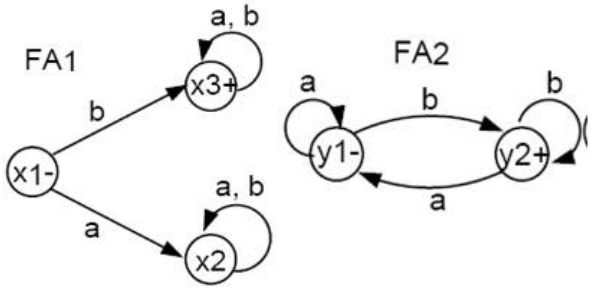
Algorithm:

1. First, create a state *z* for every state of FA_1 that we may go through before arriving at a final state.
2. For each final state x_{final} of FA_1, add a state $z = x_{final}$ or y_1, where y_1 is the start state of FA_2.
3. From the states added in step 2, add states

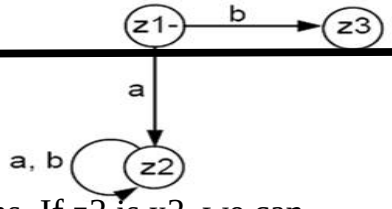
$$z = \begin{cases} x_{something} \text{ indicating that we are still continuing on } FA_1 \\ \text{OR} \\ \text{a set of } y_{something} \text{ indicating that we are on } FA_2 \end{cases}$$

4. Label every state *z* that contains a final state from FA_2 as a final state.



Example:

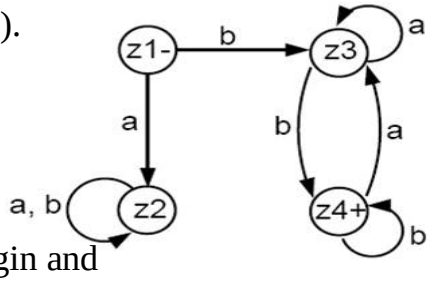
- FA_1 accepts all words that start with a *b*.
- FA_2 accepts all words that end with a *b*.
- We will use the above algorithm to construct FA_3 that accepts the product of the languages of FA_1 and FA_2, respectively.
  That is, FA_3 will accept all words that both start and end with the letter *b*.
- Initially, we must begin with $x_1 = z_1$.
- In z_1, if we read an *a*, we go to $x_2 = z_2$.
- In z_1, if we read a *b*, we go to x_3, a final state, which gives us the option to jump to y_1. Hence, we label $z_3 = x_3$ or y_1.
- From z_2 just like x_2, both an *a* or a *b* take us back to z_2, i.e., we have a loop here.



- In z3, if we read an a then the following happens. If z3 is x3, we can stay in x3 or jump to y1 (because x3 is a final state). If z3 is y1, we would loop back to y1. In any of the events, we end up at either x3 or y1, which is still z3. Hence, we have an a-loop at z3.

- In z3, if we read a b, then a different event takes place. If z3 is x3 we either stay in x3 or jump to y1. If z3 is y1, then we go to y2, a final state. Hence, we need a new final state z4 = x3 or y1 or y2.
- In z4, if we read an a, what happens? If z4 is x3 then we go back to x3 or jump to y1. If z4 is y1 then we loop back to y1. If z4 is y2, we go to y1. Thus, from z4, an a takes us to x3 or y1, which is z3.
- In z4, if we read a b, what happens? If z4 is x3, we go back to x3 or jump to y1.

  If z4 is y1, we go to y2, a final state. If z4 is y2, we loop back to y2, a final state. Hence, from z4 a b takes us to x3 or y1 or y2, which is still z4 (i.e., we have a b- loop here).



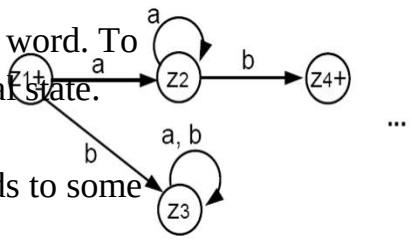
- This machine accepts all words that both begin and end with the letter b, which is what the product of the two languages (defined by FA_1 and FA_2 respectively) would be.
- If you multiply the two languages in opposite order (i.e. first FA_2 then FA_1), then the product language will be different. What is that language? Can you build a machine for that product language

Rule 4:

- **If *r* is a regular expression and FA_1 is a finite automaton that accepts exactly the language defined by *r*, then there is an FA, called FA_2, that will accepts exactly the language defined by *r*.**

Proof of Rule 4:

- We shall show that such an FA_2 exists by showing how to construct it from FA_1.
- The language defined by *r* must always contain the null word. To accept, we must indicate that the start state is also a final state. This must be done carefully.

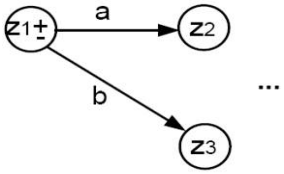


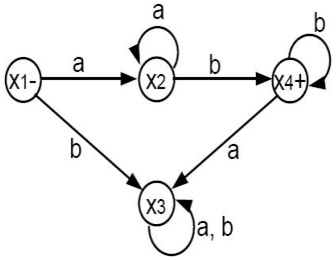
- The general rule is that each z-state (of FA_2) corresponds to some collection of x-states (of FA_1). We must remember each time we reach a final state; it is possible that we have to start over again at x_1.
- The transmissions from one collection of x-states to another based on reading certain input letters is determined by the transition rules for FA_1.
- There are only finitely many possible collections of x-states, so the machine FA_2 has only finitely many states.

Example:

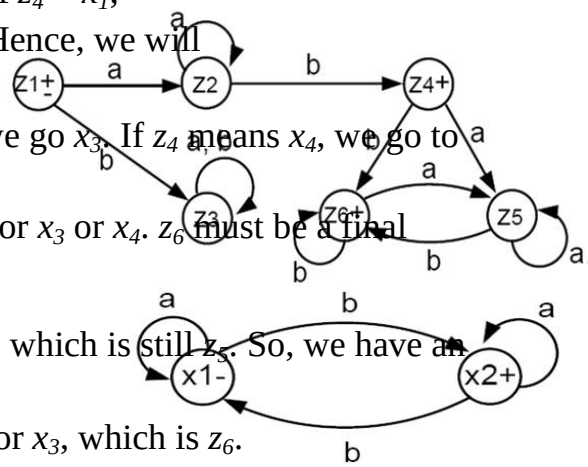
- Consider the regular expression r = aa*bb*.
- This defines the language where all the a's come before all the b's.
- The FA that accepts this language is:
- Let us now build FA_2 that accepts r* = (aa*bb*)*.
- We begin with the start state $z_1 = x_1$.
- In z_1, reading an a takes us to $x_2 = z_2$. Reading a *b* takes us to $x_3 = z_3$.

- In z_3, just like x_3, reading either an *a* or a *b*, we loop back to z_3.





• In z_2, if we read an a we go back to z_2. If we read a b, we go to z_4, or we have the option of jumping to the start state x_1 (since x_4 is a final state). Hence, let z_4 += x_4 or x_1.



• In z_4, what happens if we read an a? If $z_4 = x_1$, we go to x_2. If $z_4 = x_4$, we go to x_3. Hence, we will be in x_2 or x_3. So, let $z_5 = x_2$ or x_3.

• In z_4, if we read a b? If z_4 means x_1, we go x_3. If z_4 means x_4, we go to x_4 or jump to x_1 (due to final x_4). Thus, let $z_6 = x_1$ or x_3 or x_4. z_6 must be a final state since x_4 is.

• In z_5, reading an a takes us to x_2 or x_3, which is still z_5. So, we have an a-loop at z_5.

• In z_5, reading a b takes us to x_4 or x_1, or x_3, which is z_6.

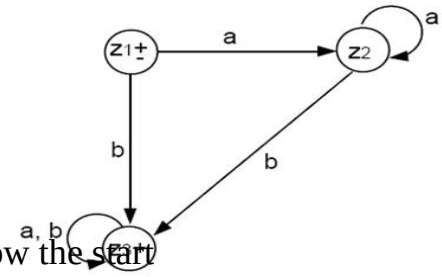
• In z_6, reading an a, take us to x_2 or x_3, which is z_5.

• In z_6, reading a b takes us to x_3, x_4, or x_1, which is still z_6. So, we have a b-loop at z_6.

• The final picture is as follows:

Example: Special Case

• Consider the FA below that accepts the language of all words having an odd number of b's.

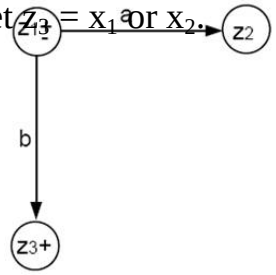


• Notice that this machine does not accept Λ and does allow the start state x_1 to be re-entered.

• In this case we need to represent x_1 as two separate z-states in FA_2, one as a start and final state ±, and the other as the non-final start state.

• The ± state is necessary for FA_2 to accept . The non-final start state is necessary for FA_2 to operate correctly, since some strings that return to the start state x_1 may not be valid words and therefore should not be accepted.

- Thus, we represent x1 (-) as two separate states:
  - z1 ± = x1 and a final state.
  - z2 = x1 and a non-final state.
- So, in z_1 reading an a, we go back to x_1 as a non-final state, which we labeled as z_2.
- In z_1, reading a b takes us to x_2, a final state, with the option of jumping back to the non-final start state x_1. So, let $z_3 = x_1$ or x_2.



- Note that z_3 must be a final state since x_2 is.
- In z2, if we read an a, we stay in z2. If we read a b, we go to x2 or back in x1 (as x2 is a final state); that is we go to z3.
- In z3, reading an a takes us to x1 or x2, which is z3 itself. In z3, reading a b takes us to x2 or x1, which is also z3. Hence, there is a a; b-loop at z3.
- The complete machine is
- We have finished the proof of part 3 of Kleene's theorem.
- Because of Rules 1, 2, 3, and 4, we know that all regular expressions have corresponding finite automata that define the same language.
- This is because while we are constructing the regular expression from elementary building blocks using recursive definition, we can simultaneously be constructing the corresponding FA using the algorithms discussed above.

Definition: A nondeterministic finite automaton (or NFA) is a TG with a unique start state and with the property that each of its edge labels is a single alphabet letter.

- The regular deterministic finite automata are referred to as DFAs, to distinguish them from NFAs.
- As a TG, an NFA can have arbitrarily many a-edges and arbitrarily many b-edges coming out of each state.
- An input string is accepted by an NFA if there exists any possible path from - to +.

Examples of NFAs: