Chapter 5

Finite Automata

5.1 Introduction

- Modern computers are often viewed as having three main components:
  1. the central processing unit (CPU)
  2. memory
  3. input-output devices (IO)
- The CPU is the "thinker"
  1. Responsible for such things as individual arithmetic computations and logical decisions based on particular data items.
  2. However, the amount of data the unit can handle at any one time is fixed forever by its design.
  3. To deal with more than this predetermined, limited amount of information, it must ship data back and forth, over time, to and from the memory and IO devices.
- Memory
  1. The memory may in practice be of several different kinds, such as magnetic core, semiconductor, disks, and tapes.
  2. The common feature is that the information capacity of the memory is vastly greater than what can be accommodated, at any one instant of time, in the CPU.

3. Therefore, this memory is sometimes called *auxilliary*, to distinguish it from the limited storage that is part of the CPU.
4. At least in theory, the memory can be expanded without limit, by adding more core boxes, more tape drives, etc.

- IO devices are the means by which information is communicated back and forth to the outside world; e.g.,

1. terminals
2. printers
3. tapes

We now will study a severely restricted model of an actual computer called a *finite automaton* (FA).

- Like a real computer, it has a central processor with fixed finite capacity, depending on its original design.
- Unlike a real computer, it has no auxiliary memory at all.
- It receives its input as a string of characters.
- It delivers no output at all, except an indication of whether the input is considered acceptable.
- It is a language-recognition device.

Why should we study such a simple model of computer with no memory?

- Actually, finite automata do have memory, but the amount they have is fixed and cannot be expanded.
- Finite automata are applicable to the design of several common types of computer algorithms and programs.
  - For example, the lexical analysis phase of a compiler is often based on the simulation of a finite automaton.
  - The problem of finding an occurrence of one string within another — for example, a particular word within a large text file — can also be solved efficiently by methods originating from the theory of finite automata.

To introduce finite automata, consider the following scenario:

- Play a board game in which two players move pieces around different squares.
- Throw dice to determine where to move.
- Players have no choices to make when making their move. The move is completely determined by the dice.
- A player wins if after 10 throws of the dice, his piece ends up on a certain square.
- Note that no skill or choice is involved in the game.
- Each possible position of pieces on the board is called a *state*.
- Every time the dice are thrown, the state changes according to what came up on the dice.
- We call the winning square a *final state* (also known as a halting state, terminal state, or accepting state).
- There may be more than one final state.

Let's look at another simple example

- Suppose you have a simple computer (machine), as described above.
- Your goal is to write a program to compute $3 + 4$.
- The program is a sequence of instructions that are fed into the computer one at a time.
- Each instruction is executed as soon as it is read, and then the next instruction is read.
- If the program is correct, then the computer outputs the number 7 and terminates execution.
- We can think of taking a snapshot of the internals (i.e., contents of memory, etc.) of the computer after every instruction is executed.

- Each possible configuration of 0's and 1's in the cells of memory represents a different *state* of the system.
- We say the machine ends in a *final state* (also called a halting, terminal, or accepting state) if when the program finishes executing, it outputs the number 7.
- Two machines are in the same state if their output pages look the same and their memories look the same cell by cell.
- The computer is *deterministic*, i.e., on reading one particular input instruction, the machine converts itself from one given state to some particular other state (which is possibly the same), where the resultant state is completely determined by the prior state and the input instruction. No choice is involved.
- The success of the program (i.e., it outputs 7) is completely determined by the sequence of inputs (i.e., the lines of code).
- We can think of the set of all computer instructions as the letters of an alphabet.
- We can then define a language to be the set of all words over this alphabet that lead to success.
- This is the language with words that are all programs that print a 7.

5.2 Finite Automata

Definition: A *finite automaton* (FA), also known as a *finite acceptor*, is a collection $M = (K, \Sigma, \pi, s, F)$ where :

1. K is a *finite* set of states.
   - Exactly one state $s \in K$ is designated as the *initial state* (or *start state*).
   - Some set $F \subset K$ is the set of *final states*, where we allow $F = \emptyset$ or $F = K$ or F could be any other subset of K.

2. An alphabet Σ of possible input letters, from which are formed strings, that are to be read one letter at a time.

3. $\pi : K \times \Sigma \to K$ is the *transition function*.

   - In other words, for each state and for each letter of the input alphabet, the function π tells which (one) state to go to next; i.e., if $x \in K$ and $\ell \in \Sigma$, then $\pi(x, \ell)$ is the state that you go to when you are in state x and read in ℓ.
   - For each state x and each letter $\ell \in \Sigma$, there is exactly one arc leaving x labeled with ℓ.
   - Thus, there is no choice in how to process a string, and so the machine is *deterministic*.

An FA works as follows:

- It is presented with an input string of letters.
- It starts in the start state.
- It reads the string one letter at a time, starting from the left.
- The letters read in determine a sequence of states visited.
- Processing ends after the last input letter has been read.
- If after reading the entire input string the machine ends up in a final state, then the input string is accepted. Otherwise, the input string is rejected.

Example: Consider an FA with three states (x, y, and z) with input alphabet $\Sigma = \{a, b\}$.

Define the following *transition table* for the FA:

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

Input the string *aaaa* to the FA:

- Start in state x and read in first a, which takes us to state y.
- From state y, read in second a, which takes us to state x.
- From state x, read in third a, which takes us to state y.
- From state y, read in fourth a, which takes us to state x.
- No more letters in input string so stop.

Note that on input *aaaa*,

- We ended up in state x, which is not a final state.
- we say that *aaaa* is *not accepted* or *rejected* by this FA.

Now consider the input string *abab*:

- Start in state x and read in first a, which takes us to state y.
- From state y, read in second letter, which is b, which takes us to state z.
- From state z, read in third letter, which is a, which takes us to state z.
- From state z, read in fourth letter, which is b, which takes us to state z.
- No more letters in input string so stop.

On the input string *abab*:

- We ended up in state z, which is a final state.
- we say that *abab* is *accepted* by this FA.

Definition: The set of all strings accepted is the *language associated* with or *accepted* by the FA.

Note that

- the above FA accepts all strings that have the letter b in them and no other strings.

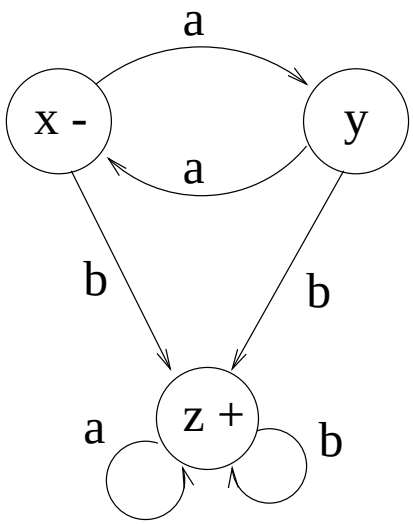
- the language accepted by this FA is the one defined by the regular expression

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

Can also draw *transition diagram*:

- directed graph
- directed edge
- every state has as many *outgoing edges* as there are letters in the alphabet.
- it is possible for a state to have no *incoming edges*.
- the start state is labeled with a $-$.
- final states are labeled with a $+$.
- some states are neither labeled with $-$ or $+$.

Example: From before.

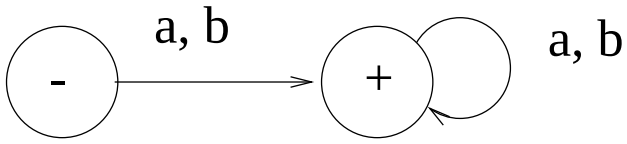


5.3 Examples of FA

Example: regular expression

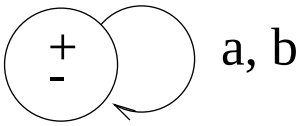
$$(\mathbf{a}+\mathbf{b})(\mathbf{a}+\mathbf{b})^* = (\mathbf{a}+\mathbf{b})^+$$

All strings over the alphabet $\Sigma = \{a, b\}$ except Λ.



Example: regular expression

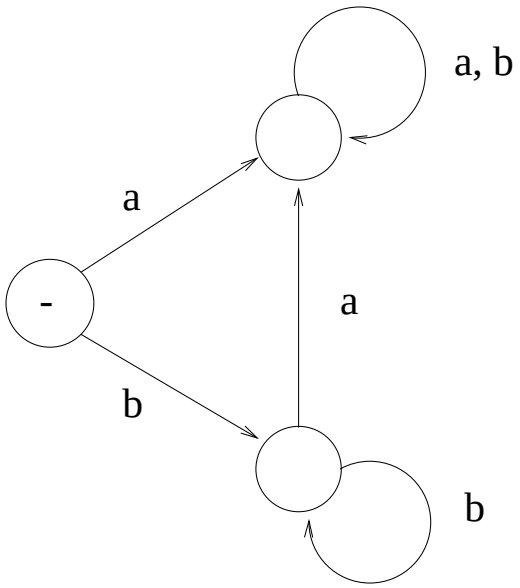
$$(\mathbf{a} + \mathbf{b})^*$$



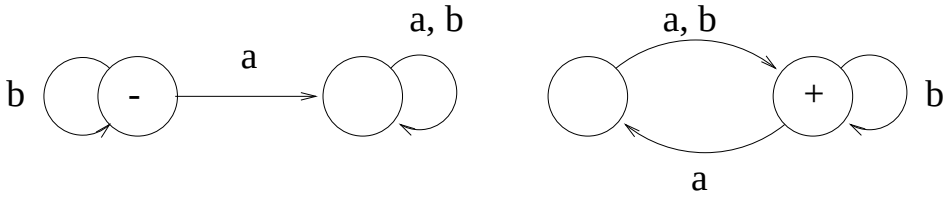
This FA accepts all strings over the alphabet $\Sigma = \{a, b\}$ including Λ.

There are FA's that accept the language having no words:

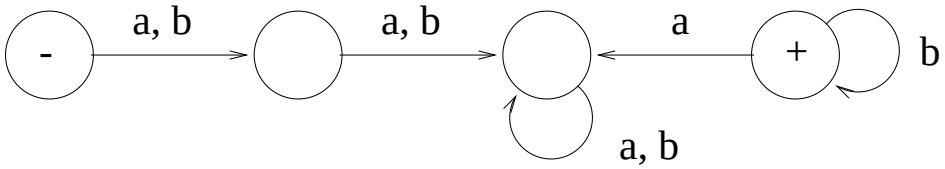
- FA has no final states



- Final state cannot be reached from start state because graph disconnected.

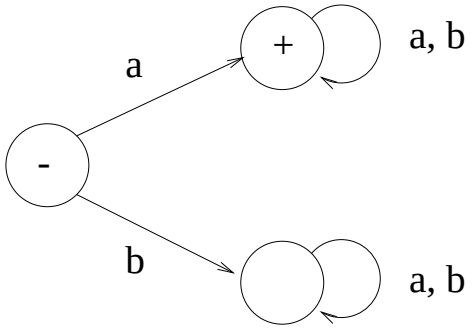


- Final state cannot be reached from start state because no path

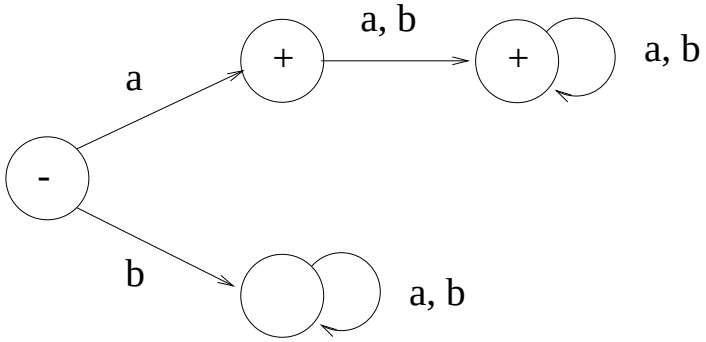


Example: Build FA to accept all words in the language

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



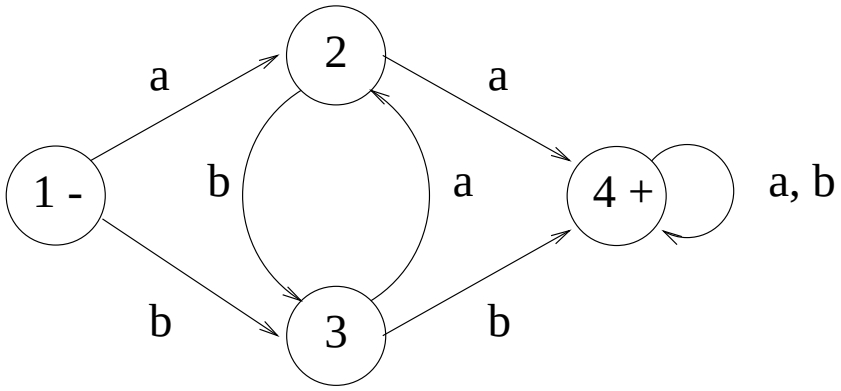
or



Note that

- more than one possible FA for any given language
- can have more than one final state

Example:

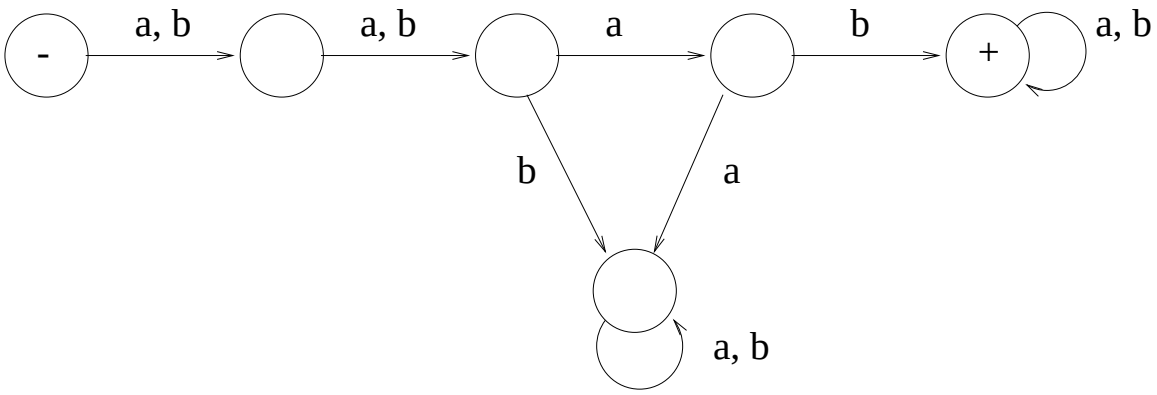


Note that

- *ababa* is not accepted.
- *baaba* is accepted.
- FA accepts strings that have a double letter
- Regular expression of language

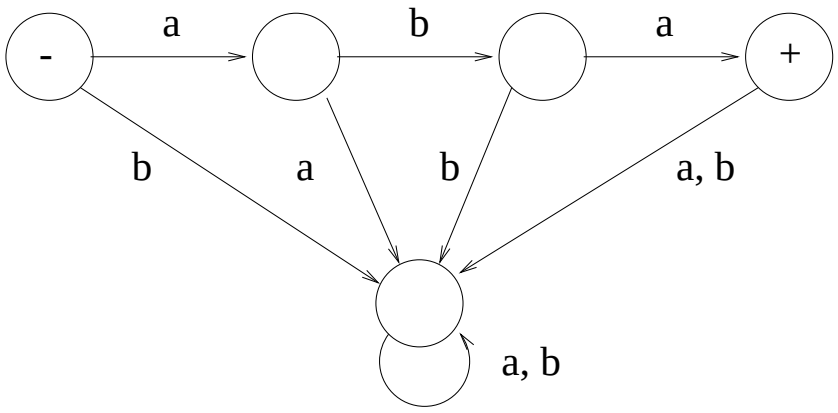
$$(\mathbf{a} + \mathbf{b})^*(\mathbf{aa} + \mathbf{bb})(\mathbf{a} + \mathbf{b})^*$$

Example:

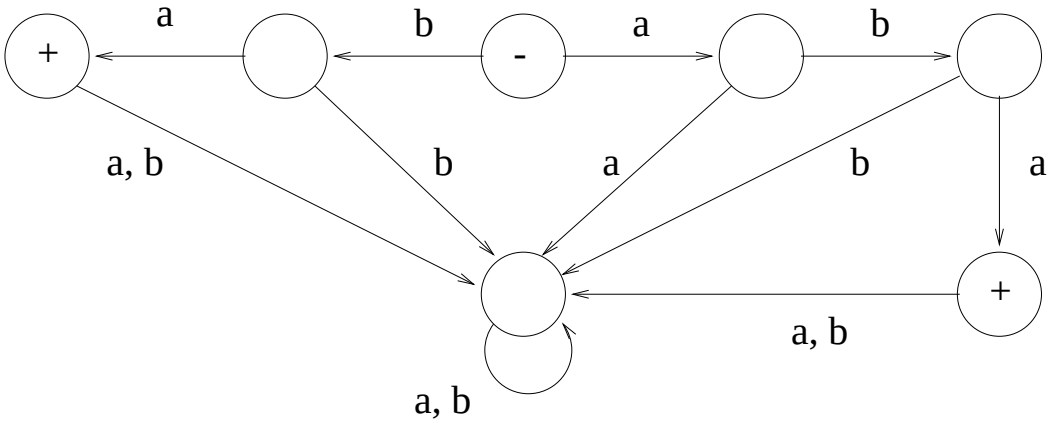


- Only accepts words whose third and fourth letters are ab.
- Rejects all other words
- Regular expressions:
  1. $(\mathbf{aaab} + \mathbf{abab} + \mathbf{baab} + \mathbf{bbab})(\mathbf{a} + \mathbf{b})^*$
  2. $(\mathbf{a} + \mathbf{b})(\mathbf{a} + \mathbf{b})\mathbf{ab}(\mathbf{a} + \mathbf{b})^*$

Example: Only accepts the word *aba*.



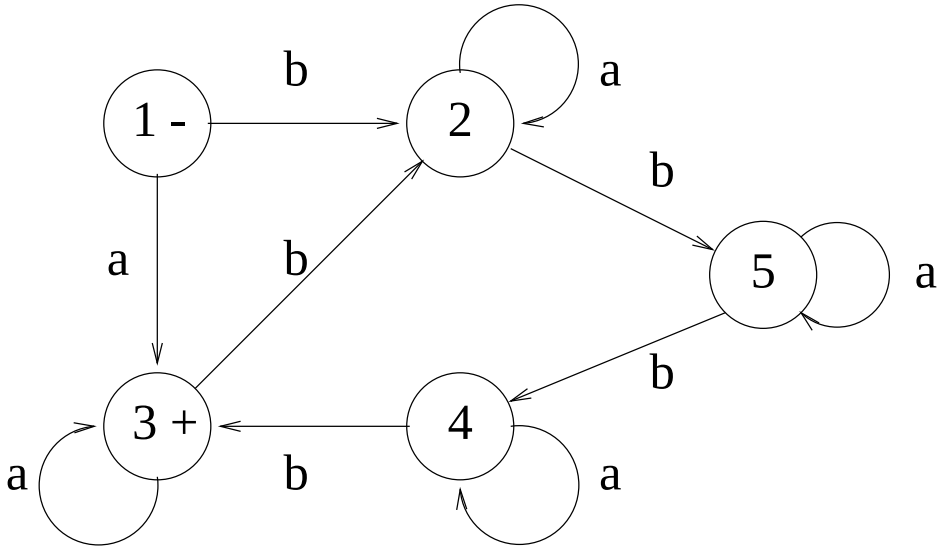
Example: Only accepts the words *aba* and *ba*.



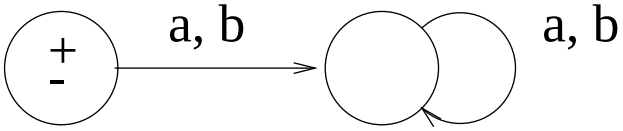
Example: Regular expression:

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

Language with words having at least one letter and the number of b's divisible by 4.



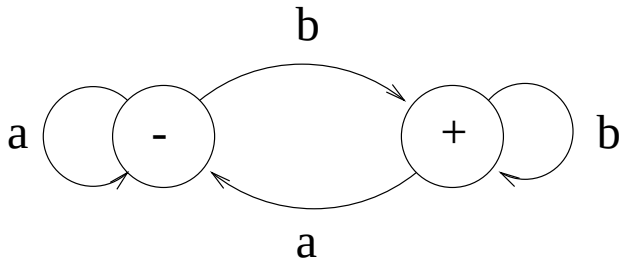
Example: Only accepts the word Λ.



Example: Regular expression:

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

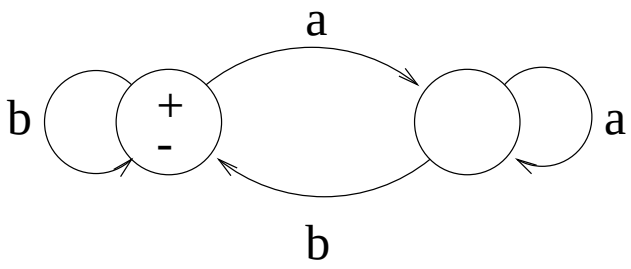
- Words that end with b
- does not include Λ.



Example: Regular expression:

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

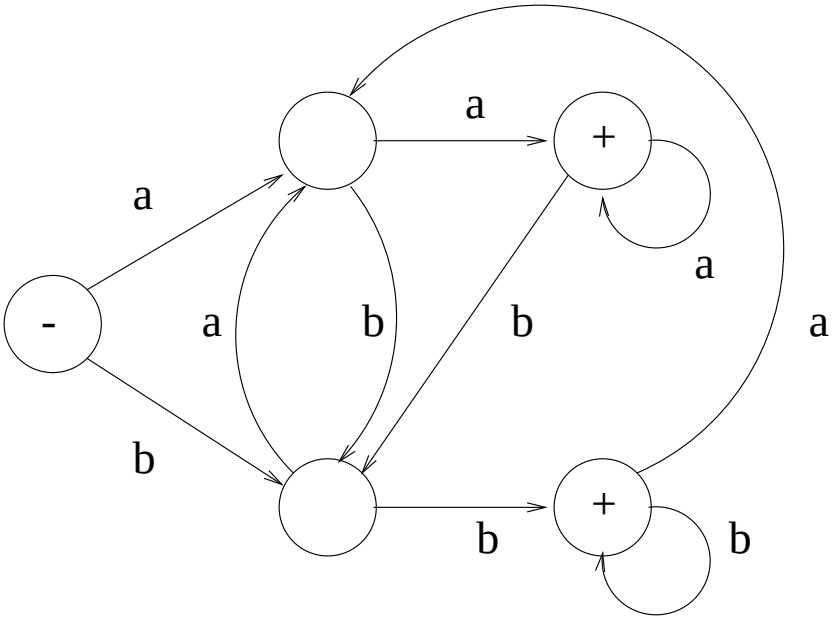
Either Λ or words that end in b; i.e., words that do not end in a.



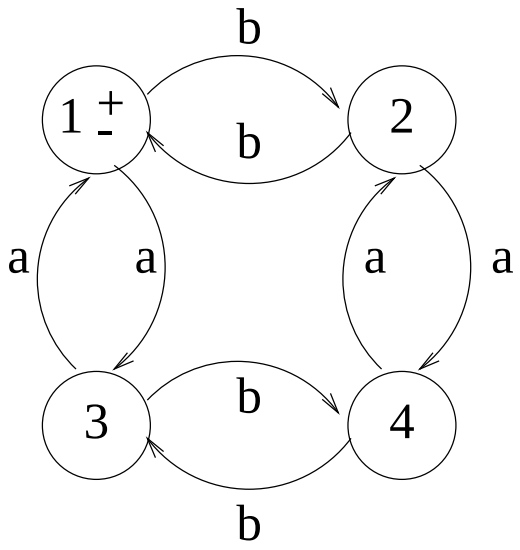
Example: Regular expression:

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

Words that end in a double letter.



Example: EVEN-EVEN



Note that

- Every b moves us either left or right.
- Every a moves us either up or down.