Coverage Estimation for Symbolic Model Checking

Yatin Hoskote*, Timothy Kam*, Pei-Hsin Ho**, Xudong Zhao*

*Strategic CAD Labs
Design Technology, Intel Corp.
{yatin.hoskote, timothy.kam, xudong.zhao}@intel.com

**Advanced Technology Group
Synopsys, Inc.
pho@synopsys.com

Abstract

Although model checking is an exhaustive formal verification method, a bug can still escape detection if the erroneous behavior does not violate any verified property. We propose a coverage metric to estimate the "completeness" of a set of properties verified by model checking. A symbolic algorithm is presented to compute this metric for a subset of the CTL property specification language. It has the same order of computational complexity as a model checking algorithm. Our coverage estimator has been applied in the course of some real-world model checking projects. We uncovered several coverage holes including one that eventually led to the discovery of a bug that escaped the initial model checking effort.

1 Introduction

Model checking is the most popular formal verification (FV) technology for property verification today in an industrial setting. Given a model of a design and some desired properties, a model checker like SMV[1] exhaustively verifies whether the model satisfies all the desired properties under all possible input sequences. The properties are specified in a property specification language such as Computation Tree Logic (CTL) [2]. Although model checking is an exhaustive FV technique, a bug can escape the model checking effort if the properties specified by the user do not check for the erroneous behavior caused by the bug. Such erroneous behavior usually occurs in some obscure corner case that has been missed by the user. This is quite common when the specification has to be manually decomposed into a set of smaller, more tractable properties that are verifiable by the model checker. To reduce bug escapes, the user needs to continuously strengthen existing properties and specify new properties, without knowing if the additional verification is insufficient or redundant.

In existing simulation-based verification methodologies, coverage metrics are used to improve the quality of the test suite and estimate the progress of the verification task. For example, a common coverage metric for simulation is *code coverage* [3], which measures the fraction of HDL statements exercised during simulation. *Transition coverage* is another metric for control state machines [4][5]. Such coverage metrics have been proven to be effective in reducing bug escapes by pointing out coverage holes in the test suite [6]. For the same purpose, a coverage metric that can identify coverage holes in the formally verified properties can certainly facilitate the process of augmenting the properties.

**This work was done when the author was with Intel Corp.

To the best of our knowledge, such coverage metrics for model checking do not exist. The need for a coverage metric may not be apparent until now because model checking is still in its infancy in industrial usage. The other reason is that existing coverage metrics for simulation do not apply directly to model checking e.g., a naive interpretation of the code coverage or transition coverage metric on a model checking task gives a meaningless coverage of 100% for every property. Logic simulation is dynamic and its coverage is driven by input simulation vectors, whereas the model checking engine is static without any notion of circuit execution. Unlike logic simulation, the likelihood of having a bug escape detection in a model checking effort depends solely on the quality of the properties verified. Therefore, we want a coverage metric that estimates the "completeness" of a set of properties against which the design has been verified.

Consider the CTL formula for *count*, a modulo-5 counter, with *stall* and *reset* as external inputs:

$$AG[((\neg stall \wedge \neg reset \wedge (count = C) \wedge (C < 5)) \rightarrow AX(count = C + 1)]$$

This formula specifies that if the *stall* and *reset* signals are deasserted and the counter value is less than 5, then the counter increments by 1 in the next step. The model checker explores the entire reachable state space to verify the property. However, in reality, it ascertains the correctness of the condition on *count* (that it increments correctly) only in those states that are immediate successors of states satisfying the antecedent. The actual checking of the correctness condition on the model state space is thus constrained by the CTL formula. Clearly, this property cannot be said to provide 100% coverage. This example illustrates that there is indeed value to defining a coverage measure for formally verified properties.

We define a coverage metric to identify that part of the state space which is covered by the verified properties. In each property, we identify one signal (or a proposition on several signals) as the *observed signal* in that property. Our metric measures the coverage of a set of properties with respect to this observed signal. For the above example, we consider *count* to be the observed signal. Informally, the coverage metric identifies the reachable states in which the value of the observed signal determines the validity of the verified properties. The model checking algorithm only checks the correctness condition on the observed signal *count* in these "covered" states to prove or disprove the property.

We also present a coverage estimation algorithm for a subset of CTL. It is of the same order of complexity as a model checking algorithm. Thus the coverage can be computed if the property can be verified. The coverage estimator has been implemented and applied in the course of some real-world model checking projects. We uncovered several coverage holes including one which eventually led to the discovery of a bug that once escaped the model checking effort. These results support our belief that this coverage metric should be useful in industrial model checking efforts.

This paper is organized as follows. Section 2 defines the coverage metric. The algorithm for computing the metric and its correctness proof outline are presented in Section 3. Methodology for usage is presented in Section 4 and experimental results are presented in Section 5. We discuss limitations of this metric in Section 6 and then conclude the paper with some observations.

2 Coverage in Formal Verification

A formula (or property) specifies the desired values of particular circuit signals at various points in time in relation to other signals. In other words, each formula specifies a correctness condition on certain circuit signals and also specifies where in the circuit state space this condition should hold. One of the signals or propositions being checked in the correctness condition is identified as the observed signal and coverage is defined on this observed signal. In this paper, we view a sequential circuit as a Mealy machine.

Definition 1: A finite state machine (FSM) M is a 4-tuple $< S, T_M, P, S_I >$, where S is a finite set of states, $T_M \subseteq S \times S$ is a transition relation of M between pairs of states in S, and $P = \{p_1, p_2, .., p_n, q\}$ is a set of signals, where q is the observed signal. Each signal is a Boolean function $S \to \{T, F\}$ representing a set of states, or equivalently, each signal corresponds to an atomic proposition. $S_I \subseteq S$ is a set of initial states.

Given a property that has been verified to be true of the circuit, we define coverage of that property for the specified observed signal in terms of a subset of circuit states reachable from the initial states. A state is reachable from the initial states if there exists an input sequence which takes the FSM from an initial state to that state. A *covered set* of states for an observed signal is the set of reachable states in which the values of the observed signal must be checked to prove satisfaction of the property.

This leads to two desirable characteristics of the covered set. First, if we change the value of the observed signal in any state of the model outside the covered set, the property should still be satisfied. Those states are not checked for the property. Second, if we change the value of the observed signal in any covered state, the property should fail. The set of covered states is a minimal set.

To determine whether a state belongs in the covered set, we modify the value of the observed signal in that state and assess its effect on the validity of the property. To facilitate this test, we first define a *dual FSM* for each state of the given FSM M.

Definition 2: Given an FSM $M = < S, T_M, P, S_I >$, where $P = \{p_1, p_2, .., p_n, q\}$, and any state $s \in S$, the *dual FSM* $\hat{M}_s$ with respect to state s is the 4-tuple $< S, T_M, \{p_1, p_2, .., .p_n, \hat{q}_s\}, S_I >$, where

$$\hat{q}_s(t) = \begin{cases} q(t) | (t \neq s) \\ \neg q(t) | (t = s) \end{cases} \text{ for every state } t \in S.$$

With this definition, we now define a covered set of states.

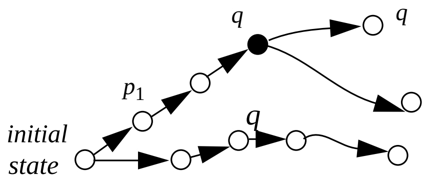

Figure 1: Covered state for AG(p_1 -> AX AX q)

Definition 3: Given a property f and an FSM M such that M satisfies f with respect to its initial state set S_I, denoted by $M, S_I \models f$, a set $C \subseteq S$ is a covered set of f on M for observed signal q if and only if for any state $s \in S$, the dual FSM satisfies the condition $(\hat{M}_s, S_I \not\models f) \Leftrightarrow (s \in C)$.

This definition is independent of the property specification language and guarantees that changing the value of the observed signal in an uncovered state will not cause the property to fail while changing the value in a covered state will cause it to fail. This set of covered states is unique. Consequently it constitutes a necessary and sufficient set to prove satisfaction of the property. Covered states are defined in this manner so that the value of the observed signal on those states is guaranteed to satisfy the correctness condition as specified by the property. Thus, coverage gives an intuitive measure of how much of the state space of the model has been checked by the verified property for the observed signal. This definition does not preclude multiple observable signals in the same property. The covered states are then simply the union of the covered states for each individual signal.

To prove by contradiction that the set of covered states is unique, assume that there are two distinct covered sets C_1 and C_2 for a property f and an observed signal q. As $C_1 \neq C_2$, there must exist a state s which belongs to one but not the other, say $s \in C_1$ and $s \notin C_2$ without loss of generality. By the definition of covered set C_1, if we change the value of q in state s, the property f fails. This implies that s should belong to all covered sets and therefore $s \in C_2$, which is a contradiction. Therefore, $C_1 = C_2$ and the set of covered states is unique.

We show a simple example to illustrate the intuition behind the definition of a covered state. Suppose that we wish to compute coverage of the simple CTL formula

$$AG(p_1 \to AX \ \ AX \ \ q)$$

with q as the observed signal. The formula specifies that whenever p_1 is asserted, q will be asserted two steps in the future. Figure 1 shows a fragment of the state transition graph of a circuit on which we are computing coverage. Consequently, q must be asserted in the marked state in Figure 1 for the formula to hold. This marked state is a covered state. Inspection shows that the condition specified in Definition 3 indeed holds for this state. Note that there are other states with q asserted but are not marked as covered since they are not critical to the validity of the given formula.

Definition 4: Coverage of a formula for an *observed signal* on a given model with a given set of initial states is computed as the fraction of reachable states of the model that are covered:

$$coverage = \frac{number\ of\ covered\ states}{number\ of\ reachable\ states} \times 100\ \%$$

Coverage for a set of properties is simply the coverage from the union of the covered sets from each individual property.

Full or 100% coverage for a particular observed signal thus means that the value of that signal has been checked by the verified properties on all reachable states of the circuit. This serves as a very useful indicator of the completeness of the properties and the quality of the verification. More importantly, the formulation of the coverage metric allows the identification of areas with low coverage in terms of uncovered states so that the user can write additional properties to increase the coverage.

2.1 Coverage for ACTL formulas

The definition of coverage presented above is general enough to be applicable to any property specification language. However, the covered set may not be easily computable for all languages. In this paper, we consider a subset of ACTL [2], the universal subset of CTL, and present an algorithm to compute the covered set for this subset. In our experience, this subset is sufficiently expressive to specify most desirable properties of sequential logic circuits in practice.

The subset of ACTL acceptable to us is defined as follows:

$f ::= b \mid b \rightarrow f \mid AXf \mid AGf \mid A[f \cup g] \mid f \wedge g$

where *b* is a propositional formula and *f* and *g* are temporal formulas within the subset. Note that *AF f* can be equivalently written as *A*[*True U f*] and we do not need to treat it separately. The only ACTL construct missing from this subset is disjunction of temporal formulas.

Applying Definition 3 to this subset of ACTL, we can compute exactly the set of states where the value of the observed signal is crucial to the validity of the formula. However, such a faithful application results in some unexpected coverage. The coverage for *eventuality* properties is extremely low. For instance, consider the property $A[p_1\ U\ q]$ and the state transition graph of a circuit as shown in Figure 2. The property specifies that p_1 should be high on any path from an initial state until observed signal *q* is asserted. Intuitively, we would expect that the first state encountered where *q* is asserted should be covered (as marked in Figure 2). However, changing the value of *q* in this state does not cause the property to fail because p_1 is high in that state. In fact, none of the states on this path will be considered covered by the definition. Thus the coverage for this property will be zero. This is contrary to our expectation from such a property. To obtain a more intuitive measure of coverage, we need to isolate the coverage effects of the two parts of the Until formula from each other and compute coverage for each part separately. To achieve this, we define a transformation on ACTL formulas that changes the syntactic structure of the formulas but maintains semantic equivalence.

Definition 5: For an FSM *M*, given a formula *f* in the acceptable ACTL subset and an observed signal *q* within the formula, we introduce a signal *q*' defined by the same function as the observed signal *q*. The *observability transformation*, φ, is defined as follows: we substitute occurrences of *q* in f with *q*' (denoted by $f|_{q \rightarrow q'}$)

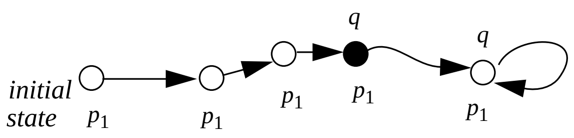

Figure 2: Computing covered states for $A[p_1\ U\ q]$

$$\varphi(b \rightarrow f) = b \rightarrow \varphi(f)$$

$$\varphi(b) = b|_{q \rightarrow q'}$$

$$\varphi(AXf) = AX\varphi(f)$$

$$\varphi(AGf) = AG\varphi(f)$$

$$\varphi(A[fUg]) = A[\varphi(f)Ug] \wedge A[(f \wedge \neg g)U\varphi(g)]$$

$$\varphi(f \wedge g) = \varphi(f) \wedge \varphi(g)$$

In the sequel, we shall write f' and g' as the shorthand for $\varphi(f)$ and $\varphi(g)$ respectively. The new signal q' is now the observed signal for the transformed formula f'.

Note that the formulas after the observability transformation are equivalent to the original formulas with respect to validity of the verification. The only two cases in which we change the syntactic structure of the formula are the implication and Until formulas. The motivation is to pinpoint the states which contribute coverage from the consequent part of the implication as well as the states which independently contribute coverage from each part of the Until formula. As a result of the observability transformation, two semantically equivalent formulas with different syntax can provide different coverage. This is acceptable because we believe the syntax of the formula better captures the verification intent of the user. The application of Definition 3 to the transformed formulas gives a more intuitive and pragmatic determination of the covered set.

3 Coverage Computation

In this section, we present a recursive algorithm to compute the set of covered states in the state space of an FSM for a given ACTL formula and a given observed signal. The algorithm operates on the original formula but gives the computed set of covered states with respect to the transformed formula. Thus, our computation of coverage does not require application of the observability transformation. Later, we shall prove that the computed covered set is the same set of states as would have been obtained by direct application of Definition 3 to the transformed formula.

Problem Statement: Given an FSM *M* with a set of initial states S_I and an acceptable ACTL formula *g* such that $M, S_I \models g$, compute the set of covered states and the coverage for observed signal *q*.

Coverage for a nested formula *g* is computed in a recursive manner on the syntactic structure of *g.* This algorithm is summarized byTable 1. Coverage for each sub-formula is computed with respect to a set of *start states*. The covered state set for formula *g* with respect to start state set S_0 is denoted by $C(S_0, g)$. As we traverse down the parse tree, the set of *start states* used to compute coverage for a particular sub-formula changes, as shown in the table. Coverage for the top-level formula g is computed with respect to the set of initial states S_I of M (substituting $S_0 = S_I$ in Table 1), i.e., $C(S_I, g)$.

The algorithm guarantees that the value of the observed signal in any covered state satisfies the correctness condition specified by the formula. If a sub-formula does not involve the observed signal, its covered set will be empty. Definitions of the functions used in the algorithm are given below.

TABLE 1. Recursive computation of covered set

$C(S_0, g)$	**Covered Set of States**
$C(S_0, b)$	$S_0 \cap depend(b)$
$C(S_0, b \to f)$	$C(S_0 \cap T(b), f)$
$C(S_0, AXf)$	$C(forward(S_0), f)$
$C(S_0, AGf)$	$C(reachable(S_0), f)$
$C(S_0, A[f_1 U f_2])$	$C(traverse(S_0, f_1, f_2), f_1) \cup$
	$C(firstreached(S_0, f_2), f_2)$
$C(S_0, f_1 \wedge f_2)$	$C(S_0, f_1) \cup C(S_0, f_2)$

Given a propositional formula b, let $T(b)$ represent the set of states which satisfy b. Note that the property is satisfied by the circuit if and only if b is true in all start states. The subset of these start states which are covered is identified as those start states where the satisfaction of predicate b actually depends on the value of observed signal q (b may also specify conditions on other signals). In other words, changing the value of observed signal q on a covered state must falsify the formula b on that state when otherwise it would be true. In the above table, this set of states is given by the function *depend(b)*.

$$depend(b) = T(b) \cap T(\neg b|_{q \to \neg q})$$

The computation for formulas of type $b \to f$, $AX\,f$, and $AG\,f$ is straightforward. The formula $b \to f$ specifies that the formula f must be true on those start states in S_0 which satisfy the predicate b. The covered set $C(S_0, b \to f)$ of the sub-formula $b \to f$ with respect to the start states S_0 is equivalent to, and computed as, covered set $C(S_0 \cap T(b), f)$ for f with respect to the new set of start states $S_0 \cap T(b)$.

The formula $AX\,f$ specifies that f holds in all successor states from the start state set S_0. The function *forward(S_0)* gives states reachable in exactly one step from the start states in S_0. This set becomes the new start states while computing coverage for f.

$$forward(S_0) = \{s'' | \exists s' \in S_0, (s', s'') \in T_M\}$$

$AG\,f$ specifies that f holds on all states reachable from S_0. The function *reachable(S_0)* gives states reachable from S_0 in any number of forward steps. This set becomes the new set of start states for computing coverage of f.

$$reachable(S_0) = \bigcup_{i=0}^{\infty} forward^i(S_0)$$

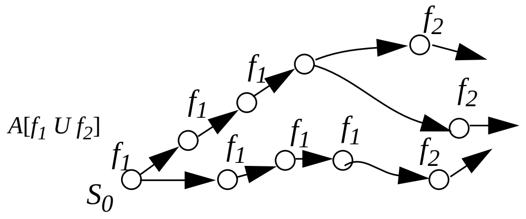

Figure 3: Computing covered states for A[f_1 U f_2]

The coverage estimation for the Until operator is a little more complicated. The computation of covered states for $A[f_1 U f_2]$ is explained with the help of the state transition graph in Figure 3. Sub-formula f_1 is verified to be true on states along paths from a start state (unique in this example) in S_0, such that f_1 is true until f_2 first becomes true. The function *traverse(S_0, f_1, f_2)* identifies states along paths starting from states in S_0 such that f_1 is true and f_2 is not true until, but not including, states where f_2 becomes true. These states are marked and labelled by f_1 in Figure 3 and become the new set of start states while computing coverage for sub-formula f_1.

$$traverse(S_0, f_1, f_2) = S'_0 \cup traverse(forward(S'_0), f_1, f_2)$$

where $S'_0 = S_0 \cap T(f_1) \cap T(\neg f_2)$.

In addition, the states satisfying f_2 first encountered while traversing forward from S_0 are considered as start states for computing coverage for propositional sub-formula f_2. These are marked and labelled by f_2 in Figure 3, and are computed by the function *firstreached(S_0, f_2)*.

$$firstreached(S_0, f_2) = (S_0 \cap T(f_2)) \cup firstreached(forward(S_0 \cap T(\neg f_2)), f_2)$$

The covered set of the Until formula is the union of the coverage from *C(traverse(S_0, f_1, f_2), f_1)* and *C(firstreached(S_0, f_2), f_2)*.

The covered set of a formula which is a conjunction of two sub-formulas is simply the union of the covered sets of the sub-formulas, because both sub-formulas must be satisfied by the FSM for the conjuncted formula to be satisfied.

Correctness Theorem: Given an FSM M with initial states S_I and an acceptable ACTL formula g with observed signal q, the above algorithm computes correctly the set of covered states as specified by Definition 3 for the transformed formula $g' = \varphi(g)$ and observed signal q', where φ is the observability transformation.

Proof: Due to length limitation on the paper, we only present a proof skeleton here. Interested readers are welcome to contact the authors for a complete proof.

The proof is by induction on the structure of the formula. Except for the three cases involving temporal operators, all other cases can be easily obtained from the definitions of coverage, the observability transformation and the algorithm. The cases of AX and AG operators follow directly from the following equations:

$$(M, forward(S_0) \models f) \Leftrightarrow (M, S_0 \models (AXf))$$

$$(M, reachable(S_0) \models f) \Leftrightarrow (M, S_0 \models (AGf))$$

The most complex case is the Until operator. The two terms in the definition of the observability transformation correspond to the two sets in the coverage algorithm. Since *traverse*(S_0, f_1, f_2) represents the set of states on paths starting from S_0 that satisfy f_1 and do not satisfy f_2 until and not including states that satisfy f_2, the covered set computed by *C*(*traverse*(S_0, f_1, f_2), f_1) is the correct covered set for $A[f'_1 \mathrm{U} f_2]$. Likewise, the covered set computed by *C*(*firstreached*(S_0, f_2), f_2) is the correct covered set for $A[(f_1 \wedge \neg f_2) \mathrm{U} f'_2]$. □

This algorithm is of the same order of complexity as conventional symbolic model checking algorithms. Both are based on fix point computation using Binary Decision Diagrams (BDDs) [7] which are exponential in the worst case. Results for sub-formulas computed during verification can be memoized and used during coverage estimation for a more efficient implementation. In practice, coverage estimation can be slightly more expensive than the verification in some cases because it requires computing the coverage space as the set of reachable states. This involves fix point computation which may not have been necessary for the actual verification of the CTL formulas. Advanced techniques for reachability analysis can be of great help here.

After computing the set of reachable states and the set of covered states, the coverage estimator gives the coverage percentage and prints out a list of uncovered states. This output can greatly aid the user in writing additional properties to cover the holes. The coverage estimator also prints out traces to uncovered states by performing a breadth first reachability analysis from the initial states to an uncovered state via the shortest path and generating an input sequence corresponding to this path [8].

4 Coverage in the Verification Flow

4.1 Methodology

As motivated earlier, a coverage metric can be very useful in achieving a high degree of confidence in the completeness of the verification. Using the metric presented here, the verification engineer is able to identify behaviors exhibited by the circuit that have not been checked by any property. The first step in this process is to inspect uncovered states provided by the coverage estimator. If it is not immediately apparent from this inspection how to strengthen the verification to cover that hole, the second step is to instruct the tool to generate traces to specific uncovered states. These traces are evidence of circuit behavior leading to uncovered states and provide strong hints as to the nature of additional properties required to achieve higher coverage. The user can then strengthen the verification either by writing additional properties or improving existing ones by weakening the antecedent or strengthening the consequent. The minimum coverage requirement recommended by us is to ensure 100% coverage for each primary output signal.

4.2 Don't cares

A large fraction of the set of states not covered by the properties could be states on which the value of the observed signal is irrelevant to the correctness of the circuit. These *don't care* states are supplied *a priori* by the user as a set of propositions on state variables and excluded from the coverage space so as to give a more realistic coverage estimate.

4.3 Fairness conditions

Fairness conditions expressed in the model checking system constrain the system to only look at fair paths during the verification of a property, i.e., paths where the fairness constraints are true infinitely often. The presence of fairness constraints therefore requires the coverage estimation algorithm also to ignore states not falling on fair paths. Coverage is computed as the fraction of states reachable along fair paths.

5 Experimental Results

The coverage estimator has been implemented on top of SMV[1] and applied to several circuits from a microprocessor design. We selected a few signals from each circuit as the observed signals and applied the estimator to determine the coverage of properties which had been verified to check behavior of those signals. Table 2 gives the names of the observed signals for which coverage was measured, the number of properties verified for that signal, the coverage obtained for the given set of properties, the performance of model checking measured in terms of the number of BDD nodes and the run time in seconds on a HP9000 workstation, and the runtime performance of the coverage estimator.

TABLE 2. Coverage results

Signal	# Prop	%COV	Verification BDDs - time	Coverage BDD - time
Circuit 1 (priority buffer)				
hi-pri	5	100.00	124k - 59.28s	150k - 60.41s
lo-pri	5	99.98	155k - 61.37s	178k - 71.26s
Circuit 2 (circular queue)				
wrap	5	60.08	26k - 8.3s	26k - 7.46s
full	2	100.00	21k - 1.55s	21k - 1.52s
empty	2	100.00	13k - 1.51s	13k - 1.55s
Circuit 3 (pipeline)				
output	8	74.36	10k - 3.58s	10k - 7.42s

Circuit 1 is a priority buffer which schedules and stores incoming entries according to their priorities (high or low). The model had 24 variables. Given the number of entries already in the buffer and the number of incoming entries, the properties specify the correct number of entries in the buffer at the next clock. For example, if the buffer currently has B entries and I incoming entries and I + B is less than the size of buffer, then the buffer in the next clock should have I + B entries. High and low priority entries are checked by different properties, and their counts are considered as the observed signals. The set of verified properties should provide a complete analysis of all possible cases, but we uncovered a missing case when the buffer is empty and low priority entries are incoming, the entries should be stored. A simple additional property was written to cover this case. Verification of this property failed and actually revealed a bug in the design of the buffer!

Circuit 2 is a circular queue controlled by a read pointer, a write pointer and a wrap bit that toggles whenever either pointer wraps around the queue. It also has stall, clear and reset signals as inputs. Properties were written to verify the correct operation of the wrap bit, the full and empty signals. The model had 38 variables. The coverage for the full and empty signals was 100%. But coverage for the wrap bit was 60%. Inspecting the uncovered states, three additional properties were written which still did not achieve 100% coverage. We traced the input/state sequences leading to these uncovered states and found that the value of wrap bit was not checked if the stall signal was asserted when the write pointer wraps around. Such a subtle corner case can easily be missed during property specification. A property was added to specify that the wrap bit remains unchanged for this case and 100% coverage was achieved.

Circuit 3 is a pipeline in the instruction decode stage of the processor. The width of the pipeline datapath was abstracted to a single bit. Properties were verified on this signal to check the correct staging of data through the pipeline [9], rather than the actual data transformations. These properties generally took the form that an input to the pipeline will eventually appear at the output given certain fairness conditions on the stalls. The final model had 15 variables. Coverage was increased to 100% by identifying uncovered states and enhancing the set of properties. The biggest hole in our pipeline control verification was that we ignored the fact that the pipeline output retains its value for 3 cycles while data is being processed by a state machine connected to the end of the pipeline.

These examples demonstrate that coverage estimation can improve the quality of FV. The runtimes and memory requirements are similar to those required by the actual verification. Furthermore, the examples are a good representation of common FV properties: the buffers involved syntactically simple properties, e.g., $AG(p_1 \rightarrow AX \ldots AX p_2)$ and the pipeline required eventuality properties using the Until operator in a nested manner, e.g., $AG(p_1 \rightarrow A[p_2 U A[p_3 U p_4]])$.

6 Limitations

The coverage metric, though very effective in uncovering missing properties, has limitations. First, it is a metric based on the circuit model itself. It can uncover functionality in the model not verified by any property, but it cannot point out functionality missing in the model (and the properties). Thus, an incomplete design can have 100% coverage. In fact, all model based coverage metrics share the same drawback. Second, the coverage metric is based on states, not paths. Path coverage would be an ideal coverage metric because it can provide coverage of actual executions of the circuit over time. State coverage is static in the sense that a state may be reached via several paths and property verification over any one of those paths will cover that state. Unfortunately, the behavior along an unverified path may be incorrect. However, path coverage is a much more intractable problem given the sheer number of execution paths that even a small design can exhibit. In our opinion, state coverage is the best possible metric which trades off completeness with computation efficiency.

Given these limitations, it is clear that 100% coverage does not guarantee completeness of the verification nor correctness of the circuit. However, coverage short of 100% definitely implies incompleteness of properties. As such, the coverage estimator has more value in helping the user uncover holes in the property suite and improve FV quality rather than provide a guarantee of completeness. In this role, it is a highly effective tool.

7 Conclusions

We addressed the need and challenge of developing a coverage metric within model checking based verification methodologies. Although model checking is exhaustive with respect to the verified properties, it is very difficult to determine whether sufficient properties have been specified and whether all circuit behavior has been verified. We have proposed a coverage metric for model checking that is applicable to a significant subset of CTL. An efficient coverage estimation algorithm was implemented and tested in the course of several real-world model checking efforts. The experiments indicate that the coverage metric can identify meaningful coverage holes that can lead to the discovery of bugs that escaped the model checking effort. We believe this paper breaks new ground in the area of formal verification and addresses an important issue that is one of the keys to making formal verification a widely used technology.

8 Acknowledgements

We would like to thank Victor Konrad and Carl Seger for valuable discussions and John O'Leary for help with the examples.

References

[1]K. L. McMillan, "Symbolic Model Checking: An Approach to the State Explosion Problem," Kluwer Academic, 1993.

[2]E. Clarke, E. Emerson and A. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," *ACM Transactions on Programming Languages and Systems*, vol 8, no. 2, pp.244-263, April, 1986.

[3]K.-T. Cheng, A. Krishnakumar, "Automatic Functional Test Generation Using the Extended Finite State Machine Model," *Proceedings of DAC*, pp.86-91, June 1993

[4]R. Ho, C. Yang, M. Horowitz, D. Dill, "Architecture Validation for Processors," *Proceedings of the 22nd Annual Symposium on Computer Architecture*, June 1995

[5]Y. Hoskote, D. Moundanos, J. Abraham, "Automatic Extraction of the Control Flow Machine and Application to Evaluating Coverage of Verification Vectors," *Proceedings of ICCD*, pp. 532-537, October 1995

[6]M. Kantrowitz, L. Noack, "I'm Done Simulating: Now What? Verification Coverage Analysis and Correctness Checking of the DEC chip 21164 ALPHA Microprocessor," *Proceedings DAC*, pp. 325-330, June 1996

[7]R. Bryant, "Graph-based Algorithms for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, 1986

[8]H. Cho, G. Hachtel, F. Somenzi, "Redundancy Identification and Test Generation for Sequential Circuits Using Implicit State Enumeration," *IEEE Transactions on CAD*, vol 12, no. 7, pp. 935-945, 1993

[9]P.-H. Ho, A.Isles, T.Kam, "Formal Verification of Pipeline Control using Controlled Token Nets and Abstract Interpretation," *Proceedings of ICCAD*, pp 529-536, November 1998.