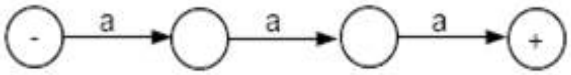
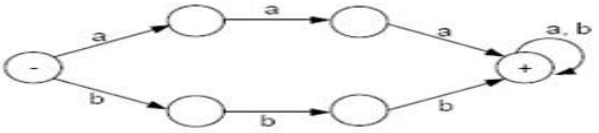
Finite Automata Examples:

Example:

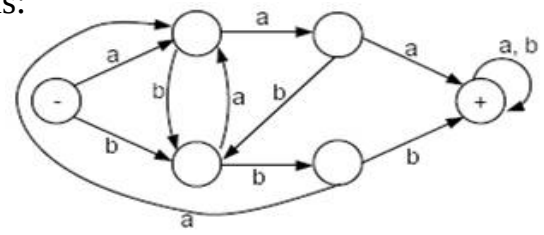
- Let's build a machine that accepts all words **containing a triple letter**, either *aaa* or *bbb*, and only those words.
- From the start state, the FA must have a path of three edges, with no loop, to accept the word *aaa*. So, we begin our FA with the following:



- 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:

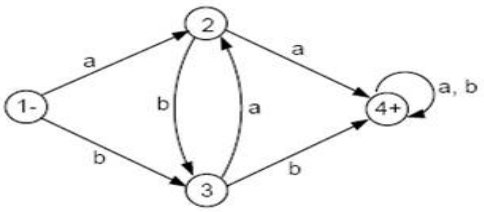


- If we are moving along the a-path and we read a *b* before the third a, we need to jump to the *b*-path in progress and vice versa. The final
- FA then looks like this:



Example:

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



Example:

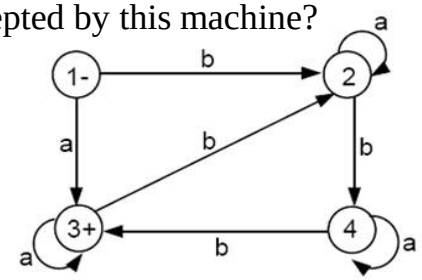
- There are only two ways to get to the final state 4 in this FA: One is from state 2 and the other is from state 3.
- The only way to get to state 2 is by reading an a while in either state 1 or state 3. If we read another a we will go to the final state 4.
- Similarly, to get to state 3, we need to 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?

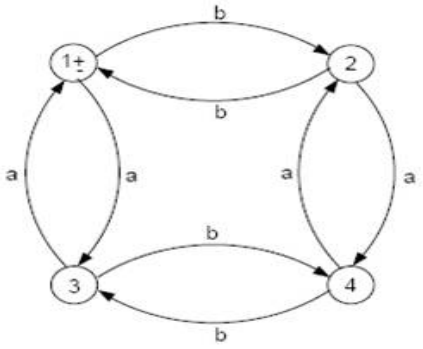


Example:

- 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.
- 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.



- 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).
- 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 numbers 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*.