Chapter 8

Test Generation from Timed Input Output Automata

The purpose of this chapter is to introduce techniques for the generation of test data from models of software based on variants of timed automata. The tests generated are intended to detect faults related to timing constraints, transitions among states, and operation errors along transitions.

8.1. Introduction

The purpose of this chapter is to introduce techniques for generating test cases for testing software in embedded systems so as to reveal errors in timing and communication. While there exist a variety of analysis techniques useful in detecting such errors prior to testing, our focus in this chapter is on dynamic techniques that actually test the software, often in its intended environment, to ensure that indeed the software behaves correctly as per its timing and communications requirements. In practice, tests derived using techniques described in this chapter would be augmented with tests derived using techniques discussed elsewhere in this book.

Real-time systems are often required to adhere to various forms of timing and resource, constraints, e.g. memory. Hard real-time systems are considered to have failed if, for example, a task deadline is missed. Soft real-time systems are tolerant of missed deadlines. Of course this distinction between hard and soft real-time systems is blurred in fault-tolerant systems. Even in such systems, a hard real-time system often triggers an error recovery procedure when a deadline is missed while a soft real-time system may tolerate a few occasionally missed deadlines.

The emergency controller used in a train collision avoidance system is an example of a hard real-time system. The routing mechanism for packets in a multimedia system is a soft real-time system. Missing a deadline in a hard real-time system might lead to a disaster, such

as deaths of civilians, while missing a deadline in a soft real-time system might cause some inconvenience or might even go unnoticed. Regardless of what kind of a real-time system a tester deals with, the goal of high quality will dictate that the timing requirements be tested and any errors reported to the management.

Real-time systems are often embedded systems. Examples include the engine controller in an automobile and the control unit inside a heart pacemaker. Such embedded systems include sensors to periodically sample environmental conditions, e.g. oxygen in the catalytic convertor of an automobile or environment temperature outside of an aircraft. The sensors sample, and perhaps process, data and send it to another processor which is often some hardware such as a microcontroller where it is processed and used in determining some control action, e.g. to control the pulse width of the fuel injector. Such interaction between various hardware devices within an embedded system often leads to software that consists of concurrent and communicating processes. In addition to the timing errors mentioned earlier, these communicating processes must also be tested for communication and other errors that may lead to race conditions and deadlocks.

While finite state and statechart models are quite common in modeling communication protocols and other real-time systems, they are often not well suited to the task of testing an IUT for timing errors and errors that result due to incorrect implementation of concurrency. Models based on variants of timed automata and Petri net are generally well accepted amongst practitioners to model timing, resource, and concurrency requirements of a real-time system. In this chapter we introduce a technique for generating tests from a variant of timed automata known as *timed input output automata*, or simply TIOA. Interestingly, several techniques proposed for generating tests from TIOA, and other variants of the timed automata, are adaptations of techniques for the generation of tests from finite state models discussed in Chapter 6. Hence these test generation techniques can also be classified as *automata theoretic* analogous to the ones introduced in Chapter 6.

We begin our exposition with an overview of the test methodology for the testing of a real-time system for conformance with the timing constraints required to be met. This is followed by a gentle introduction to timed automata, also referred to as TA. This introduction leads to the definition of a variant of TA known as timed input/output automata, also known as TIOA. Following this introduction we introduce the generalized Wp method to generate tests from TIOA. Examples are used to illustrate test generation and the detection of faults.

8.2. Overview of the test methodology

In this chapter we describe a procedure for generating tests from a formal specification of timing constraints in a real-time system. The test generation procedure is based on the timed Wp method and can be automated. The tests are generated from a formal specification expressed as a timed input/output automaton, also referred to as TIOA. Though timed Wp is a black box method in that it uses only the TIOA specification to generate tests, the testing procedure itself does need access to the code. Hence the overall test methodology is considered as a grey box.

The entire test methodology is illustrated in Figure 8.1. Given the informally expressed

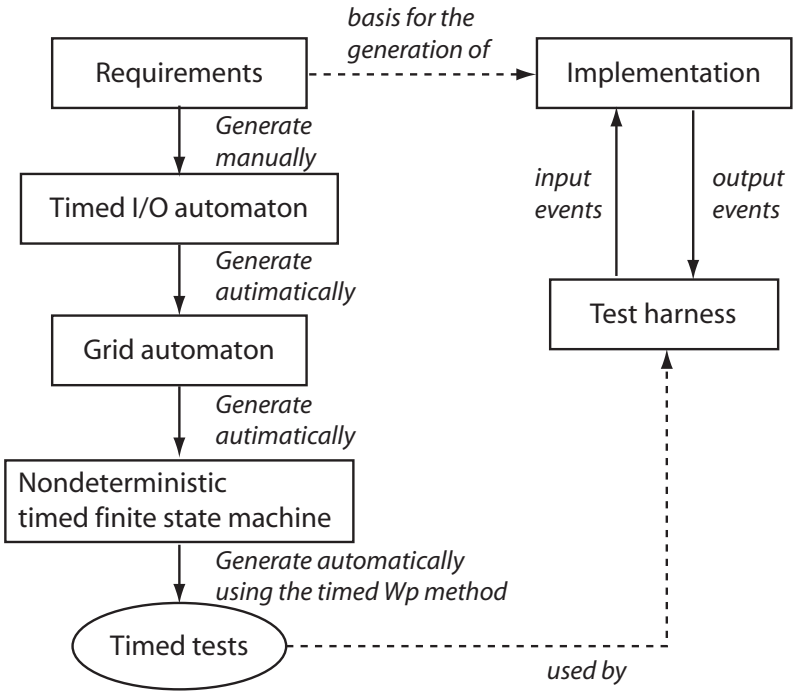

Figure 8.1: Steps in the generation of tests for timing constraints using the timed Wp method.

set of requirements, one extracts the timing constraints and expresses them in the form of a TIOA. This task is likely to be completed manually by design or test experts. The TIOA is then transformed into a grid automaton. A nondeterministic timed finite state machine, also referred to as NTFSM, is constructed from the TIOA. The tests, each being a sequence of delays and input events, are generated using the timed Wp method. For example, here is a sample timed test: $\frac{1}{4}.\frac{1}{4}.send.\frac{1}{4}.send.\frac{1}{4}$, where $\frac{1}{4}$ is time delay and $send$ is an input command that serves as an input event for the implementation. Except for the construction of the TIOA, all steps in this process can be automated using the algorithms described in this chapter.

The timed tests are available to the test harness. The harness, constructed manually, controls the implementation during the test. The implementation is derived, most likely manually, based on the available requirements. The goal of the tests generated using the method described is to ascertain whether or not the implementation satisfies the timing constraints imposed by the requirements.

The implementation may need to be modified for the purpose of providing the harness with information on its current state and the action performed. Hence the proposed test methodology falls under the grey box testing category. The harness generates the input events for the implementation to process. The input events are delayed in ways to test whether or not the implementation meets the timing constraints related to the input and output.

The TIOA model assumes an asynchronous processing of the input events by the implementation. However, by suitably modifying the test harness, synchronous processing can also be handled. For example, an application might require that inputs must arrive at specific time intervals to be processed. An input that does not arrive at its next expected time is ignored.

With the help of timers, the harness can be used to generate input events to occur periodically. As another example, an application might not impose any constraint on the arrival time of the input event but is required to ensure that the input is processed within a given time interval following its arrival. Again, with the help of timers, the harness can determine whether or not the application meets the response time requirement.

8.3. Timed automata

8.3.1. Informal introduction

A timed automata is an extension of finite state automata using *clocks*. We illustrate such an extension with respect to the transition diagrams in Figure 8.2. In this figure, M1 is an FSM (a Moore machine) with input alphabet $X = \{a, b, c\}$, set of states $Q = \{q_1, q_2\}$, an initial state q_1 which also serves as an accepting state. M1 starts in state q_0 and returns to its initial state after processing an input string in the set $(ab^*c)^*$. Thus, for example, the empty string, ac, abc, and $abbc$ will all bring M1 to its initial, and accepting, state. The language recognized by M1 is precisely the regular set $(ab^*c)^*$. Note that the empty string also belongs to the language accepted by M1.

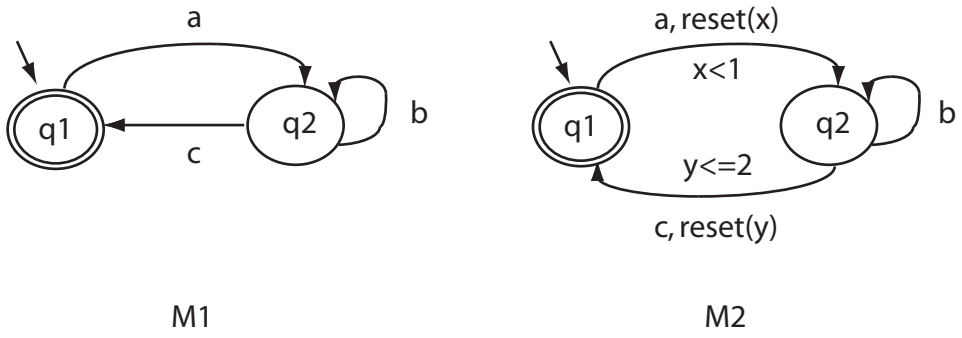

Figure 8.2: M1: A simple finite state model. M2: Finite state model M1 modified by the addition of clocks x and y and time constraints $x < 1$ and $y \leq 2$.

Machine M2 has the same input alphabet, set of states, the initial state and the final states as M1. However, the transitions in M2 have been labeled with clocks x and y, constraints on clocks, and the *reset* operation. x and y are assumed to be real-valued clocks and increment with the passage of time.

Both clocks are initialized to 0 when M2 is first started in state q_1. A clock increments until it is reset by a *reset* operation specified along a transition. Following $reset(x)$, clock x increments starting at 0. The $reset(x)$ operation is equivalent to the assignment $x := 0$. We assume that a guard along a transition Tr is evaluated *before* performing any reset operation associated with Tr. Thus, for example, the guard $x < 1$ along the (q_1, q_2) transition in Figure 8.2 is evaluated prior to resetting clock x.

While the behavior of an FSM is independent of the time of arrival of the next input, arrival of inputs in M2 must be associated with time. Association of time with an input is necessary

to determine the response of M2 to an input. Inputs to M2 are also known as *events*. Thus the following two statements are equivalent: "Input a arrives at time 0.3" and "Event a occurs at time 0.3." An input sequence with arrival times specified, is also known as a *timed input sequence*.

It is assumed that transitions in M1 and M2 occur instantaneously, i.e. require zero time. Thus while M2 might remain in a state for an infinite amount of time, when it moves to its next state it does so in zero time. The next example illustrates the behavior of M2 for several timed input sequences.

EXAMPLE 8.1. Suppose that the event sequence $E_1 = abc$ arrives at M2 in the following time sequence.

Event	*Time of arrival*
a	0.8
b	0.9
c	1.7

The notation

$$q_i \xrightarrow[t]{\texttt{a}} q_j$$

denotes the transition of a state machine from state q_i to q_j upon the arrival of event a at time t; q_i and q_j might be the same state. The behavior of M2 in response to E_1 is shown below in terms of the state transitions.

	q_1	$\xrightarrow[0.8]{\texttt{a}}$	q_2	$\xrightarrow[0.9]{\texttt{b}}$	q_2	$\xrightarrow[1.7]{\texttt{c}}$	q_1
x	0		0		0.1		0.9
y	0		0.8		0.9		0

It is assumed that time starts at 0 when M2 is first initialized. The second and third rows in the table above list the values of clocks x and y, respectively. Both clocks start at 0 when M2 is initialized to state q_1. The clocks move forward with the passage of time until they are reset to 0. Upon the return of M2 to state q_1, clock x is at 0.9 while clock y is at 0 because it is reset during the previous transition from state q_2 to q_1.

The time elapsed since the start of the machine is determined from the top row of the table above. For example, a total of 0.9 time units have elapsed upon the second entry into state q_2. Also, a total of 1.7 time units have elapsed when M2 returns to state q_1.

Next, consider the arrival times of the event sequence $E_2 = abcac$ which has the sequence E_1 as its prefix.

Event	*Time of arrival*
a	0.8
b	0.9
c	1.7
a	1.75
c	2.6

The response of M2 to E_2 is shown in the table below.

	q_1	$\xrightarrow[0.8]{\texttt{a}}$	q_2	$\xrightarrow[0.9]{\texttt{b}}$	q_2	$\xrightarrow[1.7]{\texttt{c}}$	q_1	$\xrightarrow[1.75]{\texttt{a}}$	q_2	$\xrightarrow[2.6]{\texttt{c}}$	q_1
x	0		0		0.1		0.9		0		0.85
y	0		0.8		0.9		0		0.05		0

Next, consider $E_3 = E_2 = abcac$ but with the following arrival times.

Event	*Time of arrival*
a	0.8
b	0.9
c	1.7
a	2.10
c	2.6

The response of M2, shown below, is now different because the arrival of the second a is too late and does not satisfy the clock constraint $x < 1$. Hence it is ignored and M2 gets "stuck" in state q_1. In an implementation of M2, the second occurrence of event a may be signaled as an error condition causing the violation of a timing constraint on clock x. Any subsequent event is also ignored as $x > 1$ and there is no *reset* operation in q_1.

	q_1	$\xrightarrow[0.8]{\texttt{a}}$	q_2	$\xrightarrow[0.9]{\texttt{b}}$	q_2	$\xrightarrow[1.8]{\texttt{c}}$	q_1	$\xrightarrow[2.1]{\texttt{a}}$	q_1	$\xrightarrow[2.6]{\texttt{c}}$	q_1
x	0		0		0.1		0.9		1.2		1.7
y	0		0.8		0.9		0		0.3		0.8

Lastly, consider $E_4 = abcac$ but with the following arrival times.

Event	*Time of arrival*
a	0.8
b	0.9
c	1.7
a	1.75
c	4.0

In this case M2 is stuck in state q_2 because, as shown below, event c arrives late and does not satisfy the constraint $y < 2$.

	q_1	$\xrightarrow[0.8]{\texttt{a}}$	q_2	$\xrightarrow[0.9]{\texttt{b}}$	q_2	$\xrightarrow[1.7]{\texttt{c}}$	q_1	$\xrightarrow[1.75]{\texttt{a}}$	q_2	$\xrightarrow[4.0]{\texttt{c}}$	q_2
x	0		0		0.1		0.9		0		0.75
y	0		0.8		0.9		0		0.05		2.25

Sequences E_3 and E_4 illustrate how the arrival times of events can cause M2 to behave differently for the same event sequence.