Lecture 5 – ϵ-Nondeterministic Finite Automata (ϵ-NFA)

COSE215: Theory of Computation

Jihyeok Park

PLRG

2026 Spring

Recall

PLRG

1. Deterministic Finite Automata (DFA)
   - Definition
   - Transition Diagram and Transition Table
   - Extended Transition Function
   - Acceptance of a Word
   - Language of DFA (Regular Language)
   - Examples
2. Nondeterministic Finite Automata (NFA)
   - Definition
   - Transition Diagram and Transition Table
   - Extended Transition Function
   - Language of NFA
   - Examples
3. Equivalence of DFA and NFA
   - DFA $\rightarrow$ NFA
   - DFA $\leftarrow$ NFA (Subset Construction)

Contents

Contents

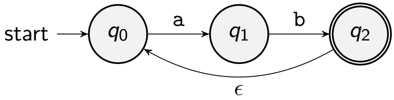
ϵ-Transition

Let's consider ϵ-**transitions** which can be taken **without consuming any input symbol** in finite automata.

ϵ-Transition

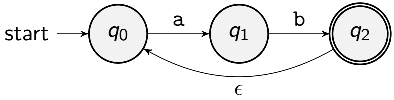
Let's consider ϵ-**transitions** which can be taken **without consuming any input symbol** in finite automata.

For example, the following automaton has an ϵ-**transition** from q_2 to q_0:



Let's consider ϵ-**transitions** which can be taken **without consuming any input symbol** in finite automata.

For example, the following automaton has an ϵ-**transition** from q_2 to q_0:



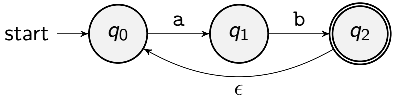
Then, the above automaton **accepts** the following words:

`ab` `abab` `ababab` $\cdots$

ϵ-Transition

Let's consider ϵ-**transitions** which can be taken **without consuming any input symbol** in finite automata.

For example, the following automaton has an ϵ-**transition** from q_2 to q_0:



Then, the above automaton **accepts** the following words:

`ab` `abab` `ababab` $\cdots$

Let's formally define ϵ-**NFA**, an extension of NFA with ϵ- transitions.

Definition of ϵ-NFA

Definition (ϵ-Nondeterministic Finite Automaton (ϵ-NFA))

An ϵ-**nondeterministic finite automaton** is a 5-tuple:

$$N^\epsilon = (Q, \Sigma, \delta, q_0, F)$$

- Q is a finite set of **states**
- Σ is a finite set of **symbols**
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is the **transition function**
- $q_0 \in Q$ is the **initial state**
- $F \subseteq Q$ is the set of **final states**

Definition of ϵ-NFA

PLRG

Definition (ϵ-Nondeterministic Finite Automaton (ϵ-NFA))

An ϵ-**nondeterministic finite automaton** is a 5-tuple:

$$N^{\epsilon} = (Q, \Sigma, \delta, q_0, F)$$

- Q is a finite set of **states**
- Σ is a finite set of **symbols**
- $\delta : Q \times (\Sigma \cup \{\epsilon\}) \to \mathcal{P}(Q)$ is the **transition function**
- $q_0 \in Q$ is the **initial state**
- $F \subseteq Q$ is the set of **final states**

$$N_1^{\epsilon} = (\{q_0, q_1, q_2\}, \{\mathtt{a}, \mathtt{b}\}, \delta, q_0, \{q_2\})$$

$$\begin{array}{lll} \delta(q_0, \mathtt{a}) = \{q_1\} & \delta(q_1, \mathtt{a}) = \varnothing & \delta(q_2, \mathtt{a}) = \varnothing \\ \delta(q_0, \mathtt{b}) = \varnothing & \delta(q_1, \mathtt{b}) = \{q_2\} & \delta(q_2, \mathtt{b}) = \varnothing \\ \delta(q_0, \epsilon) = \varnothing & \delta(q_1, \epsilon) = \varnothing & \delta(q_2, \epsilon) = \{q_0\} \end{array}$$

Definition of ϵ-NFA

```scala
// The definition of epsilon-NFA
case class ENFA(
  states: Set[State],
  symbols: Set[Symbol],
  trans: Map[(State, Option[Symbol]), Set[State]],
  initState: State,
  finalStates: Set[State],
)
```

Definition of ϵ-NFA

PLRG

```scala
// The definition of epsilon-NFA
case class ENFA(
  states: Set[State],
  symbols: Set[Symbol],
  trans: Map[(State, Option[Symbol]), Set[State]],
  initState: State,
  finalStates: Set[State],
)
```

```scala
// An example of epsilon-NFA
val enfa1: ENFA = ENFA(
  states = Set(0, 1, 2),
  symbols = Set('a', 'b'),
  trans = Map(
    (0, Some('a')) -> Set(1),      // (0, a) -> 1
    (1, Some('b')) -> Set(2),      // (1, b) -> 2
    (2, None)      -> Set(0),      // (2, ϵ) -> 0
  ).withDefaultValue(Set()),
  initState   = 0,
  finalStates = Set(2),
)
```

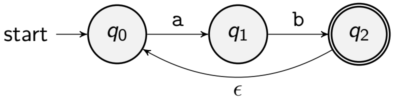
Transition Diagram and Transition Table

PLRG

$$N_1^{\epsilon} = (\{q_0, q_1, q_2\}, \{\texttt{a}, \texttt{b}\}, \delta, q_0, \{q_2\})$$

$$\delta(q_0, \texttt{a}) = \{q_1\} \qquad \delta(q_1, \texttt{a}) = \varnothing \qquad \delta(q_2, \texttt{a}) = \varnothing$$
$$\delta(q_0, \texttt{b}) = \varnothing \qquad \delta(q_1, \texttt{b}) = \{q_2\} \qquad \delta(q_2, \texttt{b}) = \varnothing$$
$$\delta(q_0, \epsilon) = \varnothing \qquad \delta(q_1, \epsilon) = \varnothing \qquad \delta(q_2, \epsilon) = \{q_0\}$$

Transition Diagram



Transition Table

q	a	b	ϵ
$\to q_0$	$\{q_1\}$	$\varnothing$	$\varnothing$
q_1	$\varnothing$	$\{q_2\}$	$\varnothing$
$*q_2$	$\varnothing$	$\varnothing$	$\{q_0\}$

ϵ-Closures

Definition (ϵ-Closures)

The ϵ-**closure** $\text{EClo}(q)$ for a state q is the set of all reachable states only through ϵ-transitions from q, and it can be inductively defined as:

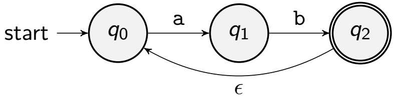
- **(Basis Case)** $q \in \text{EClo}(q)$
- **(Induction Case)** $(q' \in \delta(q, \epsilon) \land q'' \in \text{EClo}(q')) \Rightarrow q'' \in \text{EClo}(q)$

ϵ-Closures

Definition (ϵ-Closures)

The ϵ-**closure** $\text{EClo}(q)$ for a state q is the set of all reachable states only through ϵ-transitions from q, and it can be inductively defined as:

- **(Basis Case)** $q \in \text{EClo}(q)$
- **(Induction Case)** $(q' \in \delta(q, \epsilon) \land q'' \in \text{EClo}(q')) \Rightarrow q'' \in \text{EClo}(q)$



$$\text{EClo}(q_2) = \{q_0, q_2\}$$

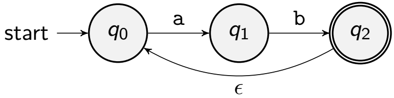
ϵ-Closures

PLRG

Definition (ϵ-Closures)

The ϵ-**closure** $\mathsf{EClo}(q)$ for a state q is the set of all reachable states only through ϵ-transitions from q, and it can be inductively defined as:

- **(Basis Case)** $q \in \mathsf{EClo}(q)$
- **(Induction Case)** $(q' \in \delta(q, \epsilon) \wedge q'' \in \mathsf{EClo}(q')) \Rightarrow q'' \in \mathsf{EClo}(q)$



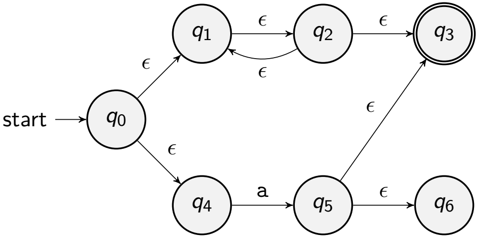
$$\mathsf{EClo}(q_2) = \{q_0, q_2\}$$

We sometimes need to define the ϵ-closure for a **set of states** $S \subseteq Q$:

$$\forall S \subseteq Q.\ \mathsf{EClo}(S) = \bigcup_{q \in S} \mathsf{EClo}(q)$$

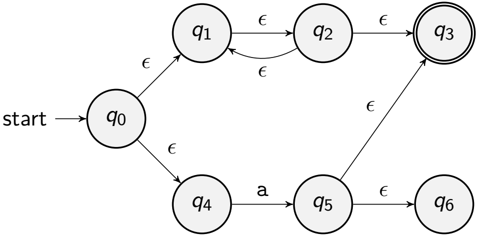
ϵ-Closures – Example

$N_2^\epsilon =$



$$\text{EClo}(q_0) \quad =$$

ϵ-Closures – Example

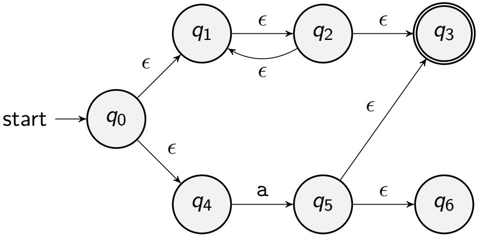
$N_2^\epsilon =$ 

$$\mathsf{EClo}(q_0) = \{q_0, q_1, q_2, q_3, q_4\}$$

$$\mathsf{EClo}(q_2) =$$

ϵ-Closures – Example

$N_2^\epsilon =$

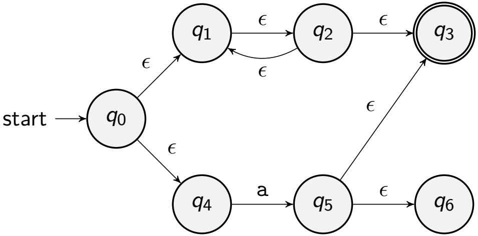


$$\text{ECl}(q_0) = \{q_0, q_1, q_2, q_3, q_4\}$$

$$\text{ECl}(q_2) = \{q_1, q_2, q_3\}$$

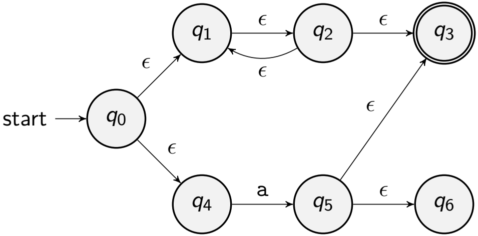
$$\text{ECl}(q_5) =$$

ϵ-Closures – Example

$N_2^{\epsilon} =$ 

$$
\begin{aligned}
\mathrm{EClo}(q_0) &= \{q_0, q_1, q_2, q_3, q_4\} \\
\mathrm{EClo}(q_2) &= \{q_1, q_2, q_3\} \\
\mathrm{EClo}(q_5) &= \{q_3, q_5, q_6\} \\
\mathrm{EClo}(\{q_2, q_5\}) &=
\end{aligned}
$$

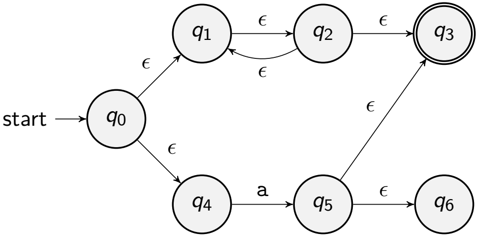
ϵ-Closures – Example

$N_2^\epsilon =$ 

$$
\begin{aligned}
\mathrm{EClo}(q_0) &= \{q_0, q_1, q_2, q_3, q_4\} \\
\mathrm{EClo}(q_2) &= \{q_1, q_2, q_3\} \\
\mathrm{EClo}(q_5) &= \{q_3, q_5, q_6\} \\
\mathrm{EClo}(\{q_2, q_5\}) &= \mathrm{EClo}(q_2) \cup \mathrm{EClo}(q_5) \\
&=
\end{aligned}
$$

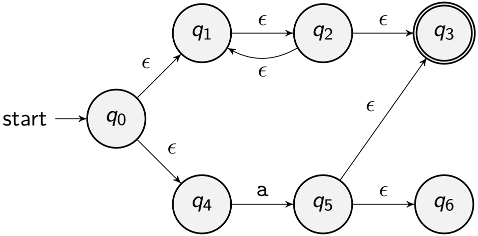
ϵ-Closures – Example

$N_2^\epsilon =$



$$
\begin{aligned}
\text{EClo}(q_0) &= \{q_0, q_1, q_2, q_3, q_4\} \\
\text{EClo}(q_2) &= \{q_1, q_2, q_3\} \\
\text{EClo}(q_5) &= \{q_3, q_5, q_6\} \\
\text{EClo}(\{q_2, q_5\}) &= \text{EClo}(q_2) \cup \text{EClo}(q_5) \\
&= \{q_1, q_2, q_3\} \cup \{q_3, q_5, q_6\} \\
&=
\end{aligned}
$$

ϵ-Closures – Example

$N_2^\epsilon =$ 

$$
\begin{aligned}
\text{EClo}(q_0) &= \{q_0, q_1, q_2, q_3, q_4\} \\
\text{EClo}(q_2) &= \{q_1, q_2, q_3\} \\
\text{EClo}(q_5) &= \{q_3, q_5, q_6\} \\
\text{EClo}(\{q_2, q_5\}) &= \text{EClo}(q_2) \cup \text{EClo}(q_5) \\
&= \{q_1, q_2, q_3\} \cup \{q_3, q_5, q_6\} \\
&= \{q_1, q_2, q_3, q_5, q_6\}
\end{aligned}
$$

ϵ-Closures

Then, how to implement the `eclo` method for ϵ-closure?

```
case class ENFA(...):
  ...

  // The epsilon-closure of a state
  def eclo(q: State): Set[State] = ???

  // The epsilon-closure of a set of states
  def eclo(qs: Set[State]): Set[State] = qs.flatMap(eclo)
```

ϵ-Closures

Then, how to implement the `eclo` method for ϵ-closure?

```
case class ENFA(...):
  ...

  // The epsilon-closure of a state
  def eclo(q: State): Set[State] = ???

  // The epsilon-closure of a set of states
  def eclo(qs: Set[State]): Set[State] = qs.flatMap(eclo)
```

The ϵ-closures for states 0, 2, 5, and $\{2, 5\}$ are as follows:

```
// Another example of epsilon-NFA
val enfa2: ENFA = ENFA(...)
enfa2.eclo(0)          // Set(0, 1, 2, 3, 4)
enfa2.eclo(2)          // Set(1, 2, 3)
enfa2.eclo(5)          // Set(3, 5, 6)
enfa2.eclo(Set(2, 5)) // Set(1, 2, 3, 5, 6)
```

ϵ-Closures

PLRG

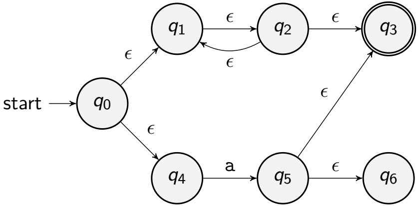
```
case class ENFA(...):
  ...
  // The epsilon-closure of a state
  def wrongEClo(q: State): Set[State] =
    val basis = Set(q)                                       // Basis Case
    val induc = enfa.trans(q, None).flatMap(wrongEClo)       // Induction Case
    basis ++ induc
```

ϵ-Closures

```
case class ENFA(...):
  ...
  // The epsilon-closure of a state
  def wrongEClo(q: State): Set[State] =
    val basis = Set(q)                                       // Basis Case
    val induc = enfa.trans(q, None).flatMap(wrongEClo)       // Induction Case
    basis ++ induc
```

The above implementation is **WRONG** because of **infinite loop**:

```
enfa2.wrongEClo(5) // Set(3, 5, 6)
enfa2.wrongEClo(2) // INFINITE LOOP -- cycle between states 1 and 2
```

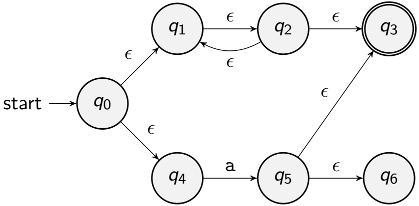


$\mathsf{EClo}(q_5) = \{q_3, q_5, q_6\}$ $\mathsf{EClo}(q_2) = \{q_1, q_2, q_3\}$

ϵ-Closures

We can resolve the infinite loop issue by keeping the **visited states**:

```
case class ENFA(...):
  ...
  // The definitions of epsilon-closures
  def eclo(q: State): Set[State] =
    def aux(rest: List[State], visited: Set[State]): Set[State] = rest match
      case Nil           => visited
      case p :: targets => aux(
        rest    = (trans((p, None)) -- visited - p).toList ++ targets,
        visited = visited + p,
      )
    aux(List(q), Set())
```



- Rest: q_2
- Visited:

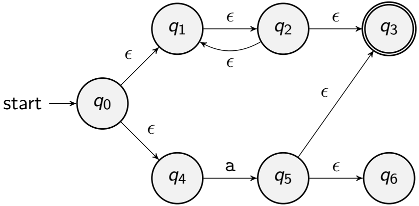
$$\text{EClo}(q_2) = \{q_1, q_2, q_3\}$$

ϵ-Closures

PLRG

We can resolve the infinite loop issue by keeping the **visited states**:

```
case class ENFA(...):
  ...
  // The definitions of epsilon-closures
  def eclo(q: State): Set[State] =
    def aux(rest: List[State], visited: Set[State]): Set[State] = rest match
      case Nil          => visited
      case p :: targets => aux(
        rest    = (trans((p, None)) -- visited - p).toList ++ targets,
        visited = visited + p,
      )
    aux(List(q), Set())
```



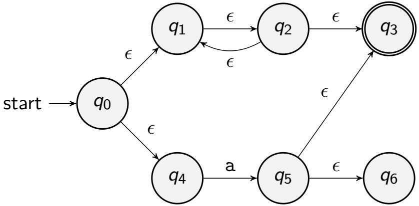
- Rest: q_1, q_3
- Visited: q_2

$$\text{EClo}(q_2) = \{q_1, q_2, q_3\}$$

ϵ-Closures

We can resolve the infinite loop issue by keeping the **visited states**:

```
case class ENFA(...):
  ...
  // The definitions of epsilon-closures
  def eclo(q: State): Set[State] =
    def aux(rest: List[State], visited: Set[State]): Set[State] = rest match
      case Nil          => visited
      case p :: targets => aux(
        rest    = (trans((p, None)) -- visited - p).toList ++ targets,
        visited = visited + p,
      )
    aux(List(q), Set())
```



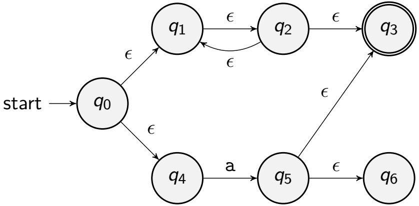
- Rest: q_3
- Visited: q_2, q_1

$$\text{EClo}(q_2) = \{q_1, q_2, q_3\}$$

ϵ-Closures

We can resolve the infinite loop issue by keeping the **visited states**:

```
case class ENFA(...):
  ...
  // The definitions of epsilon-closures
  def eclo(q: State): Set[State] =
    def aux(rest: List[State], visited: Set[State]): Set[State] = rest match
      case Nil          => visited
      case p :: targets => aux(
        rest    = (trans((p, None)) -- visited - p).toList ++ targets,
        visited = visited + p,
      )
    aux(List(q), Set())
```



- Rest:
- Visited: q_2, q_1, q_3

$$\mathsf{EClo}(q_2) = \{q_1, q_2, q_3\}$$

Extended Transition Function

PLRG

Definition (Extended Transition Function)

For a given ϵ-NFA $N^\epsilon = (Q, \Sigma, \delta, q_0, F)$, the **extended transition function** $\delta^* : \mathcal{P}(Q) \times \Sigma^* \to \mathcal{P}(Q)$ is defined as follows:

- **(Basis Case)** $\delta^*(S, \epsilon) = \mathsf{EClo}(S)$
- **(Induction Case)** $\delta^*(S, aw) = \delta^*(\bigcup_{q \in \mathsf{EClo}(S)} \delta(q, a), w)$

```
case class ENFA(...):
  ...

  // The extended transition function of epsilon-NFA
  def extTrans(qs: Set[State], w: Word): Set[State] = w match
    case ""     => eclo(qs)
    case x <| w => extTrans(eclo(qs).flatMap(q => trans(q, Some(x))), w)
```

Language of ϵ-NFA

Definition (Acceptance of a Word)

For a given ϵ-NFA $N^\epsilon = (Q, \Sigma, \delta, q_0, F)$, we say that N^ϵ **accepts** a word $w \in \Sigma^*$ if and only if $\delta^*(q_0, w) \cap F \neq \varnothing$

Language of ϵ-NFA

Definition (Acceptance of a Word)

For a given ϵ-NFA $N^\epsilon = (Q, \Sigma, \delta, q_0, F)$, we say that N^ϵ **accepts** a word $w \in \Sigma^*$ if and only if $\delta^*(q_0, w) \cap F \neq \varnothing$

```
case class ENFA(...):
  ...

  // The acceptance of a word by epsilon-NFA
  def accept(w: Word): Boolean =
    extTrans(Set(initState), w).intersect(finalStates).nonEmpty
```

Language of ϵ-NFA

Definition (Acceptance of a Word)

For a given ϵ-NFA $N^\epsilon = (Q, \Sigma, \delta, q_0, F)$, we say that N^ϵ **accepts** a word $w \in \Sigma^*$ if and only if $\delta^*(q_0, w) \cap F \neq \varnothing$

```
case class ENFA(...):
  ...

  // The acceptance of a word by epsilon-NFA
  def accept(w: Word): Boolean =
    extTrans(Set(initState), w).intersect(finalStates).nonEmpty
```

Definition (Language of ϵ-NFA)

For a given ϵ-NFA $N^\epsilon = (Q, \Sigma, \delta, q_0, F)$, the **language** of N^ϵ is defined as follows:

$$L(N^\epsilon) = \{w \in \Sigma^* \mid N^\epsilon \text{ accepts } w\}$$

Contents

Equivalence of DFA and ϵ-NFA

Theorem (Equivalence of DFA and ϵ-NFA)

A language L is the language $L(D)$ of a DFA D if and only if L is the language $L(N^\epsilon)$ of an ϵ-NFA N^ϵ.

Equivalence of DFA and ϵ-NFA

PLRG

Theorem (Equivalence of DFA and ϵ-NFA)

A language L is the language $L(D)$ of a DFA D if and only if L is the language $L(N^\epsilon)$ of an ϵ-NFA N^ϵ.

Proof) By the following two theorems.

Theorem (DFA to ϵ-NFA)

For a given DFA $D = (Q, \Sigma, \delta, q, F)$, $\exists$ ϵ-NFA N^ϵ. $L(D) = L(N^\epsilon)$.

Theorem (ϵ-NFA to DFA – Subset Construction)

For a given ϵ-NFA $N^\epsilon = (Q, \Sigma, \delta, q_0, F)$, $\exists$ DFA D. $L(D) = L(N^\epsilon)$.

The formal proofs are exercises for you

Equivalence of DFA and ϵ-NFA

Theorem (Equivalence of DFA and ϵ-NFA)

A language L is the language $L(D)$ of a DFA D if and only if L is the language $L(N^\epsilon)$ of an ϵ-NFA N^ϵ.

Proof) By the following two theorems.

Theorem (DFA to ϵ-NFA)

For a given DFA $D = (Q, \Sigma, \delta, q, F)$, $\exists$ ϵ-NFA N^ϵ. $L(D) = L(N^\epsilon)$.

Theorem (ϵ-NFA to DFA – Subset Construction)

For a given ϵ-NFA $N^\epsilon = (Q, \Sigma, \delta, q_0, F)$, $\exists$ DFA D. $L(D) = L(N^\epsilon)$.

The formal proofs are exercises for you

Let's see **examples** of the second theorem (DFA $\leftarrow$ ϵ-NFA)

Theorem (ϵ-NFA to DFA – Subset Construction)

For a given ϵ-NFA $N^\epsilon = (Q_{N^\epsilon}, \Sigma, \delta_{N^\epsilon}, q_0, F_{N^\epsilon})$, $\exists$ DFA D. $L(D) = L(N^\epsilon)$.

Proof) Define a DFA

$$D = (Q_D, \Sigma, \delta_D, \mathsf{EClo}(q_0), F_D)$$

where

- $Q_D = \{S \subseteq Q_{N^\epsilon} \mid S = \mathsf{EClo}(S)\}$

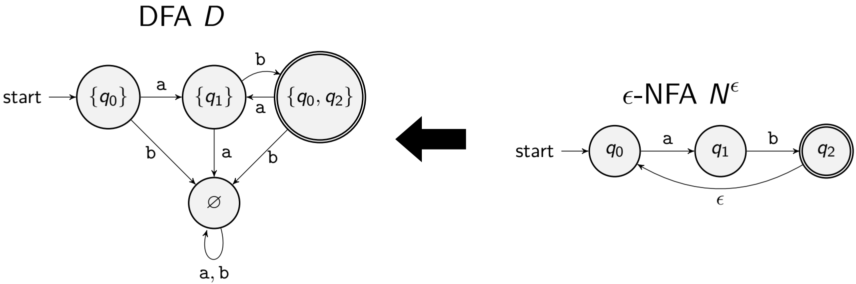
  The states of D are the sets of states of N^ϵ whose ϵ-**closures are themselves** (i.e., $\mathsf{EClo}(S) = S$).

- $\forall S \in Q_D.\ \forall a \in \Sigma.$

$$\delta_D(S, a) = \mathsf{EClo}\left(\bigcup_{q \in S} \delta_{N^\epsilon}(q, a)\right)$$

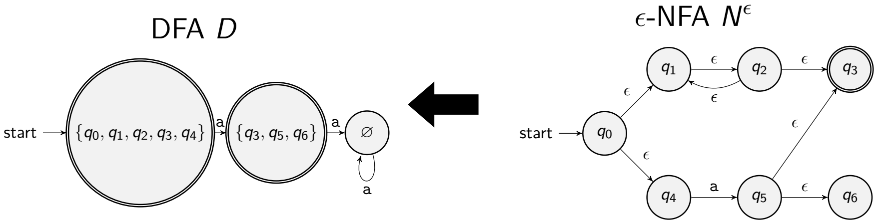
- $F_D = \{S \in Q_D \mid S \cap F \neq \varnothing\}$

DFA $\leftarrow$ ϵ-NFA (Subset Construction) – Examples



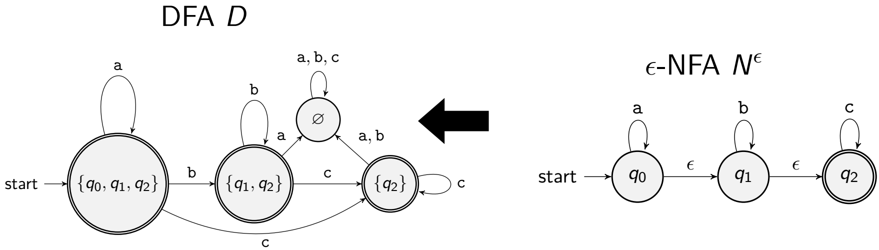
$$L(D) = L(N^\epsilon) = \{(\texttt{ab})^n \mid n \geq 1\}$$

DFA ← ϵ-NFA (Subset Construction) – Examples



$$L(D) = L(N^\epsilon) = \{\epsilon, \mathtt{a}\}$$

DFA $\leftarrow$ ϵ-NFA (Subset Construction) – Examples



$$L(D) = L(N^\epsilon) = \{\texttt{a}^i\texttt{b}^j\texttt{c}^k \mid i, j, k \geq 0\}$$

Summary

1. ϵ-Nondeterministic Finite Automata (ϵ-NFA)
   - ϵ-Transition
   - Definition
   - Transition Diagram and Transition Table
   - ϵ-Closures
   - Extended Transition Function
   - Language of ϵ-NFA

2. Equivalence of DFA and ϵ-NFA
   - DFA $\leftarrow$ ϵ-NFA (Subset Construction)

Homework #1

- Please see this document on GitHub:

https://github.com/ku-plrg-classroom/docs/tree/main/cose215/fa-examples

- The due date is 23:59 on Mar. 30 (Mon.).
- Please only submit `Implementation.scala` file to **LMS**.
- Late submission policy:
  - 1 day late: 20% penalty
  - 2 or more days late: no credit

Next Lecture

- Regular Expressions and Languages

Jihyeok Park
jihyeok_park@korea.ac.kr
https://plrg.korea.ac.kr