CSE331 Automata and Computability

Lecture 9.1: Conversion of NFA/ε-NFA to DFA

Presenter: Sabrina Zaman Ishita (SZI) *Scribe: Sabrina Zaman Ishita (SZI)*

Types of Finite Automata

There are two types of Finite Automata:

- **Nondeterministic Finite Automata (NFA):** can remain in more than one state at a time. One variation of NFA is ε-NFA where there is ε-transitions. An NFA
  - can have more than one transition from a state with same input symbol.
  - can have ε-transitions.
- **Deterministic Finite Automata (DFA):** can remain only in a single state at a time. A DFA
  - cannot have more than one transition from a state with same input symbol.
  - cannot have ε-transitions.

Conversion of NFA/ε-NFA to DFA

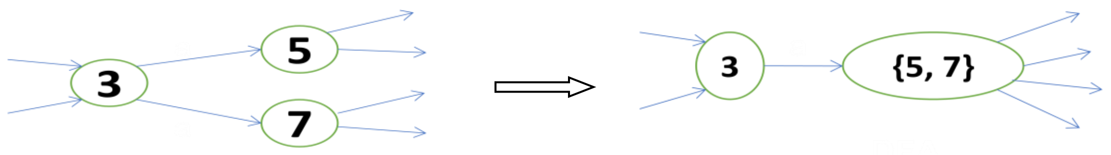
If **L** is a regular language, we can express **L** by a nondeterministic finite automata NFA or ε-NFA **N**, we can also represent it with the help of a deterministic finite automata DFA **D**. Now the language of this NFA **N** is **L(N)** and the language of this DFA **D** is **L(D)**. Both the finite automata are created from same language, hence

$$L(N) = L(D) = L$$

So, for the NFA **N** and the DFA **D**, there must be a mechanism to convert the **N** to **D**.

If there's an NFA over input symbol {a, b} and a string "aba" is given to be recognized by the NFA, we'll start from the start state, we'll look for 'a' transition, there can be more than one 'a' transition and also ε-transitions. From the states where these transitions go to, we'll look for 'b' transition, then finally 'a' transitions will be checked. If we end up in one of the final states, the the string "aba" is accepted by this NFA. This seems very lengthy. But if we apply the same thing on DFA to recognize the string "aba", we'll start from start state, look for 'a'-transition, then look for 'b'-transition and the 'a' again. This is pretty straightforward because in DFA, from a state there can be only one transition with an input symbol and also there's no ε-transition. So, **DFA is a faster recognizer**!

- **Approach:** Each state in the DFA will correspond to a set of states of NFA. This approach is called **subset-construction.**



- **Worst-case:** If there are 3 states in an NFA- A, B and C. If set of states are to be considered, 7 different sub sets can be created (ABC, AB, BC, AC, A, B, C), thus creating 7 different possible states in DFA. Here, for, number of states = 3, number of possible of states in DFA is 7≈2^3. Now, if the number of states = 100, ≈2^100 different states will be created which is a huge number. Why would we like to go from 3 states to 7 states or 100 states to 2^100? The above worst case scenario is very rare. In most of the cases, number of states decreases and makes DFA a faster recognizer.
- **Input:** A NFA

  S= States = { s_0, s_1,…, s_N} = S_{NFA}

  δ = Move function = $Move_{NFA}$

  Move(S, a) → Set of states
- **Output:** A DFA

  S = States = { ?, ?, …, ?} = S_{DFA}

  δ = Move function = $Move_{DFA}$

  Move(s, a) → Single state from S_{DFA}
- **ε-Closure:** ε-Closure(s) is the set of states reachable from s on ε-transitions.

  ε-closure (2) = { 2, 1, 5, 6 }

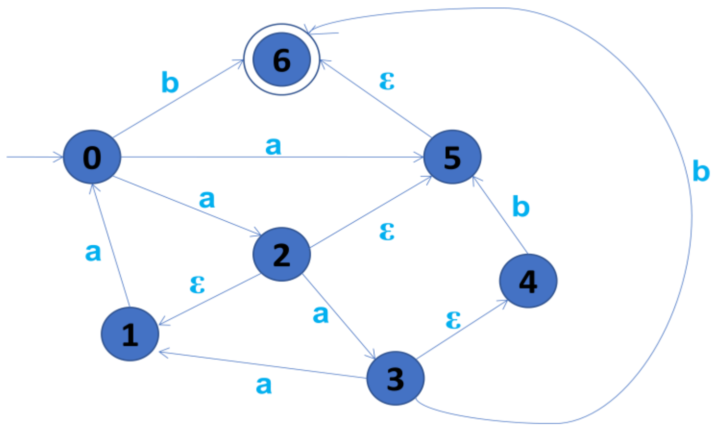


Figure 1: An ε-NFA

ε-Closure (S) = { t | t ∈ ε-closure(s) for all s ∈ S }

For example, from Figure 1,

ε-closure ({2, 3}) = { 1, 2, 3, 4, 5, 6 }

- **Move Functions:**
  - Deterministic Machine:

    Move(s, ch) → t
  - Non-deterministic Machine:

    $\mathbf{Move_{NFA}}$(S, ch) → T

    If s ∈ S and there is an edge t, then t ∈ T

For example, from Figure 1,
$\text{Move}_{\text{NFA}}(0,a) = \{2,5\}$
$\text{Move}_{\text{NFA}}(\{0,3\},a) = \{1,2,5\}$

All the pre-requisite concepts and idea to convert an NFA to DFA have been described above. It's time to convert and NFA/ε-NFA to DFA. Move on to next note.