

CMSC 330: Organization of Programming Languages

DFAs, and NFAs, and Regexps

The story so far, and what's next

- ▶ Goal: Develop an algorithm that determines whether a string s is matched by regex R
 - I.e., whether s is a member of R 's *language*
- ▶ Approach to come: Convert R to a finite automaton FA and see whether s is accepted by FA
 - Details: Convert R to a *nondeterministic FA* (NFA), which we then convert to a *deterministic FA* (DFA),
 - which enjoys a fast acceptance algorithm

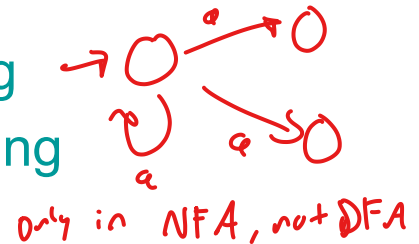
Two Types of Finite Automata

▶ **Deterministic** Finite Automata (DFA)

- Exactly one sequence of steps for each string
 - Easy to implement acceptance check
- (Almost) all examples so far

▶ **Nondeterministic** Finite Automata (NFA)

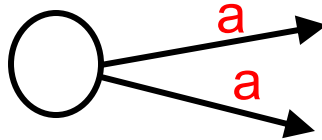
- May have many sequences of steps for each string
- Accepts if **any path** ends in final state at end of string
- More compact than DFA
 - But more expensive to test whether a string matches



$0 \xrightarrow{\epsilon} 0$ ✓

Comparing DFAs and NFAs

- ▶ NFAs can have **more** than one transition leaving a state on the same symbol



- ▶ DFAs allow only one transition per symbol
 - DFA is a special case of NFA

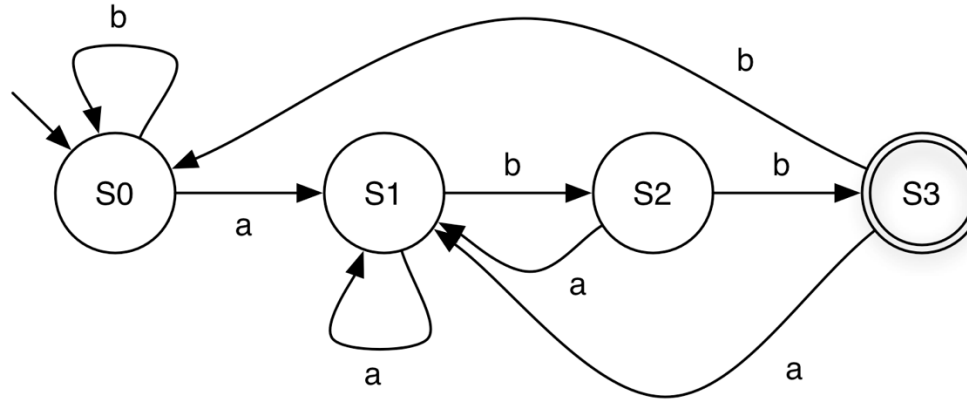
Comparing DFAs and NFAs (cont.)

- ▶ NFAs may have transitions with empty string label
 - May move to new state without consuming character

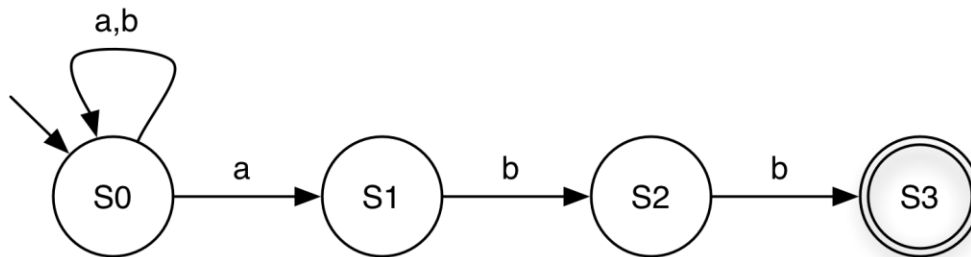


- ▶ DFA transition must be labeled with symbol
 - A DFA is a specific kind of NFA

DFA for $(a|b)^*abb$



NFA for $(a|b)^*abb$



► ba

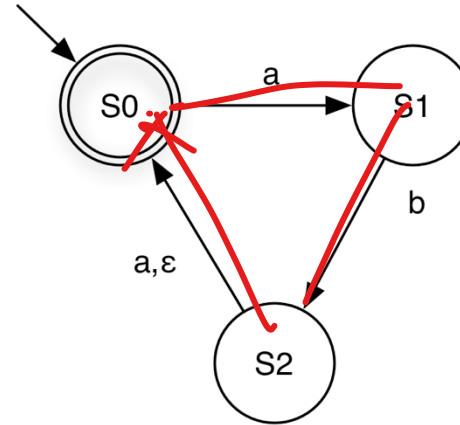
- Has paths to either S0 or S1
- Neither is final, so rejected

► babaabb

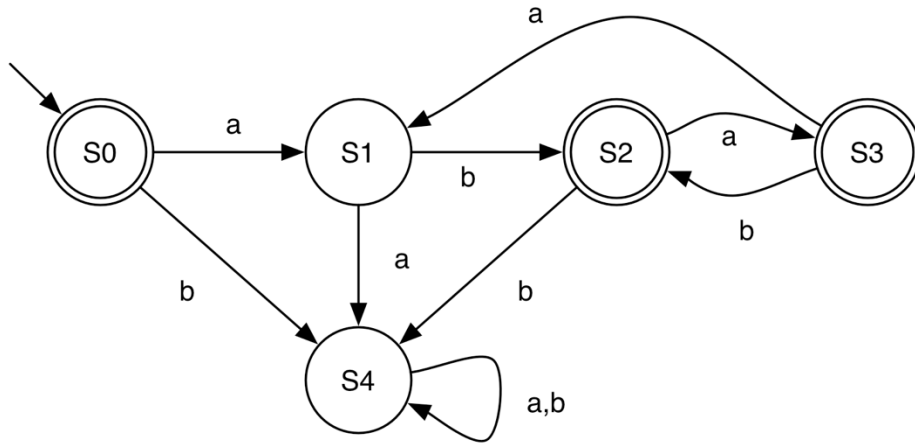
- Has paths to different states
- One path leads to S3, so accepts string

NFA for $(ab|aba)^*$

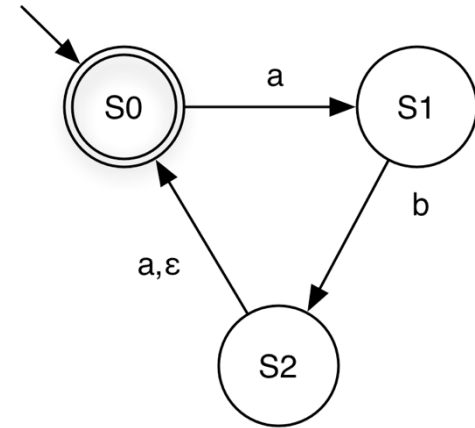
- ▶ aba ✓
- ▶ ababa ✗
 - Has paths to states S0, S1
 - Need to use ϵ -transition



NFA and DFA for $(ab|aba)^*$



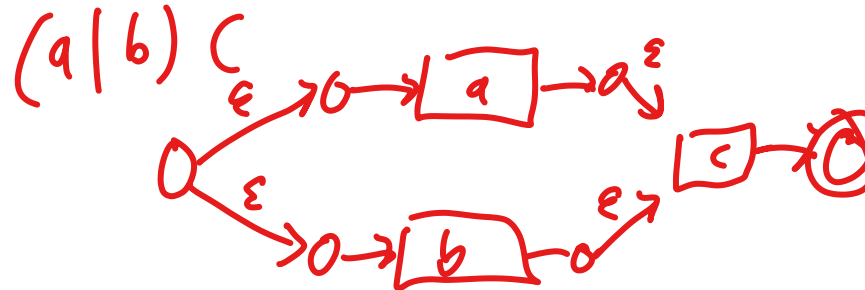
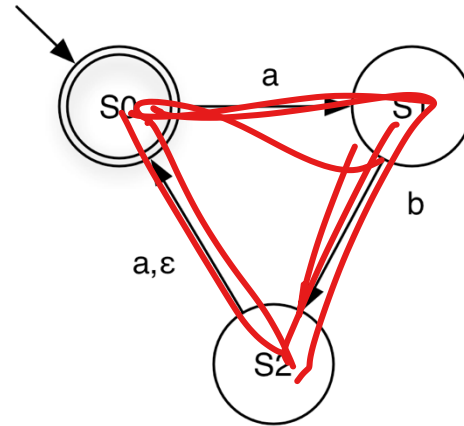
DFA



NFA

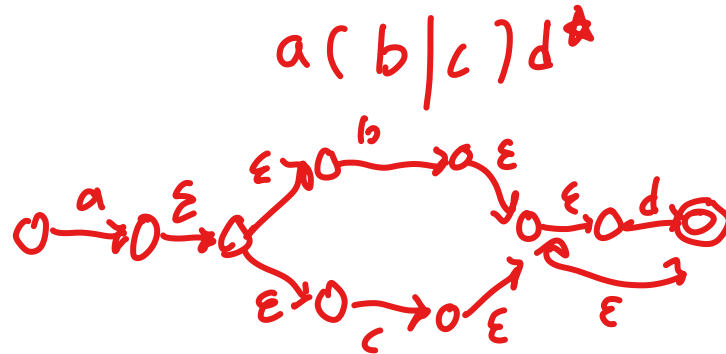
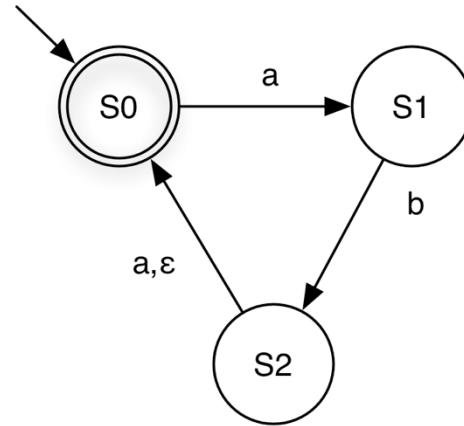
Quiz 1: Which string is **NOT** accepted by this NFA?

- ~~A.~~ ab
- B. abaa
- c. abab
- ~~D.~~ abaab



Quiz 1: Which string is **NOT** accepted by this NFA?

- A. ab
- B. **abaa**
- C. abab
- D. abaab

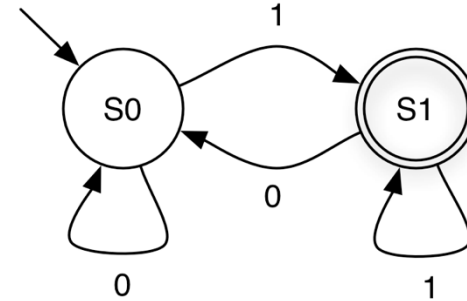


Formal Definition

- ▶ A **deterministic finite automaton** (*DFA*) is a 5-tuple $(\Sigma, Q, q_0, F, \delta)$ where
 - Σ is an alphabet
 - Q is a nonempty set of states
 - $q_0 \in Q$ is the start state
 - $F \subseteq Q$ is the set of final states
 - $\delta : Q \times \Sigma \rightarrow Q$ specifies the DFA's transitions
 - What's this definition saying that δ is?
- ▶ A DFA accepts s if it **stops** at a final state on s

Formal Definition: Example

- $\Sigma = \{0, 1\}$
- $Q = \{S0, S1\}$
- $q_0 = S0$
- $F = \{S1\}$
- $\delta =$

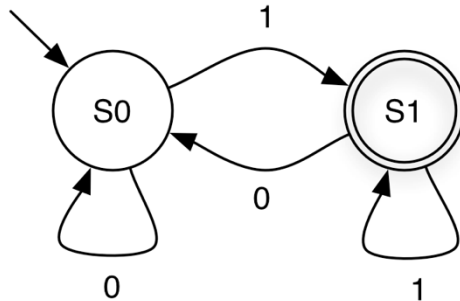


		symbol	
		0	1
input state	S0	S0	S1
	S1	S0	S1

or as $\{ (S0,0,S0), (S0,1,S1), (S1,0,S0), (S1,1,S1) \}$

Implementing DFAs (one-off)

It's easy to build
a program
which mimics a
DFA



```
cur_state = 0;
while (1) {

    symbol = getchar();

    switch (cur_state) {

        case 0: switch (symbol) {
            case '0': cur_state = 0; break;
            case '1': cur_state = 1; break;
            case '\n': printf("rejected\n"); return 0;
            default: printf("rejected\n"); return 0;
        }
        break;

        case 1: switch (symbol) {
            case '0': cur_state = 0; break;
            case '1': cur_state = 1; break;
            case '\n': printf("accepted\n"); return 1;
            default: printf("rejected\n"); return 0;
        }
        break;

        default: printf("unknown state; I'm confused\n");
            break;
    }
}
```

Implementing DFAs (generic)

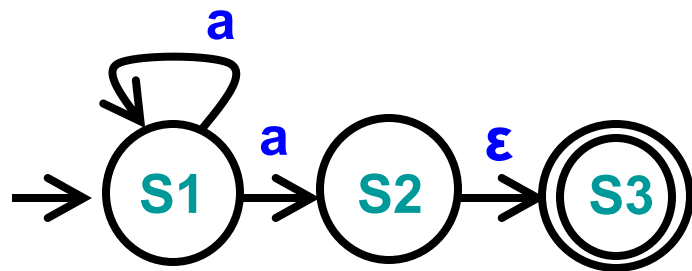
More generally, use generic table-driven DFA

```
given components  $(\Sigma, Q, q_0, F, \delta)$  of a DFA:  
let  $q = q_0$   
while (there exists another symbol  $\sigma$  of the input string)  
     $q := \delta(q, \sigma)$ ;  
if  $q \in F$  then  
    accept  
else reject
```

- q is just an integer
- Represent δ using arrays or hash tables
- Represent F as a set

Nondeterministic Finite Automata (NFA)

- ▶ An *NFA* is a 5-tuple $(\Sigma, Q, q_0, F, \delta)$ where
 - Σ, Q, q_0, F as with DFAs
 - $\delta \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times Q$ specifies the NFA's transitions



Example

- $\Sigma = \{a\}$
- $Q = \{S1, S2, S3\}$
- $q_0 = S1$
- $F = \{S3\}$
- $\delta = \{(S1, a, S1), (S1, a, S2), (S2, \epsilon, S3)\}$

- ▶ An NFA accepts s if there is **at least one path** via s from the NFA's start state to a final state

NFA Acceptance Algorithm (Sketch)

- ▶ When NFA processes a string s
 - NFA must keep track of several “current states”
 - Due to multiple transitions with same label, and ϵ -transitions
 - If any current state is final when done then accept s

- ▶ Example

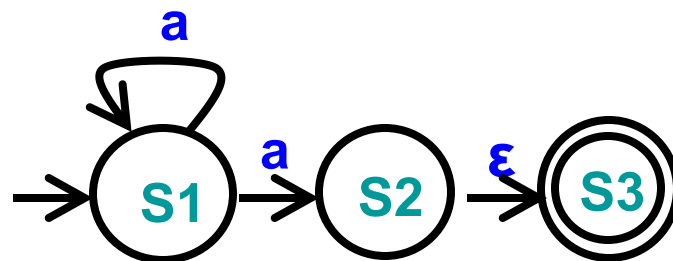
- After processing “a”
 - NFA may be in states

S1

S2

S3

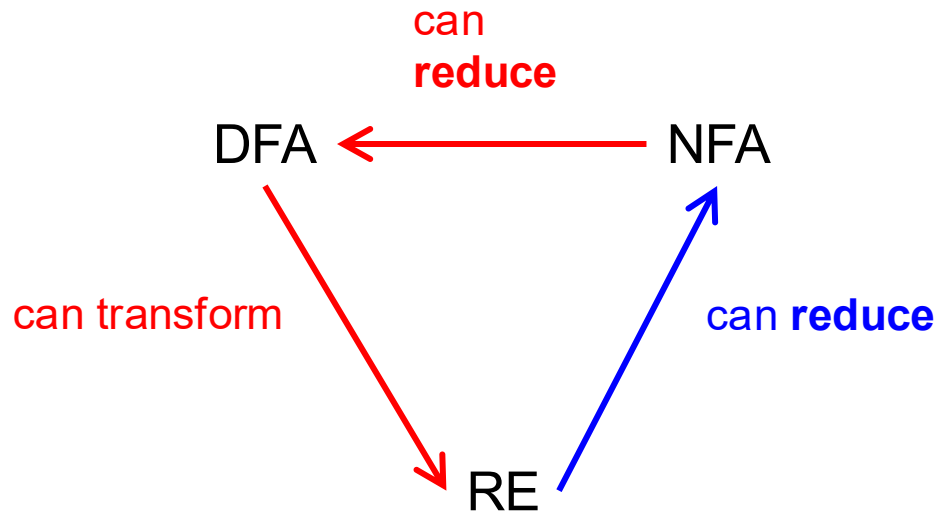
- Since S3 is final, s is accepted



- ▶ Algorithm is slow, space-inefficient; prefer DFAs!

Relating REs to DFAs and NFAs

- ▶ Regular expressions, NFAs, and DFAs accept the same languages! *Can convert between them*

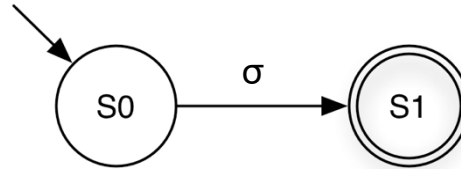


Reducing Regular Expressions to NFAs

- ▶ Goal: Given regular expression A , construct NFA: $\langle A \rangle = (\Sigma, Q, q_0, F, \delta)$
 - Remember regular expressions are defined recursively from primitive RE languages
 - Invariant: $|F| = 1$ in our NFAs
 - Recall F = set of final states
- ▶ Will define $\langle A \rangle$ for base cases: $\sigma, \varepsilon, \emptyset$
 - Where σ is a symbol in Σ
- ▶ And for inductive cases: $AB, A|B, A^*$

Reducing Regular Expressions to NFAs

► Base case: σ



Recall: NFA is $(\Sigma, Q, q_0, F, \delta)$

where

Σ is the alphabet

Q is set of states

q_0 is starting state

F is set of final states

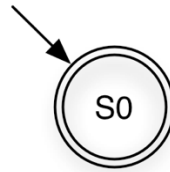
δ is transition relation

$$\langle \sigma \rangle = (\{\sigma\}, \{S0, S1\}, S0, \{S1\}, \{(S0, \sigma, S1)\})$$

$$(\Sigma, Q, q_0, F, \delta)$$

Reduction

- ▶ Base case: ϵ



$$\langle \epsilon \rangle = (\emptyset, \{S0\}, S0, \{S0\}, \emptyset)$$

Recall: NFA is $(\Sigma, Q, q_0, F, \delta)$

where

Σ is the alphabet

Q is set of states

q_0 is starting state

F is set of final states

δ is transition relation

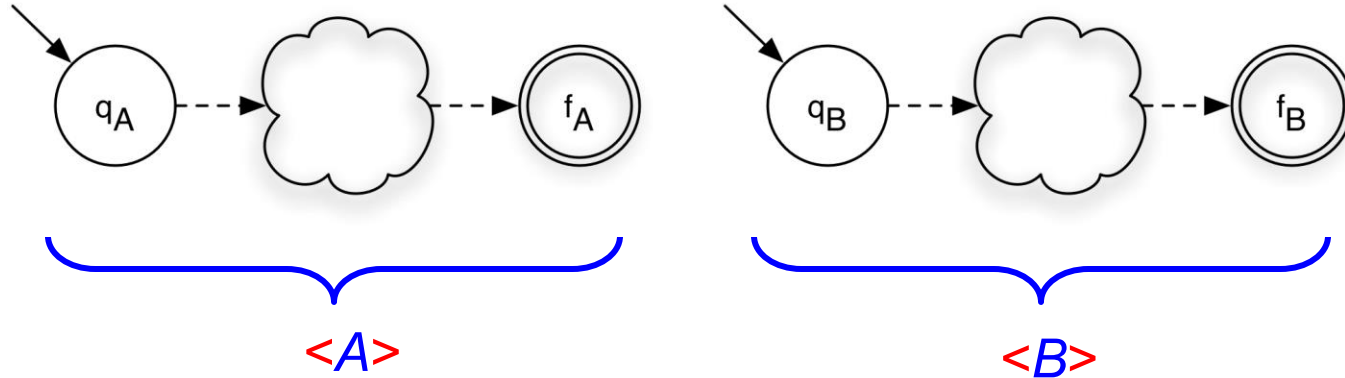
- ▶ Base case: \emptyset



$$\langle \emptyset \rangle = (\emptyset, \{S0, S1\}, S0, \{S1\}, \emptyset)$$

Reduction: Concatenation

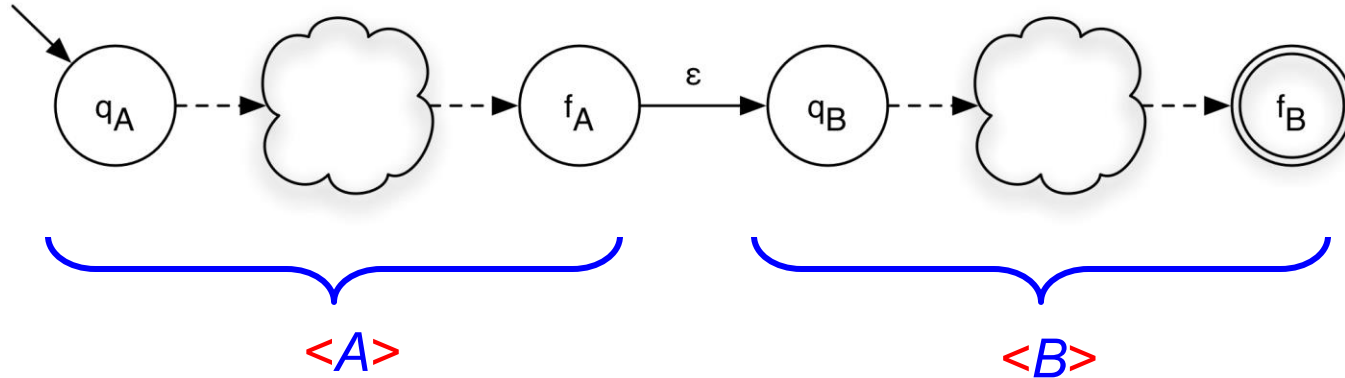
► Induction: AB



- $\langle A \rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $\langle B \rangle = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$

Reduction: Concatenation

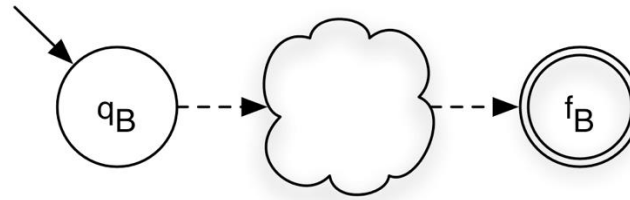
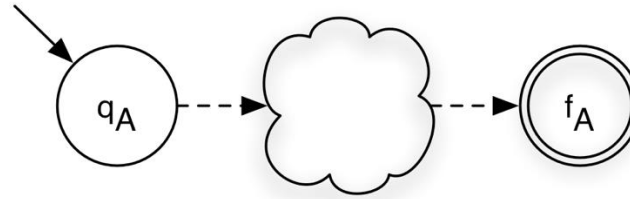
► Induction: AB



- $\langle A \rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $\langle B \rangle = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
- $\langle AB \rangle = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B, q_A, \{f_B\}, \delta_A \cup \delta_B \cup \{(f_A, \varepsilon, q_B)\})$

Reduction: Union

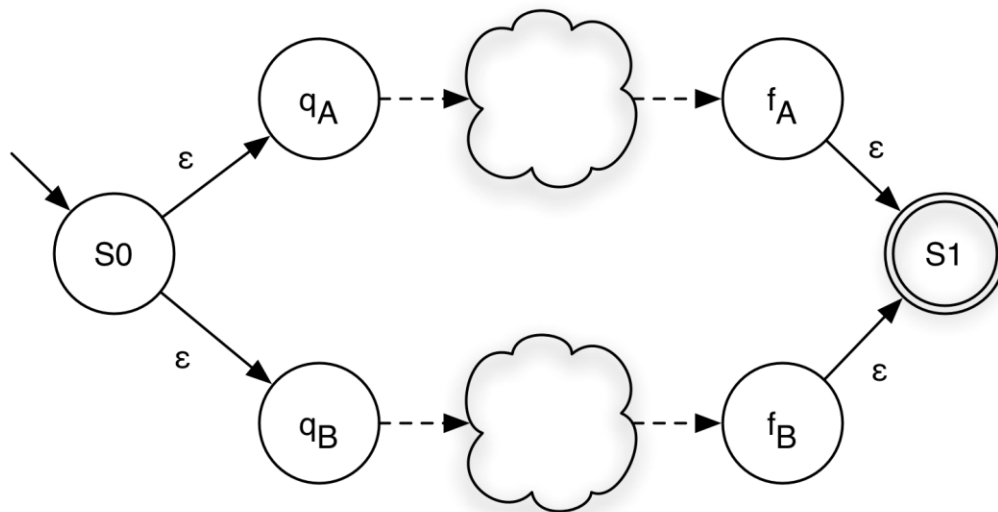
► Induction: $A|B$



- $\langle A \rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $\langle B \rangle = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$

Reduction: Union

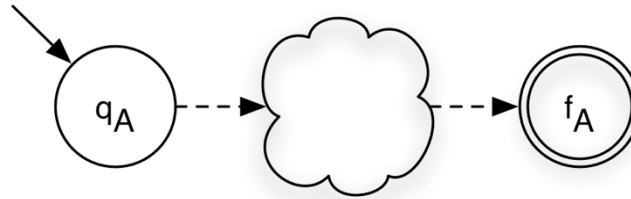
► Induction: $A|B$



- $\langle A \rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $\langle B \rangle = (\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
- $\langle A|B \rangle = (\Sigma_A \cup \Sigma_B, Q_A \cup Q_B \cup \{S0, S1\}, S0, \{S1\}, \delta_A \cup \delta_B \cup \{(S0, \epsilon, q_A), (S0, \epsilon, q_B), (f_A, \epsilon, S1), (f_B, \epsilon, S1)\})$

Reduction: Closure

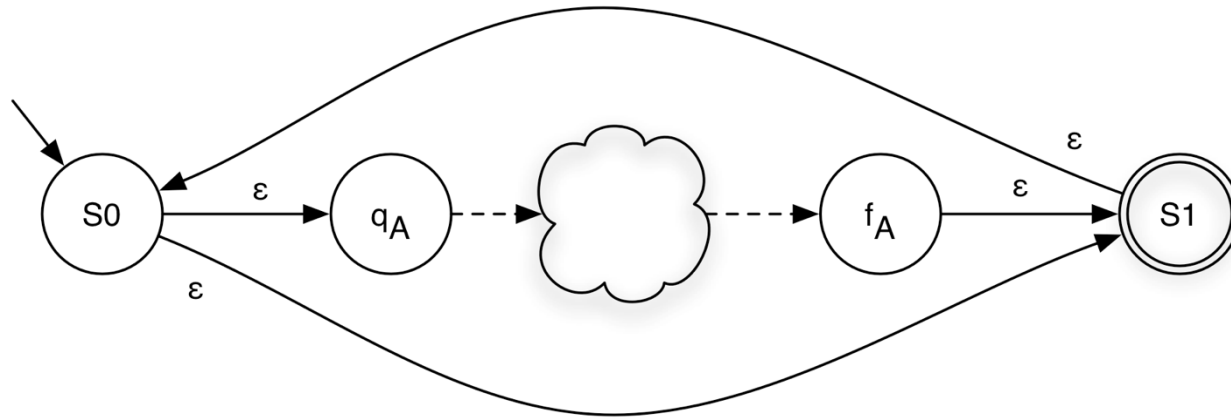
- ▶ Induction: A^*



- $\langle A \rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$

Reduction: Closure

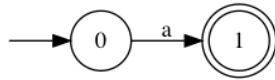
► Induction: A^*



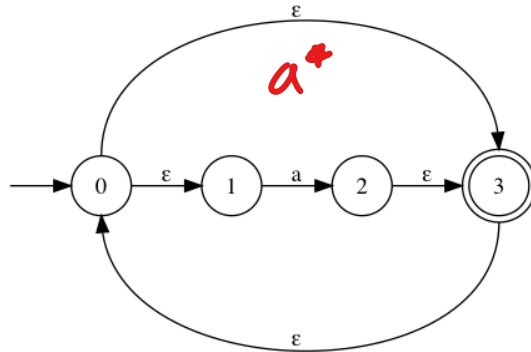
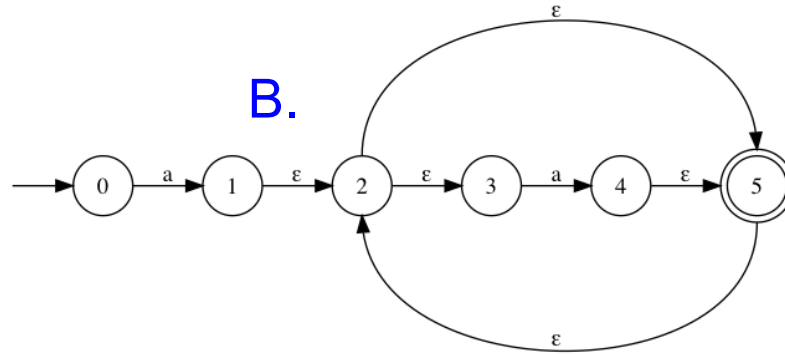
- $\langle A \rangle = (\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
- $\langle A^* \rangle = (\Sigma_A, Q_A \cup \{S0, S1\}, S0, \{S1\}, \delta_A \cup \{(f_A, \epsilon, S1), (S0, \epsilon, q_A), (S0, \epsilon, S1), (S1, \epsilon, S0)\})$

Quiz 2: Which NFA matches a^* ?

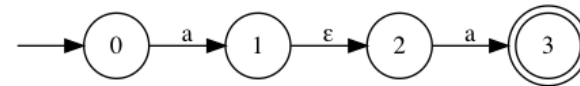
A.



B.

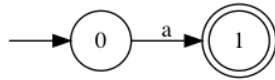


D.

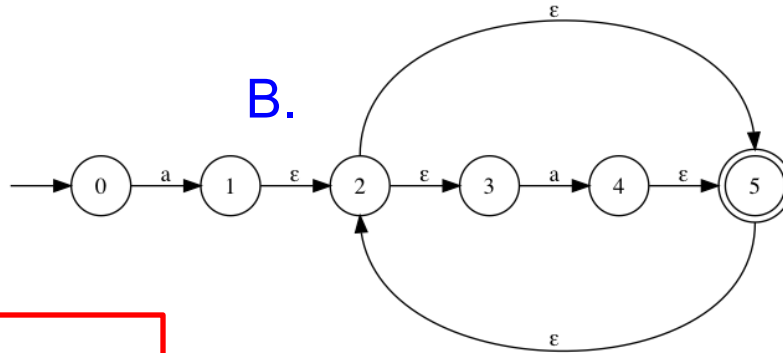


Quiz 2: Which NFA matches a^* ?

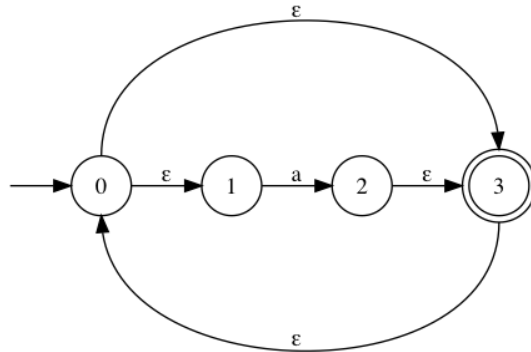
A.



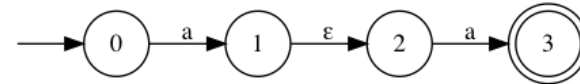
B.



C.

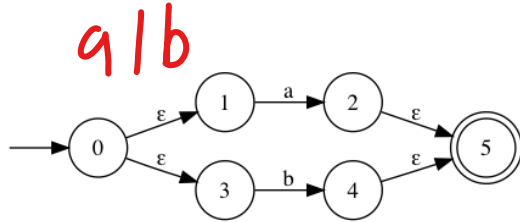


D.

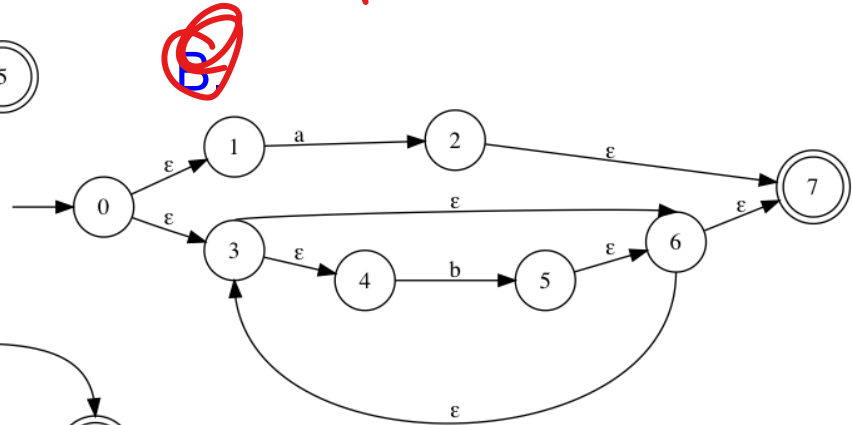


Quiz 3: Which NFA matches $a|b^*$?

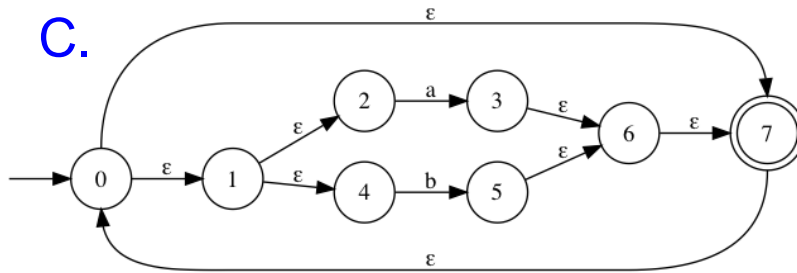
A.



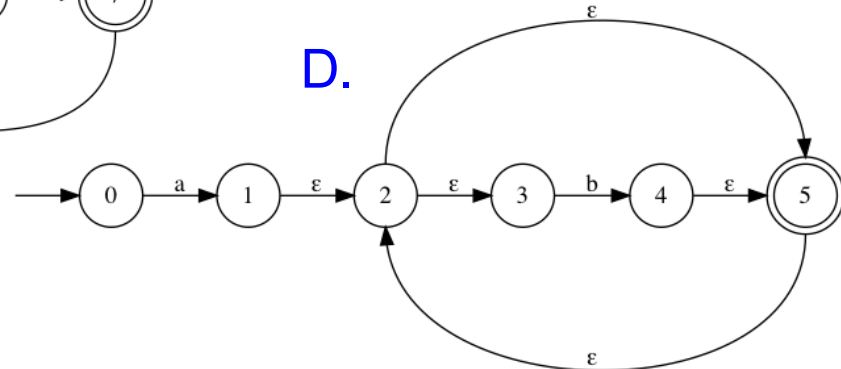
a



C.

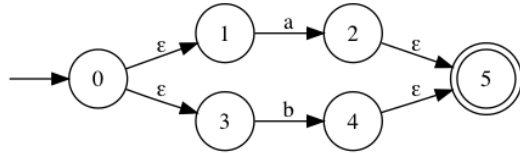


D.

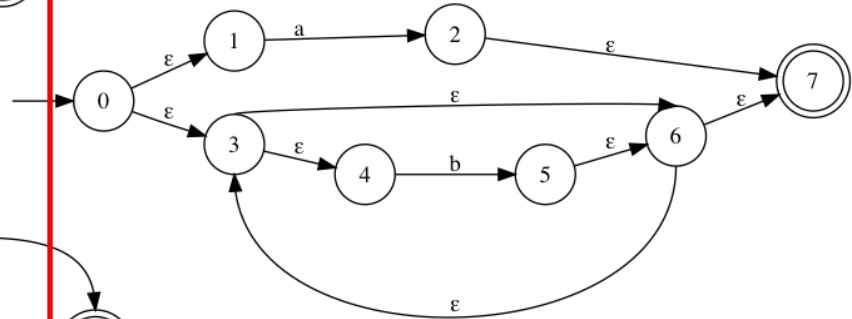


Quiz 3: Which NFA matches $a|b^*$?

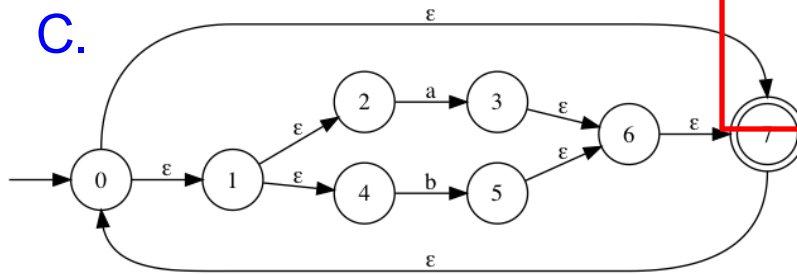
A.



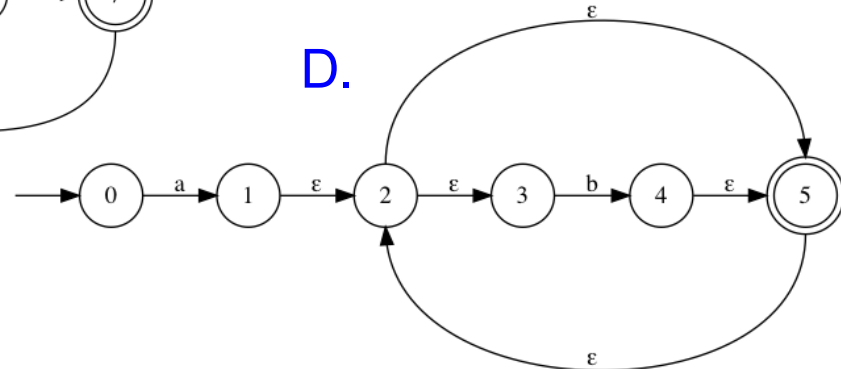
B.



C.

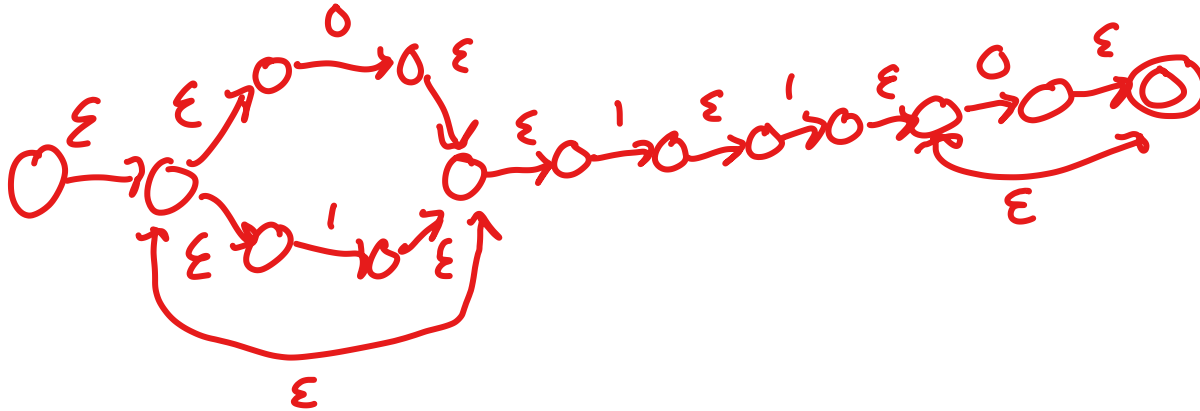


D.



RE \rightarrow NFA

Draw NFAs for the regular expression $(0|1)^*110^*$



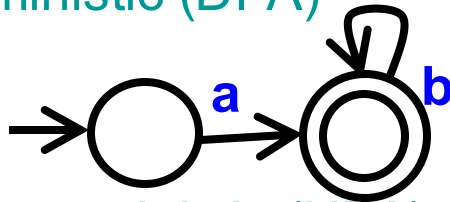
Recap

► Finite automata

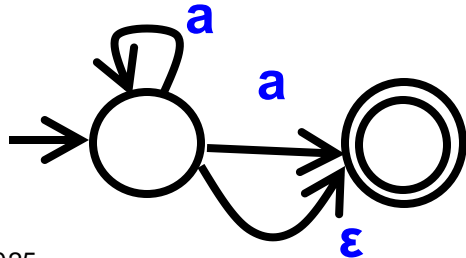
- Alphabet, states...
- $(\Sigma, Q, q_0, F, \delta)$

► Types

- Deterministic (DFA)

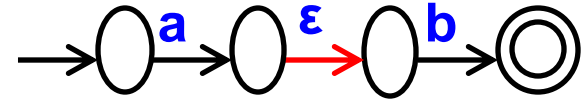


- Non-deterministic (NFA)

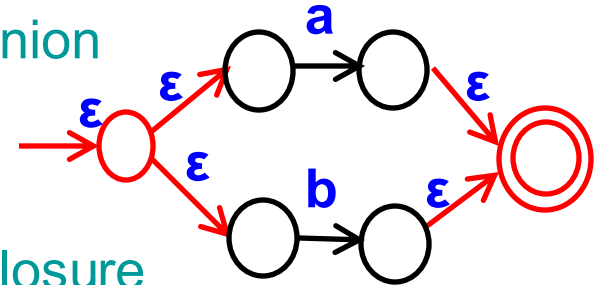


► Reducing RE to NFA

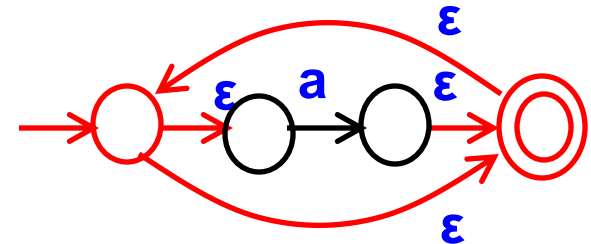
- Concatenation



- Union



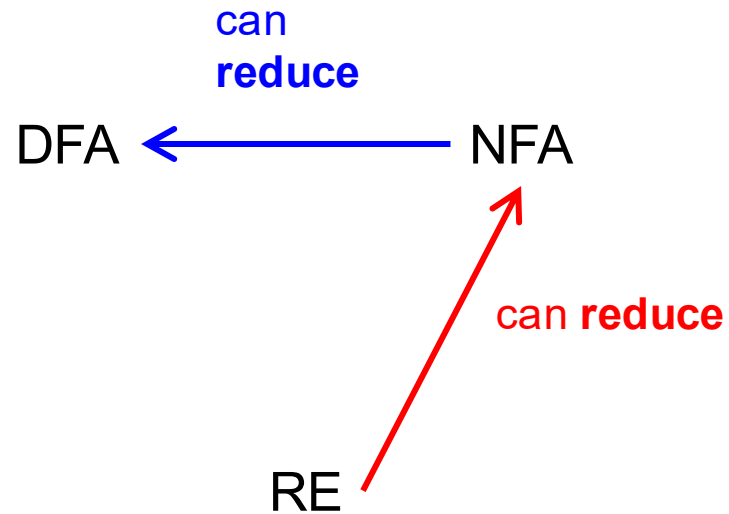
- Closure



Reduction Complexity

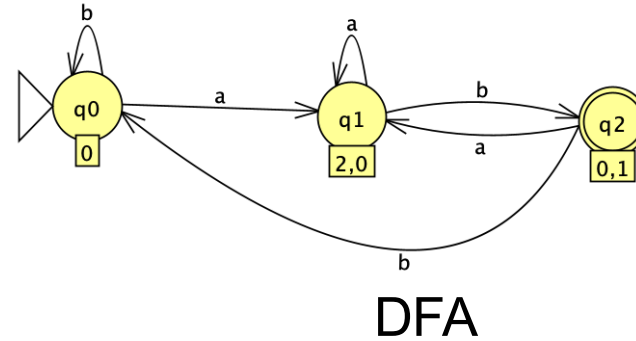
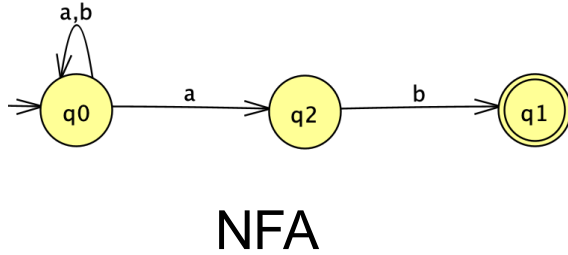
- ▶ Given a regular expression A of size n ...
Size = # of symbols + # of operations
- ▶ How many states does $\langle A \rangle$ have?
 - Two added for each $|$, two added for each $*$
 - $O(n)$
 - That's pretty good!

Reducing NFA to DFA



Why NFA \rightarrow DFA

- ▶ DFA is generally more efficient than NFA



Language: $(a|b)^*ab$

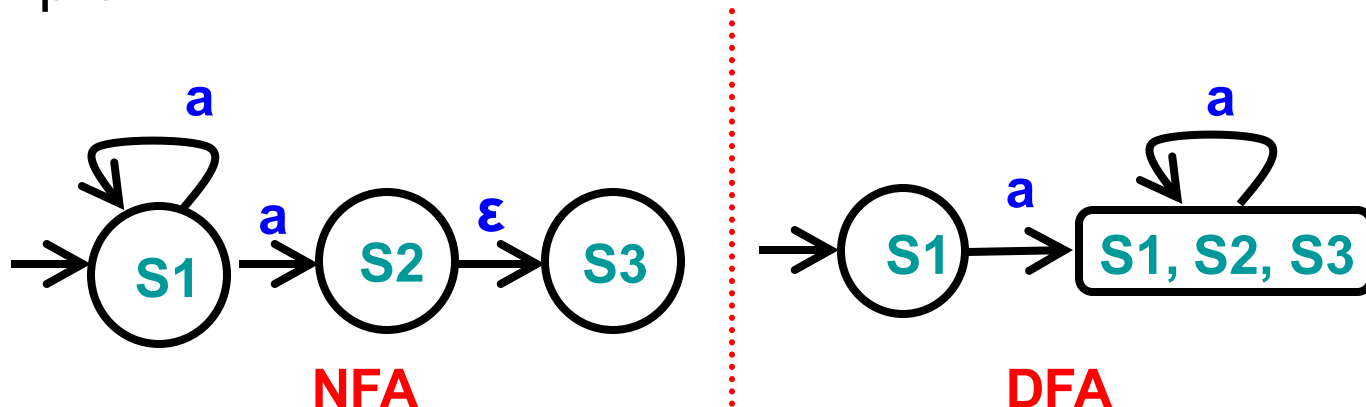
Why NFA \rightarrow DFA

- ▶ DFA has the same expressive power as NFAs.
 - Let language $L \subseteq \Sigma^*$, and suppose L is accepted by NFA $N = (\Sigma, Q, q_0, F, \delta)$. There exists a DFA $D = (\Sigma, Q', q'_0, F', \delta')$ that also accepts L . ($L(N) = L(D)$)
- ▶ NFAs are more flexible and easier to build. But DFAs have no less power than NFAs.

NFA \leftrightarrow DFA

Reducing NFA to DFA

- ▶ NFA may be reduced to DFA
 - By explicitly tracking the set of NFA states
- ▶ Intuition
 - Build DFA where
 - Each DFA state represents a set of NFA “current states”
- ▶ Example



Algorithm for Reducing NFA to DFA

- ▶ Reduction applied using the **subset** algorithm
 - DFA state is a subset of set of all NFA states
- ▶ Algorithm
 - Input
 - NFA $(\Sigma, Q, q_0, F_n, \delta)$
 - Output
 - DFA $(\Sigma, R, r_0, F_d, \delta)$
 - Using two subroutines
 - ϵ -closure(δ, p) (and ϵ -closure(δ, Q))
 - move(δ, p, σ) (and move(δ, Q, σ))
 - (where p is an NFA state)

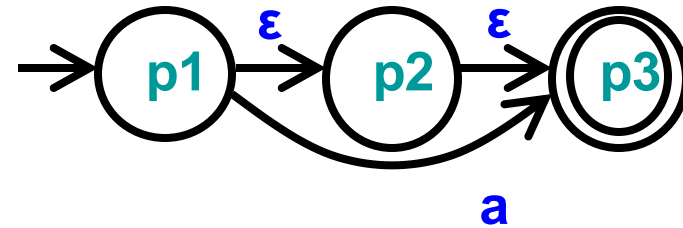
ϵ -transitions and ϵ -closure

- ▶ We say $p \xrightarrow{\epsilon} q$
 - If it is possible to go from state p to state q by taking only ϵ -transitions in δ
 - If $\exists p, p_1, p_2, \dots, p_n, q \in Q$ such that
 - $\{p, \epsilon, p_1\} \in \delta, \{p_1, \epsilon, p_2\} \in \delta, \dots, \{p_n, \epsilon, q\} \in \delta$
- ▶ ϵ -closure(δ, p)
 - Set of states reachable from p using ϵ -transitions alone
 - Set of states q such that $p \xrightarrow{\epsilon} q$ according to δ
 - ϵ -closure(δ, p) = $\{q \mid p \xrightarrow{\epsilon} q \text{ in } \delta\}$
 - ϵ -closure(δ, Q) = $\{q \mid p \in Q, p \xrightarrow{\epsilon} q \text{ in } \delta\}$
 - Notes
 - ϵ -closure(δ, p) always includes p

ϵ -closure: Example 1

► Following NFA contains

- $p1 \xrightarrow{\epsilon} p2$
- $p2 \xrightarrow{\epsilon} p3$
- $p1 \xrightarrow{\epsilon} p3$
 - Since $p1 \xrightarrow{\epsilon} p2$ and $p2 \xrightarrow{\epsilon} p3$



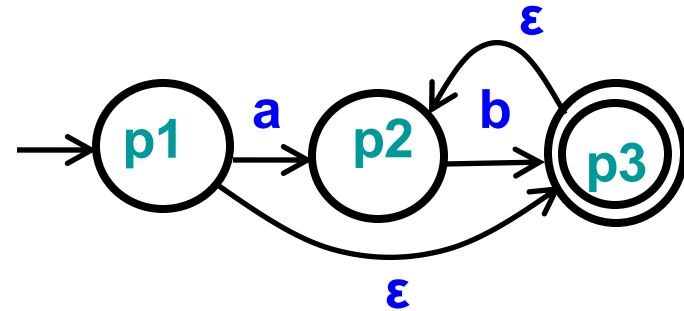
► ϵ -closures

- $\epsilon\text{-closure}(p1) = \{ p1, p2, p3 \}$
- $\epsilon\text{-closure}(p2) = \{ p2, p3 \}$
- $\epsilon\text{-closure}(p3) = \{ p3 \}$
- $\epsilon\text{-closure}(\{ p1, p2 \}) = \{ p1, p2, p3 \} \cup \{ p2, p3 \}$

ϵ -closure: Example 2

► Following NFA contains

- $p1 \xrightarrow{\epsilon} p3$
- $p3 \xrightarrow{\epsilon} p2$
- $p1 \xrightarrow{\epsilon} p2$



► ϵ -closures

- ϵ -closure($p1$) = $\{ p1, p2, p3 \}$
- ϵ -closure($p2$) = $\{ p2 \}$
- ϵ -closure($p3$) = $\{ p2, p3 \}$
- ϵ -closure($\{ p2, p3 \}$) = $\{ p2 \} \cup \{ p2, p3 \}$

ϵ -closure Algorithm: Approach

► Input: NFA $(\Sigma, Q, q_0, F_n, \delta)$, State Set R

► Output: State Set R'

► Algorithm

Let $R' = R$

// start states

Repeat

Let $R = R'$

// continue from previous

Let $R' = R \cup \{q \mid p \in R, (p, \epsilon, q) \in \delta\}$

// new ϵ -reachable states

Until $R = R'$

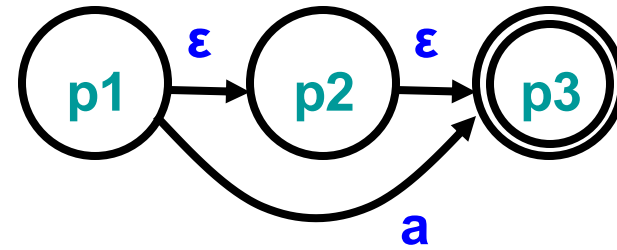
// stop when no new states

This algorithm computes a **fixed point**

ϵ -closure Algorithm Example

► Calculate $\epsilon\text{-closure}(\delta, \{p1\})$

R	R'
$\{p1\}$	$\{p1\}$
$\{p1\}$	$\{p1, p2\}$
$\{p1, p2\}$	$\{p1, p2, p3\}$
$\{p1, p2, p3\}$	$\{p1, p2, p3\}$



Let $R' = R$
Repeat
 Let $R = R'$
 Let $R' = R \cup \{q \mid p \in R, (p, \epsilon, q) \in \delta\}$
Until $R = R'$

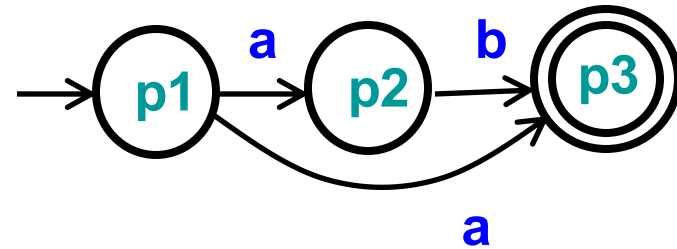
Calculating $\text{move}(p, \sigma)$

- ▶ $\text{move}(\delta, p, \sigma)$
 - Set of states reachable from p using exactly one transition on symbol σ
 - Set of states q such that $\{p, \sigma, q\} \in \delta$
 - $\text{move}(\delta, p, \sigma) = \{ q \mid \{p, \sigma, q\} \in \delta \}$
 - $\text{move}(\delta, Q, \sigma) = \{ q \mid p \in Q, \{p, \sigma, q\} \in \delta \}$
 - i.e., can “lift” $\text{move}()$ to a set of states Q
 - Notes:
 - $\text{move}(\delta, p, \sigma)$ is \emptyset if no transition $(p, \sigma, q) \in \delta$, for any q

move(p,σ) : Example 1

► Following NFA

- $\Sigma = \{ a, b \}$



► Move

- $\text{move}(p1, a) = \{ p2, p3 \}$
- $\text{move}(p1, b) = \emptyset$
- $\text{move}(p2, a) = \emptyset$
- $\text{move}(p2, b) = \{ p3 \}$
- $\text{move}(p3, a) = \emptyset$
- $\text{move}(p3, b) = \emptyset$

$$\text{move}(\{p1, p2\}, b) = \{ p3 \}$$

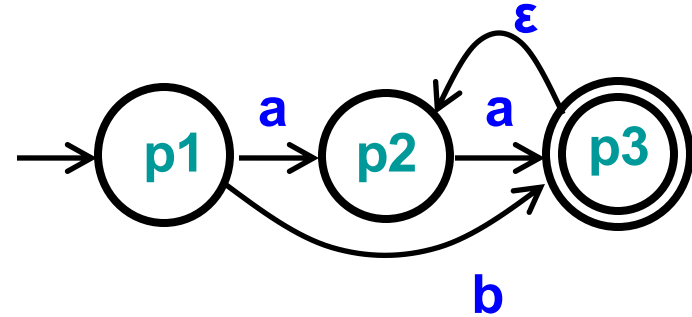
move(p,σ) : Example 2

- ▶ Following NFA

- $\Sigma = \{ a, b \}$

- ▶ Move

- $\text{move}(p1, a) = \{ p2 \}$
- $\text{move}(p1, b) = \{ p3 \}$
- $\text{move}(p2, a) = \{ p3 \}$
- $\text{move}(p2, b) = \emptyset$
- $\text{move}(p3, a) = \emptyset$
- $\text{move}(p3, b) = \emptyset$



$$\text{move}(\{p1, p2\}, a) = \{p2, p3\}$$

NFA \rightarrow DFA Reduction Algorithm (“subset”)

► Input NFA $(\Sigma, Q, q_0, F_n, \delta)$, Output DFA $(\Sigma, R, r_0, F_d, \delta')$

► Algorithm

Let $r_0 = \varepsilon\text{-closure}(\delta, q_0)$, add it to R

// DFA start state

While \exists an unmarked state $r \in R$

// process DFA state r

Mark r

// each state visited once

For each $\sigma \in \Sigma$

// for each symbol σ

Let $E = \text{move}(\delta, r, \sigma)$

// states reached via σ

Let $e = \varepsilon\text{-closure}(\delta, E)$

// states reached via ε

If $e \notin R$

// if state e is new

Let $R = R \cup \{e\}$

// add e to R (unmarked)

Let $\delta' = \delta' \cup \{r, \sigma, e\}$

// add transition $r \rightarrow e$ on σ

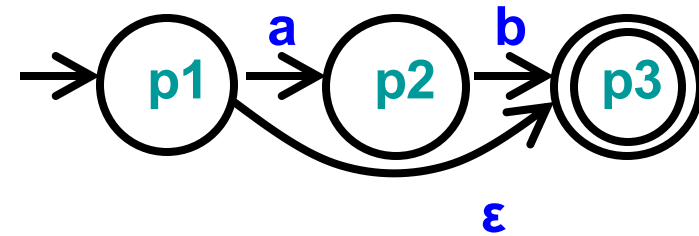
Let $F_d = \{r \mid \exists s \in r \text{ with } s \in F_n\}$

// final if include state in F_n

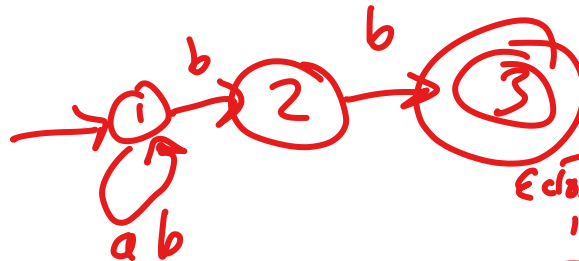
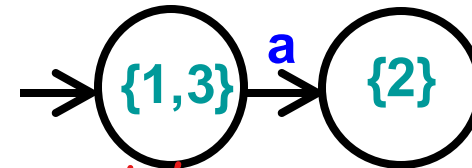
NFA \rightarrow DFA Example

- Start = ε -closure($\delta, p1$) = { {p1,p3} }
- $R = \{ \{p1,p3\} \}$
- $r \in R = \{p1,p3\}$
- $\text{move}(\delta, \{p1,p3\}, a) = \{p2\}$
 - $e = \varepsilon$ -closure($\delta, \{p2\}$) = {p2}
 - $R = R \cup \{\{p2\}\} = \{ \{p1,p3\}, \{p2\} \}$
 - $\delta' = \delta' \cup \{\{p1,p3\}, a, \{p2\}\}$
- $\text{move}(\delta, \{p1,p3\}, b) = \emptyset$

NFA



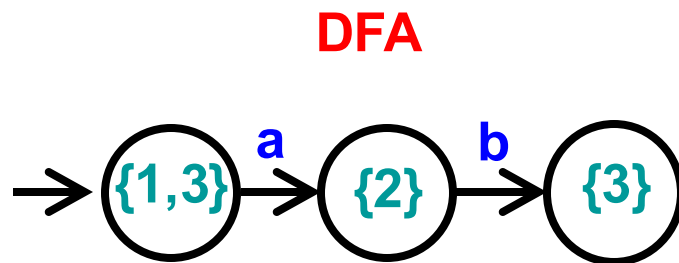
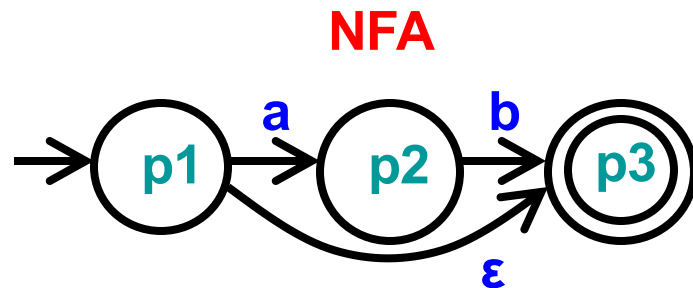
DFA



		a	b
ε cl.	1	1	1, 2
1, 2	1	1	1, 2, 3
1, 2, 3	1	1	1, 2, 3

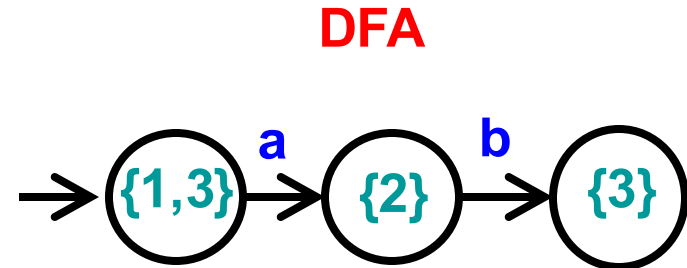
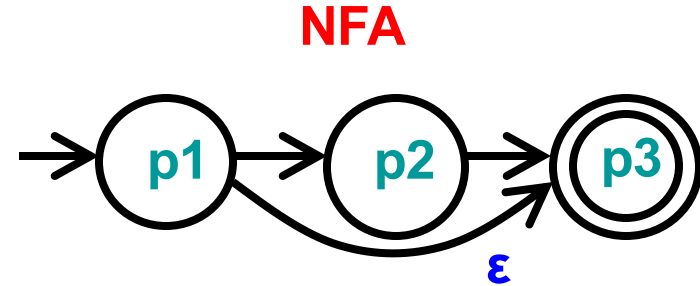
NFA \rightarrow DFA Example (cont.)

- $R = \{ \{p1,p3\}, \{p2\} \}$
- $r \in R = \{p2\}$
- $\text{move}(\delta, \{p2\}, a) = \emptyset$
- $\text{move}(\delta, \{p2\}, b) = \{p3\}$
 - $e = \varepsilon\text{-closure}(\delta, \{p3\}) = \{p3\}$
 - $R = R \cup \{\{p3\}\} = \{ \{p1,p3\}, \{p2\}, \{p3\} \}$
 - $\delta' = \delta' \cup \{\{p2\}, b, \{p3\}\}$



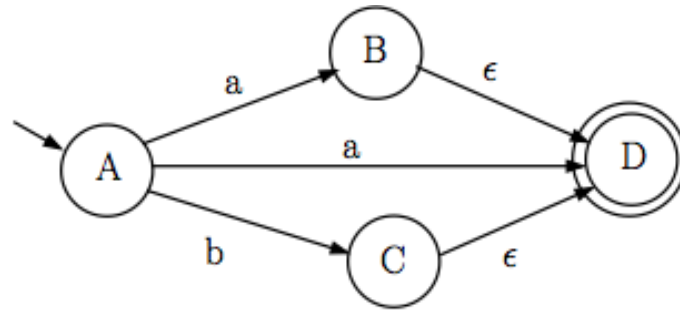
NFA \rightarrow DFA Example (cont.)

- $R = \{ \{p1,p3\}, \{p2\}, \{p3\} \}$
- $r \in R = \{p3\}$
- $\text{Move}(\{p3\}, a) = \emptyset$
- $\text{Move}(\{p3\}, b) = \emptyset$
- Mark $\{p3\}$, exit loop
- $F_d = \{ \{p1,p3\}, \{p3\} \}$
 - Since $p3 \in F_n$
- Done!

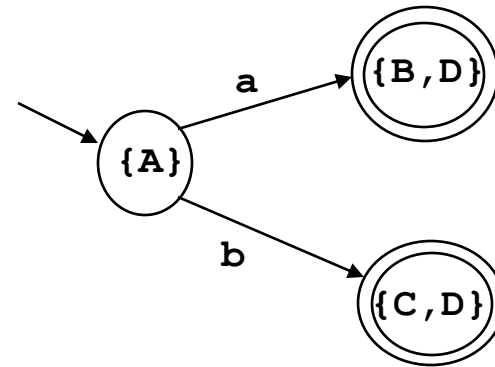


NFA \rightarrow DFA Example 2

► NFA



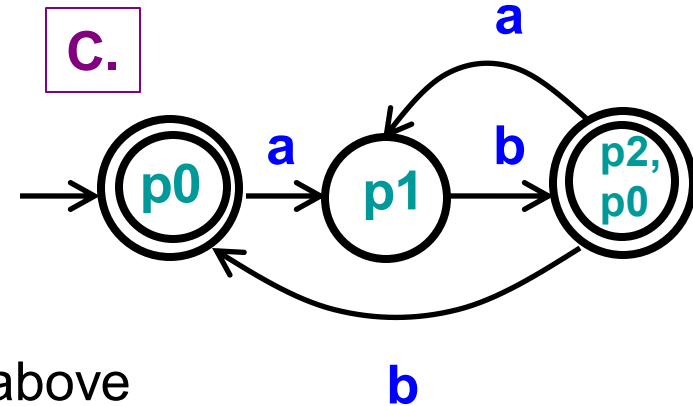
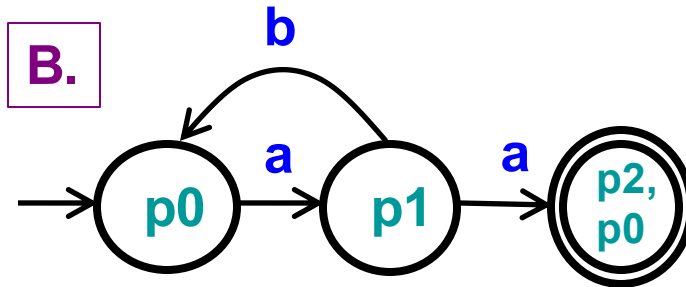
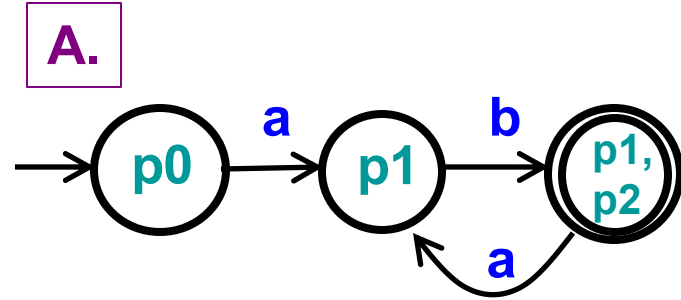
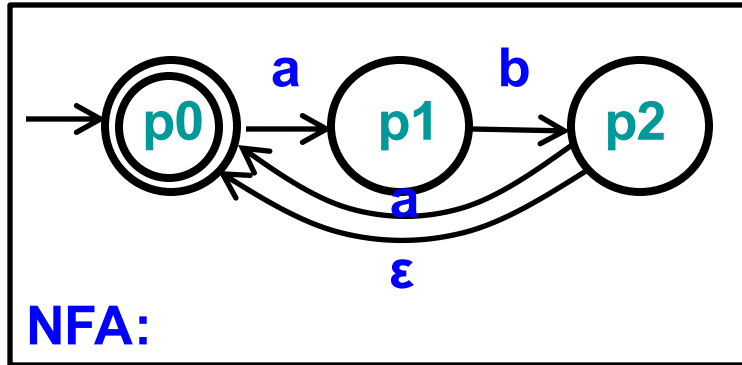
► DFA



	a	b	ε
p0	p0, p1	p0,	p0

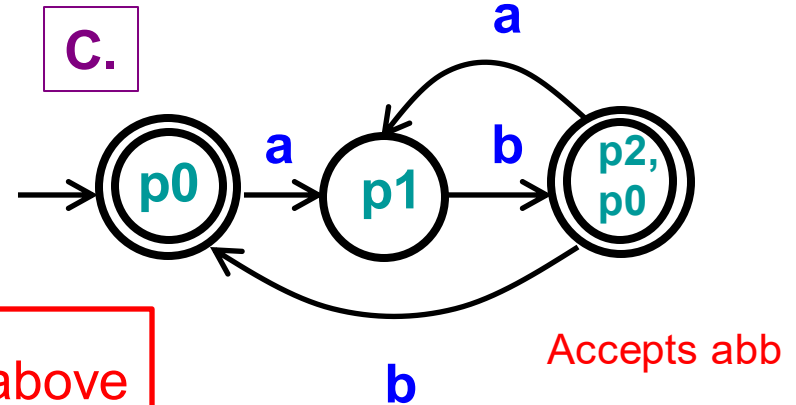
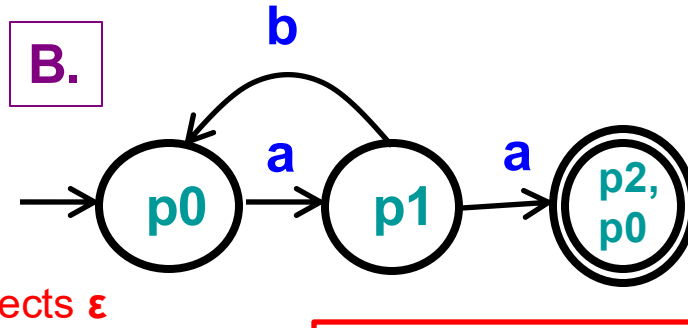
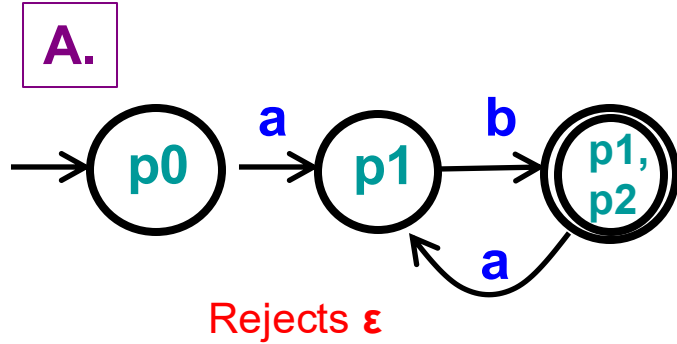
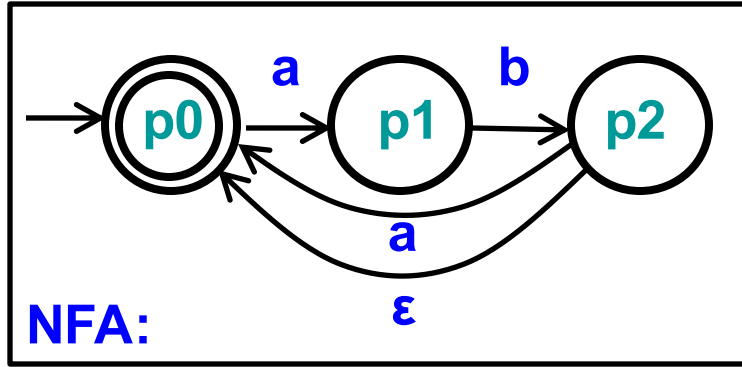
$$R = \{ \boxed{\{A\}}, \boxed{\{B,D\}}, \boxed{\{C,D\}} \}$$

Quiz 4: Which DFA is equiv to this NFA?



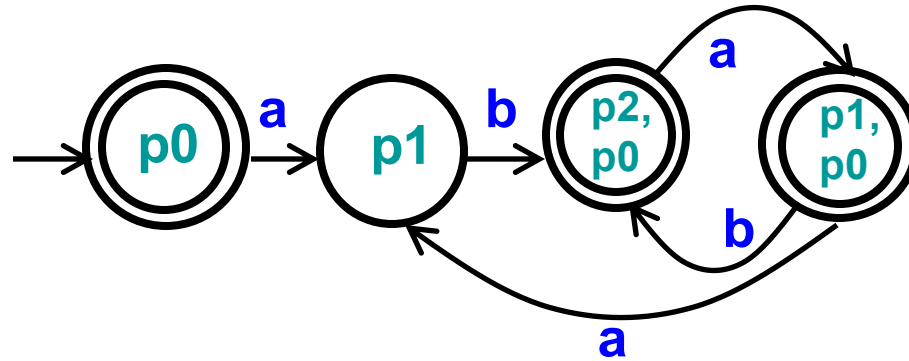
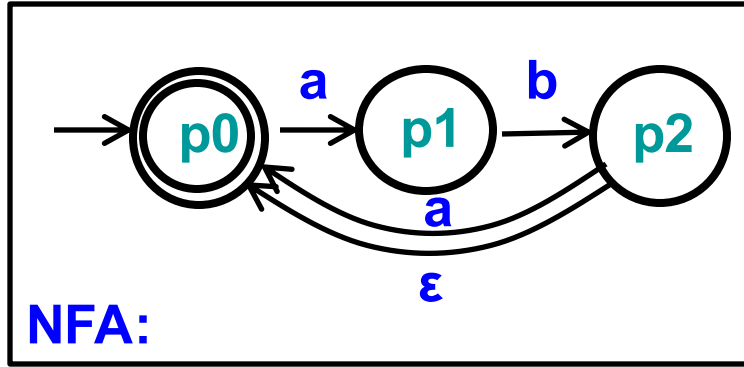
D. None of the above

Quiz 4: Which DFA is equiv to this NFA?



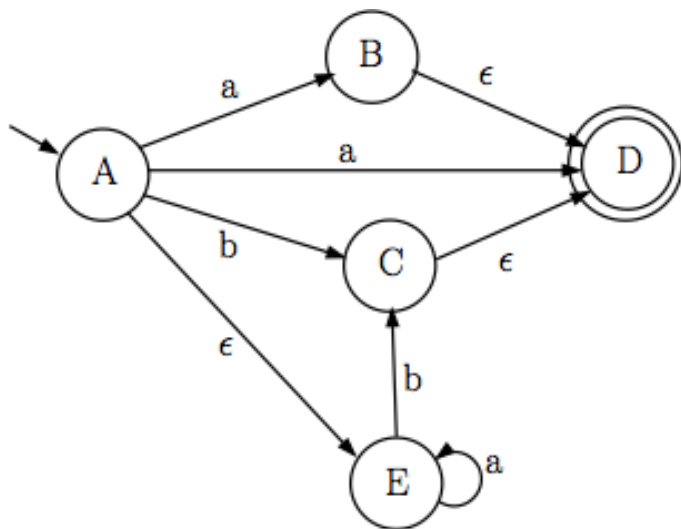
D. None of the above

Actual Answer

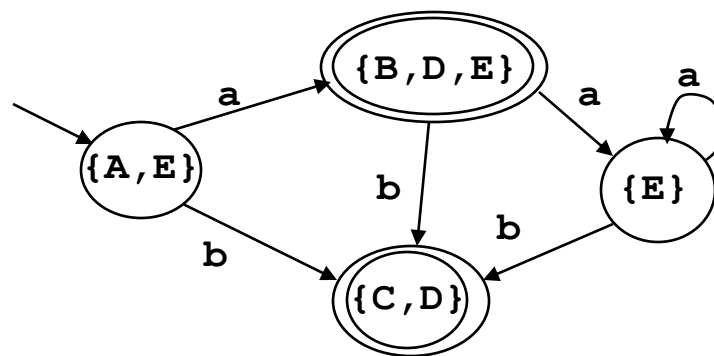


NFA \rightarrow DFA Example 3

► NFA



► DFA



$$R = \{ \boxed{\{A,E\}}, \boxed{\{B,D,E\}}, \boxed{\{C,D\}}, \boxed{\{E\}} \}$$