Numerical Coverage Estimation for the Symbolic Simulation of Real-Time Systems*

Farn Wang[1], Geng-Dian Hwang[2], and Fang Yu[2]

[1] Dept. of Electrical Engineering, National Taiwan University,
Taipei, Taiwan 106, Republic of China,
+886-2-23635251 ext. 435; FAX +886-2-23671909,
farn@cc.ee.ntu.edu.tw

[2] Institute of Information Science, Academia Sinica,
Taipei, Taiwan 115, Republic of China,
Tools available at: http://cc.ee.ntu.edu.tw/~val

Abstract. Three numerical coverage metrics for the symbolic simulation of dense-time systems and their estimation methods are presented. Special techniques to derive numerical estimations of dense-time state-spaces have also been developed. Properties of the metrics are also discussed with respect to four criteria. Implementation and experiments are then reported.

Keywords: coverage, verification, symbolic simulation, real-time

1 Introduction

Currently, with verification and integration costs increasing to more than 50 percent of the development budget in real-world projects, it is more and more difficult to use traditional simulation technology to acquire enough trace coverage to confidently create system designs. As well, application of the new formal verification technology is still hampered by its intrinsic complexity. In the forseeable future, we expect that simulation and formal verification will be combined for verification of large-scale real-time systems. Symbolic simulation is such a combination[25]. It uses symbolic techniques[3, 7, 11, 16] of formal verification to represent space of simulation traces so that abstract (as opossed to concrete) behaviors can be observed in a trace. For verification of real-time systems, tools like UPPAAL[23] and RED[26–29] support symbolic simulations.

Current simulation technology for real-time systems is still not as developed as that for untimed systems like Very Large Scale Integration (VLSI) Systems. For one thing, the important concept of *coverage* can be used to both estimate

* The work is partially supported by NSC, Taiwan, ROC under grants NSC 91-2213-E-002-131, NSC 91-2213-E-002-132, and by the Internet network protocol verification project of Institute of Applied Science & Engineering Research, Academia Sinica, 2003.

H. König, M. Heiner, and A. Wolisz (Eds.): FORTE 2003, LNCS 2767, pp. 160–176, 2003.

the value of a set of traces and to direct the generation of new traces. In short, coverage is how much of the target to be verified has been verified. The importance of this concept is that, in real-world projects, it is usually the case that we lack enough resources to either run enough traces to obtain confidence, or to complete formal verification tasks. Product designs usually need to be released before we can have 100% confidence in the designs. Therefore it is important that we have some types of coverage metrics to evaluate confidence in our designs[4]. Previously, such concepts have proven to be effective in bug escape reduction by pointing out coverage holes in the test suite.

Another reason that forces us to use symbolic simulation instead of traditional simulation is that the coverage metrics used in traditional simulation, e.g., the *visited-state* coverage metric[5, 24], may not be directly applicable to the simulation of dense real-time systems, since there are an infinite number of concrete traces and states for such systems. If we directly apply this coverage metric to the simulation of dense real-time systems, we will always have 0% coverage since the ratio of a finite number of concrete states over the infinite reachable state-space is always zero. To this end, we propose to use symbolic simulation in which symbolic traces are generated and may correspond to nonzero proportions of the dense reachable state-space.

Before we can estimate numerical coverage, we must design a metric and its estimation procedure. This engenders the first research issue of this work, that is, how do we know if a metric is good? In section 4, we present four criteria to serve as guidelines in coverage metric design. These criteria are: **accountability** (each basic *portion* of the *target function* is counted once and only once), **coverability** (100% coverage estimation is achievable), **efficiency** (the overhead in computing the coverage estimation is low), and **discernment** (risk states are discernable). According to these criteria, we adapt three coverage metrics from traditional testing research and development techniques to implement them in real-time systems. These three new metrics are: *timed automata arc coverage metric(ACM)*, *back-and-forth region coverage metric(RCM)*[24], and *triggering condition coverage metric(TCM).* ACM is a straightforward adaptation from the technology of VLSI simulation, whereas RCM and TCM are not. For dense-time systems, RCM and TCM are more precise in the estimation of coverage by considering state-space coverage. We prove in a lemma that RCM has enough power to discern reachable risk states while ACM lacks sufficient power. To maintain the four criteria for dense-time state-spaces of real-time systems, we develop techniques to quantitatively estimate the volume of a state-space, and to significantly prune irrelevant state-space portions from a verification task.

We present algorithms (based on the region-equivalence relation[1]) to quickly estimate the coverages of symbolic subspaces in RCM and TCM(see section 6.2). Our algorithms take advantage of the data-sharing capability of CDD[6] and do not enumerate all paths of CDD. In our experiment, the overhead for coverage estimation ranges from 1.2% to 25% of the total simulation time. We believe that our techniques can be used to help future development of various coverage-based verification techniques - including the design of new coverage metrics and coverage-based test-pattern generation - in real-time systems.

In section 8, we model two devices that communicate with the *Logical Link Control and Adaptation Layer Protocol(L2CAP)* of Bluetooth[15] and carry out simulation experiments to gather data on numerical coverage estimation. We then compare the three coverage metrics w.r.t. the four criteria.

In section 2, we review related works. In section 3, we briefly present our verification framework with timed automata. In section 4, we discuss the four criteria for effective coverage metrics. In sections 5 through 7, we present our three coverage metrics and their estimation procedures. Finally, in section 8, we report on our experiment results with the Bluetooth L2CAP and discuss the implications of the experiment data.

2 Previous Work

Coverage techniques have been widely discussed and applied in testing, simulation and formal verification of various system designs. In software testing, people use *control flowgraphs* [4], which are composed of processes, decisions, and junctions. Given a set of program stimuli, one can determine the statements activated by the stimuli with the coverage metrics of the flowgraphs. Programming code metrics measure syntactical characteristics of a code w.r.t its execution stimuli. For example, *line coverage metric* measures the number of distinct statements visited during the course of execution, *branch coverage* measures the number of distinct branch decisions, and *path coverage* measures the number of distinct paths (i.e., a unique combination of branch decisions and statements) exercised due to its execution stimuli[5]. The number of paths in a program may be exponentially related to the program's size and greatly hinder attaining 100% path coverage in software testing.

Coverage analysis techniques proposed for general HDL programs include: guaranteed coverage of every statement[8], transition coverage of a test set[19], and abstraction of models and semantic control over transition coverage[14]. Fallah provides OCCOM[13] to address the observability issue. Most of these HDL metrics are used to drive test-generation in simulation analysis.

Ho et al.[20] proposed a coverage metric to estimate the "completeness" of a set of properties verified in model-checking FSM against a subset of CTL. A symbolic algorithm is also presented. Chockler et al.[9] also suggested several coverage metrics to measure completeness of a verified specification, and to find uncovered parts.

Dill proposed a way to bridge the gap between simulation and formal verification [12]. Generator of Test Cases for Hardware Architecture (GOTCHA) is a prototype coverage-driven test generator implemented as an extension of the Murϕ model-checker[22]. It supports state and transition coverage analysis in FSM. On completion of the entire reachable state-space enumeration, a random coverage task is chosen from those not yet covered.

Opposed to previous work on untimed or discrete-time systems, we apply coverage techniques in our symbolic simulator with dense-time models. One difficulty arises in the design of meaningful metrics to estimate state-spaces which are both dense and infinite. Traditional state and transition coverage metrics for untimed or discrete-time systems cannot be directly copied since metrics may always be zero based on the dense domain.

3 Framework of Verification

We use the widely accepted model of *timed automata (TA)*[1]. A TA is a finite-state automaton equipped with a finite set of clocks that can hold nonnegative real-values. A TA can stay in only one *mode* (or *control location*) at a time. In operation, one transition can be triggered when its corresponding triggering condition is satisfied. Triggering conditions are Boolean combinations of atoms like $x \leq c$ and $x - x' \leq c$ where x, x' are clock variables and c is an integer. Upon being triggered, the TA instantaneously transits from one mode to another and resets some clocks to zero. Between transitions, all clocks increase their readings at a uniform rate. Due to page-limit, we refer interested readers to [1].

We adopt the *safety-analysis problem* as our verification framework for simplicity. In this framework, we want to check whether an unsafe state can be reached by repetitive generation of symbolic traces. Our main procedure is based on the well-discussed symbolic procedure, called `next()`, to compute a symbolic post-condition after a discrete transition and time-progress[16, 21]. Our symbolic simulation procedure takes the following form (details on coverage estimation in statements (1) and (4) will be explained in sections 5 through 7).

`Symbolic_Simulate`(A, η) /* A is a TA; η is the safety state predicate. */ {
  Compute the numerical estimation f of the whole *target function* F. (1)
  Let ϕ :=the initial condition of A; $\phi' :=$ *false*; $v := 0$;
  While (true) {
    Let $\phi' := \phi$;
    Select a subspace ψ of ϕ and a set $\bar{T}$ of transitions
      (possibly based on the value ϕ and $\bar{T}$); (2)
    Compute $\phi := \phi \vee \bigvee_{e \in \bar{T}}$ `next`(A, ψ, e); (3)
    Compute the estimation v of the verified proportion V
      of the whole *target function*; (4)
    Print v/f as the new numerical coverage estimation. (5)
    If $\phi \wedge \neg\eta \neq \emptyset$, print "a risk state is reachable" and exit;
    else if $v/f \geq$ *threshold*, print "The coverage threshold is reached" and exit;
    else if $\phi = \phi'$, print "no risk states are found" and exit; (6)
  }
}

In this manuscript, we use the term *"portion"* to mean a basic unit of the target function in the estimation of trace coverage. Formally speaking, given a coverage metric, a portion is an equivalence class of (syntactic or semantic) entities of the target function in which two entities cannot be distinguished by the given coverage metric. In the case of line coverage, a portion is a statement line. For state-coverage, a portion is a concrete state of the verification target. We can also use regions as the portion in the simulation of dense-time systems. In this case, a portion can contain infinitely many concrete states. The target function is conceptually defined as the set of all portions. Coverage refers to that how much of the target function has appeared in a set of simulation traces.

In statement (2), we allow for the selection flexibility of various search strategies. Indeed, we have already implemented game-based, goal-oriented, and random strategies [29]. A subspace ψ of the verified state-space ϕ and a set $\bar{T}$ of selected transitions are fed into procedure `next()` to compute the new next-step state-space after transitions and time-progress in statement (3). In statement (5), coverage is numerically estimated as the ratio of the already-verified proportion of the whole target function. The infinite loop can continue until a risk state is reached, or until we feel that enough confidence has been established (coverage of the function has reached some specific *threshold*), or until we have reached the fixed point and finished the exhaustive search in statement (6).

4 Criteria for Good Coverage Metrics

A good coverage estimation should tell us what proportion of a target function has been covered. We can partition a *target function* into *portions* and use an estimation function $\epsilon()$ to numerically estimate coverage. Formally speaking, $\epsilon : F \mapsto \mathcal{R}^+$ where $\mathcal{R}^+$ is the set of nonnegative reals. The whole target function can then be estimated as $f = \sum_{p \in F} \epsilon(p)$, and the current covered proportion V of the target function, i.e., the verified subset of F, can be estimated as $v = \sum_{p \in V} \epsilon(p)$. In ACM, a portion represents a physical entity (i.e., a transition) of the automata, then coverage of that portion means that the physical entity has been used in some traces. In RCM and TCM, a portion represents a state subspace(i.e., a region), then the occurrence of any state in the portion along some simulation traces indicates that the portion has been covered.

After experimenting with various coverage metrics and their computation methods, we identified the following four criteria for effective numerical coverage metrics.

- **Accountability:** This ensures that each portion of the target function is accounted for once and only once. If accountability is not maintained, we may run into two bizarre situations. First, some portions may not be accounted for and thus engineers simply cannot trust the metrics to check if all function portions have been covered. Second, it may happen that some portions are counted more than once and thus full coverage estimation is greater than 100% which makes no sense at all. Thus, accountability is the most important criterion.
- **Coverability:** This means that $\sum_{p \in V} \epsilon(p) = \sum_{p \in F} \epsilon(p)$ can be expected at the end of a symbolic simulation if enough traces have been generated. This is desirable in that 100% coverage can be the goal for verification. Moreover, if engineers decide to stop verification at 80% coverage, they can roughly estimate confidence in their products.

- **Efficiency:** This criterion measures the overhead in the computation of both the f (at statement (1)) and the v (at statement (4)) in procedure `Symbolic_Simulate`(). If complex formal reachability analysis is used to compute these two estimations, it is not worthwhile to estimate the coverage. In this work, we base our coverage estimation on transition-countings and state-space abstraction techniques and can efficiently calculate estimations in our three metrics.
- **Discernment:** This criterion assesses the capability of a metric to discern risk states. This can be an issue when, in some metrics, risk states and non-risk states are likely to fall in the same *portion*. A metric that frequently fails to detect existing risk states at a high numerical coverage may give users unjustified and false confidence on their system designs.

The third and fourth criteria appear contradictory. In a lot of cases, in order to discern risk states, we not only have to intelligently partition the portions, but also partition them in great resolution. And this usually results in high complexity and low efficiency to reach high coverage with enough traces through the huge space of portions.

In the following, we use these four criteria to evaluate the coverage metrics presented in the next few sections.

5 TA Arc Coverage Metric (ACM)

This is a straightforward adaptation from the technology of VLSI simulation and testing. In the computation of FSM arc coverage for VLSI, we conceptually transform a circuit to a finite-state automaton (FSM) and use the set of already-triggered transitions as V and the set of executable transitions as F to compute coverage estimation[5, 24]. The same definition of FSM arc coverage can be readily copied for the simulation of timed automata (TA). That is, we can also use the arcs of TAs to estimate coverage in the *TA arc coverage metric (ACM)*. Each portion corresponds to an executable transition and the estimation function $\epsilon_{ACM}()$ maps every portion to 1. The numerical estimation f of the whole target function in statement (1) of procedure Symbolic_Simulate() can be $|T|$, where T is the set of transitions in the TA.

Lemma 1. *ACM for dense-time systems satisfies the accountability criterion.*
Proof : It is true since $\epsilon_{ACM}(e) = 1$ for every executable transition in F, and we directly use the sizes of already-triggered and executable transition sets to calcualte the coverage. ||

The criterion of full coverability is not guaranteed. But as can be seen from our experiment data in section 8, with a tight estimation of the set of executable transitions, it is possible to get very close to 100% coverage. As for the criterion of efficiency, in each iteration, the overhead is a set-union operation and a size calculation of a set with high efficiency. Finally, ACM may not have much discernment since a transition can very often be used in both a safe trace and a trace that ends in a risk state. This means ACM may reach 100% coverage without discovering the risk state even if it exists.

6 Back-and-Forth Region Coverage Metric (RCM)

ACM can very often be too coarse to discern risk states. Another extreme that can also be adapted from VLSI verification technology is the *visited-state* coverage metric[5, 24], which uses the reachable state set in FSM to estimate coverage. The challenge to incorporate the concept into our framework arises from the fact that in VLSI's model, the states are discrete and countable while in timed automata, the states are dense and uncountable. A solution is to use equivalence classes in the dense-time state-space as portions. An equivalence relation to partition dense-time state-space is the *region*-equivalence relation between states[1]. In this paper, a region is a *minimal* state-space characterized by a mode and clock-difference constraints in the form of $x - x' \sim c$ where x, x' are two dense-time clocks, $\sim \in \{<, \leq, =, \geq, >\}$, and c is an integer in the range of $[-C_{A:\eta}, C_{A:\eta}]$ where $C_{A:\eta}$ is the biggest timing constant used in A and the safety state predicate η. In this way, we consider in this section the concept of *region coverage metric (RCM)*, in which a basic portion is a region, for the simulation of real-time systems. This coverage metric can have extra leverage with symbolic simulation since, in one step, we may generate a huge proportion of the state-space represented by a set of symbolic constraints.

There are three challenges in the implementation of RCM. First, how do we construct a tight estimation relevant to the reachability of the states? Second, how do we compute the coverage estimations of sets of portions, i.e., $\sum_{p \in V} \epsilon_{RCM}(p)$ and $\sum_{p \in F} \epsilon_{RCM}(p)$? Third, how do we maintain the accountability of the metrics? For discernment, a natural choice of a portion is the region presented in [1]. But it is very expensive (PSPACE-hard) to compute a precise representation of the entire reachable region set. A naive solution to this challenge is to use symbolic techniques with the popular data-structure of DBM (difference-bounded matrix)[11]. The problem is that DBMs are not necessarily disjoint from one another. If we represent a concave state-space as a set of DBMs and sum up the DBM proportion estimations to calculate the total estimation, it is likely that some portions will be counted more than once. In this section, we counter these three challenges in the following three steps. Finally, in subsection 6.4, we sum up these three steps to compute the RCM estimation.

6.1 For Challenge I: Tight Estimation of the Target Function

We use the space-intersection of abstractions of the backward and the forward reachable state-spaces to compute a tight estimation for the whole target function in RCM in the following procedure.

Compute F as the untimed quotient structure
of the state-space of A from initial states.
Compute B as the magnitude quotient structure
of the backward reachable state-space of A from risk states in $\neg\eta$.
Let $F := F \wedge B$;

A quotient structure is a smaller state-space, which keeps the latest information needed in model-checking. In the second statement, we employ an abstraction technique, called *magnitude abstraction*, to compute the weakest preconditions from a state-predicate. A magnitude abstraction of a state-predicate eliminates from the state-predicate all clock inequalities like $x - x' \sim c$ where x, x' are not zeros.

With these three steps, we have constrained F to a much smaller state-space that is relevant to the reachability from initial states to risk states. According to our experiments reported in section 8, this technique has brought our ultimate estimation in RCM very close to 100% and resulted in good coverability.

6.2 For Challenge II: Coverage Estimation of a Set of Portions

We shall work on the level of *zones*. A zone is a region-set characterizable by a range constraint on the `mode` variable and a set of range constraints on the clock differences. Conveniently speaking, the characterization can be represented as a pair like (Q', K) such that

- $Q' \subseteq Q$ and is the range of the `mode` variable; and
- K is a set of range constraints like $c \sim x - x' \sim' c'$ for clock differences, where $\sim, \sim' \in \{<, \leq\}$ and $c, c' \in [-C_{A:\eta}, C_{A:\eta}] \cup \{-\infty, \infty\}$.

For the efficiency of coverage estimation, we intuitively compute something like a *normalized volume* estimation of zones. The volume estimation of a rectangular polyhedron in a multi-dimensional space can be computed as the multiplication of its length in each dimension. For efficiency, we intuitively interpret a zone as a rectangular polyhedron in a space of $1 + |X|(1 + |X|)$ dimensions. The range of variable `mode`'s value spans the first dimension while $x_i - x_j$, for each $x_i, x_j \in X \cup \{0\}$ with $i < j$, spans a dimension. This intuitive and simplistic interpretation of zones neglects the fact that constraints on clock differences are not independent of one another. But our experiments show that it helps us design an efficient and coverable metric for region coverage.

In measuring the length of a clock difference constraint, we partition the real number lines into the following $4C_{A:\eta} + 3$ basic intervals

$$(-\infty, -C_{A:\eta})[-C_{A:\eta}, -C_{A:\eta}] \ldots [-1, -1](-1, 0)[0, 0](0, 1)[1, 1](1, 2) \ldots [C_{A:\eta}, C_{A:\eta}](C_{A:\eta}, \infty)$$

and use the number of basic intervals covered by the clock difference constraint for the *estimated length*. For example, $-3 \leq x - x' < 2$ has length 10 because it covers $[-3, -3], (-3, -2), \ldots, [1, 1], (1, 2)$.

Accordingly, the estimated normalized volume of a zone (Q', K) is

$$\frac{|Q'|}{|Q|} \cdot \Pi_{\text{``}c \sim x - x' \sim' c'\text{''} \in K} \frac{(\text{the estimated length of } c \leq x - x' \leq c')}{4C_{A:\eta} + 3}$$

6.3 For Challenge III: Coverage Estimation as a Set of Disjoint Zones

Although the technique in the last subsection allows us to come up with a coverage estimation of a zone, the zones may intersect with one another and accountability may not be maintained. In this subsection, we present a representation

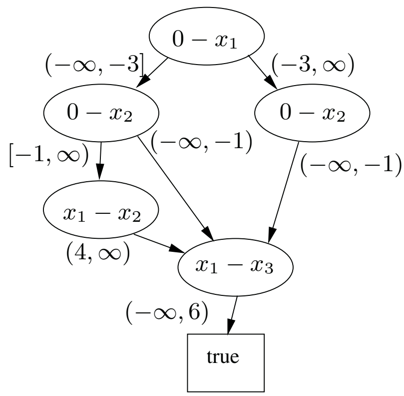


Fig. 1. CDD for $(0-x_1 \leq -3 \wedge x_1-x_3 < -4 \wedge x_2-x_1 < 6) \vee (0-x_2 < -1 \wedge x_2-x_1 < 6)$

for dense-time state-spaces such that zones represented are disjoint from one another. The representation that we have found with this property is CDD[6] with all zones in their closure forms (or all-pair shortest-path form). CDD is a BDD-like data-structure whose variables are clock differences like $x - x'$ and whose outgoing arcs from variables are disjoint value ranges. For example, the CDD for the state-space of $(0-x_1 \leq -3 \wedge x_1-x_3 < -4 \wedge x_2-x_1 < 6) \vee (0-x_2 < -1 \wedge x_2 - x_1 < 6)$ without terminal *false* is in Fig. 1. Each path in this figure represents a zone in closure form. We can prove the following lemma.

Lemma 2. *Given a CDD with all zones in their closure forms, then each two paths in the CDD represent two disjoint zones.*

Proof : From root to the terminals of the two paths, there is a branching node from which the two paths break away. The corresponding outgoing arcs from the node for the two paths are labeled with disjoint intervals. With the tightness of the zone constraints, this means that the zones of the two paths are disjoint. ||

For the convenience of presentation, we can assume that a CDD, say D, is represented as a *true*, or a *false*, or a tuple like $D = (x - x', (\lambda_1, D_1), \ldots, (\lambda_n, D_n))$, such that

- the root node of D is labeled with clock difference variable $x - x'$.
- $\lambda_1, \ldots, \lambda_n$ are disjoint intervals whose endpoints are in $\{-\infty, -C_{A:\eta}, \ldots, -1, 0, 1, \ldots, C_{A:\eta}, \infty\}$.
- for each $1 \leq i \leq n$, the arc labeled with interval λ_i points to D_i.

With the desirable feature of CDD, we can design a symbolic procedure, `NVolume`(D) in Table 1, to compute the estimated volume of a state-space represented by CDD, say D, with all zones in closure forms. Another advantage of this symbolic procedure is that we can take advantage of the data-sharing in CDD to avoid explicit enumeration of all disjoint zones. The normalized volume estimation of a substructure in a CDD can be saved in R and used for the estimation of other zones that use this same substructure.

Table 1. Symbolic procedure for normalized volume estimation

set R;

`NVolume`(D) { let $R := \emptyset$; return `rec_NVolume`(D); }

`rec_NVolume`(D) /* D is *true*, *false*, or $(x - x', (\lambda_1, D_1), \ldots, (\lambda_n, D_n))$ */ {
    if D is *true*, return 1; else if D is *false*, return 0;
    else if $(D, r) \in R$ for some r, return r;
    else if D is $(x - x', (\lambda_1, D_1), \ldots, (\lambda_n, D_n))$, then {
        let $r := \sum_{1 \le i \le n} \frac{\text{the length of } \lambda_i}{4C_{A:\eta}+3}$ `rec_NVolume`(D_i);
        let $R := R \cup \{(D, r)\}$; return r;
    }
}

6.4 Estimation of the Region Coverage

In our framework, both V and F in `Symbolic_Simulate`() on page 163 are conceptually represented as a set of pairs like (Q', D), for a state-space, where D is a CDD with all zones in closure form, with the following constraints.

- For each two pairs $(Q'_1, D_1), (Q'_2, D_2)$ in the set, $Q'_1 \cap Q'_2 = \emptyset$.
- For each pair (Q', D) in the set, D represents the zones of all states with their modes in Q'.

Then at statement (4) of each iteration of procedure `Symbolic_Simulate`() on page 163, the estimated normalized volume v for V is $\sum_{(Q',D)\in V} \frac{|Q'|}{|Q|} \cdot \mathtt{NVolume}(D)$ and $\frac{v}{f} = \frac{\sum_{(Q',D)\in V} \frac{|Q'|}{|Q|} \cdot \mathtt{NVolume}(D)}{\sum_{(Q',D)\in F} \frac{|Q'|}{|Q|} \cdot \mathtt{NVolume}(D)}$.

Lemma 3. *RCM satisfies the criterion of accountability.*
Proof : According to lemma 2, zones respectively represented by paths in a CDD in its closure form are disjoint from one another. Thus in the algorithm of `NVolume`(), we count each portion once and only once and RCM satisfies the criterion of accountability. ‖

Lemma 4. *RCM satisifies the criterion of discernment, and it is impossible to reach 100% coverage without detecting the risk state, if any.*
Proof : Given a timed automaton A and a risk predicate η, in RCM, safe states and unsafe states are not in the same portion. This is because symbolic manipulations of zones are sufficient to answer the reachability problem of timed automata[1]. ‖

According to lemma 4, RCM has enough discerning power to discover reachable risk states whereas ACM lacks such discerning power.

7 Triggering-Condition Coverage Metric (TCM)

RCM has the advantage in accountability and discernment. But it may result in low coverability since our estimation of the reachable region sets can be imprecise. On the other hand, ACM can suffer from low discernment. In this section, we propose a balanced approach called *triggering-condition coverage metric (TCM)*, in which we use the triggering conditions of all transitions as the body of the whole target function. TCM estimates the proportion of the covered triggering conditions of all transitions. It is accounted as the summation of triggering condition coverage for each transition. Formally speaking, a basic portion in TCM is a pair like (e, γ) where e is an executable transition and γ is a region in subspace $\tau(e)$ (the triggering condition of e).

The numerical estimation f of the whole target function in procedure `Symbolic_Simulate`() can be computed as $\sum_{e \in T} \mathtt{NVolume}(\tau(e))$. We use $|T|$ variables, V_e for each $e \in T$, to record the verified proportion of the triggering condition of each transition. Initially, $\forall e \in T, V_e = \mathit{false}$. At each iteration, when statement (4) is executed, we use the following procedure to compute the value of v.

for each $e \in T$, $V_e := V_e \vee (\mathtt{abstract}_e(\phi \wedge \tau(e)))$;
let $v := \sum_{e \in T} \mathtt{NVolume}(V_e)$;

Here for the sake of efficiency, we use an abstract function $\mathtt{abstract}_e(d)$ to eliminate the recording of all clock difference variables not used in $\tau(e)$. For example, if $\tau(e) = 0 - x < -5 \wedge x - y \leq 3$, then $\mathtt{abstract}_e(0 - x < -7 \wedge x - y \leq -2 \wedge y - 0 \leq 2) = 0 - x < -7 \wedge x - y \leq -2$. It can be shown that TCM has the following desirable property.

Lemma 5. *TCM satisfies the criterion of accountability.*
Proof : Since zones of the triggering conditions of each transition represented by a CDD are disjoint, TCM satisfies the criterion of accountability. ||

TCM is more efficient than the RCM since it is based on abstraction of zones whose representation complexity is usually lower. In the following experiments, we shall see that it satisfies the criterion of coverability without sacrificing its discernment.

8 Experiments with Bluetooth L2CAP

We have implemented our numerical coverage estimation techniques in our model-checker/simulator `red` 4.1. To check the possibility of using our techniques in real-world projects, we have experimented with the *Logical Link Control and Adaptation Layer Protocol(L2CAP)* of Bluetooth specification[15]. The wireless communication standard of Bluetooth has been widely discussed and adopted in many appliances since it was published. L2CAP is layered over the Baseband Protocol and resides in the data link layer of Bluetooth. This protocol supports message multiplexing, packet segmentation and reassembly, and the conveying of quality of service information to the upper protocol layer. The protocol regulates the behavior between a master device and a slave device.

In our experiment, we collect coverage and performance data for L2CAP models both with and without design faults against various trace-generation strategies. In subsection 8.2, we report the coverage data of ACM, RCM, and

TCM for the L2CAP model without faults. In subsection 8.3, we create six versions of the L2CAP model, each with an inserted fault, and report how the coverage metrics help us discern the faults before 100% coverage is reached. Data is collected on a Pentium 4 with 1.7GHz, 256MB, running Red Hat Linux 7.0.

8.1 Modelling L2CAP

The L2CAP defines the actions performed by a master and a slave. A master is a device issuing a request while a slave is the one responding to the master's request. A message sequence chart (MSC) that may better illustrate a typical scenario of event sequence in L2CAP can be found in [15]. The scenario starts when the master's upper layer issues an `L2CA_ConnectReq` (Connection Request) through the L2CA interface. The protocol goes on with messages bouncing back and forth until the master sends an `L2CAP_ConfigRsp` message to the slave. Then both parties can start exchanging data. Finally the master's upper layer issues message `L2CA_DisconnectReq` to close the connection and the slave confirms the disconnection.

We use nine processes to model the entire activity in L2CAP. They are the master's upper layer, the master's L2CAP layer, master's L2CAP time-out process, master's L2CAP extended time-out process, the slave's upper layer, the slave's L2CAP layer, slave's L2CAP time-out process, slave's L2CAP extended time-out process, and the unreliable network. Each of these processes is described as a communicating timed automaton. The safety condition is that when the master's L2CAP layer stays in the OPEN state, the slave's L2CAP layer can not enter the state W4_L2CA_DISCONNECT_RSP.

8.2 Coverage Estimation when There Is No Fault

In this subsection, we execute procedure `Symbolic_Simulate`() with *breadth-first* strategy to verify our L2CAP model without faults. That is, each time we execute statement (2) in procedure `Symbolic_Simulate`(), we let $\bar{T} = T$ and $\phi = \psi$.

In each iteration, we calculate three estimations according to the three coverage metrics respectively. The data is in Table 2. After 18 iterations, `red` 4.1 finishes the exhaustive search, and reports that the risk state is NOT reachable. It costs total cpu time 37.14 sec and memory usage 782k with ACM; total cpu time 30.59 sec and memory usage 632k with RCM; total CPU time 35.79 sec and memory usage 722k with TCM.

ACM and TCM can both reach 100% coverage estimation while RCM gets very close to 100%. The data shows that our methods have very high coverability in the experiment.

Another interesting thing is that for this correct L2CAP model, ACM and TCM can give us 100% confidence in their respective metrics before the whole reachable state-space representation is constructed. More precisely, according to ACM and TCM, we can stop at iteration 13 with 100% confidence. On the other hand, if we use straightforward formal verification, then we have to run

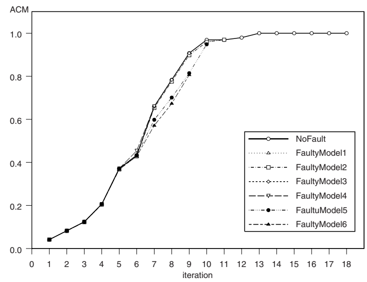
Table 2. Coverage estimations and overheads with respect to iterations when there are no bugs

iteration	ACM	ACM time overhead	RCM	RCM time overhead	TCM	TCM time overhead
1	4/97	0.00sec.	0.167816	7.39sec.	0.004092	0.02sec.
2	8/97	0.00sec.	0.173442	7.39sec.	0.382901	0.02sec.
3	12/97	0.00sec.	0.174279	7.40sec.	0.783131	0.02sec.
4	20/97	0.01sec.	0.175273	7.41sec.	0.799498	0.02sec.
5	36/97	0.02sec.	0.232154	7.41sec.	0.813138	0.03sec.
6	42/97	0.03sec.	0.295386	7.41sec.	0.815525	0.04sec.
7	64/97	0.05sec.	0.408160	7.42sec.	0.884971	0.06sec.
8	76/97	0.08sec.	0.561395	7.43sec.	0.920890	0.08sec.
9	88/97	0.12sec.	0.956820	7.44sec.	0.971241	0.11sec.
10	94/97	0.17sec.	0.965724	7.45sec.	0.975507	0.15sec.
11	94/97	0.22sec.	0.974428	7.46sec.	0.975507	0.18sec.
12	95/97	0.28sec.	0.975538	7.48sec.	0.976530	0.22sec.
13	97/97	0.34sec.	0.975783	7.49sec.	1.000000	0.26sec.
14	97/97	0.40sec.	0.981319	7.50sec.	1.000000	0.29sec.
15	97/97	0.47sec.	0.981338	7.52sec.	1.000000	0.33sec.
16	97/97	0.55sec.	0.982733	7.54sec.	1.000000	0.36sec.
17	97/97	0.63sec.	0.982734	7.56sec.	1.000000	0.40sec.
18	97/97	0.70sec.	0.982734	7.57sec.	1.000000	0.44sec.

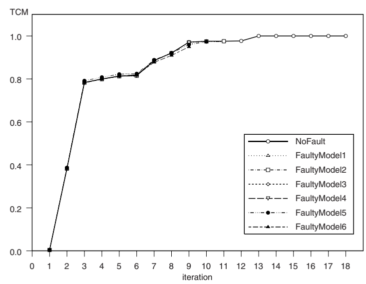
through all the 18 iterations before we can conclude that the model is fault-free. This observation suggests that symbolic simulation with our coverage metrics can greatly save verification costs.

Since RCM gets us very close to 100% coverage, we can use 100% coverage as a goal for verification in RCM. More importantly, RCM is a better alternative in discernment than ACM and TCM. For one thing, at the 17th iteration, it could still increase to reflect more portions that have been traced through while ACM and TCM have already converged to 1.

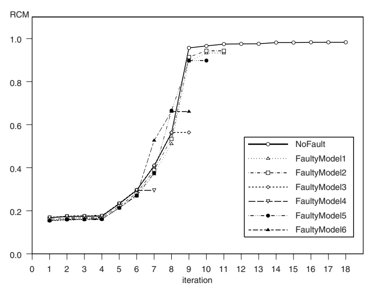
As for the efficiency of our coverage estimation methods, the overhead incurred in the coverage estimation respectively for ACM, TCM and RCM is 1.885%, 1.229%, and 24.747% of the verification CPU time. This means that our implementation for both ACM and TCM are quite efficient. In Fig. 2, we drew the growth curves of coverage estimations for the correct model and six faulty models (details in the next subsection) with respect to the iterations. Notice that both coverage metrics grow quickly between the 4th iteration and the 10th iteration and then become flattened out to convergy to 100%. It reaches 100% at the 13th iteration and finishes the exhaustive search at the 18th iteration in the correct case. For all the faulty models, we reach the risk state before the 11th iteration. The curves show that both metrics may give engineers enough confidence to make decision quickly. For example, they may stop the simulation, while the coverage estimation is greater than a predetermined coverage threshold or the coverage estimation curve becomes flattened, and save the verification resources.



(a) ACM



(b) TCM



(c) RCM

Fig. 2. Growth of coverage with respect to iterations

A detail breakdown of the computation time shows that most of the overhead for RCM is consumed in the normalized volume calculation with the much larger CDD structure. This is the price of better discernment.

Figures 2 shows the growth of coverage estimation in RCM for the same models. It is interesting that the curves dramatically increase in the first few iterations and then slow down in the latter iterations. We can also detect the faults before the 11th iteration while the coverage estimation stops to increase in the 17th iteration and the exhaustive search finishes at iteration 18. As can be seen, although RCM does not satisfy the coverability in the experiment, its curves do converge very close to 100% and flatten quickly.

Table 3. Coverage estimations with respect to two strategies for the 6 faulty models

Strategy	Faulty Models	Depth	ACM	RCM	TCM	Risk state reached?
Depth First	1	66	71/98	0.355262	0.958950	Yes
	2	64	71/98	0.354993	0.958950	Yes
	3	26	27/97	0.293884	0.437878	Yes
	4	96	90/97	0.924978	0.963982	Yes
	5	64	63/97	0.355688	0.948209	Yes
	6	62	61/98	0.357885	0.936412	Yes
Breath First	1	11	95/98	0.932724	0.975532	Yes
	2	11	95/98	0.943274	0.975532	Yes
	3	9	88/97	0.564228	0.971241	Yes
	4	7	64/97	0.294859	0.884971	Yes
	5	10	92/97	0.898077	0.973172	Yes
	6	9	79/98	0.660754	0.949243	Yes

8.3 Coverage Estimation when There Is a Fault

We design six L2CAP faulty models, each with an inserted fault. For convenience, we label these six faulty models with indices 1 through 6. In each faulty model, we change master or slave's behaviors and let the risk condition become reachable. We tried two trace-generation strategies. The first is breadth-first (see subsection 8.2). The second is *depth-first*. That is, at each time when we execute statement (2), we choose $\bar{T}$ to be of size 1 and only choose to fire one transition in T. We also keep a stack so that we can backtrack to previous iterations to choose an alternative transitions at statement (2). The coverage data is shown in Table 3. The most interesting thing in Table 3 is that the faults are all detected before we reach 100% coverage. This means that our three coverage metrics have enough discernment for the six faulty models.

9 Conclusion

Symbolic simulation combines the advantages of both simulation and formal verification and can be an important verification approach before fully automatic formal verification becomes applicable. In this paper, we present techniques for coverage estimation for dense-time systems. We hope such techniques can be the solid stepstone toward the development of powerful symbolic simulators for industry real-time systems. Many issues raised in this work also deserve future research. For example, it will be interesting to see the design of quantitative metrics for our criterion of discernment in the symbolic simulation of dense-time systems. With such metrics, the criterion becomes equivalent to the notion of observability[10].

References

1. R. Alur, C. Courcoubetis, D.L. Dill. Model Checking for Real-Time Systems, IEEE LICS, 1990.
2. R. Alur, D.L. Dill. Automata for modelling real-time systems. ICALP' 1990, LNCS 443, Springer-Verlag, pp. 322-335.
3. J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond, IEEE LICS, 1990.
4. B.Beizer. Software Testing Techniques. Van Nostrand Rheinhold, New York, 2nd ed., 1990.
5. L. Bening, H. Foster. Principles of Verifiable RTL Design, a Functional Coding Style Supporting Verification Processes in Verilog, 2nd ed., Kluwer Academic Publishers, 2001.
6. G. Behrmann, K.G. Larsen, J. Pearson, C. Weise, Wang Yi. Efficient Timed Reachability Analysis Using Clock Difference Diagrams. CAV'99, July, Trento, Italy, LNCS 1633, Springer-Verlag.
7. R.E. Bryant. Graph-based Algorithms for Boolean Function Manipulation, IEEE Trans. Comput., C-35(8), 1986.
8. K-T. Cheng and A. S. Krishnakumar. Automatic Functional Test Generation Using the Extended Finite State Machine Model. In Proceedings of the 30th Design Automation Conference, pp. 86-91, June 1993.
9. Hana Chockler, Orna Kupferman, Moshe Y. Vardi. Coverage Metrics for Temporal Logic Model Checking. LNCS 2031, pp. 528-552, 2001.
10. S. Devadas, A. Ghosh, K. Keutzer. An Obervability-Based Code Coverage Metric for Functional Simulation. IEEE ICCAD'96.
11. D.L. Dill. Timing Assumptions and Verification of Finite-state Concurrent Systems. CAV'89, LNCS 407, Springer-Verlag.
12. D.L. Dill. What's Between Simulation and Formal Verification? In Proceedings of DAC1998.
13. Farzan Fallah, Srinivas Devadas, Kurt Keutzer. OCCOM: Efficient Computation of Observability-Based Code Coverage Metrics for Functional Verification. Design Automation Conference, pp. 152-157, 1998.
14. D. Geist, M. Farkas, A. Landver, Y.Lichtenstein, S. Ur, Y. Wolsfthal. Coverage-Directed Test Generation Using Symbolic Techniques. In Proceedings of the Int'l Conference on Formal Methods in CAD, November 1996.
15. J. Haartsen. Bluetooth Specification, version 1.0. http://www.bluetooth.com/
16. T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-Time Systems, IEEE LICS 1992.
17. C.A.R. Hoare. Communicating Sequential Processes, Prentice Hall, 1985.
18. 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.
19. Y. Hoskote, D. Moundanos, and J. A. Abraham. Automatic Extraction of the control flow machine and application to evaluating coverage of verification vectors. In Proceedings of the Int'l Conference on Computer Design, pp 532-537, October 1995.
20. Y. Hoskote, T. Kam, P.-H. Ho, and X. Zhao. Coverage estimation for symbolic model checking. In Proceedings of DAC1999, pp. 300-305.

21. Kim G. Larsen, Paul Pettersson and Wang Yi. Model-Checking for Real-Time Systems. In Proceedings of the 10th International Conference on Fundamentals of Computation Theory, Dresden, Germany, August 22-25, 1995. LNCS 965, pp. 62-88, Horst Reichel.
22. B. Mike, G. Daniel, H. Alan, W. Yaron, M. Gerard, S. Ralph. A Study in Coverage-Driven Test Generation. In Proceedings of DAC1999, pp. 970-975.
23. P. Pettersson, K.G. Larsen. UPPAAL2k. In Bulletin of the European Association for Theoretical Computer Science, Vol. 70, pp. 40-44, 2000.
24. P. Rashinkar, P. Paterson, L. Singh. System-on-a-chip Verification, Methodology and Techniques. Kluwer Academic Publishers, 2001.
25. C.-J.H. Seger, R.E. Bryant. Formal Verification by Symbolic Evaluation of Partially-Ordered Trajectories. Formal Methods in System Designs, Vol. 6, No. 2, pp. 147-189, Mar. 1995.
26. F. Wang. Efficient Data-Structure for Fully Symbolic Verification of Real-Time Software Systems. TACAS'2000, March, Berlin, Germany. LNCS 1785, Springer-Verlag.
27. F. Wang. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram, to appear in Proceedings of FORTE, August 2001, Cheju Island, Korea.
28. F. Wang. Efficient Verification of Timed Automata with BDD-like Data-Structures. to appear in proceedings of VMCAI 2003, New York City, USA, Jan. 2003. LNCS 2575, Springer-Verlag.
29. F. Wang, G-D. Hwang, F. Yu. Symbolic Simulation of Real-Time Concurrent Systems to appear in proceedings of RTCSA 2003, Tainan, Taiwan. Feb. 2003. LNCS, Springer-Verlag;
30. S. Yovine. Kronos:A Verification Tool for Real-Time Systems. International Journal of Software Tools for Technology Transfer, Vol. 1, Nr. 1/2, October 1997.