Finite Automata

Initial text by Maggie Johnson.

Introduction

Several children's games fit the following description: Pieces are set up on a playing board; dice are thrown (or a wheel is spun) and a number is generated at random. Based on the generated number, the pieces on the board are rearranged specified by the rules of the game. Then, another child throws or spins and rearranges the pieces again. There is no skill or choice involved - the entire game is based on the values of the random numbers.

Consider all possible positions of the pieces on the board and call them **states**. We begin with the **initial state** of the starting positions of the pieces on the board. The game then changes from one state to another based on the value of the random number. For each possible number, there is one and only one resulting state given the input of the number, and the prior state. This continues until one player wins and the game is over. This is called a **final state**.

Now consider a very simple computer with an input device, a processor, some memory and an output device. We want to calculate 3 + 4, so we write a simple list of instructions and feed them into the machine one at a time (e.g., STORE 3 TO X; STORE 4 TO Y; LOAD X; ADD Y; WRITE TO OUTPUT). Each instruction is executed as it is read. If all goes well, the machine outputs '7' and terminates execution. This process is similar to the board game. The state of the machine changes after each instruction is executed, and each state is completely determined by the prior state and the input instruction (thus this machine is defined as **deterministic**). No choice or skill is involved; no knowledge of the state of the machine 4 instructions ago is needed. The machine simply starts at an initial state, changes from state to state based on the instruction and the prior state, and reaches the final state of writing '7'.

Finite Automata

A general model (of which the previous two examples are instances) of this type of machine is called a **Finite Automaton**; 'finite' because the number of states and the alphabet of input symbols is finite; automaton because the structure or machine (as it is more commonly called) is deterministic, i.e., the change of state is completely governed by the input.

A **finite automata** has:

1) A finite set of states, one of which is designated the initial state or **start state**, and some (maybe none) of which are designated as **final states.**
2) An alphabet $\sum$ of possible input symbols.
3) A finite set of **transitions** that tell for *each* state and for *each* symbol of the input alphabet, which state to go to next.

In the simple computer example, the start state is the original state of the machine before program execution and the final state is '7' written on the output device. The input alphabet is the set of strings representing the instructions.

Example 1
Suppose $\sum$ = {a,b}, the set of states = {x, y, z} with x the start state and z the final state, and we have the following rules of transition:

1) from state x and with input a, goto state y.
2) from state x and with input b, goto state z.
3) from state y and with input a, goto state x.
4) from state y and with input b, goto state z.
5) from state z and with any input, stay at state z.

This is a perfectly defined 'FA' (short for finite automaton) according to the definition above. We now need to examine what happens to various input strings when presented to this FA. Consider 'aaa': we begin in state x and goto state y. The next symbol is also an 'a', so from state y we go to state x. Then on the last 'a', we go from state x back to state y. That's all the input we have - since we do not end up in state z, the final state, we have an unsuccessful end.

This exercise illustrates how an FA can be used to *recognize or accept* strings of a particular language. The set of all strings that leave us in a final state of an FA is called **the language accepted or defined by the FA**. 'aaa' is not a word in the language defined by the FA given above. This is why FA's are also called **language recognizers.**

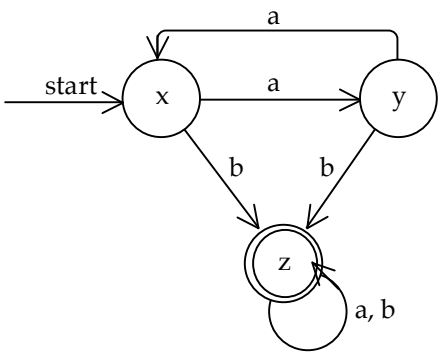
Now, try 'abba': from state x to state y, from state y to state z, from state z to state z, from state z to state z. This word is a part of the language defined by this FA. In fact, any input string that has just one or more will be accepted by this FA. We can define this by the regular expression:

$$(\mathbf{a} + \mathbf{b})^* \, \mathbf{b} \, (\mathbf{a} + \mathbf{b})^*$$

This is a very simple FA. Typically, the list of transition rules can be quite long, so an alternative representation is frequently used. One method is to use a **transition table**:

		a	b
start	x:	y	z
	y:	x	z
final	z:	z	z

This table has all the information required to define an FA. Even though it is no more than a table of symbols or a list of rules, we consider an FA to be a machine, i.e., we understand that an FA has dynamic capabilities. It moves. It processes input. Something goes from state to state as the input is read. One way to represent an FA that feels more like a machine is a **transition diagram:**

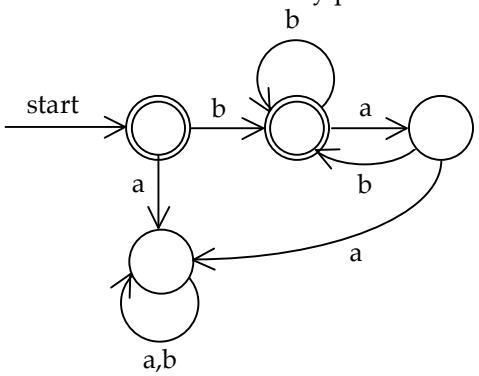


The vertices are the states, and the edges indicate the input into each state. The start state (there's always just one) is explicitly identified, and the final states (also called the accept states) are double circles. This representation makes it much easier to see that this FA accepts only strings with at least one b in them. Technically, the states themselves don't need to be labeled, but very often it helps to label the states with information about what that state represents. Here, the x, y, and x aren't all that useful—they're just holdovers from the transition table version of the FA. But we can label the states with anything we want, and all the rest of the examples here will do just that.

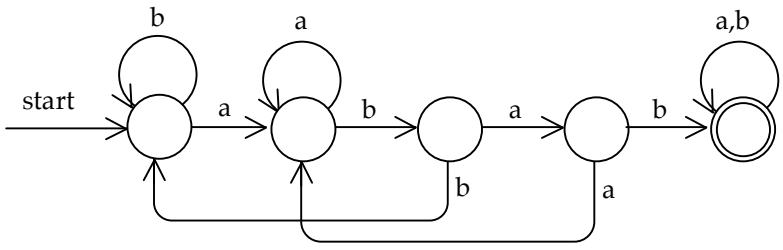
Additional Examples

Construct deterministic finite automata accepting each of the following languages:

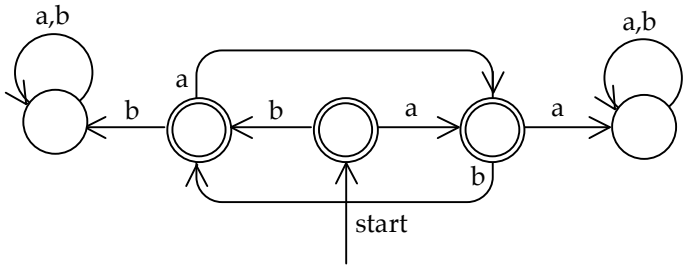
a. $\left\{ w \in (a+b)^* : \text{ each a in w is immediately preceded and immediately followed by a b} \right\}$



b. $\left\{ w \in (a+b)^* : \text{ w included abab as a substring} \right\}$

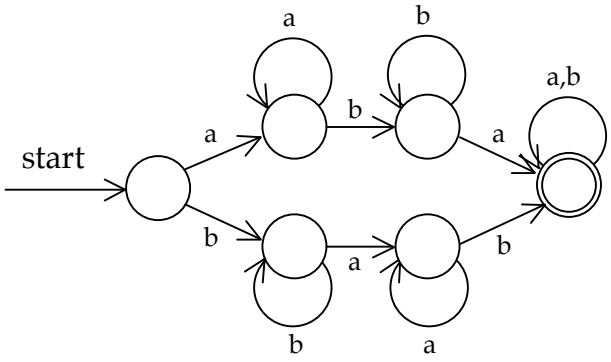


c. $\left\{ w \in (a+b)^* : \text{ w has neither aa nor bb as a substring} \right\}$



d. $\left\{ w \in (a+b)^* : \text{ w includes both ab and ba as (possibly overlapping) substrings} \right\}$

All those strings in the language that start with an a have ab before ba. Those strings that begin with the letter b have ba before ab. So I viewed this language as the union of two languages: $a^+b^+a(a+b)^* + b^+a^+b(a+b)^*$.

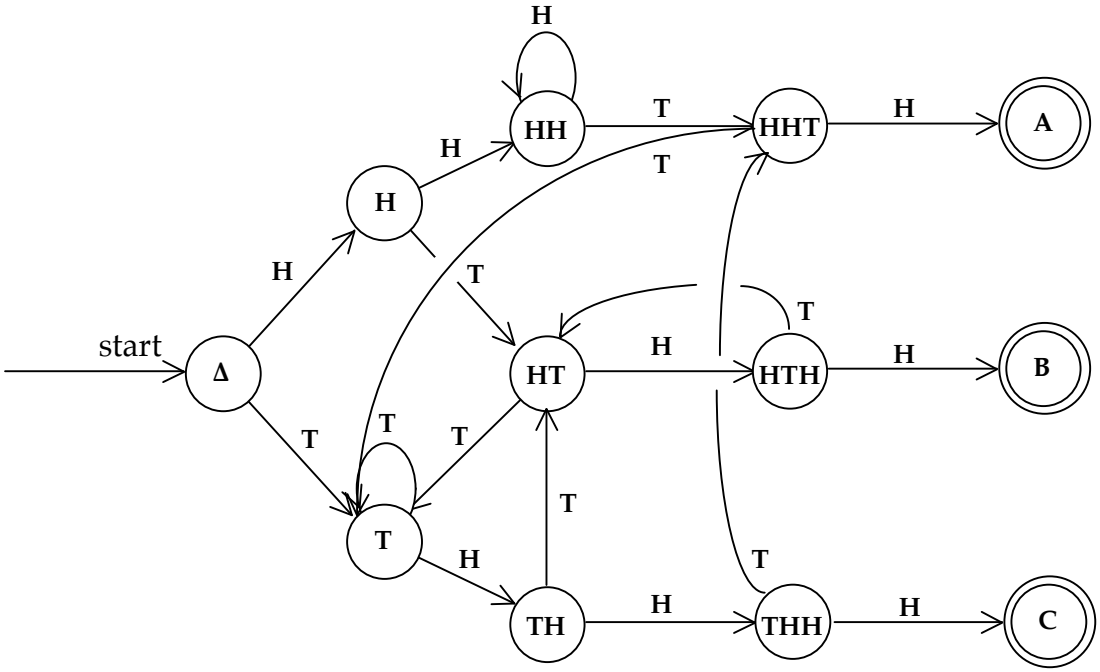


Modeling **Step By Step Processes**

These finite state machines are useful to illustrating the various degrees of progress one can make toward any one of a number of goals. The generalization isn't difficult to follow: rather than framing a computation in terms of string acceptance or rejection, you can decorate transitions with rolls of a single die, or coin flips, or steps in an algorithm that ultimately terminates. Here's a variation on the Laura and Ming problem from our probability days:

The Game: Abigail, Beatrice, and Christopher keep flipping a fair coin until one of their respective patterns **HHTH**, **HTHH**, or **THHH** comes up. Draw a deterministic finite state machine that tracks the progress of the game, where each of three final states corresponds to a win for one of the three players. (You needn't draw edges out of the final states, since at that point the game is over.)

This really isn't that different at all. It's really another string acceptance problem where the letters of our alphabet are **H** and **T**. Here's the automaton that tracks everything for you. By imposing structure on the step-by-step process, we're in a better position to answer questions about the probability that one will win over the other, or the expected number of flips needed before someone wins.



Notice that the state labels really mean something this time—they're used to convey the most exciting stream of flips for the one player closest to winning. A, B, and C are labels that denote that either Abigail, Beatrice, or Chris won the game. And we don't bother with transitions out of the final states, even though the FA is deterministic everywhere else, because the end of the computation comes with the end of the game, not with the end of some string.