CSE331 Automata and Computability

Lecture 7: Nondeterministic Finite Automata (NFA)

Presenter: Warida Rashid *Scribe: Warida Rashid*

Nondeterministic Finite Automata (NFA)

Nondeterministic Finite Automata is a Finite State Machine that accepts or rejects a string according to the pattern it defines. Like a DFA, an NFA also consists of states and transitions. However, it has a lot less restrictions than a DFA. An NFA can have more than one transition on one input symbol. It can be in more than one state at any given point. In each step, whenever there are two or more transitions, it "copies" itself and each copy follows a transition. If there is no possible transition, the copy “dies”.

Formal Definition

An NFA is represented formally by a 5-tuple, $(Q, \Sigma, \delta, q_0, F)$, consisting of:

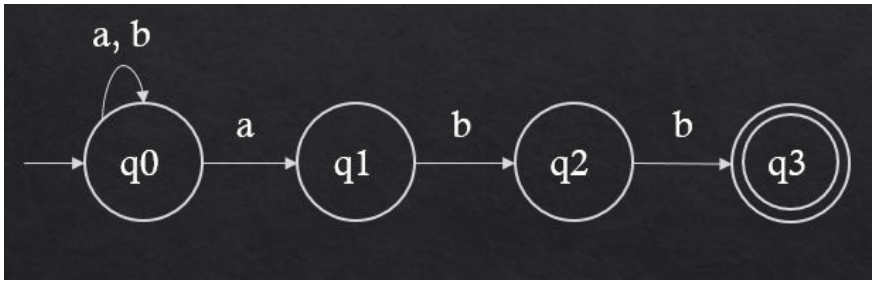
- a finite set of states, Q.
- a finite set of input symbols, Σ .
- a transition function δ where δ: $Q \times \Sigma \rightarrow P(Q)$ where P(Q) is the power set of Q.
- an initial (or start) state $q_0 \in Q$
- a set of states $F \subseteq Q$

Acceptance of a String by an NFA

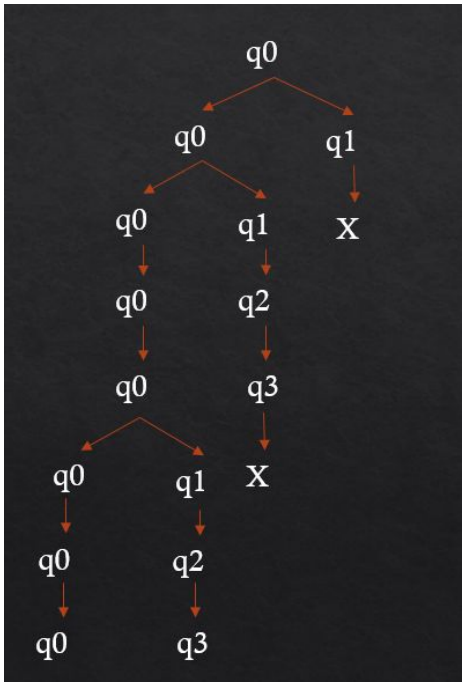
An NFA accepts the input string if there exists some choice of transitions that leads to the NFA ending in an accepting state. Thus, one accepting branch is enough for the overall NFA to accept, but every branch must reject for the overall NFA to reject.

Example

- L = {w | w ∈ (a, b)* and w ends with the substring abb}



Let's take a string *aabbabb* and simulate the NFA for the string. The states that it goes through can be demonstrated as a tree structure::



The last states the NFA is in after the processing of the string is done are q0 and q3.

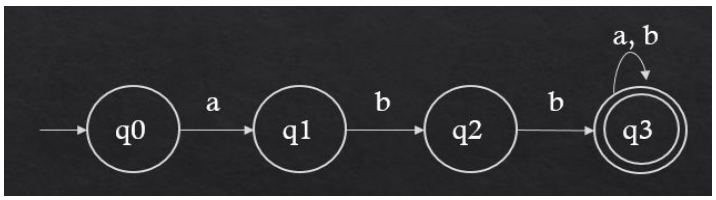
Therefore, the extended transition function δ(q0, *aabbabb*) = {q0, q3}. Since it contains the final state q3, the string *aabbabb* is accepted in the language.

The transition table for this NFA:

	a	**b**
q0	{q0, q1}	{q0}
q1	***Φ***	{q2}
q2	***Φ***	{q3}
q3	***Φ***	***Φ***

Here, the symbol, ***Φ***, denotes an empty i.e. there are not valid transitions.

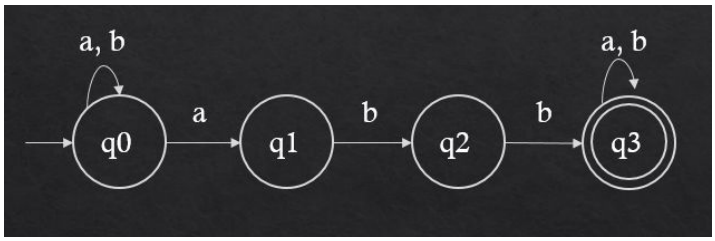
- L = {w | w ∈ (a, b)* and w starts with the substring abb}



The transition table:

	a	**b**
q0	{q1}	***Φ***
q1	***Φ***	{q2}
q2	***Φ***	{q3}
q3	{q3}	{q3}

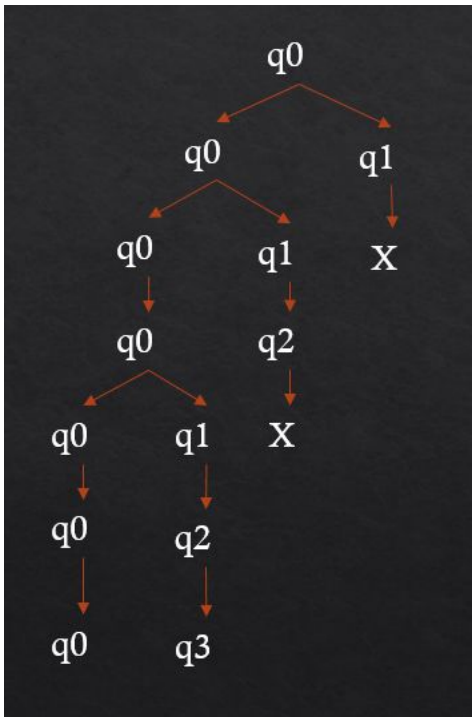
- L = {w | w ∈ (a, b)* and w contains the substring abb}



The transition table:

	a	**b**
q0	{q0, q1}	{q0}
q1	***Φ***	{q2}
q2	***Φ***	{q3}
q3	{q3}	{q3}

Processing the string *aabbabb* can be shown by the following diagram:



CSE331 Automata and Computability

Lecture 8.1: NFA with ε Transitions

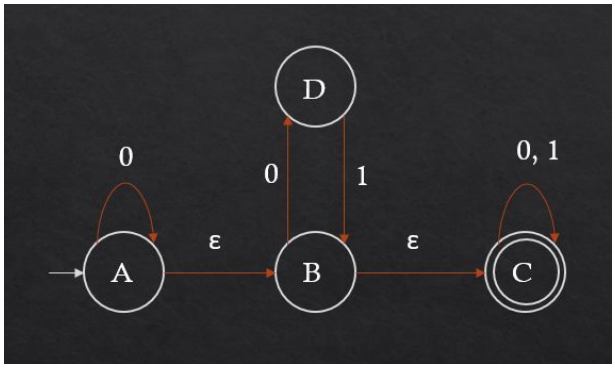
Presenter: Warida Rashid *Scribe: Warida Rashid*

Epsilon NFA (*ε*-NFA)

The class of NFAs can be extended by allowing spontaneous transitions without reading any input symbols. In the diagram, such transitions are represented by an edge labeled by epsilon (***ε***). It is important to note that, ***ε*** is not considered as an input symbol. In fact, it is assumed that ***ε*** does not belong to any input alphabet. An NFA with **epsilon** transition(s) is called ***ε*-NFA.** Both NFAs and ***ε*-NFA**s recognize the same class of languages that are defined as **Regular Languages**. However, allowing transitions on epsilon adds certain programming conveniences.

Example Simulation

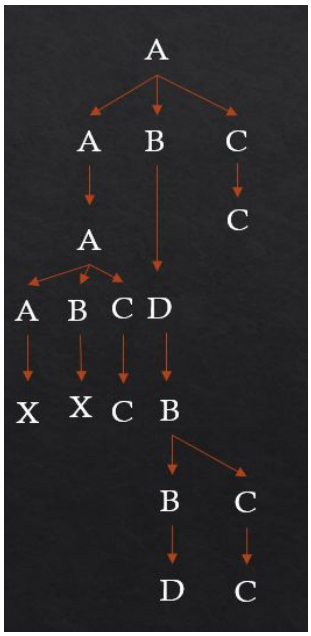
Consider the following ***ε*-NFA**.



When the automaton is at state A, it "copies" itself. One of the copies remain at A and the other one goes to B following the **epsilon(*ε*)** transition from A to B. Again, when the automaton is at state B, it "copies" itself. One of the copies remain at B and the other one goes to C following the **epsilon(*ε*)** transition from B to C. Basically, When the automaton is at state A, after completing the epsilon transitions, the states that the automaton is at are: A, B, C. This is called the epsilon**(*ε*)** closure. ***ε*-Closure** of a state q is the set of states that can be reached from q by

following the epsilon transitions and it includes the state q itself. According to the definition, **ε-Closure** of A is the set {A, B, C} and the **ε-Closure** of B is the set {B, C}.

If we process the string *010* can be represented by the following diagram:



Language of an NFA/ε-NFA

The language of an NFA/ε-NFA is the set of strings that it accepts. A string w is accepted by an NFA if δ(q0, w) contains a final/accepting state.

Comparison between DFA and NFA/ε-NFA

- A **DFA** cannot be in more than one state at any given point but an **NFA/ε-NFA** can.
- **DFA** does not allow moving to two or more different states, from the same state, on the same input symbol but an **NFA/ε-NFA** does.
- In **DFA**, the transition function is
  $\boldsymbol{\delta}: \boldsymbol{Q} \times \sum \rightarrow \boldsymbol{Q}$

  In **NFA/ε-NFA**, the transition function is
  $\boldsymbol{\delta}: \boldsymbol{Q} \times \sum \rightarrow \mathbf{P}(\boldsymbol{Q})$

  Here, $\mathbf{P}(\boldsymbol{Q})$ is the power set of Q.
- Determinism does not allow epsilon transitions but nondeterminism does.

CSE331 Automata and Computability

Lecture 8.2: RE to NFA (Thompson's Construction)

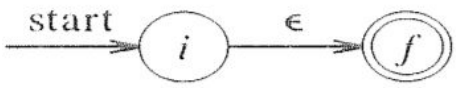
Presenter: Warida Rashid *Scribe: Warida Rashid*

Thompson's construction algorithm, also called the McNaughton-Yamada-Thompson algorithm, is a method of transforming a regular expression into an equivalent nondeterministic finite automaton (NFA). Regular Expressions define the pattern for a language and Finite Automaton is suited for implementing the pattern in terms of programming. Therefore, we often have to convert a Regular Expression into a Finite Automaton.

Thompson's Construction Algorithm

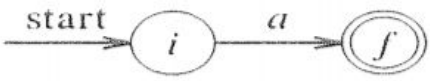
The algorithm works recursively by splitting an expression into its constituent subexpressions. The rules for constructing an NFA consist of some basis for handling the subexpressions with no operators, and inductive rules for constructing larger NFA's from the NFA's for the immediate subexpressions of a given expression.

Basis: For the Regular Expression R = ε, the NFA is:



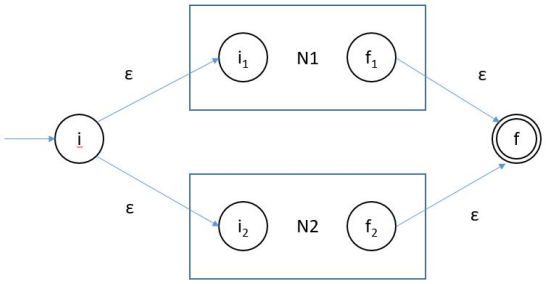
Here, i is the start state and f is the final state.

For a subexpression that is a symbol in the alphabet such as R = a, a ∈ Σ, the NFA is:

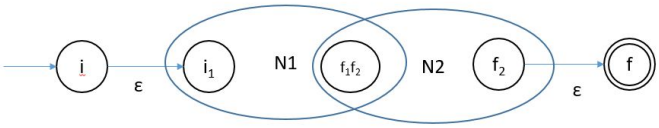


Induction: Suppose N1 and N2 are the NFAs for the Regular Expressions R1 and R2 respectively.

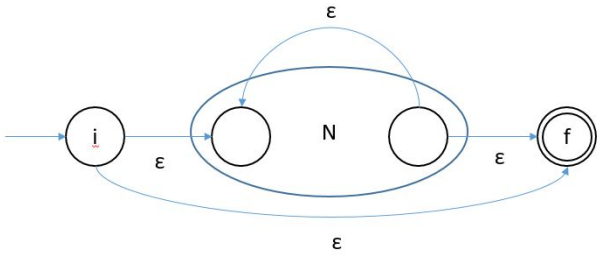
1. The NFA for the RE **R1|R2** is



2. The NFA for the RE **R1.R2** is



3. If R is a Regular Expression and N is the NFA for R, then the NFA for the **R*** is



Conditions

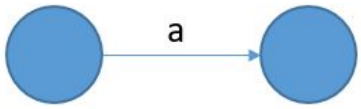
The NFA derived from Thompson's Construction must follow some conditions:

1. There can be only one start state.
2. The start state cannot be accessible from any states i.e. it cannot have transitions to it from other states.
3. There can be only one final state. The final state cannot have any transition from it to other states.
4. The number of transitions leaving any state is at most two.

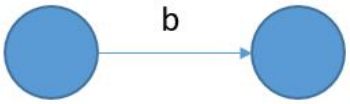
Example:

Consider the Regular Expression **(a|b)*abb.** The steps following the Thompson's Construction algorithm are:

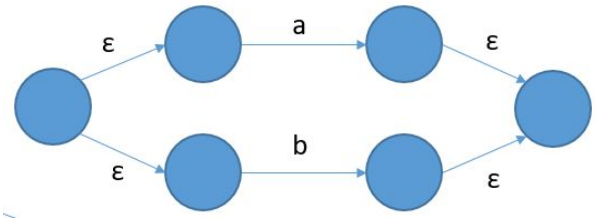
1. Subexpression: a



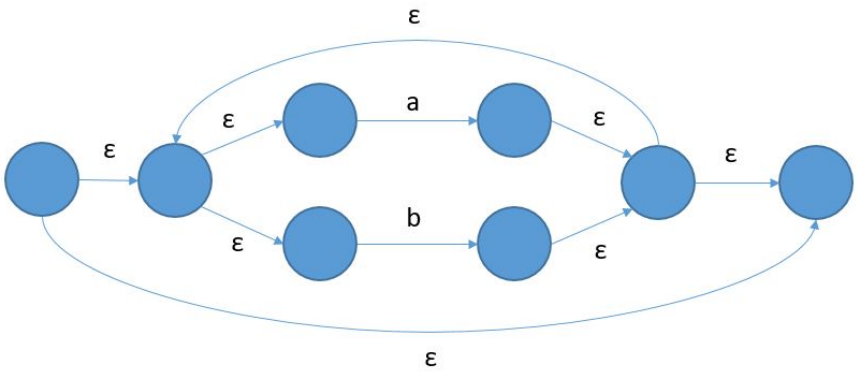
2. Subexpression: b



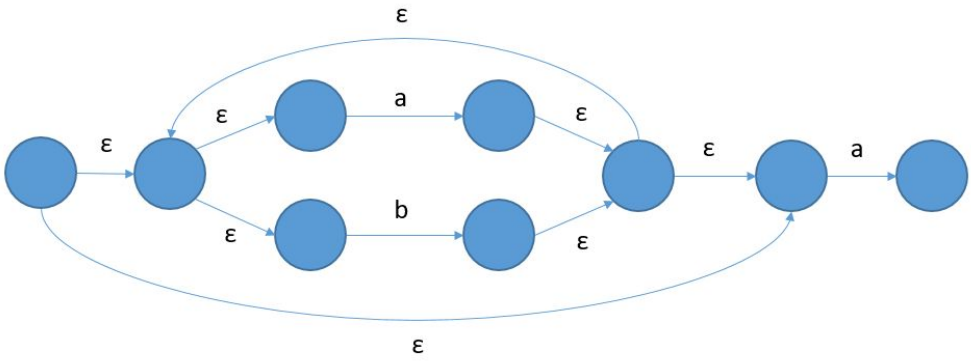
3. Subexpression: a | b



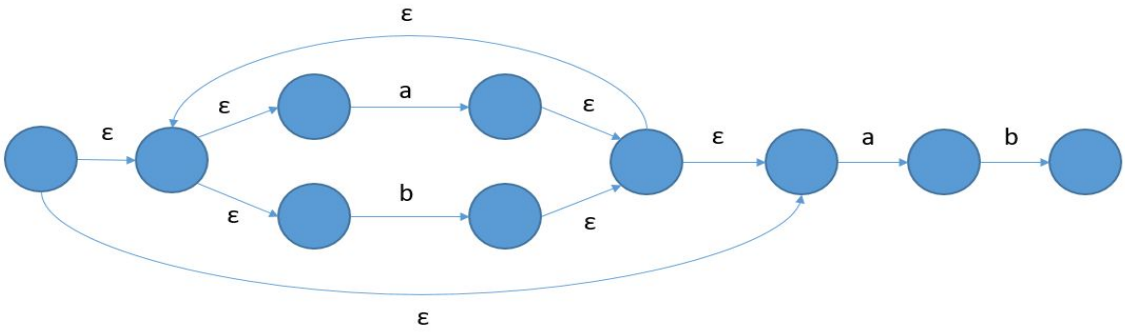
4. Subexpression: (a | b)*



5. Subexpression: (a | b)*a



6. Subexpression: (a | b)*ab



7. The NFA for the entire expression: (a | b)*abb