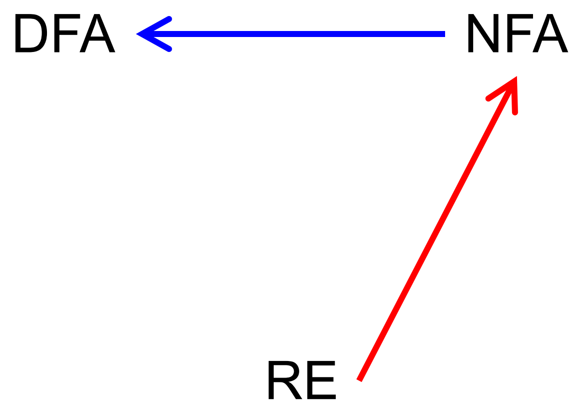
CMSC 330: Organization of Programming Languages

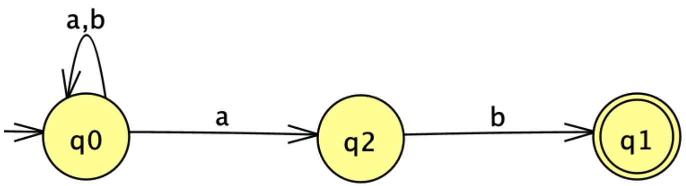
Reducing NFA to DFA and DFAs Minimization

Reducing NFA to DFA

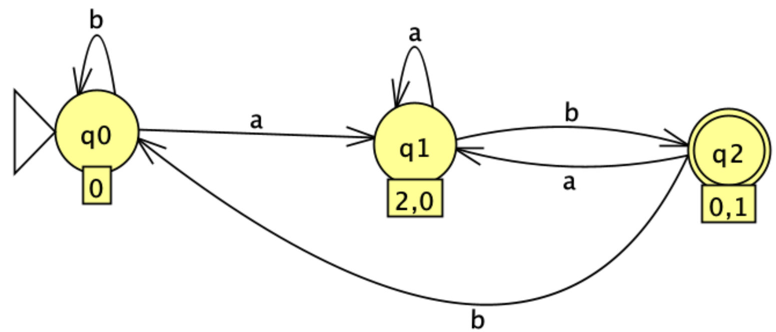


Why NFA → DFA

- DFA is generally more efficient than NFA



NFA



DFA

Language: (a|b)*ab

How to accept bab?

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 it is not more powerful than DFAs

$$\text{NFA} \leftrightarrow \text{DFA}$$

How to Convert NFA to DFA

Subset Construction Algorithm

Input NFA $(\Sigma, Q, q_0, F_n, \delta)$

Output DFA $(\Sigma, R, r_0, F_d, \delta')$

Subset Construction Algorithm

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

Let $r_0 = \varepsilon$-closure(δ, q_0), add it to R

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

  Mark r

  For each $\sigma \in \Sigma$

  Let E = move(δ, r, σ)

    Let $e = \varepsilon$-closure(δ, E)

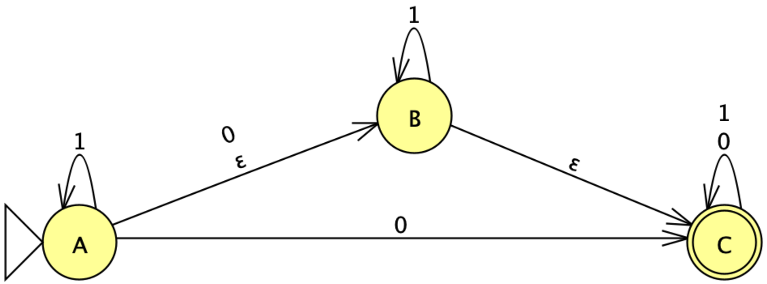
    If $e \notin R$

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

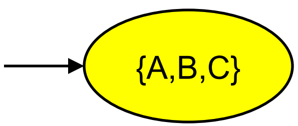
    Let $\delta' = \delta' \cup \{r, \sigma, e\}$

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

NFA



DFA



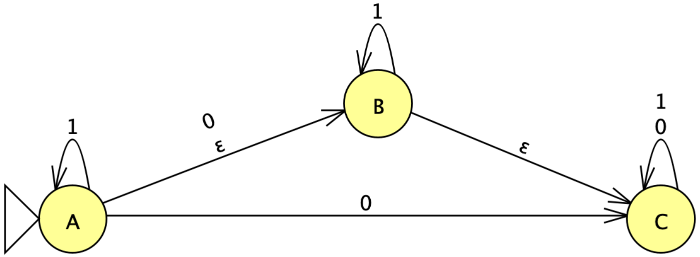
New Start State

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

- Mark r
- For each σ ∈ Σ
- Let E = move(δ,r,σ)
    - Let e = ε-closure(δ,E)
    - If e ∉ R
        - Let R = R ∪ {e}
    - Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}



Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //0

Let E = move(δ,r,σ)

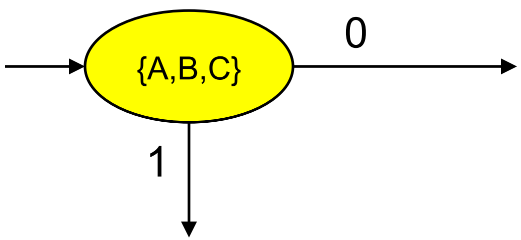
Let e = ε-closure(δ,E)

If e ∉ R

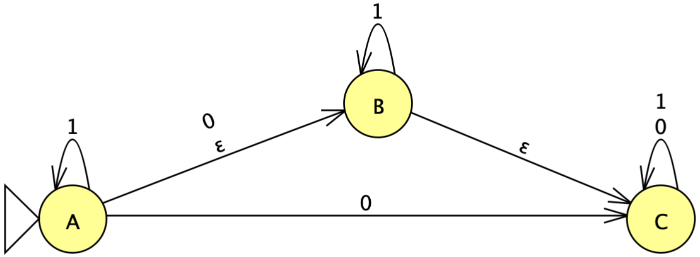
Let R = R ∪ {e}

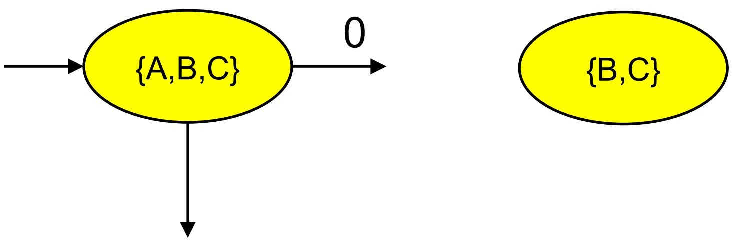
Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}



	0	1
{A,B,C}		





	0	1
{A,B,C}	{B,C}	

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ

Let E = move(δ,r,σ)

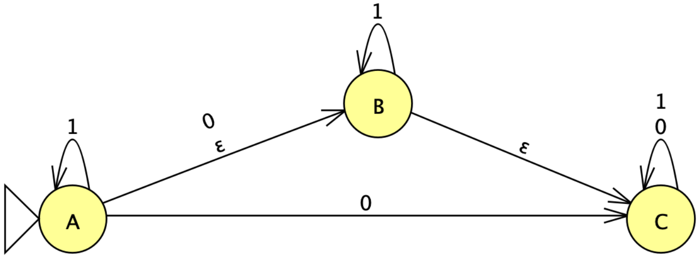
Let e = ε-closure(δ,E)

If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}



Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ

Let E = move(δ,r,σ)

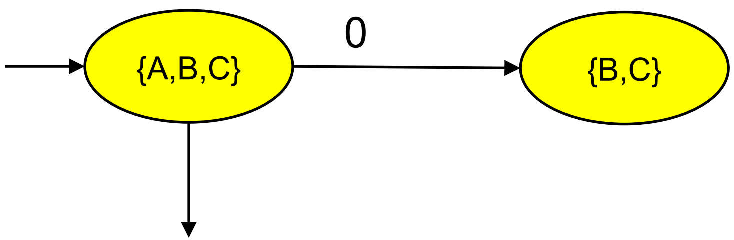
Let e = ε-closure(δ,E)

If e ∉ R

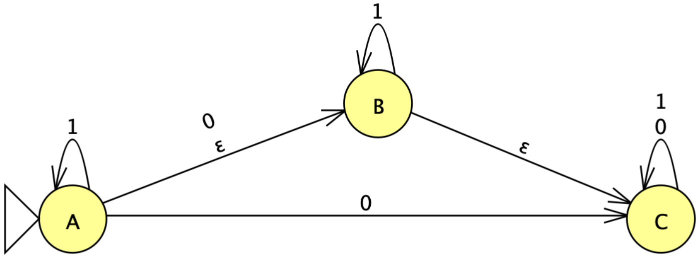
Let R = R ∪ {e}

Let δ’ = δ’ ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}



	0	1
{A,B,C}	{B,C}	
{B,C}		



Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //1

Let E = move(δ,r,σ)

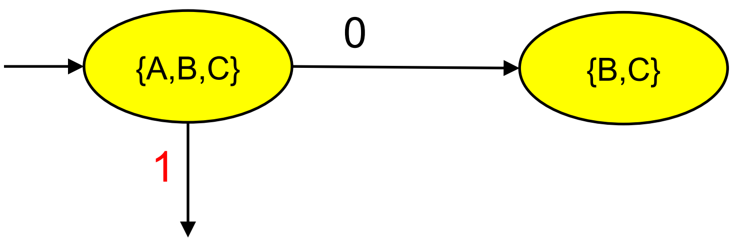
Let e = ε-closure(δ,E)

If e ∉ R

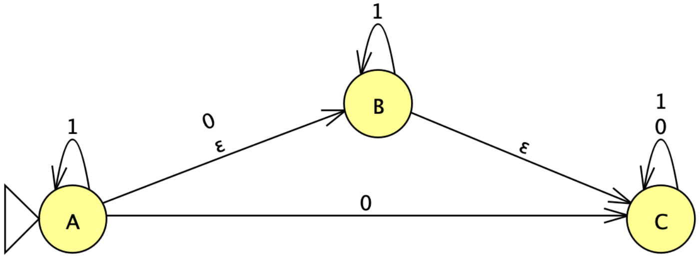
Let R = R ∪ {e}

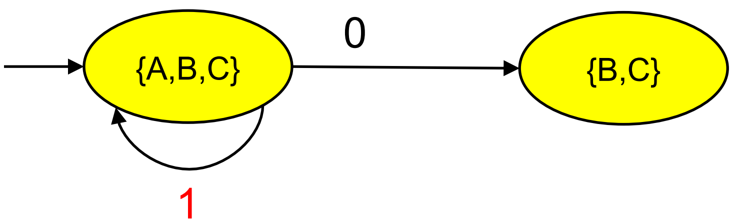
Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}



	0	1
{A,B,C}	{B,C}	
{B,C}		





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}		

Let r_0 = ε-closure(δ,q_0), add it to R

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

  Mark r

  For each $\sigma \in \Sigma$ //1

  Let E = move(δ,r,σ)

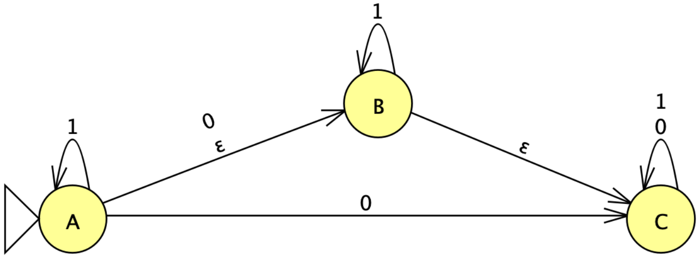
   Let e = ε-closure(δ,E)

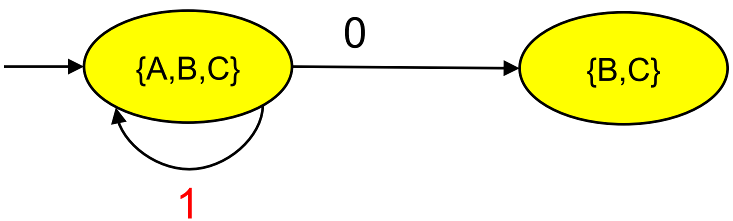
   If e $\notin$ R

    Let R = R $\cup$ {e}

   Let δ' = $\delta' \cup$ {r, σ, e}

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





Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

  Mark r

  For each σ ∈ Σ //1

  Let E = move(δ,r,σ)

   Let e = ε-closure(δ,E)

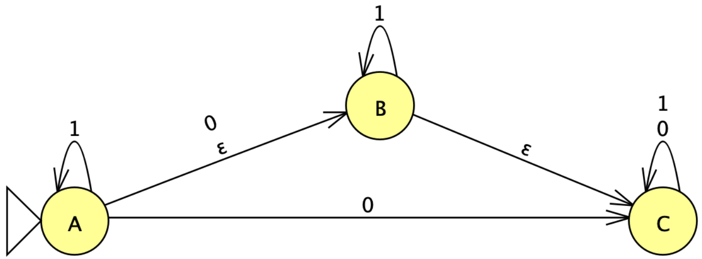
   If e ∉ R

    Let R = R ∪ {e}

   Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}		



Let r_0 = ε-closure(δ,q_0), add it to R

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

Mark r

For each $\sigma \in \Sigma$ //1

Let E = move(δ,r,σ)

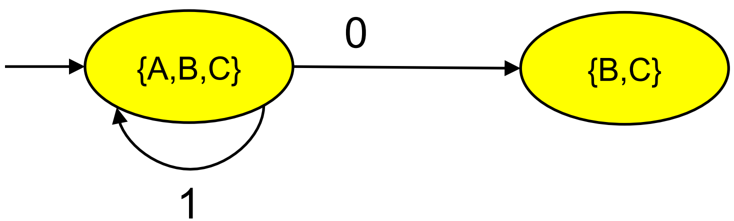
Let e = ε-closure(δ,E)

If e $\notin$ R

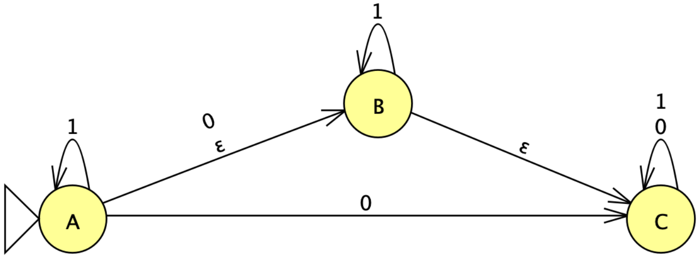
Let R = R $\cup$ {e}

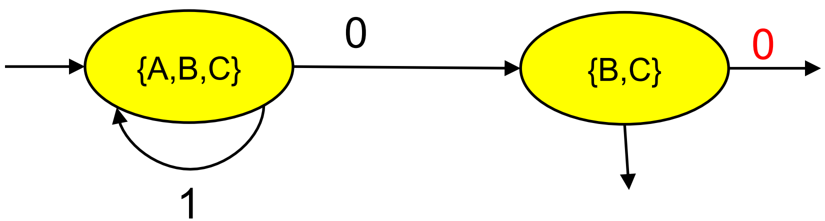
Let δ' = δ' $\cup$ {r, σ, e}

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



	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}		





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}		

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //0

Let E = move(δ,r,σ)

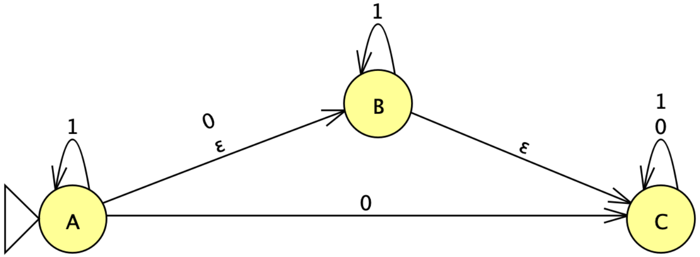
Let e = ε-closure(δ,E)

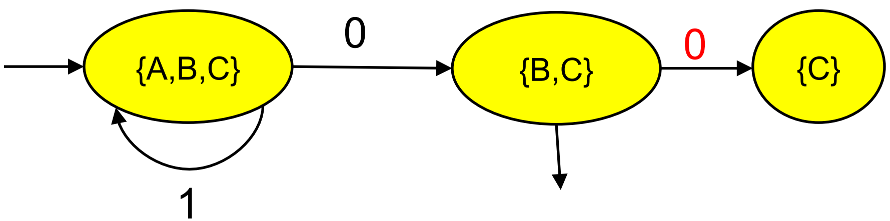
If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //0

Let E = move(δ,r,σ)

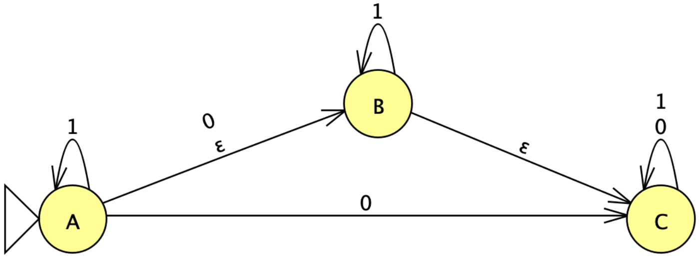
Let e = ε-closure(δ,E)

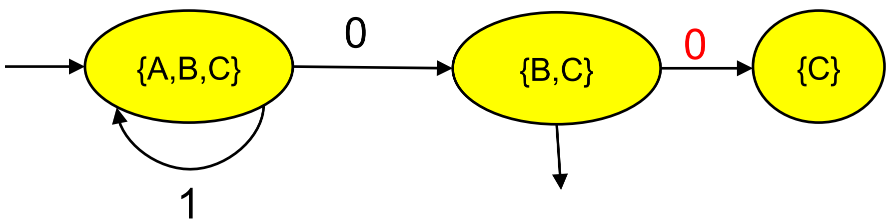
If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}





Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //0

Let E = move(δ,r,σ)

Let e = ε-closure(δ,E)

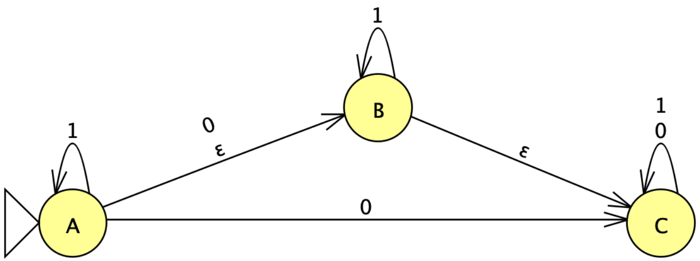
If e ∉ R

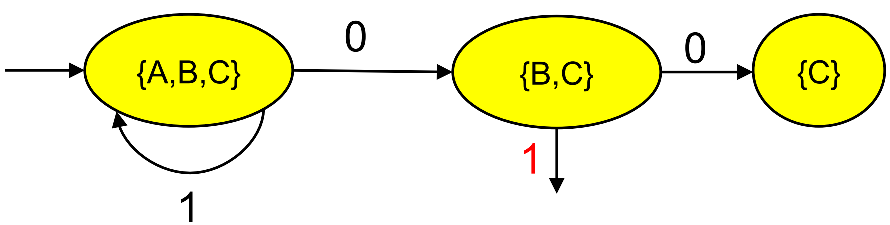
Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	
{C}		





Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //1

Let E = move(δ,r,σ)

Let e = ε-closure(δ,E)

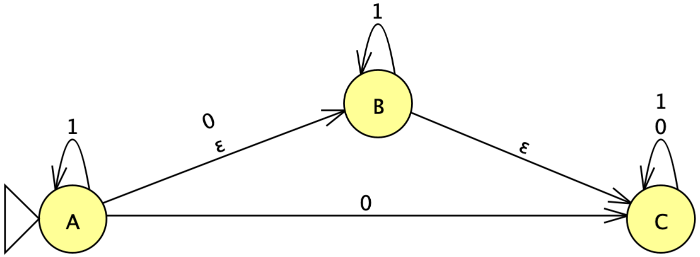
If e ∉ R

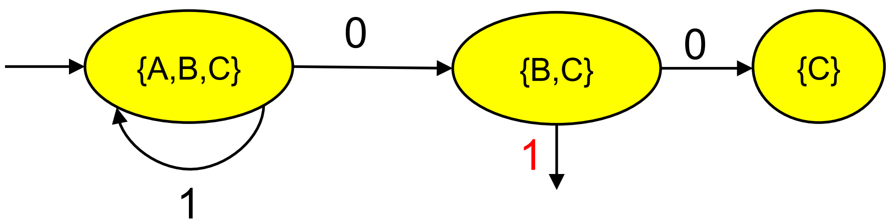
Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	?
{C}		





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}		

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //1

Let E = move(δ,r,σ)

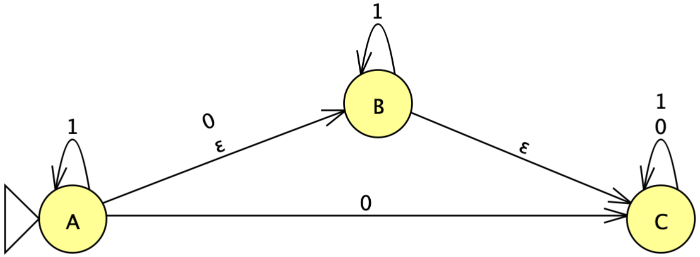
Let e = ε-closure(δ,E)

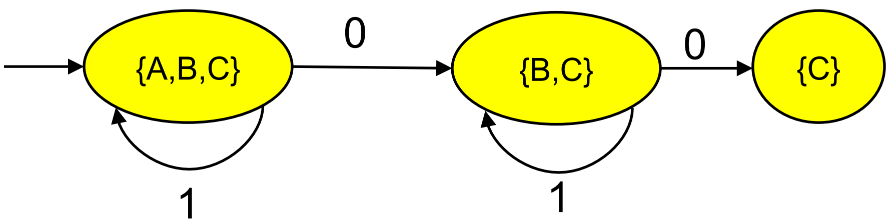
If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}		

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //1

Let E = move(δ,r,σ)

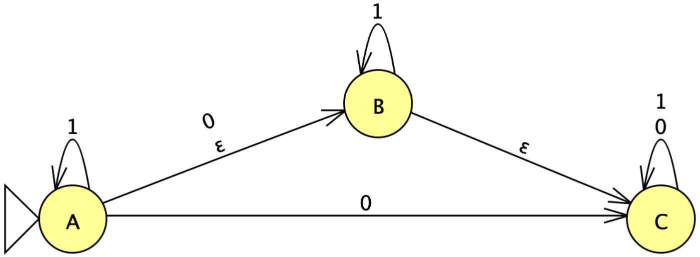
Let e = ε-closure(δ,E)

If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}



Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

  Mark r

  For each σ ∈ Σ //1

  Let E = move(δ,r,σ)

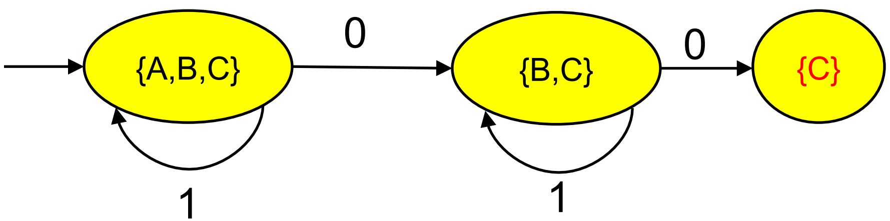
    Let e = ε-closure(δ,E)

    If e ∉ R

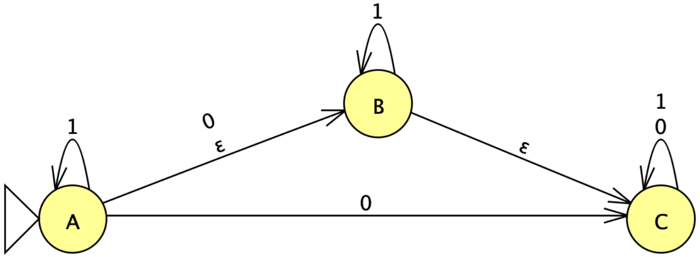
      Let R = R ∪ {e}

    Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}



	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}		



Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ

Let E = move(δ,r,σ)

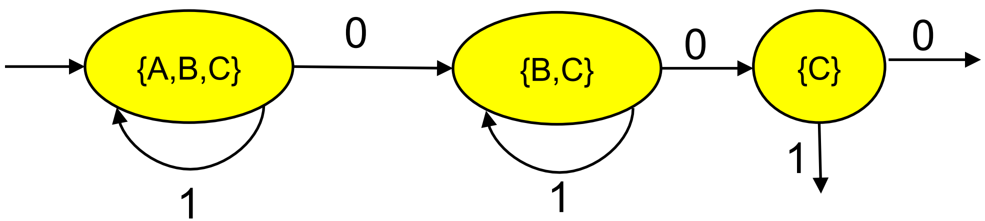
Let e = ε-closure(δ,E)

If e ∉ R

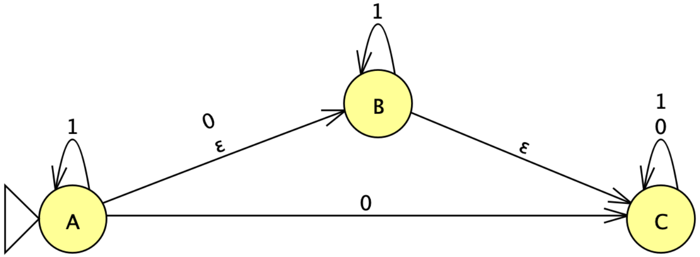
Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}



	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}		



Let r_0 = ε-closure(δ,q_0), add it to R

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

Mark r

For each $\sigma \in \Sigma$ //0

Let E = move(δ,r,σ)

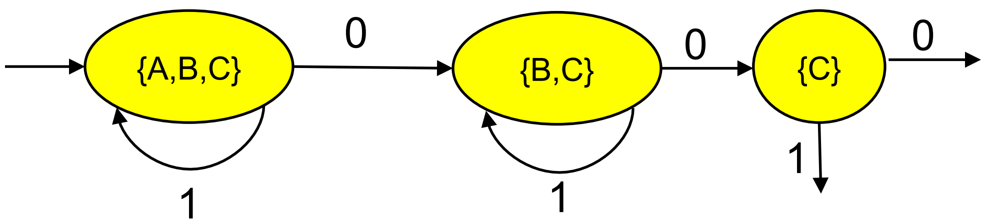
Let e = ε-closure(δ,E)

If e $\notin$ R

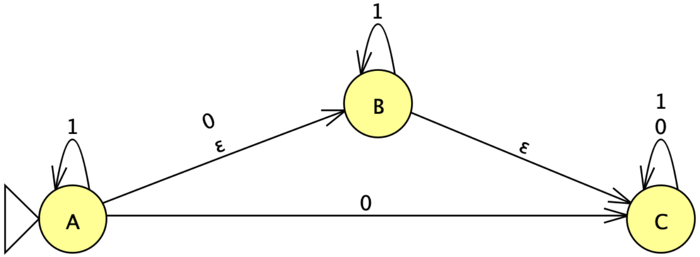
Let R = R $\cup$ {e}

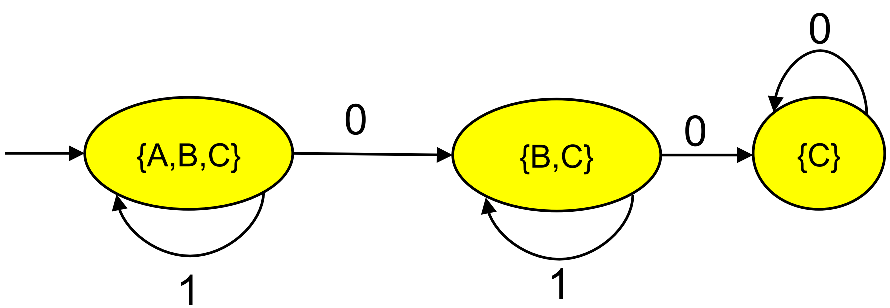
Let δ' = δ' $\cup$ {r, σ, e}

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



	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //0

Let E = move(δ,r,σ)

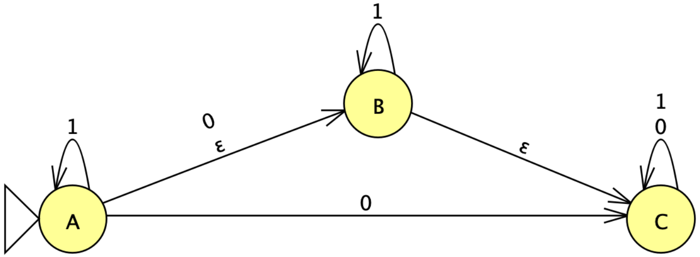
Let e = ε-closure(δ,E)

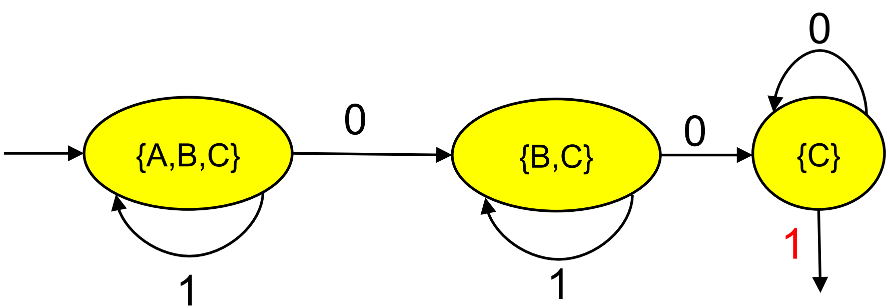
If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //1

Let E = move(δ,r,σ)

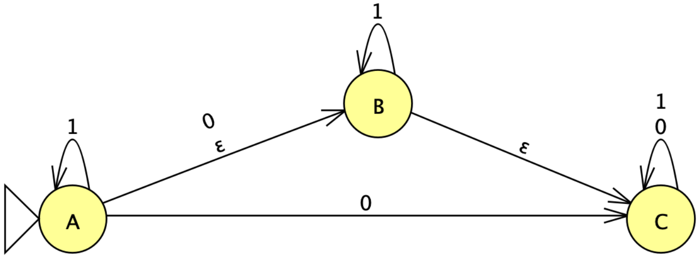
Let e = ε-closure(δ,E)

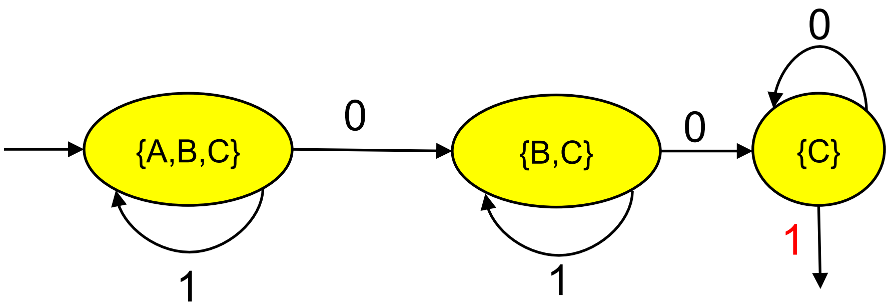
If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

  Mark r

  For each σ ∈ Σ   //1

  Let E = move(δ,r,σ)

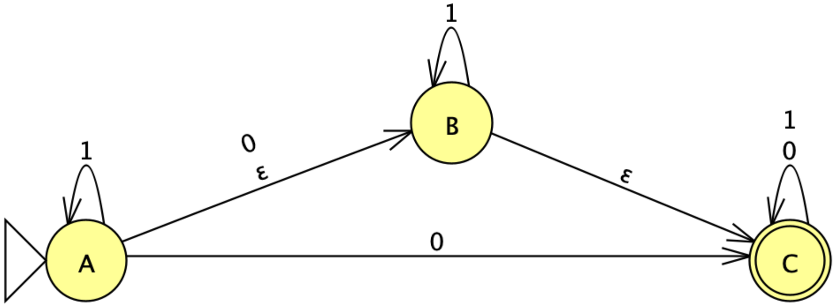
  → Let e = ε-closure(δ,E)

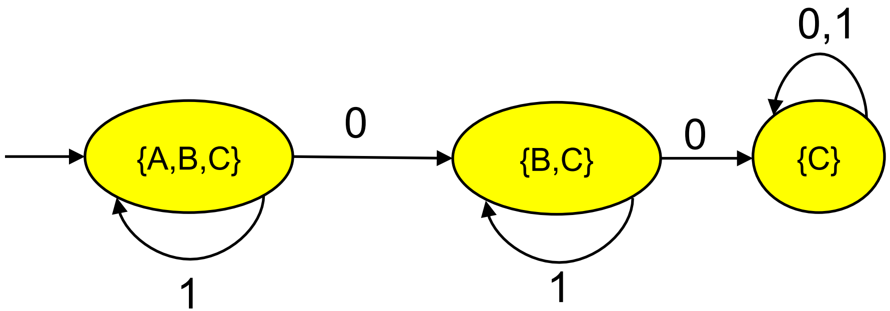
   If e ∉ R

    Let R = R ∪ {e}

   Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}





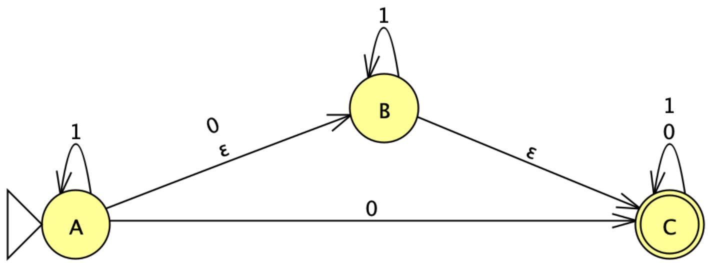
Let r_0 = ε-closure(δ,q_0), add it to R

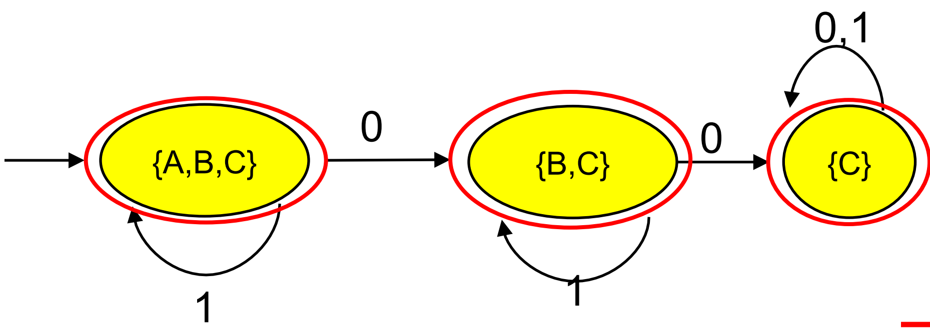
While ∃ an unmarked state r ∈ R

- Mark r
- For each σ ∈ Σ //1
- Let E = move(δ,r,σ)
  - Let e = ε-closure(δ,E)
  - If e ∉ R
    - Let R = R ∪ {e}
  - Let δ' = δ' ∪ {r, σ, e}

Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}





	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}

Let r_0 = ε-closure(δ,q_0), add it to R

While ∃ an unmarked state r ∈ R

Mark r

For each σ ∈ Σ //1

Let E = move(δ,r,σ)

Let e = ε-closure(δ,E)

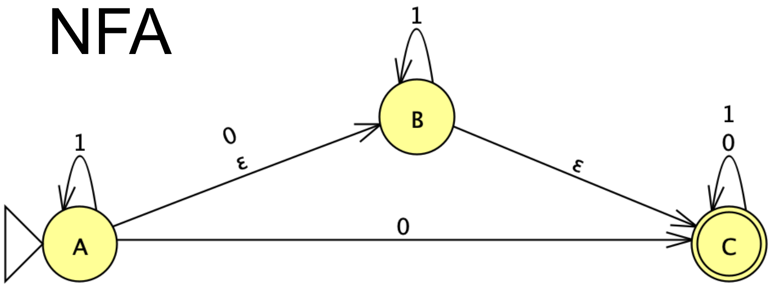
If e ∉ R

Let R = R ∪ {e}

Let δ' = δ' ∪ {r, σ, e}

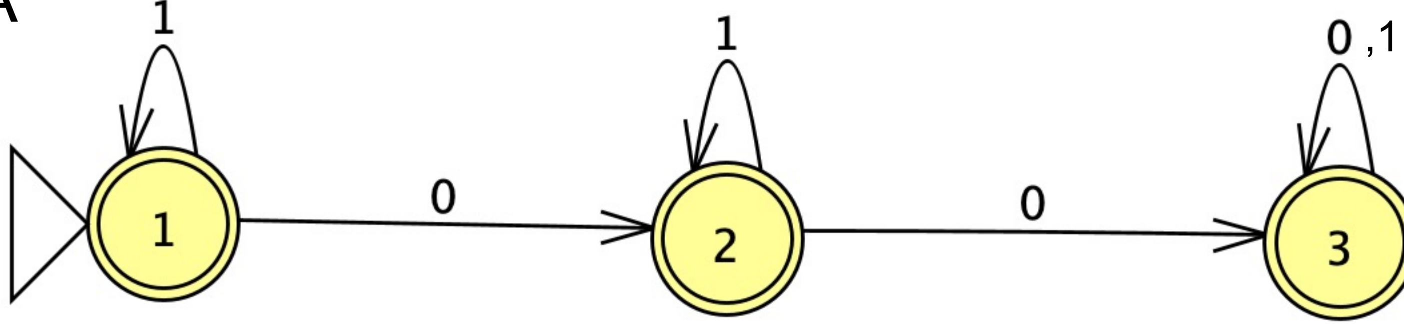
Let F_d = {r | ∃ s ∈ r with s ∈ F_n}

NFA

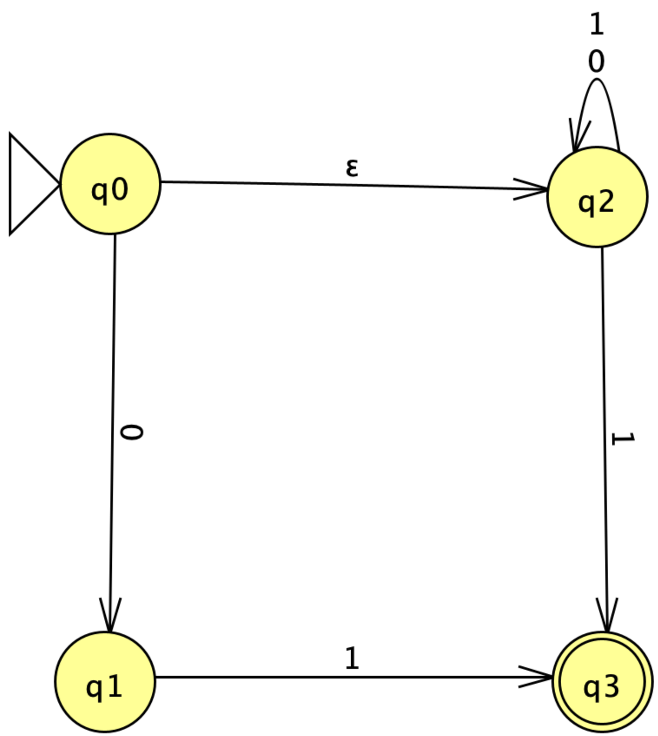


	0	1
{A,B,C}	{B,C}	{A,B,C}
{B,C}	{C}	{B,C}
{C}	{C}	{C}

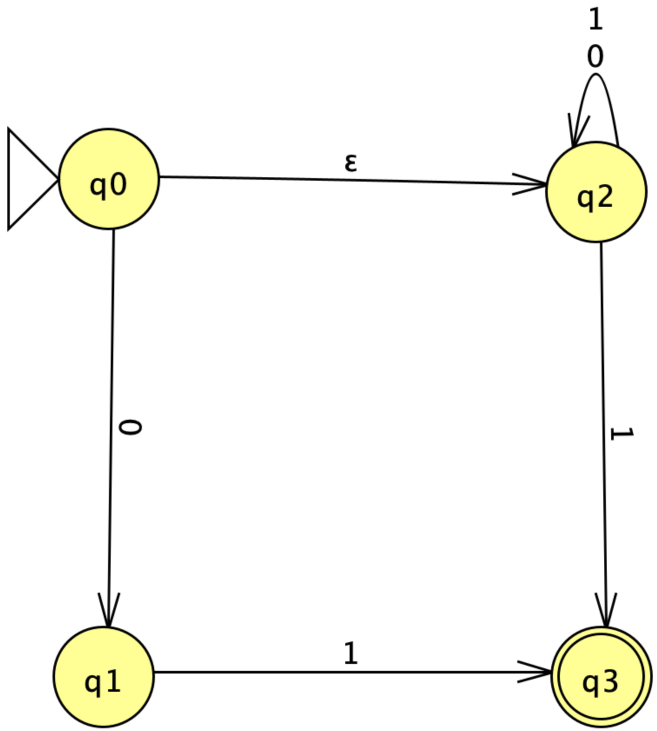
DFA

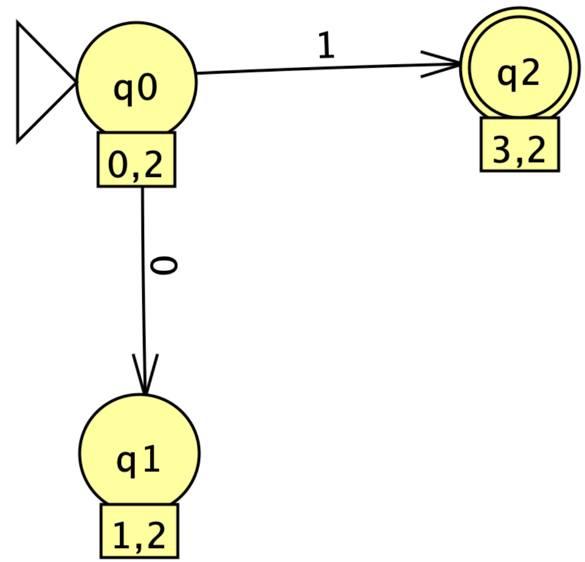


NFA → DFA Another Example

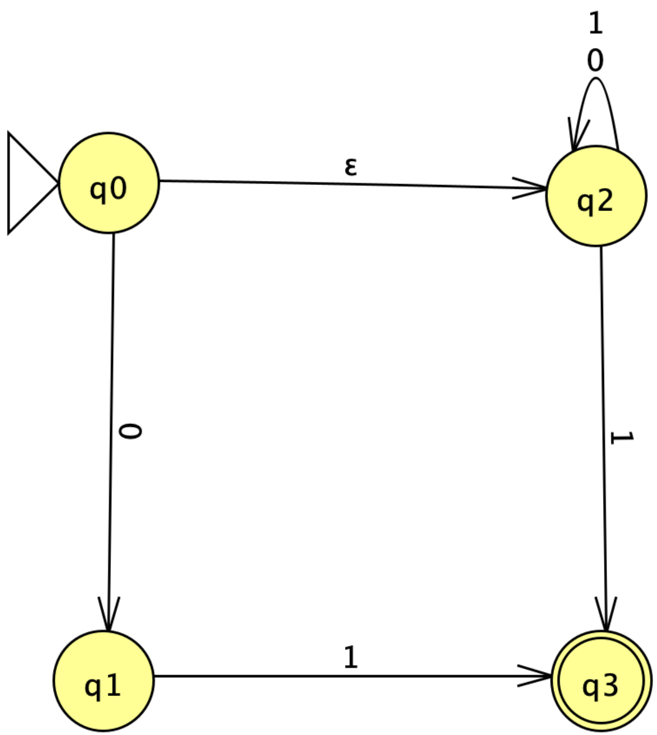


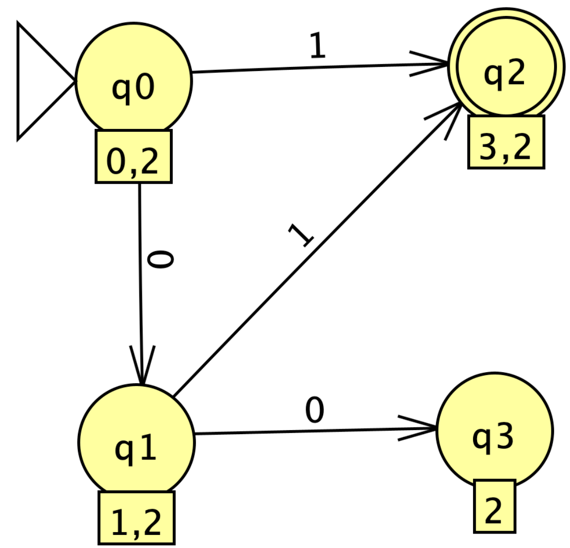
NFA → DFA Another Example



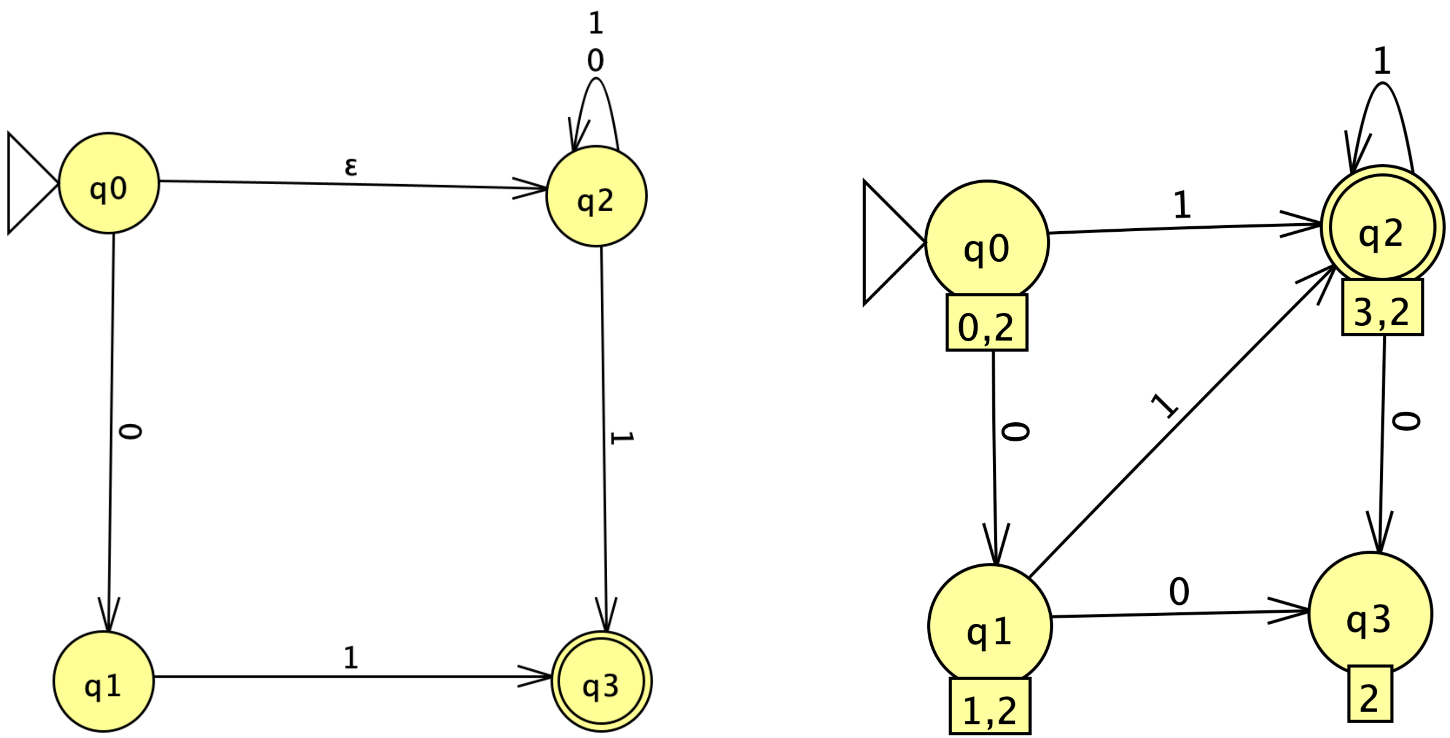


NFA → DFA Another Example

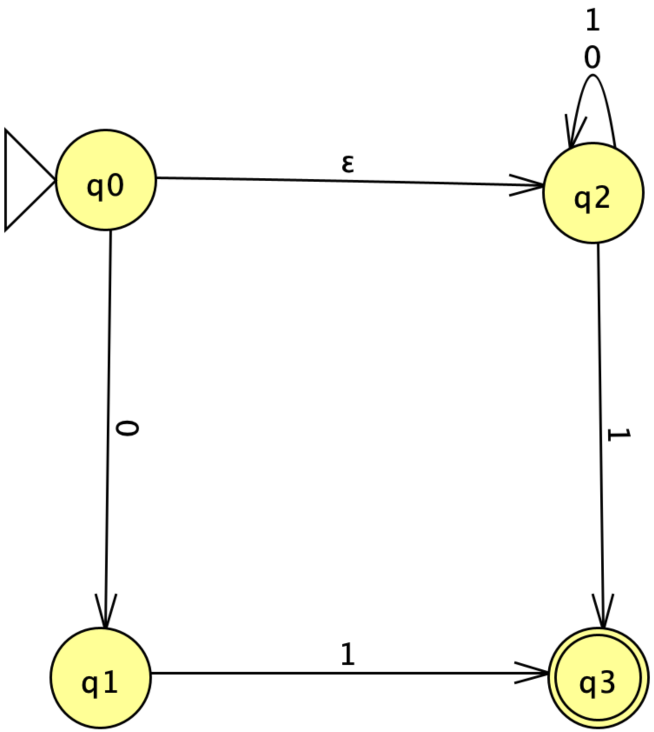




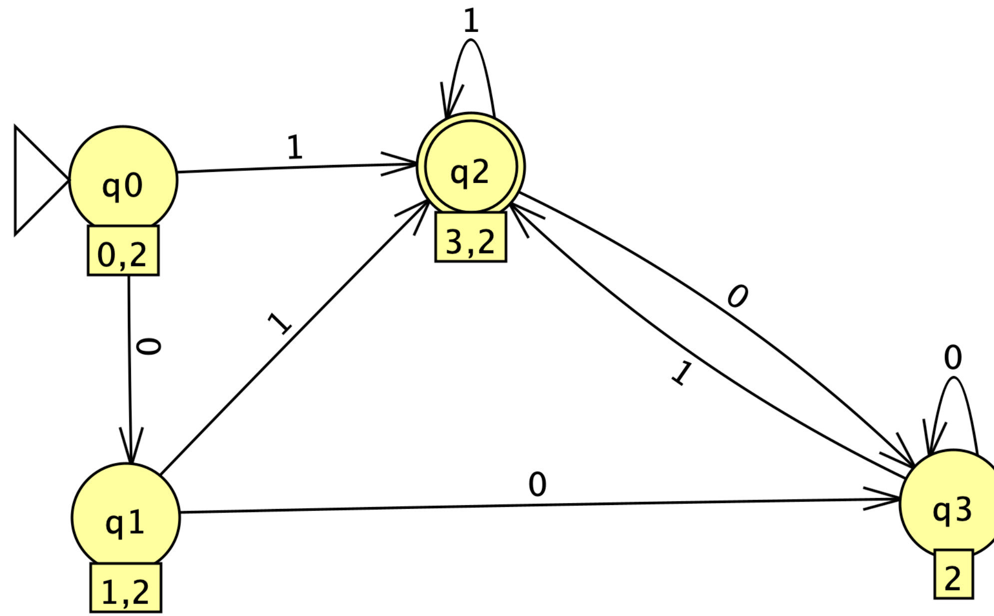
NFA → DFA Another Example



NFA → DFA Another Example

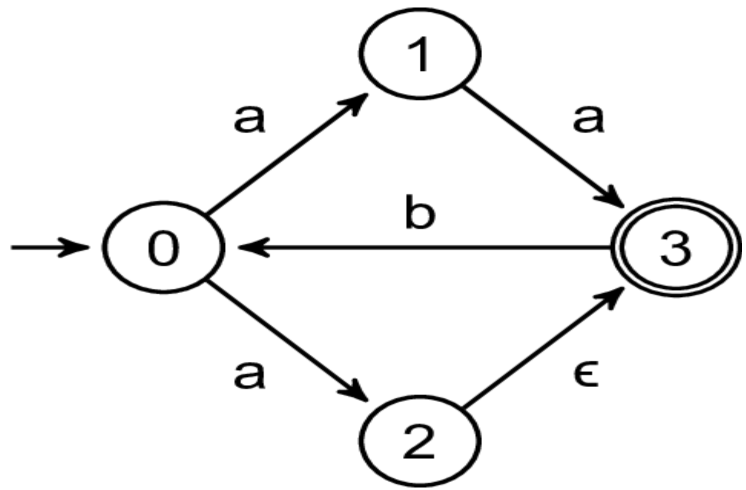


NFA

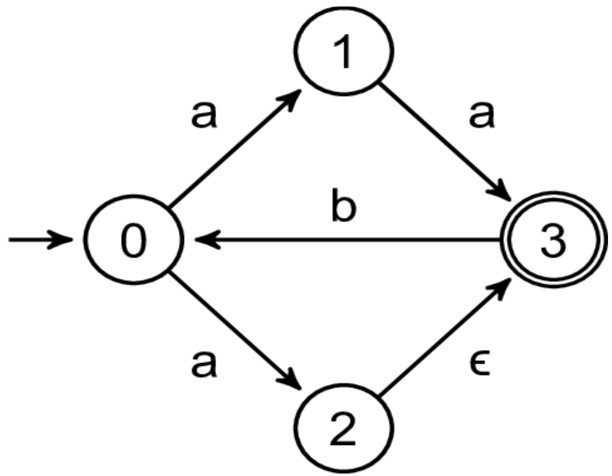


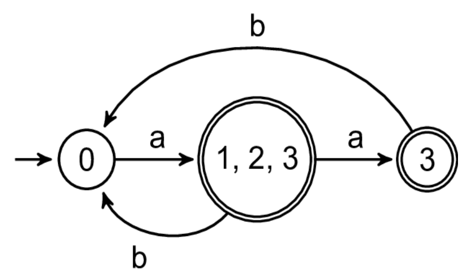
DFA

NFA → DFA Practice



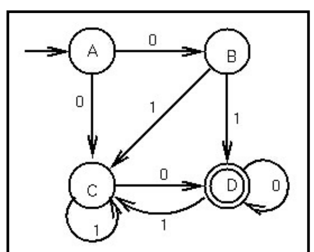
NFA → DFA Practice

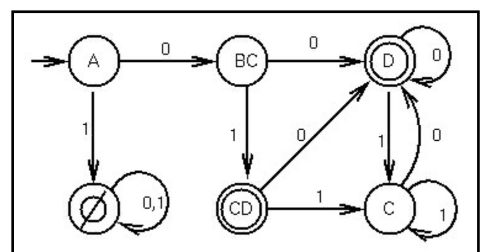




Analyzing the Reduction

- Can reduce any NFA to a DFA using subset alg.
- How many states in the DFA?
  - Each DFA state is a subset of the set of NFA states
  - Given NFA with n states, DFA may have 2^n states
    - Since a set with n items may have 2^n subsets
  - Corollary
    - Reducing a NFA with n states may be $O(2^n)$

NFA


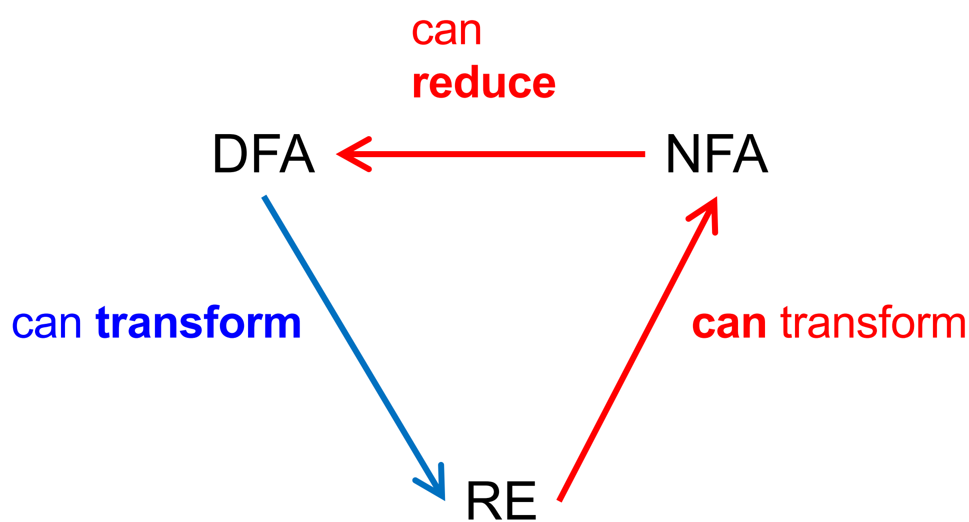


DFA

Recap: Matching a Regexp *R*

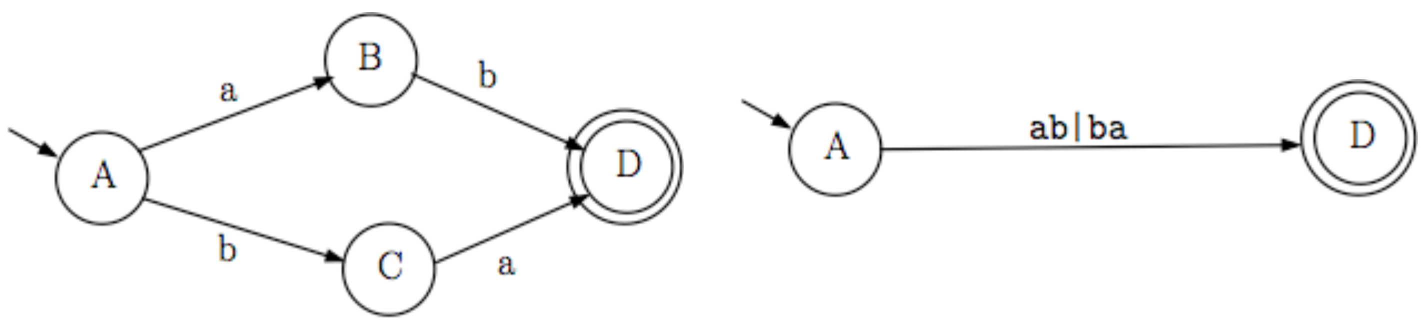
- Given *R*, construct NFA. Takes time O(*R*)
- Convert NFA to DFA. Takes time $O(2^{|R|})$
  - But usually not the worst case in practice
- Use DFA to accept/reject string s
  - Assume we can compute $\delta(q,\sigma)$ in constant time
  - Then time to process s is O(|s|)
    - Can't get much faster!
- Constructing the DFA is a one-time cost
  - But then processing strings is fast

Closing the Loop: Reducing DFA to RE



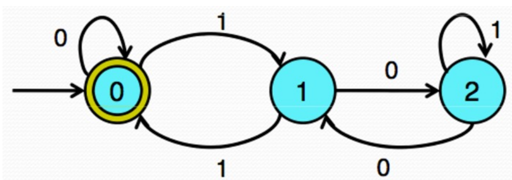
Reducing DFAs to REs

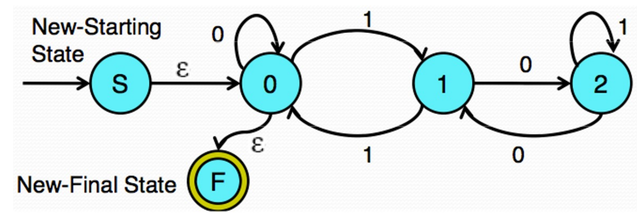
- General idea
  - Remove states one by one, labeling transitions with regular expressions
  - When two states are left (start and final), the transition label is the regular expression for the DFA

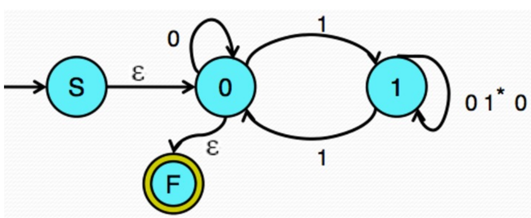


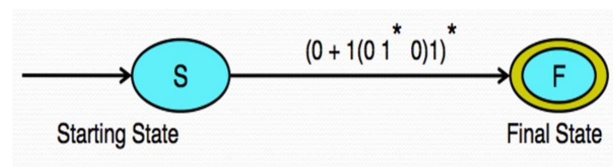
DFA to RE example

Language over Σ= {0,1} such that every string is a multiple of 3 in binary





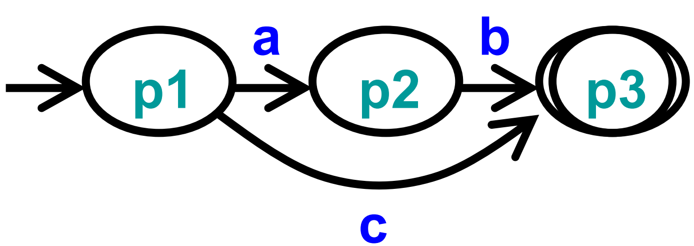


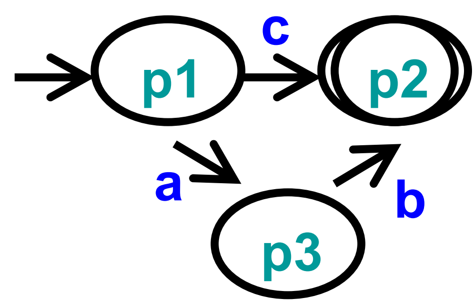


$$(0 + 1(0\,1^*\,0)1)^*$$

Minimizing DFAs

- Every regular language is recognizable by a unique minimum-state DFA
  - Ignoring the particular names of states
- In other words
  - For every DFA, there is a unique DFA with minimum number of states that accepts the same language





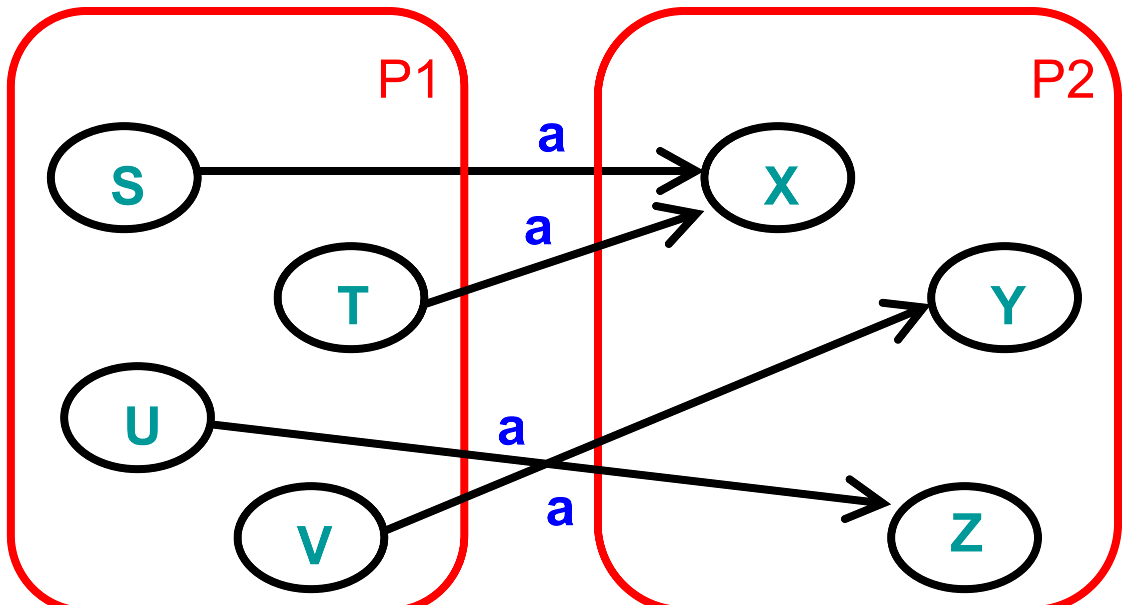
J. Hopcroft, “An n log n algorithm for minimizing states in a finite automaton,” 1971

Minimizing DFA: Hopcroft Reduction

- Intuition
  - Look to distinguish states from each other
    - End up in different accept / non-accept state with identical input
- Algorithm
  - Construct initial partition
    - Accepting & non-accepting states
  - Iteratively split partitions (until partitions remain fixed)
    - Split a partition if **members in partition have transitions to different partitions for same input**
      - Two states x, y belong in same partition if and only if for all symbols in Σ they transition to the same partition
  - Update transitions & remove dead states

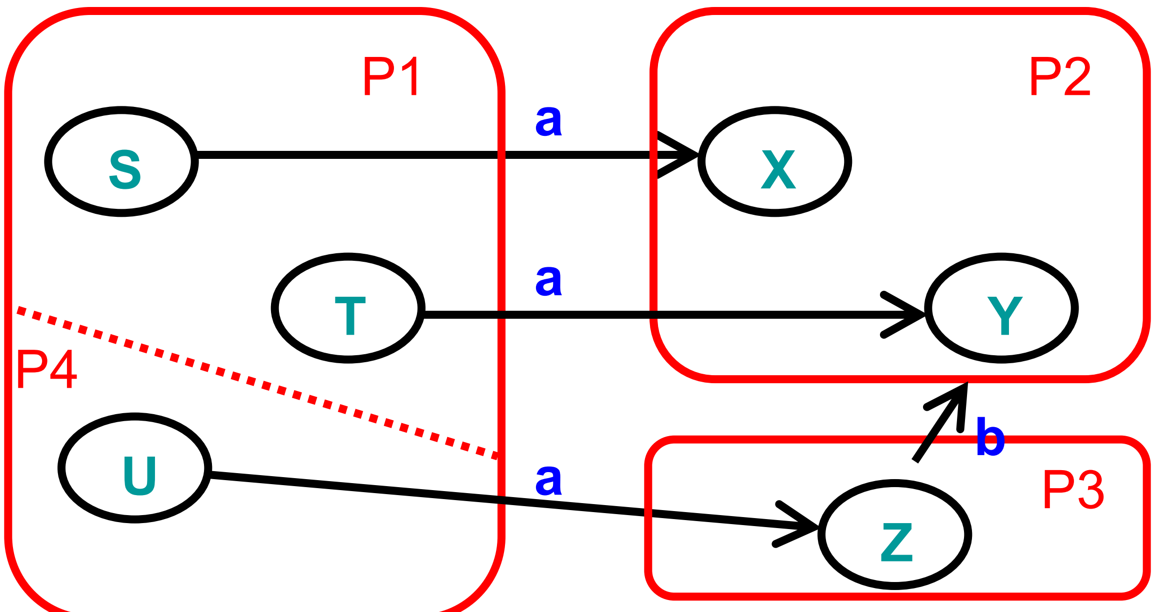
Splitting Partitions

- No need to split partition {S,T,U,V}
  - All transitions on a lead to identical partition P2
  - Even though transitions on a lead to different states



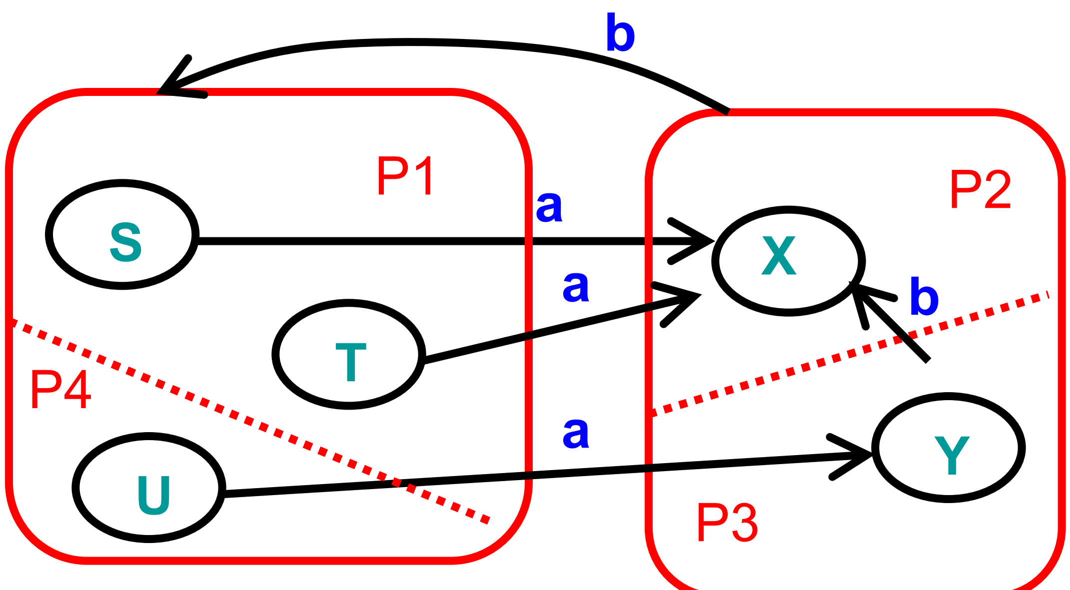
Splitting Partitions (cont.)

- Need to split partition {S,T,U} into {S,T}, {U}
  - Transitions on a from S,T lead to partition P2
  - Transition on a from U lead to partition P3



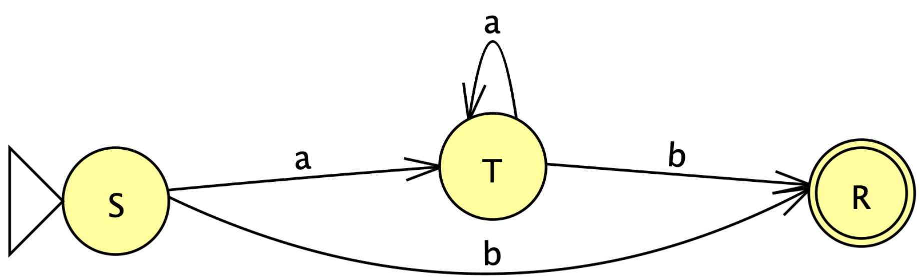
Resplitting Partitions

- Need to reexamine partitions after splits
  - Initially no need to split partition {S,T,U}
  - After splitting partition {X,Y} into {X}, {Y} we need to split partition {S,T,U} into {S,T}, {U}



Minimizing DFA: Example 1

- DFA

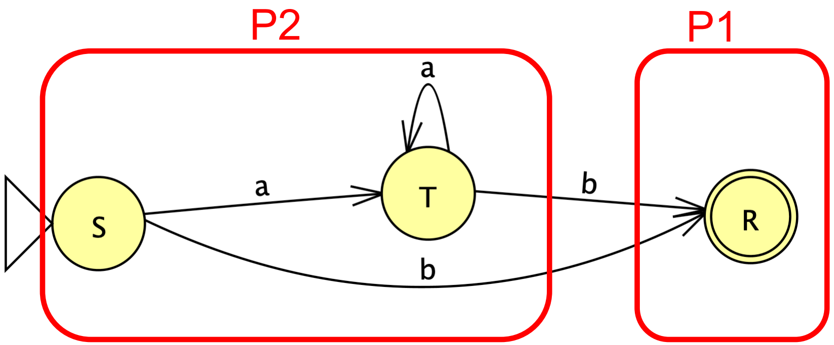


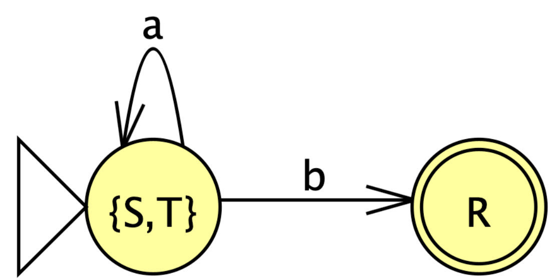
- Initial partitions

- Split partition

Minimizing DFA: Example 1

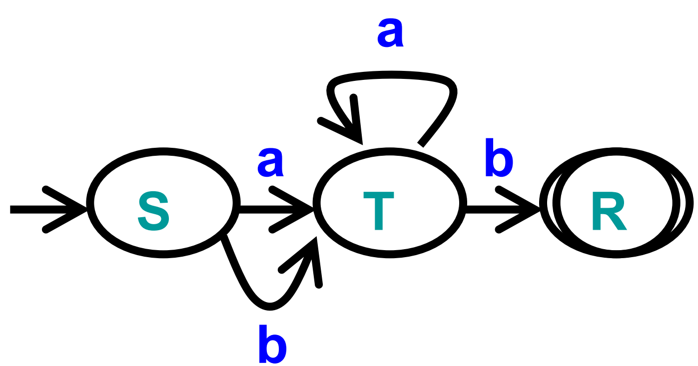
- DFA





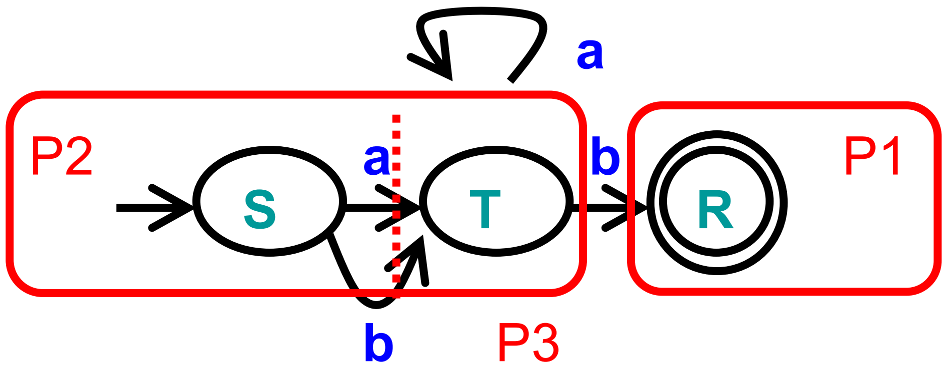
- Initial partitions
  - Accept { R } = P1
  - Reject { S, T } = P2
- Split partition? → Not required, minimization done
  - move(S,a) = T ∈ P2
    – move(S,b) = R ∈ P1
  - move(T,a) = T ∈ P2
    – move (T,b) = R ∈ P1

Minimizing DFA: Example 2



Minimizing DFA: Example 2

- DFA



DFA already minimal

- Initial partitions
  - Accept { R }= P1
  - Reject { S, T } = P2
- Split partition? → Yes, different partitions for B
  - move(S,a) = T ∈ P2
    - move(S,b) = T ∈ P2
  - move(T,a) = T ∈ P2
    - move (T,b) = R ∈ P1

Brzozowski's algorithm

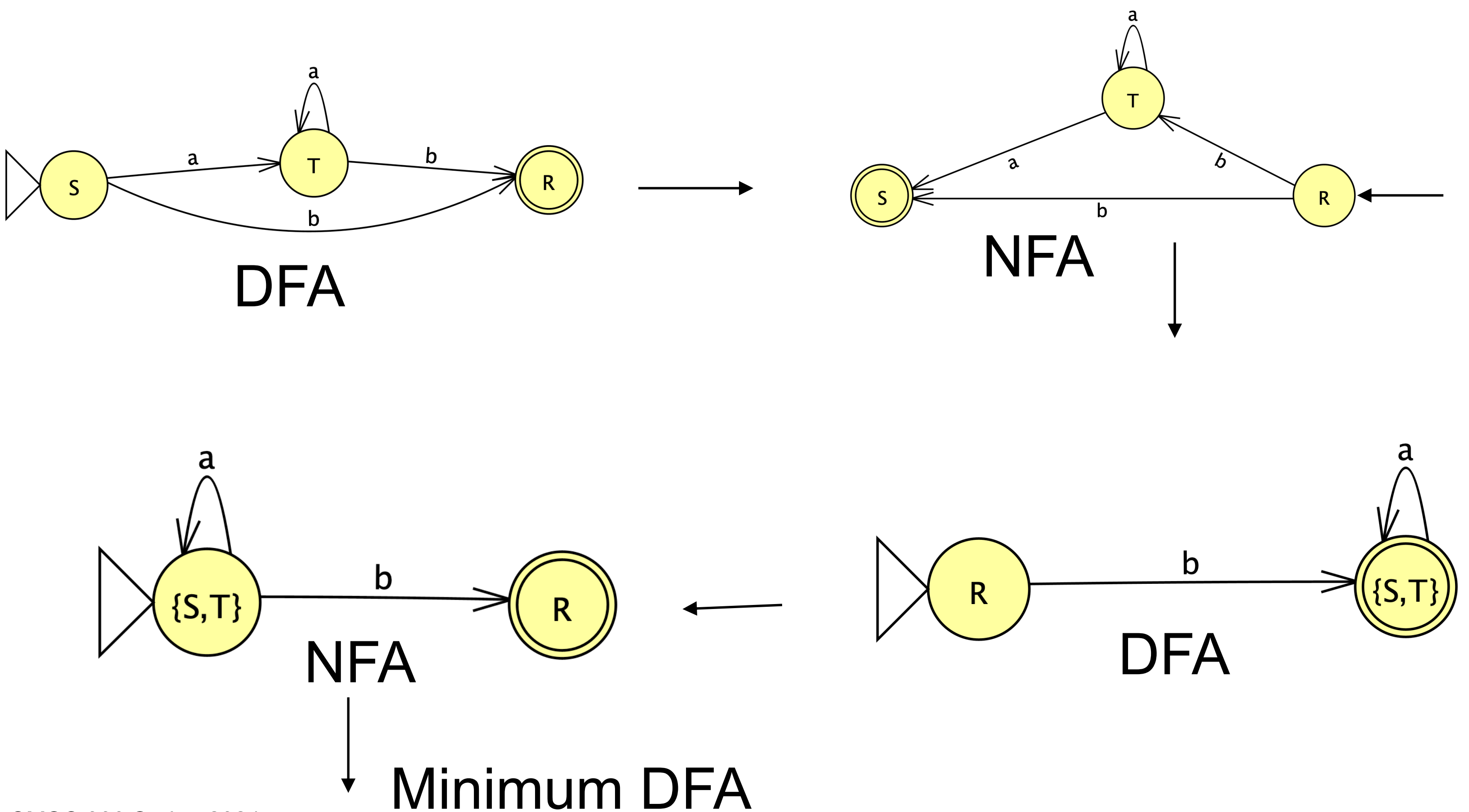
1. Given a DFA, reverse all the edges, make the initial state an accept state, and the accept states initial, to get an NFA

2. NFA-> DFA

3. For the new DFA, reverse the edges (and initial-accept swap) get an NFA

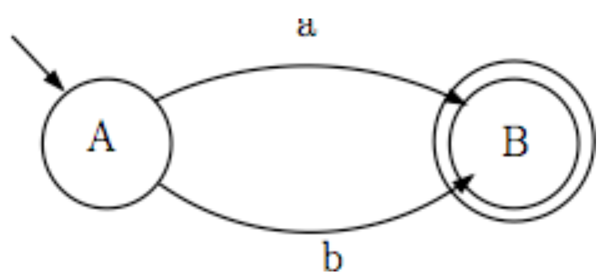
4. NFA -> DFA

Brzozowski's algorithm



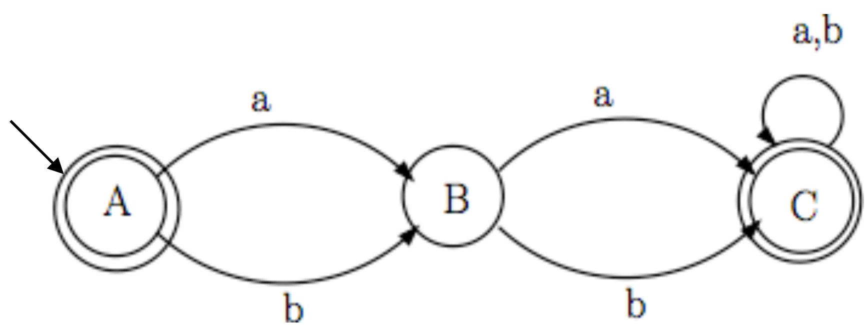
Complement of DFA

- Given a DFA accepting language L
  - How can we create a DFA accepting its complement?
  - Example DFA
    - $\Sigma = \{a,b\}$



Complement of DFA

- Algorithm
  - Add explicit transitions to a dead state
  - Change every accepting state to a non-accepting state & every non-accepting state to an accepting state
- Note this only works with DFAs
  - Why not with NFAs?



Summary of Regular Expression Theory

- Finite automata
  - DFA, NFA
- Equivalence of RE, NFA, DFA
  - RE $\rightarrow$ NFA
    - Concatenation, union, closure
  - NFA $\rightarrow$ DFA
    - ε-closure & subset algorithm
- DFA
  - Minimization, complementation