7. State Transition Tables

George Huling
Disciplined Software Consulting
P.O. Box 1753
Agoura Hills, CA 91376-1753
g.huling@ieee.org (805) 381-4936
http://www.laacn.org/firms/dsc/

A state transition table is a systematic method of exploring or analyzing the behavior of a discrete machine or object that exhibits history dependent behavior.

1. Discrete means the input can be divided into well defined individual data items or events that must arrive one at a time. The inputs form a sequence.
2. History dependent behavior means that the behavior resulting from an input depends on the previous sequence of inputs.

The state transition table technique is only useful if

1. the set of all inputs can be partitioned into a relatively small number of *buckets*, and
2. The set of all possible behaviors can be partitioned into a relatively small number of high-level *states*.

Otherwise, a table can be unmanageably large.

An Elevator Example

Description of elevator system

The system consists of a floor subsystem and a single elevator subsystem.

1. The floor subsystem consists of the up and down request buttons located adjacent to the elevator on each floor. There is no down button on the bottom floor, and no up button on the top floor. After being pressed to request

service, the button will remain on until cancelled by the elevator. The elevator can interrogate this set of buttons which are referred to as *floor buttons*.

2. The elevator subsystem has three subsystems.
   a. The door subsystem accepts commands from the elevator to open or close. When the door completely closes (opens) it sends a door closed (open) event to the elevator. When interfered with by a person or obstruction it opens automatically. When open it closes automatically after a predetermined interval. The elevator can interrogate the door to determine whether it is open, closed, opening, or closing.
   b. The button subsystem consists of the *numbered* buttons inside the elevator. (The open and close buttons are *not* part of this subsystem.) After being pressed to request a stop, the button will remain on until cancelled by the elevator. The elevator can interrogate this set of buttons which are referred to as *elevator buttons*.
   c. The elevator itself:

   The *home* floor of the elevator can be anything from the bottom floor to the top floor. If there are no outstanding service requests the elevator goes to the home floor (or stays at the home floor, if already there). While *going home* the elevator continually scans the buttons for a new service request and will respond immediately, changing direction if necessary.

   When the elevator initially stops at a floor, the *close button* is ignored until door has fully opened at least once. After the initial open, the elevator responds to the *open* and close buttons by sending commands to the door subsystem.

   The elevator continues to move in the same direction it is or was moving. This direction is indicated by the *arrow*, which controls the arrow indicator displayed on each floor and inside the elevator. If the arrow is *off* and there are requests both above and below the home floor (the only time the arrow will be off), the direction of initial motion is left undefined.

   The elevator controls the floor displays in the elevator and above the elevator doors on each floor by setting the value of *floor*. There is a *floor sensor* that signals the elevator when to increment or decrement the floor display and, if there is a service request for the floor being approached, to stop at that floor.

This description is a model of the elevator system external behavior and not a model of the software or hardware implementation of the elevator system. For example, the *floor sensor* is a modeling artifact and is not assumed to correspond to any physical device. The details of how the elevator accelerates, decelerates, stops, and achieves precise floor registry are all ignored. If the arrow indicates a direction and the state is Moving, then it is assumed that the elevator is or will be moving in the direction indicated.

Transition table description

All inputs recognized by the elevator are grouped under the super heading *Inputs*. The table cell at the intersection of a *State Name* row and input column specifies the action to take when the column name input is received in the row name state.

The *on entry* column specifies the action to take when a state is entered. The *always* column specifies the action to be taken continually while in that state.

Transition table

Initially the elevator is stopped at its home floor with no outstanding service requests.

Table 1. Transition Table

State Name	Inputs					on entry	always
	open button	close button	door open	floor sensor	door closed		
Stopped	A	B	ignore	can’t happen	C	call Stopping	none
Moving	ignore	ignore	can’t happen	D	can’t happen	none	none
Stopping	ignore	ignore	return	can’t happen	can’t happen	A	none
Going Home	ignore	ignore	can’t happen	E	can’t happen	none	F

Transition table actions

A. If the door is not open or opening, issue the door open command.

B. If the door is not closed or closing, issue the door close command.

C. If idle (arrow = off and floor = home) then Table 2 on page 7-4, else Table 3 on page 7-4

D. Table 5 on page 7-5

E. Table 6 on page 7-5

F. Table 7 on page 7-6

Decision table description

Definition: The *turnaround floor* is:

1. If the arrow is *up*, it is the *highest* numbered floor for which there is an outstanding service request.
2. If the arrow is *down*, it is the *lowest* numbered floor for which there is an outstanding service request.

The turnaround floor is determined by scanning the floor buttons and elevator buttons *whenever* its value is needed in a condition.

Table 2. Idle Decision Table

Idle Table arrow = off		R1	R2	R3	R4	R5	CT
C1	up, down, or elevator button request at current floor	T	F	F	F	F	
C2	up, down, or elevator button request above current floor	-	T	F	T	F	
C3	up, down, or elevator button request below current floor	-	F	T	T	F	
	column count	4	1	1	1	1	8=8
A1	issue door open command	X					
A2	cancel buttons for current floor	X					
A3	arrow = up		X				
A4	arrow = down			X			
A5	state = Moving		X	X			
A6	Call Direction Table (undefined)				X		
A6	repeat					X	
X1	exit	X	X	X	X		

Table 3. Select Direction Decision Table

Select Direction Table arrow = up (down)		R1	R2	R3	R4	CT
C1	up (down) or elevator button request at current floor	T	F	F	F	
C2	up, down, or elevator button request above (below) current floor	-	T	F	F	
C3	down (up) button request at current floor or an up, down, or elevator button request below (above) current floor	-	-	T	F	
	column count	4	2	1	1	8=8
A1	issue door open command	X				
A2	cancel buttons for current floor	X				
A3	state = moving		X	X		
A4	arrow = down (up)			X		
X1	go to Going Home Table (Table 4)				X	
X2	exit	X	X	X		

Table 4. Going Home Decision Table

Going Home Table		R1	R2	R3	CT
C1	floor = home	T	F	F	
C2	floor > home	-	T	F	
	column count	2	1	1	4=4
A1	arrow = off	X			
A3	state = Going Home		X	X	
A4	arrow = up			X	
A5	arrow = down		X		
X1	go to Idle Table (Table 2)	X			
X2	exit		X	X	

Table 5. Moving Floor Sensor Decision Table

Floor Sensor Table state =Moving		R1	R2	R3	R4	CT
C1	arrow = up	T	T	F	F	
C2	up or elevator button request for floor + 1 or floor + 1 = turnaround floor	T	F	-	-	
C3	down or elevator button request for floor – 1 or floor – 1 = turnaround floor	-	-	T	F	
	column count	2	2	2	2	8=8
A1	stop at next floor: state = Stopped	X		X		
A2	floor = floor + 1	X	X			
A3	floor = floor – 1			X	X	
A4	cancel buttons for floor	X		X		
X1	exit	X	X	X	X	

Table 6. Going Home Floor Sensor Decision Table

Floor Sensor Table state = Going Home		R1	R2	R3	R4	CT
C1	arrow = up	T	T	F	F	
C2	next floor (floor + 1) = home	T	F	-	-	
C3	next floor (floor – 1) = home	-	-	T	F	
	column count	2	2	2	2	8=8

Table 6. Going Home Floor Sensor Decision Table

Floor Sensor Table state = Going Home		R1	R2	R3	R4	CT
A1	stop at next floor: state = Stopped	X		X		
A2	floor = floor + 1	X	X			
A3	floor = floor – 1			X	X	
A4	cancel buttons for floor	X		X		
A5	arrow = off	X		X		
X1	exit	X	X	X	X	

Table 7. Going Home Idle Decision Table

Idle Table state = Going Home		R1	R2	R3	R4	CT
C2	up, down, or elevator button request above current floor	T	F	T	F	
C3	up, down, or elevator button request below current floor	F	T	T	F	
	column count	1	1	1	1	4=4
A1	arrow = up	X				
A2	arrow = down		X			
A3	state = Moving	X	X			
A4	Call Direction Table (undefined)			X		
A5	repeat				X	
X1	exit	X	X	X		

Discussion

The idea behind the state transition table (or any other tabular method) is divide and conquer. We want to solve a large problem by dividing it into pieces each of which is small enough to be significantly easier to solve and which together solve the complete problem.

To be useful a state transition table should partition[1] each of the input set and the state set into a reasonably small number of subsets. If the number of cells it too large, the table is unwieldy. If the number of cells is too small,

1. A *partition* of a set A is a set of subsets of A that are pairwise disjoint, and their union is the entire set A.

the table may not make the divisions small enough to make each cell manageable.

To be correct a state transition table must be complete and consistent[1]. Mechanical consistency is fairly easy to check. There are three basic rules to follow.

1. The input subsets or buckets must partition the *entire* input space. The input space is any external stimulus recognized by the object being modeled. Every external stimulus recognized by the object being modeled must be in one and only one of the input subsets.
2. In every pair of states, the states must be distinct in at least one clearly defined attribute. For example, given two states that share some state variables, at least one of the variables must have values in one state that are disjoint from its values in the other state. If this is true, the states are distinct because a variable cannot take on two different values at once.
3. The specification of the action taken in each cell of the table must be mechanically consistent. If the action is simple enough, consistency can be determined by inspection. If not, decision tables can be used because there are mechanical ways of checking them for completeness and consistency.

Completeness is often much harder to determine because it involves comparing the specified external behavior with the behavior generated by the model. To begin with, we need a specification of the external behavior that will enable us to generate scenarios with which the model can be tested. It is helpful if the specification allows us to generate generalized scenarios rather than testing one specific stimulus history at a time. In this connection, a specification in terms of use cases is quite helpful. The idea of the test is to look for input sequences that force us to split a state.

Suppose we have a state S and two input sequences X and Y that both lead to state S. If we can find an input I whose behavior when appended to X must be different from its behavior when appended to *Y*, then we must split state S into two states, say S_x and S_y, with X leading to S_x and Y leading to S_y.

Questions

1. The elevator can determine it has nothing to do either at the home floor or at some other floor. Can an observer at the home floor distinguish these two cases, and, if so, how?

1. *Consistent* means conflict free. It must not be possible to get two or more different actions from the same input stimulus.

2. Suppose I arrive at the elevator just as the door is closing. Rather than interfere with the door, I press a request button. What happens? Does it make any difference which button I press or what the status of the arrow indicator is?

Summary

To check for completeness and consistency.

1. Every stimulus recognized by the system must belong to one and only one input subset or bucket.
2. Every pair of states must differ in at least one well defined attribute.
3. In any state, the same input must produce the same behavior. For every two input histories *X* and *Y* both of which lead to the same state and for every input *I*, it must be true that *Behavior(XI) = Behavior(YI).*
4. The specification given in each cell must be checked for consistency and completeness.