Theory of Automata
Finite Automata

Week 3

Contents

- Finite Automata - Introduction
- Examples
- Model & Definition
- Abstract/Formal Definition
- Transition Table & Transition Diagrams
- FAs & their languages
- Discrete Finite Automata

Finite Automata (FA) - Introduction

- Finite State Automata (FSA) or Finite State Machine (FSM)
  - An FA is a model of a system with discrete inputs and outputs
  - The system can be in any ONE of a finite number of the internal configurations or States.
  - The states of the system summarizes the information concerning the past inputs that is needed to determine the behavior of the system on subsequent inputs.
  - The system is changing the state as a result of new inputs.

Examples

- ON/OFF switch
- Control mechanism of an Elevator
  - It has finite set of states, the floors, to move to
  - It has finite set of inputs,
    - the call buttons on each floor and
    - the floor buttons in the carriage
  - The system only knows the current state/floor it is on
  - The after receiving an input the control determines the next state/floor to move to and the direction of the movement in relation to the current floor.

A News Paper wending Machine

- Input to machine consists of Nickels(5), Dimes(10) and Quarters(25)
- When 30 cents are inserted, the cover may be opened and a news paper removed
- If total of coins exceed 30 cents, the machine accepts the over payment and does not give change.
- Machine has no additional memory, however it knows that an additional 5 cents will unlatch the cover when 25 cents has previously been inserted.

- What is the language of the machine?
- All strings of *n, d, q* representing sum of 30 cents or more
- What are possible states?
- Need 30 cents, needs 25 cents, ----, needs 5 cents, needs 0 cent, to open the latch
- What is the final state?
- need 0 cents

News Paper Wending Machine FA Model

The Model

- Σ = {n, d , q}
- Q = { 0, 5,10, 15, 20, 25, 30)
- F = { 0 }
- S = { 30 }

δ	n	d	q
0	0	0	0
5	0	0	0
10	5	0	0
15	10	5	0
20	15	10	0
25	20	15	0
30	25	20	5

Definition

A **finite automaton** is a collection of three things:

1. A finite set of states, **one** of which is designated as the initial state, called the **start state**, and **some** (maybe **none**) of which are designated as **final states**.

2. An **alphabet Σ** of possible input letters.

3. A finite set of **transitions** that tell for each state and for each letter of the input alphabet which state to go next.

How Does a Finite Automaton work?

- Finite Automaton (FA) works by being presented with an input string of letters that it reads letter by letter starting from the leftmost letter of the input;
- Beginning at the start state, the letters read from input determine a sequence of states and guide the movement of the control along the path in the FA.
- This sequence of states ends when the last input letter has been read and the movement of the control stops.
- If the current state in which control happen to be one of the final states, the input is accepted as valid string from the language of the FA.

Example

Consider the following FA:

- The input alphabet has only the two letters a and b. (We usually use this alphabet throughout the chapter.)

- There are only three states, *x*, *y* and *z*, where *x* is the start state and *z* is the final state.

- The transition list for this FA is as follows:
  - Rule 1: From state *x* and input *a*, go to state y.
  - Rule 2: From state *x* and input *b*, go to state *z*.
  - Rule 3: From state *y* and input *a*, go to state *x*.
  - Rule 4: From state *y* and input *b*, go to state *z*.
  - Rule 5: From state *z* and any input, stay at state *z*.

Example Contd.

- Let us examine what happens when the input string '*aaa*' is presented to this FA.
- First input a: state *x* → *y* by Rule 1.
- Second input *a*: state *y* → *x* by Rule 3.
- Third input *a*: state *x* → *y* by Rule 1.
- We did not finish up in the final state z, and therefore have an unsuccessful termination.

Example contd.

- The set of all strings that lead to a final state is called the language defined by the finite automaton.
- Thus, the string *aaa* is not in the language defined by this FA.
- We may also say that the string aaa is not accepted by this FA, or the string *aaa* is rejected by this FA.
- The set of all strings accepted is also called the language associated with the FA.
- We also say, “This FA accepts the language L”, or “L is the language accepted by this FA”, or “L is the language of the FA”, by which we mean that all the words in L are accepted, and all the inputs accepted are words in L

Example contd.

- It is not difficult to find the language accepted by this FA.
- If an input string is made up of only letter a's then the action of the FA will be to jump back and forth between state x and state y.
- To get to state z, it is necessary for the string to have the letter b in it. As soon as a b is encountered, the FA jumps to state z. Once in state z, it is impossible to leave. When the input string runs out, the FA will be in the final state z.
- This FA will accept all strings that have the letter b in them. Hence, the language accepted by this FA is defined by the regular expression

$$(a + b)^*b(a + b)^*$$

Abstract/Formal definition of FA

1. A finite set of states Q = $\{q_0, q_1, q_2\ q_3 \ldots\}$ of which q_0 is start state.

1. A subset of Q called final state (s).

1. An alphabet $\sum = \{ x_1, x_2, x_3, \ldots\}$.

1. A transition function δ associating each pair of state and letter with a state:

   $\delta(q,x_j) = x_k$

Transition Table

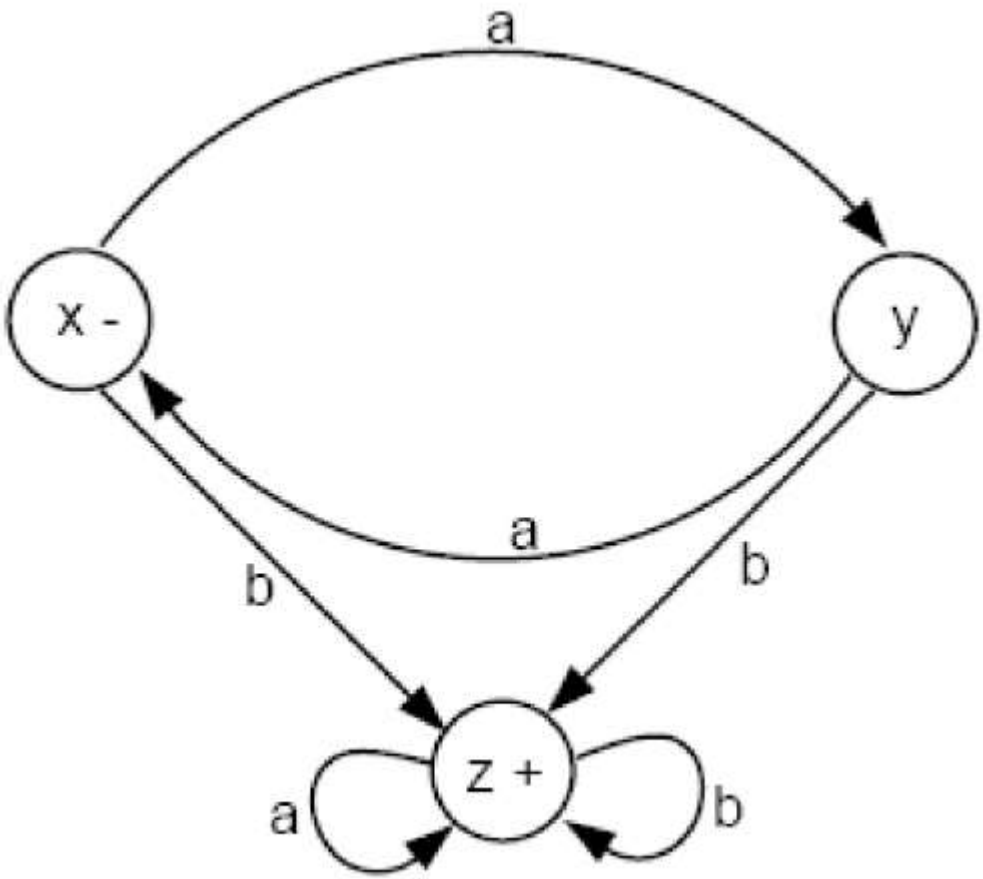
- The transition list can be summarized in a table format in which each row is the name of one of the states, and each column is a letter of the input alphabet.
- For example, the **transition table** for the FA above is

	a	b
Start x	y	z
y	x	z
Final z	z	z

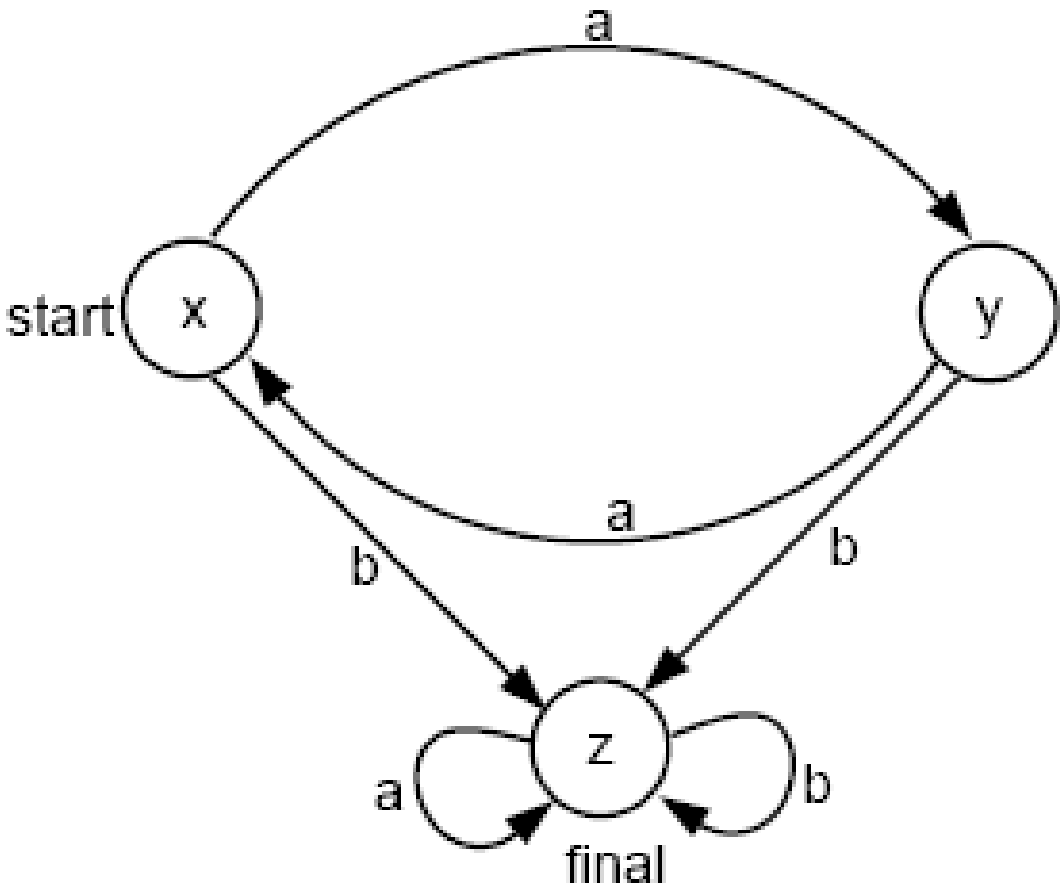
Transition Diagrams

- Pictorial representation of an FA gives us more of a feeling for the motion.
- We represent each state by a small **circle**.
- We draw **arrows** showing to which other states the different **input letters** will lead us. We label these arrows with the corresponding input letters.
- If a certain letter makes a state go back to itself, we indicate this by a **loop**.
- We indicate the start state by a **minus sign**, or by labeling it with the word **start**.
- We indicate the final states by **plus signs**, or by labeling them with the word **final**.
- Sometimes, a start state is indicated by **an arrow**, and a final state is indicated by drawing **concentric circles.**

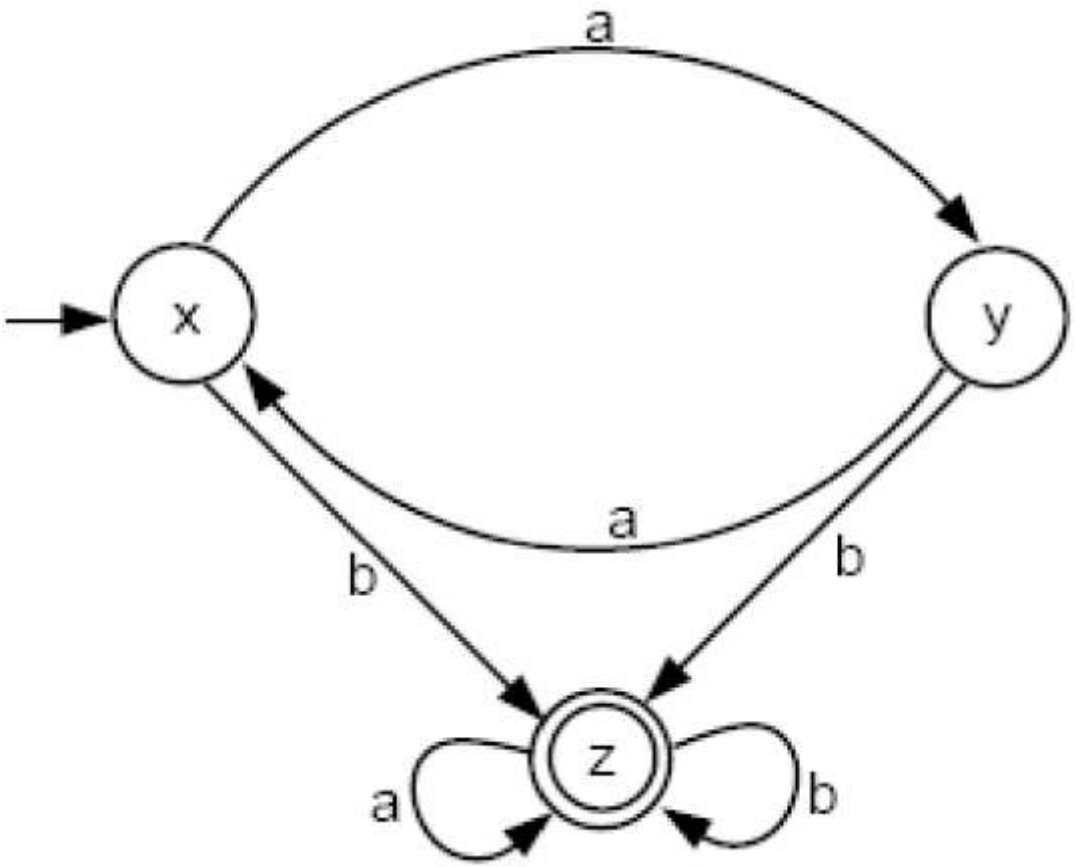
Transition Diagram (cont.)



Transition Diagram (cont.)



Transition Diagram (cont.)

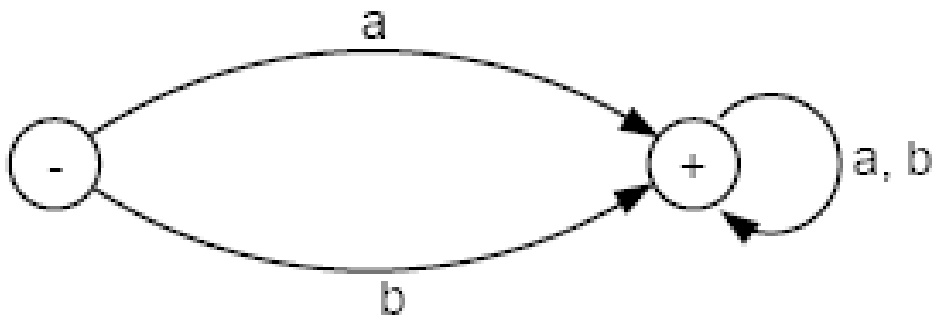


Transition Diagrams contd.

- When we depict an FA as circles and arrows, we say that we have drawn a **directed graph**.
- We borrow from Graph Theory the name **directed edge**, or simply **edge**, for the arrow between states.
- ***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 or to have many.**

Example – Null String as an Input

- *By convention, we say that the null string starts in the start state and ends also in the start state for all FAs.*
- Consider this FA:



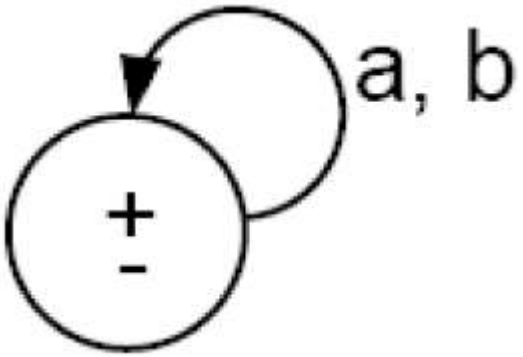
- The language accepted by this FA is the set of all strings except **Λ**.

The regular expression of this language is

$$(a + b)(a + b)^* = (a + b)^+$$

Example

- One of many FAs that accept all words is



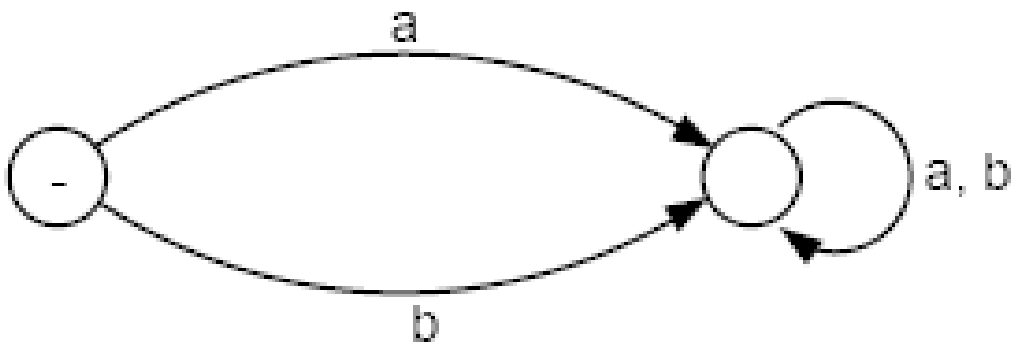
- Here, the ± means that the same state is both a start and a final state.
- The language for this machine is

  (a + b)*

Example

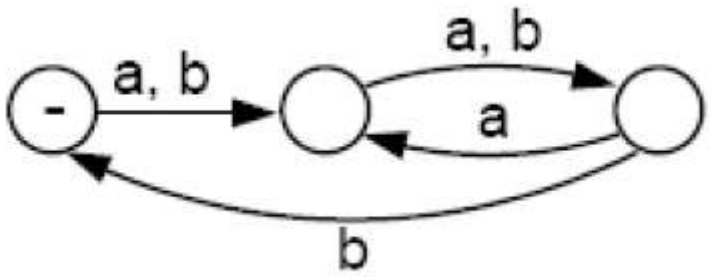
- There are FAs that accept no language. These are of two types:

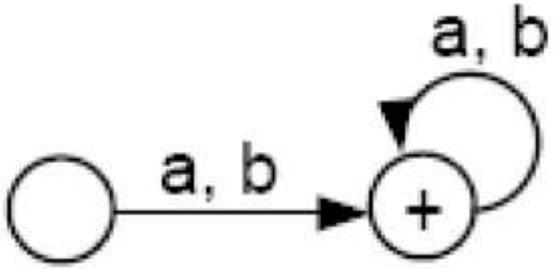
1. The first type includes FAs that have no final states, such as



Example

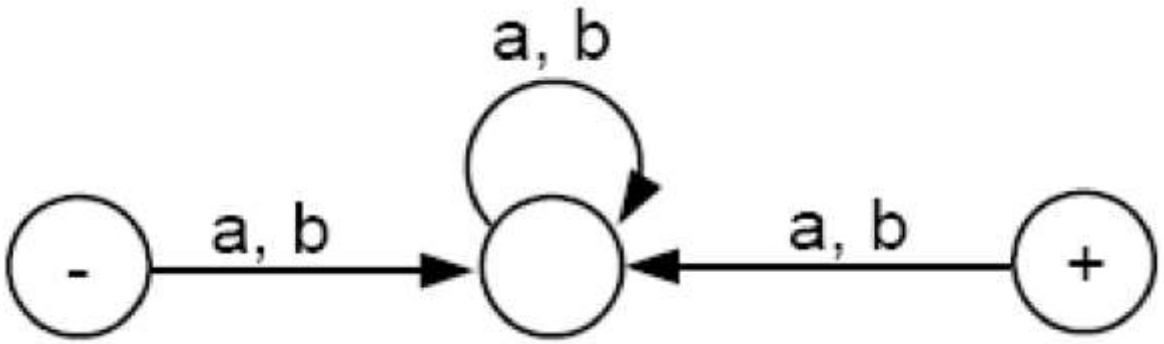
2. The second type include FAs of which the final states can not be reached from the start state.
    1. This may be either because the diagram is in two separate components. In this case, we say that the graph is **disconnected**, as in the example below:





Example

2. Or it is because the final state has no incoming edges, as shown below:



FA and their Languages

- We will study FA from two different angles:

1. Given a language, can we build a machine for it?

2. Given a machine, can we deduce its language?

Example

- Let us build a machine that accepts the language of all words over the alphabet Σ = {a, b} with an **even number of letters**.

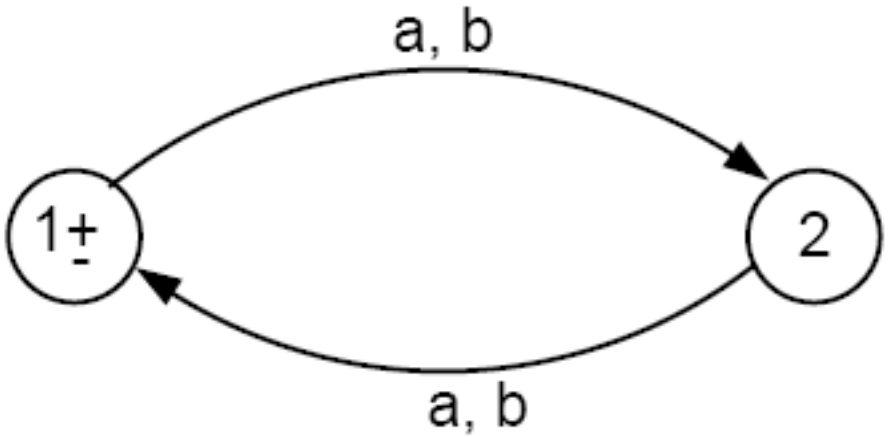
- A mathematician could approach this problem by counting the total number of letters from left to right. A computer scientist would solve the problem differently since it is not necessary to do all the counting:

- Use a Boolean flag, named E, initialized with the value TRUE. Every time we read a letter, we reverse the value of E until we have exhausted the input string. We then check the value of E. If it is TRUE, then the input string is in the language; if FALSE, it is not.

- The FA for this language should require only 2 states:
  - State 1: E is TRUE. This is the start and also final state.
  - State 2: E is FALSE.

Example Contd.

- So the FA is pictured as follows:



Example

- Let us build a FA that accepts all the words in the language

  $$a(a + b)^*$$

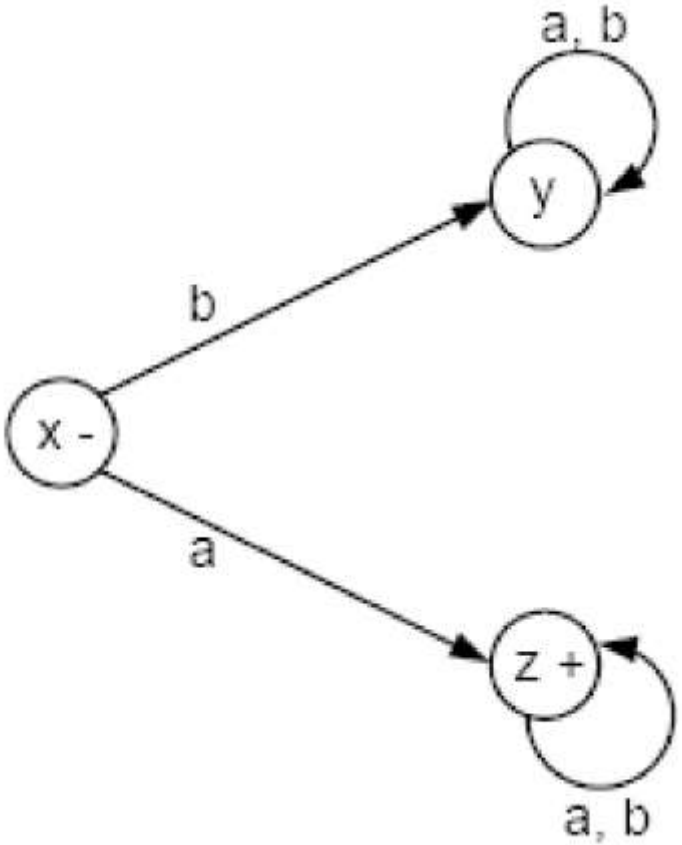
- This is the language of all strings that begin with the letter a.

- Starting at state x, if we read a letter b, we go to a **dead-end** state y. *A dead-end state is one that no string can leave once it has entered.*

- If the first letter we read is an a, then we go to the dead-end state z, which is also a final state.

Example

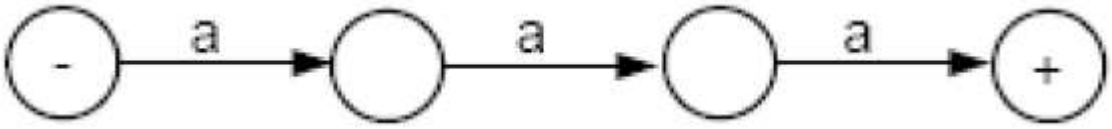
- The machine looks like this:



Example

- Let's build a machine that accepts all words **containing a triple letter**, either *aaa* or *bbb*, and only those words.

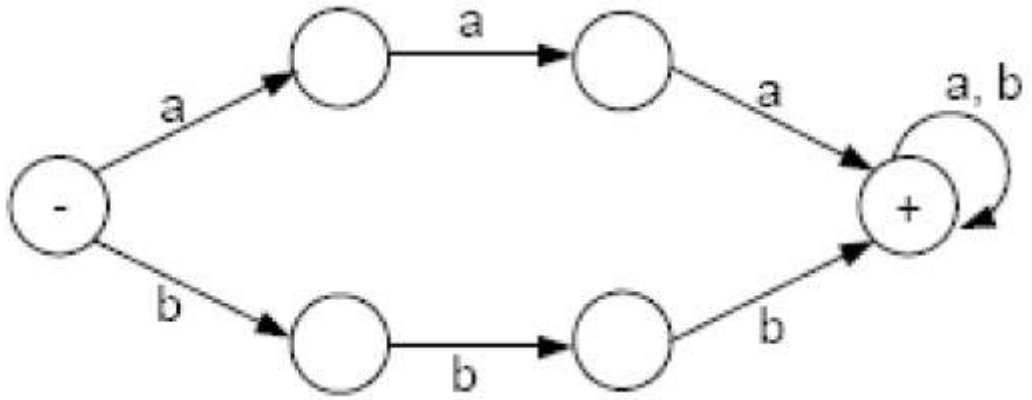
- From the start state, the FA must have a path of three edges, with no loop, to accept the word *aaa*. So, we begin our FA with the following:



Example Contd.

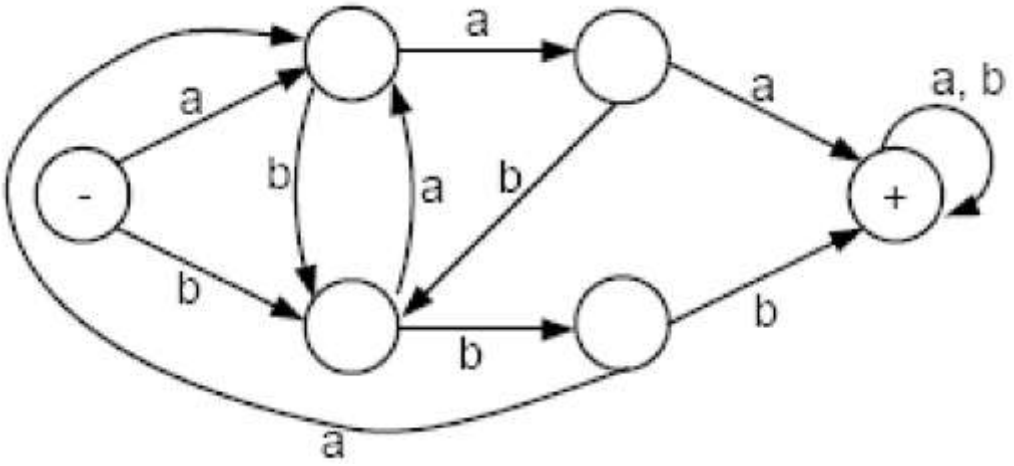
- For similar reason, there must be a path for bbb, that has no loop, and uses entirely differently states. If the b-path shares any states with the a-path, we could mix a's and b's to get to the final state. However, the final state can be shared.

- So, our FA should now look like:



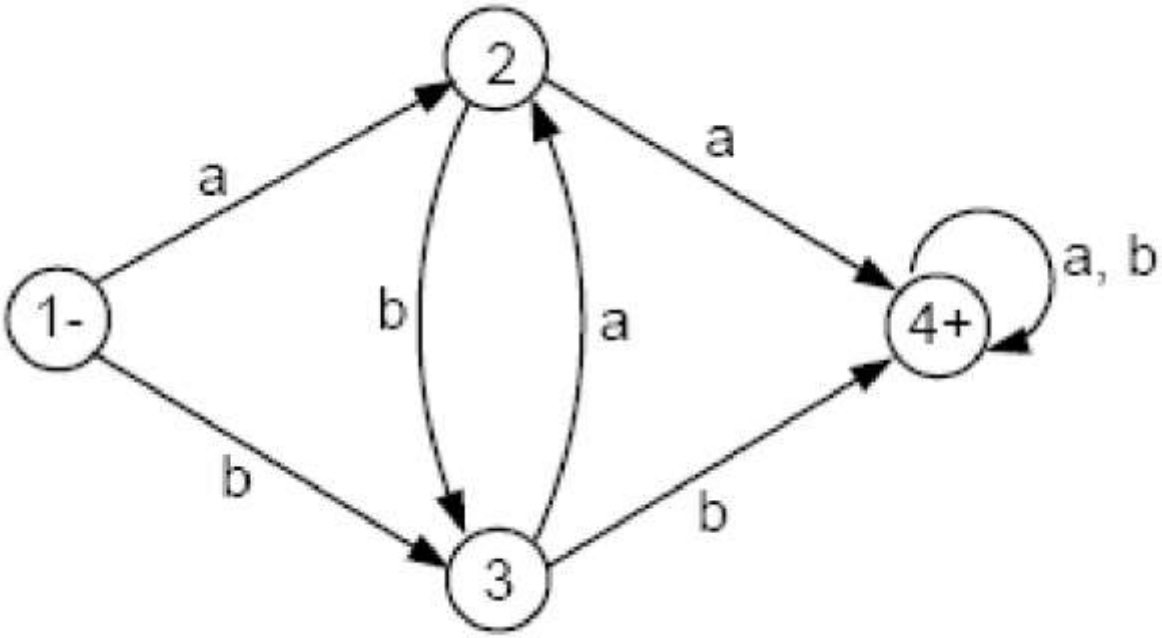
Example Contd.

- If we are moving along the a-path and we read a *b* before the third a, we need to jump to the *b*-path in progress and vice versa. The final FA then looks like this:



Example

- Consider the FA below. We would like to examine what language this machine accepts.



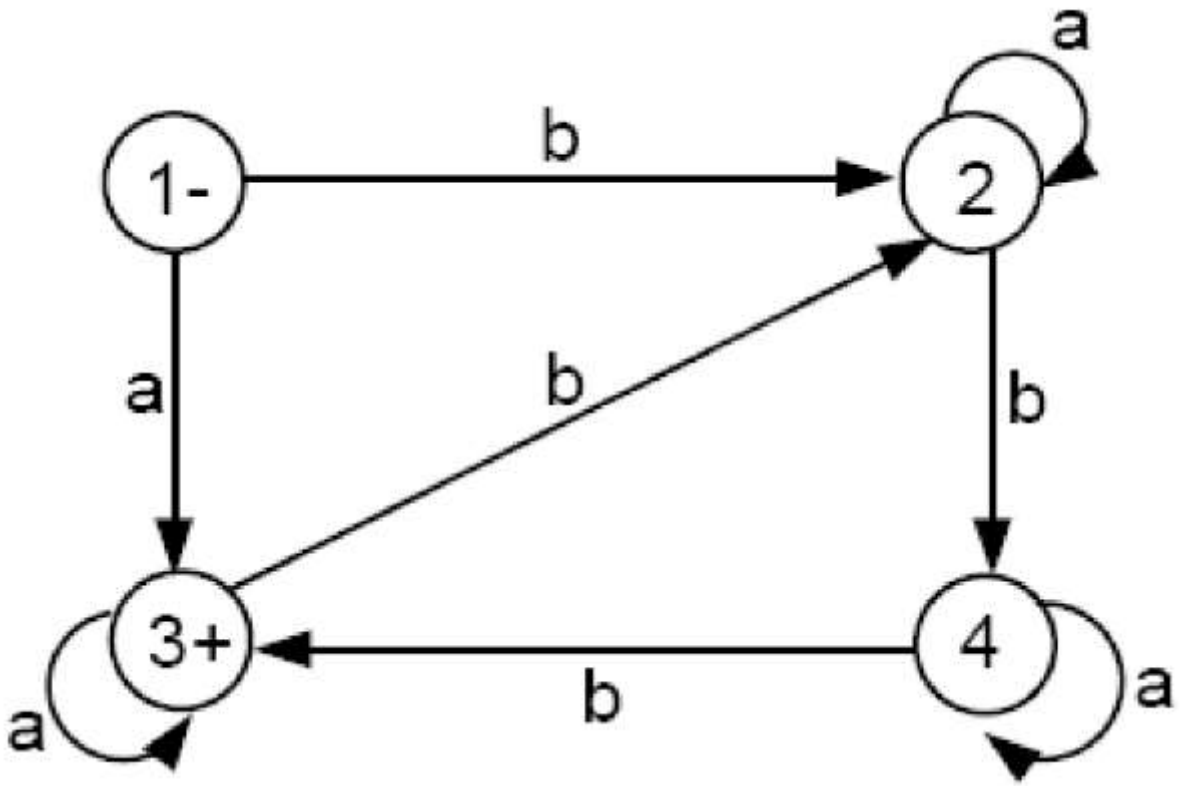
Example

- There are only two ways to get to the final state 4 in this FA: One is from state 2 and the other is from state 3.
- The only way to get to state 2 is by reading an **a** while in either state 1 or state 3. If we read another **a** we will go to the final state 4.
- Similarly, to get to state 3, we need to read the input letter **b** while in either state 1 or state 2. Once in state 3, if we read another **b**, we will go to the final state 4.
- Thus, the words accepted by this machine are exactly those strings that have a double letter *aa* or *bb* in them. This language is defined by the regular expression

$$(a + b)^*(aa + bb)(a + b)^*$$

Example

- Consider the FA below. What is the language accepted by this machine?



Example

- Starting at state 1, if we read a word beginning with an **a,** we will go straight to the final state 3. We will stay in state 3 as long as we continue to read only **a**'s. Hence, all words of the form **aa** are accepted by this FA.

- What if we began with some a's that take us to state 3 and then we read a b? This will bring us to state 2. To get back to the final state 3, we must proceed to state 4 and then state 3. This trip requires two more b's.

- Notice that in states 2, 3, and 4, all a's that are read are ignored; and only b's cause a change of state.

- Summarizing what we know: If an input string starts with an a followed by some b's, then it must have 3 b's to return to the final state 3, or 6 b's to make the trip twice, or 9 b's, or 12 b's and so on.

- In other words, an input string starting with an a and having a total number of b's **divisible by 3** will be accepted. If an input string starts with an a but has a total number of b's not divisible by 3, then it is rejected because its path will end at either state 2 or 4.

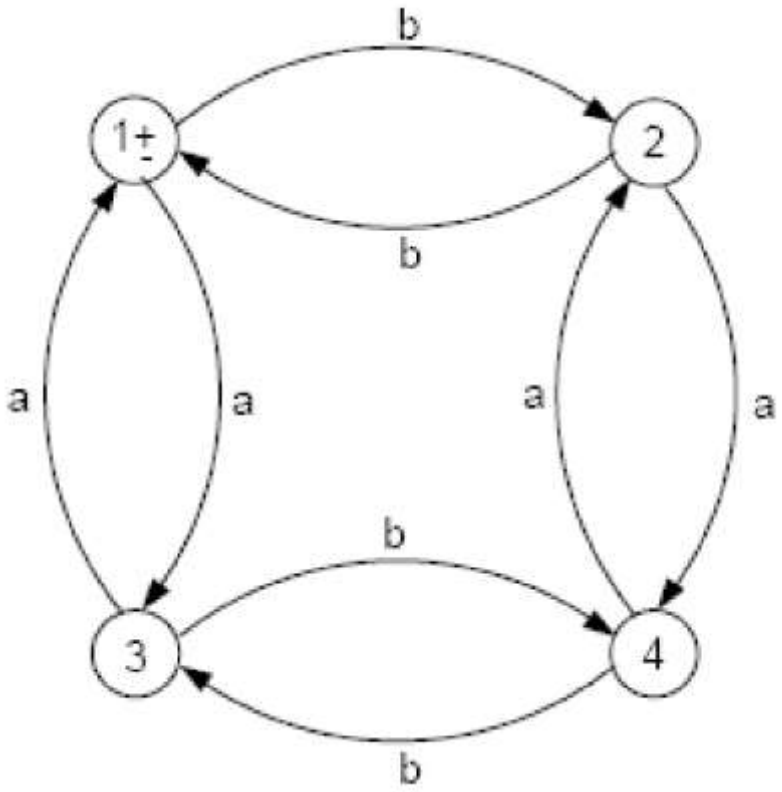
Example Contd.

- What happens to an input string that begins with a b?
- Such an input string will lead us to state 2. It then needs two more b's to get to the final state 3. These b's can be separated by any number of a's. Once in state 3, it needs no more b's, or 3 more b's, or 6 more b's and so on.
- All in all, an input string, whether starting with an **a** or a **b**, must have a total number of b's divisible by 3 to be accepted.
- The language accepted by this machine therefore can be defined by the regular expression

$$(a + ba^*ba^*b)^+ = (a + ba^*ba^*b)(a + ba^*ba^*b)^*$$

Example *EVEN-EVEN* revisited

- Consider the FA below.



Example *EVEN-EVEN* revisited Contd.

- There are 4 edges labeled a. All the a-edges go either from one of the upper two states (states 1 and 2) to one of the lower two states (states 3 and 4), or else form one of the lower two states to one of the upper two states.
- Thus, if we are north and we read an a, we go south. If we are south and we read an a, we go north.

- If a string gets accepted by this FA, we can say that the string must have had an even number of a's in it. Every a that took us south was balanced by some a that took us back north.
- So, every word in the language of this FA has an even number of a's in it. Also, we can say that every input string with an even number of a will finish its path in the north (ie., state 1 or state 2).

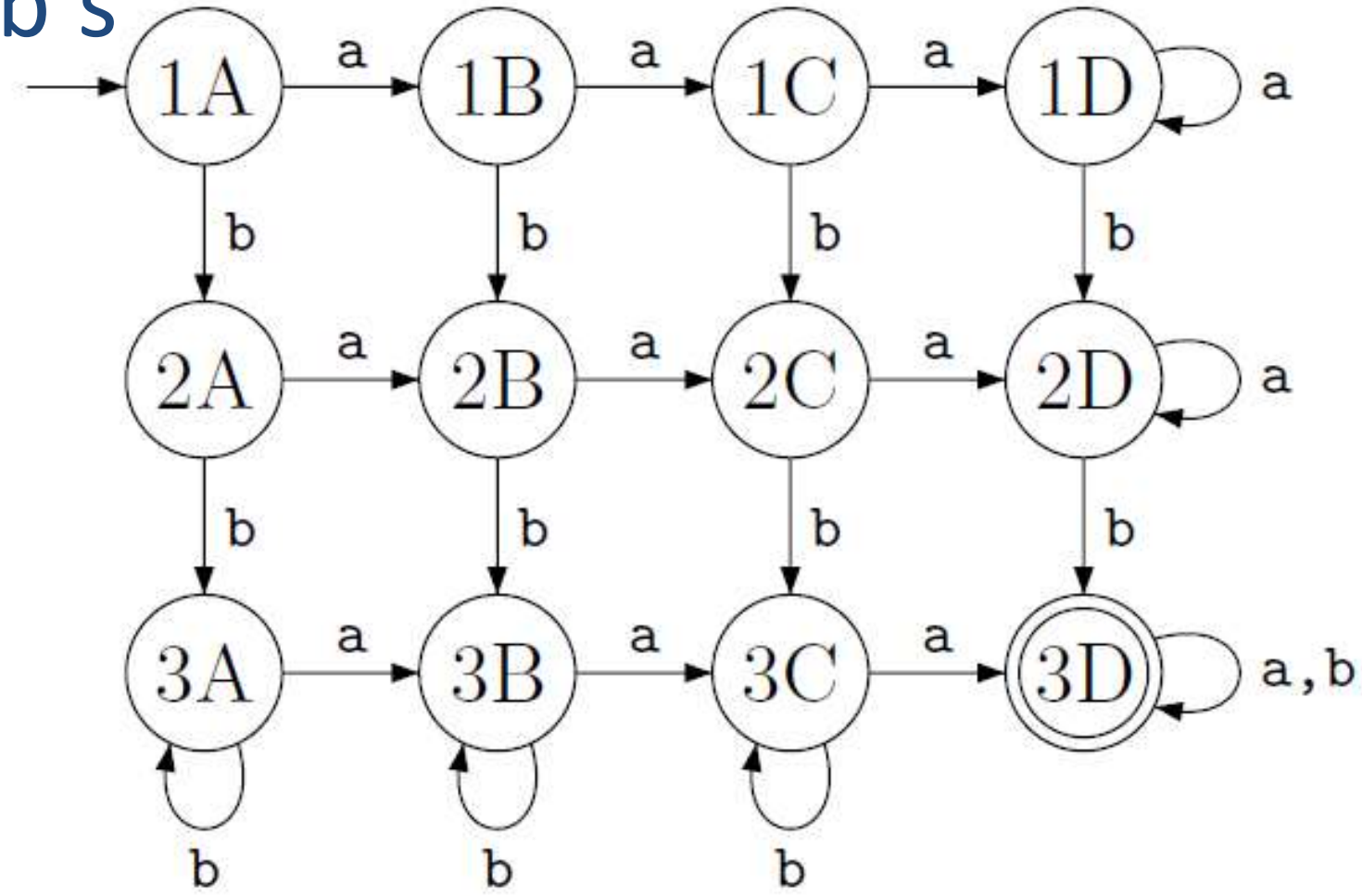
Example *EVEN-EVEN* revisited Contd.

- Therefore, all the words in the language accepted by this FA must have an even number of a's and an even number of b's. So, they are in the language *EVEN-EVEN.*

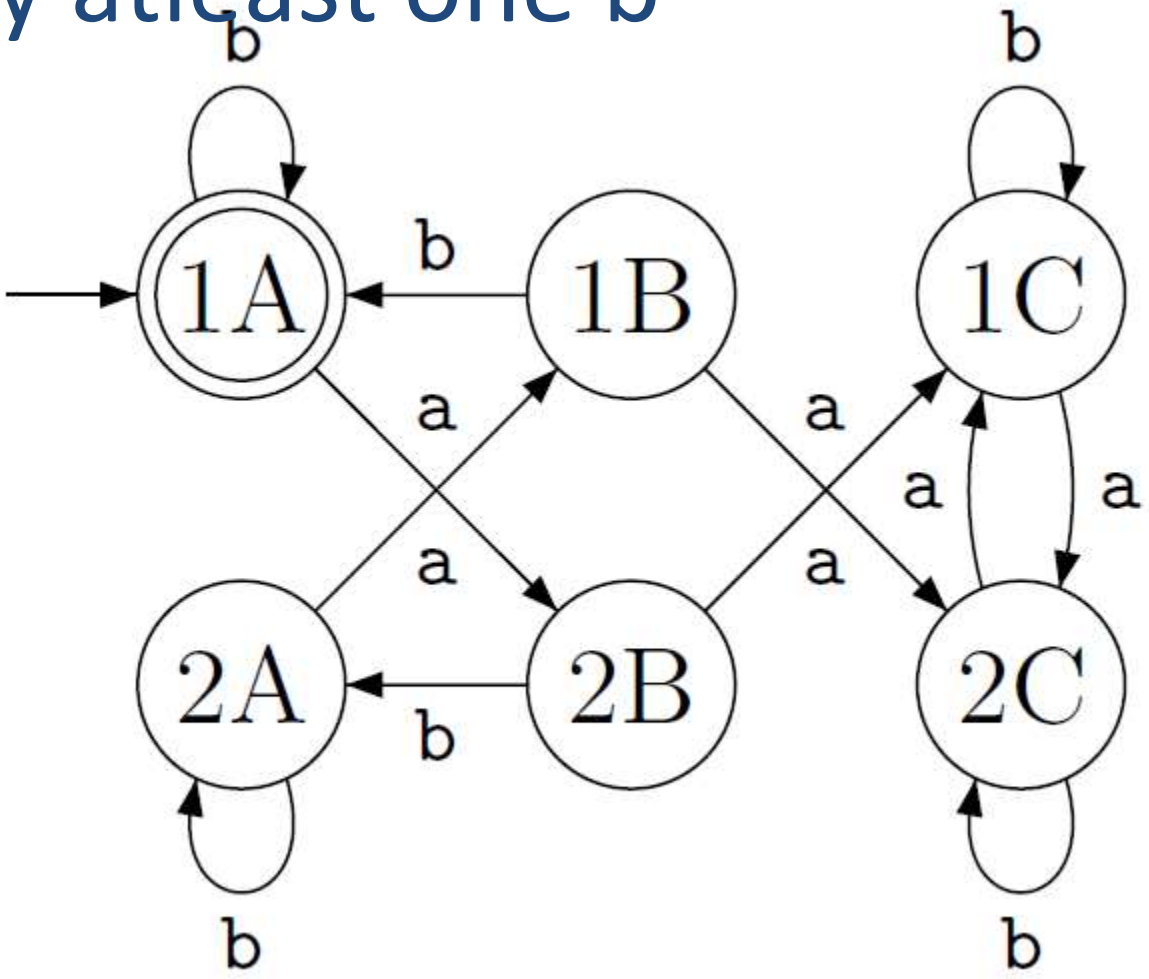
- Notice that all input strings that end in state 2 have an even number of a's but an odd number of b's. All strings that end in state 3 have an even number of b's but an odd number of a's. All strings that end in state 4 have an odd number of a's and an odd number of b's. Thus, every word in the language *EVEN - EVEN* must end in state 1 and therefore be accepted.

- Hence, the language accepted by this FA is *EVEN-EVEN.*

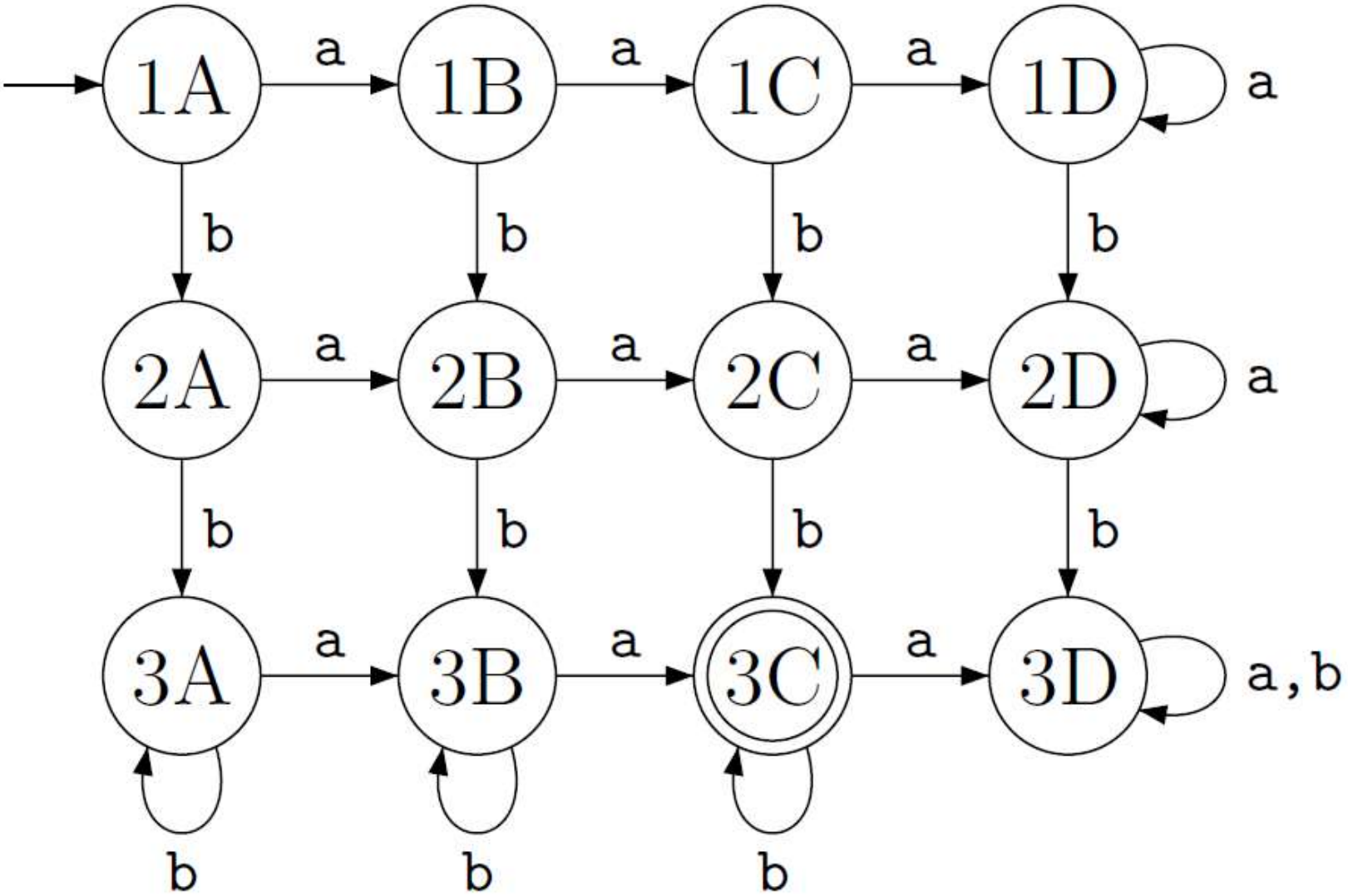
FA – at least three a's and at least two b's



FA – even number of a's and each a is followed by atleast one b



FA – exactly two a's and at least two b's



DFA

- DFA is the FA we have covered so far.