Chapter 7

Test Generation from Statechart Models

The purpose of this chapter is to introduce techniques for the generation of test data from statechart representations of requirements. After an introduction to the statechart notation we show step-by-step how automata theoretic techniques can be applied to generate tests from statecharts. Various kinds of statecharts, from simple containing no hierarchy, to complex, containing orthogonal components, are considered.

7.1. Statecharts

A statechart provides a graphical means for specifying the behavior of complex systems. The adjective "complex" is used for systems that are difficult or practically impossible to model using finite state machines. Aircraft engine controller, heart pacemaker, and autombille cruise control system, are some examples of "complex" systems that are modeled using statecharts due primarily to the complexity of their interactions with the environment and amongst their own components. Behavior of objects, such as those in a Java program, can also be modeled using statecharts.

The statechart notation is an extension of the transition diagrams used as a graphic representation of finite state machines. It represents the flow of control in the system modeled in response to external and internal events. Events could be occurrences in the environment in which case they are external to the system. For example, pressing the PLAY button on a VCR generates an external event. Events could also be generated by components within the system modeled. Such events are *internal* to the system. For example, TAPE_END event is generated by a VCR when it detects the end of a tape.

In this section we describe the elements of the statechart notation and show through examples the benefit of using statecharts over finite state machines for complex systems. First we describe the syntax of statecharts which is followed by a description of its semantics. There are several variants of the semantics of statecharts. There are two standards used in the industry.

One standard is the semantics as specified by the STATEMATE tool which is used widely for the design of real-time embedded systems. Another standard is the one specified in UML 2 which differs from that used in STATEMATE.

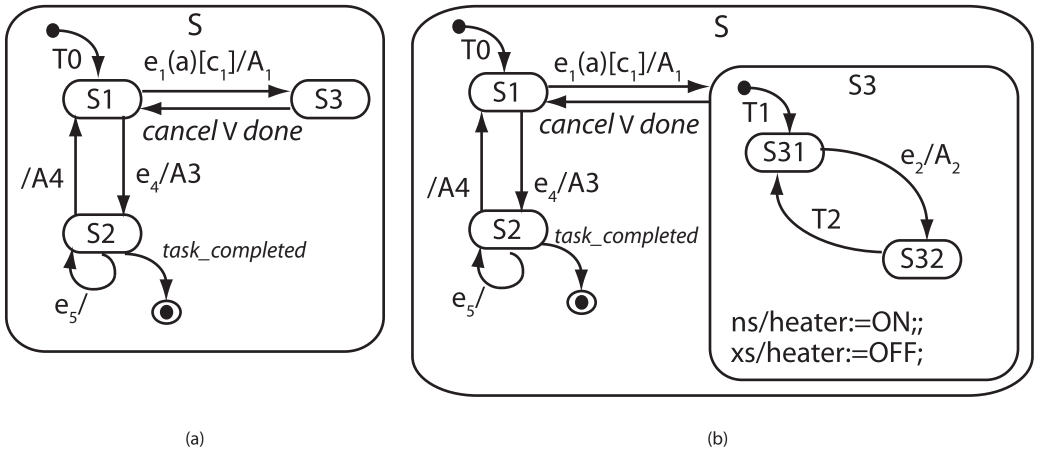

Figure 7.1: Two simple statecharts. (a) Three basic states. (b) Two basic and one composite state. The composite state is an OR state. S is a composite state in both statecharts. Note that ";;" is used to separate activities specified inside a state; inside S3 in this example.

7.1.1. States

A statechart consists of a collection of states connected via transitions. As we will see below, the concept of a state in statecharts is much more elaborate than in a finite state machine. States are represented by rounded rectangles and transitions by directed, and often labeled, arrows. Figure 7.1 shows two simple statecharts. The statechart in (a) contains three states named S1, S2, and S3. These three states are known as *basic* or *simple* states. A state that is composed of one or more *substates* is a *composite* state. State S3 in Figure 7.1(b) is a composite state. It is composed of states S31 and S32 which are also known as *substates* of state S3. We shall ruse the notation $substate(S)$ to refer to all direct substates of state S. For example, in Figure 7.1(a), $substate(S) = \{S1, S2, S3\}$. In Figure 7.1(b) $substate(S) = \{S1, S2, S3\}$ and $substate(S3) = \{S31, S32\}$.

Each state has five parts: a name, entry actions, exit actions, a do activity, a set of internal transitions, and a set of deferred transitions. In Figure 7.1, S3 is the name of a state. It has an entry activity indicated by `ns`. It also has an exit activity indicated by `xs`. A state may also specify a *do* activity as a sequence of operations. A do-activity can be triggered by an event in some state such as a change in the value of a variable, or an entry or exit from a state. We use the notation $ns(S)$, $xs(S)$, and $do(S)$ to denote, respectively, any entry, exit, and do activities associated with state S. These activities are also known as *static reactions*.

A composite state can be either an OR state or an AND state. For example, S3 in Figure 7.1(b) is an OR state while S5 in Figure 7.2 is an AND state. An AND state consists of two or more regions separated by a dotted line as shown. In our example, S5 is separated into two regions, one corresponding to state S3 and the other to state S4. Note that both S3 and S4 are composite states with their respective statecharts. Thus, S5 in Figure 7.2 is composed of two statecharts, one labeled S3 and the other labeled S4. Note that states S31 and S32 are *substates* of S3 and states S41 and S42 are substates of state S4. In this example, S4 is a *superstate* an contains S41 and S42 while S5 is another superstate that contains S3 and S4. We can also say that S3 and S4 are substates of S5. S3 and S4 are considered orthogonal states. While the substates of an OR state are sequential states, those of an AND state are concurrent. While only one substate of an OR state can be active at any time, all substates of an AND state are active simultaneously.

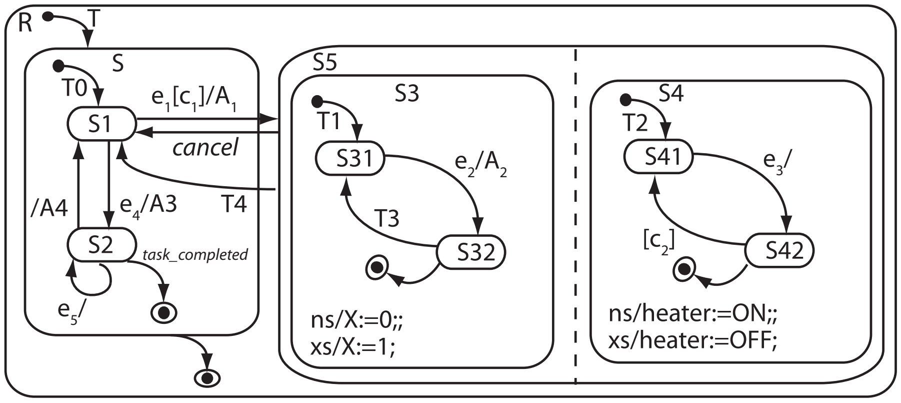

Figure 7.2: A statechart having a composite state with concurrency.

Each statechart has one special state known as the *initial* state or the *start* state. This state is also known as the *root state* of the statechart. It is indicated by a filled blob (●). For example, the transition to state S1 in Figure 7.1(a) has a initial state as its source and a default state S1 as its target. Note that we have another initial state in the statechart labeled S3. An initial state has no incoming transitions. There can be at most one initial state in a composite state. Similarly, a final state is one that has no outgoing transition. It is indicated by a filled blob surrounded by an unfilled circle (◉). For example, the transition going out of state S2 in Figure 7.1(a) has a final state as its destination.

7.1.2. Transitions

A transition specifies a relationship amongst two states. One of these two states is known as the source state for the transition and the other as its target or destination state. A transition is indicated by a a labeled arrow originating from its source state and terminating at its target state. A transition whose originating and terminating states are the same is known as *self-transition*. The transition labeled $e_5/$ in Figure 7.1(a) is a self-transition.

Note that the term "label" of a transition, as used here, is not to be confused with the name *of a transition such as T0, T1, and T2 used in Figure 7.2. Unlike a label, the name of a transition is purely for referencing purposes and has no effect on the actual semantics associated with that transition.*

The label of a transition may have up to three components: a trigger, a guard, and action. A trigger is any event; at most one event can be specified in the trigger. A guard is any Boolean expression without side effects. An action is a procedure, or a sequence of procedures to be executed when the transition fires. For example, in Figure 7.2, the transition from state S1 to state S5 has the label $e_1(a)[c_1]/A_1$. Here e_1 denotes an event trigger, a a parameter of the event, $[c_1]$ is the guard condition, and A_1 is an action. A guard is a Boolean predicate which, in conjunction with the trigger, decides when the transition fires. All components of a label are optional. Thus, for example, in Figure 7.2 the transition from state S32 to S31 has an empty label, the one from state S42 to state S41 has only a guard consisting of condition c_2, and another transition from state S5 to state S1 has only a trigger indicated as $cancel \vee done$. A transition without a trigger is known as a *completion* transition.

7.1.3. Pseudo states

Statecharts provide several types of pseudo states that are useful in combining transitions in various ways. The initial and final states are two pseudo states we have already introduced earlier. We introduce the remaining pseudo states in this section.

History: A history state allows entry into a non-default state inside a composite state. There are two types of history states: shallow history indicated by Ⓗ and deep history indicated by H^*. The shallow history state is a shorthand notation to represent the most recently visited substate in a composite state but not substate of a substate. Deep history state is a shorthand notation for the most recently visited substate or a state; this substate could be nested within other substates. History states are useful in expressing designs of systems that deal with interrupts. The next two examples illustrate the use of history states.

EXAMPLE 7.1. Let us examine Figure 7.3(a) to understand the effect of the shallow history state. Suppose that transition T1 is enabled and state S is entered. Notice that S is a compound OR state with a deep history state providing the default entry. Suppose also that this is the first time S has been entered. The default transition will be enabled and state S1 will be entered.

Suppose now that event e_1 occurs and state S2 entered. While S2 is active suppose that event e_3 occurs. Event e_3 represents an interrupt that causes control to move from the current state to a state where the interrupt is processed. This interrupt will cause S to become inactive and the target state of transition T2 will become active. Note that a guard can be also be attached to T2. This guard could serve to model the interrupt mask that masks e_3.

Now suppose that after some time transition T1 becomes active and state S is entered once again. As exit from S subsequent to the first entry was due to e_3 that caused an interruption in the processing of S2, the history state assures that the execution will resume inside S2. Thus because of the presence of the history state in the default entrance, state S2 and not S1, will be entered. ■

EXAMPLE 7.2. The deep history state allows a statechart to remember the state at the previous exit in the presence of nesting of states. For example, consider state S in Figure 7.3(b). Suppose that T1 is enabled, the guard evaluates to true and S is entered. If this is the first entry into S then the default state S1 will be entered and the statechart is in state S.S1. Next, event e_3 occurs and S2 is entered. As S21 is the default start state in S2, the statechart is now in

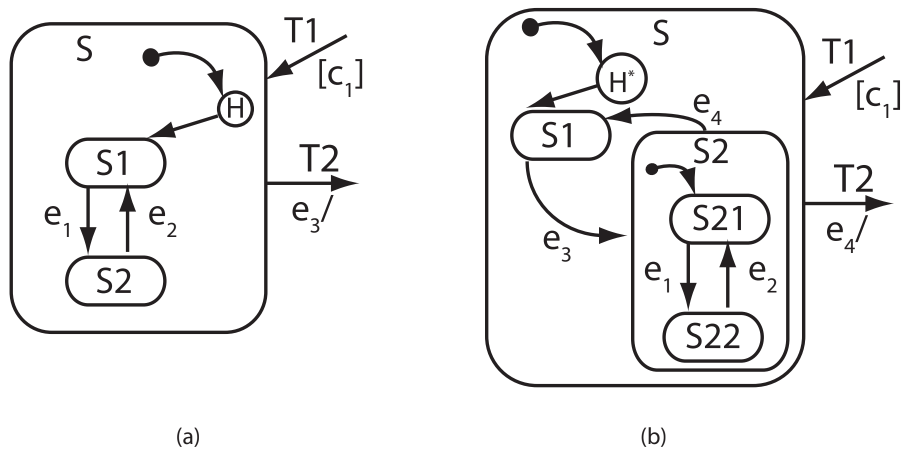

Figure 7.3: (a) Shallow history pseudo state indicated by an H inside a circle. (b) Deep history pseudo state indicated by an H* inside a circle.

state S.S2. Next, event e_1 occurs and state S22 is entered.

Now suppose that event e_4 occurs while the statechart is in S22. This causes an exit from state S through transition T2. After taking some transitions suppose that T1 is enabled and fires causing entry into S. Due to the presence of the deep history connector, control will enter state S22 as this was the active state at the last exit from S. Note that if the default entry in S was through a shallow history connector, S2 would be entered, not S22, followed by S21. ■

The entry and exit action sequence is observed regardless of which state is entered inside a composite state. Thus, for example, in Figure 7.3(a), the entry action sequence upon the second entry explained earlier will be:

$entry(S) \rightarrow entry(S2)$.

In Figure 7.3(b), the entry action sequence upon the second entry explained earlier will be:

$entry(S) \rightarrow entry(S2) \rightarrow entry(S22)$.

Conditional: Figure 7.4(a) shows three transitions coming into a composite state S and entering its three substates. These transitions share the event e and differ only in their respective guards. The conditional pseudo state is used to simplify such a structure as shown in Figure 7.4(b). In this figure the three transitions coming directly into S have been replaced by a single transition coming at S with e as the trigger. Inside S there is a C-connector that has three outgoing transitions that differ only in their guards.

The guards should be such that exactly one of them evaluates to true when e occurs. However, *nondeterminism* occurs when more than one guard is true. In this case the interpreter of the statechart decides which one of the enabled transitions to take.

Selection: Figure 7.5(a) shows three transitions entering three substates of S. These transitions share the same condition c and event e and differ only in their respective triggers. Note that e in this case is an abstract event. For example, e could represent `key_pressed` on a key-

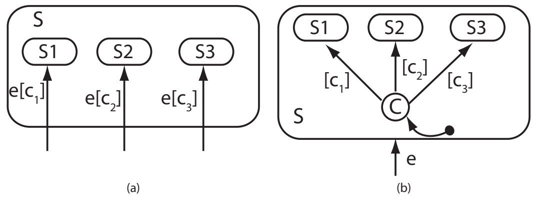

Figure 7.4: (a) One event and different guards leading to different states. (b) Equivalent of (a) using the C-connector.

pad while e_1, e_2 and e_3 denote exactly which button was pressed. As illustrated in Figure 7.5(b), the selection pseudo state is used to simplify statecharts with such transitions. The three incoming transitions shown in this figure are replaced by one transition that arrives at state S. Within S the default entry is via a selection pseudo state which has three outgoing transitions. Each of these transitions has an event trigger that corresponds to the triggers in Figure 7.5(a).

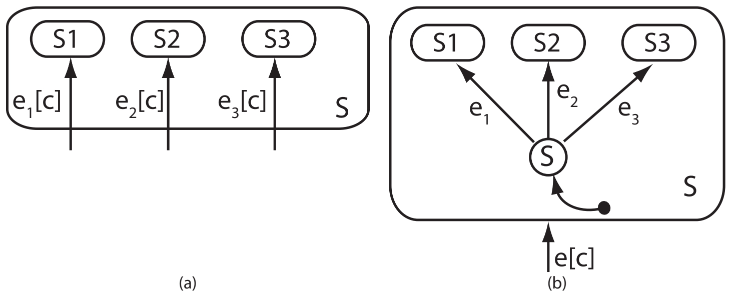

Figure 7.5: (a) One guard and different events leading to different states. (b) Equivalent of (a) using the S connector.

Junction: A junction connector allow joining and splitting of transitions. Figure 7.6(a) shows that transitions T1 and T2 are joined and a single transition, labeled e, transmitted to state S3. Here T1 and T2 might be completion transitions while e an event that serves as a common trigger. Figure 7.6(b) shows how a transition with action A is split into two transitions that share the action but correspond to different events.

Join: A *join* is used to merge two or more incoming transitions into one outgoing transition. The incoming transitions must have their corresponding source states in different regions of a compound concurrent state. Figure 7.7(a) shows two transitions with triggers e_1 and e_2 coming out of states S1 and S2, respectively, that are merged in to one transition that enters state S3. Transitions entering a join cannot have guards.

Fork: A *fork* is used to split an incoming transition into two or more outgoing transitions. The target of the outgoing transitions must be that have their target states in different regions of a composite concurrent state. Figure 7.7(b) shows a transition labeled e_1 being forked into two transitions that enter states S1 and S2. Forked transitions cannot have guards. The fork and