F
Finite State Automata

A *finite automaton* (FA, also called a *finite-state automaton* or a *finite-state machine*) is a mathematical tool used to describe processes involving inputs and outputs. An FA can be in one of several states and can switch between states depending on symbols that it inputs. Once it settles in a state, it reads the next input symbol, performs some computational task associated with the new input, outputs a symbol, and switches to a new state depending on the input. Notice that the new state may be identical to the current state. Figure F.2a shows an example of an FA with four states labeled 1 through 4. The input consists of a string of symbols from the alphabet `abc` and the output is a string of symbols from `mnpq`. The FA starts in state 1. If it inputs an `a`, it stays in state 1 and outputs an `m`. If it inputs a `b`, it switches to state 2 and outputs a `p`. Notice that this particular FA can get "stuck" if it happens to be in a state that does not tell it what to do with a certain input. This will happen, for example, with the input string "`abbc...`", since the `c` will be input while the FA is in state 4, where it expects an `a`. We say that "`abbc...`" is not accepted by the FA. The edges of the graph are labeled by pairs (input, output).

Figure F.2b shows an example of a two-state FA that can be used to add binary numbers. The FA starts in state 1 (no carry) and inputs a pair of bits. If the pair is 11, the FA outputs a 0 and switches to state 2 (carry), where the next pair of bits is input and is added to a carry bit of 1. Table F.1 shows the individual steps in adding the two 6-bit numbers 14 and 23 (these are actually 5-bit numbers, but the simple design of this FA requires that an extra 0 bit be appended to the left end of every number). Since adding numbers is done from right to left, the six columns of the table should also be read in this direction.

Figure F.2c shows an example of a three-state FA where each edge is labeled by an input symbol and a probability, but no output. The diagram illustrates an FA that inputs a stream of bits where the first bit has a 50% probability of being a 0 or a 1, and each consecutive bit is likely to be identical to its predecessor. A zero follows another zero with a probability of 2/3, and a one follows a one with a

Step:	6	5	4	3	2	1
14:	0	0	1	1	1	0
23:	0	1	0	1	1	1
Input:	00	01	10	11	11	01
Initial state:	2	2	2	2	1	1
Output (37):	1	0	0	1	0	1
Final state:	1	2	2	2	2	1

Table F.1: Adding 14 and 23.

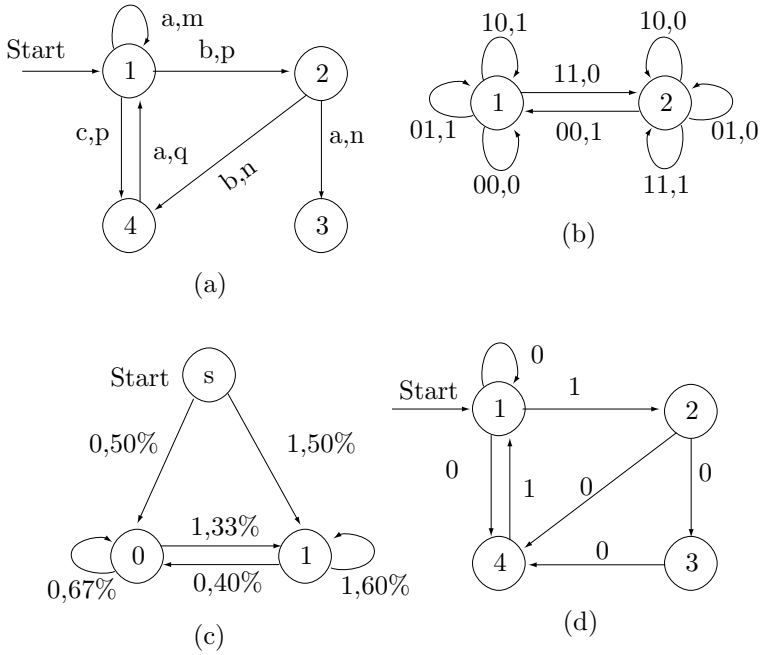

Figure F.2: Examples of FAs.

probability of 3/5. This FA does not produce any outputs, but any practical FA should, of course, do something (i.e., produce some output).

In general, an FA is fully defined by specifying the following:

1. The set of states S.
2. The alphabet A of the input and output symbols.
3. A function $f: S \times A \to S \times A$. An example is $f(X, j) = (Y, i)$ meaning, if the current state is X and the next input is j, switch to state Y and output i.
4. The initial state S_0.

Mathematically, an FA is a quartet (S, A, f, S_0).

An FA is normally *deterministic* (DFA), since for every state and input there is just one output and one next state. However, there can also be *nondeterministic* FA,

denoted by NFA. Figure F.2d shows an example. When a zero is input in state 2, the NFA can transit to either state 3 or state 4. NFAs are useful in proving theorems and also in the theory of languages and the theory of computations. Following the states and inputs, it is trivial to verify that the input string "10101010..." is accepted by the NFA of Figure F.2d (it alternates between states 1, 2, and 4), as are the strings 100 and 101, but the string 1000 is not accepted. It should be noted that every DFA is an NFA.

◇ **Exercise F.1:** Is the string "1001010010..." accepted by the NFA of Figure F.2d?

The set of all input strings accepted by an FA is called the *language* of the FA. It can be shown that the languages accepted by finite automata are the languages denoted by *regular expressions*, so we conclude this short chapter by defining regular expressions. We start by illustrating the concepts of *concatenation* and *closure*.

Let $L_1 = (10, 1)$ and $L_2 = (011, 11)$ be two sets of symbols (binary strings). The concatenation of L_1 and L_2 is denoted by L_1L_2 and is the set of strings that are concatenations of 10 (or 1) and 011 (or 11). Thus, $L_1L_2 = (10011, 1011, 111)$. The closure of a set L of symbols is denoted by L^*. It is the set of all strings made of symbols from L (including the empty set). Thus, if $L = (10, 11)$, then $L^* = ((), 10, 11, 1010, 1011, 1110, 1111, \ldots)$, where the empty pair of parentheses denotes the empty set. If L is a set of symbols, then the regular expressions over L are defined recursively as follows:

1. The empty symbol is a regular expression.
2. The empty set () is a regular expression.
3. For each $s \in L$, the set (s) is a regular expression.
4. If r and s are regular expressions, then $(r + s)$ (union of sets), (rs) (set concatenation), and (r^*) (set closure) are regular expressions.

Finite automata are used by several compression methods, among them WFA (Section 4.31) and dynamic Markov coding (Section 8.8).

[Hopcroft and Ullman 79] is a detailed introduction to FAs and their properties.

Bibliography

Hopcroft, John E. and Jeffrey D. Ullman (1979) *Introduction to Automata Theory, Languages, and Computation*, Reading, MA, Addison-Wesley.

CAT, n. A soft, indestructible automaton provided by nature to be kicked when things go wrong in the domestic circle.

Ambrose Bierce. *The Devil's Dictionary*