Finite-State Machines

CS236 - Discrete Structures
Instructor: Brett Decker
Fall 2020

Finite-State Machines

We will now study *finite-state machines* which are the recognizers for the languages generated by regular expressions. Finite-state machines are used in many areas of Computer Science.

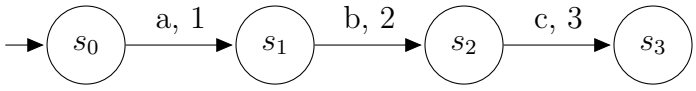
Finite-State Machine: Definition 1, Section 13.2.2*

A *finite state machine* $M = (S, I, O, f, g, s_0)$ consists of the following:

a finite set S of *states*
a finite *input alphabet* I
a finite *output alphabet* O
a *transition function* f that assigns to each state and input pair a new state
an *output function* g that assigns to each state and input pair an output
an *initial state* s_0

Finite-State Machine Example:

Let's put the above definition into something concrete. There are two ways to represent a finite-state machine (FSM): 1) diagram form; 2) table form. The state diagram form is often more intuitive, so let's start there. The following is a state diagram of an FSM:



Each state is a circle with a label. The arrows show transitions from one state to another. Above the arrows, there is a pair of symbols separated by a comma. The first symbol is the input symbol read, and the second symbol is the output symbol written. The arrow without a label shows where the machine starts. Thus we say the machine starts at s_0. Conceptually, machines are given a sequence of characters called the input. At the start state, the machine tries to read the first character of the input. In our example above, the machine can only read an a (all other inputs are considered invalid and lead to error state – we ignore error states for simpler diagrams). Once the machine reads an a the character is consumed from the input, the machine will output '1', and transition to state s_1. The same process occurs at each state until the input is completely consumed. If there are more characters on the

input when the machine is done, this is also an error. The machine above then only reads the input *abc*.

Formally, our example machine, call it M, is defined as follows: $M = (S, I, O, f, g, s_0)$ where:

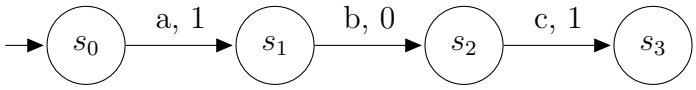
$S = \{s_0, s_1, s_2, s_3\}$
$I = \{a, b, c\}$
$O = \{1, 2, 3\}$
$f = \{(s_0, a, s_1), (s_1, b, s_2), (s_2, c, s_3)\}$
$g = \{(s_0, a, 1), (s_1, b, 2), (s_2, c, 3)\}$
$s_0 = s_0$

Here is the state table representation:

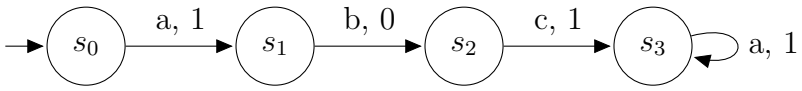
	f **Input**			g **Input**		
State	**a**	**b**	**c**	**a**	**b**	**c**
s_0	s_1			1		
s_1		s_2			2	
s_2			s_3			3

Language Recognition

We can use the output of a machine to determine which input strings are recognized. One way to do this is to say that an input string is recognized when the last output bit is a 1 (see Example 7, Section 13.2.2*). This gives a precise way to use finite-state machines as language recognizers. Let's modify the FSM from the above example as follows:



What strings will the FSM recognize or accept? Recall the definition states that an accept string is indicated by an output of '1'. Thus, the above FSM will accept the string a, and the string abc (such FSMs – that output a '1' for string acceptance – are referred to as Mealy machines). That corresponds to the language $L = \{a, abc\}$. Let's connect this with regular expressions. The regular expression that generates L is $(a \cup abc)$. Now we have made the connection between FSMs and regular expressions. Every language generated by a regular expression can be recognized by a finite-state machine. But wait. You may recall that regular expressions can generate languages with an infinite number of words? How does a finite-state machine recognize infinite words? The *finite* in finite-state machine only specifies that the number of states much be finite. Consider the following change to our FSM:

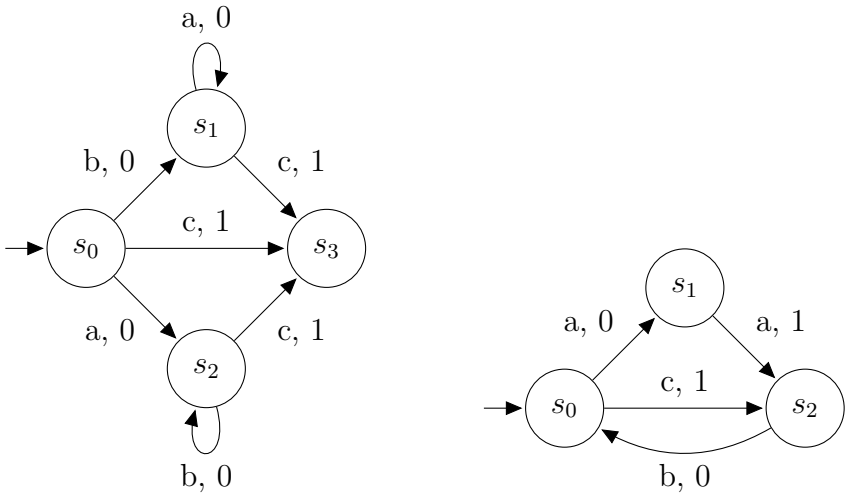


Note the self-loop at s_3. It reads an a and outputs 1 which can happen infinitely many times: no matter how many a's you give the machine when in s_3 it continues to read

them and outputs 1. Thus, the FSM can now recognize the following language $L = \{a, abc, abca, abcaa, abcaaa, abcaaaa, \ldots\}$ – the language that contains words of a or abc with zero or more a's afterward – which corresponds to the regular expression $(a \cup abc(a^*))$.

Language Recognition Example:

Try to create the regular expressions for the following FSMs:



The first recognizes the language generated by this regular expression: $(b(a^*)c \cup c \cup a(b^*)c)$. The second recognizes the language generated by this regular expression: $(aa \cup c)[b(aa \cup c)]^*$.

Language recognition is so useful with FSMs that we often use a special kind of finite-state machine called a *finite-state automaton* (FSA) – the plural is finite-state automata – that are used especially for recognition. We will explore finite-state automata next.

Conclusion

Finite-state machines are extremely important. The book* lists the following areas that use finite-state machines: spell checking, Grammar checking, Speech recognition, text indexing and searching, markup languages: xml/html, and network protocols.

*All definitions are from *Discrete Mathematics and Its Applications*, by Kenneth H. Rosen.