Theorem 7:

For every NFA, there is some FA that accepts exactly the same language. Proof 1:

- By the proof of part 2 of Kleene's theorem, we can convert an NFA into a regular expression, since an NFA is a TG.
- By the proof of part 3 of Kleene's theorem, we can construct an FA that accepts the same language as the regular expression. Hence, for every
- NFA, there is a corresponding FA.

Notes:

- Theorem 7 means that all NFAs can be converted into FAs.
- Clearly, all FAs can be considered as NFAs that do not make use of the option of extra freedom of edge production.
- Hence, as language acceptors, NFA = FA.

Proof 2 of Theorem 7:

- We present the following constructive algorithm showing how to build a FA for a given NFA:

Algorithm:

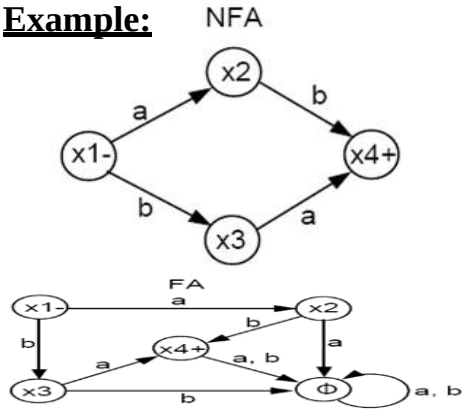
- Each state in FA is a collection of states from the original NFA, e.g. x_i, or x_j , or x_k, ...
- For every state z in the FA, the new state that an a-edge (or a b-edge) will take us to is just the collection of possible states that result from being in x_i and taking the *a-edge*, or being in x_j and taking the *a-edge* and so on.
- The start state of the FA that we are constructing is the same old start state we had to begin with in the NFA. Its a-edge (or b-edge) goes to the collection of the

x-states that can be reached by an a-edge (or a b-edge) from the start state in the NFA.

- Since there may be no a-edges (or no b-edges) leaving the start state in the NFA (or leaving any particular state), we must add a state in the FA for the a-edge (or the b-edge) to reach in this case.
- The Φ state in the FA must have an a; b-loop going back to itself.

- A state in the FA is a final state if the collection of the x-states that it represents has an old final state in it.

Example:



NFAs and Kleene's theorem:

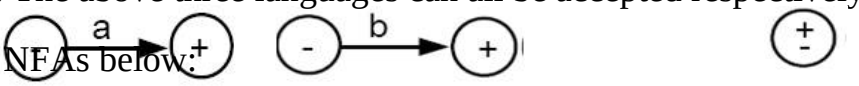
- Rules 1, 2, 3, and 4 in Part 3 of Kleene's theorem can be proved differently, using NFAs.
- We will show here how to prove Rule 1 and Rule 2. Proofs for Rule 3 and 4 using NFAs are considered in the problems section.

Proof 2 of Rule 1, Part 3, Theorem 6:

- Rule 1 states that there are FAs for the languages {a}, {b}, and {Λ}.

Proof:

- Step 1: The above three languages can all be accepted respectively by the NFAs below:



- Step 2: By Theorem 7, for every NFA, there is an equivalent FA. Hence, there must be FAs for these three languages as well.

Proof 2 of Rule 2, Part 3, Theorem 6:

- Given FA_1 and FA_2, we shall present an algorithm for constructing a union machine $FA_1 + FA_2$.

Algorithm:

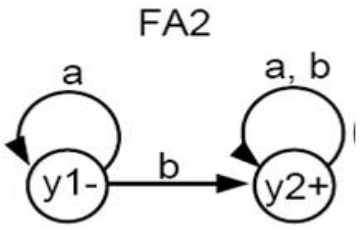
Step 1: Introduce a new and unique start state with two outgoing a-edges and two outgoing b-edges but no incoming edges. Connect them to the states that the start states of FA_1 and FA_2 are connected to. Do not erase

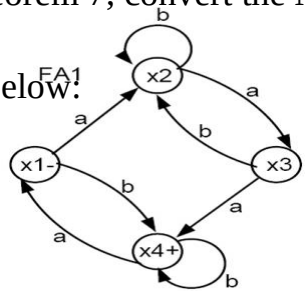
the start states of FA_1 and FA_2, but erase their - signs, leaving all their edges intact. The new machine is an NFA that clearly accepts exactly language(FA_1) + language(FA_2).

Step 2: Using the algorithm of Theorem 7, convert the NFA into an FA.

Example:

- Consider the FA_1 and FA_2 below:





- Using the above algorithm (Step 1) we produce the following NFA.