232	1087	14	76.0 / 748.3
IFU [i]	992	4224	24652	693	1202	9745	8	169.8 / 984.8
GCT [i,d]	439	810	24127	312	431	18078	11	39.4 / 402.2
ERAT [d]	144	4281	24250	106	2703	21603	7	12.0 / 1020.0
EMQ [d]	89	127	682	34	104	533	2	13.7 / 0.1
RING [d]	9763	8343	137775	8343	4598	137992	23	57.9 / 183.2
SLB [d]	76	155	1118	23	78	873	3	23.8 / 30.4
FPU	218	2174	29078	218	1469	19627	14	184.0 / 632.6
BW	70	1248	4264	68	522	3012	7	20.5 / 35.4
DCLA	67	252	2618	67	155	2398	3	91.0 / 238.6
IBM_17_2	49	325	1584	49	317	2059	2	12.6 / 145.9
IBM_31_1	52	122	1675	28	102	1257	6	132.4 / 95.4
IBM_31_2	52	122	1425	45	36	769	11	120.0 / 93.7
IBM_27	102	99	656	29	7	110	14	52.8 / 53.2
IBM_17_1	45	263	1365	12	15	240	7	2.5 / 40.2

Table 1: **CMSA reduction results**

a seed from c_i, we again will obtain $A = \{c_i, d_i\}$ and $B = \{e_i\}$. At this stage, however, the prior minimization provides us a min-cut at f_1 with $|V_C| = 2 \ll |A| = n$, hence this is now a good candidate for minimization.

We select our seeds by iteratively applying the algorithm of Figure 4 to every register in the netlist which has not already been considered as an element of A. Note that the order in which seeds are considered does not alter the resulting components.

Unlike the resource-intensive circuit partitioning algorithms for synthesis [23], ours extracts components to minimize V_C with no regard for $V_{\overline{C}}$ due to the nature of our minimization. We may thus restrict our analysis to combinational logic slices. Note that the overall component selection process may readily be tuned to run in linear time, noting the following: **(1)** No gate needs to be considered as an element of A twice, nor as an element of B twice. **(2)** Relative to a given seed, we only need to traverse each gate twice fanout-wise and twice fanin-wise. **(3)** While min-cut algorithms cannot be *guaranteed* to yield linear complexity bounds even on graphs without directed cycles, we may ensure linearity with minimal sacrifice in optimality as follows. We use the *augmenting-path* algorithm [24], which we have always observed to yield linear run-times even on the largest netlists. A linear overall run-time may be guaranteed by bounding the number of iterations of the *augmenting-path* algorithm; if prematurely terminated, this algorithm still offers a high-quality cut (though one which may be slightly suboptimal).

Optimality of Bit-Level Predicate Selection. We have found that this component selection algorithm efficiently detects cuts which correlate to word-level predicates of the netlist (e.g., a vector equality check $X \equiv Y$), while operating solely upon the bit-level netlist. In many cases, it finds significantly *better* predicates due to its ability to coalesce bit-level control nets into high-quality predicates. This bit-level algorithm requires negligible resources even on large netlists, while offering the unique benefits mentioned in Section 1. Consequently, *we have not found the loss of word-level predicates in the bit-level netlist to be of any significance* to achieve powerful predicate-based reductions.

5 Experimental Results

In this section we provide experimental results illustrating the reduction potential of CMSA. All experiments were run on a 1.7GHz Pentium 4 machine with 1GB memory, using the IBM internal transformation-based verification tool *SixthSense*. Though CMSA offers substantial reduction potential to a variety of netlist types (e.g., up to 79% register and input reductions on ISCAS89 benchmarks), for brevity, we will focus only on more meaningful verification problems in this section.

Table 1 summarizes our reduction experiments. The first column indicates the name of the netlist under verification. The last five are elements of the IBM FV Benchmarks [25]. The others are difficult control-intensive industrial property checking examples. IFPF is a prefetch unit. IFU and GCT are instruction fetching and completion units, respectively. ERAT, SLB, DCLA, and EMQ are address-translation units. FPU is a pipelined floating-point unit. RING is a networking unit. BW is a branch management unit. Those suffixed with [d] include data-routing logic. The *Original Size* columns indicate the size of the netlist *after* cone-of-influence reductions and aggressive redundancy removal [9]. Our netlist representations are comprised of only constants, inputs, two-input AND gates, inverters, and registers. The *Reduced Size* columns indicate the size after applying CMSA to the reduced netlist. The *Components* column indicates the number of distinct minimized components. The final column reflects the run-time and peak memory requirements to obtain the reduction results. Note that the peak memory consumption is modest on all of these examples, implying that the BDD-based analysis did not become prohibitively expensive.

These results clearly illustrate that CMSA offers significant reduction potential, yielding an additional 34.2% input reductions and 50.9% register reductions on average to the *already-reduced* netlists. Though our approach risks increasing AND count through the addition of filter logic, it almost always *reduces* AND count, here averaging 31.8%. In several of these examples (marked with [i]), CMSA yielded iterative reductions that leveraged the higher-quality predicates created during prior applications of CMSA. This illustrates the benefit of our bit-level partitioning algorithm over prior word-level approaches to exploit synergistic transformations to structurally expose higher-quality bit-level

predicates against which to compute equivalence. In several of the data-oriented designs such as SLB, our automated reductions actually *exceeded* those thought optimal using manual abstractions.

6 Conclusion

In this paper, we have presented a novel *compositional minimization via static analysis* (CMSA) technique. This technique is based upon bisimulation minimization, hence is sound and complete for property checking. Our fully-automated technique operates solely on a bit-level netlist format, and offers reduction potential qualitatively beyond that possible with other approaches such as datapath reductions, symmetry reductions, and retiming. Our technique requires only *symbolic static analysis*, hence efficiently scales to large industrial netlists.

Overall, we have found this technique able to efficiently yield substantial reductions to a variety of design styles. In many cases, this minimization yielded significant reductions even after alternate reduction flows involving iterative localization, retiming, and resynthesis had been exhausted. Our minimization thereafter synergistically enabled additional reduction potential through those transformations under a transformation-based verification framework. CMSA has become a critical technique to automatically reduce complex verification problems, in cases enabling a conclusive result (proof or counterexample) which otherwise was unattainable. For example, we were unable to prove our BW example without using this approach; in contrast, iterative application of CMSA with retiming and localization sufficiently reduced this example to enable a low-cost reachability computation.

For future work, we are investigating alternate component selection heuristics, as well as simple state re-encoding techniques. We also would like to explore the applicability of this method to the synthesis domain.

Acknowledgments. The authors would like to thank Viresh Paruthi, Geert Janssen, Jessie Xu, Mark Williams, Hari Mony, Robert Kanzelman, and Ali El-Zein for contributions to the verification framework used in the experiments.

References

[1] A. Aziz, V. Singhal, G. Swamy, and R. Brayton, "Minimizing interacting finite state machines: A compositional approach to language containment," in *International Conference on Computer Design*, Oct. 1994.

[2] B. Lin, H. J. Touati, and A. R. Newton, "Don't care minimization of multi-level sequential logic networks," in *Design Automation Conference*, June 1990.

[3] K. Fisler and M. Vardi, "Bisimulation and model checking," in *Correct Hardware Design and Verification Methods*, pp. 338–341, Sept. 1999.

[4] S. Sinha, A. Kuehlmann, and R. K. Brayton, "Sequential SPFDs," in *Int'l Conference on Computer-Aided Design*, pp. 84–90, Nov. 2001.

[5] S. K. Lahiri, E. Randal, E. Bryant, and B. Cook, "A symbolic approach to predicate abstraction," in *Computer-Aided Verification*, pp. 141–153, July 2003.

[6] E. Clarke, O. Grumberg, M. Talupur, and D. Wang, "Making predicate abstraction efficient," in *Computer-Aided Verification*, pp. 126–140, July 2003.

[7] D. Wang, *SAT based Abstraction Refinement for Hardware Verification*. PhD thesis, Carnegie Mellon University, May 2003.

[8] A. Kuehlmann and J. Baumgartner, "Transformation-based verification using generalized retiming," in *Computer-Aided Verification*, July 2001.

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

[10] A. Kuehlmann, M. Ganai, and V. Paruthi, "Circuit-based Boolean reasoning," in *Design Automation Conference*, pp. 232–237, June 2001.

[11] P. Bjesse and A. Boralv, "DAG-aware circuit compression for formal verification," in *Int'l Conference on Computer-Aided Design*, Nov. 2004.

[12] H. Zhou, V. Singhal, and A. Aziz, "How powerful is retiming?," in *Int'l Workshop on Logic & Synthesis*, May 1998.

[13] G. D. Micheli, "Synchronous logic synthesis: Algorithms for cycle-time minimization," *IEEE Transactions on Computer-Aided Design*, vol. 10, Jan. 1991.

[14] E. A. Emerson and R. J. Trefler, "From asymmetry to full symmetry: New techniques for symmetry reduction in model checking," in *Correct Hardware Design and Verification Methods*, 1999.

[15] R. Hojati, A. Isles, D. Kirkpatrick, and R. Brayton, "Verification using uninterpreted functions and finite instantiations," in *Formal Methods in Computer-Aided Design*, Nov. 1996.

[16] Z. S. Andraus and K. A. Sakallah, "Automatic abstraction and verification of Verilog models," in *Design Automation Conference*, June 2004.

[17] P.-H. Ho, A. J. Isles, and T. Kam, "Formal verification of pipeline control using controlled token nets and abstract interpretation," in *Int'l Conference on Computer-Aided Design*, Nov. 1998.

[18] A. Isles, R. Hojati, and R. Brayton, "Computing reachable control states of systems modeled with uninterpreted functions and infinite memory," in *Computer-Aided Verification*, 1998.

[19] O. Grumberg and D. E. Long, "Model checking and modular verification," *ACM Transactions on Programming Languages and System*, vol. 16(3), 1994.

[20] B. Lin and A. R. Newton, "Implicit manipulation of equivalence classes using binary decision diagrams," in *International Conference on Computer Design*, Oct 1991.

[21] J. H. Kukula and T. R. Shiple, "Building circuits from relations," in *Computer-Aided Verification*, pp. 113–123, July 2000.

[22] J. Baumgartner and H. Mony, "Maximal input reduction of sequential netlists via synergistic reparameterization and localization strategies," in *Correct Hardware Design and Verification Methods*, Oct. 2005.

[23] C. Alpert, J.-H. Huang, and A. Kahng, "Multilevel circuit partitioning," *IEEE Transactions on Computer-Aided Design*, vol. 8, no. 17, pp. 655–667, 1998.

[24] L. R. Ford and D. R. Fulkerson, "Maximal flow through a network," *Canadian Journal of Mathematics*, vol. 8, pp. 399–404, 1956.

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