Test Plan Generation for Concurrent Real-Time Systems based on Zone Coverage Analysis

Farn Wang[1,2] Geng-Dian Huang[1]

1: Dept. of Electrical Engineering, National Taiwan University, Taiwan, ROC
2: Grad. Inst. of Electronic Engineering, National Taiwan University, Taiwan, ROC

farn@cc.ee.ntu.edu.tw

Abstract. The state space explosion due to concurrency and timing constraints of concurrent real-time systems (CRTS) presents significant challenges to the verification engineers. In this paper, we investigate how to use coverage techniques to generate efficient test plans for such systems. We first discuss how to use communicating timed automata to model CRTS. We present a new coverage technique, AZC (active zone coverage), based on the zone equivalence relation between states of CRTS. We discuss techniques to estimate AZC values of active zones represented in BDD-like diagrams. We explain how to construct zone trees and map their root-to-leaf paths to test cases. We then present an algorithm to generate test plans by prioritizing the test cases. The test plans that we generate can efficiently achieve full coverage in AZC. We have implemented our ideas with our TCTL model-checker RED. Experiment report with the Bluetooth L2CAP showed improvement of the coverage growth rate in the test plan execution.

1 Introduction

Although there have been several alternative technologies, e.g. model-checking and simulation, *testing* [4, 19, 29, 30] is still the major verification technique in the software industry over the last few decades. To verify a *system under test* (*SUT*), we need a set of *test cases*. Each test case specifies a sequence of input events and expected output events from the SUT. A *test plan* is a sequence of test cases to be executed in succession. A bad test plan may spend valuable time in repetitively testing equivalent behaviors and missing some other faulty behaviors. In contrast, a good test plan can methodically and efficiently execute test cases to give us high confidence of the SUT in a short time. At this moment, the generation of efficient test plans is still in the center of software testing research.

However, the complexity of new-generation *concurrent real-time systems* (*CRTS*) is driving the cost of testing to over fifty percent of the development budgets. Especially, for CRTS, there are infinitely many time instances in which an event can happen. Thus it is impossible to test all behaviors in either theory or practice. In the last few decades, the software industry has heavily relied on the wisdom of *coverage* techniques [29, 30] to maintain the high quality of software systems. Coverage techniques partition the behavior space of an SUT into finitely many equivalence classes. Coverage techniques can be used to measure how much of an SUT has been tested. In general, the more coverage you get, the more confidence you have in your SUT.

One coverage techique is the *visited-state coverage* [2,22] that uses the ratio of the number of visited states over the number of reachable states to measure the progress of test execution. Visited-state coverage has great discerning power since each state records the information necessary for the reaction to all future events from the states. However for CRTS, visited-state coverage is neither practical since the number of states is usually infinite [1]. In this work, we investigate how to use state-based coverage techniques for CRTS. In [1], a very fine partition of dense-time state-spaces is proposed. In this partition, an equivalence class is called a *region*. Given a description of a CRTS, the number of regions in a state-space is usually exponential to the description size. It is usually infeasible to cover all regions in a testing task. Another partition is with the equivalence classes of *zones* [14]. Zones can be constructed with on-the-fly techniques and usually partition a state-space in a coarse but precise enough granularity for the model-checking of dense-time models. In this work, we investigate how to adapt the zone technology for the efficient generation of test plans for CRTS. Specifically, we propose the following techniques.

- *Communicating timed automata* (*CTA*) [23,31,32] for the modeling of CRTS. A CTA may have many processes that interact through message channels and measure event intervals with dense-time clocks.
- *Active zone coverage (AZC) for the progress estimation of testing tasks for CRTS.* We first propose *zone-coverage* (*ZC*) for the measurement of progress of testing tasks for CRTS. We then propose a technique, based on inactive variable analysis, to enhance ZC to *active ZC* (*AZC*) for the construction of efficient test plans. With AZC, an equivalence class is called an *active zone*.
- *Test plan generation with priority for gains in AZC.* We first present an algorithm to construct zone trees that symbolically represent the branching structure of a CRTS. Then we discuss how to convert paths in the zone trees to linear test cases. Then we present a heuristic algorithm that prioritizes the test cases and construct a test plan. We prove that the test plan can always be constructed and visits all active zones.

We have implemented our ideas and experimented with the Bluetooth L2CAP [11, 32]. The experiment result shows that comparing with non-prioritized test plans, the prioritized test plans indeed improve the efficiency for high AZC coverage.

The rest of this paper is structured as follows. We review related work in section 2. We present our modeling language CTA in section 3. We explain some basic techniques for dense-time space manipulation and for zone forest construction in section 4. We present AZC in section 5. We explain how to generate a set of test cases for full AZC coverage in section 6. We present an algorithm to generate test plans with priority for high gains in AZC in section 7.

We report our experiment in section 8. Finally, we present the conclusion and discuss future work in section 9.

2 Related work

A popular coverage technique for software testing is the *statement coverage* [6] which measures the proportion of already-executed statements during testing. The *transition coverage* [15] measures that of the executed transitions of the control flow machines. The *visited-state coverage* [2,22] from the VLSI industry measures that of the visited states.

For dense-time systems, ACM, TCM, and RCM has been introduced to measure the progress of model-checking [32].

For testing dense-time systems, the popular *timed automata (TA)* [1] and its variations have been used as the formal specification languages [25]. Test case generation algorithms for untimed systems can also be applied by first discretizing the dense state-space. In [17], the discretization is achieved through digital clock automata. In [25], a timed automaton is discretized into a finite grid automaton with granularity 2^{-n} where n is the number of regions. In [7], a timed automaton is discretized at a sampling frequency $\frac{1}{|X|+2}$ where X is the set of clocks. These methods encounter the state space explosion problem even with small systems since they partition the state spaces with fine granularities.

In [12], the fastest diagnostic trace facility of UPPAAL [21] (a model checker for real-time systems) is used to generate time-optimal test cases. Three coverage techniques: *edge coverage*, *location coverage*, *definition-use pair coverage* for untimed systems are used.

In [20], real-time systems are modeled as event-recording automata. Then symbolic techniques are used to construct the reachability graph out of the equivalence-class graph. Finally, test cases are generated to cover all equivalence classes. In [12,17,20], test cases are generated to guarantee traditional coverage techniques, like arc coverage, domain analysis, location coverage, data-flow analysis, .

In [3], a state space exploration algorithm is proposed for efficiently computing the minimum cost of reaching a goal state in the model of *uniformly priced timed automata*. They used the cost metric to guide the state space exploration.

In [8], test cases are prioritized according to the rate of fault detection in regression testing. Such a technique has been implemented in Echelon for regression testing [24].

Huang and Wang [16] used symbolic techniques to automatically generate test cases, with coverage annotations for dense-time systems, with the CRD data-structures [28].

3 Communicating timed automata (CTA)

We first need to introduce our modeling language for CRTS. A *communicating timed automaton (CTA)* [23,31,32] is a set of *process timed automata (PTA)*,

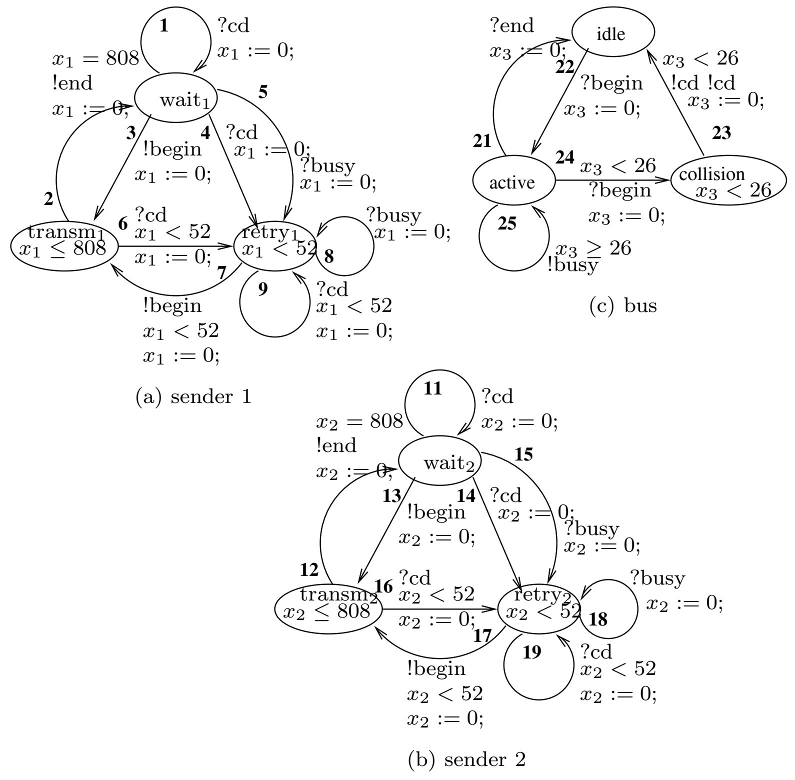

Fig. 1. Specification of a bus-contending protocol

equipped with a finite set of dense-time clocks and synchronization channels. A PTA is structured as a directed graph whose nodes are *modes (control locations)* and whose arcs are *transitions*. The modes are labeled with *invariance conditions* while the transitions are labeled with *triggering conditions* and a set of clocks to be reset during the transitions. The invariance conditions and triggering conditions are Boolean combinations of inequalities comparing clocks with integers. At any moment, each PTA can stay in only one *mode* (or *control location*).

In the operation of a CTA, a minimal subset (called a *global transition*) of the PTA transitions can be triggered when the corresponding triggering conditions are satisfied and their input/output events are synchronized. Upon being triggered, all PTAs participating in the global transition instantaneously transit from one mode to another and resets some clocks to zero. In between transitions, all clocks in the CTA increase their readings at a uniform rate.

In figure 1, we have one bus process PTA and 2 sender PTAs for the modeling of a bus-contending protocol. The circles represent modes while the arcs represent transitions, which may be labeled with input/output events (e.g., !`begin`, ?`end`, ...), triggering conditions (e.g., $x < 52$), and assignments (e.g., $x := 0;$).

For convenience, we have labeled the transitions with numbers. In the system, a sender process may synchronize through channel `begin` with the bus to start sending messages on the bus. While one sender is using the bus, the second one may also synchronize through channel `begin` to start placing messages on the bus and corrupting the bus contents. When this happens, the bus then signals bus collision (`cd`) to all the senders.

Due to page limit, we only give a brief definition of CTA. For detailed definition, please refer to [31,32]. 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 conjunctions of inequalities of the forms q and $x \sim c$, where $q \in Q$, $x \in X$, '$\sim$'$\in \{\leq, <, =, >, \geq\}$, and c is an integer constant. An element in $B(Q, X)$ is called *conjunctive* if the only Boolean operators in the element are conjunctions. $\mathbb{R}^{\geq 0}$ is the set of nonnegative real numbers.

Definition 1. **Process timed automata (PTA)** A PTA P is a tuple $\langle X, \Sigma, Q, I, \mu, E, \delta, \lambda, \tau, \pi \rangle$. X is a finite set of clocks. Σ is a finite set of synchronization channels. Q is a finite set of modes. $I \in B(Q, X)$ is the initial condition. $\mu : Q \mapsto B(\emptyset, X)$ defines the conjunctive invariance condition of each mode. E is the set of process transitions. $\delta : E \mapsto (Q \times Q)$ defines the source and destination modes of each process transition. $\lambda : (\Sigma \times E) \mapsto \mathbb{Z}$ defines the number of events sent and received at each process transition. When $\lambda(\sigma, e) \leq 0$, it means that process transition e receives $|\lambda(\sigma, e)|$ events through channel σ. When $\lambda(\sigma, e) > 0$, it means that process transition e sends $\lambda(\sigma, e)$ events through channel σ. $\tau : E \mapsto B(\emptyset, X)$ and $\pi : E \mapsto 2^X$ respectively define the conjunctive triggering condition and the clock set to reset of each transition. ■

Definition 2. **Communicating timed automata (CTA)** A CTA A of m processes is a tuple $\langle \Sigma, P_1, P_2, \ldots, P_m \rangle$, where Σ is the set of synchronization channels and for each $1 \leq p \leq m$, $P_p = \langle X_p, \Sigma, Q_p, I_p, \mu_p, E_p, \delta_p, \lambda_p, \tau_p, \pi_p \rangle$ is the PTA for process p and for each $1 \leq p < p' \leq m$, $Q_p \cap Q_{p'} = \emptyset$. ■

A *valuation* of a set is a mapping from the set to another set. Given an $\eta \in B(\bigcup_{1 \leq p \leq m} Q_p, \bigcup_{1 \leq p \leq m} X_p)$ and a valuation ν of $\bigcup_{1 \leq p \leq m}(X_p \cup Q_p)$, we say ν *satisfies* η, in symbols $\nu \models \eta$, iff η is evaluated *true* when the variables in η are interpreted according to ν.

Definition 3. **States** Suppose we are given a CTA $A = \langle \Sigma, P_1, P_2, \ldots, P_m \rangle$ such that for each $1 \leq p \leq m$, $P_p = \langle X_p, \Sigma, Q_p, I_p, \mu_p, E_p, \delta_p, \lambda_p, \tau_p, \pi_p \rangle$. A state ν of A is a valuation of $\bigcup_{1 \leq p \leq m}(X_p \cup Q_p)$ with the following constraints.

- For each $q \in \bigcup_{1 \leq p \leq m} Q_p$, $\nu(q) \in \{\mathit{false}, \mathit{true}\}$. Moreover for each $1 \leq p \leq m$, there is exactly a $q \in Q_p$ such that
$$\nu(q) \wedge \forall q' \in Q - \{q\}(\neg \nu(q')).$$
Given $q \in Q_p$, if $\nu(q)$ is true, we denote q as $\mathtt{mode}_p(\nu)$.
- For each $x \in \bigcup_{1 \leq p \leq m} X_p$, $\nu(x) \in \mathbb{R}^{\geq 0}$ such that $\nu \models \bigwedge_{1 \leq p \leq m} \mu_p(\mathtt{mode}_p(\nu))$.

For any $t \in \mathbb{R}^{\geq 0}$, $\nu + t$ is a state identical to ν except that for every clock $x \in \bigcup_{1 \leq p \leq m} X_p$, $(\nu + t)(x) = \nu(x) + t$. ■

A *global transition* γ of a CTA is a mapping from process indices p, $1 \leq p \leq m$, to $E_p \cup \{\perp\}$, where $\perp$ means no transition (i.e., a process does not participate in γ). A legitimate global transition has to be *synchronized*, that is, each output event from a process is received by exactly one unique corresponding process with a matching input event. In arithmetic, that is $\forall e \in E, \sum_{1\leq p\leq m;\gamma(p)\neq\perp} \lambda(e,\gamma(p)) = 0$. Moreover, to be compatible with the popular interleaving semantics, we require that a global transition must be minimal, i.e., it cannot be broken down to two non-empty synchronized global transitions. For example, in figure 1, we may have a legitimate global transition γ with $\gamma(1) = 1$, $\gamma(2) =\perp$, and $\gamma(3) = 6$. In contrast, another global transition γ' with $\gamma'(1) =\perp$, $\gamma'(2) =\perp$, and $\gamma'(3) = 6$ is not. In the following, whenever we say "global transition", we actually mean "legitimate global transition" for briefness. Given a CTA A, we let $\Gamma(A)$ be the set of legitimate global transitions of A.

Given two states ν, ν' and $t \in \mathbb{R}^{\geq 0}$, $\nu \xrightarrow{t} \nu'$ iff $\nu' = \nu + t$. Also $\nu \xrightarrow{\gamma} \nu'$ iff

- $\nu \models \bigwedge_{1\leq p\leq m;\gamma(p)\neq\perp} \tau_p(\gamma(p))$, and
- ν is identical to ν' except that for all $1 \leq p \leq m$,
  - if $\gamma(p) =\perp$, $\mathtt{mode}_p(\nu) = \mathtt{mode}_p(\nu')$ and $\forall x \in X_p(\nu(x) = \nu'(x))$.
  - if $\gamma(p) \neq\perp$, then
    - $\delta(\gamma(p)) = (\mathtt{mode}_p(\nu), \mathtt{mode}_p(\nu'))$,
    - $\forall x \in (X_p \cap \pi_p(\gamma(p)))(\nu'(x) = 0)$, and
    - $\forall x \notin (X_p \cap \pi_p(\gamma(p)))(\nu'(x) = \nu(x))$.

In the verification of a CRTS A, usually we are given a specification formula ϕ and want to check whether ϕ is violated in the test execution. We assume the framework of safety analysis, in which the specification is a safety predicate in $\phi \in B(\bigcup_{1\leq p\leq m} Q_p, \bigcup_{1\leq p\leq m} X_p)$.

4 Regions, zones, and zone forests

In [32], *RCM (region coverage metric)* is proposed to measure the progress of reachability analysis. Specifically, RCM is an adaptation of the *visited-state coverage* technique from VLSI verification technology, which measures the visited states in an FSM. Instead of measuring the visited states directly, region-equivalence relation [1] can be used to partition the dense-time state-space into a finite set of equivalent classes. Each such equivalence class is called a *region*. Let $C_{A:\phi}$ be the biggest timing constant used in a CTA A and a TCTL specification ϕ. Given a real number t, we let $\mathit{frac}(t) = t - \lfloor t \rfloor$ be the fractional part of t. Two states ν, ν' are in the same region if and only if they satisfy the following constraints.

- Every discrete variable has the same value in the two states.
- For every clock variable x, if either $\nu(x) \leq C_{A:\phi}$ or $\nu'(x) \leq C_{A:\phi}$, then $\lfloor \nu(x) \rfloor = \lfloor \nu'(x) \rfloor$.
- For every two clocks x_1 and x_2, if $\nu(x_1) \leq C_{A:\phi}$ and $\nu(x_2) \leq C_{A:\phi}$, then
$$\mathit{frac}(\nu(x_1)) \leq \mathit{frac}(\nu(x_2)) \text{ if and only if } \mathit{frac}(\nu'(x_1)) \leq \mathit{frac}(\nu'(x_2)).$$

It can be shown that for any subformula ϕ' of ϕ and two states ν, ν' in the same region, ν satisfies ϕ' if and only if ν' satisfies ϕ' [1]. With RCM, we measure the visited regions instead of the concrete states. It is shown that RCM has better discerning power in timing bugs than both TCM (trigger coverage metric) and ACM (arc coverage metric) [32].

However, the number of regions is tremendous for any non-trivial CRTS. It is not infeasible to trace through all the reachable regions. In this work, we propose a new coverage technique, called *zone coverage* (*ZC*), for the testing of CRTS. For efficient verification of dense-time systems, *zones*, instead of regions, have very often been used as the basic unit in state-space manipulation [14]. A zone ζ is a state-space characterizable with a conjunction of atoms (positive or negative literals) in either of the following two forms.

- q_p, for some $1 \leq p \leq m$ and $q_p \in Q_p$.
- $x - x' \sim c$ for clock differences, where x and x' are clocks or 0, '$\sim$' $\in \{\leq, <\}$, and c is an integer.

A region is a smallest zone that is not properly contained by a different region. A zone may contain many regions. Although the number of zones can be of the same complexity as that of regions, in practice verification tools implemented with zones perform much better than those with regions [21, 27, 34].

Our symbolic trace computation is based on a well-discussed procedure, called `post`(), to compute a symbolic post-condition of a zone after a global transition and time-progress [14]. Given a zone ζ and a global transition γ,

$$\texttt{post}(\zeta, \gamma) = \{\nu' | \exists \nu \in \zeta \exists t \in \mathcal{R}^{+} \exists \nu'' (\nu \xrightarrow{\gamma} \nu'' \wedge \nu'' \xrightarrow{t} \nu')\}.$$

Note that the results of the post-condition procedure can also be represented as zones. With relation `post`(), we can construct a *zone forest* for a CTA and a TCTL specification. A zone forest is a set $\langle V, R, K \rangle$ of trees such that V is a set of zones, R is a set of initial zones, and K is a set of triples (ζ, γ, ζ') with $\zeta \in V$, $\zeta' \in V$, and $\zeta' = \texttt{post}(\zeta, \gamma)$.

We can use the following procedure to construct a zone forest for a CTA and a TCTL specification.

```
(1)   ZoneForest($A, \phi$) {
(2)      Rewrite the initial condition of $A$ in DNF $\zeta_1 \vee \ldots \vee \zeta_n$.
(3)      $R := \{\zeta_1, \zeta_2, \ldots, \zeta_n\}$; $V := R$; $\Phi := R$; $K := \emptyset$; $\psi := \bigvee_{\zeta \in V} \zeta$.
(4)      While $\Phi \neq \emptyset$, {
(5)         Pick a zone $\zeta$ from $\Phi$; Let $\Phi := \Phi - \{\zeta\}$.
(6)         For each $\gamma \in \Gamma(A)$, {
(7)            Let $\zeta' :=$ post$(\zeta, \gamma)$.
(8)            If $\zeta' \wedge \neg\psi$ is satisfiable, {
(9)               $\psi := \psi \vee \zeta'$; $\Phi := \Phi \cup \{\zeta'\}$;
(10)              $V = V \cup \{\zeta'\}$; $K := K \cup \{(\zeta, \gamma, \zeta')\}$.
(11)     } } }
(12)     return $\langle V, R, K \rangle$.
(13)  }
```

Note that at line (5), we randomly pick a zone from Φ. In practice, generating a complete zone forest can still be very expensive since the number of zones can be exponential to the sizes of the given CTA and specification. So sometimes we may want to compromise with a partial zone forest. In that case, it is important for us to know that the zones in the forest cover an appropriate portion of the behaviors of the SUT. In section 8, we report experiment with various strategies in expanding the frontier of the zone forest construction.

Lemma 1. *For a CTA A and a CTL formula ϕ, if* $\texttt{ZoneForest}(A, \phi) = \langle V, R, E\rangle$, *then for every state ν that can be reached from an initial state of A, there is a $\zeta \in V$ such that $A, \nu \models \zeta$.* ■

5 Active zone coverage (AZC) for CRTS

Lemma 1 suggests that we can use the zones in a zone forest to estimate the coverage of equivalent state classes in the testing of a CRTS. Straightforwardly, we can count the zones in $\texttt{ZoneForest}(A, \phi)$ to estimate the coverage of a testing task. However, each zones in $\texttt{ZoneForest}(A, \phi)$ may include different number of regions. Based on the techniques discussed in [32] for estimating the number of regions included in a zone, we can define *zone coverage* (*ZC*) that estimates the ratio of number of regions in visited zones over those in reachable zones. Specifically, given a set Z of visited zones, we use $\texttt{RCM}_{A:\phi}(\bigvee_{\zeta\in Z} \zeta)$ to denote the estimation of coverage for those regions in zones in Z for CTA A and TCTL formula ϕ. Thus $\texttt{RCM}_{A:\phi}(\bigvee_{\zeta\in Z} \zeta)$ seems a reasonable definition for zone coverage estimation.

However we can further improve the above-mentioned ZC techniques without sacrificing its discerning power. We present an adaptation of ZC, called *active ZC* (*AZC*), to cut down the number of regions. The idea is to use inactive variable analysis. A variable x is *inactive* in a state if along all computations from the state, x is never read before being written to. Thus the values of inactive variables in a state do not affect the behaviors of a system. There are model-checkers that use inactive variable analysis for state-graph reduction [27]. To check if a variable x is inactive in a state ν, we can construct a CTL formula to characterize those states in which x is active. First, we let $\texttt{Read}(x)$ be the disjunction of the following two constraints.

- The disjunction of invariance conditions (of modes of PTAs) in which x is referenced.
- The disjunction of triggering conditions of all process transitions in which x is read before being writing to. Note that if x appears in both the $\tau()$ and the $\pi()$ components of a process transition, then it is considered read before being written to.

Then we also define $\texttt{PureWrite}(x)$ as the disjunction of the triggering conditions of all transitions e with $x \in \pi_p(x)$ for some process p. With these two conditions, we can now construct the following CTL formula [5] to characterize the states in which x is active.

$$\texttt{active}(x) \equiv \exists(\neg\texttt{PureWrite}(x))\mathcal{U}\texttt{Read}(x).$$

Thus we can use $\neg\texttt{active}(x)$ to characterize those states in which x is inactive.

Now we define *active zones* as new equivalence classes. Given a zone ζ and a variable x, we say that x is inactive in ζ if and only if for x is inactive in every state in ζ. A zone ζ is active if and only if no inactive variable x in ζ appears in the representation of ζ. Given a zone ζ with inactive variables $x_1, \ldots, x_n$, the smallest active zone that contain ζ can be constructed as $\exists x_1 \exists x_2 \ldots \exists x_n(\zeta)$. For convenience, we let $\texttt{active}(\zeta)$ be the smallest active zone that contains ζ. We can establish the following lemma.

Lemma 2. *Given a CTA A, a subformula ϕ' of a TCTL specification ϕ, and a zone ζ, all states of A in ζ satisfies ϕ' if and only if all states in $\texttt{active}(\zeta)$ of A satisfies ϕ'.* ■

It is clear that $\texttt{active}(\zeta)$ can be larger than ζ. To cover a whole reachable state-space, we may need fewer active zones than regular zones. Thus given a set Z of visited zones, we define

$$\texttt{AZC}_{A:\phi}(Z) = \texttt{RCM}_{A:\phi}(\bigvee_{\zeta \in Z} \texttt{active}(\zeta))$$

as the value of Z in the new *active zone coverage* (*AZC*).

6 Test cases from a CTA

Suppose we are given a path through zones $\zeta_0, \zeta_1, \ldots, \zeta_n$ in a zone forest such that for each $0 \le i < n$, $\zeta_{i+1} = \texttt{post}(\zeta_i, \gamma_{i+1})$. We can represent the path with the following notations.

$$\zeta_0 \xrightarrow{\gamma_1} \zeta_1 \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_n} \zeta_n$$

We call a path in this representation a *symbolic trace*. In figure 2, we show such a symbolic trace of the bus-contending protocol (in figure 1). The trace describes

$$\begin{tikzcd} \zeta_0 \arrow[r, "\gamma_1"] & \zeta_1 \arrow[r, "\gamma_2"] & \zeta_2 \arrow[r, "\gamma_3"] & \zeta_3 \arrow[r, "\gamma_4"] & \zeta_4 \end{tikzcd}$$

ζ_0 : $\texttt{mode}_1 = \texttt{idle} \wedge \texttt{mode}_2 = \texttt{wait} \wedge \texttt{mode}_3 = \texttt{wait}$

ζ_1 : $\texttt{mode}_1 = \texttt{active} \wedge \texttt{mode}_2 = \texttt{wait} \wedge \texttt{mode}_3 = \texttt{transmit} \wedge x_1 = x_3$

ζ_2 : $\texttt{mode}_1 = \texttt{collision} \wedge \texttt{mode}_2 = \texttt{transmit} \wedge \texttt{mode}_3 = \texttt{transmit} \wedge$
$x_1 < 26 \wedge x_2 < 26 \wedge x_1 = x_2 \wedge x_3 < 52 \wedge x_3 - x_1 < 26 \wedge x_3 - x_2 < 26$

ζ_3 : $\texttt{mode}_1 = \texttt{idle} \wedge \texttt{mode}_2 = \texttt{retry} \wedge \texttt{mode}_3 = \texttt{retry} \wedge x_2 < 52 \wedge x_3 < 52 \wedge x_2 = x_3$

ζ_4 : $\texttt{mode}_1 = \texttt{active} \wedge \texttt{mode}_2 = \texttt{wait} \wedge \texttt{mode}_3 = \texttt{transmit} \wedge$
$x_1 < 52 \wedge x_2 < 52 \wedge x_3 < 52 \wedge x_1 - x_2 \le 0 \wedge x_1 = x_3 \wedge x_3 - x_2 \le 0$

$\gamma_1(1) = 1, \gamma_1(2) = \perp, \gamma_1(3) = 6$

$\gamma_2(1) = 4, \gamma_2(2) = 6, \gamma_2(3) = \perp$

$\gamma_3(1) = 5, \gamma_3(2) = 11, \gamma_3(3) = 11$

$\gamma_4(1) = 1, \gamma_4(2) = \perp, \gamma_4(3) = 12$

Fig. 2. A path of the bus-contending protocol

that the two senders both send messages to the bus. After detecting the collision, A_3 retransmits its message.

Given a symbolic trace $\zeta_0 \xrightarrow{\gamma_1} \zeta_1 \xrightarrow{\gamma_2} \cdots \xrightarrow{\gamma_n} \zeta_n$, we may write $\zeta_0 \rightsquigarrow \zeta_n$. We borrow the techniques of generating a test case from the a symbolic trace from [16]. Given a symbolic trace of a CTA, we can extract a test case in TTCN format [9] for testing the CTA. A test case consists of a sequence of input events and expected output events annotated with symbolic timing-constraints between events. The global transitions in a symbolic trace correspond to the events in the test case. The zone descriptions in between transitions correspond to timing constraints between events. For the second sender process in figure 2, we can convert the path in figure 2 to the following test case in TTCN format.

(1)	START Time	/*start a clock called Time*/
(2)	?begin	/*receive a "begin" message*/
(3)	READTIMER Time(t_1)	/*store current reading of Time in t_1*/
(4)	!collision	/*send a "collision" message*/
(5)	READTIMER Time(t_2)	/*store current reading of Time in t_2*/
(6)	$[t_2 - t_1 < 52]$	/*check if $t_2 - t_1 < 52$*/
(7)	?begin	/*receive a "begin" message*/
(8)	READTIMER Time(t_3)	/*store current reading of Time in t_3*/
(9)	$[t_3 - t_2 < 52]$	/*check if $t_3 - t_2 < 52$*/

There is a global clock called `Time`. There are two TTCN commands, `START` (to start the ticking of a clock from zero) and `READTIMER()` (to read the current reading of a clock and save the reading in a variable). The test case can be used to check if a sender retries the transmission within 52 time unit after a bus collision is detected. Given a symbolic trace ρ, we denote the test case constructed out of ρ as $\texttt{testcase}(\rho)$. A symbolic trace in $\texttt{ZoneForest}(A, \phi)$ is *root-to-leaf* if its first zone is a root and its last zone is a leaf.

7 Prioritized test plan generation

Based on the materials in the last two sections, we can now define a set of test cases out of the `ZoneForest`. Given a CRTS A and a safety predicate ϕ, we denote the set of root-to-leaf symbolic traces in $\texttt{ZoneForest}(A, \phi)$ as $\texttt{Rt2Lves}(A, \phi)$. We let $\texttt{TestCases}(A, \phi)$ be the following set.

$$\texttt{TestCases}(A, \phi) = \{\texttt{testcase}(\rho) \,|\, \rho \in \texttt{Rt2Lves}(A, \phi)\,\}$$

A test plan that executes test cases $\theta_1, \ldots, \theta_n$ in sequence can be denoted as sequence $\theta_1 \ldots \theta_n$.

Different test plans may increase the AZC value with different efficiencies. In this section, we want to generate test plans that increase the AZC value with the maximum efficiency. Given a symbolic trace $\rho = \zeta_1\gamma_2\zeta_2\gamma_3 \ldots \zeta_n$, we let the disjunction of the zones in ρ be denoted as $\texttt{PathCons}(\rho) = \zeta_1 \vee \ldots \vee \zeta_n$. Also for convenience, given a symbolic trace ρ and a test case such that $\texttt{testcase}(\rho) = \theta$, we also write $\texttt{PathCons}(\theta) = \zeta_1 \vee \ldots \vee \zeta_n$. Given two sequences Θ and Θ', their concatenation is denoted as $\Theta\Theta'$. We have the following procedure for test plan generation.

```
(1)  TestPlan($A, \phi$) {
(2)    $\Phi := \texttt{Rt2Lves}(A, \phi)$; let $\Theta$ be an empty sequence; $\psi := \mathit{false}$.
(3)    While $\Phi \neq \emptyset$, {
(4)      Pick a $\rho \in \Phi$ such that for all $\rho' \in \Phi$,
(5)        $\texttt{AZC}(\texttt{PathCons}(\rho) \vee \psi) \geq \texttt{AZC}(\texttt{PathCons}(\rho') \vee \psi)$.
(6)      $\Phi := \Phi - \{\rho\}$; $\Theta := \Theta\texttt{testcase}(\rho)$; $\psi := \psi \vee \texttt{PathCons}(\rho)$.
(7)    }
(8)    return $\Theta$;
(9)  }
```

We can prove that the returned test plan of $\texttt{TestPlan}(A, \phi)$ can reach all the active zones that are reachable from the initial states.

Lemma 3. *For any CTA A and safety predicate ϕ, procedure* $\texttt{TestPlan}(A, \phi)$ *terminates. When it terminates, it returns a test plan that traverses all active zones that are reachable from the initial states of A.*
Proof : Note that in procedure $\texttt{ZoneForest}(A, \phi)$, each node is added to the tree if it has some regions that are not covered by other nodes. This means that at each iteration of the loop at statement (3) in procedure $\texttt{TestPlan}()$, we cover some new regions that have not been covered before. Since the number of regions is finite, we know the loop at statement (3) eventually terminates.

Also when the loop terminates, we have added all test cases in $\texttt{TestCases}(A, \phi)$ to Θ. Since the test cases are derived from all root-to-leaf symbolic traces that together cover all reachable active zones, we know that the test plan also covers all reachable active zones. ■

Note that this procedure, $\texttt{ZoneForest}(A, \phi)$, may not generate the shortest test plan in achieving an AZC threshold. But it works with a greedy heuristic to try to get the best AZC gain in every decision point. The wisdom from NP problems [10] is that in practice, very often such a heuristics may still lead to acceptable solutions for difficult problems like set cover and bin packing.

8 Experiment

We have implemented our ideas in **RED** [26, 28, 31], a model-checker/simulator for CTAs and linear hybrid systems based on a BDD-like diagram called Clock-Restriction Diagram. We have experimented with the Philips Audio Protocol [13, 18] and Bluetooth L2CAP [11, 32]. Due to page-limit, we only report the experiment with Bluetooth L2CAP. Experiment data is collected on a 3.2 GHz PC running Red Hat Linux 9.0. In subsection 8.1, we first discuss the performance of procedure $\texttt{ZoneForest}()$ for AZC gain with respect to the following three exploration strategies: DFES (depth-first exploration strategy), BFES

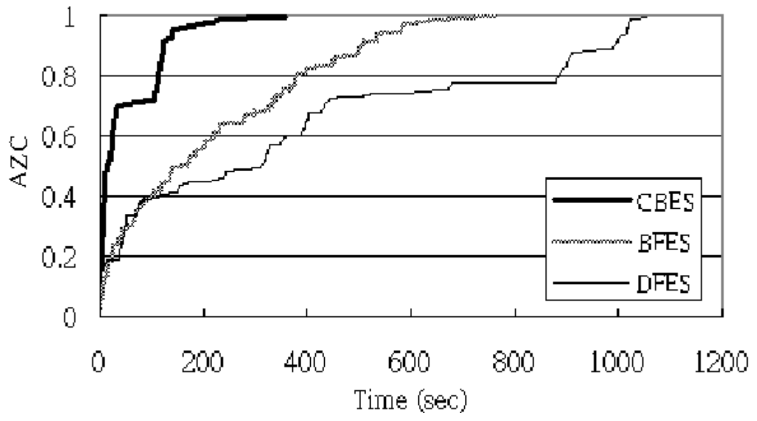

Fig. 3. AZC growth rate in symbolic state space exploration

Strategy	CPU time(sec)	Memory(Kbyte)	# Symbolic trace
DFES	1055	7411	1957
BFES	762	6378	1815
CBES	359	4920	1610

Table 1. Experiment result of symbolic state space exploration

(breadth-first exploration strategy), and CBES (AZC coverge-based exploration strategy). Then in subsection 8.2, we report the improvement in AZC coverage growth rate with procedure `TestPlan`().

8.1 Coverage estimations for strategies in procedure `ZoneForest()`

We now report our experiment with procedure `ZoneForest`(). We applied DFES, BFES, and CBES to explore the whole active zone space of the Bluetooth L2CAP specification. In figure 3, we show the growth curves in AZC values with strategies DFES, BFES, and CBES. The AZC value grows much faster with CBES.

Table 1 shows the performance statistics of procedure `ZoneForest`(). "CPU time" is the time spent on exploring the whole active zone space in seconds. "Memory" is the memory used by the procedure in kilobytes. "# Symbolic trace" is the number of symbolic traces in the constructed trees of active zones. The data shows that CBES outperforms DFES and BFES against the L2CAP benchmark in the efficiency of AZC gain. Also, CBES uses the least resources (in both time and memory) and generated the fewest symbolic traces.

In summary, the experiment data shows that CBES could be promising in helping us achieving high coverage in state space exploration with limited resources.

`TestPlan()` \ `ZoneForest()`	DFES	BFES	CBES
Not prioritized	DFES_no	BFES_no	CBES_no
Prioritized	DFES_prior	BFES_prior	CBES_prior

Table 2. Six test plans and their strategy combinations

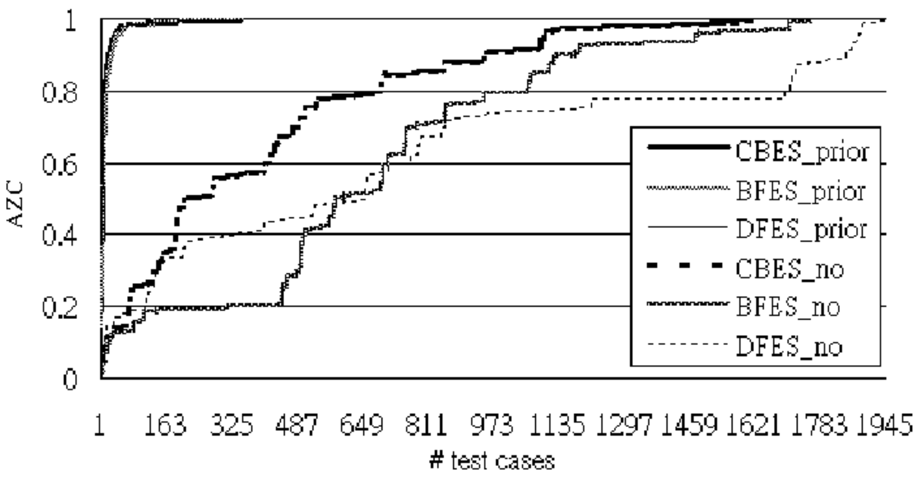

Fig. 4. AZC growth rate of the six test plans

8.2 Coverage growth rates in procedure `TestPlan()`

In this subsection, we report our experiment with procedure `TestPlan()`. In table 2, we show six test plans, denoted as DFES_no, BFES_no, CBES_no, DFES_prior, BFES_prior, and CBES_prior. The six plans come from the combination of the choice of strategies in procedure `ZoneForest()` and the two options in generating the test plans. DFES, BFES, and CBES are the choices for strategies in procedure `ZoneForest()`. With plans DFES_no, BFES_no, and CBES_no, we generate test plans in the 2nd step by simply enumerating the symbolic traces according to the depth-first order of the leaves of the traces in the zone forest. With DFES_prior, BFES_prior, and CBES_prior, we prioritized all symbolic traces in the 2nd step according to their AZC gain estimations.

To reduce the testing cost, it is desirable to achieve high coverage with few test cases. In figure 4, we show the growth curves of AZC values for the six test plans. DFES_prior, BFES_prior, and CBES_prior achieved 100% in AZC respectively after the 393rd, 413th, and 407th test cases were applied. The growth curves in AZC of the prioritized test plans with DFES_prior, BFES_prior, and CBES_prior, are similar in shape. In fact, in the chart, their curves almost coincide and reach the full coverage quickly.

On the other hand, with the non-prioritized test plans, including DFES_no, BFES_no, and CBES_no, the coverage curves grow much slower. Moreover, all test cases have to be executed in order to reach the full AZC. Since there are 1957 test cases for DFES_no, 1815 for BFES_no, and 1610 for CBES_no, significant testing time could be saved by using the prioritized test plans instead of the non-prioritized plans. The experiment data shows good promise of our techniques for the testing of CRTS.

Finally, test plan DFES_prior uses the least number of test cases because the test cases in DFES_prior tend to be longer than those in the other test plans.

9 Conclusion

In this paper, we propose techniques to model CRTS, to evaluate the progress of test execution with AZC, and test plan generation that gives priority to AZC gain. The experiment shows that the prioritized test plans can achieve high coverage more efficiently than the non-prioritized ones. Knowledge obtained in this work does point out some new directions for future research. For example, we assumed that the execution of every test case incurs the same cost (for examples, budget or time). In practice, some test cases may cost more than the others. It would be interesting to see how the factor of test case execution cost could be incorporated into our framework so that our test plan generation procedure could help in the accurate management of the verification budget.

Acknowledgment

The authors would like to thank the reviewers of TESTCOM/FATES 2008 for their helpful suggestions for improving this paper.

References

1. R. Alur, C. Courcoubetis, D.L. Dill. Model Checking for Real-Time Systems. IEEE LICS, 1990.
2. 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.
3. G. Behrmann, A. Fehnker T.S. Hune, K.G. Larsen, P. Pettersson, J. Romijn. Efficient Guiding Towards Cost-Optimality in UPPAAL. TACAS 2001, LNCS 2031, Springer, 2001.
4. G.V. Bochmann, A. Petrenko. Protocol Testing: Review of Methods and Relevance for Software Testing. Proceedings of the 1994 ACM SIGSOFT international symposium on Software testing and analysis.
5. E. Clarke, E.A. Emerson, Design and Synthesis of Synchronization Skeletons using Branching-Time Temporal Logic. Proceedings of the Workshop on Logic of Programs, LNCS 131, Springer-Verlag.

6. 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.
7. A. En-Nouaary, R. Dssouli, F. Khendek. Timed Wp: Testing Real-time Systems. IEEE Transactions on Software Engineering, Vol. 29, No. 11, IEEE Computer Society, (2002).
8. S. Elbaum, A. G. Malishevsky, G. Rothermel Test Case Prioritization: A Family of Empirical Studies IEEE Transactions on Software Engineering archive Volume 28 , Issue 2, 2002
9. ETSI: Methods for Testing and Specification (MTS) The Testing and Test Control Notation version 3. ETSI ES 201873, parts 1 to 7, v3.1.1, 2005-06.
10. M.R. Garey, D.S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, W. H. Freeman, New York, 1983.
11. J. Haartsen. Bluetooth Specification, version 1.0. http://www.bluetooth.com/.
12. A. Hessel, K.G. Larsen, B. Nielsen, P. Pettersson, A. Skou. Time-Optimal Real-Time Test Case Generation Using Uppaal. FATES 2003.
13. P.-H. Ho, H. Wong-Toi. Automated Analysis of an Audio Control Protocol. CAV 1995, LNCS 939, Springer Verlag, 1995.
14. T.A. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. Symbolic Model Checking for Real-Time Systems. IEEE LICS 1992.
15. 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.
16. G.-D. Huang, F. Wang. Automatic Test Case Generation with Region-Related Coverage Annotations for Real-time Systems. The Third International Symposium on Automated Technology for Verification and Analysis, October 2005, LNCS 3707, Springer Verlag.
17. M. Krichen, S. Tripakis. Black-box Conformance Testing for Real-Time Systems. In SPIN'04 Workshop on Model Checking Software.
18. K.G. Larsen, P. Pettersson, W. Yi. Diagnostic Model-Checking for Real-Time Systems. In Proceedings of the 4th DIMACS Workshop on Verification and Control of Hybrid Systems, New Brunswick, New Jersey, 22-24 October, 1995.
19. D. Lee, M. Yannakakis. Principles and Methods of Testing Finite State Machines - A Survey. Proceedings of The IEEE, Vol. 84, No. 8, August 1996, pp. 1090-1123.
20. B. Nielsen, A. Skou. Automated Test Generation from Timed Automata. International Journal on Software Tools for Technology Transfer (STTT), 4, 2002.
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. A. Shaw. Communicating Real-time State Machines. IEEE Transactions on Software Engineering 18(9), September, 1992.
24. A. Srivastava and J. Thiagarajan. Effectively prioritizing tests in development environment. Proceedings of the 2002 ACM SIGSOFT international symposium on Software testing and analysis.
25. J. Springintveld, F. Vaandrager, P.R. D'Argenio Testing Timed Automata. Theoretical Computer Science, Vol. 254, Issue 1-2, 2001.
26. F. Wang. Symbolic Verification of Complex Real-Time Systems with Clock-Restriction Diagram, in Proceedings of FORTE, August 2001, Cheju Island, Korea.

27. F. Wang. Efficient Verification of Timed Automata with BDD-like Data-Structures, STTT (Software Tools for Technology Transfer), Vol. 6, Nr. 1, June 2004, Springer-Verlag; special issue for the 4th VMCAI, Jan. 2003, LNCS 2575, Springer-Verlag.
28. F. Wang. Symbolic Parametric Safety Analysis of Linear Hybrid Systems with BDD-like Data-Structures. IEEE Transactions on Software Engineering, Volume 31, Issue 1 (January 2005), pp. 38-51, IEEE Computer Society. A preliminary version is in proceedings of 16th CAV, 2004, LNCS 3114, Springer-Verlag.
29. E.J. Weyuker. In Defense of Coverage Criteria. in Proceedings of the 11th ACM/IEEE International Conf. on Software Engineering(ICSE), Pittsburgh, Pa, May 1989.
30. E.J. Weyuker. How to Judge Testing Progress. Journal of Information and Software Technology. Volume/Number:Vol. 45(5). Pages:323-328 (2004).
31. F. Wang, G.-D. Huang, F. Yu. Symbolic Simulation of Real-Time Concurrent Systems. RTCSA2003, LNCS 2968, Springer-Verlag.
32. F. Wang, G.-D. Huang, Fang Yu. Numerical Coverage Estimation for Dense-Time Systems. in proceedings of FORTE'2003, LNCS 2767, Springer-Verlag.
33. H. Wong-Toi. Symbolic Approximations for Verifying Real-Time Systems. Ph.D. dissertation, Stanford Univ., Stanford, CA, 1995.
34. 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.