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

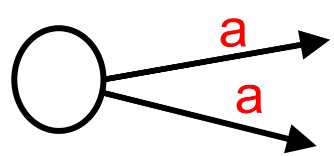
Two Types of Finite Automata

- 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

Comparing DFAs and NFAs

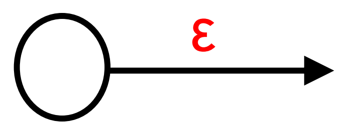
- NFAs can have more than one transition leaving a state on the same symbol



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

Comparing DFAs and NFAs (cont.)

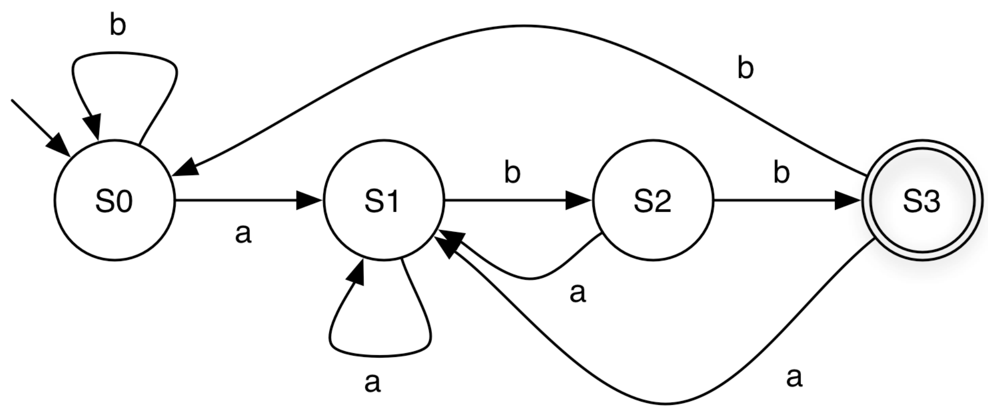
- NFAs may have transitions with empty string label
  - May move to new state without consuming character



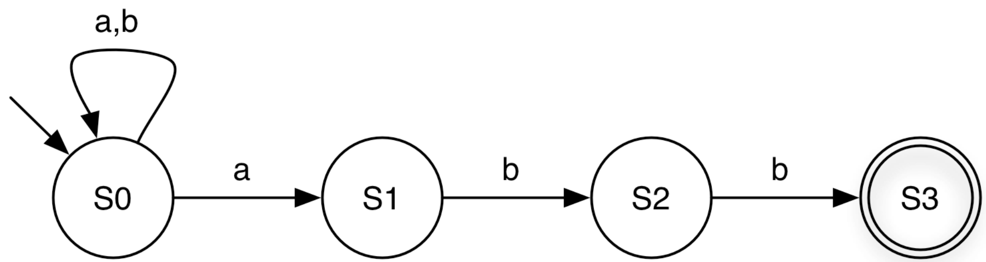
ε-transition

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

DFA for (a|b)*abb

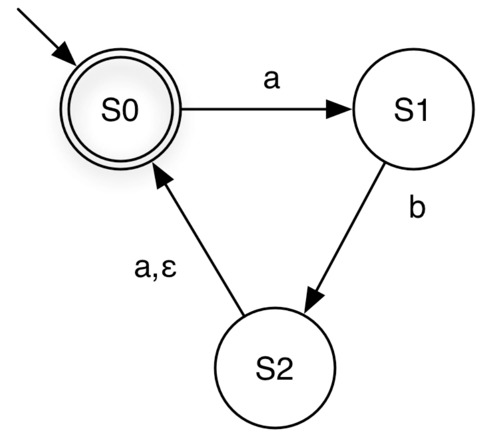


NFA for (a|b)*abb



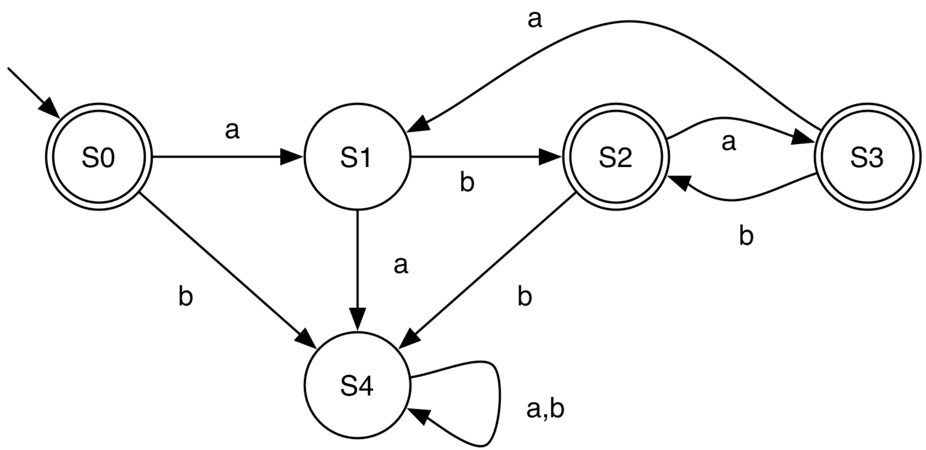
- ba
  - Has paths to either S0 or S1
  - Neither is final, so rejected
- babaabb
  - Has paths to different states
  - One path leads to S3, so accepts string

NFA for (ab|aba)*

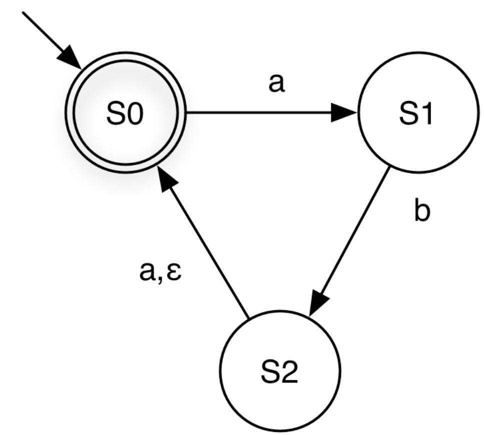


- aba
- ababa
  - Has paths to states S0, S1
  - Need to use ε-transition

NFA and DFA for (ab|aba)*



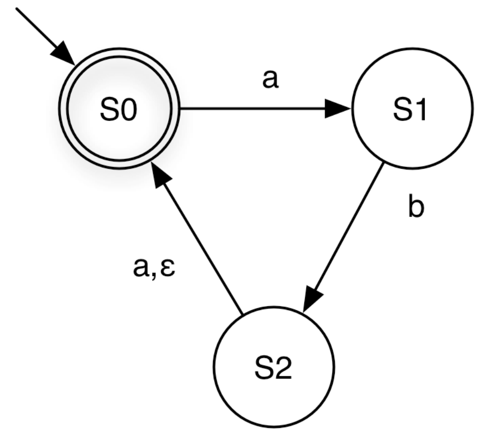
DFA



NFA

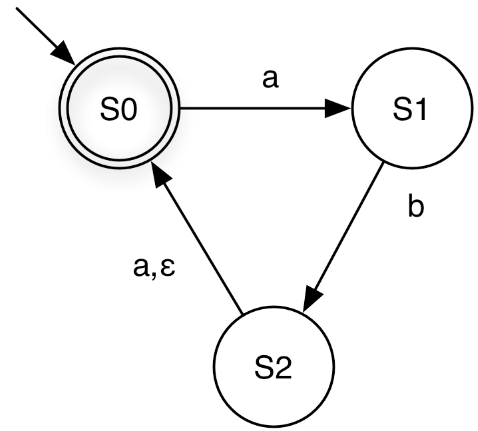
Quiz 1: Which string is NOT accepted by this NFA?

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

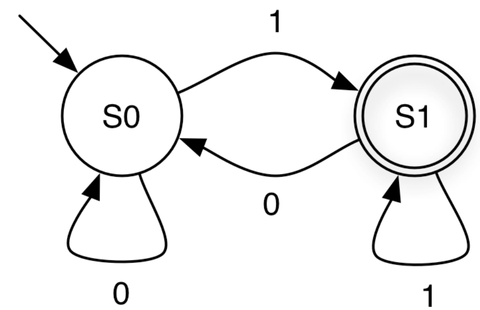


Formal Definition

- 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

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



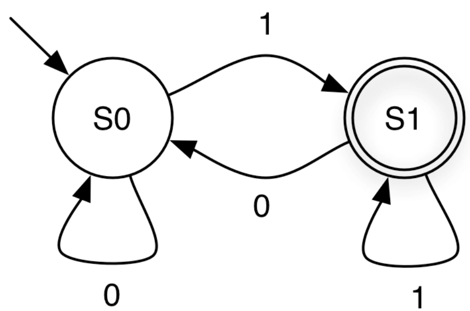
symbol

input state	0	1
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)

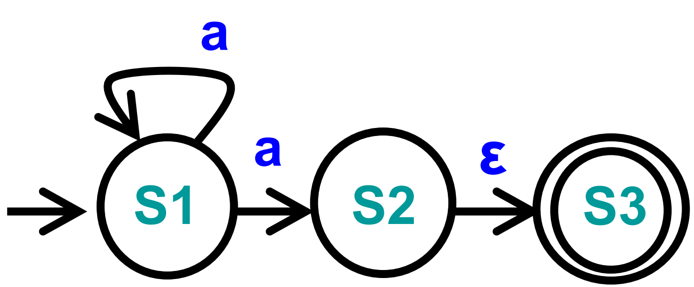
More generally, use generic table-driven DFA

```
given components (Σ, Q, q0, F, δ) of a DFA:
let q = q0
while (there exists another symbol σ of the input string)
    q := δ(q, σ);
if q ∈ 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, $q0$, F as with DFAs
  - $\delta \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ specifies the NFA's transitions



Example

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

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

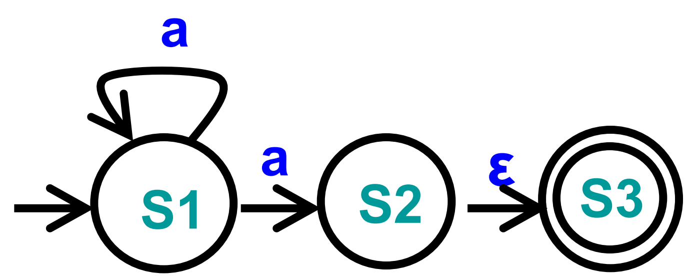
NFA Acceptance Algorithm (Sketch)

- 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

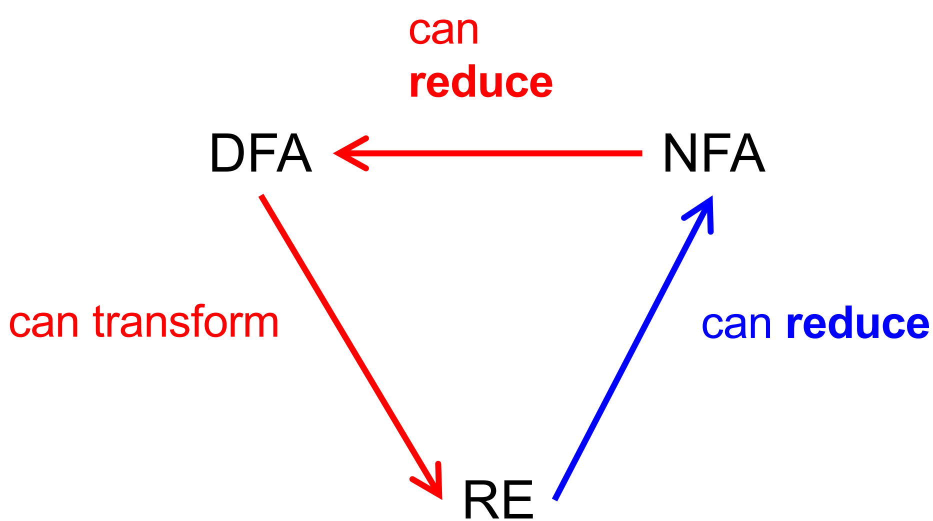
      S3
    - Since S3 is final, s is accepted



- 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*



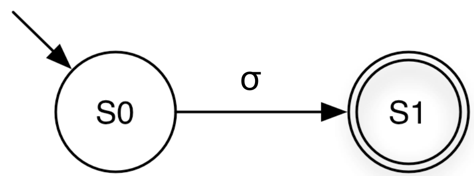
Reducing Regular Expressions to NFAs

- 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: σ , ε , $\emptyset$
  - 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

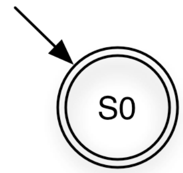


$$\langle\sigma\rangle = (\{\sigma\}, \{S0, S1\}, S0, \{S1\}, \{(S0, \sigma, S1)\})$$

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

Reduction

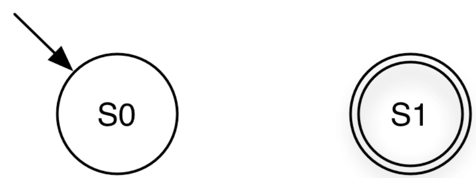
- Base case: ε



<ε> = (∅, {S0}, S0, {S0}, ∅)

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

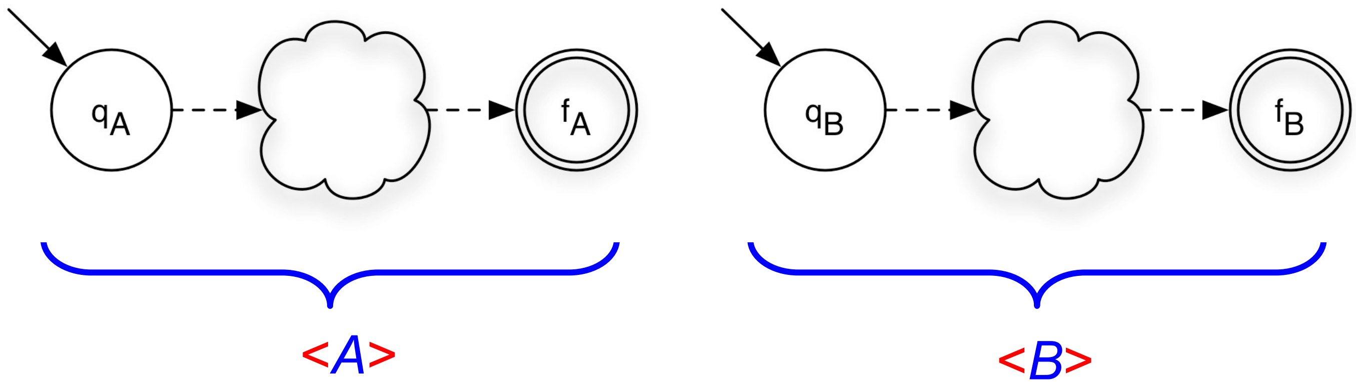
- Base case: ∅



<∅> = (∅, {S0, S1}, S0, {S1}, ∅)

Reduction: Concatenation

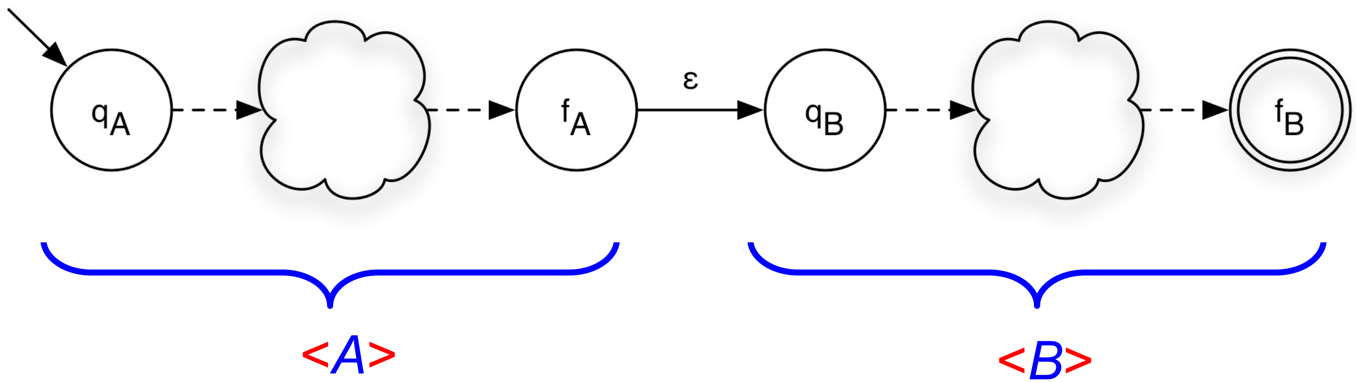
- Induction: *AB*



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

Reduction: Concatenation

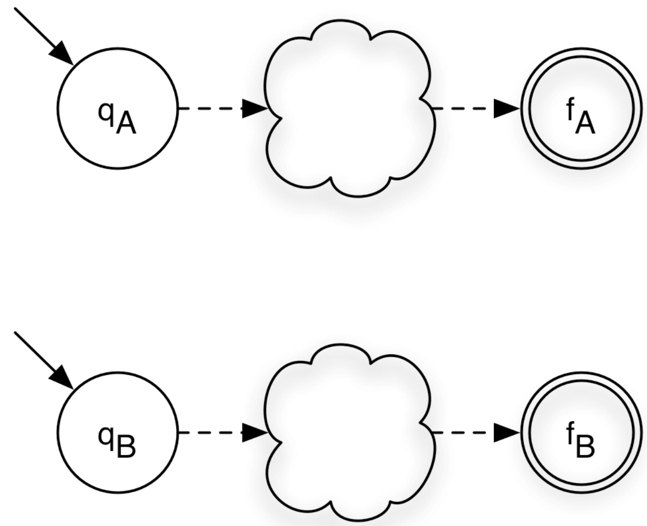
- Induction: *AB*



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

Reduction: Union

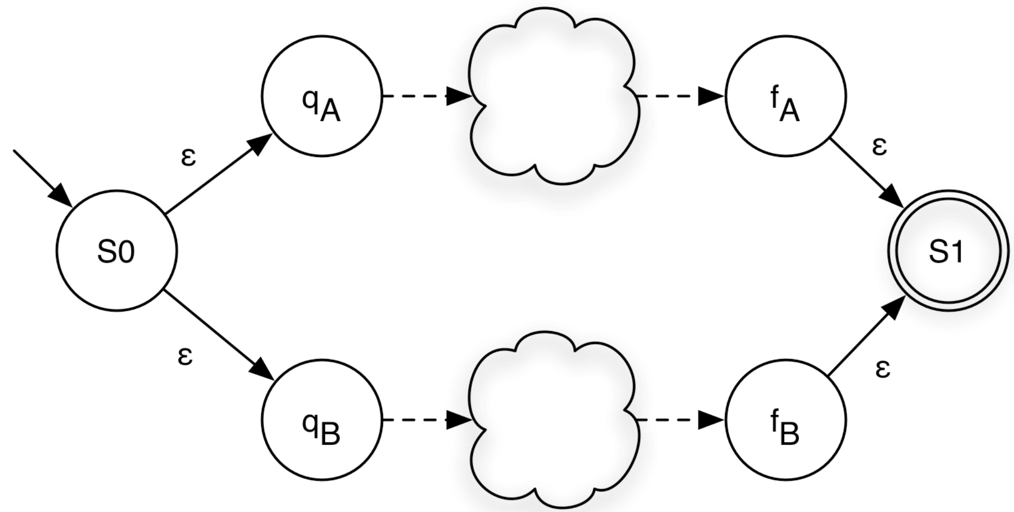
- Induction: *A*|*B*



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

Reduction: Union

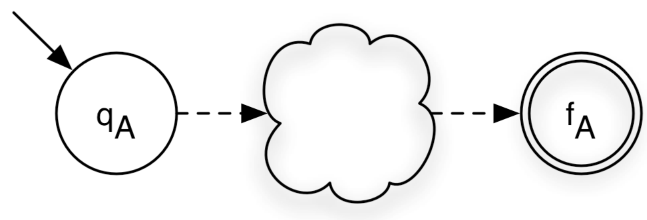
- Induction: *A*|*B*



  - <*A*> = $(\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
  - <*B*> = $(\Sigma_B, Q_B, q_B, \{f_B\}, \delta_B)$
  - <*A*|*B*> = $(\Sigma_A \cup \Sigma_B, Q_A \cup Q_B \cup \{S0,S1\}, S0, \{S1\},$
    $\delta_A \cup \delta_B \cup \{(S0,\varepsilon,q_A), (S0,\varepsilon,q_B), (f_A,\varepsilon,S1), (f_B,\varepsilon,S1)\})$

Reduction: Closure

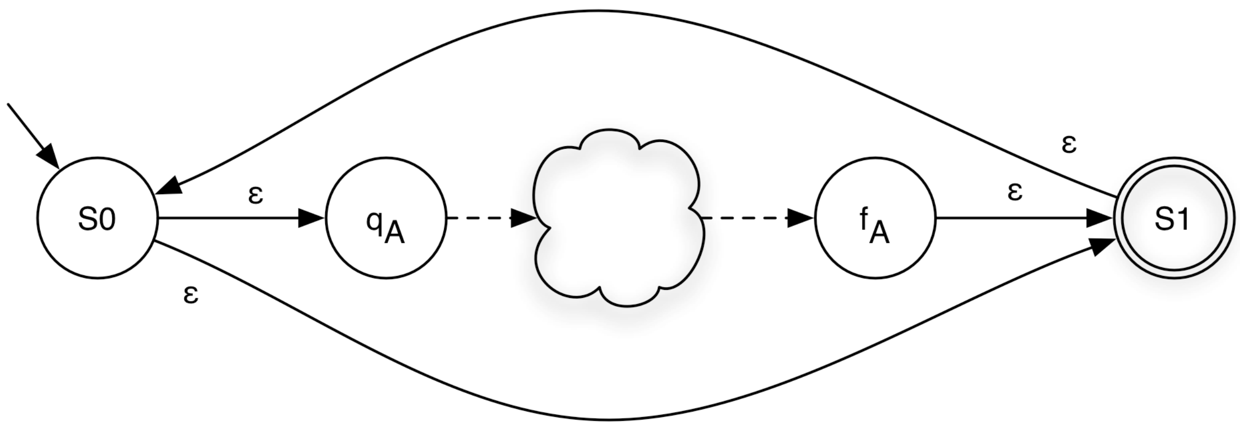
- Induction: *A**



- <*A*> = $(\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$

Reduction: Closure

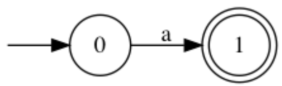
- Induction: *A**



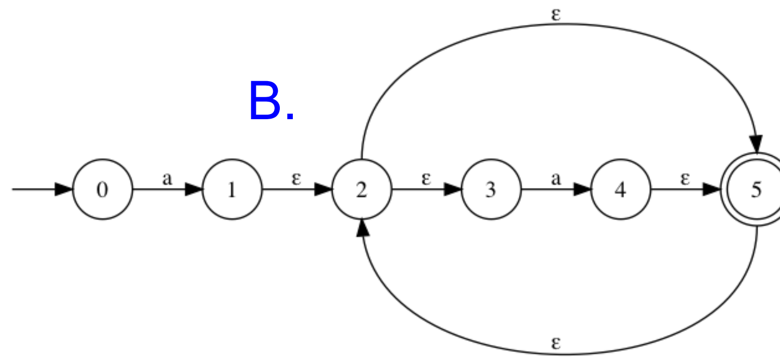
  - <*A*> = $(\Sigma_A, Q_A, q_A, \{f_A\}, \delta_A)$
  - <*A**> = $(\Sigma_A, Q_A \cup \{S0,S1\}, S0, \{S1\},$
    $\delta_A \cup \{(f_A,\varepsilon,S1), (S0,\varepsilon,q_A), (S0,\varepsilon,S1), (S1,\varepsilon,S0)\})$

Quiz 2: Which NFA matches **a*** ?

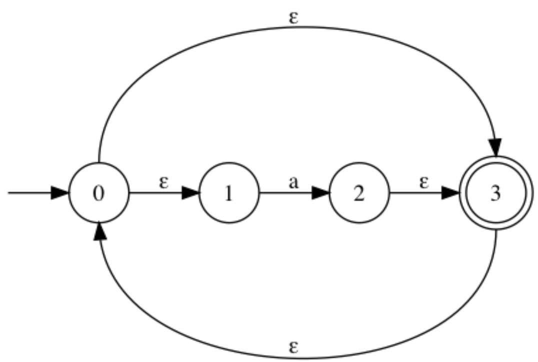
A.



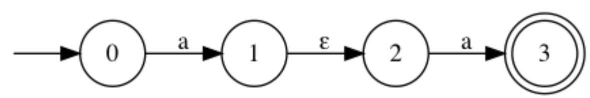
B.



C.

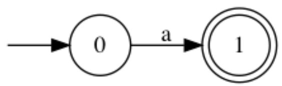


D.

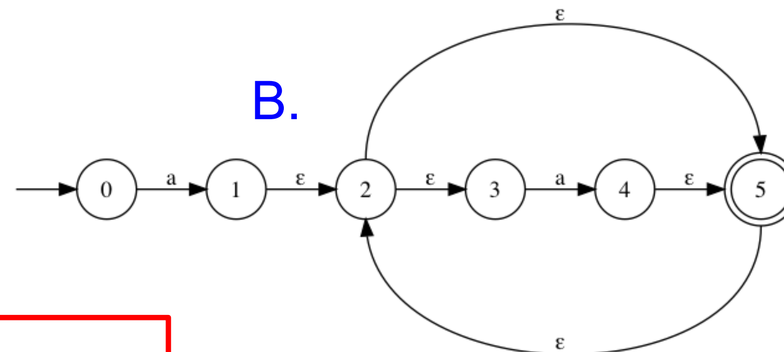


Quiz 2: Which NFA matches **a*** ?

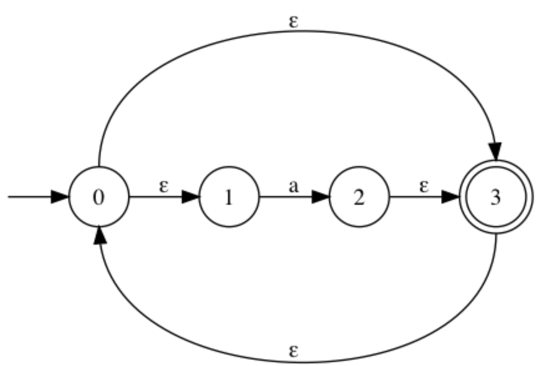
A.



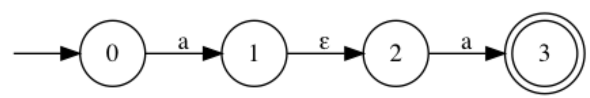
B.



C.

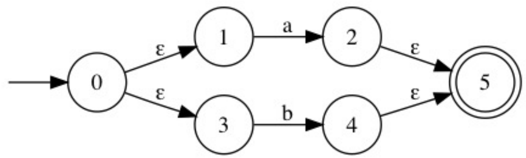


D.

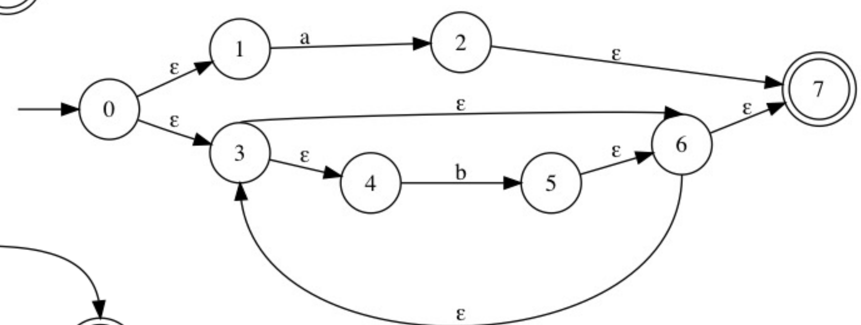


Quiz 3: Which NFA matches **a|b*** ?

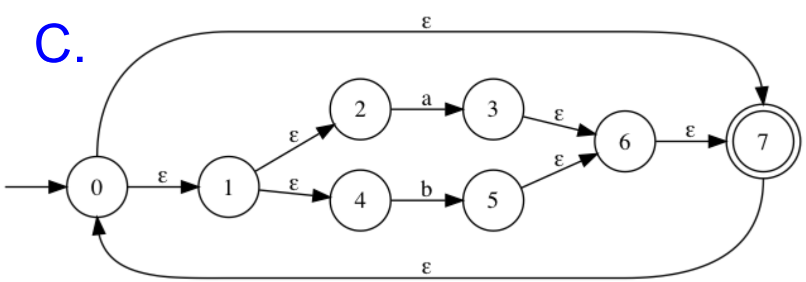
A.



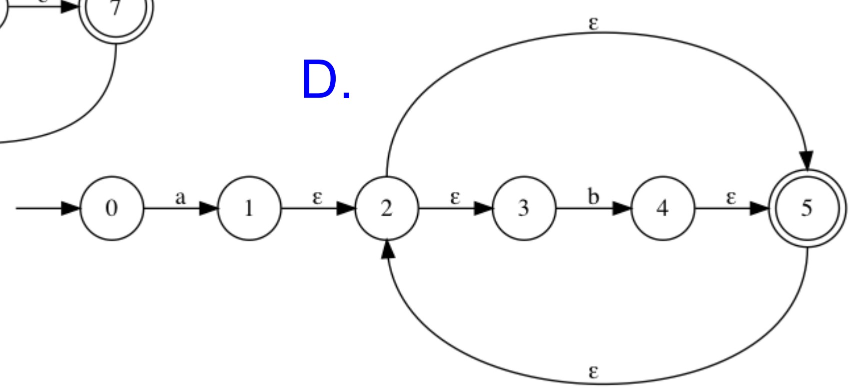
B.



C.

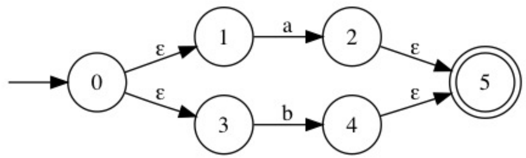


D.

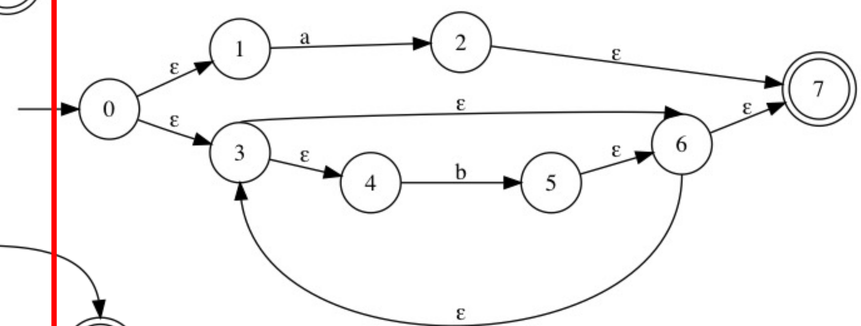


Quiz 3: Which NFA matches **a|b*** ?

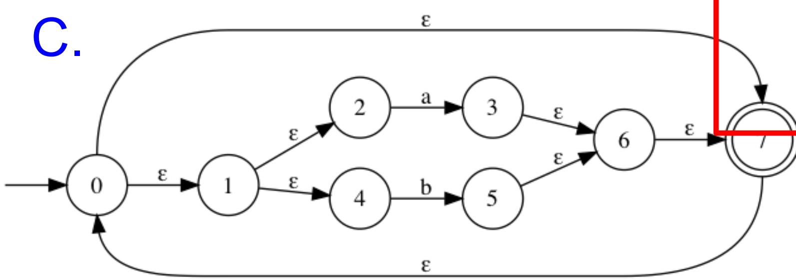
A.



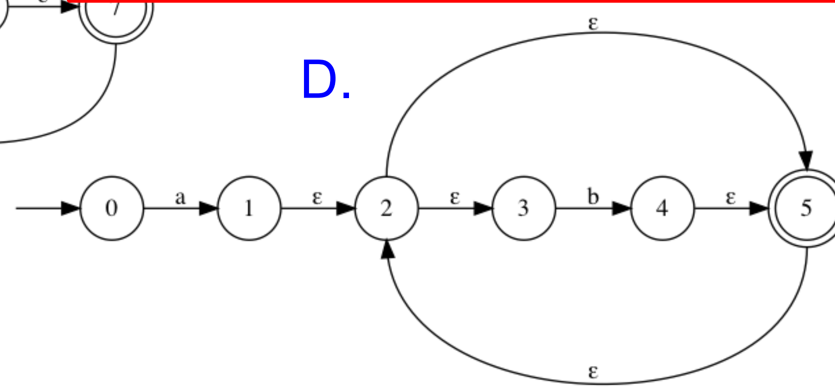
B.



C.



D.

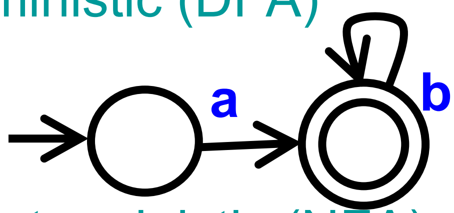


RE → NFA

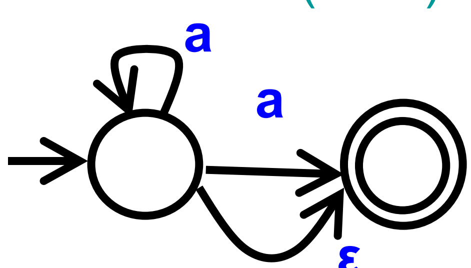
Draw NFAs for the regular expression (0|1)*110*

Recap

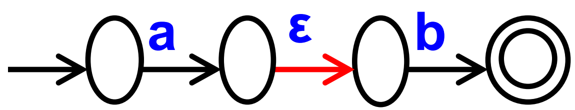
- Finite automata
  - Alphabet, states…
  - (Σ, Q, q_0, F, δ)
- Types
  - Deterministic (DFA)



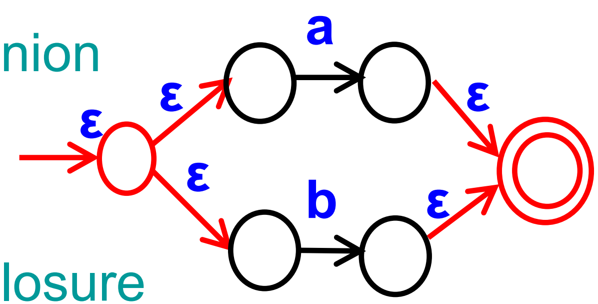
  - Non-deterministic (NFA)



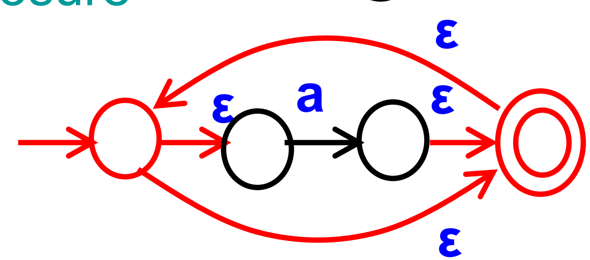
- Reducing RE to NFA
  - Concatenation



  - Union



  - Closure



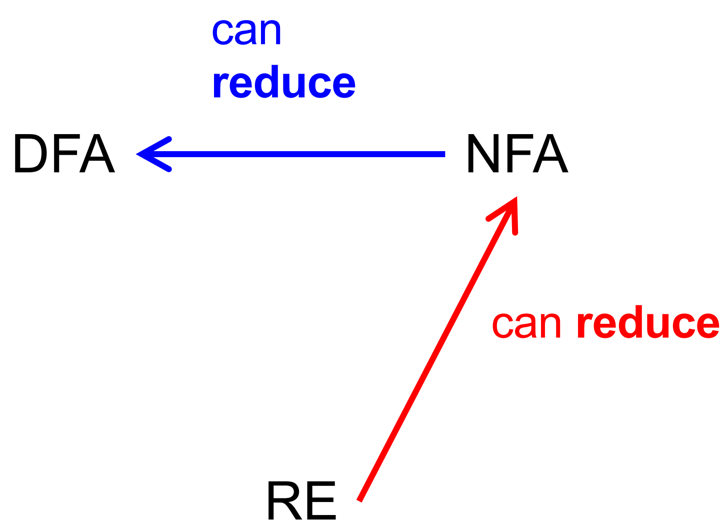
Reduction Complexity

- Given a regular expression *A* of size n...

  Size = # of symbols + # of operations

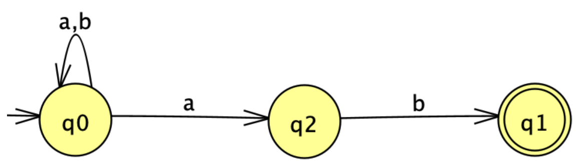
- How many states does <*A*> have?
  - Two added for each **|**, two added for each *
  - O(n)
  - That’s pretty good!

Reducing NFA to DFA

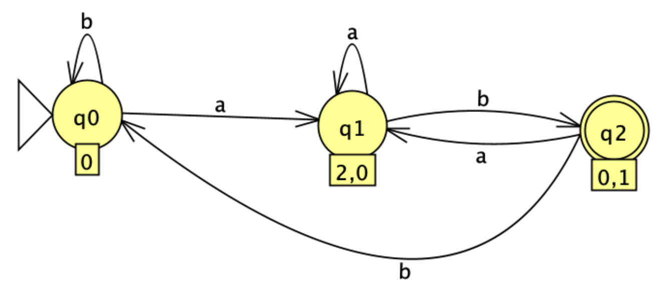


Why NFA → DFA

- DFA is generally more efficient than NFA



NFA



DFA

Language: (a|b)*ab

Why NFA → DFA

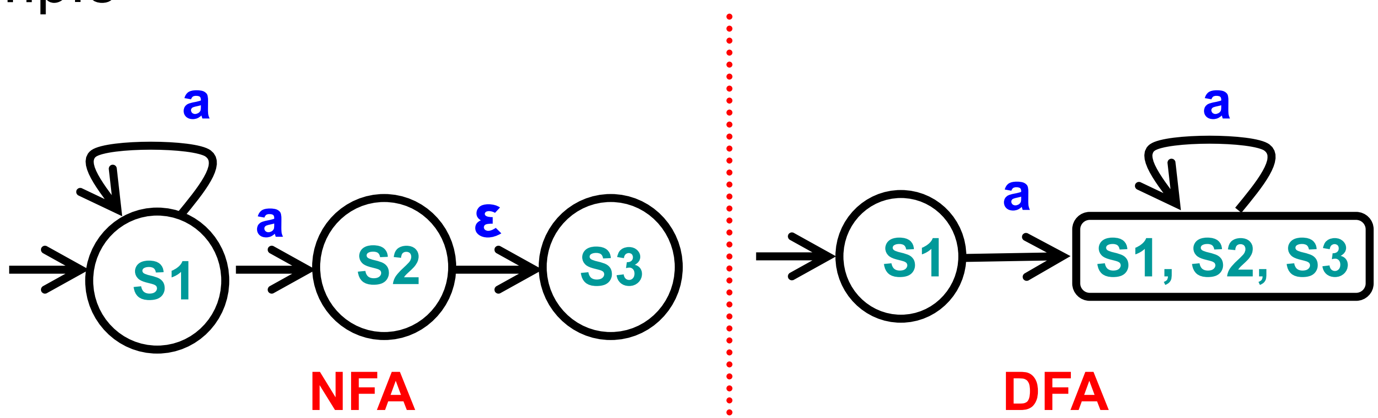
- DFA has the same expressive power as NFAs.
  - Let language $L \subseteq \Sigma^*$, and suppose L is accepted by NFA $N = (\Sigma, Q, q_0, F, \delta)$. There exists a DFA $D= (\Sigma, Q', q'_0, F', \delta')$ that also accepts L. ($L(N) = L(D)$)

- NFAs are more flexible and easier to build. But DFAs have no less power than NFAs.

NFA ↔ 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

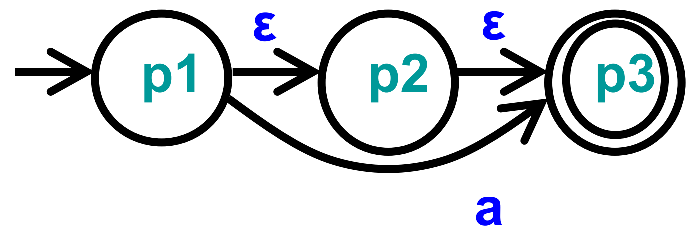
- Reduction applied using the subset algorithm
  - DFA state is a subset of set of all NFA states
- Algorithm
  - Input
    - NFA (Σ, Q, q_0, F_n, δ)
  - Output
    - DFA (Σ, R, r_0, F_d, δ)
  - Using two subroutines
    - ε-closure(δ, p) (and ε-closure(δ, Q))
    - move(δ, p, σ) (and move(δ, Q, σ))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

ε-transitions and ε-closure

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

ε-closure: Example 1

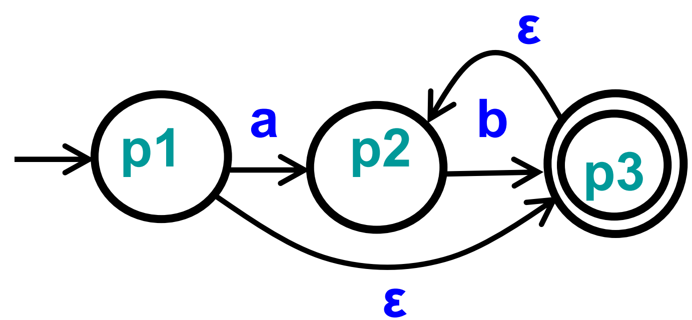
- Following NFA contains
  - $p1 \xrightarrow{\varepsilon} p2$
  - $p2 \xrightarrow{\varepsilon} p3$
  - $p1 \xrightarrow{\varepsilon} p3$
    - Since $p1 \xrightarrow{\varepsilon} p2$ and $p2 \xrightarrow{\varepsilon} p3$



- ε-closures
  - ε-closure(p1) = { p1, p2, p3 }
  - ε-closure(p2) = { p2, p3 }
  - ε-closure(p3) = { p3 }
  - ε-closure({ p1, p2 }) = { p1, p2, p3 } ∪ { p2, p3 }

ε-closure: Example 2

- Following NFA contains
  - $p1 \xrightarrow{\varepsilon} p3$
  - $p3 \xrightarrow{\varepsilon} p2$
  - $p1 \xrightarrow{\varepsilon} p2$



- ε-closures
  - ε-closure(p1) = { p1, p2, p3 }
  - ε-closure(p2) = { p2 }
  - ε-closure(p3) = { p2, p3 }
  - ε-closure({ p2,p3 }) = { p2 } ∪ { 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 $\cup$ $\{q \mid p \in R, (p, \varepsilon, q) \in \delta\}$ // new ε-reachable states

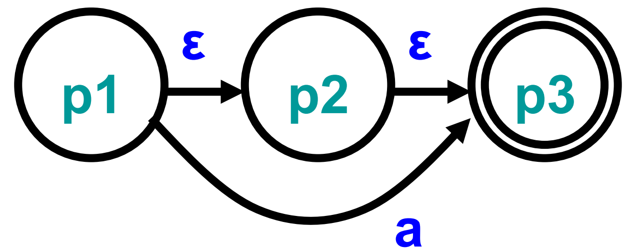
Until R = R' // stop when no new states

This algorithm computes a fixed point

ε-closure Algorithm Example

- Calculate ε-closure(δ,{p1})

R	R’
{p1}	{p1}
{p1}	{p1, p2}
{p1, p2}	{p1, p2, p3}
{p1, p2, p3}	{p1, p2, p3}



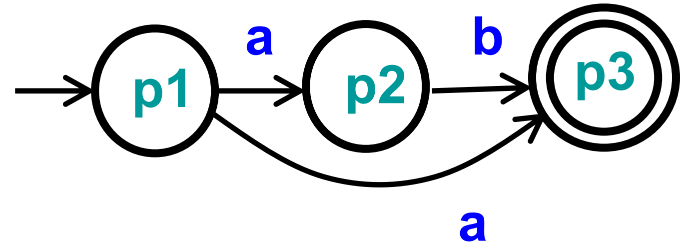

```
Let R’ = R
Repeat
   Let R= R’
   Let R’ = R ∪ {q | p ∈ R, (p, ε, q) ∈ δ}
Until R = R’
```

Calculating move(p,σ)

- move(δ,p,σ)
  - 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” move() to a *set* of states Q
  - Notes:
    - move(δ,p,σ) is Ø if no transition $(p,\sigma,q) \in \delta$, for any q

move(p,σ) : Example 1

- Following NFA
  - Σ = { a, b }

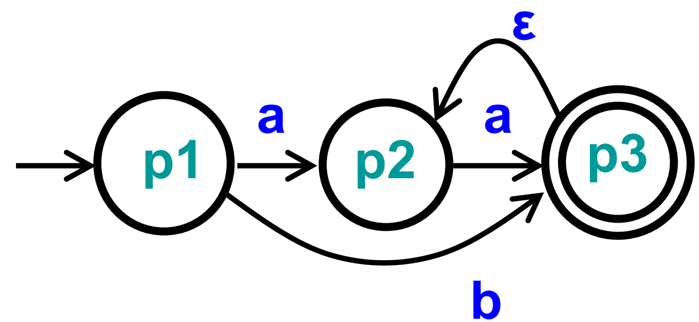


- Move
  - move(p1, a) = { p2, p3 }
  - move(p1, b) = Ø
  - move(p2, a) = Ø
  - move(p2, b) = { p3 }
  - move(p3, a) = Ø
  - move(p3, b) = Ø

move({p1,p2},b) = { p3 }

move(p,σ) : Example 2

- Following NFA
  - Σ = { a, b }



- Move
  - move(p1, a) = { p2 }
  - move(p1, b) = { p3 }
  - move(p2, a) = { p3 }
  - move(p2, b) = Ø
  - move(p3, a) = Ø
  - move(p3, b) = Ø

move({p1,p2},a) = {p2,p3}

NFA $\rightarrow$ DFA Reduction Algorithm ("subset")

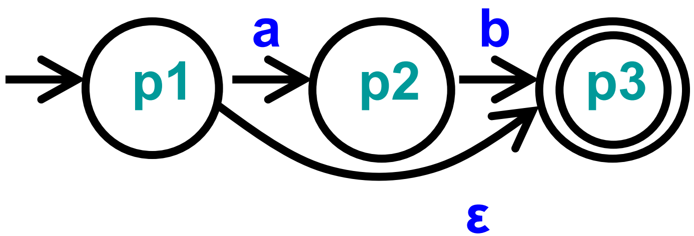
- Input NFA $(\Sigma, Q, q_0, F_n, \delta)$, Output DFA $(\Sigma, R, r_0, F_d, \delta')$
- Algorithm

```
Let r₀ = ε-closure(δ,q₀), add it to R          // DFA start state
While ∃ an unmarked state r ∈ R                // process DFA state r
    Mark r                                     // each state visited once
    For each σ ∈ Σ                             // for each symbol σ
        Let E = move(δ,r,σ)                    // states reached via σ
        Let e = ε-closure(δ,E)                 // states reached via ε
        If e ∉ R                               // if state e is new
            Let R = R ∪ {e}                    // add e to R (unmarked)
        Let δ' = δ' ∪ {r, σ, e}                // add transition r→e on σ
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}           // final if include state in F_n
```

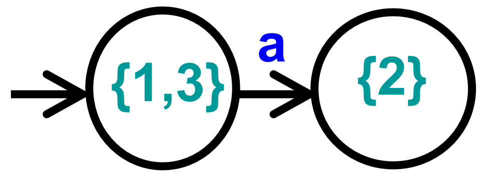
NFA → DFA Example

- Start = ε-closure(δ,p1) = { {p1,p3} }
- R = { {p1,p3} }
- r ∈ R = {p1,p3}
- move(δ,{p1,p3},a) = {p2}
  - e = ε-closure(δ,{p2}) = {p2}
  - R = R ∪ {{p2}} = { {p1,p3}, {p2} }
  - δ' = δ' ∪ {{p1,p3}, a, {p2}}
- move(δ,{p1,p3},b) = Ø

NFA



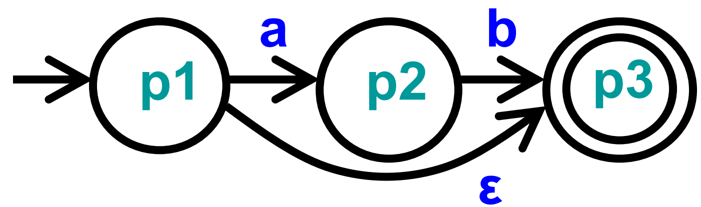
DFA



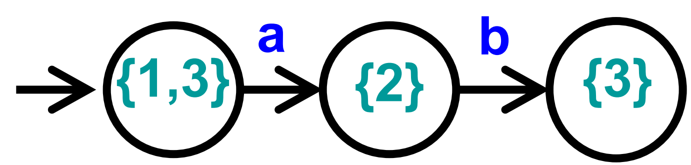
NFA → DFA Example (cont.)

- R = { {p1,p3}, {p2} }
- r ∈ R = {p2}
- move(δ,{p2},a) = Ø
- move(δ,{p2},b) = {p3}
  - e = ε-closure(δ,{p3}) = {p3}
  - R = R ∪ {{p3}} = { {p1,p3}, {p2}, {p3} }
  - δ’ = δ’ ∪ {{p2}, b, {p3}}

NFA



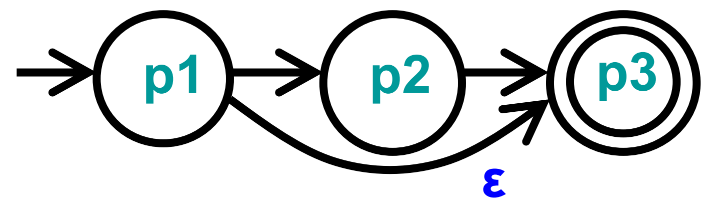
DFA



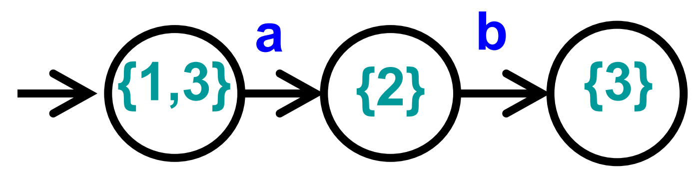
NFA → DFA Example (cont.)

- R = { {p1,p3}, {p2}, {p3} }
- r ∈ R = {p3}
- Move({p3},a) = Ø
- Move({p3},b) = Ø
- Mark {p3}, exit loop
- F_d = {{p1,p3}, {p3}}
  - Since p3 ∈ F_n
- Done!

NFA

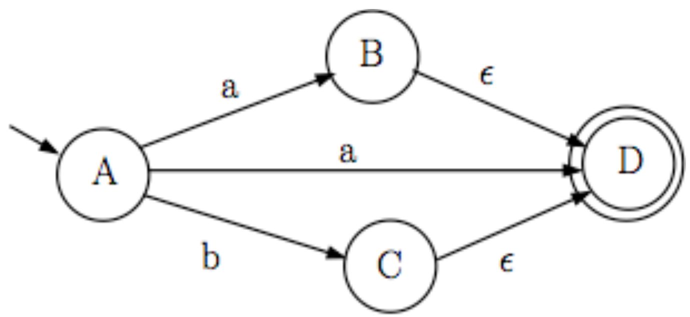


DFA

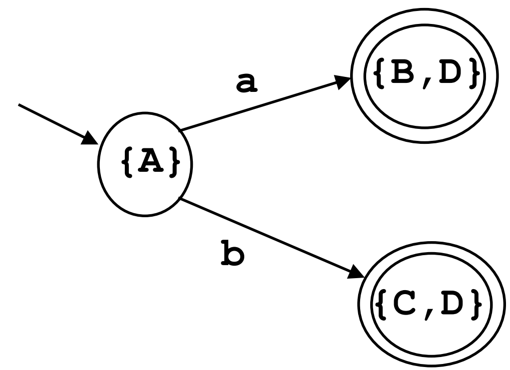


NFA → DFA Example 2

- NFA

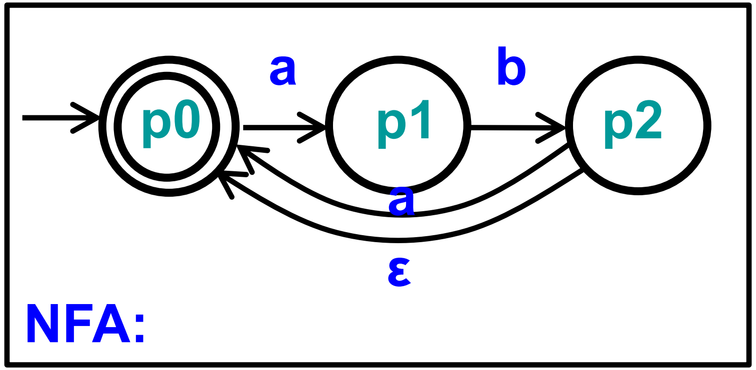


- DFA

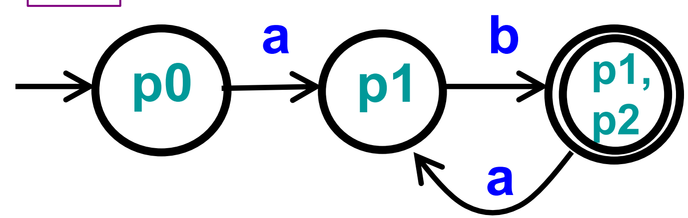


R = { {A}, {B,D}, {C,D} }

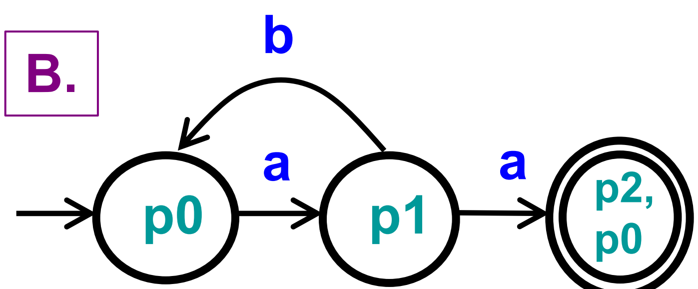
Quiz 4: Which DFA is equiv to this NFA?



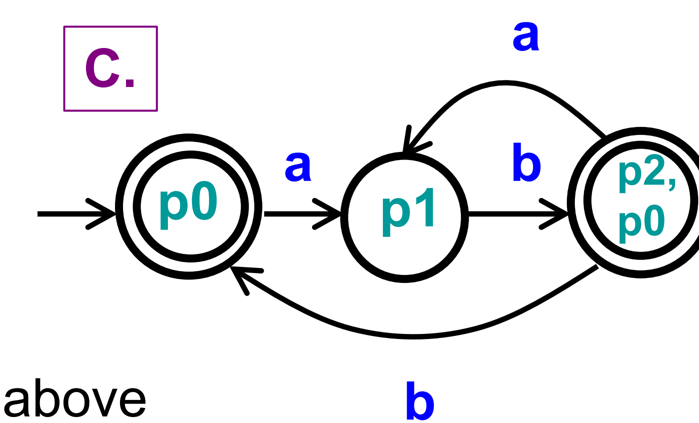
A.



B.

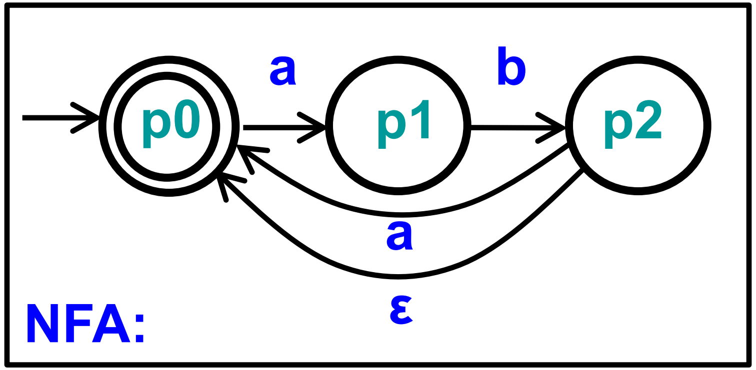


C.

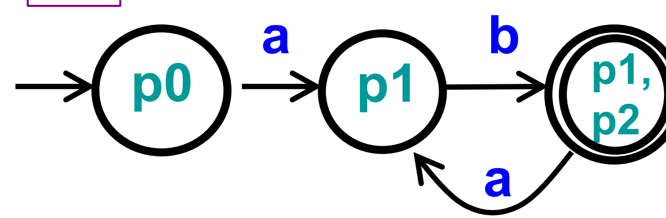


D. None of the above

Quiz 4: Which DFA is equiv to this NFA?

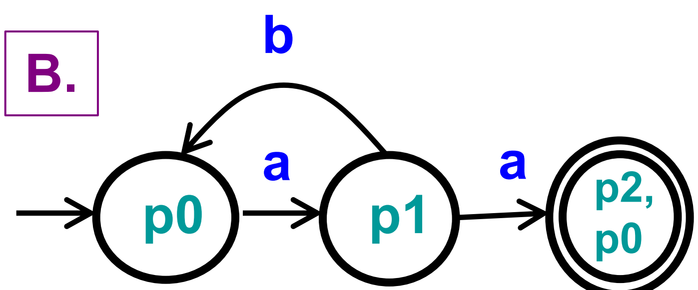


A.



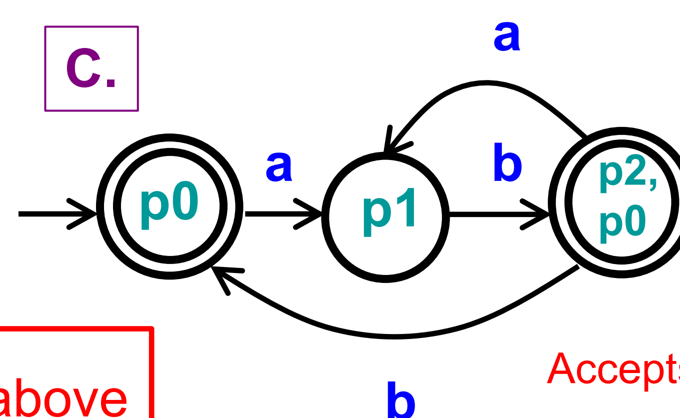
Rejects **ε**

B.



Rejects **ε**

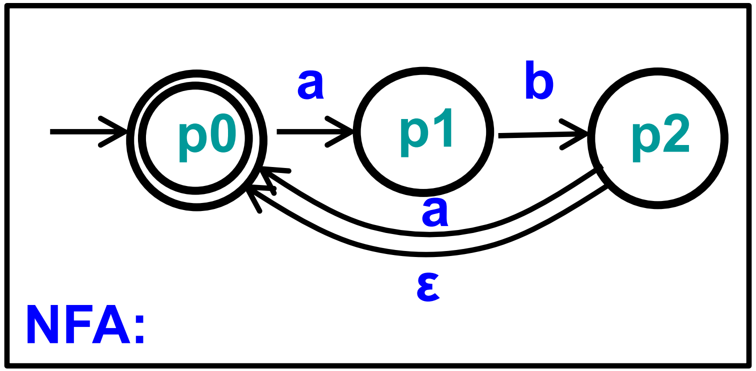
C.

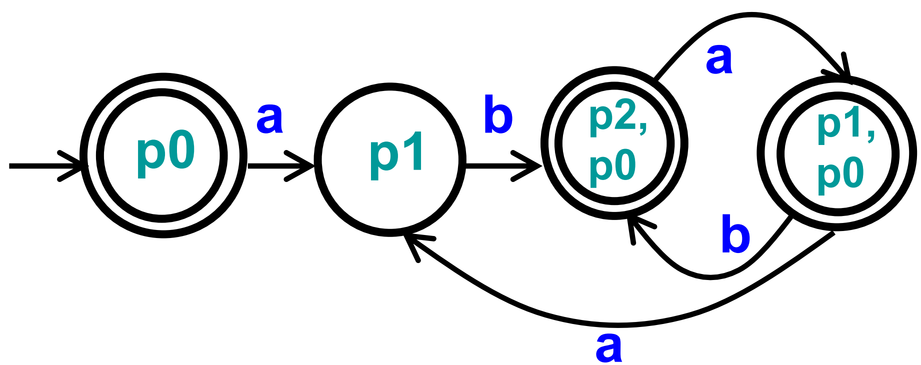


Accepts abb

D. None of the above

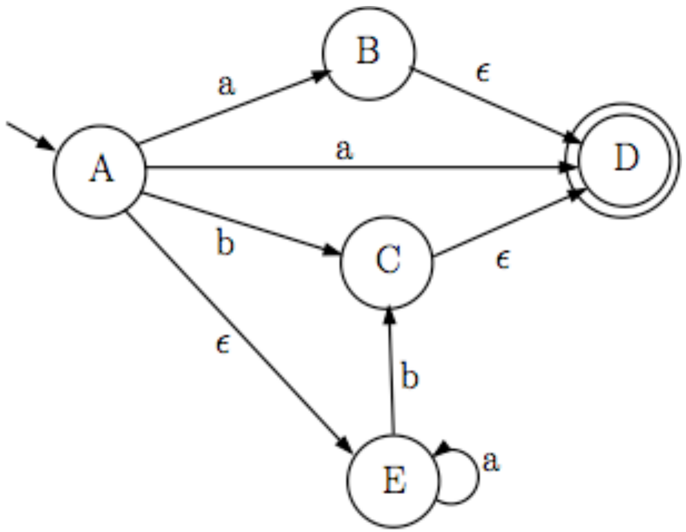
Actual Answer



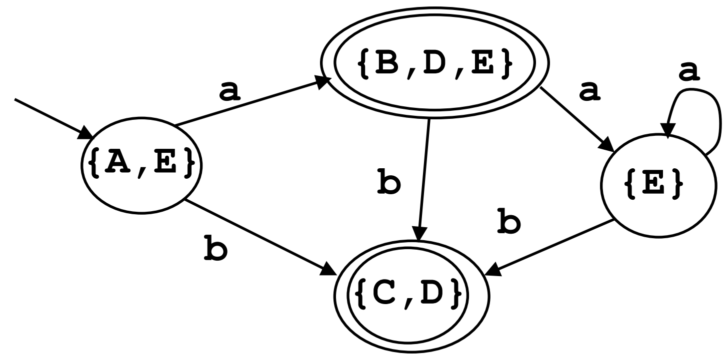


NFA $\rightarrow$ DFA Example 3

- NFA



- DFA



R = { {A,E}, {B,D,E}, {C,D}, {E} }