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

Farn Wang
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
Tools available at: http://val.iis.sinica.edu.tw/

Geng-Dian Hwang, Fang Yu
Institute of Information Science, Academia Sinica
Taipei, Taiwan 115, Republic of China

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 w.r.t. four criteria. Implementation and experiments are then reported.

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

1 Introduction

Nowadays, 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 give us enough trace coverage to obtain enough confidence in system designs. On the other hand, the application of the new technology of formal verification is still hampered by its intrinsic complexity. In the forseeable future, we expect that simulation and formal verification will be combined for the verification of large-scale real-time systems. Symbolic simulation is such a combination[23]. It uses the symbolic techniques[3, 7, 11, 18] of formal verification to represent space of simulation traces so that abstract (rather than concrete) behaviors can be observed in a trace. For the verification of real-time systems, tools like UPPAAL[21] and RED[24, 25, 26] support symbolic simulations.

[*]The work is partially supported by NSC, Taiwan, ROC under grants NSC 90-2213-E-001-006, NSC 90-2213-E-001-035, and the by the Broadband network protocol verification project of Institute of Applied Science & Engineering Research, Academia Sinica, 2001.

The current symbolic simulation technology for real-time systems is still not as developed as the one for untimed systems like VLSI. For one thing, the important concept of *coverage* can be used both to estimate how good a set of traces is and to direct the generation of new traces. In short, coverage means how much of the *target function* to be verified has been verified. The importance of this concept is the following. In real-world projects, it is usually the case that we do not have enough resources to either run a large enough number of traces to obtain enough confidence or complete the formal verification task. Product designs usually need to be released before we can be 100% confident of the designs. Therefore it is important that we have some kind of metrics to evaluate the confidence on our designs. A common coverage metric for simulation is *code coverage*, which measures the proportion of already-executed HDL statements during simulation. *Transition coverage* is another metric for control state machines[27]. Such coverage metrics have been proved to be effective in reducing bug escapes by pointing out coverage holes in the test suite. Coverage goals are used to measure the degree of confidence in our total verification effort and to help the design team predict the optimal time for design release[4].

But such an important concept has not been properly explored in the simulation of dense real-time systems. In this work, we propose techniques to estimate numerical coverage for this purpose. We believe that our techniques can be used in the future to help the development of various coverage-based verification techniques, including the design of new coverage metrics, and coverage-based test-pattern generation, for real-time systems.

Before we can estimate the numerical coverage, we have to design a metric and its estimation procedure. This incurs 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 guide us in the designing of coverage metrics. The criteria are **accountability** (each basic *portion* of the *target function* is counted once and exactly once), **coverability** (100% coverage estimation is achievable as the goal), **efficiency** (the overhead in computing the coverage estimation is low), and **discernment** (the capability to discern risk states). According to these criteria, we adapt three coverage metrics from traditional testing research and develop techniques to implement them for real-time systems. These three new metrics are: *TA (timed automata) arc coverage (ACM)*, *triggering condition coverage (TCM)*, and *back-and-forth region coverage (RCM)*[22]. To maintain the four criteria for dense-time state-spaces of real-time systems, we have developed techniques to quantitatively estimate the volume of a state-space and to significantly prune state-space portions irrelevant to a verification task.

To demonstrate the usefulness of our techniques for real-world projects, in section 8, we have experimented to model and verify the *L2CAP (Logical Link Control and Adaptation Layer Protocol)* of Bluetooth specification[15], a wireless communication standard, which has been widely adopted in industry. We model two devices communicating with the L2CAP of Bluetooth and carry out simulation experiments to gather data of numerical coverage estimation. We then compare the three coverage metrics according to our experiment data and the four criteria.

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

2 Previous work

Coverage techniques have been widely discussed and applied in the 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. The programming code metrics measure syntactical characteristics of the code due to execution stimuli. For example, *line coverage metric* measures the number of times a statement is visited during the course of execution, *branch coverage* measures the number of times a segment of code diverges into a unique flow, and *path coverage* measures the number of times each path(i.e., a unique combination of branches and statements) is exercised due to its execution stimuli[5]. The number of paths in a program may be exponentially related to program size and makes reaching 100% path coverage hard in software testing.

Coverage analysis techniques proposed for general HDL modes include guaranteed coverage of every statement[8], transition coverage of a test set[16], and abstraction of models and semantic control over transition coverage[14]. Fallah provides OCCOM[13] to address the observability. Most of these HDL metrics are used to drive test-generation in simulation analysis.

Ho et al[17] 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 suggest several coverage metrics to measure how complete the verified specification is and to find the uncovered parts.

Simulation-based verification techniques are ways to bridge the gap between simulation and formal verification[12]. GOTCHA (Generator of Test Cases for Hardware Architecture) is a prototype coverage-driven test generator implemented as an extension to the Murϕ model-checker[20]. It supports state and transition coverage analysis in FSM. On completion of the enumeration of the entire reachable state space, a random coverage task is chosen from amongst those that have not yet been covered.

We apply the coverage techniques in our symbolic simulator with dense-time model. 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 straightforwardly copied for dense-time systems.

3 Framework of verification

3.1 System model

We use the widely accepted model of *timed automata (TA)*[1]. We assume familiarity with this model and will not go into much detail due to the page-limit. A (TA) is a finite-state automaton equipped with a finite set of clocks which can hold nonnegative real-values. At any moment, a TA can stay in only one *mode* (or *control location*). In its operation, one of the transitions can be triggered when the corresponding triggering condition is satisfied. Upon being triggered, the TA instantaneously transits from one mode to another and resets some clocks to zero. In between transitions, all clocks increase their readings at a uniform rate.

For convenience, given a set Q of modes and a set X of clocks, we use $B(Q, X)$ as the set of all Boolean combinations of mode predicate $\texttt{mode} = q$, where $\texttt{mode}$ is a special auxiliary variable, and inequalities of the forms $x - x' \sim c$, where $q \in Q$, $x, x' \in X \cup \{0\}$, "$\sim$" is one of $\leq, <, =, >, \geq$, and c is an integer constant.

Definition 1 **<u>timed automata (TA)</u>** A timed automaton A is given as a tuple $\langle X, Q, I, \mu, T, \tau, \pi \rangle$ with the following restrictions. X is the set of clocks. Q is the set of modes. $I \in B(Q, X)$ is the initial condition on clocks. $\mu : Q \mapsto B(\emptyset, X)$ defines the invariance condition of each mode. $T \subseteq Q \times Q$ is the set of transitions. $\tau : T \mapsto B(\emptyset, X)$ and $\pi : T \mapsto 2^X$ respectively define the triggering condition and the clock set to reset of each transition. $\|$

A *valuation* of a set is a mapping from the set to another set. Given an $\eta \in B(Q, X)$ and a valuation ν of X, we say ν *satisfies* η, in symbols $\nu \models \eta$, iff it is the case that when the variables in η are interpreted according to ν, η will be evaluated *true*.

Definition 2 **<u>states</u>** A state ν of $A = \langle X, Q, I, \mu, T, \tau, \pi \rangle$ is a valuation of $X \cup \{\texttt{mode}\}$ such that

- $\nu(\texttt{mode}) \in Q$ is the mode of A in ν; and
- for each $x \in X$, $\nu(x) \in \mathcal{R}^+$ such that $\mathcal{R}^+$ is the set of nonnegative real numbers and $\nu \models \mu(\nu(\texttt{mode}))$. $\|$

For any $t \in \mathcal{R}^+$, $\nu + t$ is a state identical to ν except that for every clock $x \in X$, $\nu(x) + t = (\nu + t)(x)$. Given $\bar{X} \subseteq X$, $\nu\bar{X}$ is a new state identical to ν except that for every $x \in \bar{X}$, $\nu\bar{X}(x) = 0$.

Definition 3 **<u>runs</u>** Given a timed automaton $A = \langle X, Q, I, \mu, T, \tau, \pi \rangle$, a *run* is an infinite sequence of state-time pair $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$ such that $\nu_0 \models I$ and $t_0 t_1 \ldots t_k \ldots\ldots$ is a monotonically increasing real-number (time) divergent sequence, and for all $k \geq 0$,

- for all $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \mu(\nu_k(\texttt{mode}))$; and
- either $\nu_k(\texttt{mode}) = \nu_{k+1}(\texttt{mode})$ and $\nu_k + (t_{k+1} - t_k) = \nu_{k+1}$; or
  - $(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode})) \in T$ and
  - $\nu_k + (t_{k+1} - t_k) \models \tau(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode}))$ and
  - $(\nu_k + (t_{k+1} - t_k))\pi(\nu_k(\texttt{mode}), \nu_{k+1}(\texttt{mode})) = \nu_{k+1}$. $\|$

3.2 Procedure of simulation and coverage analysis

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. Formally speaking, a safety analysis problem instance consists of a timed automata A and a *safety state-predicate* $\eta \in B(Q, X)$. A is *safe* w.r.t. η, in symbols $A \models \eta$, iff for all runs $(\nu_0, t_0)(\nu_1, t_1) \ldots (\nu_k, t_k) \ldots\ldots$, for all $k \geq 0$, and for all $t \in [0, t_{k+1} - t_k]$, $\nu_k + t \models \eta$, i.e., the safety requirement is guaranteed.

Our main procedure is constructed based on the well-discussed symbolic procedure, call `next()` here, to compute the symbolic post-condition after a discrete transition and time-progress[18]. Our symbolic simulation procedure takes the following form (with details on coverage estimation in statements (1), (4), and (5) to be filled in later).

```
Symbolic_Simulate($A$, $\eta$) /* $A$ is a TA; $\eta$ is the safety state predicate. */ {
  Compute the estimation $f$ of the whole *target function* coverage $F$.      (1)
  Let $\phi := I$; $\phi' :=$ *false*; $v := 0$;
  While $v/f < 100\%$ and $\phi \neq \phi'$, {                                  (2)
    Let $\phi' := \phi$;
    Select a subspace $\psi$ of $\phi$ and a set $D$ of transitions
      (possibly based on the value $\phi$ and $D$);                             (3)
    Compute $\phi := \phi \vee \bigvee_{e \in D}$ next($A, \psi, e$);
    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 out "a risk state is reachable" and exit;   (6)
  }
  Print "At 100% coverage of the chosen metric, no risk states are found."
}
```

The *target function* of the verification can be the set of TA transitions (arcs), the regions in the reachable state-space, or the triggering-conditions of all transitions. A subspace ψ of the verified state-space ϕ and a set D of selected transitions are fed to procedure `next()` to compute the new next-step state-space after transitions and time-progress. The loop at statement (2) can continue until we have reached 100% coverage, or we feel that enough confidence (coverage of the function) has been established, or we have found that a risk state is reachable at statement (6). In statement (5), the coverage is numerically estimated as the ratio of the already-verified proportion of the whole target function.

In statement (3), we allow for the flexibility of various search strategies. Indeed, we have implemented some strategies like game-based strategy, goal-oriented strategy, and random strategy. But this simulation framework is actually more general than simulation. For example, if in every iteration, we choose $\psi = \phi$ and $D = T$, then the whole procedure becomes a forward reachability analyzer. In the next few sections, we shall discuss how to compute the coverage estimations according to our three coverage metrics. As for the research of using various strategies to guide the search, we believe it deserves more effort in the future.

4 Criteria for good coverage metrics

Supposedly, a good coverage estimation can tell us how much proportion of the target function has been covered. Conceptually, we can partition the *target function* into *portions* and use an estimation function $\epsilon()$, from the set of portions to the set of nonnegative reals, to numerically estimate the coverage. Let the set of all portions be F. Formally speaking, $\epsilon : F \mapsto \mathcal{R}^+$ where $\mathcal{R}^+$ is the set of nonnegative reals. Then the whole target function can 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 the case that a portion represents a physical entity (e.g. a transition, or a branch) of the automata, then the coverage of the portion means that such a physical entity has been used along some traces. In the case that a portion represents a state subspace, like a region, then the occurrence of any state in the portion along some simulation traces means the portion has been covered.

But in practice, it can be difficult to design a good metric for dense-time systems. For example, we may want to use visited states as portions. Then in a dense-time state-space, we have to decide how a state should correspond to a portion. For discernment, a natural choice of a portion is the region presented in [1]. But it is very expensive (PSPACE-complete) to compute a precise representation of the whole 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]. Then the challenge is that DBMs are not necessarily disjoint from one another. If we sum up the portion estimations of each state-space characterizable by a DBM to calculate the total estimation, then it is likely that some portions will be counted more than once.

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

- **accountability:** Intuitively, this means that each portion of the target function is accounted for once and exactly once. If accountability is not maintained, then we may run into the following two bizarre situations. First, it may happen that some portions are not accounted for and the engineers simply cannot trust the metrics to check if all function portions have been covered. Second, it may happen that some portions are accounted for more than once and the full coverage estimation is greater than 100% and makes no sense at all. Thus this criteria of accountability is the most important. If we are going to use state-space or its abstraction to estimate the coverage for target functions in dense-time systems, then we have to develop new techniques, other than DBMs, to make sure each portion is accounted for exactly once.
- **coverability:** In notations, this means that at the end of the symbolic simu-

lation, it can be expected that $\sum_{p\in V}\epsilon(p)=\sum_{p\in F}\epsilon(p)$ if enough simulation iterations have been carried out. This is desirable in that the engineers can look to the 100% coverage as a goal for verification. Or when they decide to stop verification at coverage 80%, they roughly know how confident they can be of their products. On the other hand, it is likely to leave verification engineers clueless if the coverage estimation seems converging to a small percentage number no matter how many traces they have generated. The 100% coverage can only be achieved if we have a precise numerical estimation of the whole target function F to be verified.

- **efficiency:** The third criterion is the efficiency in the computation of both the f (at statement (1)) and the v (at statement (4)) in procedure `Symbolic_Simulate`(). If it takes the complexity of formal reachability analysis to compute these two estimations, then 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:** The last, but not the least, is the capability of a metric to discern risk states from others. This can be an issue when in the metric, risk states and non-risk states are likely to fall in the same portion. A metric that frequently fails to detect the existing risk states at high numerical coverage may give the users unjustified and false confidence on their system designs.

The third and fourth criteria are kind of contradictory to each other. In a lot of cases, in order to discern risk states, we not only have to partition the portions smartly, but also have to 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 shall 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 automata (FSM) and use the set of already-triggered transitions as V and the set of executable transitions as F to compute coverage estimation[5, 22]. The same definition of FSM arc coverage can readily be copied for the simulation of timed automata (TA). That is, we can also use the arcs of TAs to estimate coverage according to the *TA arc coverage metric (ACM).* The straightforwardness of this metric has many desirable features. Each portion corresponds to an executable transition and the estimation function $\epsilon_{ACM}()$ maps everything to 1. The numerical estimation f of the whole target function in statement (1) of procedure Symbolic_Simulate() can be $|T|$, the number of transitions in the TA. But it can be much preciser and tighter. In our implementation, we actually compute an untimed quotient structure of A's state-space through forward analysis and eliminate those transitions which are actually not triggerible. In this way, we can usually come up with a much smaller bound on f, the number of executable transitions in ACM.

As for the computation of the numerical coverage estimation v, we can use V a static set variable of transitions such that $V = \emptyset$ initially and each time when statement (4) in procedure `Symbolic_Simulate`() is executed, we perform the following two steps.

$V := V \cup D$; $v := |V|$;

LEMMA 1 *ACM for dense-time systems satisfies the accountability criteria.*
Proof : True since $\epsilon_{ACM}(e) = 1$ for every executable transition in F. ||

The criteria 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% of coverage. As for the criteria of efficiency, in each iteration, the overhead is a set-union operation and a size calculation of set and the efficiency is high. Finally, ACM may not have very much of discernment. A transition can very often be used in both a safe trace and a trace that ends in a risk state.

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 straightforwardly adapted from VLSI verification technology is the *visited-state* coverage metric[5, 22], 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 short, a region is a *minimal* state-space characterizable 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 states-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 unsafe 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 ? In this section, we counter these three challenges in three steps. For the first challenge, we use the intersection of abstractions of both backward and forward reachable state-space representations to construct a tight estimation of the whole target function (i.e., the whole reachable region set in RCM). For the second challenge, we work on the level of *zones* (which are state-spaces characterizable by set of clock difference constraints), and develop a procedure to calculate the region coverage estimation of a zone in the state-space. Many model-checkers for TAs are built around the central manipulation procedures of zones[21, 24, 25, 26, 28]. Finally, for the third challenge, we present a data-structure and show that the data-structure can represent a state-space as a set of disjoint zones. With this data-structure, we can estimate the coverage of a state-space as the sum of coverages of a set of disjoint zones.

6.1 Tight estimation of the target function

In general, it is very expensive to compute representations for the exact reachable state-spaces. In our previous implementations, we use abstractions of either the backward or the forward reachable state-spaces to compute the estimation for the whole target function in RCM. But such estimations seem very imprecise. In some experiments, the final coverage estimations, when the whole reachable state-space representations have been constructed, fall in the range of 10^{-5}. Moreover, much proportion of the reachable state-space seems irrelevant to the reachability from initial states to risk states.

We have observed that to analyze this reachability, we only have to trace through those states which are both backward reachable from a risk state and forward reachable from an initial state. So we use the following steps to compute an estimation of the whole target function.

```
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 $F := F \wedge B$;
```

In the second statement, we employ an abstraction technique, called *magnitude abstraction*, to compute the weakest precondtions 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. Due to page-limit, we shall not go into details of this abstraction technique.

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 much better coverability.

6.2 Coverage estimation of a zone

A *zone* is a state-space characterizable by a range constraints 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 $c, c' \in [-C_{A:\eta}, C_{A:\eta}] \cup \{-\infty, \infty\}$.

We shall intuitively compute something like a *normalized volume* estimation of zones. The volume estimation of a rectangular 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 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 desgin 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 $[-3,-3], (-3,-2), \ldots, [1,1], (1,2)$.

Such volume estimations can result in huge numbers not representable by integers in computers' hardware. Thus instead of using the absolute length in each dimension to compute the estimated volume, we choose to use the normalized lengths (i.e. the floating point numbers of the length divided by the maximum length of the difference variables). The normalized length for Q' is thus $|Q'|/|Q|$. The normalized length for a clock difference constraint is broken down to the following eight cases:

- $(2(c'-c)+1)/(4C_{A:\eta}+3)$ for $c \leq x - x' \leq c'$ with $c \geq -C_{A:\eta}$ and $c' \leq C_{A:\eta}$,
- $(2(c'-c))/(4C_{A:\eta}+3)$ for either $c < x - x' \leq c'$ or $c \leq x - x' < c'$ with $c \geq -C_{A:\eta}$ and $c' \leq C_{A:\eta}$,
- $(2(c'-c)-1)/(4C_{A:\eta}+3)$ for $c < x - x' < c'$ with $c \geq -C_{A:\eta}$ and $c' \leq C_{A:\eta}$,
- $(2(c'+C_{A:\eta})+2)/(4C_{A:\eta}+3)$ for $-\infty < x - x' \leq c'$ with $c' \leq C_{A:\eta}$,
- $(2(c'+C_{A:\eta})+1)/(4C_{A:\eta}+3)$ for $-\infty < x - x' < c'$ with $c' \leq C_{A:\eta}$,
- $(2(C_{A:\eta}-c)+2)/(4C_{A:\eta}+3)$ for $c \leq x_1 - x_1' < \infty$ with $c \geq -C_{A:\eta}$,
- $(2(C_{A:\eta}-c)+1)/(4C_{A:\eta}+3)$ for $c < x_1 - x_1' < \infty$ with $c \geq -C_{A:\eta}$,
- 1 for $-\infty < x_1 - x_1' < \infty$; /* this case is usually not represented in zones */

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

$$\frac{|Q'|}{|Q|} \cdot \Pi_{\text{"}c \sim x - x' \sim' c'\text{"} \in K}(\text{the normalized length of } c \leq x - x' \leq c')$$

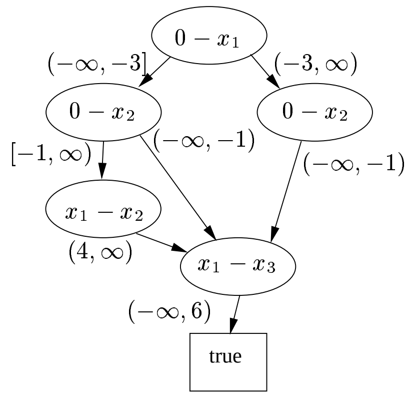

Figure 1: CDD for $(0 - x_1 \leq -3 \wedge x_1 - x_3 < -4 \wedge x_2 - x_1 < 6) \vee (0 - x_2 < -1 \wedge x_2 - x_1 < 6)$

6.3 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 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 (Clock-Difference Diagram)[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 figure 1. Each path in this figure represents a zone in closure form. Due to page-limit, we refer the interested readers to [6] for the definition and manipulations of CDD.

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. $||$

In our implementation, we kind of follow the approach in [6] that a set of zones are first normalized to their closure forms before being stored in a CDD. Thus, 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 the following symbolic procedure to compute the estimated volume of a state-space represented by CDD with all zones in closure forms.

```
normalized_volume($D$) /* $D$ 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 $D$ is $(x-x',(\lambda_1,D_1),\ldots,(\lambda_n,D_n))$, then
    return $\sum_{1\leq i\leq n}\frac{\text{the number of basic intervals covered by } \lambda_i}{4C_{A:\eta}+3}$normalized_volume($D_i$);
}
```

6.4 Estimation of the region coverage

In our framework, both V and F in `Symbolic_Simulate()` in page 5 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'.

The procedure to transform a state-space representation in BDDs and DBMs to this representation can be found in [6]. Suppose at statement (4) of each iteration of procedure `Symbolic_Simulate()` in page 5, then the estimated normalized volume v for V is $\sum_{(Q',D)\in V}\frac{|Q'|}{|Q|}\cdot\texttt{normalized\_volume}(D)$ and $\frac{v}{f}=\frac{\sum_{(Q',D)\in V}\frac{|Q'|}{|Q|}\cdot\texttt{normalized\_volume}(D)}{\sum_{(Q',D)\in F}\frac{|Q'|}{|Q|}\cdot\texttt{normalized\_volume}(D)}$. With lemma 2 and the algorithm of `normalized_volume()`, we can prove the following lemma.

LEMMA 3 *RCM satisfies the criteria of accountability.*
Proof : Omitted due to page-limit. ||

The following lemma shows that our RCM has discernment.

LEMMA 4 *Given a timed automata A and a risk predicate η, in RCM, safe states and unsafe states are not in the same portion.*
Proof : This is true because symbolic manipulations of zones are sufficient to answer the reachability problem of timed automata[1]. ||

According to lemma 4, with RCM, it is not possible to report 100% coverage without the detection of an existing risk state, if any.

7 Triggering-condition coverage metric (TCM)

RCM has the advantage in accountability and discernment. But it may result in low coverability. 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. Triggering condition coverage is accounted as the summation of triggering condition coverage for each transition. For a transition e with triggering constraints on clock differences $x_1 - x'_1, \ldots, x_n - x'_n$, then a basic portion of TCM for e is a minimal state-space, characterizable by a set of constraints like $c_i \sim x_i - x'_i \sim' c'_i$ with $1 \leq i \leq n$ and $c_i, c'_i \in \{-\infty, -C_{A:\eta}, \ldots, -1, 0, 1, \ldots, C_{A:\eta}, \infty\}$.

The numerical estimation f of the whole target function in procedure `Symbolic_Simulate`() can be computed as $\sum_{e \in T}$ `normalized_volume`$(\tau(e))$. We use $|T|$ variables, V_e for each $e \in T$, to record the verified proportion of the triggering condition of each transition. Initially, for all $e \in T$, $V_e =$ *false*. At each iteration's execution of statement (4), we execute the following steps to compute the value v.

```
for each $e \in D$, $V_e := V_e \vee (\texttt{abstract}_e(\phi \wedge \tau(e)))$;
let $v := \sum_{e \in T} \texttt{normalized\_volume}(V_e)$;
```

Here $\texttt{abstract}_e(d)$ is a procedure which eliminates 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 $\texttt{abstract}_e(0 - x < -7 \wedge x - y \leq -2 \wedge y - 0 \leq 2) = 0 - x < -7 \wedge x - y \leq -2$ and the constraint literal $y - 0 \leq 2$ is filtered out since no constraint on difference $y - 0$ is used in the triggering condition of e. Also in these two steps, we assume that while invoking `normalized_volume`(V_e) for each e, V_e has already been transformed to the representation like $\{(Q'_1, D_1), \ldots, (Q'_n, D_n)\}$.

It can be shown that TCM has the following desirable property.

LEMMA 5 *TCM satisfies the criteria of accountability.*
Proof : Proof omitted due to page-limit. $\|$

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. The input language of `red` is a set of *communicating timed automata (CTA)* which communicate with one another through CSP-style synchronization channels[19]. For each channel σ, two processes have to execute at the same instant to achieve a synchronization through the channel. In the synchronization instant, one process executes a transition with event $!\sigma$ for output and the other executes a transition with event $?\sigma$ for input.

To check the possibility of using our techniques in real-world projects, we have experimented with the *L2CAP (Logical Link Control and Adaptation Layer Protocol)* 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 higher level message multiplexing, packet segmentation and reassembly, and the conveying of quality of service information. The protocol regulates the behaviors 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. In appendix D, more coverage and performance data of our implementation with various trace-generation strategies can be found. Data is collected on a Pentium 4 with 1.7GHz, 256MB, running Red Hat Linux 7.0.

8.1 Modelling L2CAP

An L2CAP device may play the role of either master or 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 appendix A. The scenario starts when the master's upper layer issues an `L2CA_ConnectReq` (Connection Request) through the L2CA interface. Upon receiving the request, the master communicates the request through the unreliable network to the slave (with an `L2CAP_ConnectReq`), who will then convey the request to the slave's upper layer (with an `L2CA_ConnectInd`).

The protocol goes on with messages bouncing back and forth until the master sends an `L2CAP_ConfigRsp` message to the slave. Then both sides exchange data. Finally the master upper layer issues message `L2CA_DisconnectReq` to close the connection and the slave confirms the disconnection.

We have made the following assumption in the model. When an upper layer process needs to send out an event in response to the receiving of an event, the time between the receiving and sending is in $[0, 5]$. Also, we assume that the timeout value of RTX timers and ERTX timers are all 60 time units. With one timeout from RTX, the L2CAP process starts ERTX. With the timeout from ERTX, the L2CAP process aborts the session and changes to state `CLOSED`.

We use nine processes to model the whole activity in L2CAP. They are the *master upper* (user on the master side), the *master* (L2CAP layer), master L2CAP time-out process, master L2CAP extended time-out process, the *slave upper* (user on the slave side), the *slave* (L2CAP layer), slave L2CAP time-out process, slave L2CAP extended time-out process, and the *network*. Each of these processes is described as a communicating timed automaton. The CTA for both the master and the slave can be found in appendix B. The safety condition is that while master L2CAP device still stays in the OPEN state at the time, the slave L2CAP device may not enter the W4_L2CA_DISCONNECT_RSP state. According to the protocol state transition table, this safety condition is guaranteed.

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.

Table 1: Coverage estimations and overheads w.r.t. iterations when there is no bugs

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 without faults. That is, each time we execute statement (3) in procedure `Symbolic_Simulate()`, we let $D = T$ and $\phi = \psi$.

In each iteration, we calculate three estimations according to the three coverage metrics respectively. The data is in table 8.2. After 18 iterations, `red` 4.1 finishes the exhaustive search consuming total cpu time 37.14 sec and memory usage 782k with ACM, total cpu time 30.59 sec and memory usage 632k with RCM, and total CPU time 35.79 sec and memory usage 722k with TCM, and reports that the risk state is NOT reachable. 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 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 saves verification cost.

Since RCM gets us very close to 100% coverage, we can still 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 17'th iteration, it could

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

Table 2: Coverage estimations w.r.t. two strategies for the 6 faulty models

still increase to reflect more portions have been traced through while ACM and TCM have already converged to 1.

As for the efficiency of our coverage estimation methods, we find that at the end of the state-space construction, the overhead incurred in the coverage estimation respectively for ACM, RCM and TCM are about 0.70, 7.57 and 0.44 seconds. Compared with the verification time, we find that for ACM, only $0.70/37.14 \approx 0.01885 = 1.885\%$ of the verification CPU time are used in the coverage estimation. For TCM, only $0.44/35.79 \approx 0.01229 = 1.229\%$ of the verification CPU time are used. This means that our implementation for both ACM and TCM are quite efficient. For RCM, $7.57/30.59 \approx 0.24747 = 24.747\%$ of the verification CPU time is used in the coverage estimation. Detailed breaking-down of the computation time shows that most of the overhead is consumed in the construction of both the backward and the forward reachable state-space representations. This is the price to pay for better discernment.

8.3 Coverage estimation when there is a fault

We have created 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. The description of the six faulty models are given in appendix C. We tried two trace-generation strategies. The first one is breadth-first (see subsection 8.2). The second one is *depth-first*. That is, at each time when we execute statement (3), we choose D 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 (3). The coverage data is shown in table 2. The most interesting thing in table 2 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.

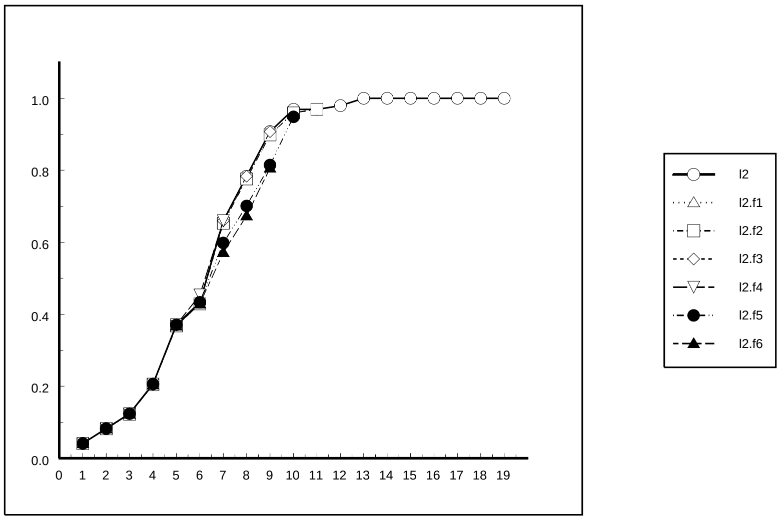

Figure 2: Growth of coverage w.r.t. iterations in ACM

In figure 2, we draw the growth of coverage in ACM for the six faulty models and the correct model with respect to the iterations. Figures 3 and 4 are for RCM and TCM respectively.

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].

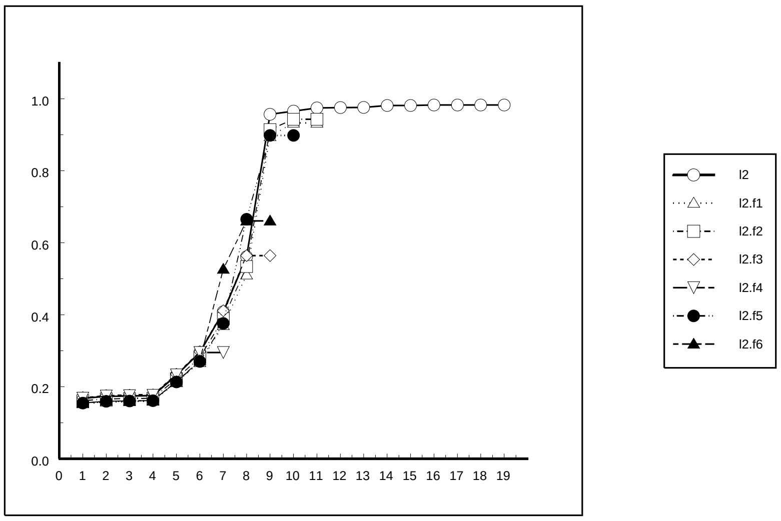

Figure 3: Growth of coverage w.r.t. iterations in RCM

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, second edition, 1990.

[5] L. Bening, H. Foster. Principles of Verifiable RTL Design, a Functional Coding Style Supporting Verification Processes in Verilog,li 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.

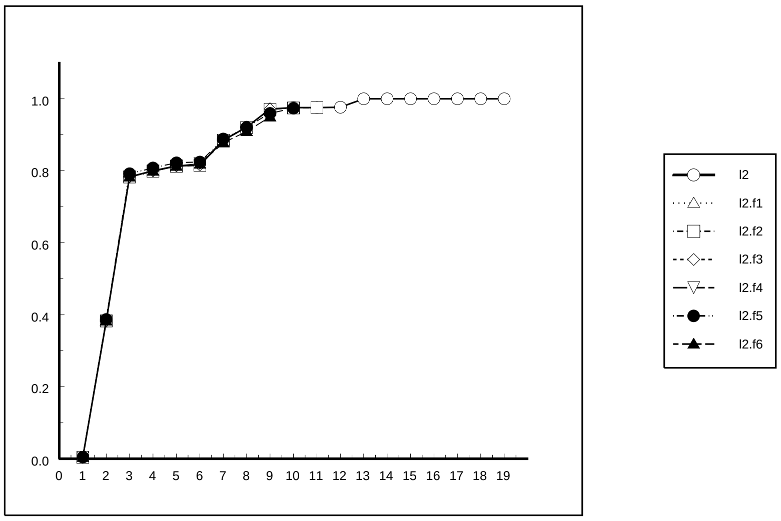

Figure 4: Growth of coverage w.r.t. iterations in TCM

[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 30^th Design Automation Conference, page 86-91, June 1993.

[9] Hana Chockler, Orna Kupferman, Moshe Y. Vardi Coverage Metrics for Temporal Logic Model Checking. Lecture Notes in Computer Science, vol 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, pages 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] 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.

[17] Y. Hoskote, T. Kam, P.-H. Ho, and X. Zhao Coverage estimation for symbolic model checking. In Proceedings of DAC1999, pp. 300-305.

[18] T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-Time Systems, IEEE LICS 1992.

[19] C.A.R. Hoare. Communicating Sequential Processes, Prentice Hall, 1985.

[20] 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.

[21] P. Pettersson, K.G. Larsen, UPPAAL2k. in Bulletin of the European Association for Theoretical Computer Science, volume 70, pages 40-44, 2000.

[22] P. Rashinkar, P. Paterson, L. Singh. System-on-a-chip Verificatoin, Methodology and Techniques. Kluwer Academic Publishers, 2001.

[23] 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.

[24] F. Wang. Efficient Data-Structure for Fully Symbolic Verification of Real-Time Software Systems. TACAS'2000, March, Berlin, Germany. in LNCS 1785, Springer-Verlag.

[25] F. Wang. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram, to appear in Proceedings of FORTE, August 2001, Cheju Island, Korea.

[26] F. Wang. Efficient Verification of Timed Automata with BDD-like Data-Structures. to appear in proceedings of VMCAI 2003, Jan. 2003, in LNCS, Springer-Verlag; New York City, USA.

[27] 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.

[28] 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.

APPENDICES

A Message sequence chart for Bluetooth L2CAP

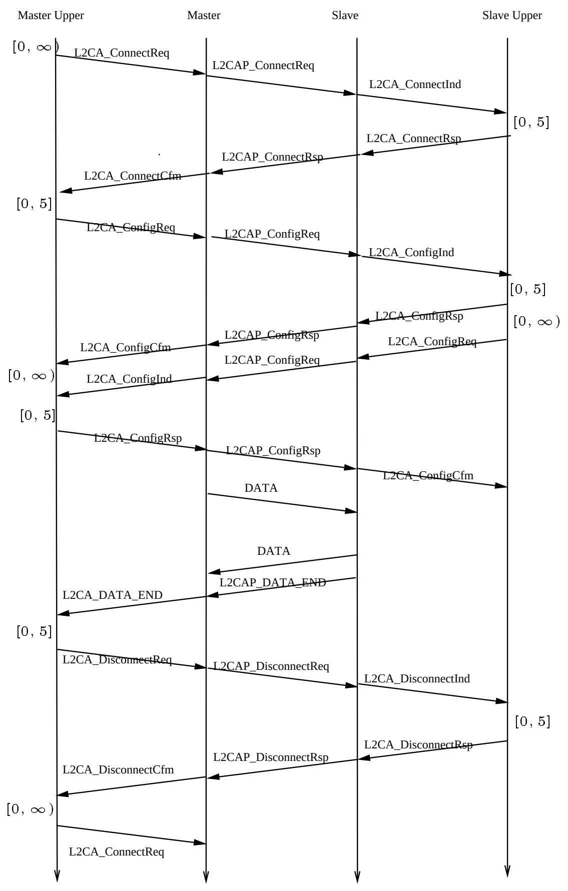

Figure 5: A message sequence chart of L2CAP

The two outside vertical lines represent the L2CA interfaces from (slave's and master's) upper layers to the L2CAP layers (slave and master respectively).

B Bluetooth L2CAP

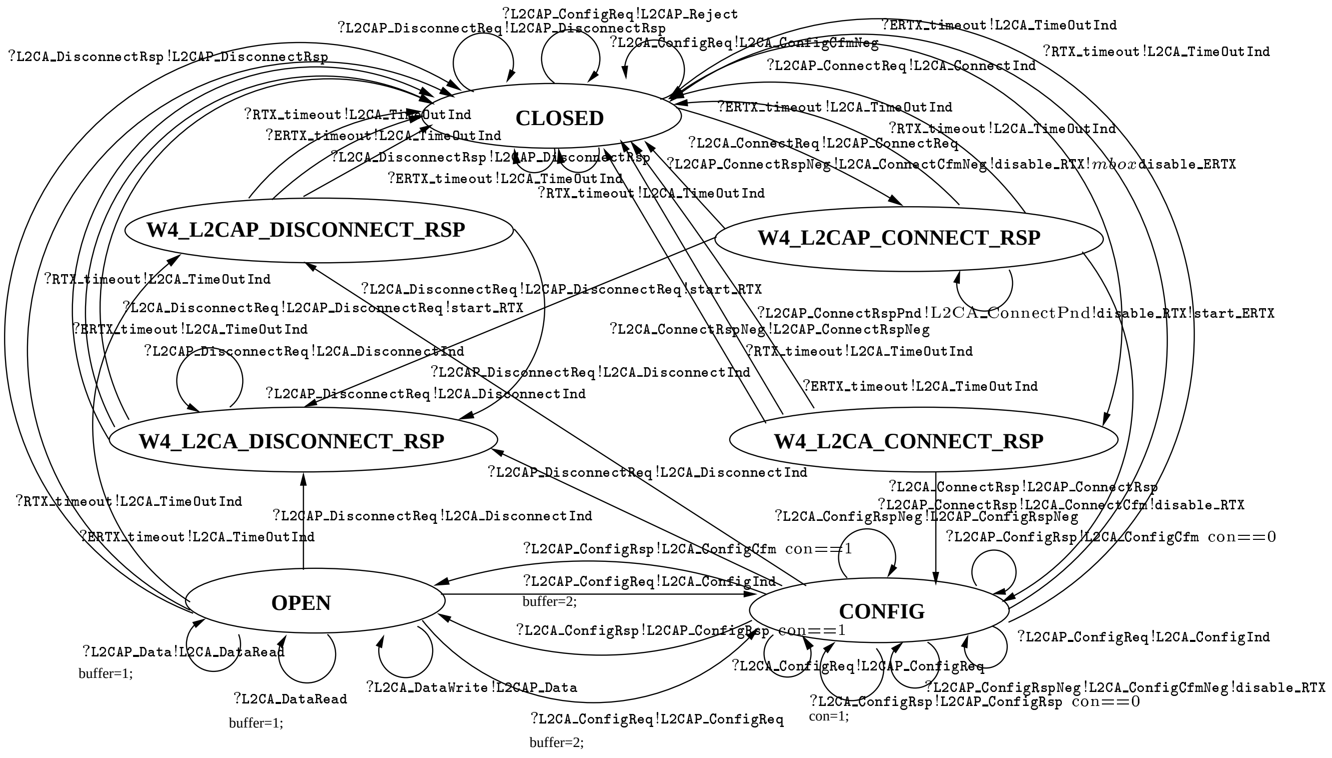
CLOSED
W4_L2CAP_DISCONNECT_RSP
W4_L2CAP_CONNECT_RSP
W4_L2CA_DISCONNECT_RSP
W4_L2CA_CONNECT_RSP
OPEN
CONFIG
?L2CAP_ConfigReq!L2CAP_Reject
?L2CAP_DisconnectReq!L2CAP_DisconnectRsp
?L2CA_ConfigReq!L2CA_ConfigCfmNeg
?L2CA_DisconnectRsp!L2CAP_DisconnectRsp
?ERTX_timeout!L2CA_TimeOutInd
?RTX_timeout!L2CA_TimeOutInd
?L2CAP_ConnectReq!L2CA_ConnectInd
?L2CA_ConnectReq!L2CAP_ConnectReq
?L2CAP_ConnectRspNeg!L2CA_ConnectCfmNeg!disable_RTX!mboxdisable_ERTX
?L2CA_DisconnectRsp!L2CAP_DisconnectRsp
?L2CA_DisconnectReq!L2CAP_DisconnectReq!start_RTX
?L2CAP_ConnectRspPnd!L2CA_ConnectPnd!disable_RTX!start_ERTX
?L2CA_ConnectRspNeg!L2CAP_ConnectRspNeg
?L2CAP_DisconnectReq!L2CA_DisconnectInd
?L2CA_ConnectRsp!L2CAP_ConnectRsp
?L2CAP_ConnectRsp!L2CA_ConnectCfm!disable_RTX
?L2CA_ConfigRspNeg!L2CAP_ConfigRspNeg
?L2CAP_ConfigRsp!L2CA_ConfigCfm con==1
?L2CAP_ConfigRsp!L2CA_ConfigCfm con==0
?L2CAP_ConfigReq!L2CA_ConfigInd
buffer=2;
?L2CA_ConfigRsp!L2CAP_ConfigRsp con==1
?L2CA_ConfigReq!L2CAP_ConfigReq
?L2CAP_ConfigRspNeg!L2CA_ConfigCfmNeg!disable_RTX
?L2CA_ConfigRsp!L2CAP_ConfigRsp con==0
con=1;
?L2CAP_Data!L2CA_DataRead
buffer=1;
?L2CA_DataRead
buffer=1;
?L2CA_DataWrite!L2CAP_Data
?L2CA_ConfigReq!L2CAP_ConfigReq
buffer=2;

Figure 6: CTA of a Bluetooth device

C Description of the six faulty models

All faulty models lead the risk state reachable and violate the safety condition. Recall that the safety condition is that while master L2CAP device still stays in the OPEN state at the time, the slave L2CAP device may not enter the W4_L2CA_DISCONNECT_RSP state. We compare the fault and correct behaviors of these faulty models as below:

- `Faulty Model1` : The slave will enter into W4_L2CA_DISCONNECT_RSP state while receiveing master's data from network and notifying upper layer. In the correct model, the slave shall stay in OPEN state.
- `Faulty Model2` : The slave will leave OPEN state and enter into W4_L2CA_DISCONNECT_RSP state while reciveing upper layer's disconnect command, sending this request to master through network, and starting the timer. In the correct model, the slave shall leave OPEN state and enter into W4_L2CAP_DISCONNECT_RSP state to wait for the master's response from network.
- `Faulty Model3` : The master remains staying in OPEN state while recive-

ing upper layer's disconnect command, sending this request to slave through network, and starting the timer. In the correct model, the master shall leave OPEN state and enter into W4_L2CAP_DISCONNECT_RSP state to wait for the slave's response from network.

- `Faulty Model4` : The master will leave CONFIG state and enter into OPEN state while reciveing upper layer's disconnect command, sending this request to slave through network, and starting the timer. In the correct model, the master shall enter into W4_L2CAP_DISCONNECT_RSP state to wait for the slave's response from network.
- `Faulty Model5` : The slave will leave CONFIG state and enter into W4_L2CAP_DISCONNECT_RSP state while receiving upper layer's configuration response and having received master's response. In the correct mode, the slave shall leave CONFIG state and enter into OPEN state after finishing the configuration process.
- `Faulty Model6` : The slave will leave CONFIG state and enter into W4_L2CAP_DISCONNECT_RSP state while receiving upper layer's configuration response but not yet having received master's response. In the correct mode, the slave should stay in CONFIG state since it doesn't finish the configuration process.

D Coverage estimation with various search strategies

It is also interesting to see how our techniques can be used together with various trace-generation strategies in symbolic simulation. Due to page-limit, we only briefly describe the trace-generation strategies that we have implemented for our experiments in the following.

- *Random Walk Strategy*: Each time we execute statement (3), `red` 4.1 randomly choose a firable transition to be the sole element in D.
- *Game-based Strategy*: We use the term *"game"* here because we envision the concurrent system operation as a game. Users can specify some processes to be treated as *players* while the other processes are treated as *opponents*. At each time we execute statement (3), we either randomly choose a firable transition from the opponent processes or choose D to be the set of all firable transitions of the player processes. In this strategy, we alternately execute sequences of all players' transitions and sequences of a random-walk of the opponents' transitons. In this experiment, we view all processes whose local variables and clocks appear in the safety predicate as players. All other processes are opponents.
- `Goal-oriented Strategy`: According to this strategy, heuristics are designed for the choice of a single transition in each execution of statement (3) in the hope that a short trace leading to a risk state can be constructed.

Our coverage data with the generation of a single symbolic trace is in table 3. The experiments shows that it is viable to integrate our techniques with other verification and simulation techniques.

Strategy	Faulty Models	Depth	ACM	RCM	TCM	Risk state reached?
Random Walk	1	21	34/98	0.437014	0.623092	No
	2	11	23/98	0.181328	0.423497	No
	3	11	22/97	0.172719	0.411343	No
	4	8	26/97	0.168460	0.217365	No
	5	30	44/97	0.294042	0.839828	No
	6	19	27/98	0.179093	0.636663	No
Goal Oriented	1	15	19/98	0.187034	0.613695	No
	2	14	12/98	0.164324	0.597204	No
	3	8	26/97	0.169848	0.217365	No
	4	13	12/97	0.199808	0.597815	No
	5	14	28/97	0.157394	0.613456	No
	6	19	19/98	0.163539	0.613695	No
Game Based	1	14	30/98	0.429387	0.421141	No
	2	8	26/98	0.160757	0.217143	No
	3	23	33/97	0.309106	0.631373	No
	4	17	29/97	0.188423	0.429172	No
	5	20	32/97	0.183961	0.626350	No
	6	24	20/98	0.158451	0.798677	No

Table 3: Coverage estimations w.r.t. automatic trace generation strategies for the 6 faulty models