Automated Test Generation from Timed Automata

Brian Nielsen and Arne Skou

Aalborg University
Department of Computer Science
Fredrik Bajersvej 7E
DK-9220 Aalborg, Denmark
Email: {bnielsen | ask}@cs.auc.dk

Abstract

Testing is the most dominating validation activity used by industry today, and there is an urgent need for improving its effectiveness, both with respect to the time and resources for test generation and execution, and obtained test coverage. We present a new technique for automatic generation of real-time black-box conformance tests for non-deterministic systems from a determinizable class of timed automata specifications with a dense time interpretation. In contrast to other attempts, our tests are generated using a coarse equivalence class partitioning of the specification. To analyze the specification, to synthesize the timed tests, and to guarantee coverage with respect to a coverage criterion, we use the efficient symbolic techniques recently developed for model checking of real-time systems. Application of our prototype tool to a realistic specification shows promising results in terms of both the test suite size, and the time and space used for test generation.

1 Introduction

Testing consists of executing a program or a physical system with the intention of finding undiscovered errors. In typical industrial projects, as much as a third of the total development time is spent on testing, and it therefore constitutes a significant portion of the cost of the product. Since testing is the most dominating validation activity used by industry today, there is an urgent need for improving its effectiveness, both with respect to the time and resources for test generation and execution, and obtained coverage.

A potential improvement that is being examined by researchers is to make testing a formal method, and to provide tools that automate test case generation and execution. This approach has experienced some level of success: Formal specification and automatic test generation are being applied in practice and commercial test generations tools are emerging. Typically, a test generation tool inputs some kind of finite state machine description of the behavior required of the implementation. A formalized *implementation relation* describes exactly what it means for an implementation to be correct with respect to a specification. The tool interprets the specification or transforms it to a data structure appropriate for test generation, and then computes a set of test sequences. Since exhaustive testing is generally infeasible, it must select only a subset of tests for execution.

However, these tools do not address real-time systems, or only provide a limited support of testing the timing aspects. They often abstract away the actual time at which events are supplied or expected, or does not select these time instances thoroughly and systematically. We present a new technique for automatic generation of timed tests from a restricted (but possibly non-deterministic) class of dense timed automata specifications. We select test cases by partitioning the state space into coarse grained equivalence classes which preserve essential timing and deadlock information, and select a few tests for each class. To ensure coverage of the equivalence classes and to construct the tests, we employ recently developed efficient symbolic reachability techniques based on constraint solving for model checking of timed automata. Our techniques are implemented in a prototype tool, RTCAT.

2 Related Work

Springintveld et al. proved in [14] that *exhaustive* testing wrt. trace equivalence of deterministic timed automata with *dense time* is theoretically possible, but highly infeasible. Another result generating checking sequences for a discretized deterministic timed automaton is presented by En-Nouaary et al. in [9]. Although the required discretiza-

tion step size in [9] is larger than in [14], it still appears to be too small for most practical applications because too many tests are generated. Both of these techniques are based on the so-called *region* graph technique due to Alur and Dill [1]. Clock regions are very fine-grained equivalence classes of clock valuations. We argue that coarser partitions are needed in practice.

Clarke and Lee [6] propose domain testing for real-time systems. Their technique appear to produce much fewer tests than region based generation. The time requirements are specified as directed acyclic graphs called *constraint graphs*. Compared to timed automata this specification language appear very restricted, e.g., since their constraint graphs must be acyclic this only permits specification of finite behaviors. Their domains are "nice" linear intervals which are directly available in the constraint graph. In our work they are (convex) polyhedra of a dimension equal to the number of clocks. Test generation from *discrete* timed systems is reported by [5], from timed Petri nets by [4], and from a discrete time temporal logic by [11].

3 Hennessy Tests

In Hennessy's testing theory [12] specifications $\mathcal{S}$ are defined as finite state labelled transition systems over a given finite set of actions Act. Also, it assumes that implementations $\mathcal{I}$ (and specifications) can be observed by finite tests $\mathcal{T}$ via a sequence of synchronous CCS-like communications. Hennessy tests have the following abstract syntax $\mathcal{L}_{\mathrm{tlts}}$: (1) $\mathbf{after}\ \sigma\ \mathbf{must}\ A$, (2) $\mathbf{can}\ \sigma$, and (3) $\mathbf{after}\ \sigma\ \mathbf{must}\ \emptyset$, where $\sigma \in Act^*$ and $A \subseteq Act$. Informally, (1) is successful if at least one of the observations in A (called a *must set*) can be observed whenever the trace σ is observed, (2) is successful if σ is a prefix of the observed system, and (3) is successful is this is not the case (i.e. σ is not a prefix). To facilitate and ease systematic generation of all relevant tests, the specification can be converted to a trace equivalent *deterministic* state machine whose states are labelled with the must sets for that state (similar to the acceptance graph of [7]). We propose a simple timed generalization of Hennessy's tests. In a timed test $\mathbf{after}\ \sigma\ \mathbf{must}\ A$ (or $\mathbf{after}\ \sigma\ \mathbf{must}\ \emptyset$), σ becomes a timed trace (a sequence of alternating actions and time delays), after which an action in A must be accepted immediately. A test $\mathbf{can}\ \sigma$ ($\mathbf{after}\ \sigma\ \mathbf{must}\ \emptyset$) becomes a timed trace satisfied if σ is (is not) a prefix trace of the observed system. A test will be modelled by an executable timed automaton whose locations are labelled with pass, fail, or inconclusive verdicts.

4 Event Recording Automata

Two of the surprising undecidability results from the theoretical work on timed languages described by timed automata is that 1) a non-deterministic timed automaton cannot in general be converted into a deterministic (trace) equivalent timed automaton, and 2) trace (language) inclusion between two non-deterministic timed automata is undecidable [2]. Thus, unlike the untimed case, deterministic and non-deterministic timed automata are not equally expressive. The Event Recording Automata model (ERA) was proposed by Alur, Fix, and Henzinger in [2] as a determinizable subclass of timed automata, which enjoys both properties.

Definition 1 *Event Recording Automaton*:

1. An ERA $\mathcal{M}$ is a tuple $\langle Act, N, l_0, E\rangle$ where Act is the set of actions, N is a (finite) set of locations, $l_0 \in N$ is the initial location, and $E \subseteq N \times G(X) \times Act \times N$ is the set of edges. The term *location* denotes a node in the automaton, and the term *state* denotes the semantic state of the automaton also including clock values.

2. $X = \{x_a \mid a \in Act\}$ is the set of clocks. The guards $G(X)$ are generated by the syntax $g ::= \gamma \mid g \wedge g$ where γ is a constraint of the form $x_1 \sim c$ or $x_1 - x_2 \sim c$ with $\sim \in \{\leq, <, =, >, \geq\}$, c a non-negative integer constant, and $x_1, x_2 \in X$.

Like a timed automaton, an ERA has a set of clocks which can be used in guards on actions, and which can be reset when an action is taken. In ERAs however, each action a is uniquely associated with a clock x_a, called the *event clock* of a. Whenever an action a is executed, the event clock x_a is automatically reset. No further clock assignments are permitted. The event clock x_a thus *records* the amount of time passed since the last occurrence of a. In addition, no internal τ actions are permitted. These restrictions are sufficient to ensure determinizability [2].

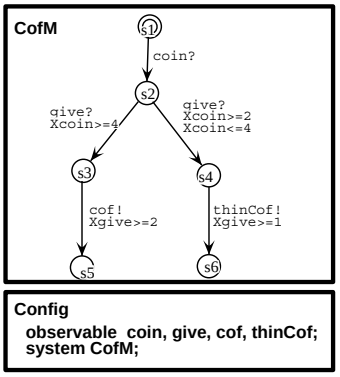

Figure 2. ERA specification of a coffee vending machine.

Figure 2 shows a small ERA which models a coffee vending machine built for impatient users such as busy researchers. When the user has inserted a coin (`coin`), he must press the give button (`give`) to indicate his eager to get a drink. If he is very eager, he presses `give` soon after inserting the coin, and the vending machine outputs thin coffee (`thinCof`); apparently, there is insufficient time to brew good coffee. If he waits more than four time units, he is certain to get good coffee (`cof`). If he presses `give` after exactly four time units, the outcome is non-deterministic.

5 A Test Generation Algorithm

Since exhaustive testing is generally infeasible, it is important to systematically select and generate a limited amount of tests. A test *selection criterion* (or coverage criterion) is a rule describing what behavior or requirements should be tested. *Coverage* is a metric of completeness with respect to a test selection criterion.

We propose a criterion based on partitioning the state space of the specification into coarse equivalence classes, and requiring that the test suite for each class makes a set of required observations of the implementation when it is expected to be in a state in that class. These observations are used to increase the confidence that the equivalence classes are correctly implemented. The partitioning and observations can be done in numerous ways, and some options are explored and formally defined in [13]. Given the partitioning stated in the following, the *stable edge set criterion* implemented in RTCAT requires that all relevant *simple deadlock* observations of the forms $\mathbf{after}\ \epsilon\ \mathbf{must}\ A$ (a *must* property), $\mathbf{after}\ a\ \mathbf{must}\ \emptyset$ (a *refusal* property), and $\mathbf{can}\ a$ (a *may* property) are made at least once in each class.

From each control location L (recall that a location in a deterministic automaton is the set of locations of the original automaton that the automaton can possibly occupy after a given trace), the clock valuations are partitioned such that two clock valuations belong to the same equivalence class iff they enable precisely the same edges from L, i.e. the states are equivalent wrt. the enabled edges. This partitioning is based on the guards that actually occur in a specification, and is therefore much coarser than e.g., the region partitioning which is based on the guards that could possibly occur in an automaton according to the syntax in Definition 1. It also has the nice formal property that the states in the same equivalence class are also equivalent with respect the previously stated *simple deadlock* properties.

The test generation procedure is outlined in Algorithm 3. It first constructs the equivalence classes by a data structure which we refer to as the *equivalence class graph*. In this process it implicitly determinizes the specification. The equivalence class graph preserves all timed traces of the specification, and furthermore preserves the required deadlock information for our timed Hennessy tests of the specification by the M, C, and R action sets stored in each node. All timed Hennessy tests that the specification passes can thus be generated from this graph. Reachability analysis is needed to select only states for testing that are actually reachable, and to compute a timed trace to the target state. Densely timed automata cannot be analyzed by enumerative finite state techniques, but must rather be analyzed symbolically [1]. Efficient symbolic techniques have been developed for model checking of timed automata. Specifically, we employ the so-called *zone* and *difference bound matrix* techniques [8] like those developed for the UPPAAL tool [10].

Algorithm 3 *Overall Test Case Generation Algorithm*:
input: ERA specification $\mathcal{S}$.
output: A complete covering set of timed Hennessy tests.

1. Compute $\mathcal{S}_p = \text{Equivalence Class Graph}(\mathcal{S})$.
2. Compute $\mathcal{S}_r = \text{Reachability}(\mathcal{S}_p)$.
3. Label every symbolic State $S \in \mathcal{S}_r$ with the sets M (the minimized set of must sets), C (the possible actions in the state not contained in M), R (the actions that must be refused).
4. Traverse $\mathcal{S}_r$. For each reached equivalence class S in $\mathcal{S}_r$:
   (a) Choose a state to be tested $s \in S$
   (b) Compute a timed trace σ from initial state to s.
   (c) Make test cases to be passed:
   if $A \in M(S)$ then $\mathbf{after}\ \sigma\ \mathbf{must}\ A$ is a test.
   if $a \in C(S)$ then $\mathbf{can}\ \sigma \cdot a$ is a test.
   if $a \in R(S)$ then $\mathbf{after}\ \sigma \cdot a\ \mathbf{must}\ \emptyset$ is a test.

6 Experimental Results

Figure 4 shows some examples of generated test cases from the coffee machine specification in Figure 2. RTCAT has been configured to select test points in the interior of the equivalence classes, and to use breadth first state exploration.

To analyze the feasibility of our techniques we have created an ERA version of the frequently studied Philips audio protocol [3] and a simple token passing protocol, applied RTCAT, and measured the number and length of the generated tests, the number of reached (convex) equivalence classes and symbolic states, and the space and time needed to generate the tests and output them to a file. The ERA models can be found in [13]. The platform used in the experiment consists of a Sun Ultra-250 workstation running Solaris 5.7. The machine is equipped with 1 GB RAM and 2x400 MHz CPU's. The results for coffee machine (`CofM`), the Philips audio protocol receiver component (`phil(R)`), sender component (`Phil(S)`) with collision detection, and 7-node token passing protocol ($\texttt{tok}^7$) are shown in Table 5.

The size of the produced test suites is in all cases quite manageable, and constitute test suites that could easily be

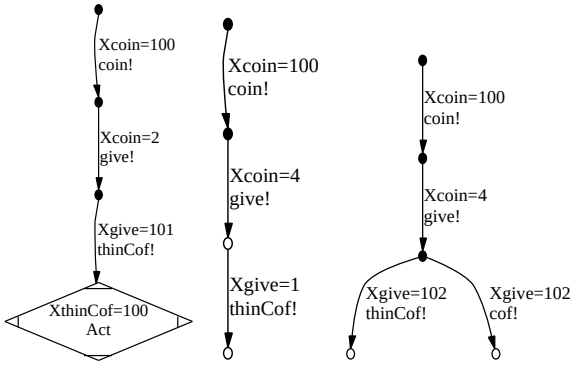

Figure 4. Example tests generated from the coffee machine in Figure 2 (○=pass, •=fail, ◊=refusal, Act is all actions.

	CofM	Phil(R)	Phil(S)	Tok[7]
Equiv. Classes	14	60	59	42
Symbolic States	17	71	120	15427
Time (s)	1	1	2	541
Memory (MB)	5	5	5	40
No. of Tests	22	118	116	84
Total Length	58	614	622	665

Table 5. Experimental results.

executed in practice. There is thus a large margin allowing for more test points per equivalence class, or longer tests. Moreover, coverage of even larger specifications can also be obtained. For the first three specifications, the space and time consumption is quite low, and indicates that fairly large specifications can be handled. However, we have also encountered a problem with our current implementation which occurs for some specifications (such as the token passing protocol that uses a large set of active clocks), where our application of the symbolic reachability techniques becomes a bottleneck. It is important to note that the size of the produced test suite is still quite reasonable. We believe that this problem can be alleviated by applying the reachability analysis on the original specification automaton rather than presently done on the equivalence class graph. This should result in larger and fewer symbolic states.

7 Future Work

Much other work remain to be done. In particular we are examining the possibilities for generalizing our specification language. It will be important to allow specification and effective test of timing uncertainty, i.e., that an event must be produced or accepted at some (unspecified) point in an interval. Further, it should be possible to specify environment assumptions and to take these into account during test generation. Finally, our techniques should be examined with real applications, and the generated test should be executed against real implementations.

References

[1] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 25 Apr. 1994.

[2] R. Alur, L. Fix, and T. A. Henzinger. Event-Clock Automata: A Determinizable Class of Timed Automata. In *6th Conference on Computer Aided Verification*, 1994. Also in LNCS 818.

[3] D. Bosscher, I. Polak, and F. Vaandrager. Verification of an Audio Protocol. TR CS-R9445, CWI, Amsterdam, The Netherlands, 1994. Also in LNCS 863, 1994.

[4] V. Braberman, M. Felder, and M. Marré. Testing Timing Behaviors of Real Time Software. In *Quality Week 1997. San Francisco, USA.*, pages 143–155, April-May 1997 1997.

[5] R. Cardell-Oliver and T. Glover. A Practical and Complete Algorithm for Testing Real-Time Systems. In *5th international Symposium on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'98)*, pages 251–261, September 14–18 1998. Also in LNCS 1486.

[6] D. Clarke and I. Lee. Automatic Test Generation for the Analysis of a Real-Time System: Case Study. In *3rd IEEE Real-Time Technology and Applications Symposium*, 1997.

[7] R. Cleaveland and M. Hennessy. Testing Equivalence as a Bisimulation Equivalence. *Formal Aspects of Computing*, 5:1–20, 1993.

[8] D. L. Dill. Timing Assumptions and Verification of Finite-State Concurrent Systems. In *International Workshop on Automatic Verification Methods for Finite State Systems*, pages 197–212, Grenoble, France, June 1989. LNCS 407.

[9] A. En-Nouaary, R. Dssouli, and F. Khendek. Timed Test Cases Generation Based on State Characterization Technique. In *19th IEEE Real-Time Systems Symposium (RTSS'98)*, pages 220–229, December 2–4 1998.

[10] K. G. Larsen, F. Larsson, P. Petterson, and W. Yi. Efficient Verification of Real-Time Systems: Compact Data Structures and State-Space Reduction. In *18th IEEE Real-Time Systems Symposium*, pages 14–24, 1997.

[11] D. Mandrioli, S. Morasca, and A. Morzenti. Generating Test Cases for Real-Time Systems from Logic Specifications. *ACM Transactions on Computer Systems*, 13(4):365–398, 1995.

[12] R. D. Nicola and M. Hennessy. Testing Equivalences for Processes. *Theoretical Computer Science*, 34:83–133, 1984.

[13] B. Nielsen. *Specification and Test of Real-Time Systems*. PhD thesis, Department of Computer Science, Aalborg University, Denmark, april 2000. Submitted.

[14] J. Springintveld, F. Vaandrager, and P. D'Argenio. Testing Timed Automata. TR CTIT 97-17, University of Twente, 1997. To appear in *Theoretical Computer Science*.