Chapter 4
Finite State Machines

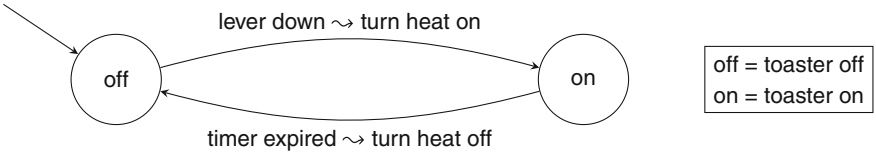
The Braitenberg vehicles and the line following algorithms (Chap. 3) demonstrate reactive behavior, where the action of the robot depends on the *current* values returned by robot's sensors, not on events that happened previously. In Sect. 4.1 we present the concept of state and finite state machines (FSM). Sections 4.2 and 4.3 show how certain Braitenberg vehicles can be implemented with FSMs. Section 4.4 discusses the implementation of FSMs using state variables.

4.1 State Machines

The concept of state is very familiar. Consider a toaster: initially, the toaster is in the off state; when you push the lever down, a transition is made to the on state and the heating elements are turned on; finally, when the timer expires, a transition is made back to the off state and the heating elements are turned off.

A *finite state machine (FSM)*[1] consists of a set of *states* s_i and a set of *transitions* between pairs of states s_i, s_j. A transition is labeled *condition/action*: a condition that causes the transition to be taken and an action that is performed when the transition is taken.

FSMs can be displayed in *state diagrams*:



A state is denoted by a circle labeled with the name of the state. States are given short names to save space and their full names are given in a box next to the state

[1] Finite state machines are also called finite automata.

M. Ben-Ari and F. Mondada, *Elements of Robotics*,
https://doi.org/10.1007/978-3-319-62533-1_4

diagram. The incoming arrow denotes the initial state. A transition is shown as an arrow from the *source* state to the *target* state. The arrow is labeled with the condition and the action of the transition. The action is not continuing; for example, the action turn left means set the motors so that the robot turns to the left, but the transition to the next state is taken without waiting for the robot to reach a specific position.

4.2 Reactive Behavior with State

Here is the specification of a Braitenberg vehicle whose behavior is non-reactive:

> **Specification (Persistent)**: The robot moves forwards until it detects an object. It then moves backwards for one second and reverses to move forwards again.

Figure 4.1 shows the state diagram for this behavior.

Initially, when the system is turned on, the motors are set to move forwards. (The condition true is always true so this is done unconditionally.) In state fwd, if an object is detected, the transition to state back is taken, the robot moves backwards and the timer is set. When one second has passed the timer expires; the transition to state fwd is taken and the robot moves forwards. If an object is detected when the robot is in the back state, *no* action is performed, because there is no transition labeled with this condition. This shows that the behavior is not reactive, that is, it depends on the current state of the robot as well as on the event that occurs.

Activity 4.1: Consistent

- Draw the state diagram for the Consistent Braitenberg vehicle.

> **Specification (Consistent):** The robot cycles through four states, changing state once every second: moving forwards, turning left, turning right, moving backwards.

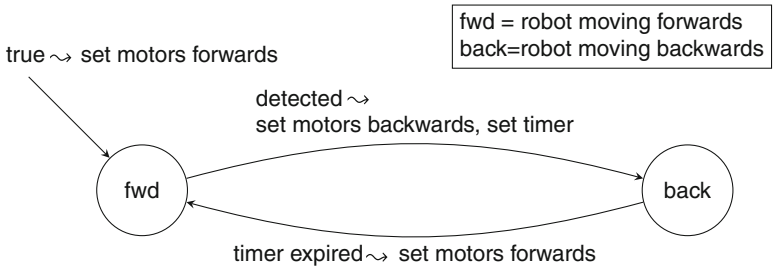


Fig. 4.1 FSM for the persistent Braitenberg vehicle

4.3 Search and Approach

This section presents a more complex example of a robotic behavior that uses states.

Specification (Search and approach): The robot searches left and right (±45°). When it detects an object, the robot approaches the object and stops when it is near the object.

There are two configurations of the sensors of a robot that enable it to search left and right as shown in Fig. 2.8a, b. If the sensors are fixed to the body of the robot, the robot itself has to turn left and right; otherwise, if the sensor is mounted so that it can rotate, the robot can remain still and the sensor is rotated. We assume that the robot has fixed sensors.

Figure 4.2 shows the state diagram for search and approach. The robot is initially searching left. There are two new concepts that are illustrated in the diagram. There is a *final state* labeled found and denoted by a double circle in the diagram. A finite state machine is finite in the sense that it contains a finite number of states and transitions. However, its behavior can be finite or infinite. The FSM in Fig. 4.2 demonstrates finite behavior because the robot will stop when it has found an object and approached it. This FSM also has infinite behavior: if an object is never found, the robot continues indefinitely to search left and right.

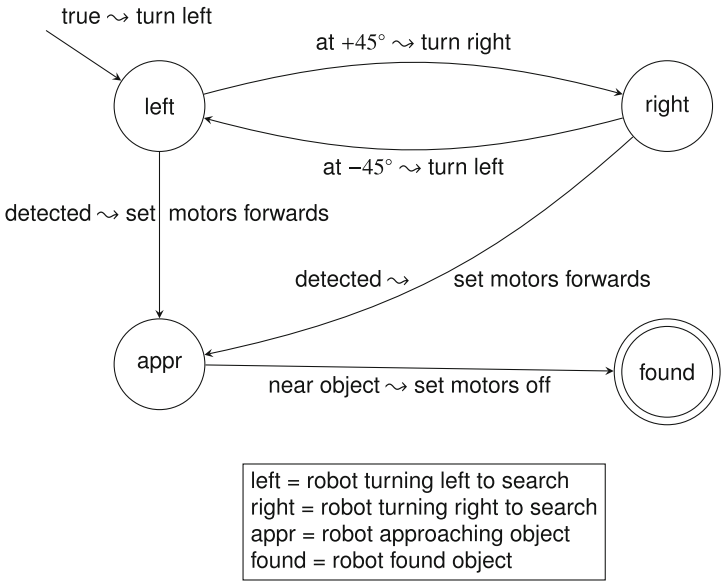


Fig. 4.2 State diagram for search and approach

The Persistent Braitenberg vehicle (Fig. 4.1) has infinite behavior because it continues to move without stopping. A toaster also demonstrates infinite behavior because you can keep toasting slices of bread forever (until you unplug the toaster or run out of bread).

The second new concept is *nondeterminism*. States left and right each have *two* outgoing transitions, one for reaching the edge of the sector being searched and one for detecting an object. The meaning of nondeterminism is that any of the outgoing transitions may be taken. There are three possibilities:

- The object is detected but the search is not at an edge of the sector; in this case, the transition to appr is taken.
- The search is at an edge of the sector but an object is not detected; in this case, the transition from left to right or from right to left is taken.
- The search is at an edge of the sector exactly when an object is detected; in this case, an arbitrary transition is taken. That is, the robot might approach the object or it might change the direction of the search.

The nondeterministic behavior of the third case might cause the robot to fail to approach the object when it is first detected, if this event occurs at the same time as the event of reaching $\pm 45°$. However, after a short period the conditions will be checked again and it is likely that only one of the events will occur.

4.4 Implementation of Finite State Machines

To implement behaviors with states, variables must be used. The Persistent vehicle (Sect. 4.2) needs a timer to cause an event after a period of time has expired. As explained in Sect. 1.6.4, a timer is a variable that is set to the desired period of time. The variable is decremented by the operating system and when it reaches zero an event occurs.

Algorithm 4.1 describes how the FSM of Fig. 4.1 is implemented. The variable current contains the current state of the robot; at the conclusion of the processing of an event handler, the value of the variable is set to the target state of the transition. The values of current are named fwd and back for clarity, although in a computer they would be represented by numerical values.

Activity 4.2: Persistent

- Implement the Persistent behavior.

Algorithm 4.1: Persistent

```
integer timer                                          // In milliseconds
states current ← fwd
left-motor-power ← 100
right-motor-power ← 100
loop
    when current = fwd and object detected in front
        left-motor-power ← −100
        right-motor-power ← −100
        timer ← 1000
        current ← back

    when current = back and timer = 0
        left-motor-power ← 100
        right-motor-power ← 100
        current ← fwd
```

Activity 4.3: Paranoid (alternates direction)

- Draw the state diagram for the Paranoid (alternates direction) Braitenberg vehicle:

  Specification (Paranoid (alternates direction)):

  - When an object is detected in front of the robot, the robot moves forwards.
  - When an object is detected to the right of the robot, the robot turns right.
  - When an object is detected to the left of the robot, the robot turns left.
  - If the robot is turning (even if it no longer detects an object), it alternates the direction of its turn every second.
  - When no object is detected and the robot is not turning, the robot stops.

- Implement this specification. In addition to a variable that stores the current state of the robot, use a variable with values left and right to store the direction that the robot turns. Set a timer with a one-second period. In the event handler for the timer, change the value of the direction variable to the opposite value and reset the timer.

Algorithm 4.2 is an outline of the implementation of the state diagram in Fig. 4.2.

Algorithm 4.2: Search and approach

```
        states current ← left
 1: left-motor-power ← 50                  // Turn left
 2: right-motor-power ← 150
 3: loop
 4:     when object detected
 5:         if current = left
 6:             left-motor-power ← 100     // Go forwards
 7:             right-motor-power ← 100
 8:             current ← appr
 9:         else if current = right
10:             ...
11:     when at +45°
12:         if current = left
13:             left-motor-power ← 150     // Turn right
14:             right-motor-power ← 50
15:             current ← right
16:     when at −45°
17:         ...
18:     when object is very near
19:         if current = appr
20:             ...
```

Activity 4.4: Search and approach

- Fill in the missing lines in Algorithm 4.2.
- Implement Algorithm 4.2.

4.5 Summary

Most robotics algorithms require that the robot maintain an internal representation of its current state. The conditions that the robot uses to decide when to change state and actions taken when changing from one state to another are described by finite state machines. State variables are used to implement state machines in programs.

4.6 Further Reading

The classic textbook on automata was published in 1979 by John Hopcroft and Jeffrey D. Ullman; its latest edition is [2]. For a detailed explanation of *arbitrary* choice among alternatives in nondeterminism (Sect. 4.3), see [1, Sect. 2.4].

References

1. Ben-Ari, M.: Principles of Concurrent and Distributed Programming, 2nd edn. Addison-Wesley, Boston (2006)
2. Hopcroft, J., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation, 3rd edn. Pearson, Boston (2007)