Coverage in Interpolation-based Model Checking

Hana Chockler
IBM Haifa Research Lab

Daniel Kroening
University of Oxford

Mitra Purandare*
ETH Zürich

ABSTRACT

Coverage is a means to quantify the quality of a system specification, and is frequently applied to assess progress in system validation. Coverage is a standard measure in testing, but is very difficult to compute in the context of formal verification. We present efficient algorithms for identifying those parts of the system that are covered by a given property. Our algorithm is integrated into state-of-the-art SAT-based Model Checking using Craig interpolation. The key insight of our algorithm is to re-use previously computed inductive invariants and counterexamples. This re-use permits a quick conclusion of the vast majority of tests, and enables the computation of a coverage measure with 96% accuracy with only 5x the runtime of the Model Checker.

Categories and Subject Descriptors

B.6.3 [**Logic Design**]: Design aids—*Verification*

General Terms

Verification, Algorithms

1. INTRODUCTION

Model Checking is an algorithmic method for deciding whether a given design satisfies a given formal property [7, 9]. In case the design violates the property, the Model Checker provides a counterexample trace that demonstrates how the property can be violated [8]. The designer subsequently fixes the design (or reconsiders the property).

On the other hand, when the answer to the Model Checking query is positive, most tools terminate with no further feedback. However, properties used in Model Checking are rarely a full specification of the design's requirements. It is therefore good practice to suspect the design of containing an error even when the verification of the properties succeeds. Properties are written manually, and their completeness depends entirely on the competence of the verification engineer.

A sanity check for the completeness of the set of properties is to measure the *coverage* of the properties. The use of coverage metrics is commonplace in testing. As testing all conceivable executions of a design is infeasible, there is a need for a measure of exhaustiveness of a given suite of test vectors [1]. There has been extensive research in the simulation-based verification community on coverage metrics which provide a heuristic measure of exhaustiveness of the test suite [18]. In this context, coverage metrics answer the question "*Have I written enough tests?*".

The basic approach to coverage in testing, which is recording which parts of the design were exercised during the execution, cannot be used in formal verification because formal methods are exhaustive. In formal verification, coverage metrics accompany a positive answer of the verification process and are used as an indication of completeness of the *specification*. The motivation is clear: an erroneous behavior of the design can escape the verification effort if this behavior is not captured by the specification. In fact, these unspecified behaviors often elude the attention of the designer, and are thus likely to contain bugs. Intuitively, coverage metrics answer the question "*Have I written enough properties?*".

The earliest research on coverage in Model Checking suggested coverage metrics based on *mutations*, which are small "atomic" changes to the design. A mutation of the design is said to be covered by a property if the original design satisfies the property, whereas the introduction of the mutation renders the property invalid [14]. This approach is used in the vast majority of later papers on coverage in formal verification [4, 3, 2, 5, 15, 16]. In case of a design given as a net-list, an atomic mutation corresponds to replacing a signal by a constant or a new primary input. The resulting net-list is called the *mutant*.

The straightforward way to measure coverage is then to Model Check the property for each of the mutant designs; due to the sheer number of conceivable mutations, this approach is prohibitively expensive even on medium-size designs. The existing papers on coverage in Model Checking attempt to alleviate the complexity of computing coverage by adding non-deterministic BDD variables for mutations, so that the symbolic encoding of the design contains both the original and the mutant designs [4, 5], and by exploiting similarity between mutants in enumerative Model Checking algorithms, so that only a (small) part of the Model Check-

*Supported by the Semiconductor Research Corporation (SRC) under contract no. 2006-TJ-1539 and by the EU FP7 STREP MOGENTES (project ID ICT-216679).

DAC 2010, June 13-18, 2010, Anaheim, California, USA.

ing process needs to be repeated for each mutant [4].

An alternative approach to measuring coverage is to check whether each output is fully determined by the specification given a combination of input values [11, 6]. In [13], a similar check is performed using BMC tools. We note that this technique is very different from mutation coverage and neither one implies the other.

Contribution: In this paper, we present a novel algorithm that computes the coverage of a given property by means of Craig interpolation, which is the state-of-the-art technique for checking reachability properties [17]. The key insight is to perform an inexpensive analysis of the (expensive) Craig interpolant and relate it to nodes in the net-list. This enables re-use of the interpolant for other parts of the net-list.

The algorithm is implemented and has been evaluated using a broad range of circuits. The experimental results demonstrate that the parts of the design covered by the property can be computed with reasonable cost in relation to the time taken by the original Model Checking run. To the best of our knowledge, this is the first interpolant-based algorithm for computing coverage.

The rest of the paper is organized as follows. Section 2 covers preliminaries on Model Checking and Craig interpolation. Section 3 formally defines coverage of circuits given as net-lists. Section 4 presents our algorithm for computing coverage in an interpolating Model Checker. Details about the implementation of our algorithm and the experimental results appear in Section 5.

2. BACKGROUND

We begin with a brief review of finite-state symbolic Model Checking using Craig interpolants computed from resolution proofs.

2.1 Finite-State Model Checking

A transition system $M = (S, T)$ is a finite set of states S and a transition relation $T \subseteq S \times S$. Fix the sets S_0 and F as sets of initial and failure states, respectively. A system is correct if no state in F is reachable from any state in S_0. The image operator $img : \wp(S) \to \wp(S)$ maps a set of states to its successors: $img(Q) = \{s' \in S | s \in Q \text{ and } (s, s') \in T\}$. Let $img^0(Q) = Q$ and $img^{i+1}(Q) = img(img^i(Q))$. A set of states P is *inductive* if $img(P) \subseteq P$. The set P is an *inductive invariant* if P is inductive and $S_0 \subseteq P$. Given S_0 and F, the *strongest inductive invariant* R_{S_0} is the set of states reachable from S_0. An *approximate* image operator $\hat{post} : \wp(S) \to \wp(S)$ satisfies that $img(Q) \subseteq \hat{post}(Q)$ for all $Q \in \wp(S)$. An approximation of the set of reachable states is the set $\hat{R}_{S_0} = \bigcup_{i \geq 0} \hat{post}^i(S_0)$. Observe that if $\hat{R}_{S_0} \cap F = \emptyset$, then F is not reachable from S_0. Thus, it suffices to compute an approximation $\hat{R}_{S_0}$ to decide correctness.

2.2 Interpolant-based Model Checking

Interpolant-based model checking is a method for computing an approximation $\hat{R}_{S_0}$ as above. An approximate operator $\hat{post}$ is implemented using a SAT solver that generates refutations and an interpolation system.

In SAT-based Model Checking, finite sets and relations are encoded in propositional logic. We use sets or relations and their encoding interchangeably. For instance, the propositional encoding of $T \subseteq S \times S$ is written $T(s_0, s_1)$, where s_0 and s_1 are vectors of propositional variables of current and next states, respectively. Consider a set of states Q and a constant $k \geq 1$. A *Bounded Model Checking* (BMC) instance from Q with bound k is a formula $A(s_0, s_1) \wedge B(s_1, \ldots, s_k)$, where A and B are as follows.

$$\begin{array}{lcl} A(s_0, s_1) & \stackrel{\text{def}}{=} & Q(s_0) \wedge T(s_0, s_1) \\ B(s_1, \ldots, s_k) & \stackrel{\text{def}}{=} & T(s_1, s_2) \wedge \ldots \wedge T(s_{k-1}, s_k) \wedge \\ & & (F(s_1) \vee \ldots \vee F(s_k)) \end{array} \quad (1)$$

If the BMC instance is satisfiable, F is reachable from a state in Q. Otherwise, we can obtain a formula that encodes R_{S_0} by iteratively replacing Q by $img(Q) \vee Q$. The computation of the precise image $img(Q)$ is expensive, however. Instead, observe that $img(Q) = \exists s_0.A(s_0, s_1)$. An efficient procedure for computing a formula $I(s_1)$ such that $\exists s_0.A(s_0, s_1) \Rightarrow I(s_1)$ provides an implementation of $\hat{post}$ applicable to compute $\hat{R}_{S_0}$. An interpolation system is such a procedure.

We use a special case of Craig's interpolation theorem [10] for the case of propositional logic. Let $\mathrm{Var}(A)$ be the set of propositional variables occurring in a formula A.

Definition 1. An *interpolant* for a pair of inconsistent propositional formulae (A, B) is a propositional formula I such that 1) $A \Rightarrow I$, 2) I and B are inconsistent, and 3) $\mathrm{Var}(I) \subseteq \mathrm{Var}(A) \cap \mathrm{Var}(B)$.

If the pair $(A(s_0, s_1), B(s_1, \ldots, s_k))$ in Equation 1 is inconsistent, an interpolant $I(s_1)$ is an approximate image. Successive images are computed by replacing Q in A by $I(s_0)$.

2.3 Resolution Refutations

Interpolants are computed from resolution refutations. Let X be a set of propositional variables and $\{x, \neg x \,|\, x \in X\}$ be the set of literals over X, where $\neg t$ is the negation of t. A clause C is a set of literals. The empty clause $\Box$ contains no literals. The disjunction of two clauses C and D is their union, denoted $C \vee D$, which is further simplified to $C \vee t$ if D is the singleton $\{t\}$. A formula in Conjunctive Normal Form (CNF) is a conjunction of clauses, also represented as a set of clauses.

The *resolution principle* states that an assignment satisfying the clauses $C \vee x$ and $D \vee \neg x$ also satisfies $C \vee D$. Let $\mathrm{Res}(C, D, x)$ denote the *resolvent* of the clauses C and D with the pivot x.

Definition 2. A *resolution proof* $\mathcal{P}$ is a DAG $(V_\mathcal{P}, E_\mathcal{P}, piv_\mathcal{P}, \ell_\mathcal{P}, \mathtt{s}_\mathcal{P})$, where $V_\mathcal{P}$ is a set of vertices, $E_\mathcal{P}$ is a set of edges, $piv_\mathcal{P}$ is a pivot function, $\ell_\mathcal{P}$ is the clause function, and $\mathtt{s}_\mathcal{P} \in V_\mathcal{P}$ is the sink vertex. The pivot function maps internal vertices to variables. For an internal vertex v and $(v_1, v), (v_2, v) \in E_\mathcal{P}$, $\ell_\mathcal{P}(v) = \mathrm{Res}(\ell_\mathcal{P}(v_1), \ell_\mathcal{P}(v_2), piv_\mathcal{P}(v))$.

A vertex v_1 in $\mathcal{P}$ is a *parent* of v_2 if $(v_1, v_2) \in E_\mathcal{P}$. A proof $\mathcal{P}$ is a *resolution refutation* if $\ell(\mathtt{s}_\mathcal{P}) = \Box$. An (A, B)-*refutation* $\mathcal{P}$ of an inconsistent CNF pair (A, B) is one in which $\ell_\mathcal{P}(v)$ is an element of A or B for each initial vertex $v \in V_\mathcal{P}$. There are multiple methods to compute interpolants from $\mathcal{P}$, which generate interpolants of different strength [12]. Next, we explain McMillan's system of interpolation [17].

2.4 McMillan's System of Interpolation

Let $\mathcal{P}$ be an (A,B)-refutation of an inconsistent CNF pair (A, B). For all vertices $v \in V_\mathcal{P}$, let $p(v)$ be a formula computed in the following manner: Consider an initial vertex v. If $\ell_\mathcal{P}(v) \in A$, $p(v) = g(v)$ where $g(v)$ is $\vee$ of literals in $\ell_\mathcal{P}(v)$

that appear in B. Otherwise, $p(v) = true$. Consider an internal vertex v_i with parents v_1 and v_2. If $piv_{\mathcal{P}}(v_i) \notin B$, $p(v_i) = p(v_1) \vee p(v_2)$. Otherwise, $p(v_i) = p(v_1) \wedge p(v_2)$. The formula $p(s_{\mathcal{P}})$ is an interpolant of (A, B).

3. FORMALIZING COVERAGE

We restrict the presentation to models that are circuits given as net-lists. We offer a formalization, and then define coverage criteria that are applicable to them.

Definition 3. A net-list N is a directed graph (V, E, τ) where V is a finite set of vertices, $E \subseteq V \times V$ is the set of directed edges and $\tau : V \to \{\text{AND}, \text{INV}, \text{REG}, \text{INPUT}\}$ maps a node to its type, where AND is an AND gate, INV is an inverter, REG is a register, and INPUT is a primary input. The in-degree of vertices of type AND is at least two, of type INV and REG is exactly one and of type INPUT is zero.

A state of a circuit is a mapping of the registers to the Boolean values $\mathbb{B} = \{0, 1\}$. A net-list N with r registers gives rise to a transition system $M = (S, T)$ where $S = \mathbb{B}^r$ is the set of states. The transition relation T is defined in the obvious way.

The mutations we consider depend on the representation of the design [5]. Essentially, a mutation is a modification of the design so that the effect of this change is not masked by any another possible modification.[1] When a design is modeled as a net-list, the smallest possible modification is changing the type of a single node. We consider changing the type of a node to INPUT.[2] This new input can be kept open or fixed to 0 or 1.

Formally, the semantics of a mutant net-list is defined by means of a new labeling function τ'_v, which replaces a node v by a new primary input:

$$\tau'_v(w) := \begin{cases} \tau(w) & : \text{ if } w \neq v \\ \text{INPUT} & : \text{ otherwise} \end{cases}$$

We say that τ'_v *cuts* v from N. If a property satisfied by the original net-list fails on the mutant net-list, we say that the node is NONDET-COVERED. The new input v can also be held to zero or one. These mutations are known as *stuck-at-0* and *stuck-at-1* mutation, respectively. The corresponding coverage notions are called ZERO-COVERAGE and ONE-COVERAGE. Observe that a node that is ZERO-COVERED or ONE-COVERED is also NONDET-COVERED. A node can be NONDET-COVERED, but not be ZERO-COVERED or ONE-COVERED. We do not consider mutations that affect the initial state.

Our coverage metric for a property φ is defined as the fraction of the mutants discovered by checking φ, i.e., φ passes on the original net-list, but fails on the mutant net-list. The coverage of a set of properties can be measured as the fraction of mutants discovered by at least one property. A naive way to compute the coverage is to mutate each node and run a Model Checker on each mutated system. This is necessarily expensive. In the next section, we propose a more efficient algorithm.

[1] As an example, changing the values of two or more signals simultaneously is ill-advised, because the effect of changing the value of the first signal can be masked by changing the value of the second signal.

[2] Changing a node type to other types either may not be useful (REG to INV) or its implications are hard to understand.

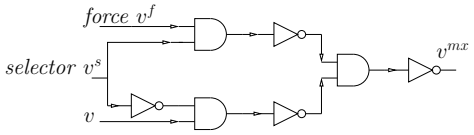

Figure 1: Mutating v using a multiplexer

4. COMPUTING COVERAGE

4.1 Overview

Algorithm 1 takes a net-list $N = (V, E, \tau)$ and a set of failure states F (the property) as input. It computes three maps *nc*, *zc*, and *oc*. These variables track the status of the nodes and map a node v to COVERED, NOTCOVERED, or UNKNOWN. Initially, these variables map all nodes to UNKNOWN.

Algorithm 1 COVERAGECHECKS

Input: Net-list $N = (V, E, \tau)$ and failure states F
Output: *nc*, *zc*, and *oc* (coverage results)

1: **for all** $v \in V$ **do** $nc[v] := zc[v] := oc[v] :=$ UNKNOWN;
2: Check $(S, T^{SEL_o}_{mx}) \models \neg F$ using interpolation
3: Let $\mathcal{P}$ be the final resolution proof, R the inductive invariant, k the bound, m #(approx. image steps)
4: **for all** $v \in V$ **do**
5: **if** $\forall t. v^s_t \notin \ell_{\mathcal{P}}(v_p)$ for any $v_p \in V_{\mathcal{P}}$ **then** ▷ CORE
6: $nc[v] := zc[v] := oc[v] :=$ NOTCOVERED
7: Add $S_0(s_0) \wedge \bigwedge_{i=0}^{e-1} T_{mx}(s_i, s_{i+1}) \wedge \bigvee_{i=0}^{e} F(s_i)$ to solver S, where $e = k + m$
8: **for all** $v \in V$ **do** ▷ COUNTEREXAMPLE
9: **for** each $nc[v]$, $zc[v]$, and $oc[v] =$ UNKNOWN **do**
10: Assume SEL_v and solve S
11: **if** SATISFIABLE **then**
12: Mark $nc[v]$, $oc[v]$ or $zc[v]$ as COVERED.
13: Solve Formula 4 and let $\mathcal{P}_I$ be the resolution proof.
14: Add $\bigwedge_{v \in V_{\mathcal{P}_I}} (\ell_{\mathcal{P}_I}(v_{\mathcal{P}_I}) \mid \ell_{\mathcal{P}_I}(v_{\mathcal{P}_I}) \notin X)$ to solver S_I
15: **for all** $v \in V$ **do** ▷ INDUCTION
16: **for** each $nc[v]$, $zc[v]$, and $oc[v] =$ UNKNOWN **do**
17: Assume SEL_v and solve S_I
18: **if** UNSATISFIABLE **then**
19: Mark $nc[v]$, $oc[v]$ or $zc[v]$ as NOTCOVERED.
20: Add $R(s_0) \wedge T_{mx}(s_0, s_1) \wedge \neg R(s_1)$ to solver S_R
21: **for all** $v \in V$ **do** ▷ INDUCTIONRESTART
22: **for** each $nc[v]$, $zc[v]$, and $oc[v] =$ UNKNOWN **do**
23: Assume SEL_v and solve S_R
24: **if** UNSATISFIABLE **then**
25: Mark $nc[v]$, $oc[v]$ or $zc[v]$ as NOTCOVERED.
26: **for all** $v \in V$ **do** ▷ INTERPOLATION
27: **for** each $nc[v]$, $zc[v]$, and $oc[v] =$ UNKNOWN **do**
28: Run interpolant-based MC on the new design
29: Mark $nc[v]$, $oc[v]$ or $zc[v]$ according to outcome

In order to force a vertex v in N to 0, 1, or to make it nondeterministic without re-building the encoding of the net-list, a multiplexer is introduced in N as shown in Figure 1. We further modify the edges of the net-list such that the tails of all the edges directed from v are changed to v^{mx}. Forcing the selector vertex v^s to 1 cuts a vertex v. In addition, fixing v^f to 0 or 1 causes v^{mx} to be stuck-at-0 or stuck-at-1,

respectively.

We write T_{mx} for the transition relation of this new net-list, and we denote the formula that constrains the selectors by *SEL*. The transition relation and the constraints together are denoted as $T_{mx}^{SEL} = T_{mx} \wedge SEL$. The resulting transition system is (S, T_{mx}^{SEL}). As the first step, the algorithm runs an interpolation-based Model Checker on the model in which all selectors are set to 0 (i.e., the model is equivalent to the original circuit). This selector constraint is denoted by SEL_o. Let v_t^s denote the selector variable of the multiplexer of vertex v at time t. Thus, for a bound k, $SEL_o = \bigwedge_{t=0}^{k} \bigwedge_{v \in V} \neg v_t^s$. The algorithm saves the final proof $\mathcal{P}$ and inductive invariant R produced by the Model Checker. The algorithm proceeds with five tests:

1. CORE: In the first test, nodes not mentioned in the proof are identified, and are declared not covered.
2. COUNTEREXAMPLE: For each of the three types of mutations, a BMC run provides a counterexample which is used to identify some covered nodes.
3. INDUCTION: We check if an inductive argument can be made that a node is not covered.
4. INDUCTIONRESTART: We check if the inductive invariant supplied by the Model Checker allows concluding that a node is not covered.
5. INTERPOLATION: Full interpolant-based Model Checking is applied to all nodes that are still undecided.

The following subsections elaborate on these tests.

4.2 Coverage Analysis with Interpolants

We apply interpolant-based Model Checking to $M_o = (S, T_{mx}^{SEL_o})$. The BMC-style unwinding for this step is shown below:

$$\begin{aligned} BMC_I &\stackrel{\text{def}}{=} Q(s_0) \wedge T_{mx}(s_0, s_1) \\ BMC_T &\stackrel{\text{def}}{=} \textstyle\bigwedge_{i=1}^{i=k-1} T_{mx}(s_i, s_{i+1}) \wedge \bigvee_{i=1}^{i=k} F(s_i) \\ BMC_S &\stackrel{\text{def}}{=} SEL_o \end{aligned} \quad (2)$$

Note that the vector variables s_i include the variables corresponding to the vertices v^f and v^s of the multiplexers. Let $A \stackrel{\text{def}}{=} BMC_I$ and $B \stackrel{\text{def}}{=} BMC_T \wedge BMC_S$. In the beginning, $Q = S_0$. As described in Section 2.2, the interpolant-based Model Checking procedure successively computes approximate images by interpolating the pair (A, B). Assume that it reaches a fixed point after m approximate image steps at bound k. Let R denote the final inductive invariant. At this point, we compute another interpolant L with a different partitioning:

$$(BMC_S,\ BMC_I \wedge BMC_T)$$

The following holds by definition of Craig interpolants:

$$BMC_S \rightarrow L, \quad BMC_I \wedge BMC_T \wedge L \rightarrow \mathit{false} \quad (3)$$

The following theorem states that the interpolant L consists of conjunctions of selector variables present in the proof of unsatisfiability.

Theorem 1. Let $\mathcal{P}$ be the final proof of unsatisfiability of $\overbrace{R(s_0) \wedge T_{mx}(s_0, s_1) \wedge BMC_T}^{B'} \wedge \overbrace{BMC_S}^{A'}$ during Model Checking of M_o. The interpolant L of (A', B') is of the form $\bigwedge \ell_{\mathcal{P}}(v_{\mathcal{P}})$ such that for all initial vertices $v_{\mathcal{P}}$ of $\mathcal{P}$ with $\ell_{\mathcal{P}}(v_{\mathcal{P}}) \in BMC_S$, we have $\ell_{\mathcal{P}}(v_{\mathcal{P}}) \in L$.

PROOF. We provide a proof in McMillan's system of interpolation [17]. The partial interpolant of an initial vertex $v_{\mathcal{P}} \in \mathcal{P}$ and $\ell_{\mathcal{P}}(v_{\mathcal{P}}) \in B'$ is *true*. For an initial vertex $v_{\mathcal{P}}$ with $\ell_{\mathcal{P}}(v_{\mathcal{P}}) \in A'$, $\ell_{\mathcal{P}}(v_{\mathcal{P}})$ is a single literal clause corresponding to some $\neg v_t^s$. This must be resolved with a clause containing v_t^s which is present only in B', since A' does not contain any positive literal v_t^s. It follows that the variables in $\ell_{\mathcal{P}}(v_{\mathcal{P}})$ such that $\ell_{\mathcal{P}}(v_{\mathcal{P}}) \in A'$ are also in B'. Thus, for an initial vertex $v_{\mathcal{P}} \in \mathcal{P}$ and $\ell_{\mathcal{P}}(v_{\mathcal{P}}) \in A'$, the partial interpolant is $\ell_{\mathcal{P}}(v_{\mathcal{P}})$. Also, at any internal node $v_{\mathcal{P}}$ in $\mathcal{P}$, $piv_{\mathcal{P}}(v_{\mathcal{P}})$ belongs to either only B' or both, A' and B'. Thus, the partial interpolants of the parent vertices are always conjoined. □

The following theorem states that selector settings according to L preserve the inductive invariant.

Theorem 2. The inductive invariant R of $M_o = (S, T_{mx}^{SEL_o})$ is also an inductive invariant of $M_L = (S, T_{mx}^{L})$.

PROOF. For M_o, $\overbrace{R(s_0) \wedge T_{mx}(s_0, s_1)}^{A} \wedge \overbrace{BMC_T \wedge BMC_S}^{B}$ is UNSAT. Let $\mathcal{P}$ be the proof of its unsatisfiability. Let I_o denote the interpolant for (A, B), which is the approximate image of R contained in R (the fixed point). Thus, $I_o \subseteq R$.

For M_L, we begin with $Q = R$. From Eq. 3, we conclude that $\overbrace{R(s_0) \wedge T_{mx}(s_0, s_1)}^{AL} \wedge \overbrace{BMC_T \wedge L}^{BL}$ is UNSAT. From Theorem 1, we know that L conjoins only those selector literals from BMC_S that are used in $\mathcal{P}$. The proof $\mathcal{P}$ is also a proof of unsatisfiability of $AL \wedge BL$. Thus, the approximate image of R in M_L, i.e., the interpolant of (AL, BL), is also I_o, which is contained in R. This indicates that a fixed point is reached. □

Lemma 1. Let SEL_v represent the selector settings corresponding to mutating a vertex v of N according to some coverage criterion. If $SEL_v \rightarrow L$, then v is not covered by that mutation.

Thus, we can run interpolation-based Model Checking on $M_o = (S, T_{mx}^{SEL_o})$ and compute the interpolant L from the final proof of unsatisfiability. The selectors of the multiplexers occurring in L when held to 0 guarantee that the resulting system does not violate the property. The following section states that for the purpose of coverage computation, a simple analysis of the proof of unsatisfiability is enough, and that it is not necessary to compute the interpolant.

4.3 The Unsatisfiable Core Test

The following lemma states that absence of some selector variables in the final proof $\mathcal{P}$ during interpolant-based model checking of M_o elucidates which vertices are not covered.

Lemma 2. Consider a vertex v in the net-list N. For all t such that $0 \leq t \leq k$ if the selector variables v_t^s do not appear in $\mathcal{P}$, then v is not NONDET-COVERED.

PROOF. We show that the selector formula corresponding to mutating v into a new primary input implies L. The selector formula $SEL_v = \bigwedge_{t=0}^{t=k} v_t^s \wedge \bigwedge_{t=0}^{t=k} \bigwedge_{w \in V} \{\neg w_t^s | w \neq v\}$ corresponds to this mutation. Using Theorem 1, we conclude $L = \bigwedge_{v_{\mathcal{P}} \in V_{\mathcal{P}}} \{\ell_{\mathcal{P}}(v_{\mathcal{P}}) \mid \ell_{\mathcal{P}}(v_{\mathcal{P}}) \in SEL_o\}$. Each conjunct in SEL_v is a conjunct in L, except v_t^s as $\mathcal{P}$ does not contain v_t^s in any time frame. Thus, $SEL_v \rightarrow L$. □

Thus, a proof-logging SAT solver is not required; a solver able to identify the clauses that result in a proof is sufficient. Note that even if v is not NONDET-COVERED, it is possible that v_t^s appears in the proof $\mathcal{P}$ under consideration.

4.4 The Counterexample Test

A counterexample, i.e., a path from S_0 to F indicates at least one covered mutation. It is essential to identify such mutations early before expensive Model Checking runs are performed. Counterexamples of a given length can be obtained at moderate cost using BMC.

Since $R \supseteq \bigcup_{i=0}^{m} img^i(S_0)$, there is a path of length $\leq m$ from S_0 to R in the original system. As R does not reach F in k steps, all paths of length $\leq (k+m)$ from S_0 do not reach F. We check if a counterexample of length $\leq (k+m)$ exists in $(S, T_{mx}^{SEL_v})$. Note that we only mutate one vertex at a time. For each trace obtained from the SAT solver, the mutations found covered are recorded.

4.5 Two Induction-based Tests

We exploit the inductive invariant R for M_o to analyze the remaining mutations at low cost. We check the following formula for satisfiability:

$$\overbrace{R(s_0) \wedge T_{mx}(s_0, s_1) \wedge \neg R(s_1)}^{Y} \wedge \overbrace{\textstyle\bigwedge_{v \in V} (\neg v_0^s \wedge \neg v_1^s)}^{X} \qquad (4)$$

Let $\mathcal{P}_I$ be the proof of unsatisfiability of Formula 4.

Lemma 3. Let SEL_v be the selector settings for some mutation of v. Let $Z = \bigwedge_{v \in V_{\mathcal{P}_I}} \{\ell_{\mathcal{P}_I}(v) \mid \ell_{\mathcal{P}_I}(v) \notin X\}$. If $SEL_v \wedge Z$ is unsatisfiable, the mutation of v is not covered.

PROOF. Note that $\ell_{\mathcal{P}_I}(v) \notin X$ iff $\ell_{\mathcal{P}_I}(v) \in Y$. It holds that $Y \rightarrow Z$. Thus, if Z and SEL_v are inconsistent, Y and SEL_v are inconsistent. Also, $S_0 \subseteq R$, as our mutation does not affect the initial states. Hence, R is an inductive invariant of $(S, T_{mx}^{SEL_v})$. □

If the above check fails, it may be that the remaining clauses from Y are needed to prove inductiveness. This can be done by checking $Y \wedge SEL_v$ using the following SAT instance:

$$R(s_0) \wedge T_{mx}(s_0, s_1) \wedge SEL_v \wedge \neg R(s_1) \qquad (5)$$

Lemma 4. Let SEL_v be the selector setting for some mutation of v. If Formula 5 is unsatisfiable, the mutation is not covered.

PROOF. Recall $S_0 \subseteq R$. Since Formula 5 is unsatisfiable, the image of R for the mutated transition relation is contained in R, which is strong enough to prove the property. □

If Formula 5 is satisfiable, there may still exist a different inductive invariant for the mutated system. In this case, we fall back to interpolant-based Model Checking.

5. EXPERIMENTS

We have implemented the algorithm described in the previous section. Our tool accepts Verilog as well SMV files as input. The properties are given as assertions. We use the circuits from the Hardware Model Checking Competition 2008 as benchmarks. Each of these benchmarks is shipped with one property. We ran our tool on 161 benchmarks that our interpolant-based model checker is able to complete within a timeout of 1800 seconds. The average number of registers/latches in these benchmarks is about 110 with a median 75, and maximum number of latches of 567. About 25% of the benchmarks have a high-coverage property (an average of 40% of the latches are NONDET-COVERED). The remainder of the benchmarks have low-coverage properties (an average of 5% of the latches are NONDET-COVERED).

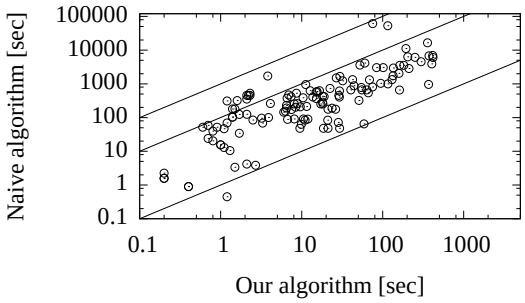

Figure 2: Comparison of naive vs. COVERAGECHECKS

We restrict the coverage analysis to the registers. For each register in the design, we check whether it is ZERO-COVERED, ONE-COVERED, or NONDET-COVERED. Figure 2 is a log-scale scatter plot comparing the time taken by our algorithm with the time it would take to run Model Checking on each of the mutated designs.[3] The speedup obtained using our method is several orders of magnitude.

We quantify the success rates of the five methods presented in Section 4. We call a test successful if it is able to classify a given mutation as covered or not covered. The histogram in Figure 3 depicts the relative success rates of each of our five tests for computing coverage. The coverage of the vast majority of mutations is decided by the CORE and the COUNTEREXAMPLE (CE) tests. Our method is indeed able to avoid most of the expensive calls to the interpolating Model Checker: in 50% of the benchmarks, we call the Model Checker in only 10% of the cases.

For benchmarks with a low-coverage property, most of the selector constraints are not in the core. Thus, the CORE test is frequently successful on this category of benchmarks. On benchmarks with a high-coverage property, a counterexample for one mutant is often applicable to many others, which means that the COUNTEREXAMPLE test is successful. The remaining tests contribute roughly equal parts.

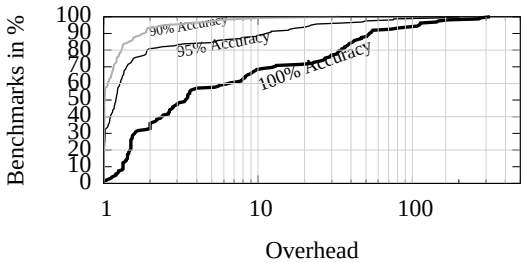

Figure 4: Overhead for a given accuracy

In most cases, an approximate coverage metric is sufficient. We observe that our algorithm is very well-suited for

[3]This time is estimated based on the time spent on Model Checking some mutated systems, as the full check is prohibitively expensive.

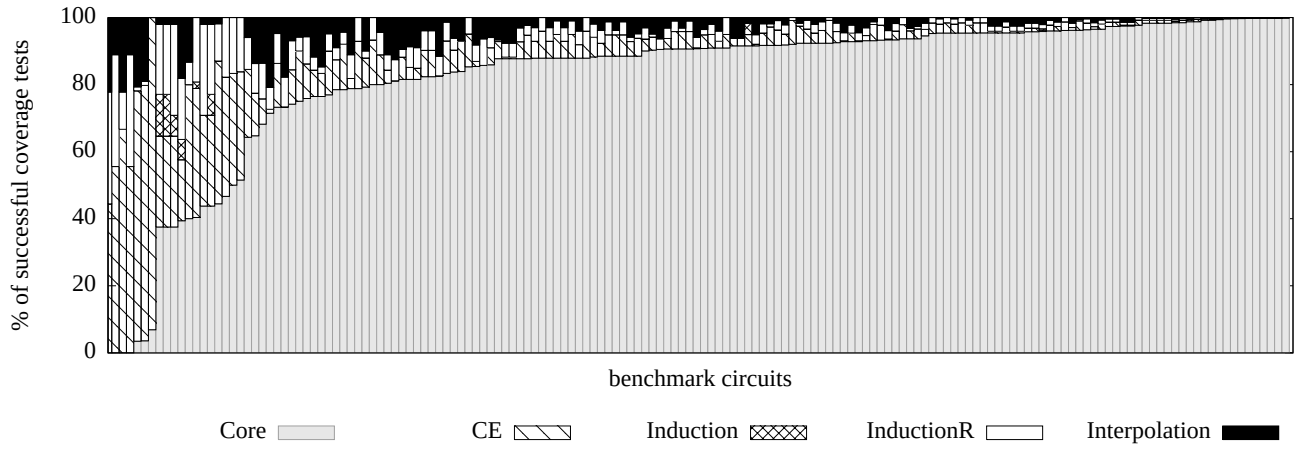

Figure 3: Histogram of the success rates of the proposed tests

obtaining a quick approximation. Figure 4 reports the time overhead in relation to the original Model Checking run for a given accuracy, averaged over the benchmark set. We note that 90% accuracy can be obtained for 90% of the benchmarks analyzed with an overhead of only 2.

6. CONCLUSION

Low coverage indicates a possible incompleteness in the specification, which may lead to missed bugs in the non-covered parts of the design. We present an algorithm for computing mutation coverage of designs represented as netlists. Our algorithm is integrated and implemented in a Craig interpolant-based Model Checker. The main advantage of our algorithm is that it exploits the information in the final resolution proof, and thus the Model Checker needs to be executed only on a very small percentage of mutations. Our tool also allows to trade precision for speed. We show that a very accurate coverage measure can be computed with little overhead compared to the time taken by the model checking run on the original design. Our technique provides a speedup of several orders of magnitude compared to the naive method, and it is well-suited for computing very inexpensive approximations.

7. REFERENCES

[1] L. Bening and H. Foster. *Principles of verifiable RTL design – a functional coding style supporting verification processes*. Kluwer, 2000.

[2] H. Chockler and O. Kupferman. Coverage of implementations by simulating specifications. In *Conference on Theoretical Computer Science*, IFIP Conference Proceedings 223, pages 409–421. Kluwer, 2002.

[3] H. Chockler, O. Kupferman, R. Kurshan, and M. Vardi. A practical approach to coverage in model checking. In *CAV*, LNCS 2102, pages 66–78. Springer, 2001.

[4] H. Chockler, O. Kupferman, and M. Vardi. Coverage metrics for temporal logic model checking. In *TACAS*, LNCS 2031, pages 528 – 542. Springer, 2001.

[5] H. Chockler, O. Kupferman, and M. Vardi. Coverage metrics for formal verification. In *CHARME*, LNCS 2860, pages 111–125. Springer, 2003.

[6] K. Claessen. A coverage analysis for safety property lists. In *FMCAD*, pages 139–145. IEEE, 2007.

[7] E. Clarke and E. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, LNCS 131, pages 52–71. Springer, 1981.

[8] E. Clarke, O. Grumberg, K. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *DAC*, pages 427–432. IEEE, 1995.

[9] E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.

[10] W. Craig. Linear reasoning. A new form of the Herbrand-Gentzen theorem. *J. Symb. Log.*, 22(3):250–268, 1957.

[11] S. Das, A. Banerjee, P. Basu, P. Dasgupta, P. P. Chakrabarti, C. R. Mohan, and L. Fix. Formal methods for analyzing the completeness of an assertion suite against a high-level fault model. In *International Conference on VLSI Design*, pages 201–206, 2005.

[12] V. D'Silva, M. Purandare, G. Weissenbacher, and D. Kroening. Interpolant strength. In *VMCAI*, volume 5944 of *LNCS*, pages 129–145. Springer, 2010.

[13] D. Große, U. Kühne, and R. Drechsler. Estimating functional coverage in bounded model checking. In *DATE*, pages 1176–1181. ACM, 2007.

[14] Y. V. Hoskote, T. Kam, P.-H. Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *DAC*, pages 300–305. ACM, 1999.

[15] O. Kupferman. Sanity checks in formal verification. In *Concurrency Theory (CONCUR)*, LNCS 4137, pages 37–51. Springer, 2006.

[16] O. Kupferman, W. Li, and S. A. Seshia. A theory of mutations with applications to vacuity, coverage, and fault tolerance. In *FMCAD*, pages 1–9. IEEE, 2008.

[17] K. L. McMillan. Interpolation and SAT-based model checking. In *CAV*, LNCS 2725, pages 1–13. Springer, 2003.

[18] S. Tasiran and K. Keutzer. Coverage metrics for functional validation of hardware designs. *IEEE Design and Test of Computers*, 18(4):36–45, 2001.