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

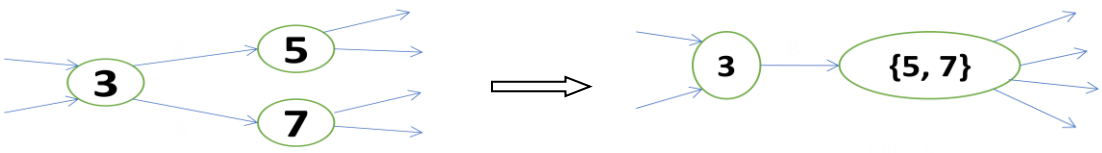
If **L** is a regular language, we can express **L** by a nondeterministic finite automata NFA or ε-NFA **N**, we can also represent it with the help of a deterministic finite automata DFA **D**. Now the language of this NFA **N** is **L(N)** and the language of this DFA **D** is **L(D)**. Both the finite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

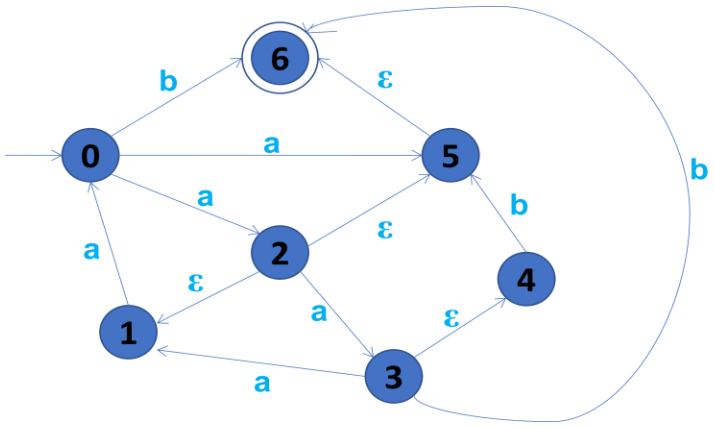


Figure 1: An ε-NFA

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:
    $Move_{NFA}$(S, ch) → T
    If s ∈ S and there is an edge t, then t ∈ T

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

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.

CSE331 **Automata and Computability**

Lecture 9.2: Conversion of NFA/ε-NFA to DFA

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

This is the continuation note on conversion of NFA/ε-NFA to DFA.

To convert an NFA to DFA, we need to use:

$Move_{NFA}(S,a)$ → the transition function from NFA

ε-Closure (s) → where s is a single state from NFA

ε-Closure (S) → where S is a set of states from NFA

By using these, we need to construct:

S_{DFA} → the set of states in the DFA

Initially, **S_{DFA}** ← { }

Add x to S_{DFA} where x is some set of NFA states

Example: "Add 

to **S_{DFA}**"

Here, 3, 5, and 7 –all are distinct states of NFA and they have been added to the DFA as a combined single state.

We'll "mark" some of the states in the S_{DFA}

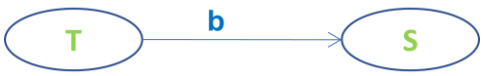
Marked = "We've done this one" (√)

Unmarked = "Still need to do this one"

$Move_{DFA}(T,b)$ → The transition function from DFA

To add an edge to the growing DFA

Set **$Move_{DFA}(T,b)$** to S (where S and T are sets of NFA states)



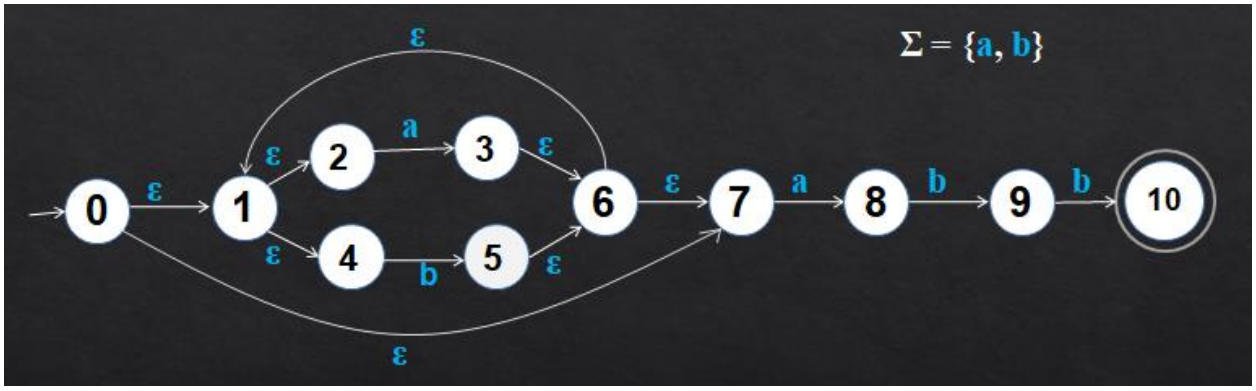
Where S and T are sets of NFA states which became single states in DFA.

The algorithm to convert an NFA to DFA is given below, after that an example workout is shown for your convenience.

Algorithm: NFA to DFA

```
S_DFA = {}
Add ε-Closure(s0) to S_DFA as the start state
Set the only state in S_DFA to "unmarked"
while S_DFA contains an unmarked state do
      Let T be that unmarked state
      Mark T
      for each a in Σ do
            S = ε-Closure(Move_NFA(T, a))
            if S is not in S_DFA already then
                  Add S to S_DFA (as an "unmarked" state)
            endIf
                  Set Move_DFA(T, a) to S
      endFor
endWhile
for each S in S_DFA do
      if any s ∈ S is a final state in the NFA then
            Mark S as a final state in the DFA
      endIf
endFor
```

Example:



Let's work on the start state of the DFA first. The start state of the NFA is 0. It should be included in the start state of the DFA as well. All the other states reachable from 0 should also be included here since from state 0 you can go to those states with zero cost. So, the start state of the DFA is basically,

- Start state:

  ε-Closure (0)

  = { 0, 1, 2, 4, 7 }

  = **A**

Let's work with **A** now, we need to define transition functions for **A** with each input symbol from Σ={a,b}. So, [**A** = { 0, 1, 2, 4, 7 }]

- $Move_{DFA}$ (**A**, a)

  = ε-Closure ($Move_{NFA}$ (**A**, a))

  = ε-Closure ({3, 8})

  = {1, 2, 3, 4, 6, 7, 8}

  = **B**

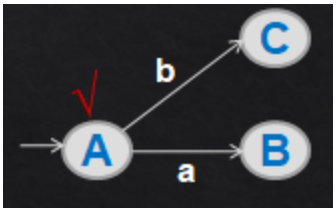
- $Move_{DFA}$ (**A**, b)

  = ε-Closure ($Move_{NFA}$ (**A**, b))

  = ε-Closure ({5})

  = {1, 2, 4, 5, 6, 7}

  = **C**



Let's work with **B** now,

B={1, 2, 3, 4, 6, 7, 8}

- $Move_{DFA}$ (**B**, a)

  = ε-Closure ($Move_{NFA}$ (**B**, a))

  = ε-Closure ({3, 8})

  = {1, 2, 3, 4, 6, 7, 8}

  = **B**

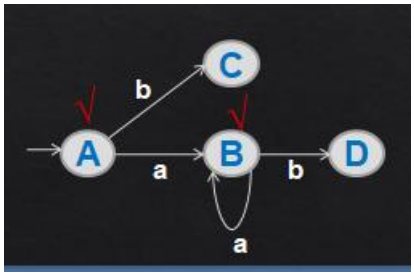
- $Move_{DFA}$ (**B**, b)

  = ε-Closure ($Move_{NFA}$ (B, b))

  = ε-Closure ({5, 9})

  = {1, 2, 4, 5, 6, 7, 9}

  = **D**



Let's work with **C** now,

C = {1, 2, 4, 5, 6, 7}

- $Move_{DFA}$ (**C**, a)

  = ε-Closure ($Move_{NFA}$ (**C**, a))

  = ε-Closure ({3, 8})

  = {1, 2, 3, 4, 6, 7, 8}

  = **B**

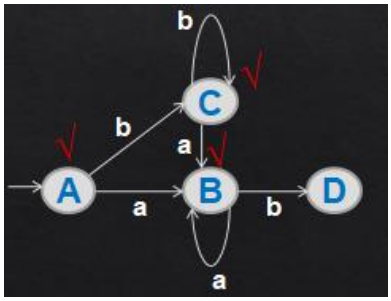
- $Move_{DFA}$ (**C**, b)

  = ε-Closure ($Move_{NFA}$ (**C**, b))

  = ε-Closure ({5})

  = {1, 2, 4, 5, 6, 7}

  = **C**



Let's work with **D** now,

D = {1, 2, 4, 5, 6, 7, 9}

- $Move_{DFA}$ (**D**, a)

  = ε-Closure ($Move_{NFA}$ (**D**, a))

  = ε-Closure ({3, 8})

  = {1, 2, 3, 4, 6, 7, 8}

  = **B**

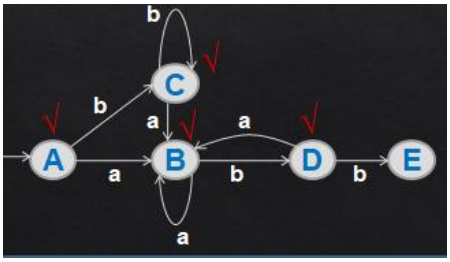
- $Move_{DFA}$ (**D**, b)

  = ε-Closure ($Move_{NFA}$ (**D**, b))

  = ε-Closure ({5, 10})

  = {1, 2, 4, 5, 6, 7, 10}

  = **E**



Let's work with **E** now, **E** = { 1, 2, 4, 5, 6, 7, 10}

- $Move_{DFA}$ (**E**, a)

  = ε-Closure ($Move_{NFA}$ (**E**, a))

  = ε-Closure ({3, 8})

  = {1, 2, 3, 4, 6, 7, 8}

  = **B**

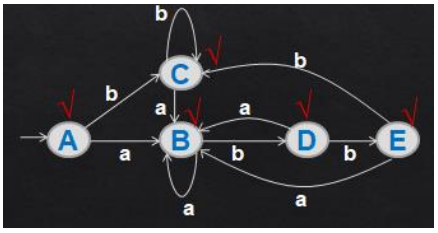
- $Move_{DFA}$ (**E**, b)

  = ε-Closure ($Move_{NFA}$ (**E**, b))

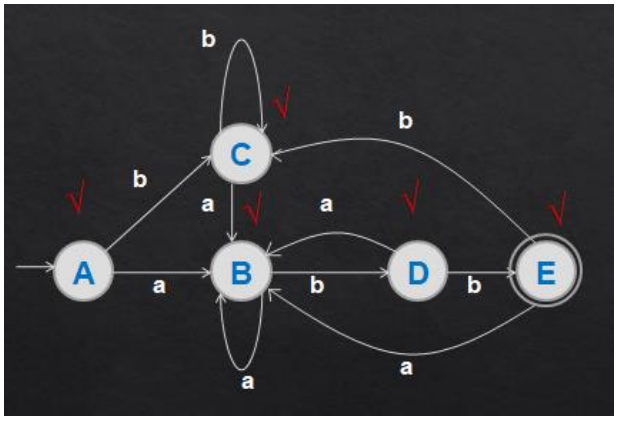
  = ε-Closure ({5})

  = {1, 2, 4, 5, 6, 7}

  = **C**



Since, **E** contains the final state of the NFA (state 10), the final state of the DFA should be **E**. The final DFA is given below:



CSE331 **Automata and Computability**

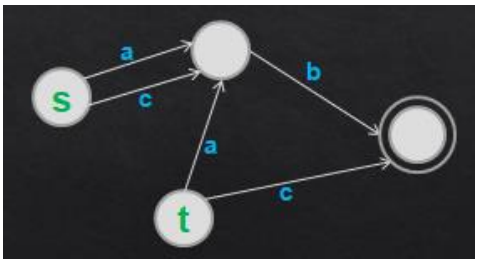
Lecture 10.1: Minimization of DFA

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

We can further minimize a DFA by decreasing the number of states. For this we are going to follow Hopcroft's Algorithm. For that, some concepts need to be explained first:

- **Sufficiently Different States**
  Merge states, if at all possible. Two states cannot be merged if they are "sufficiently different"
  State **s** is "distinguished" from state **t** by some string w iff:
  Starting at **s**, given characters in w, the DFA ends up accepting, but starting at **t**, the DFA does not accept. For example,

  "ab" does not distinguish **s** and **t**.
  But "c" distinguishes **s** and **t**.
  Therefore, **s** and **t** cannot be merged. Because they are sufficiently different states.

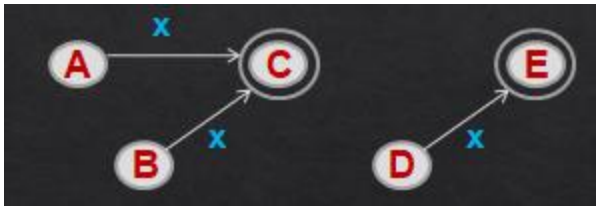


- **Partitioning**
  A partitioning of a set is simply breaking the set into non-overlapping subsets or "groups" based on the "sufficiently different" concept.
  Example:

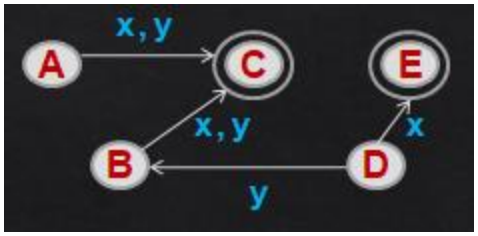
  S = { A, B, C, D, E, F, G }
  Π_1 = { (A B) (C D E F) (G) }
  Π_2 = { (A B) (C) (DE F) (G) }

- **Refine a Partitioning**
  Π_i = { (A B D) (C E) }
  G_1 G_2

  Consider one group (A B D)
  Look at output edges on some symbol (e.g. "x")
  On "x", all states in G_1 go to states belonging to the same group.



Now consider another symbol (e.g. “y”)
D is distinguished from A and B!
So *refine* the partition!
Π_{i+1} = { (A B) (D) (C E) }
G_3 G_4 G_2

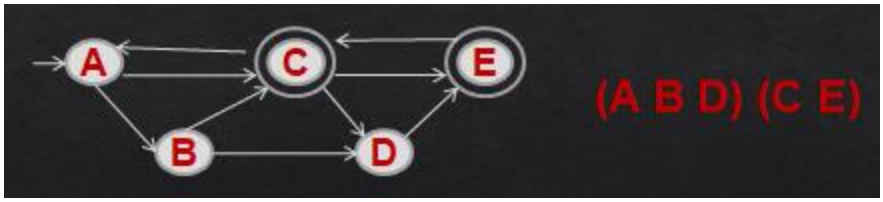


Hopcroft’s Algorithm

Consider the set of states.
Partition it…

- Final States
- All Other States

Repeatedly “refine” the partitioning.
Two states will be placed in different “groups”
…If they can be “distinguished”



Repeat until no group contains states that can be distinguished.
Each group in the partitioning becomes one state in a newly constructed DFA
DFA_{MIN} = The minimal DFA

CSE331 **Automata and Computability**

Lecture 10.2: Minimization of DFA

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

Hopcroft's Algorithm

```
Add dead state and transitions to it if necessary.
      (Now, every state has an outgoing edge on every symbol.)
Π = initial partitioning
loop
       Π_NEW = Refine( Π )
       if ( Π_NEW = Π ) then break
       Π  = Π_NEW
endLoop
Construct DFA_MIN
         • Each group in Π  becomes a state
         • Choose one state in each group
           (throw all other states away)
         • Preserve the edges out  of the chosen state
         • Deal with start state and final states
         • If desired…
                Remove dead state
                Remove any state unreachable from the start state
```

Let's apply this algorithm on an example,

Initial Partitioning:

Π_1 = (**A B C D**) (**E**)

Consider (**A B C D**)

Consider "**a**"

Break apart? **No**

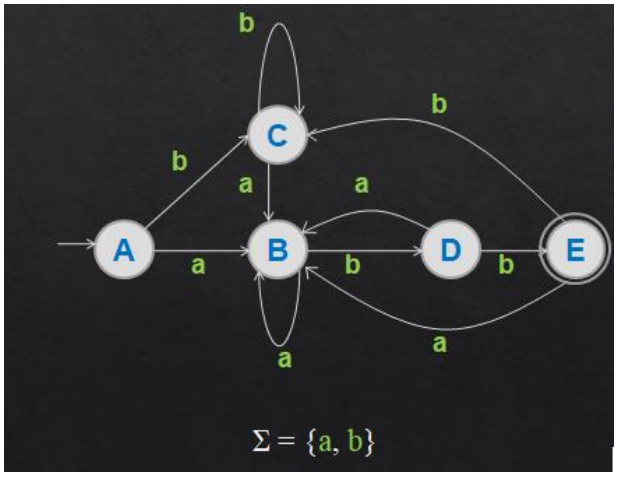
Consider "**b**"

Break apart? **Yes**

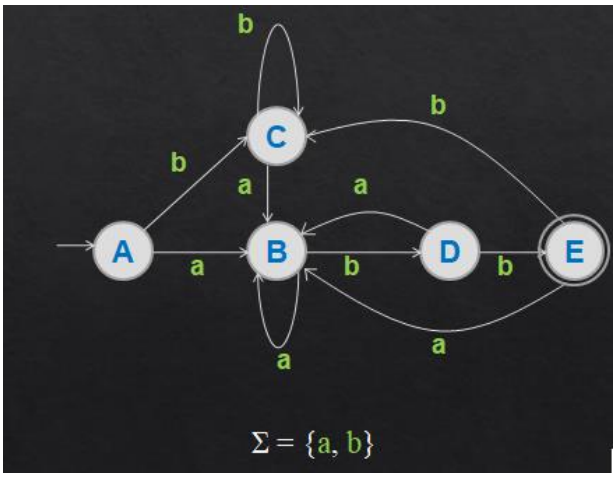
(**A B C**) (**D**)

Consider (**E**)

Not possible to break apart.



New Partitioning:



Π_2 = (**A B C**) (**D**) (**E**)

Consider (**A B C**)

Consider "**a**"

Break apart? No

Consider "**b**"

Break apart? Yes

(**A C**) (B)

Consider (**D**)

Not possible to break apart.

Another New Partitioning:

Π_3 = (**A C**) (**B**) (**D**) (**E**)

Consider (**A C**)

Consider "**a**"

Break apart? No

Consider "**b**"

Break apart? No

Consider (**B**)

Not possible to break apart.

So, **Π_3** = (**A C**) (**B**) (**D**) (**E**) is the final Partitioning.

The Final Minimal DFA: