Mutation Coverage Estimation for Model Checking

Te-Chang Lee and Pao-Ann Hsiung

Department of Computer Science and Information Engineering,
National Chung Cheng University, Chiayi, Taiwan−621, ROC
hpa@computer.org

Abstract. When engineers design a system, there is always a question about how exhaustive the system has been examined to be correct. Coverage estimation provides an answer to this question in testing. A model checker verifies a design exhaustively, and proves the satisfaction of property specifications. However, people have noticed that design errors exist even after model checking is done, which goes to show that the question "How complete is the model checking once done?" is still left relatively unaddressed by model checkers, except for some state-based coverage metrics and the coverage estimator for symbolic simulation in RED. As a more complete solution, we propose several structural mutation models and coverage metrics to cover different design aspects in a state graph and to estimate the completeness of model checking, respectively. Once a system state graph satisfies a given set of property specifications, we estimate the coverage of completeness for the set of properties by applying some mutations to the state graph and checking if the given set of properties is sensitive to the mutation. Our experiences on five application examples demonstrate how the proposed coverage estimation methodology helps verification engineers to find the uncovered hole.

1 Introduction

A model checker explores all possible states of a system model and proves if it satisfies a given set of property specifications. If a system does not satisfy a property specification, a model checker will provide a *counterexample*, that is, a system computation to show how it ran into a wrong state. With the capability to exhaustively verify a system, this method has successfully verified complex circuit designs and communication protocols. However, model checking still faces some obstacles, we are not sure if a design will function correctly even after model checking. At the worst, we may have checked a model satisfies a vacuous property, which does not check anything meaningful. Take as an example, a property $\mathrm{AG}(\mathrm{req} \rightarrow \mathrm{AF}(\mathrm{granted}))$ to check if a model is starvation-free. If there are no requests issued in a system model, this property is indeed true. Though we can detect such vacuous properties [3], but how do we know whether a given set of properties fully verifies the whole model. Due to this need, Hoskote et al. [9] defines a state coverage metric in symbolic model checking. The computation algorithm is based on mutating an observed signal at a certain state, and this state is covered if the truth values of properties are changed. Some other research work [5,6,7,10] are all based on this state coverage metric, which do not consider the computation runs or behavior of a system model. To obtain a more comprehensive coverage, we propose coverage metrics that not only considers the individual states, but also takes the behavior of models into consideration.

Coverage has been widely applied in simulation and testing in the recent few decades. Various metrics have been proposed and applied to real world cases to assess progress of the verification. A low coverage will allow lots of interesting corner cases to escape, which always leads to functional failure. On the contrary, 100% coverage means that the corresponding kind of error is fully verified by the given test patterns. Model checking faces the same situation, the given properties may overlook some desired behaviors of a system. Based on our verification experiences, we know there are lots in common between simulation and formal verification. Because of the similarity, we can adapt existing test based coverage techniques to ensure the quality and reliability of model checking. Some prior researches also tried to apply coverage estimation to formal verification. Some mutation metrics we proposed here are inspired by coverage metrics in testing.

In contrast to state coverage metrics [5,6,7,9,10] being static estimates on model structure, our proposed metrics estimate the coverage of behavior for a system design model. Six mutation models and coverage metrics are proposed to address different aspects of a system design. We implement the six metrics for coverage estimation, and integrate them into overall coverages. We provide a comprehensive estimation of the target model with respect to the specifications. We can prove that the proposed metrics are suitable for analyzing the model checking efforts.

The remaining portion is organized as follows. Section 2 gives the coverage approaches in simulation and pioneer researches in improving quality of formal verification. Some related definitions are given in Section 3. Section 4 will formulate each of our metrics and show how the values of coverage of metrics are calculated. The overall estimation methodology is also described in this section. In Section 5, we apply the new methodology to some real cases, we employ coverage analysis to find the uncovered holes and specify more detailed properties to uncover design holes. The article is concluded and future research directions are given in Section 6.

2 Previous Work

Coverage estimation has been used to assess the quality of design and implementation verification for a long time. For both software and hardware, we feed test cases into a program or a circuit and observe if outputs are correct. After test for expected scenarios, most verification engineers think the verification is done and prepare to tape out. However, due to design getting more and more complicated, more unexpected errors escape from verification. Some of them cause huge financial damage, even threatening peoples' lives. To achieve more reliable verification, coverage estimation is employed in general. Coverage provides a quantitative way to assess how thorough is the verification we have done. Coverage-driven testing has been the main stream of modern hardware simulation, which use coverage to direct the generation of test vectors for exercising untested functions. Many coverage metrics have been been proposed and applied in simulation. Each of them have been created for different needs.

Code coverage [13] measures how complete a piece of HDL code has been exercised by a given set of test patterns. For example, statement coverage calculates the percentage of lines of code stimulated during simulation, so that designers can redirect the generation of test suite or modify the HDL program to gain higher coverage. However, the interaction between modules, simultaneous events, and sequences of events are not evaluated. In spite of these drawbacks, this kind of metrics is simple and easy to understand, verification engineers treat them as the basic requirements of design validation.

Fault simulation [14] is widely employed in gate-level circuit testing. It estimates the fault coverage with respect to several kinds of fault model. By simulating the test pattern upon faulty models, we can calculate how many injected faults are detected by comparing the output to the original one. Fault coverage is calculated as the proportion of detected faults to injected faults. Some gate-level fault models can also be applied to higher-level designs [1,12]. We exploit the principle in these techniques in our proposed formal approach. Experience in testing shows that fault simulation is an effective way to reflect the target fault and directs test pattern generation. Applying the adequate fault models can make sure that interested faults are under verification.

There are some other prior approaches proposed to address the same problem in formal verification. Vacuity detection [3] of the specification provides an answer on the validity of individual properties. Some properties are trivially true like antecedent failure. In their experience, about 20 percent formulae passed vacuously during the first verification. Vacuity detection can avoid those meaningless properties to decrease verification load. Another approach is to estimate the coverage of a specification with respect to a system model [9,11]. Katz et al. [11] suggested that the reduced tableau of a set of ACTL properties should be bisimilar to the implementation. The relevance between the implementation and the specification is compared to estimate verification coverage and to find out the incompleteness of the model or the insufficiency of the specification. Hokote et al. proposed a state coverage metric, which was inspired by *mutation coverage* [17]. They apply a mutation on an observable signal in a particular state. A state is *covered* if the mutation leads to the violation of a given property. The authors of [10] improve the algorithm from [9] to be more efficient and general. The seminal work of Hoskote et al. [9] was also extended to LTL model checking [5], to full CTL model checking [6], and to the simulation of specifications [4]. However, all these work are based on the state coverage metric. Chockler et al. [7] made a brief survey of several different kinds of coverage metrics for formal verification, but did not address the practicality of the metrics, nor provide any application examples to show their usability.

Wang et al. proposed several numerical coverage metrics for the symbolic simulation of real-time systems [16]. Four criteria were introduced for analyzing coverage metric properties. Coverage metrics were also proposed for region spaces in that work. The coverage estimator for symbolic simulation was implemented in the RED model checker for real-time systems.

In this work, we propose a formal coverage estimation methodology for analyzing *the completeness of a set of specification properties* in the model checking of real-time systems. We use a mutation-based approach where a state-graph model is mutated and it is checked if a given set of properties can distinguish the mutation. Six different mutation models and corresponding coverage metrics are proposed. In contrast to conventional state-based metrics, we also propose behavioral (transition-based) metrics, which is similar in some respects to that proposed in [16], except that it is used for estimating the model coverage by a set of properties. The proposed mutation models and coverage metrics were implemented in the *State-Graph Manipulators* (SGM) model checker [15].

3 System Model and Specification

Our system model with real-time clocks is based on the *timed automata* model [2], which is defined as follows.

Definition 1. **Mode Predicate**
Given a set C of clock variables and a set D of discrete variables, the syntax of a *mode predicate* η over C and D is defined as: $\eta := \mathit{false} \mid x \sim c \mid x - y \sim c \mid d \sim c \mid \eta_1 \wedge \eta_2 \mid \neg\eta_1$, where $x, y \in C$, $\sim \in \{\leq, <, =, \geq, >\}$,$c \in \mathcal{N}$, $d \in D$, and η_1, η_2 are mode predicates.

Let $B(C, D)$ be the set of all mode predicates over C and D.

Definition 2. **Timed Automaton**
A *Timed Automaton* (TA) is a tuple $\mathcal{A}_i = (M_i, m_i^0, C_i, D_i, L_i, \chi_i, T_i, \lambda_i, \tau_i, \rho_i)$ such that: M_i is a finite set of modes, $m_i^0 \in M$ is the initial mode, C_i is a set of clock variables, D_i is a set of discrete variables, L_i is a set of synchronization labels, and $\epsilon \in L_i$ is a special label that represents asynchronous behavior (i.e. no need of synchronization), $\chi_i : M_i \mapsto B(C_i, D_i)$ is an *invariance* function that labels each mode with a condition true in that mode, $T_i \subseteq M_i \times M_i$ is a set of transitions, $\lambda_i : T_i \mapsto L_i$ associates a synchronization label with a transition, $\tau_i : T_i \mapsto B(C_i, D_i)$ defines the transition triggering conditions, and $\rho_i : T_i \mapsto 2^{C_i \cup (D_i \times \mathcal{N})}$ is an *assignment* function that maps each transition to a set of assignments such as resetting some clock variables and setting some discrete variables to specific integer values.

A system state space is represented by a *system state graph* as defined in Definition 3.

Definition 3. **System State Graph**
Given a system $\mathcal{S}$ with n components modeled by $\mathcal{A}_i = (M_i, m_i^0, C_i, D_i, L_i, \chi_i, T_i,$

$\lambda_i, \tau_i, \rho_i)$, $1 \le i \le n$, the system model is defined as a state graph represented by $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n = \mathcal{A}_\mathcal{S} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$, where:

- $M = M_1 \times M_2 \times \ldots \times M_n$ is a finite set of system modes, for a system mode $m = m_1.m_2.\ldots.m_n \in M$, we use the shorthand $m(i)$ to denote the mode m_i.
- $m^0 = m^0_1.m^0_2.\ldots.m^0_n \in M$ is the initial system mode,
- $C = \bigcup_i C_i$ is the union of all sets of clock variables in the system,
- $D = \bigcup_i D_i$ is the union of all sets of discrete variables in the system,
- $L = \bigcup_i L_i$ is the union of all sets of synchronization labels in the system,
- $\chi : M \mapsto B(\bigcup_i C_i, \bigcup_i D_i)$, $\chi(m) = \wedge_i \chi_i(m_i)$, where $m = m_1.m_2.\ldots.m_n \in M$.
- $T \subseteq M \times M$ is a set of system transitions which consists of two types of transitions:
  - *Asynchronous transitions*: $\exists i, 1 \le i \le n, e_i \in T_i$ such that $e_i = e \in T$
  - *Synchronized transitions*: $\exists i, j, 1 \le i \ne j \le n, e_i \in T_i, e_j \in T_j$ such that $\lambda_i(e_i) = (l, in)$, $\lambda_j(e_j) = (l, out)$, $l \in L_i \cap L_j \ne \emptyset$, $e \in T$ is synchronization of e_i and e_j with conjuncted triggering conditions and union of all transitions assignments (defined later in this definition)
- $\lambda : T \mapsto L$ associates a synchronization label with a transition, which represents a blocking signal that was synchronized, except for $\epsilon \in L$.
- $\tau : T \mapsto B(\bigcup_i C_i, \bigcup_i D_i)$, $\tau(e) = \tau_i(e_i)$ for an asynchronous transition and $\tau(e) = \tau_i(e_i) \wedge \tau_j(e_j)$ for a synchronous transition, and
- $\rho : T \mapsto 2^{\bigcup_i C_i \cup (\bigcup_i D_i \times \mathcal{N})}$, $\rho(e) = \rho_i(e_i)$ for an asynchronous transition and $\rho(e) = \rho_i(e_i) \cup \rho_j(e_j)$ for a synchronous transition. □

For hardware and software designs, a property specification is usually expressed in some *temporal logic*. The SGM model checker chooses TCTL as its logical formalism, as defined below.

Definition 4. Timed Computation Tree Logic (TCTL)
A *timed computation tree logic* formula has the following syntax:

$$\phi ::= \eta \mid EG\phi' \mid E\phi' U_{\sim c} \phi'' \mid \neg\phi' \mid \phi' \vee \phi',$$

where η is a mode predicate, ϕ' and ϕ'' are TCTL formulae, $\sim \in \{<, \le, =, \ge, >\}$, and c $\in \mathcal{N}$. $EG\phi'$ means there is a computation from the current state, along which ϕ' is always true. $E\phi' U_{\sim c} \phi''$ means there exists a computation from the current state, along which ϕ' is true until ϕ'' becomes true, within the time constraint of $\sim c$. Traditional shorthands like EF, AF, AG, AU, $\wedge$, and $\rightarrow$ can all be defined [8].

4 Mutation Coverage Estimation

Coverage estimation techniques for formal verification such as model checking are not as mature as that for simulation-based verification. A well-studied metric is a purely state-based one [4,5,6,9], where an observed signal in a state is flipped (value toggled) to check if the satisfaction of any user-given property is affected (model checking result toggled). If the model checking result differs after toggling a signal value in a state, then the state is said to be *covered*. Intuitively, besides an observed signal in a state, there are other basic elements in a state-transition model that also needs to be *covered* [7,16]. This work provides some basic insights to the coverage estimation techniques based on *mutating* other elements of a system model.

This work is based on mutation coverage, which makes changes to a system model and then checks if a given property suite can detect those changes. *Mutating* a system model can be done is two ways: (1) Semantic Mutation, that is, changing the value some basic elements in the model, e.g., variable values in a state, clock resets along a transition, etc., and (2) Structural Mutation, that is, changing the structure of the model, e.g., inserting or deleting a state or a transition. Currently, our work is focused on structural mutation for timed automata with timed computation tree logic properties.

Similar to the conventional fault models in simulation-based testing, we propose *structural mutation models* that can be applied to system state-space representations such as state graphs. Henceforth, unless explicitly mentioned, we will simply use mutation models to denote structural mutation models.

A mutation model is a small change that is applied to some basic element of a system model such as a mode or a transition of a state graph. Given a state-graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$ and a mutation model μ that is applicable to some basic element $b \in M \cup T$, the resulting state-graph is called a *mutated* state graph and is denoted as $\mathcal{A}^{\mu(b)} = (M', m'^0, C, D, L, \chi', T', \lambda', \tau', \rho')$. If $\mathcal{A}^{\mu(b)}, m'^0 \not\models \phi$ for some $\phi \in P$ then the mutation $\mu(b)$ is said to be *covered* by ϕ and the mode or transition b is said to be *covered* by ϕ. A property ϕ is said to be *insensitive* to a mutation $\mu(b)$ on a state graph $\mathcal{A}$ if $\mathcal{A}^{\mu(b)}, m'^0 \models \phi$ unvacuously. A mutation $\mu(b)$ is said to be *not covered* by P if $\mathcal{A}^{\mu(b)}, m'^0 \models \phi$ for all $\phi \in P$.

4.1 Coverage Estimation Methodology

Our target *mutation coverage estimation* problem can be formulated as follows. Given a system modeled by a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$, a set of properties specified as TCTL formulae $P = \{\phi\}$, and a mutation model μ, suppose $\mathcal{A}, m^0 \models \phi, \forall \phi \in P$, estimate the *completeness* of P with respect to $\mathcal{A}$ and μ, where completeness is the fraction of the mutations detected by some property in P.

As shown in Figure 1, a solution to the above stated problem is proposed as a mutation coverage estimation procedure that can be integrated with model checking. Given a system state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$ and a set $P = \{\phi\}$ of TCTL property specifications, the mutation coverage estimation procedure starts only after it is verified that $\mathcal{A}, m^0 \models \phi$ for all $\phi \in P$. In other words, if a system state graph violates some given properties, a designer should revise the model or the properties before estimating the coverage. When a state graph satisfies all given properties, a mutation model μ is applied to

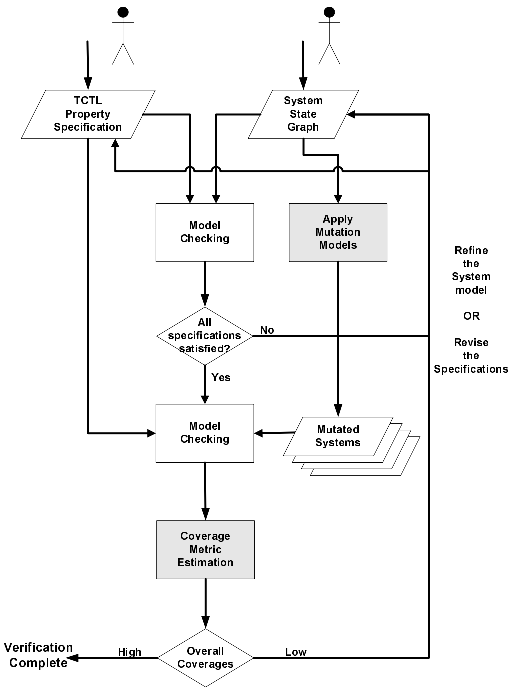

Fig. 1. Coverage Estimation Methodology

some basic element $b \in M \cup T$ of the graph to obtain a mutated state graph $\mathcal{A}^{\mu(b)}$. Model checking is re-performed on the mutated graph and it is checked if the mutation is covered (Definition 5). The application of mutation model and the model checking are repeated for each mode or each transition depending on the mutation model. After coverage estimation, if the value of a coverage metric is too low, that is, smaller than some user-defined threshold value, then more properties are to be specified by analyzing the uncovered parts in a state graph.

Definition 5. Covered Mutation
For a given system state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$ and a set of TCTL properties $P = \{\varphi\}$, suppose $\mathcal{A}, m^0 \models \varphi$ for all $\varphi \in P$. Further, for a given mutation model μ that is applied on some element $b \in M \cup T$ of the state graph $\mathcal{A}$, suppose the mutated state graph is $\mathcal{A}^{\mu(b)} = (M', m'^0, C, D, L, \chi', T', \lambda', \tau', \rho')$. If $\mathcal{A}^{\mu(b)}, m'^0 \not\models \varphi$, for some $\varphi \in P$, then the mutation $\mu(b)$ is said to be *covered* by P for $\mathcal{A}$.

In this work, we propose several mutation models and corresponding mutation-based coverage metrics to aid engineers in analyzing if a set of properties has covered most functionalities of a system. The proposed coverage metrics are calculated automatically without any effort beyond model checking. Ideally, the coverage should achieve 100% for each proposed metric, which implies the given set of specifications has covered every corner of a system design in each design aspect. However, when a state space becomes large, it is really hard to achieve 100% coverage for each metric, especially the transition-based ones.

The value of a coverage metric $cov(\mathcal{A}, P, \mu)$ with respect to a mutation model μ for a state graph $\mathcal{A}$ and a set of properties P is a ratio of the number of covered mutations to the total number of mutations applied, where covered mutations are as defined in Definition 5 and total number of mutations is the total number of basic elements reachable in a state graph, to which the corresponding mutation was applied.

$$cov(\mathcal{A}, P, \mu) = \frac{\#\textbf{Covered Mutations}}{\#\textbf{Total Mutations}} = \frac{|\{b|\exists \phi \in P, \mathcal{A}^{\mu(b)} \not\models \phi\}|}{|\{b\}|} \tag{1}$$

In the above, the total number of mutations differ based on whether the mutation was applied to system modes or transitions.

For a given state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$ and a set $\{\varphi\}$ of TCTL formulae, the complexity of labeling in model checking $\mathcal{A}$ against φ is $\mid \varphi \mid \times \mid \mathcal{A} \mid$, where $\mid \varphi \mid$ is the number of sub-formulae in φ and $\mid \mathcal{A} \mid = |M| + |T|$ is the size of the state-graph. The complexity of our proposed mutation-based coverage estimation is $O(model\ checking) \times |M|$ for state-based mutations and $O(model\ checking) \times |T|$ for transition-based mutations.

If any entry of TCTL properties has zero coverage for all metrics, this property is said to be *vacuous*. This feature is similar to *vacuity detection* [3].

4.2 Mutation Models and Coverage Metrics

Based on our verification experiences, we propose six different structural mutation models and corresponding coverage metrics. Each mutation model characterizes a different aspect of a system state graph and identifies different characteristics of TCTL properties. In general, a more restricted property will cover more parts of a system model. For example, when the time interval in a TCTL formula gets shorter, the larger is the coverage. On the contrary, the satisfaction of an eventuality property $EF\phi$ only requires a state satisfying ϕ along some path. So, it always covers few states, but several transitions. In the following, we will assess each mutation model and corresponding coverage metric when verifying different kinds of properties. Metrics can also be combined together to give overall estimations.

Mutated Initial. This model is a mode-based mutation that changes the *initial mode* of a system state graph to be another one. Given a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$, the mutated initial model $\mu_{initial}$ when applied to a non-initial mode $m \in M, m \neq m^0$ gives a mutated state graph $\mathcal{A}^{\mu_{initial}(m)} = (M, m, C, D, L, \chi, T, \lambda, \tau, \rho)$.

An initial state specifies the initial values of all variables in the system. For a good system design, each initial value should be explicitly specified, that is, there should be a unique initial state. If a property specifies initial values for all variables, that is, it is satisfied only in a particular initial state, then the corresponding coverage estimation metric will take a value of 100 %. Otherwise, it will be a percentage of the number of non-initial states that cannot play the role of an initial state. A formal definition of the coverage metric for this mutation model is given in Equation 2.

$$cov(\mathcal{A}, P, \mu_{initial}) = \frac{|\{m | m \in M, \exists \phi \in P, \mathcal{A}^{\mu_{initial}(m)}, m \not\models \phi\}|}{|M|-1} \tag{2}$$

Because this model changes only the initial mode to another one and because the labeling algorithm in model checking does not distinguish between initial and non-initial states, the labels in the modes need not be re-computed. Hence, the coverage estimation procedure for this mutation model is constant for each iteration.

Delayed Transition. This model is a transition-based mutation that delays a mode transition by inserting a new mode between the source and destination modes of the transition. For each transition $e = (m_s, m_d) \in T$, a new mode m is inserted between m_s and m_d such that the following conditions are met: (1) $\chi(m) = \chi(m_s)$, (2) the transition $e' = (m_s, m)$ is non-deterministically timed that is $\tau(e') = true$, and (3) the transition e is now changed to originate from m, that is, $e = (m, m_d)$. Given a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$, the delayed transition model μ_{delay} when applied to a transition $e = (m_s, m_d) \in T$ gives a mutated state graph $\mathcal{A}^{\mu_{delay}(e)} = (M', m^0, C, D, L, \chi', T', \lambda', \tau', \rho')$, where $M' = M \cup \{m_{delay}\}$, m_{delay} is a newly introduced mode, $\chi'(m) = \chi(m), \forall m \in M$ and $\chi'(m_{delay}) = \chi(m)$, $T' = T \cup \{(m_s, m_{delay}), (m_{delay}, m_d)\} \backslash \{e\}$, $\lambda'(e') = \lambda(e'), \forall e' \in T \backslash \{e\}$, $\lambda'((m_s, m_{delay})) = \epsilon$, $\lambda'((m_{delay}, m_d)) = \lambda(e)$, $\tau'(e') = \tau(e'), \forall e' \in T \backslash \{e\}$, $\tau'((m_s, m_{delay})) = true$, $\tau'((m_{delay}, m_d)) = \tau(e)$, $\rho'(e') = \rho(e'), \forall e' \in T \backslash \{e\}$, $\rho'((m_s, m_{delay})) = \emptyset$, and $\rho'((m_{delay}, m_d)) = \rho(e)$.

The coverage metric corresponding to the delayed transition model is defined in Equation (3).

$$cov(\mathcal{A}, P, \mu_{delay}) = \frac{|\{e | e \in T, \exists \phi \in P, \mathcal{A}^{\mu_{delay}(e)}, m^0 \not\models \phi\}|}{|T|} \tag{3}$$

This metric provides a measure of whether the timing in a system design is correct to the specification. The metric was inspired by the *delay fault models* found in simulation-based coverage estimation.

Stuttering Mode. This model is a mode-based mutation that adds a self-loop transition to a mode. Given a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$, the stuttering mode model $\mu_{stutter}$ when applied to a mode $m \in M$ gives a mutated state graph $\mathcal{A}^{\mu_{stutter}(m)} = (M, m^0, C, D, L, \chi, T', \lambda', \tau', \rho')$, where $T' = T \cup \{e\}$, $e = (m, m)$, $\lambda'(e) = \epsilon$, $\tau'(e) = true$, $\rho'(e) = \emptyset$, and all other transitions have unaltered synchronization labels, triggers, and assignments.

Applying the stuttering mode mutation model to a mode creates a single-node *strongly connected component* (SCC) if the mode itself was not in an SCC before mutation. SCCs are required for infinite computation runs such as in checking fairness constraints and in checking CTL properties such as EGϕ and AFϕ. In hardware systems, such a mutation model has an effect of allowing a component to remain in a particular state forever, while still synchronizing with a clock along the looping transition. The coverage metric corresponding to this mutation model is thus an estimation of the number of modes that could be detected by the properties if they were to stutter. Though this metric is a mode-based one, it actually estimates the progress of transitions or computation runs.

$$cov(\mathcal{A}, P, \mu_{stutter}) = \frac{|\{m | m \in M, \exists \phi \in P, \mathcal{A}^{\mu_{stutter}(m)}, m^0 \not\models \phi\}|}{|M|} \tag{4}$$

Skipped Mode. This model is a mode-based mutation that makes a non-initial mode unreachable by redirecting all of its incoming transitions to all of its child modes. Given a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$, the skipped mode model μ_{skip} when applied to a mode $m \in M, m \neq m^0$ gives a mutated state graph $\mathcal{A}^{\mu_{skip}(m)} = (M', m^0, C, D, L, \chi', T', \lambda', \tau', \rho')$, where $M' = M \backslash \{m\}$, $\chi'(m') = \chi(m'), \forall m' \in M'$, $T' = T \backslash \{e \mid e = (m_s, m)\} \cup \{e \mid e = (m_s, m_d)$, m_s, m_d are the predecessor and successor modes of m, for each new transition e' that corresponding to a deleted transition e, $\lambda'(e') = \lambda(e)$, $\tau'(e) = \tau(e)$, $\rho'(e) = \rho(e)$, and all other transitions have unaltered synchronization labels, triggers, and assignments.

This mutation model has a reversed effect compared to that of the *Delayed Transition* model because skipping a mode implies a reduction of the computation run into a shorter one, which in turn, implies a shorter time is required to reach the modes beyond the skipped one. The temporal sequence of modes differs after mutation. This mutation model is not applied on the initial state because it has no incoming transition.

Since the skipped mode mutation model tests the presence of each individual mode, the corresponding coverage metric is an estimation on the number of modes whose existence can be detected by a set of properties.

$$cov(\mathcal{A}, P, \mu_{skip}) = \frac{|\{m | m \in M, m \neq m^0, \exists \phi \in P, \mathcal{A}^{\mu_{skip}(m)}, m^0 \not\models \phi\}|}{|M|-1} \tag{5}$$

Removed Transition. This model is a transition-based mutation that deletes a transition from a state graph. This mutation checks if the existence of a transition is detectable by a given set of property specifications. If a property is sensitive to the existence of a particular transition, then it will not be satisfied in the mutated state graph. Given a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$, the removed transition model $\mu_{removed}$ when applied to a transition $e = (m_s, m_d) \in T$ gives a mutated state graph $\mathcal{A}^{\mu_{removed}(e)} = (M', m^0, C, D, L, \chi', T', \lambda', \tau', \rho')$, where $M' = M \backslash \{m \mid m \in M,\ m \text{ is unreachable after removing } e\}$, $\chi'(m') = \chi(m), \forall m' = m \in M' \cap M$, $T' = T \backslash \{e' \mid e' \in T,\ e' \text{ is unreachable after removing } e\}$, and all remaining transitions in T' have the same synchronization labels, triggers, and assignments as their original counterpart in T.

The coverage metric corresponding to the removed transition mutation model gives an estimate on the number of transitions whose existence can be detected by a given set of properties.

$$cov(\mathcal{A}, P, \mu_{removed}) = \frac{|\{e | e \in T, \exists \phi \in P, \mathcal{A}^{\mu_{removed}(e)}, m^0 \not\models \phi\}|}{|T|} \tag{6}$$

Mutated Invariant. This model is a mode-based mutation and is a simplification of that proposed by Hoskote et al. in [9]. In this mutation, instead of toggling the value of an observed signal in a state, its invariant is changed. This is more of a semantic mutation than a structural one.

In our implementation, we have skipped the *observability transformation* and the *dual operation* on observed propositions as defined in [9]. Further, we mutate each mode twice: once with the *false* invariant and once with the *true* invariant. Model checking is also done twice for each mutated mode. Then, we take the union of the covered modes as the coverage for this mutation model. If a mode with a *false/true* invariant does not affect the satisfaction of any given property, then the mode is not covered. Given a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$, the mutated invariant model μ_{inv} when applied to a mode $m \in M$ gives two mutated state graphs $\mathcal{A}^{\mu_{inv}(m,true)} = (M, m^0, C, D, L, \chi', T, \lambda, \tau, \rho)$, where $\chi'(m') = \chi(m'), \forall m' \neq m$ and $\chi'(m) = false$, and $\mathcal{A}^{\mu_{inv}(m,false)} = (M, m^0, C, D, L, \chi', T, \lambda, \tau, \rho)$, where $\chi'(m') = \chi(m'), \forall m' \neq m$ and $\chi'(m) = true$.

As defined in Equation (7), the coverage metric corresponding to the mutated invariant mutation model gives an estimation on the number of modes that can be detected once its invariant is set to false.

$$cov(\mathcal{A}, P, \mu_{inv}) = \frac{|\{m | m \in M, \exists \phi \in P, (\mathcal{A}^{\mu_{inv}(m,true)}, m^0 \not\models \phi) \lor (\mathcal{A}^{\mu_{inv}(m,false)}, m^0 \not\models \phi)\}|}{|M|} \tag{7}$$

4.3 The Overall Coverages

After computing coverage respect to each metric, we can also calculate the overall coverages. The overall coverages show how many parts of a design were ever covered by any metric and integrates all related metrics to show the holes that were never covered in any metric computation for a given set of property specifications.

Based on the basic element we apply mutation to, that is, the type of mutation model, we classify the metrics into two categories.

– *Mode Coverage Metrics*: The metrics corresponding to mode-based mutation models including *Mutated Initial*, *Stuttering Mode*, *Skipped Mode*, and *Mutated Invariant* are called *mode coverage metrics.*
– *Transition Coverage Metrics*: The metrics corresponding to transition-based mutation models including *Delayed Transition* and *Removed Transition* are called *transition coverage metrics.*

Definition 6. Overall Mode Coverage
Given a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$ and a set of TCTL properties $P = \{\phi\}$, the overall mode coverage is defined as ratio of modes covered by any one of the four mode coverage metrics for P. Let the sets of covered states for each of the mode coverage metrics be, respectively, $CS_{initial}$, $CS_{stutter}$, CS_{skip}, and CS_{inv}. These sets are defined in the numerators of the fractions in Equations 2, 4, 5, and 7, respectively. The overall mode coverage is then defined as in Equation (8).

$$cov(\mathcal{A}, P, \mu_{mode}) = \frac{CS_{initial} \cup CS_{stutter} \cup CS_{skip} \cup CS_{inv}}{|M|} \tag{8}$$

Definition 7. Overall Transition Coverage
Given a state graph $\mathcal{A} = (M, m^0, C, D, L, \chi, T, \lambda, \tau, \rho)$ and a set of TCTL properties $P = \{\phi\}$, the overall transition coverage is defined as ratio of transitions covered by any one of the two transition coverage metrics for P. Let the sets of covered transitions for each of the transition coverage metrics be, respectively, CT_{delay} and $CT_{removed}$. These sets are defined in the numerators of the fractions in Equations 3 and 6, respectively. The overall transition coverage is then defined as in Equation (9).

$$cov(\mathcal{A}, P, \mu_{trans}) = \frac{CT_{delay} \cup CT_{removed}}{|T|} \tag{9}$$

5 Experimental Results

We have implemented all the six proposed mutation models into the *State Graph Manipulators* (SGM) model checker [15], which is a high-level model checker for both real-time systems as well as systems-on-chip modeled by a set of timed automata. Several optimizations were implemented into the mutated model generation and the coverage metric calculations. Due to page-limits, we skip this part of the discussions.

We applied our proposed mutation models and estimated the coverage using the corresponding proposed metrics for several practical design models that we created in the course of this project work, including a simple timer, a bridge model for the ARM AMBA bus architecture, an APB slave model in AMBA, a traffic light controller, and a bakery scheduler. Table 1 shows the overall coverage estimation results for the above examples along with the performance results of the implemented coverage estimators. The model checker with coverage estimation programs were executed on a Linux Mandrake 8.1 workstation with a 1.0 GHz Intel Pentium CPU and 512MB memory.

The simple timer is a timer with three stages. After a user-defined time interval, the timer goes to the next stage. When the timer reaches a pre-defined maximum value, it issues a reset signal. At the initial attempt, we specified three properties to verify the timer model, which included checking the time progress (AG(timer=0 $\rightarrow$ AF(timer $>$ 0))), the initial behavior (timer $=$ 1), and the effect of reset (AG(reset=0 $\rightarrow$ AF(timer=0))). After applying our proposed

Table 1. Overall Coverage Estimation Results

State Graph ($\mathcal{A}$)	**Graph Size** $\lvert M\rvert/\lvert T\rvert$	$\lvert P\rvert$	**Time**[*]/ **Memory**[**]	$cov(\mathcal{A}, P, \mu_{mode})(\%)$/ $cov(\mathcal{A}, P, \mu_{trans})(\%)$
Simple Timer	18/18	3	0.30/0.40	100.00/ 91.67
		4	0.32/0.44	100.00/100.00
AMBA APB Bridge	10/24	8	0.45/0.01	100.00/ 91.67
AMBA APB Slave	26/168	3	1.46/0.08	100.00/100.00
Traffic Light Controller	253/542	5	210.89/1.70	90.00/ 79.17
Bakery Scheduler	1293/2073	5	2087.65/1.08	68.36/ 10.51

[*]Time is in seconds, [**]Memory is in MB,
Note: the μ_{inv} model is not included in these overall estimations.

coverage metric estimation, as shown in Table 1, the mode coverage was 100 %, but the transition coverage was only 91.67 %. On analyzing the uncovered traces generated by the SGM coverage estimator, we found one of the transitions in the state graph model of the timer was not covered by the three properties. The uncovered transition deasserts the reset signal after it was asserted. Later, we wrote an additional property (AG(reset=1 $\rightarrow$ EF(timer=0))) to cover this deassertion behavior and it was possible to achieve 100% transition coverage.

Table 2 shows the coverage metrics estimated for each mutation model. From the numeric coverages, we get an idea of the completeness of the given properties. In Table 2, a threshold of 30% was assumed for high-lighting all coverages that are below the threshold percentage. We observe that the AMBA APB slave was the most uncovered application. Evidently, the three given properties are insufficient and more are needed to perform a thorough verification of the model. The other poorly covered application is the bakery scheduler. Though five properties were specified for this system, however the model itself was the largest in size among all the examples, hence many more properties are required to have a higher coverage. From the above two poorly covered examples, we can conclude that poor coverages are a result of an unproportionate number of properties

Table 2. Coverage Metric Estimations for each Mutation Model

State Graph ($\mathcal{A}$)	$\lvert P\rvert$	$cov(\mathcal{A}, P, \mu)$ (%)					
		$\mu_{initial}$	μ_{delay}	$\mu_{stutter}$	μ_{skip}	μ_{remove}	μ_{inv}
Simple Timer	4	100.00	94.44	**5.88**	100.00	94.44	100.00
AMBA APB Bridge	8	44.44	91.66	90.00	100.00	**20.83**	100.00
AMBA APB Slave	3	**0.00**	100.00	**0.00**	**4.00**	**4.77**	100.00
Traffic Light Controller	5	53.57	72.61	**11.68**	**0.00**	**13.69**	100.00
Bakery Scheduler	5	52.02	**10.36**	**22.19**	**1.29**	**0.49**	100.00

Boldface represents below threshold coverages (threshold = 30%).

compared to the size of the system model. In other words, larger the system, the more properties are required for a more complete verification. The two best covered application examples are the simple timer and the AMBA APB Bridge, which also show that it is relatively easier to achieve high coverages for small and simple systems.

As far as mutation models are concerned, we can observe from Table 2 that the μ_{inv} model achieved a 100 % coverage for all the examples, whereas the μ_{remove} model achieved an above threshold coverage only for the simple timer. These observations illustrate the different natures of the models and the relative ease or difficulty with which we can achieve higher coverages for different mutation models. Hence, it is deduced that not necessarily do we have to increase all coverages. It depends on the characteristics of the application example itself as described in the following.

Timing delay is an important factor in both the simple timer and the traffic light controller examples, hence its coverage as modeled by μ_{delay} was required to be as high as possible. Currently, the obtained coverages of 94.44% and 72.61%, respectively, are still quite low. Starvation is an undesired feature in the bakery scheduler example, hence its coverage as modeled by $\mu_{stutter}$ was required to be as high as possible. The five given properties achieved only 22.19% stuttering mode coverage, which shows that more properties are required.

6 Conclusions and Future Work

We have proposed a coverage estimation methodology to give formal verification a quantitative statistics on how exhaustive it is. Based on the estimation and analysis of uncovered traces, the verification engineer can decide whether a further verification iteration is required or not. Besides, the log file shows what parts of a system model are not covered and thus need more properties to exercise, or a user may also choose to refine the system model. Instead of focusing on the static analysis of states as in several previous work, we proposed six different mutation models and their corresponding coverage metrics to capture behaviors of a model, which is also the most error-prone. The proposed estimation needs no extra effort beyond conventional model checking. Further, the complexity of estimation is acceptable in practice, as we have shown in Section 5. Future work consist of proposing some semantic mutation models and coverage metrics and making the structural mutation models more exhaustive and complementary.

References

1. M. Abramovici. Dos and don'ts in computing fault coverage. In *Proceedings of the International Test Conference (ITC'93)*, page 594, October 1993.
2. R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

3. I. Beer, S. Ben-David, C. Eisner, and Y. Rodeh. Efficient detection of vacuity in ACTL formulas. In *Proceedings of the International Conference on Computer-Aided Verification (CAV'97), Lecture Notes in Computer Science 1254*, pages 279–290. Springer-Verlag, June 1997.
4. H. Chockler and O. Kupferman. Coverage of implementations by simulating specifications. In *Proceedings of the IFIP International Conference on Theoretical Computer Science (TCS 2002)*, pages 409–421. Kluwer, August 2002.
5. H. Chockler, O. Kupferman, R. P. Kurshan, and M. Y. Vardi. A practical approach to coverage in model checking. In *Proceedings of the International Conference on Computer Aided Verification (CAV'01), Lecture Notes in Computer Science 2102*, pages 66–78. Springer-Verlag, July 2001.
6. H. Chockler, O. Kupferman, and M. Y. Vardi. Coverage metrics for temporal logic model checking. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science 2031*, pages 528–542. Springer-Verlag, April 2001.
7. H. Chockler, O. Kupferman, and M. Y. Vardi. Coverage metrics for formal verification. In *Proceedings of the International Conference on Correct Hardware Design and Verification Methods (CHARME), Lecture Notes in Computer Science 2860*, pages 111–125. Springer-Verlag, October 2003.
8. T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. In *Proceedings of the IEEE International Conference on Logics in Computer Science (LICS'92)*, pages 394–406, June 1992.
9. Y. Hoskote, T. Kam, P.-H. Ho, and X. Zhao. Coverage estimation for symbolic model checking. In *Proceedings of the Design Automation Conference*, pages 300–305, June 1999.
10. N. Jayakumar, M. Purandare, and F. Somenzi. Dos and don'ts of CTL state coverage estimation. In *Proceedings of the Design Automation Conference (DAC'03)*, pages 292–295. ACM Press, June 2003.
11. S. Katz, O. Grumberg, and D. Geist. "Have I written enough properties?" – A method of comparison between specification and implementation. In *Proceedings of the Correct Hardware Design and Verification Methods (CHARME'99), Lecture Notes in Computer Science 1703*, pages 280–297. Springer-Verlag, September 1999.
12. Von-Kyoung Kim, Tom Chen, and Mick Tegetho. Fault coverage estimation for early stage of VLSI design. In *Proceedings of Ninth Great Lakes Symposium on VLSI (GLSVLSI'99)*, pages 105–108, March 1999.
13. C.-N. Liu and J.-Y. Jou. Efficient coverage analysis metric for HDL design validation. In *Proceedings of IEEE International Conference on Computers and Digital Techniques*, pages 1–6, January 2001.
14. W. Mao and Gulati. R. K. Improving gate level fault coverage by RTL fault grading. In *Proceedings of IEEE International Test Conference 1996, Test and Design Validity*, pages 150–159. IEEE Computer Society, October 1996.
15. F. Wang and P.-A. Hsiung. Efficient and user-friendly verification. *IEEE Transactions on Computers*, 51(1):61–83, January 2002.
16. F. Wang, G.-D. Hwang, and F. Yu. Numerical coverage estimation for the symbolic simulation of real-time systems. In *Proceedings of the 23rd IFIP International Conference on Formal Techniques for Networked and Distributed Systems (FORTE), Lecture Notes in Computer Science 2676*, pages 160–176. Springer-Verlag, September 2003.
17. M.R. Woodward. Mutation testing – an evolving technique. In *Proceedings of IEE Colloquium on Software Testing for Critical Systems*, pages 3/1–3/6, June 1990.