Finite Automata

We shall now discover how Lex turns its input program into a lexical analyzer. At the heart of the transition is the formalism known as *finite automata.* These are essentially graphs, like transition diagrams, with a few differences:

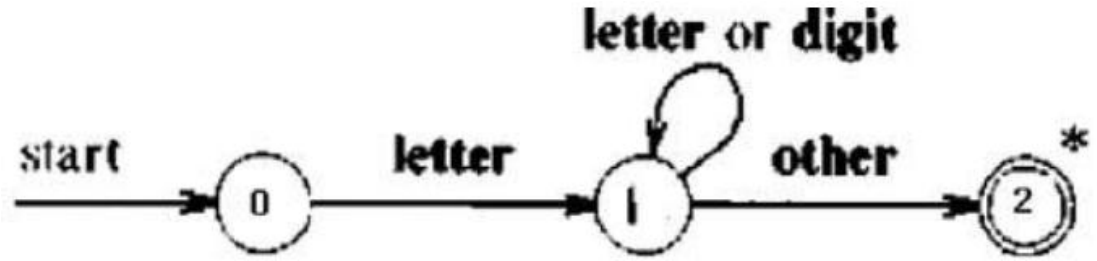
1. Finite automata are *recognizers;* they simply say "yes" or "no" about each possible input string.

2. Finite automata come in two flavors:

(a) *Nondeterministic finite automata* (NFA) have no restrictions on the labels of their edges. A symbol can label several edges out of the same state, and E, the empty string, is a possible label.

(b) *Deterministic finite automata* (DFA) have, for each state, and for each symbol of its input alphabet exactly one edge with that symbol leaving that state.

EX)



State 0 : C = GETCHAR ( )
if LETTER(C) then goto state1
else FAIL ( )
State1 : C= GETCHAR ( )
if LETTER(C) or DIGIT(C) then goto state1
else if DELIMITER(C) then goto state2
else FAIL ( )
State2: RETRACT( )
return(id, INSTALL( ))

Nondeterministic Finite Automata

A nondeterministic finite automation is a mathematical model consists of

1. a set of states S;
2. a set of input symbol, Σ, called the input symbols alphabet.
3. a transition function move that maps state-symbol pairs to sets of states.
4. a state so called the initial or the start state.
5. a set of states F called the accepting or final state.

An NFA can be described by a transition graph (labeled graph) where the nodes are states and the edges shows the transition function. The labeled on each edge is either a symbol in the set of alphabet, Σ, or denoting empty string.

EX) The transition graph for an NFA recognizing the language of regular expression (a | b)* a bb is shown in Fig. 1.1. This abstract example, describing all strings of a's and b's ending in the particular string abb, will be used throughout this section. It is similar to regular expressions that describe languages of real interest, however. For instance, an expression describing all files whose name ends in .o is any* .o, where any stands for any printable character .

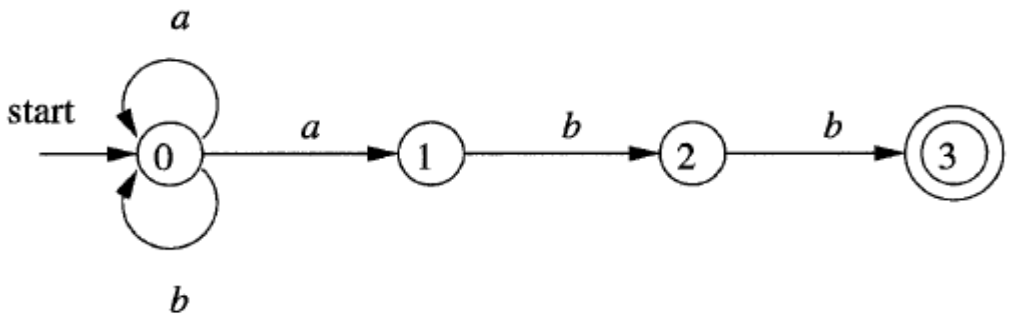


Figure 1.1: A nondeterministic finite automaton

Following our convention for transition diagrams, the double circle around state 3 indicates that this state is accepting. Notice that the only ways to get from the start state 0 to the accepting state is to follow some path that stays in state 0 for a while, then goes to states 1, 2, and 3 by reading abb from the input. Thus, the only strings getting to the accepting state are those that end in abb.

Transition Tables

We can also represent an NFA by a transition table, whose rows correspond to states, and whose columns correspond to the input symbols and c. The entry for a given state and input is the value of the transition function applied to those arguments. If the transition function has no information about that state-input pair, we put **Q)** in the table for the pair

Example: The transition table for the NFA of Fig. 1.1 is shown in Fig.1.2. The transition table has the advantage that we can easily find the transitions on a given state and input. Its disadvantage is that it takes a lot of space, when the input alphabet is large, yet most states do not have any moves on most of the input symbols.

STATE	a	b	ϵ
0	$\{0,1\}$	$\{0\}$	$\emptyset$
1	$\emptyset$	$\{2\}$	$\emptyset$
2	$\emptyset$	$\{3\}$	$\emptyset$
3	$\emptyset$	$\emptyset$	$\emptyset$

Figure 1.2: Transition table for the NFA of Fig. 1.1

The string *aabb* is accepted by the NFA of Fig. **1.1.** The path labeled by *aabb* from state 0 to state **3** demonstrating this fact is:

$$0 \xrightarrow{a} 0 \xrightarrow{a} 1 \xrightarrow{b} 2 \xrightarrow{b} 3$$

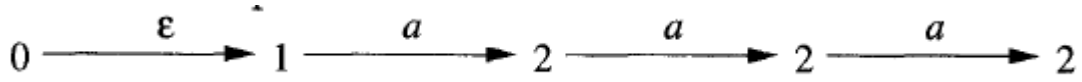
Note that several paths labeled by the same string may lead to different states. For instance, path

$$0 \xrightarrow{a} 0 \xrightarrow{a} 0 \xrightarrow{b} 0 \xrightarrow{b} 0$$

is another path from state 0 labeled by the string *aabb.* This path leads to state 0, which is not accepting. However, remember that an NFA accepts a string as long as *some* path labeled by that string leads from the start state

to an accepting state. The existence of other paths leading to a no accepting state is irrelevant.

Example: Figure **3.26** is an NFA accepting L(aa* **lbb*).** String *aaa* is accepted because of the path



Note that E'S "disappear" in a concatenation, so the label of the path is *aaa.*

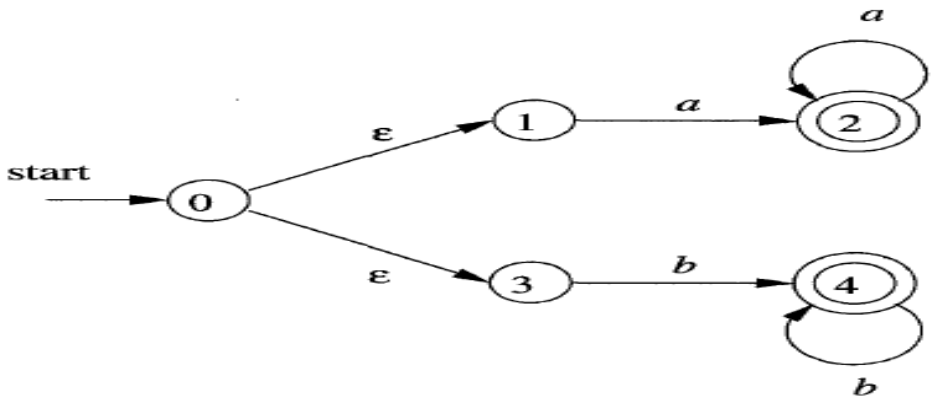


Figure 1.4: NFA accepting aa* / bb*

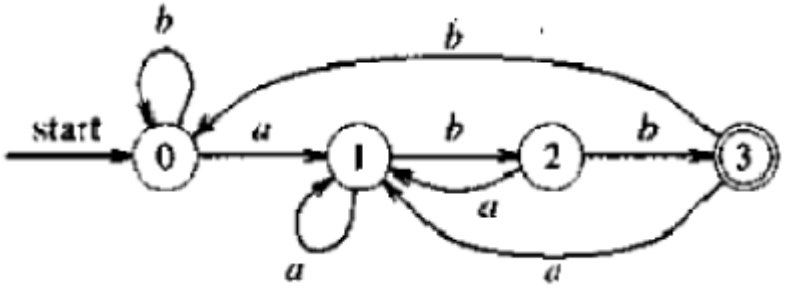
Deterministic Finite Automata

A *deterministic finite automaton* (DFA) is a special case of an NFA where

1. There are no moves on input ***E,*** and

2. For each state *s* and input symbol a, there is exactly one edge out of *s* labeled a. If we are using a transition table to represent a DFA, then each entry is a single state. we may therefore represent this state without the curly braces that we use to form sets. While the NFA is an abstract representation of an algorithm to recognize the strings of a certain language, the DFA is a simple, concrete algorithm for recognizing strings. It is fortunate indeed that every regular expression and every NFA can be converted to a DFA accepting the same language, because it is the DFA that we really implement or simulate when building lexical analyzers. The following algorithm shows how to apply a DFA

Example : Following figure shows a DFA that recognizes the language (a|b)*abb.



The transition table is

state	a	b
0	1	0
1	1	2
2	1	3
3	1	0

With this DFA and the input string "ababb", above algorithm follows the sequence of states: 0,1,2,1,2,3 and returns "yes"

Algorithm for Simulating a DFA

INPUT:

• string x

• a DFA with start state, so . . .

• a set of accepting state's F.o a string.

OUTPUT:

• The answer 'yes' if D accepts x; 'no' otherwise.

The function move (S, C) gives a new state from state S on input character C.

The function 'nextchar' returns the next character in the string.

Initialization:

```
S := S0
C := nextchar;
while not end-of-file do
S := move (S, C)
C := nextchar;
end
If S is in F then
return "yes"
else
return "No".
```