Department of Electrical and Computer Engineering
Dalhousie University

State Diagrams

Larry Hughes

Introduction

Many communication systems and protocols are described in terms of *state diagrams* or *state machines*. State diagrams permit an unambiguous, visual description of a system's individual events and their related actions. This document presents an overview of state diagrams and includes an example of the design and implementation of a state diagram and its associated application.

Concepts

Any task that can be broken into a series of distinct steps (or *states*) can be defined in terms of a state diagram. Control remains in a state until an *event* occurs, which causes a *transition* to a new state. For example, the task of traversing a road at a pedestrian crossing could be broken into two states:

STOP Wait for a green light.

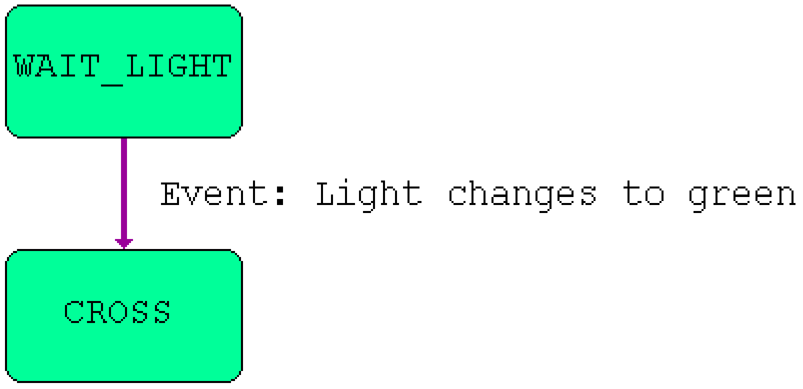
Cross the road.

The event that causes the state transition (from waiting for a green light to crossing the road) is the availability of a green light. In other words, when the light turns green, the road can be crossed.

A state diagram is a visual representation of a task's various states and events. Formally, a state diagram is a weighted, directed graph; with each vertex corresponding to a state and each directed edge representing a transition. At a minimum, a state diagram consists of three parts:

- the state (typically a labelled box),
- the transition between states (typically an arrow joining the two boxes), and
- a description of the event that caused the transition.

For example, the following figure is the state diagram for the street traversing task:



Each event (or **condition**) that causes a transition to a new state is associated with an **action** or **output**. In a state diagram, this is drawn as follows:

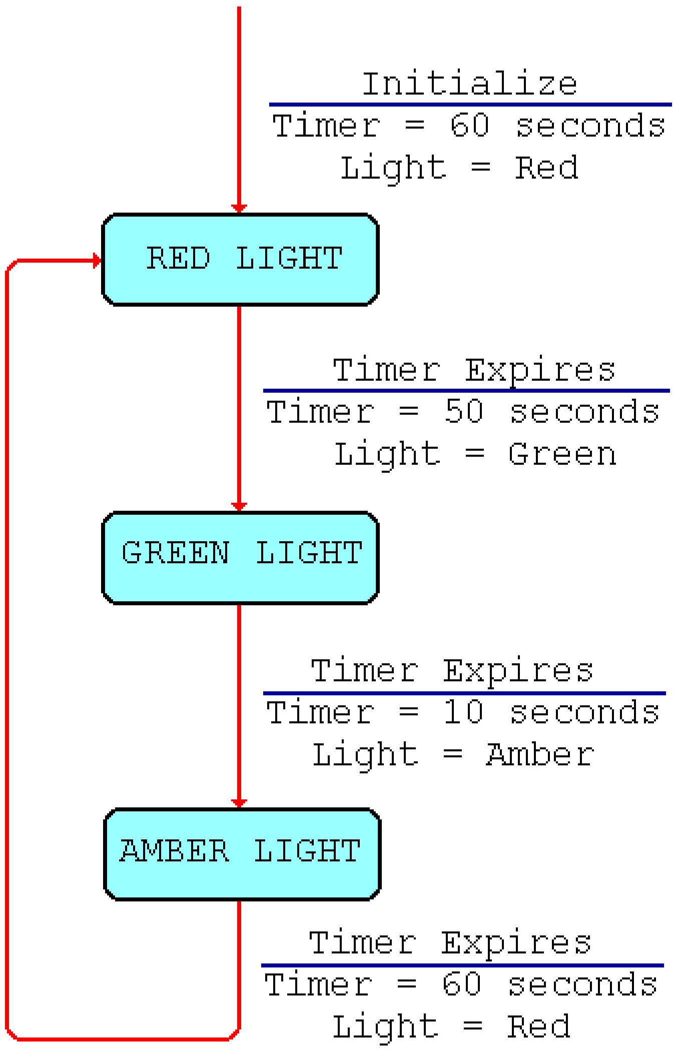
$$\frac{\text{CONDITION}}{\text{ACTION}}$$

In cases (such as the light changing to green), where there is no action associated with the condition, the action should be written as "Ø", indicating that nothing is taking place.

A slightly more complex example of a state diagram involves the states and transitions associated with the traffic light:

- Show red light until the timer expires; at which point, turn the light green and start the timer for 50 seconds.
- Show green light until the timer expires; at which point, turn the light amber and start the timer for 10 seconds.
- Show amber light until the timer expires; at which point, turn the light red and start the timer for 60 seconds.

In this task, control remains in each state until the timer expires, at which point the light changes colour and the timer is assigned a new value. A possible state diagram for the traffic light is shown below. Note the use of the initialization transition, which causes the timer to run for 60 seconds and the light to turn red.



A state can be associated with transitions to more than one state. In these situations, different conditions are applied to determine the new state. Furthermore, it is possible for a condition to lead back to the same state; for example, control may stay within a state until a certain number of events have occurred. The following example shows how incoming keystrokes can be monitored; if a carriage return is detected, control passes to the next state, otherwise the character is stored in the keyboard buffer (the array `string`). Note that the event in this example is the arrival of a character which is subsequently examined.

Implementation

Once a state diagram has been developed for a task, it can be implemented. Implementation is most easily achieved using a `switch` statement; the `switch` expression is the new state, while the different states are listed as `case` labels.

In the state diagram, control remains in the state until a condition arises that causes a transition to a new state. In a system consisting of multiple processes, it may not be possible to remain physically in the state while waiting for the event to occur; instead, the process must relinquish the CPU until the event occurs.

As an example of implementing a state diagram, consider the traffic light task described in the previous section. In this case, the state diagram consists of four transitions and three states. The first transition, initialization, consists of starting the timer and turning the light on, then entering the `RED_LIGHT` state. The task, running as long as `power_on` remains `TRUE`, waits until the timer expires, at which point, the traffic light state changes to its new state. The traffic light state diagram could be implemented as follows:

```
/* Initialization */
start_timer(60);
light_on(RED);
state = RED_LIGHT;
while(power_on)
{
   wait_for_timer();
   /* At this point, the timer has expired */
   switch(state)
   {
   case RED_LIGHT:     /* Light is red, turn to green */
      start_timer(50);
      light_on(GREEN);
      state = GREEN_LIGHT;
      break;
   case GREEN_LIGHT:   /* Light is green, turn to amber */
      start_timer(10);
      light_on(AMBER);
      state = AMBER_LIGHT;
      break;
   case AMBER_LIGHT:  /* Light is amber, turn to red */
      start_timer(60);
      light_on(RED);
      state = RED_LIGHT;
      break;
   }
}
```

When there are several possible conditions that can lead from a state, it is necessary to select the new state by testing the various conditions. For example, in the keystroke state diagram, the state would change only when a carriage return was detected:

```
case BUILD_STRING:
   if (ch != CR)
      *kbb++ = ch;
   else
   {
      *kbb = NUL;
      state = END_OF_LINE;
   }
```
