

CS1Ah Lecture Note 12

Finite State Machines 1

12.1 Finite State Machines: Definition

Round about us in our world we encounter many examples of *reactive systems*¹ Traditionally, typical examples of these systems are a whole range of vending machines, security door entry systems, parking ticket machines or tills and cash registers. Each of these has the same characteristics of detecting action in the world via some *sensors* (e.g. pushbuttons, swipe card readers, coin counters) and then *reacting* to take action in the world via *actuators* (e.g. printers to print tickets or solenoids and motors that control the delivery of merchandise like confectionery or drinks)

Over the past twenty years or so, one of the main ways of adding value to existing merchandise has been to augment basic goods with reactive systems that either make the combined system easier to use or extend its capabilities. A good example of this is the telephone. The basic system that could make and receive calls has been overlaid with the capacity to store numbers, receive text messages, manage access to the phone, play games, The process has proceeded apace and now it is rare to interact with any electro-mechanical device without the mediation of some kind of reactive system – soon nearly everything will have some kind of computer built into it! This class of systems is both economically significant and (as we will see) worthy of study.

In the early 20th century when reactive systems began to become commonplace a number of Computer Scientists (of course nobody was called a Computer Scientist then) began to formulate various mathematical models that were intended to capture the essence of reactive systems. In the next couple of lecture notes we consider one of the simplest models of such systems. These are called Finite State Machines (sometime people use *Automata* instead of *Machines*, in this context they mean the same thing).

Deliberately, this is a very restrictive model of reactive systems that throws out much of what we might think of as distinctive but this very simple model is still very rich and today remains the basis for much design activity across a range of application areas in hardware and software engineering, in information indexing and in linguistics for speech and language computer applications. In this area the Computer Science

¹This is jargon for the kinds of system where we provide the system with stimuli via various kinds of sensor that detect activity in the world and the reactive system “reacts” by influencing the world by various kinds of “actuator” that cause effects in the real world under the control of the system.

theory is fairly mature and there is a rich collection of theoretical work that informs practice and underpins the development of tools to support engineering activity.

12.1.1 The Main Idea

The main observation to make about reactive systems is that the response to a particular stimulus is not the same on every occasion. For example, in the case of a parking ticket machine, it will not print a ticket when you press the button unless you have already inserted some money. Thus the response to the print button depends on the previous history of the use of the system.

The finite-state machine model restricts the number of different responses to a particular stimulus to be finite and fixed in the description of the machine. Thus the model consists of a finite number of different states and the effect of a stimulus is to (possibly) generate a response and change state to some new state that reflects what we want to remember about the past inputs. Thus we are restricted to remembering a fixed finite amount about what has gone before. This may seem like a very weak restriction, but as we will see later there are many simple counting tasks that do not fit within this model.

12.1.2 Details

In this section we describe the details of the finite-state machine definition. This comprises a section on how to describe finite-state machines and how this description relates to the behaviour of the machine as a reactive system. We then go on to discuss some further technicalities of the definition.

Description

We describe a FSM by providing:

1. A description of the stimuli that the machine will take into account. This is defined by giving a finite *input alphabet*. This is an abstraction of the kind of stimuli accepted by reactive systems. The input alphabet is usually called Σ (pronounced sigma). For example, if $\Sigma = \{a, b\}$ then we imagine that the reactive system has two pushbuttons one labelled a , the other b and the stimuli are pressing one or other of the buttons in turn. The other way of seeing this is that the reactive system has a single stream of input that can accept a sequence of letters that can either be a or b .
2. A description of the responses that the machine can generate. This is defined by giving a finite *output alphabet*. This is an abstraction of the kind of responses generated by a reactive system. The output alphabet is usually called Λ (pronounced lambda). For example, if $\Lambda = \{p, q\}$ then we imagine that the reactive system has two lights on it, one labelled p , the other labelled q and the system can turn them on and off independently in response to stimuli. Alternatively, we can see the reactive system as having a single output stream that consists of a sequence of ps and qs .

3. We describe the machine as a diagram consisting of blobs (usually circles) that stand for distinct states in the machine.
4. The action of the machine is provided by giving transitions that are arrows that point from one state to another. Each transition is labelled a/b where a is drawn from Σ and b is drawn from Λ . The idea is that if the machine is in a particular state when it senses stimulus a and there is a transition labelled a/b from that state to another then it generates output b and changes state to the new state indicated by the transition.

On some occasions we might want to be able to take a transition when we have no stimulus or make a transition without generating a response. In these cases we can replace the letters on the transition by the symbol ϵ (pronounced epsilon) indicating that no stimulus or response is required. Usually we omit ϵ s in diagrams since we can always clearly see where we would have used an ϵ .

5. Finally we have to indicate the state the system starts in. This is indicated by a transition arrow pointing at a state with no state at the other end.

Here is a very simple example of a parking ticket machine. The input alphabet is $\Sigma = \{m, t, r\}$ standing for: inserting money, requesting a ticket, and requesting a refund. The output alphabet is $\Lambda = \{p, d\}$ standing for: print ticket, and deliver refund. The transitions and start state of the machine are shown in figure 12.1. The idea here is

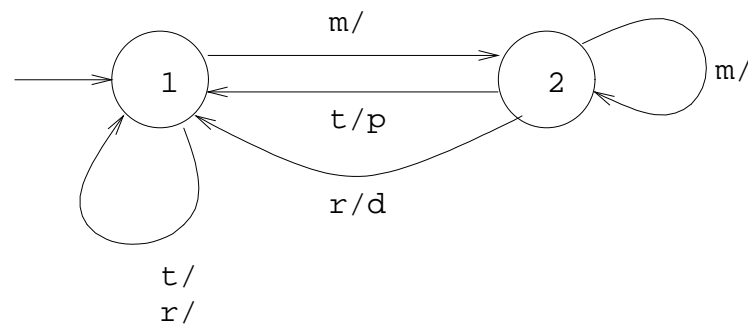


Figure 12.1: FSM for a Parking Ticket Machine

that the machine responds differently to a ticket request depending on whether any money has been deposited by the user of the system. If the user presses the button without inserting money then no ticket is printed. If money has been inserted then a ticket will be printed and the state changes back to the start state. So, the states capture the situation where no money has been inserted since the last ticket was printed and the situation where money has been inserted since the last ticket was printed.

Behaviour

The last section showed how to give a description of a finite-state machine. In this section we tighten up what behaviour a particular machine can exhibit. We do this by defining a *trace* of a finite-state machine. A *trace* for a finite-state machine M is defined to be:

1. A finite sequence of alternating states and transition labels starting and ending with a state: $s_0, i_1/o_1, s_1, \dots, s_{n-1}, i_n/o_n, s_n$.
2. The first state, s_0 , is the start state of M .
3. For each state, label, state sequence $s_{j-1}, i_j/o_j, s_j$ appearing in the sequence, there must be a transition in M from state s_{j-1} to s_j labelled i_j/o_j .

For example, one trace of the parking meter machine is: $1, m/, 2, m/, 2, m/, 2, r/d, 1$. This is the behaviour where the user deposits three coins and then decides to ask for a refund. The following is *not* a trace: $1, m/, 1, m/, 2, m/, 2, r/d, 1$ because the very first transition is not in the parking meter machine description (if it was and we kept the interpretation of the states then this would mean the machine could forget about money that had been deposited after the last ticket was printed). If we take a trace of a machine and drop the states from the trace then we get a possible sequence of stimulus/response pairs that could be generated by the machine. The overall behaviour of the machine is the collection of all traces that the machine could exhibit (notice that even for the parking machine this is an infinite set).

Transducers and Acceptors

Up to now we have mostly considered FSMs as reactive systems, which is how they are used in engineering applications, but the scientific study of FSMs often wants to use a more static representation of the behaviour of the machine to allow us to reason about its behaviour. In the standard literature on Finite State Machines there are two ways of doing this, considering the machine as a *transducer* or as an *acceptor*.

Viewed as an *acceptor* a FSM is something that defines a subset of the set of all sequences of the input alphabet. To do this we need to extend the definition of FSM a little. The extension is to mark some states as being *accepting*. Used as an acceptor the output alphabet Λ is empty and all the response labels on the transitions are ϵ . A sequence of symbols $i_1 \dots i_n$ from the input alphabet, Σ , is *accepted* if there is a trace $s_0, i_1/\epsilon, \dots, i_n/\epsilon, s_n$ and s_n is an accepting state. Rather than write the full transition label we just write the input symbol since the output symbol is always ϵ .

Figure 12.2 illustrates an acceptor for sequences of 0s and 1s in which the number of 0s never exceeds the number of 1s by more than one and *vice versa*. The accepting states are marked by a double circle. In this case the sequence 0101101 is accepted because the trace corresponding to it has an accepting state (C) as its final state. The trace corresponding to this sequence is:

$$B, 0, A, 1, B, 0, A, 1, B, 1, C, 0, B, 1, C$$

By contrast, the sequence 0101001 is not accepted by the FSM because the trace corresponding to it is:

$$B, 0, A, 1, B, 0, A, 1, B, 0, A, 0, D, 1, D$$

and D is not an accepting state. Note that the failure to accept a sequence can arise in two ways:

- We are able to construct a trace corresponding to the sequence but the final state in the trace is not an accepting trace.

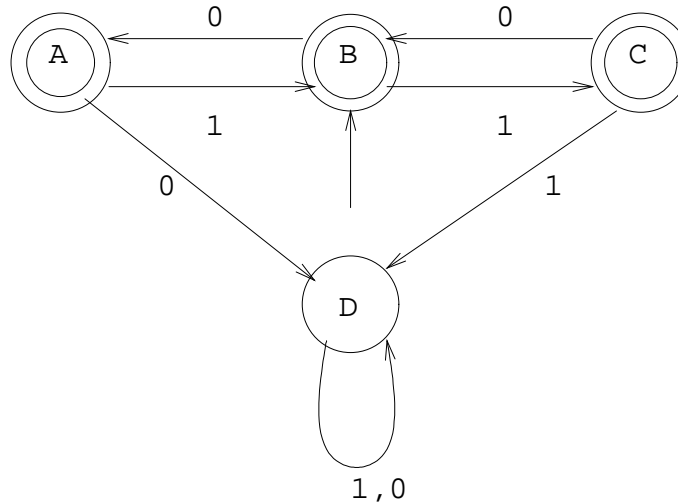


Figure 12.2: An FSM Acceptor

- We cannot construct a trace for the given sequence. For example, if we omit state D in the above acceptor then it would be impossible to construct a trace for the sequence 0101001.

For a given FSM M we say that the *language accepted by M* is the set of a sequences accepted by M .

Viewed as a transducer a finite state machine with accepting states defines a mapping from sequences of the input alphabet to sequences of the output alphabet. We illustrate this by considering Figure 12.1 (the ticket machine). If we suppose that state 1 is an accepting state then a typical trace with an accepting final state is:

$$1, m/, 2, r/d, 1, t/, 1, m/, 2, m/, 2, t/p, 1$$

We now look just at the sequence of input symbols and then at the sequence of output symbols and we say that the FSM maps from the input sequence $mrtmmt$ to the output sequence dp . For each trace with an accepting final state we can do this and this defines how the machine maps from a sequence of inputs to a sequence of outputs. Notice that there is loss of information in going from the trace to this mapping because we lose information about when in the input sequence a particular output is generated.

Deterministic Machines

In this discussion we just consider acceptors, most of what we say here carries over to transducers but in some circumstances things can be a bit more delicate. Nothing we have said so far requires us to ensure that two transitions originating at the same state need to be labelled with different inputs and labelling a transition with ϵ on the input allows the machine to move along the transition without any external stimulus. This means that there may be more than one trace corresponding to a particular sequence of inputs. An input sequence is accepted if at least one trace for the sequences is accepting. In some circumstance we would like to know that there is at most one trace for any input sequence. If this is so then the FSM is called *deterministic*, machines for which this is not true are called *non-deterministic*. Later, in your career you will

see there is a method for converting any non-deterministic acceptor to a deterministic acceptor. However, this may result in an exponential increase in the number of states.²

12.1.3 Summary

In this lecture note we have:

- Considered reactive systems.
- Studied the definition of Finite State Machines that can provide a model for reactive systems.
- Studied how to interpret the definition of FSMs as a description of the behaviour of reactive systems.
- Seen how FSMs can be considered to be acceptors or transducers.
- Considered the distinction between deterministic and non-deterministic FSMs.

The basic reference for all the material on Finite State Machines is Chapter 10 of *Foundations of Computer Science* by A.V. Aho and J.D. Ullman, Computer Science Press.

Murray Cole, 1st November 2002.

²This means if the non-deterministic machine has n states, the equivalent deterministic FSM may have as many as 2^n states. This is not always the case but it can happen.