Compiler Design

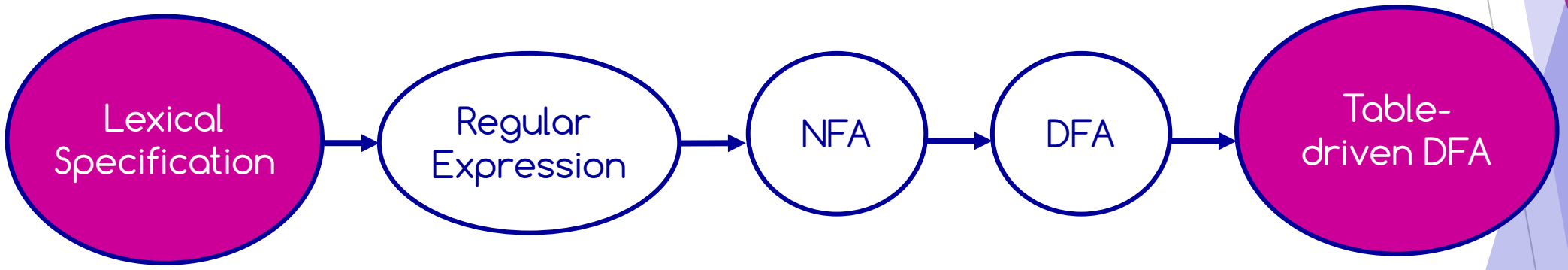
Lecture 4: Lexical Analysis III

Sahar Selim

Agenda

1. Conversion From a Regular Expression to Non-Deterministic Finite Automaton (NFA)
2. Transition Table
3. Conversion From NFA to DFA
4. Minimizing DFA

Implementing Lexical Analyzer

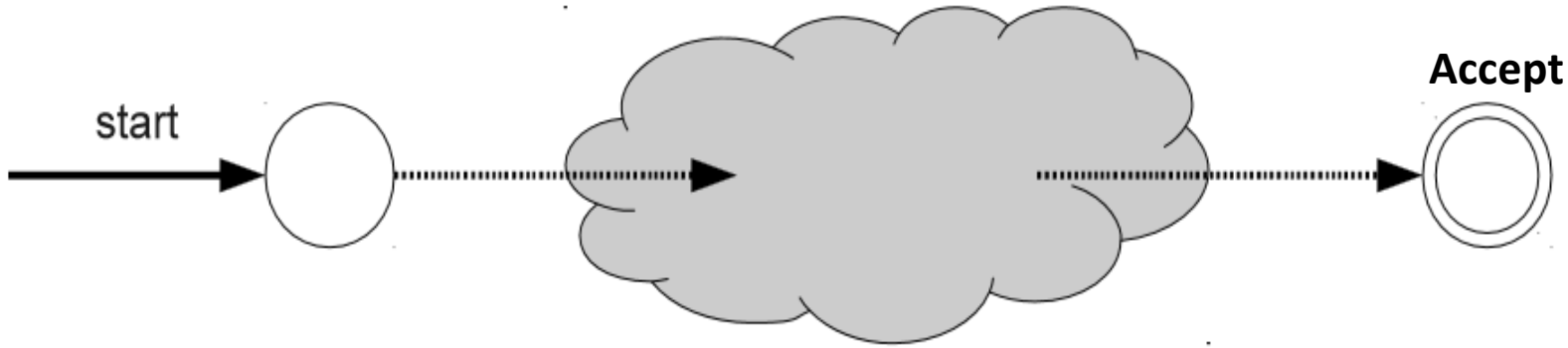


The steps of implementing the lexical analyzer of a compiler

From a Regular Expression to an NFA

From Regular Expressions to NFAs

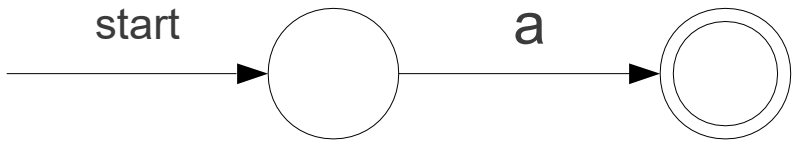
- Associate each regular expression with an NFA with the following properties:
  - There is exactly one accepting state.
  - There are no transitions out of the accepting state.
  - There are no transitions into the starting state.



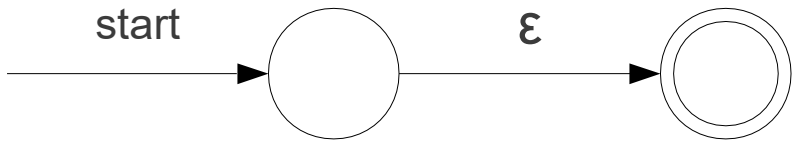
Thompson's Construction

- Use ε-transitions
  - to *"glue together"* the machine of each piece of a regular expression
  - to form a machine that corresponds to the whole expression

Base Cases



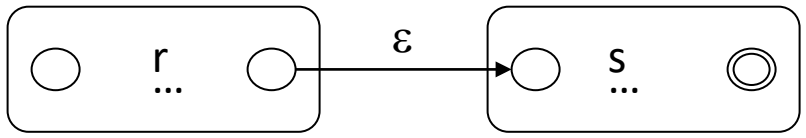
Automaton for single character **a**



Automaton for ε

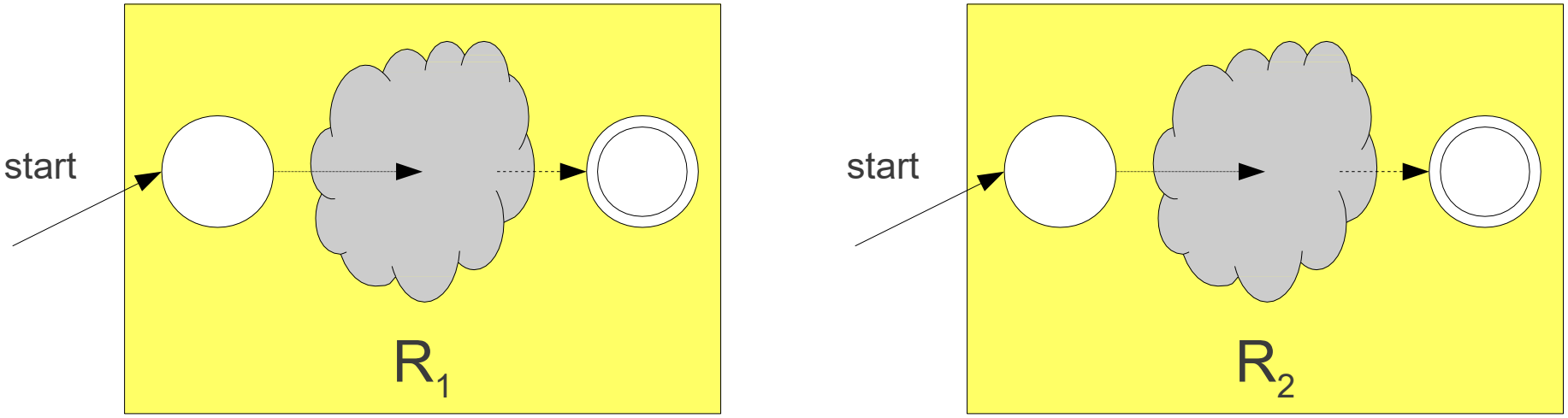
Thompson's Construction: Concatenation (A o B)

- Construct an NFA equal to *rs*
  - *To* connect the accepting state of the machine of *r* to the start state of the machine of *s* by an ε-transition.
  - The start state of the machine of *r* as its start state and the accepting state of the machine of *s* as its accepting state.
  - This machine accepts L(rs) = L(r)L(s) and so corresponds to the regular expression rs.

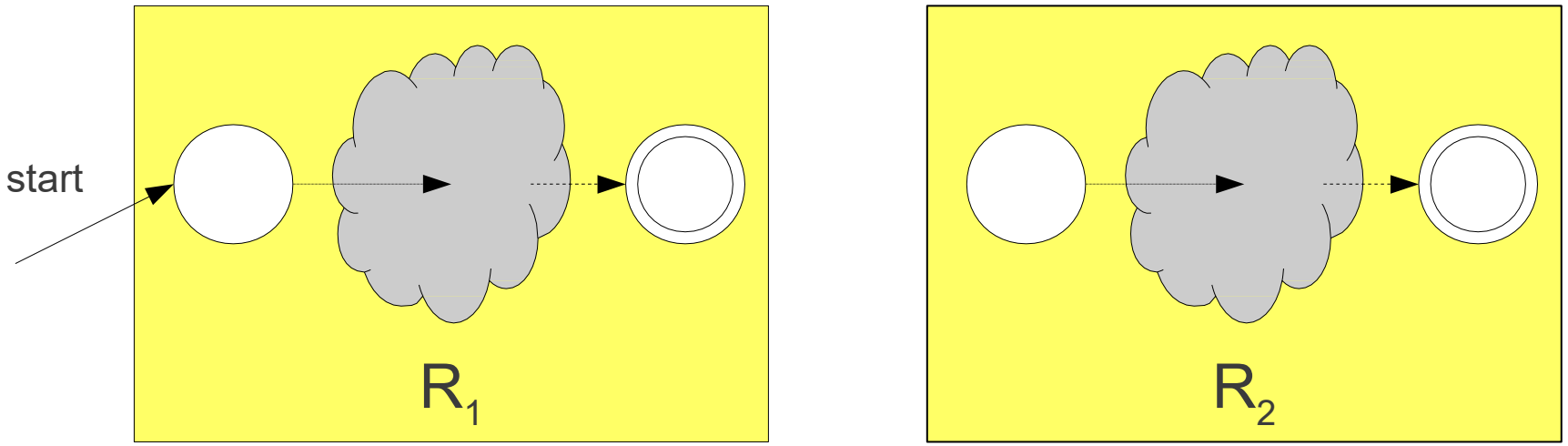


Construction for R_1R_2

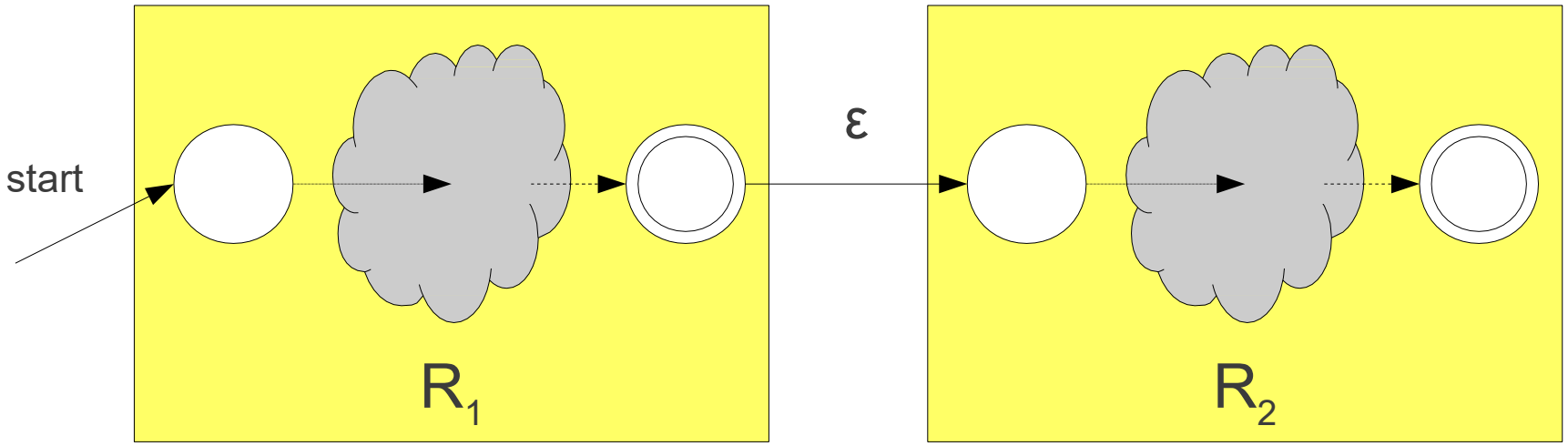
Construction for R_1R_2



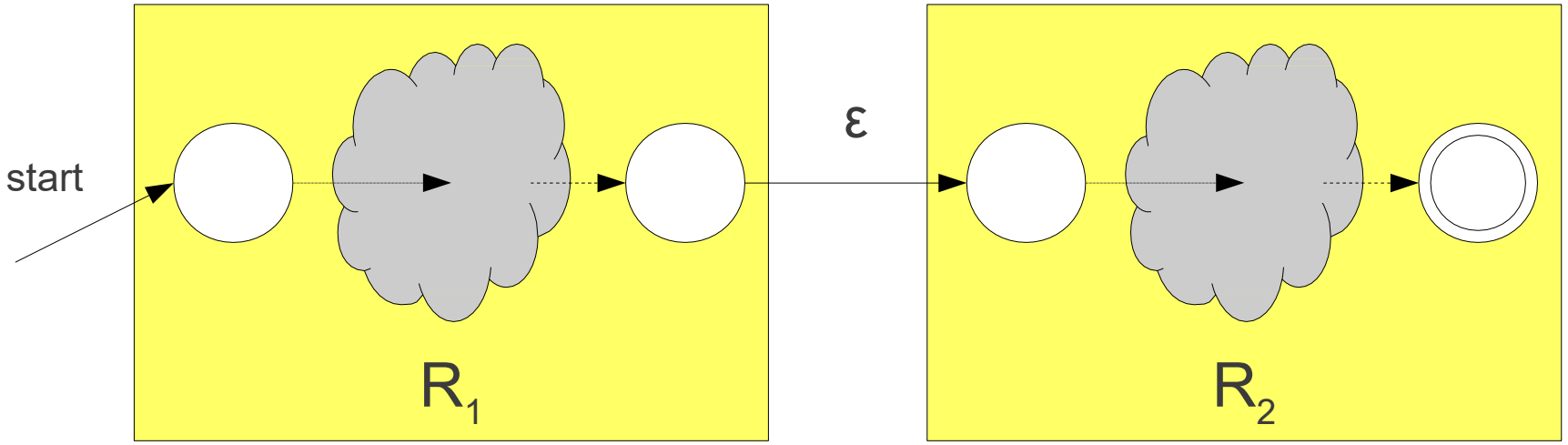
Construction for R_1R_2



Construction for R_1R_2



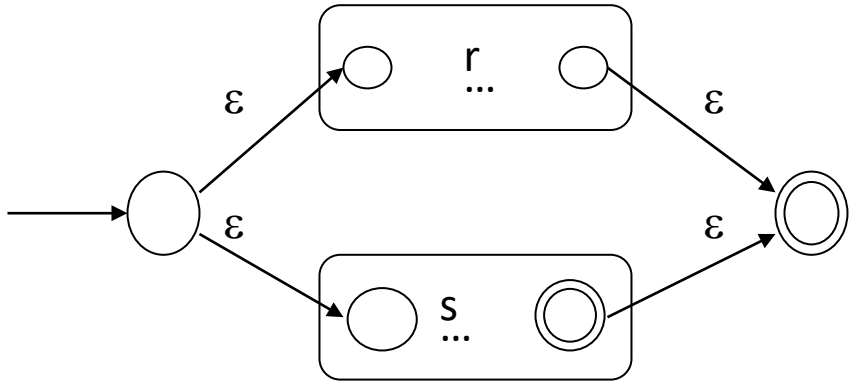
Construction for R_1R_2



Thompson's Construction: Alternatives (A ∪ B)

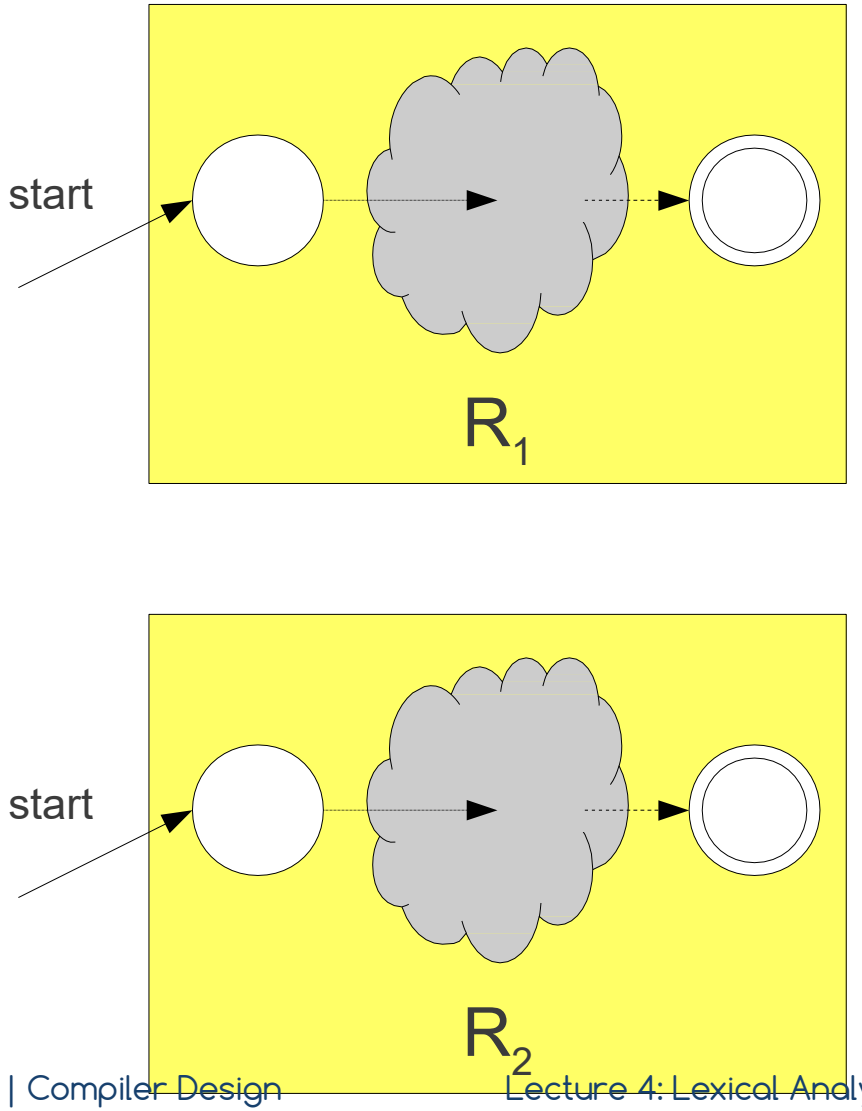
- Construct an NFA equal to **r | s**
  - Add a new start state and a new accepting state and connect them using ε-transitions.
  - Clearly, this machine accepts the language

L(r|s) = L(r) ∪ L(s), and so corresponds to the regular expression r|s.

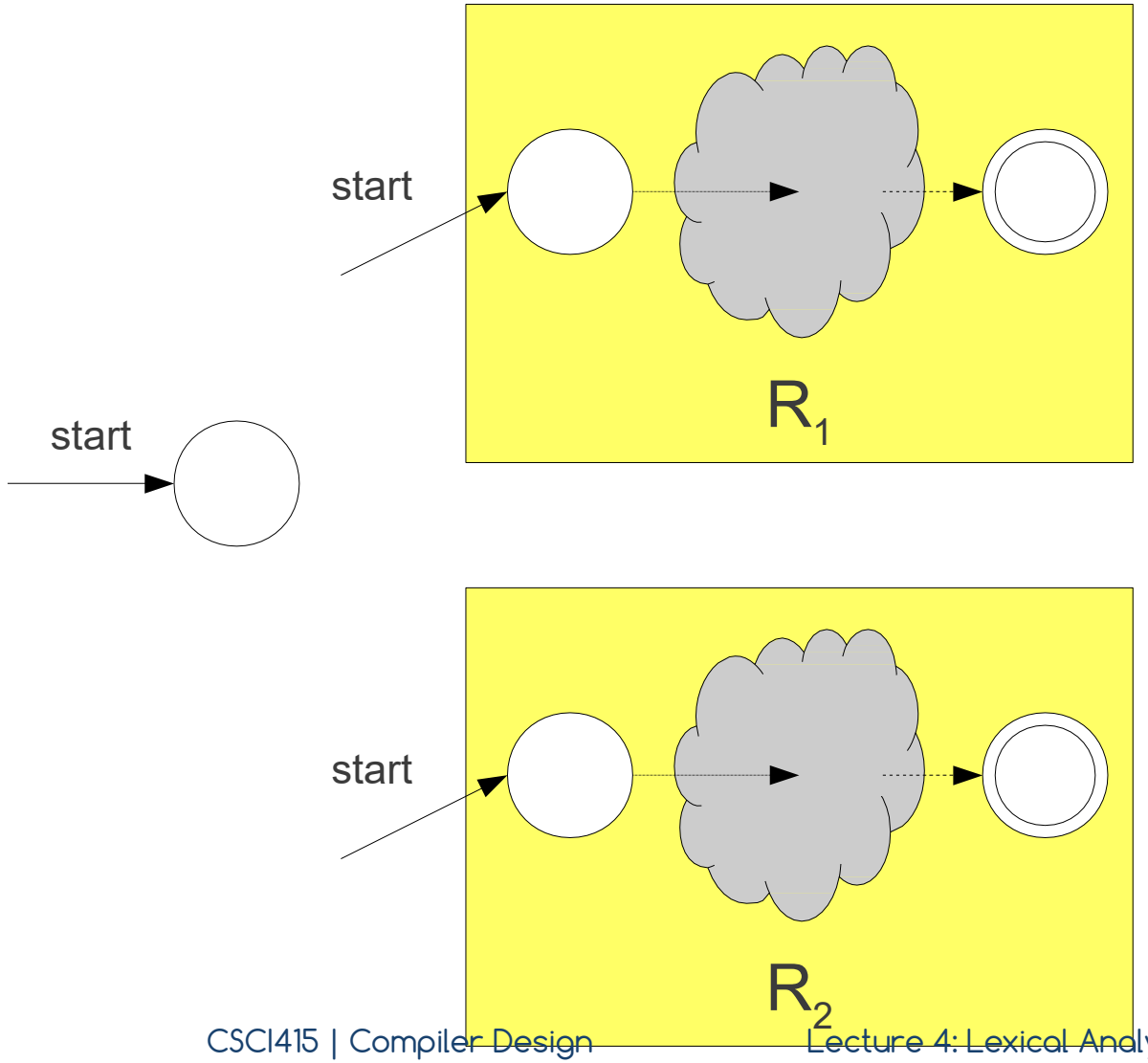


Construction for $R_1 | R_2$

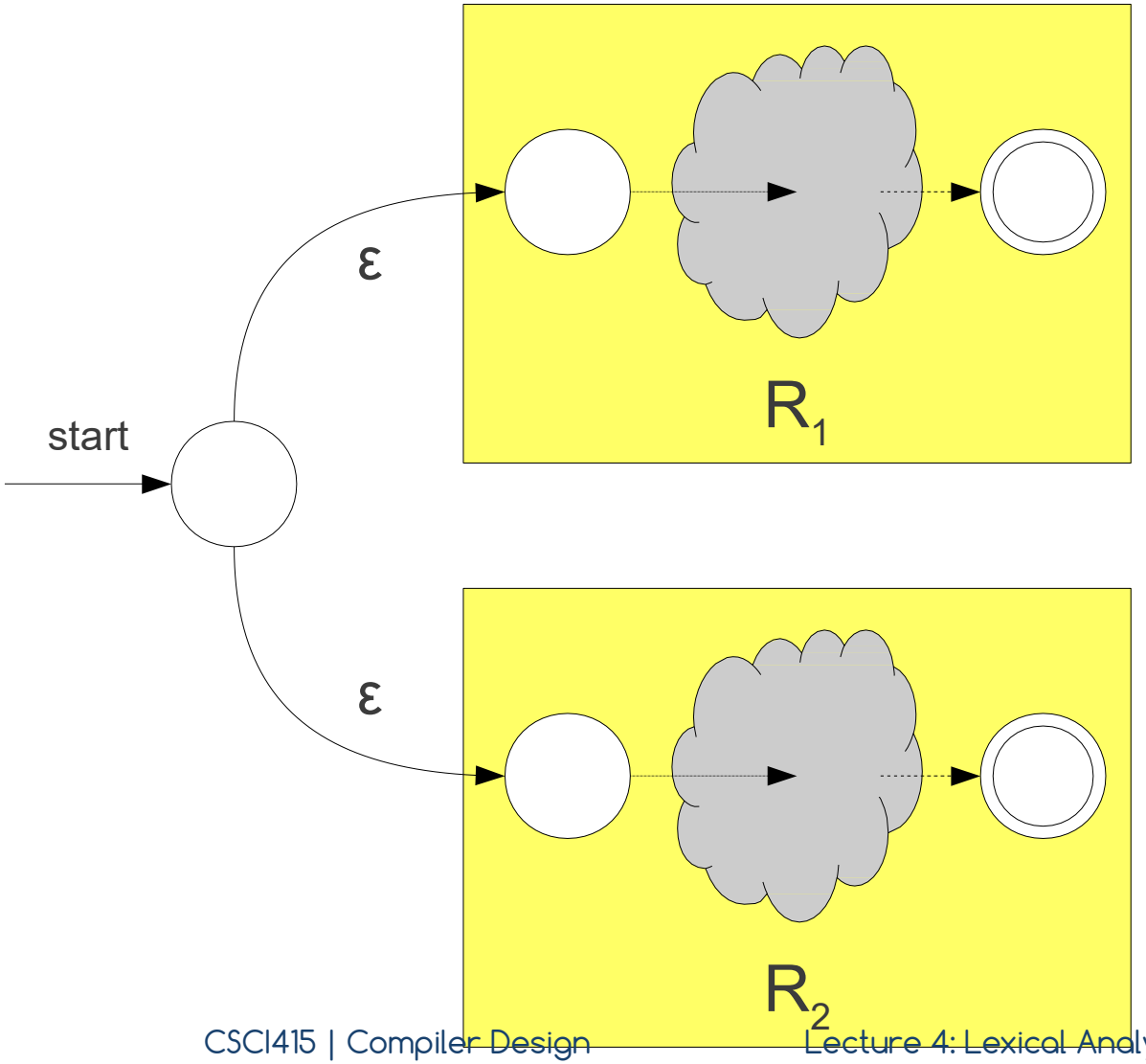
Construction for $R_1 | R_2$



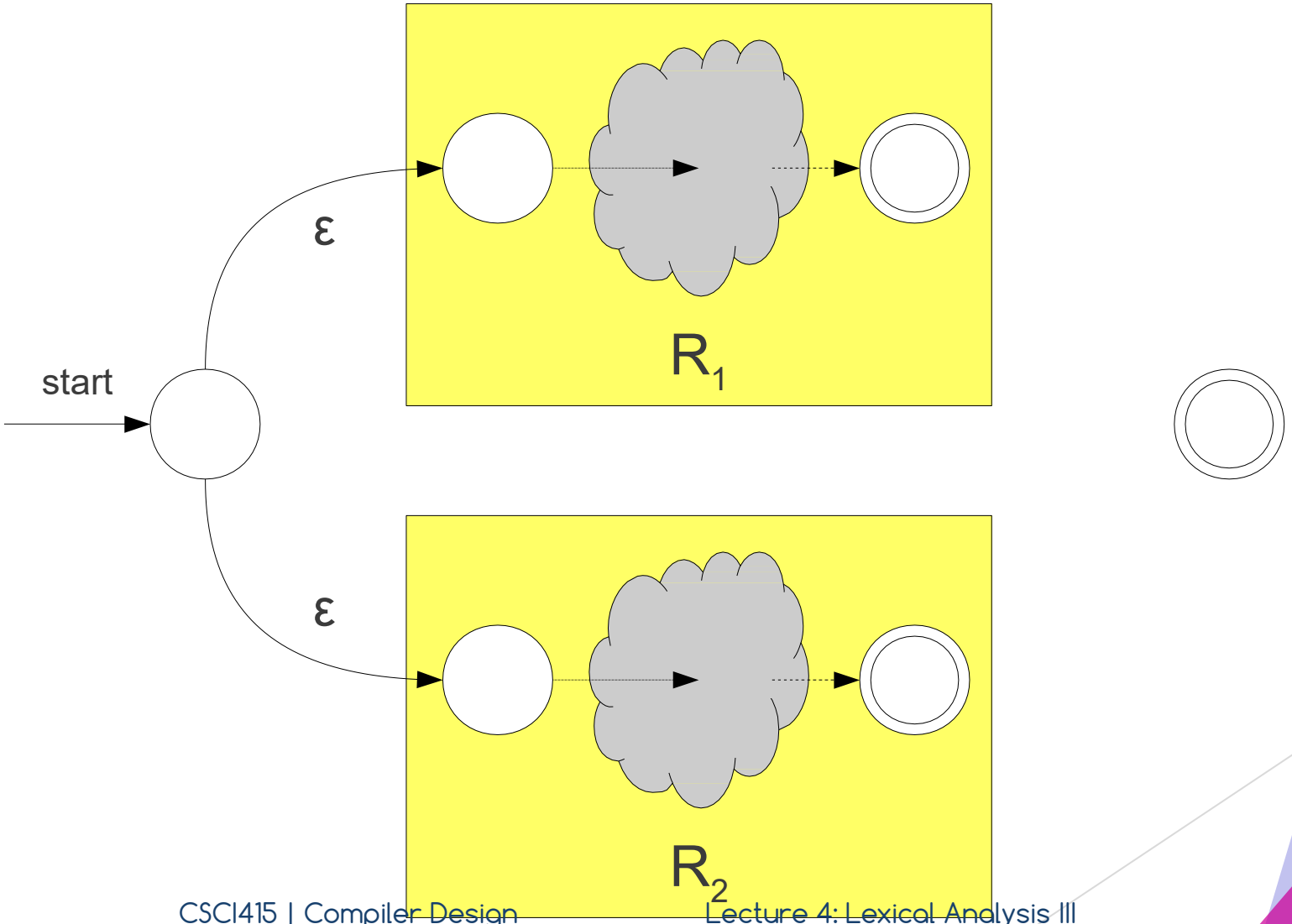
Construction for $R_1 | R_2$



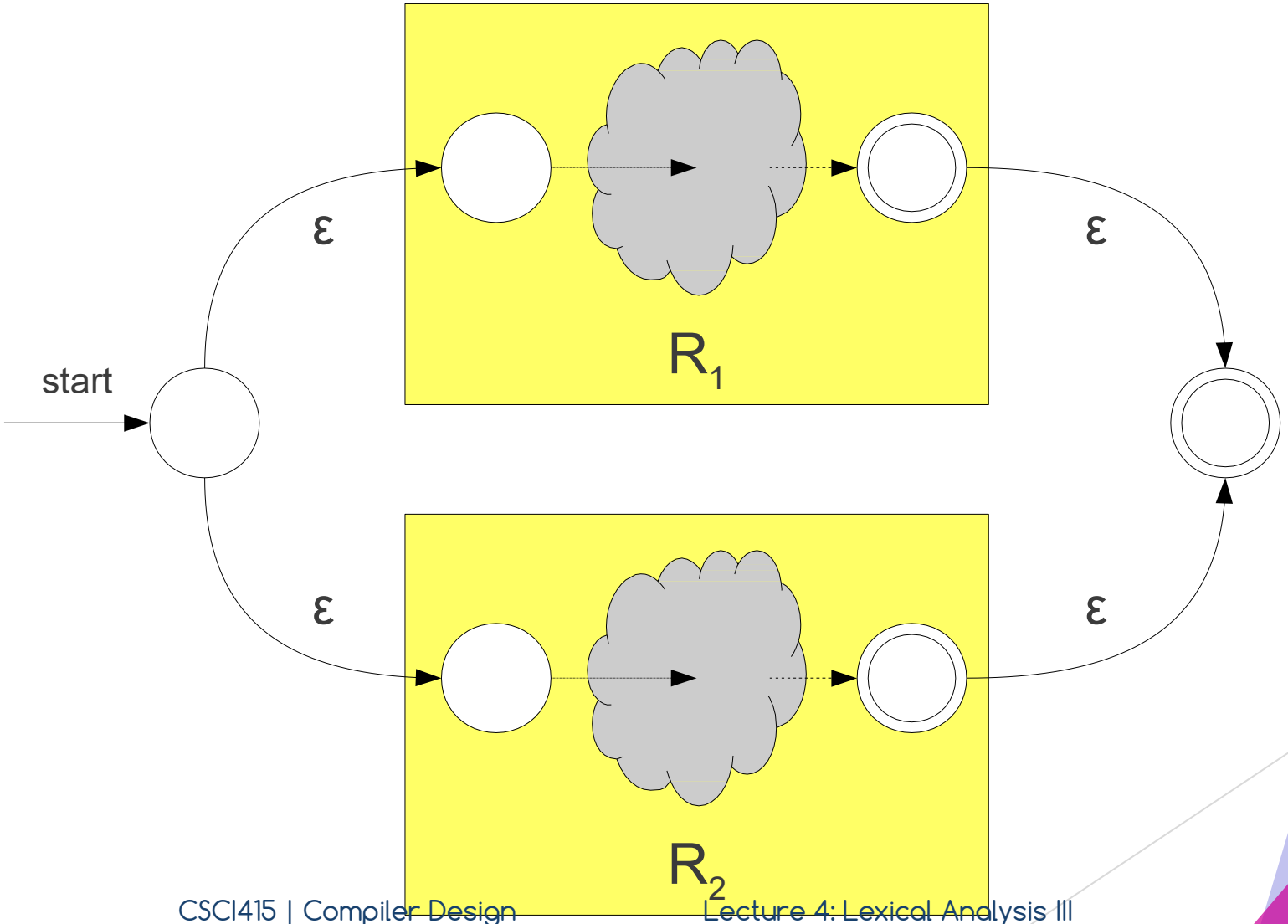
Construction for $R_1 | R_2$



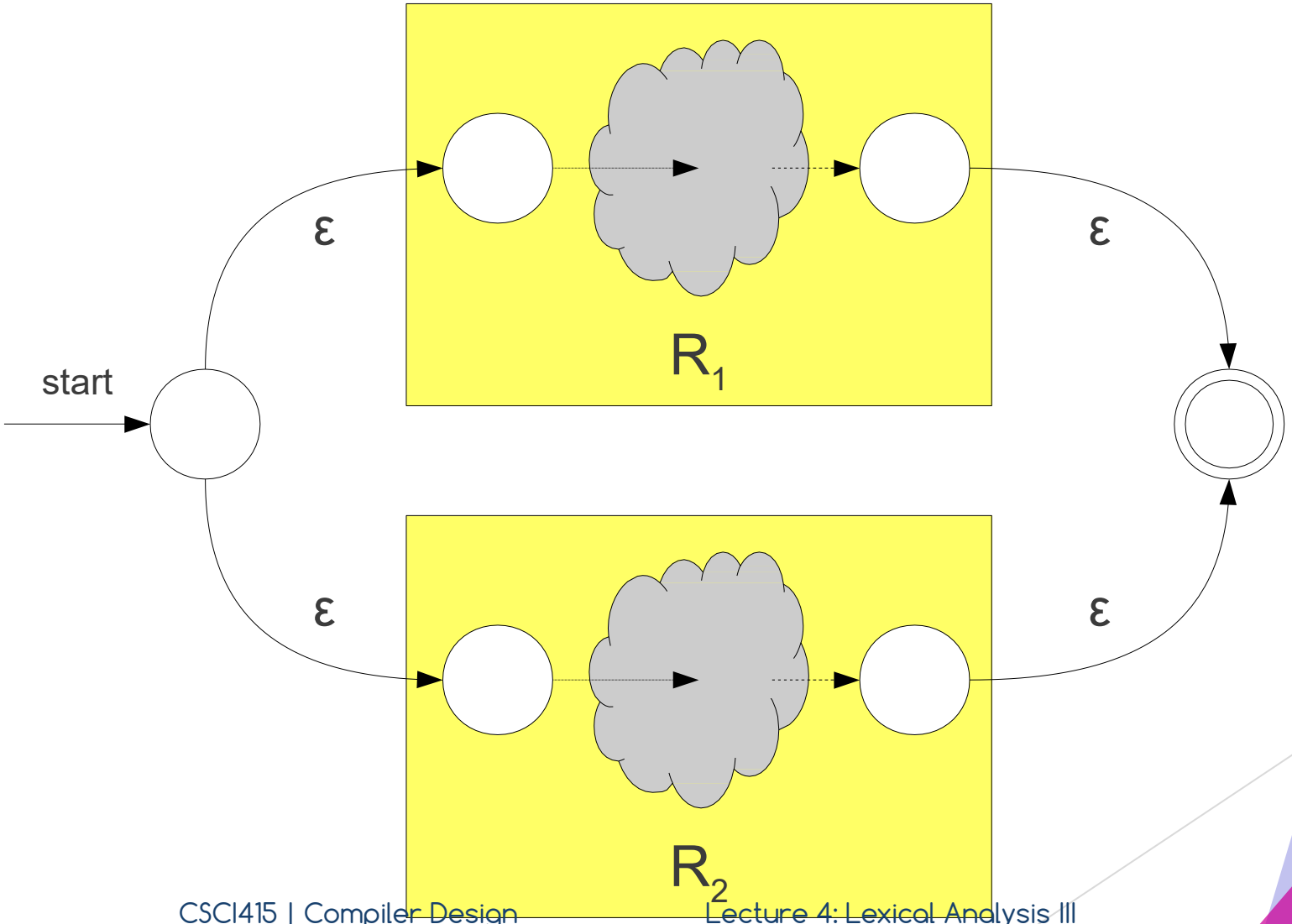
Construction for $R_1 | R_2$



Construction for $R_1 | R_2$

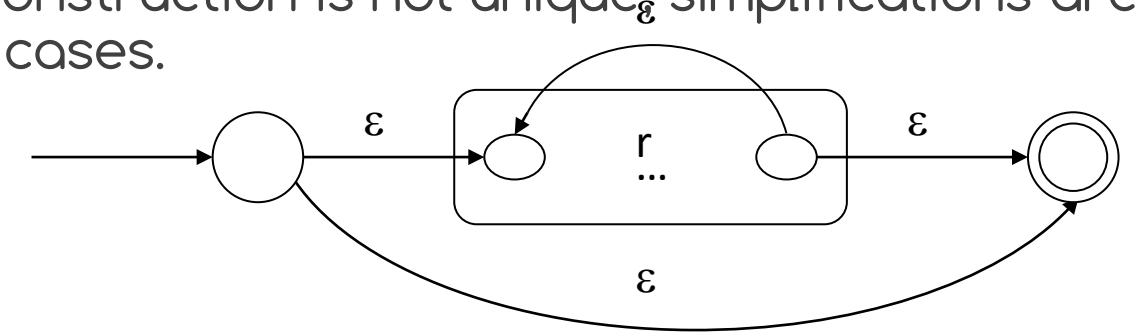


Construction for $R_1 | R_2$



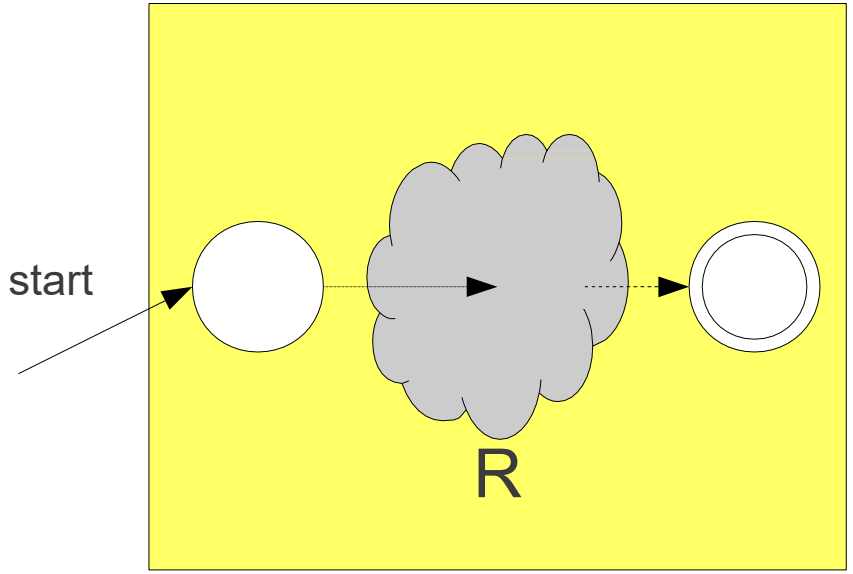
Thompson's Construction: Repetition

- Given a machine that corresponds to *r*, Construct a machine that corresponds to **r***
  - Add two new states, a start state and an accepting state.
  - The repetition is afforded by the new ε-transition from the accepting state of the machine of *r* to its start state.
  - Draw an ε-transition from the new start state to the new accepting state.
  - This construction is not unique, simplifications are possible in the many cases.

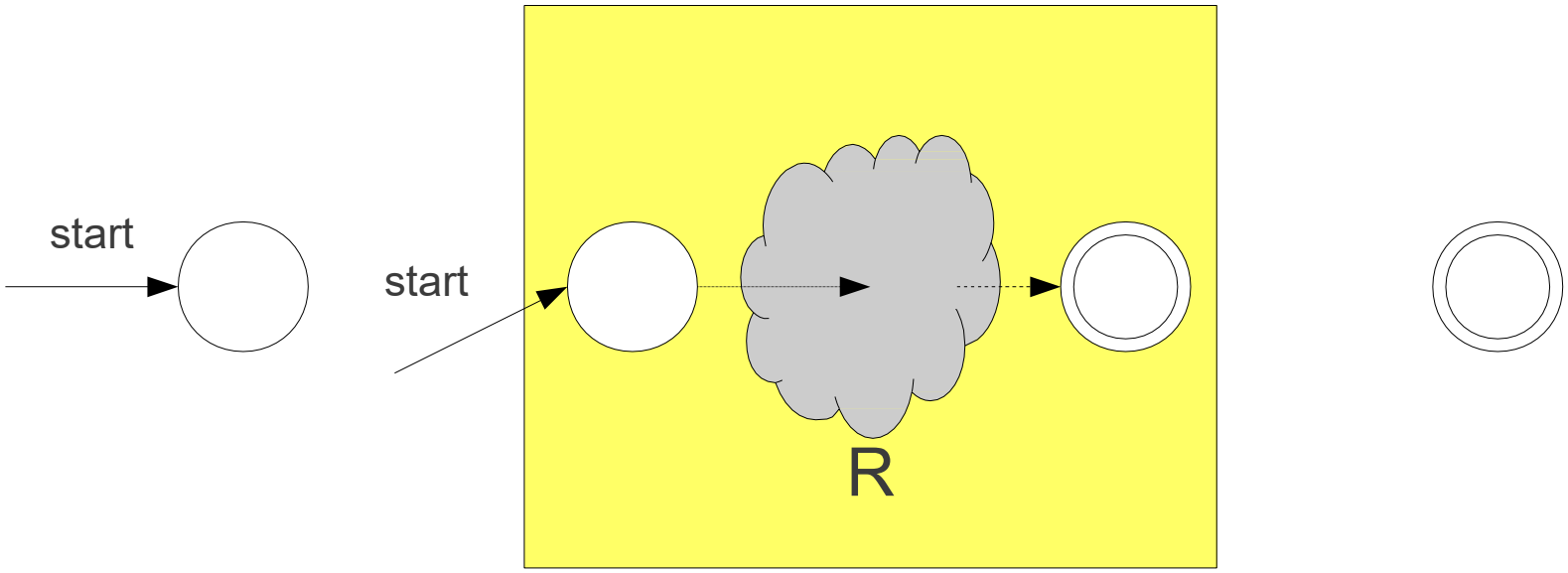


Construction for R*

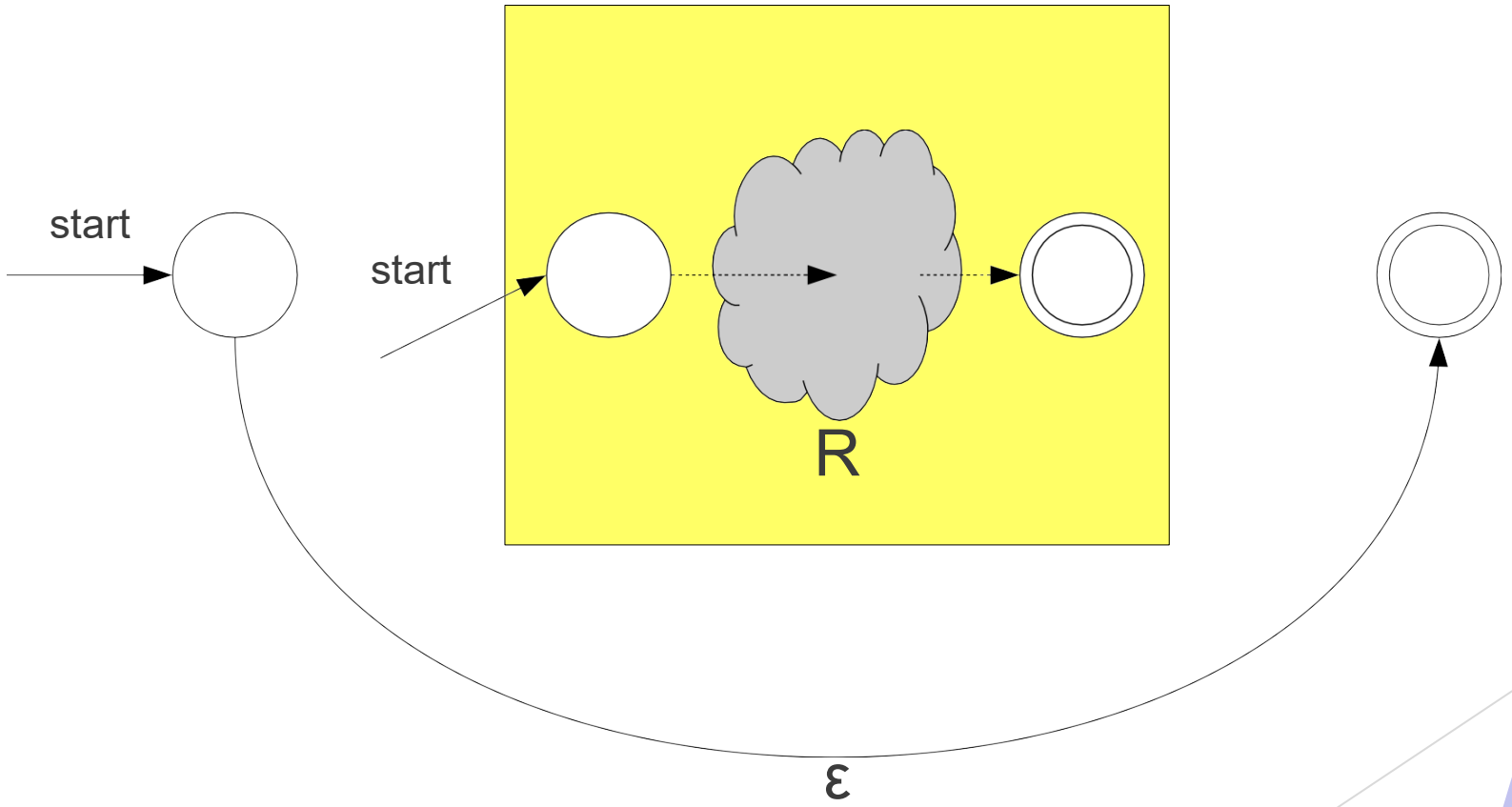
Construction for R*



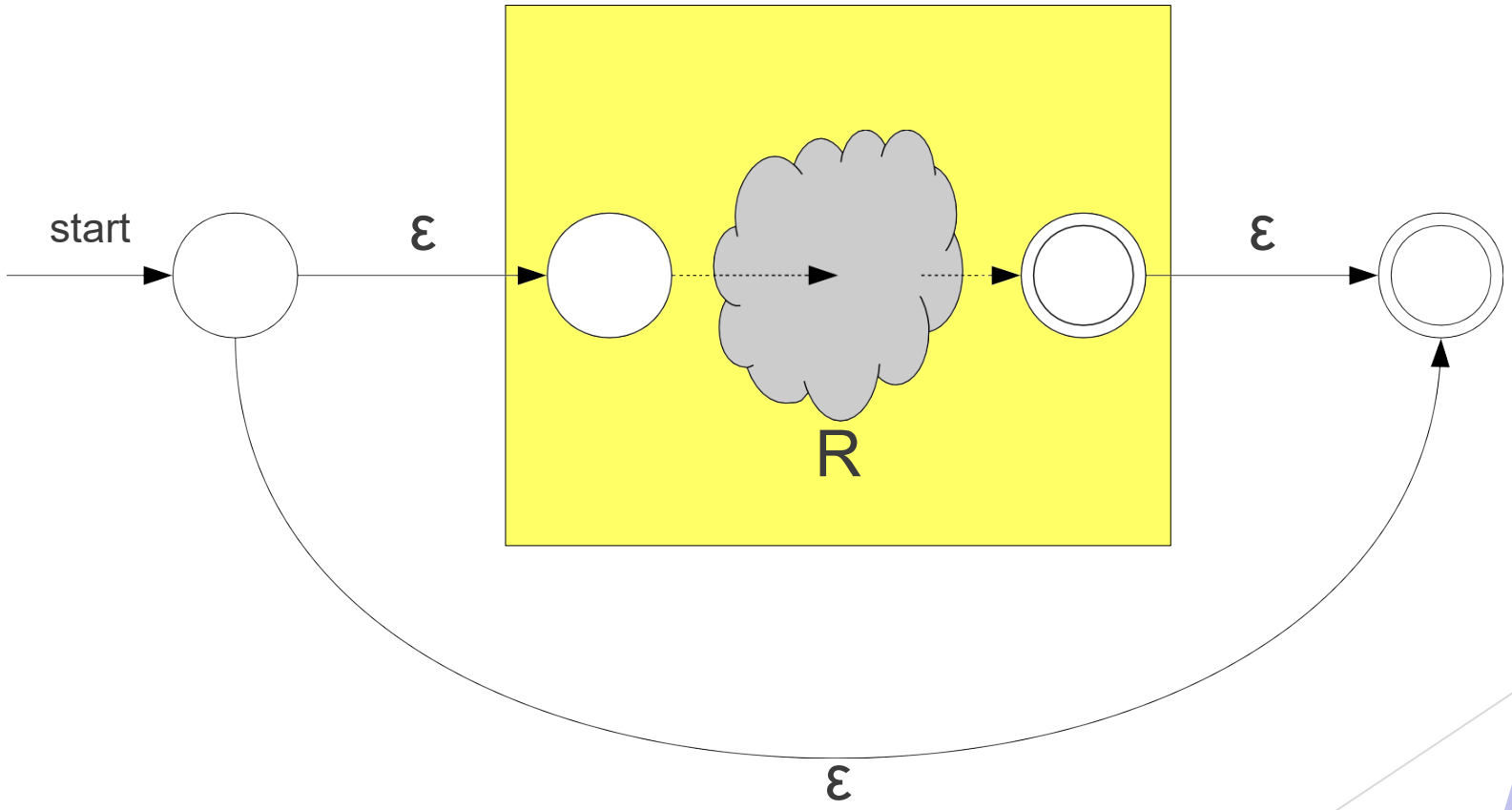
Construction for R*



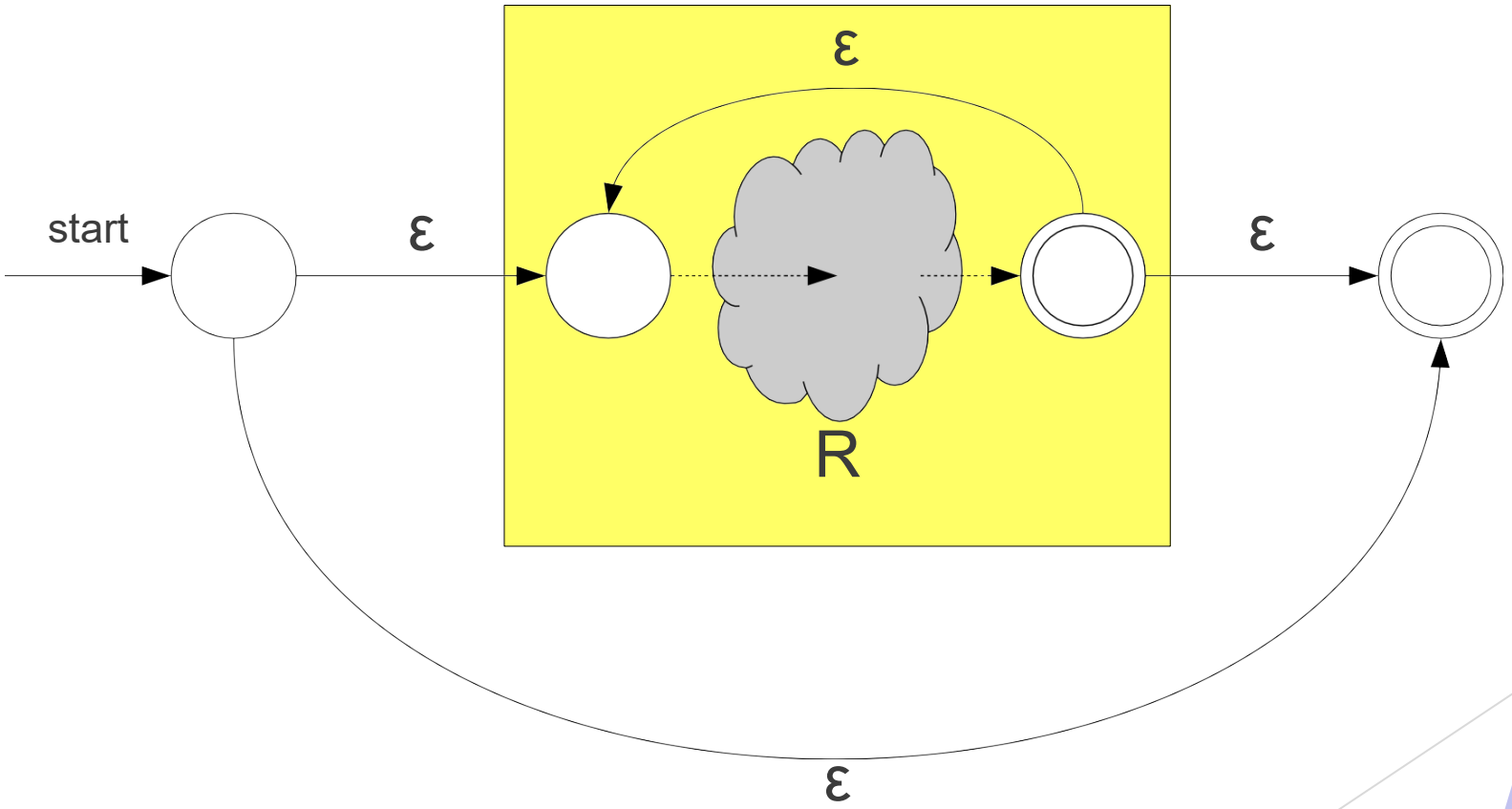
Construction for R*



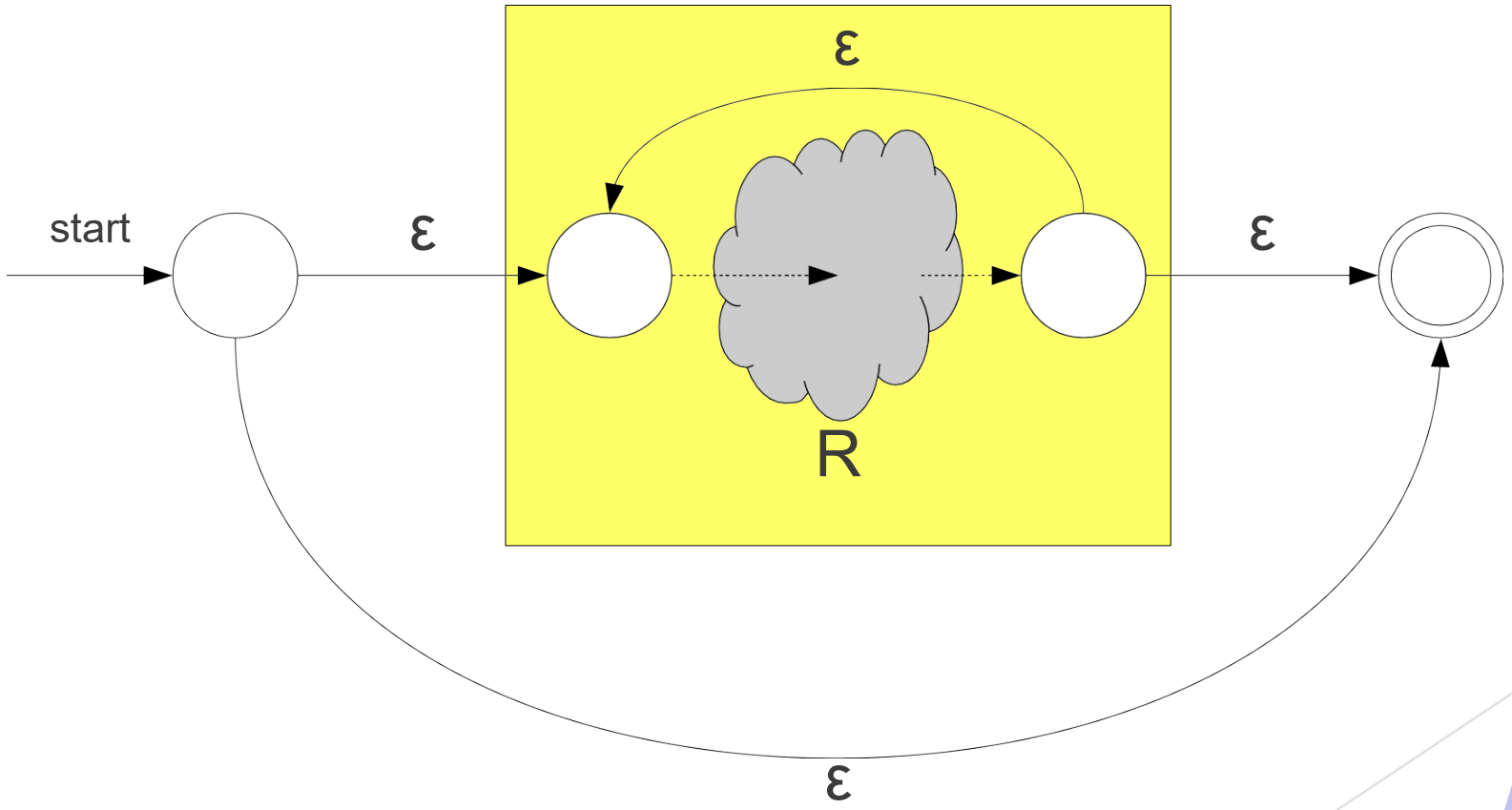
Construction for R*



Construction for R*



Construction for R*

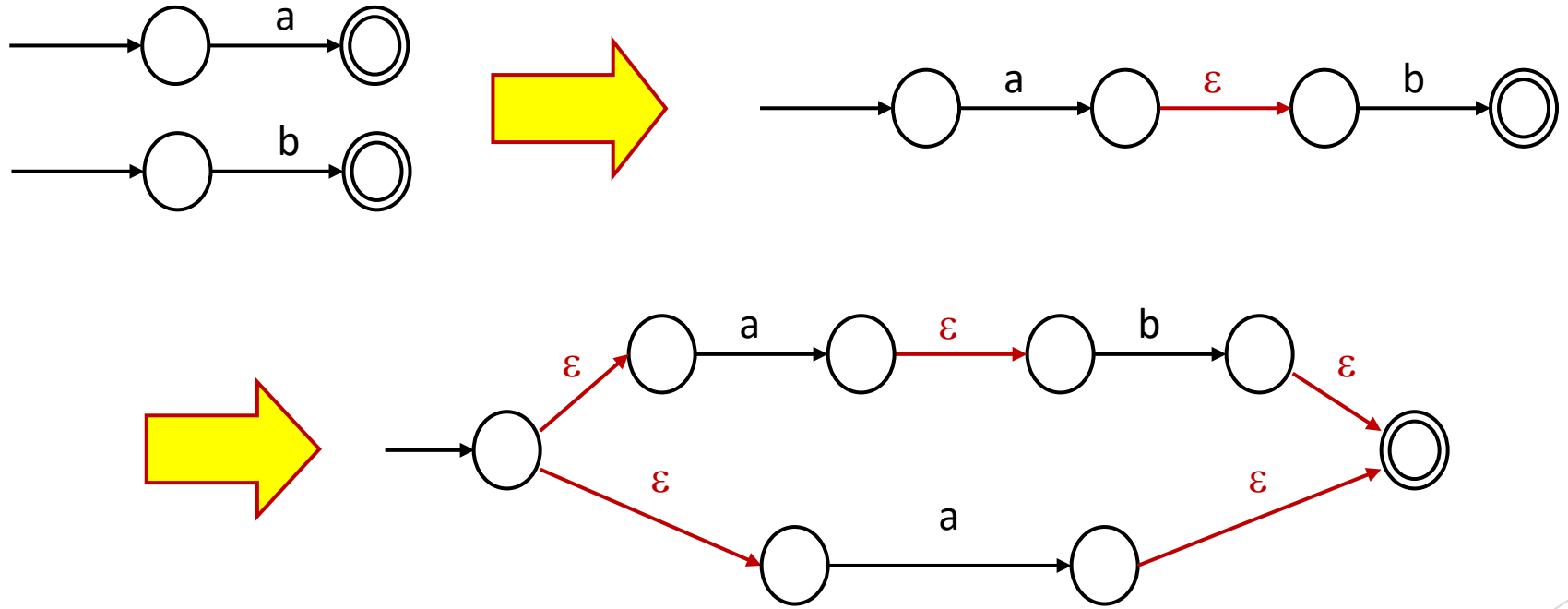


Example 1

Translate regular expression **ab|a** into NFA

Example 1

Translate regular expression **ab|a** into NFA

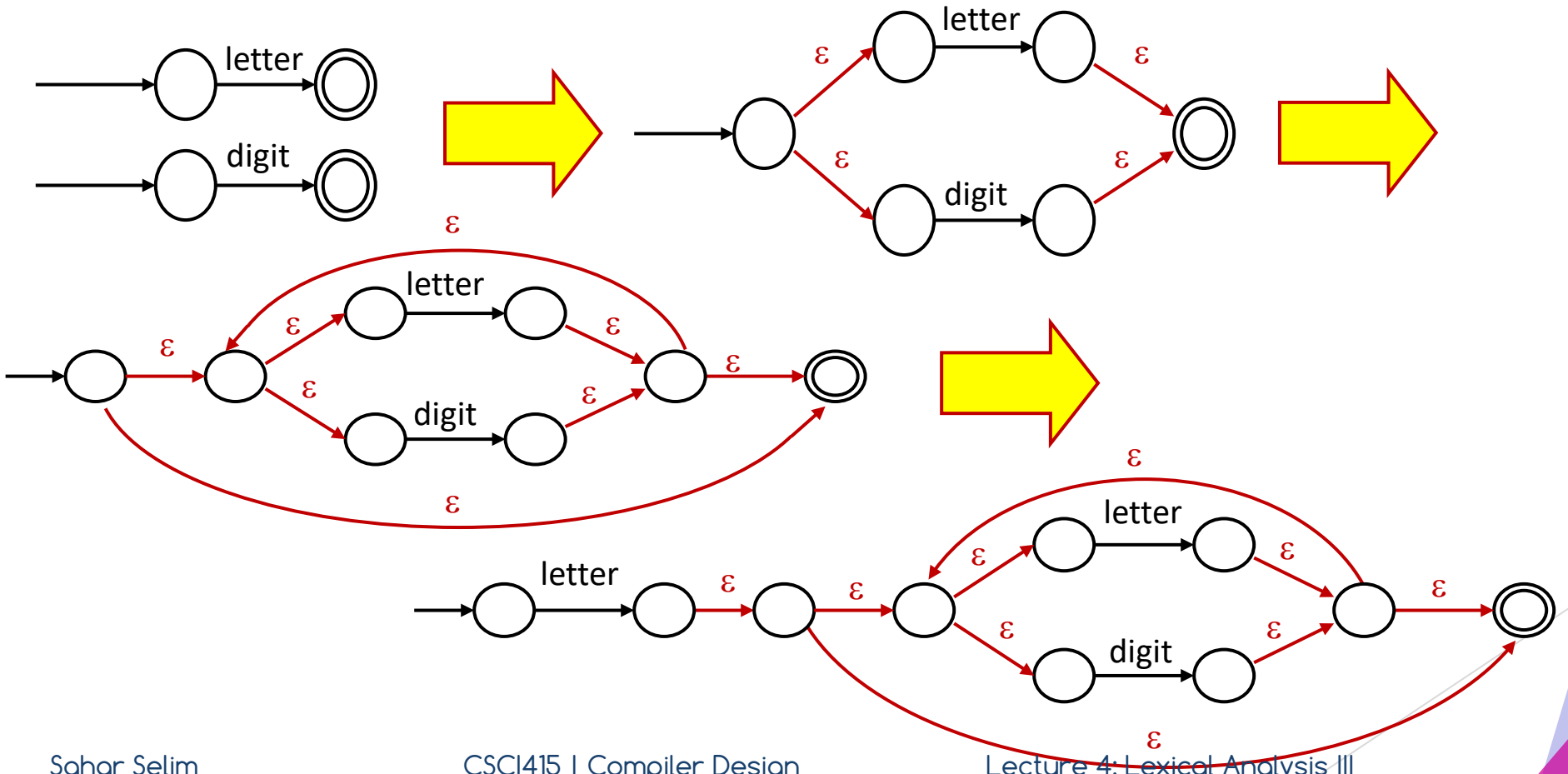


Example 2

Translate regular expression *letter(letter|digit)** into NFA

Example 2

Translate regular expression *letter(letter|digit)** into NFA



Example 3

- Construct the NFA for the regular expression

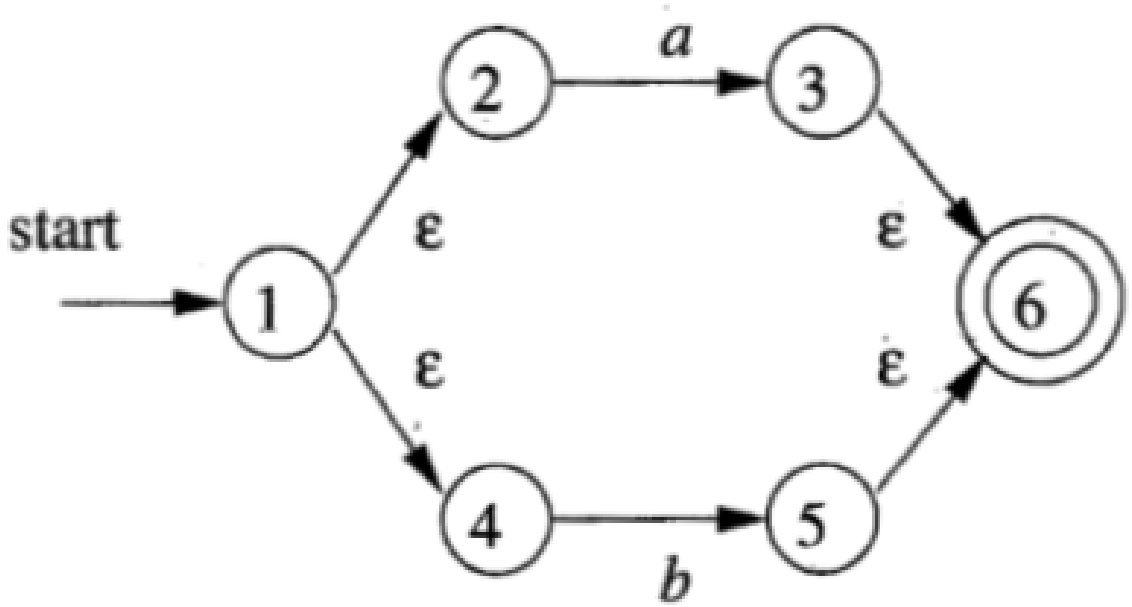
  (a | b)*abb

Example 3

- Construct the NFA for the regular expression

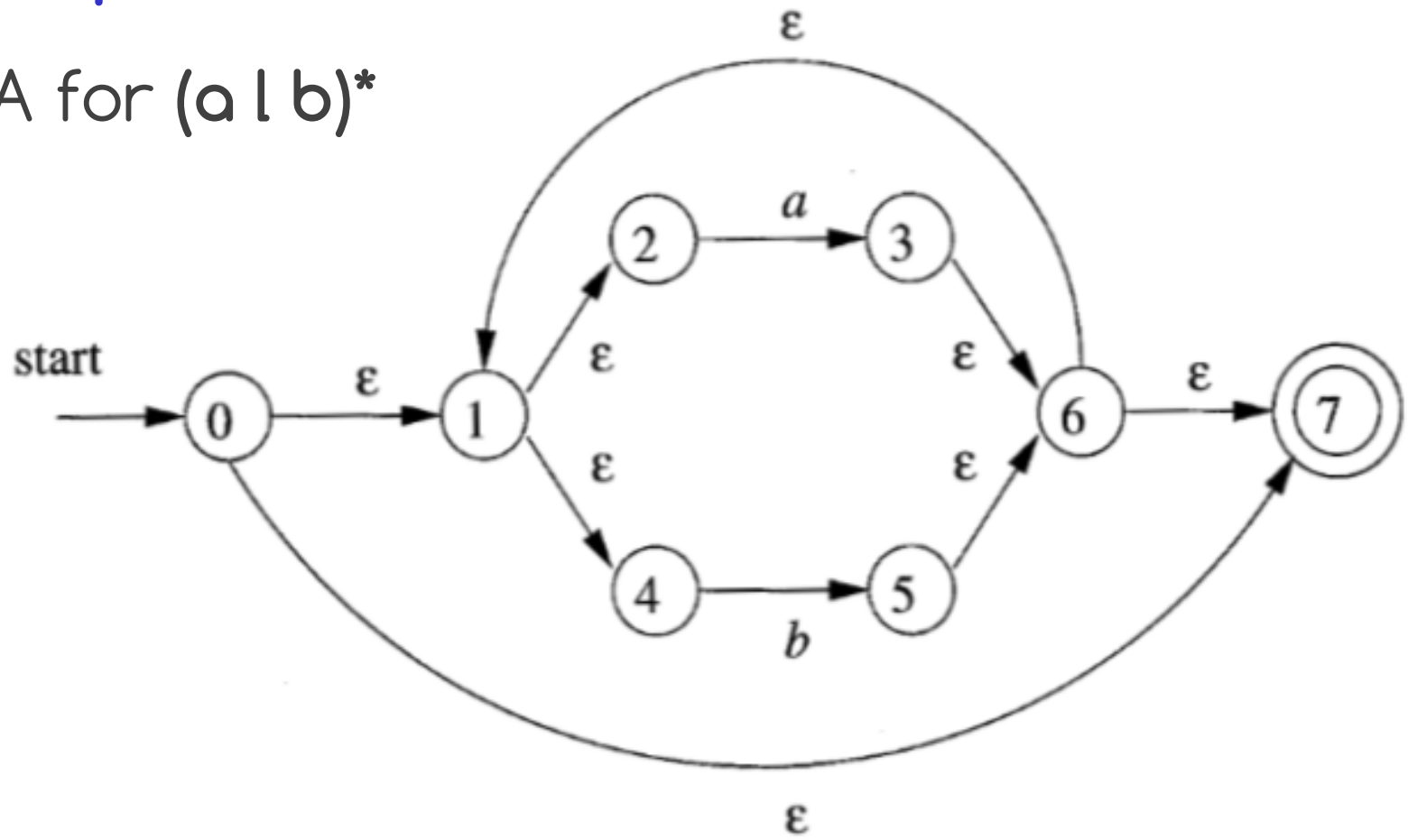
  (a | b)*abb

- NFA for a | b



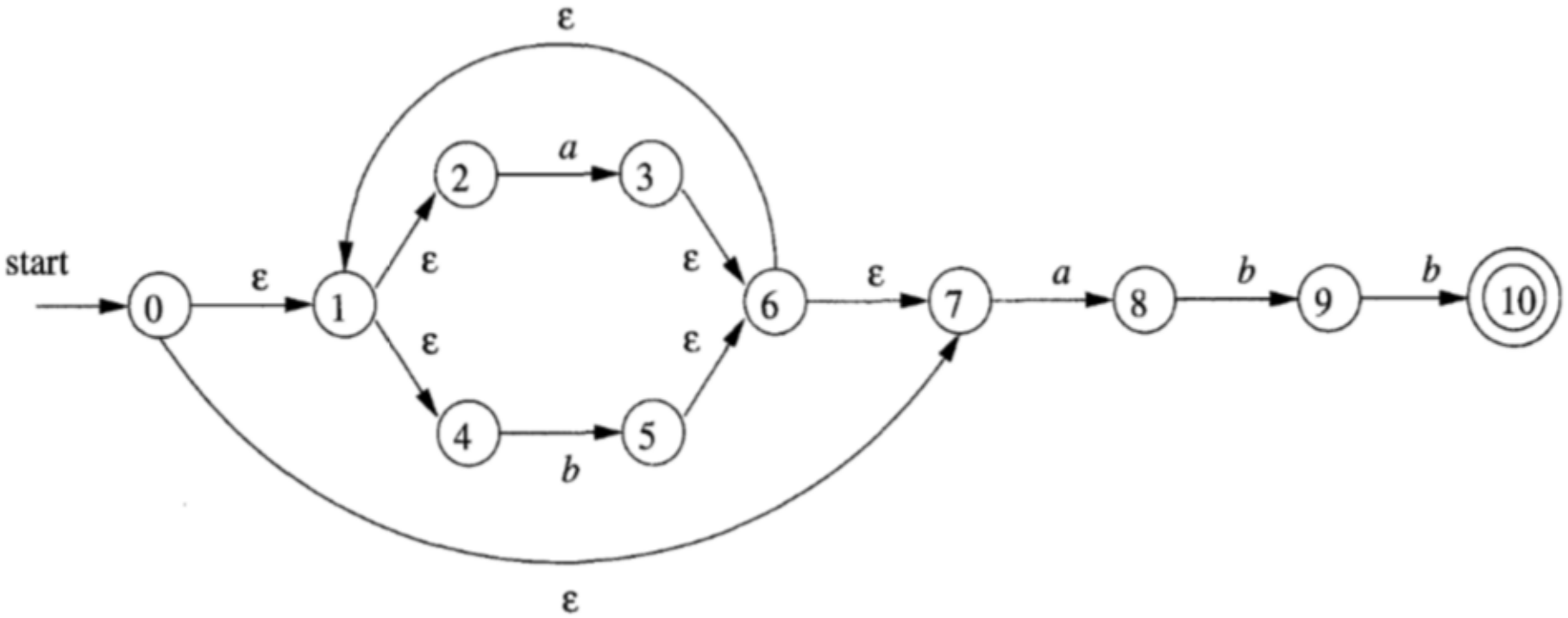
Example 3

- NFA for (a | b)*



Example 3

- NFA for (a|b)*abb



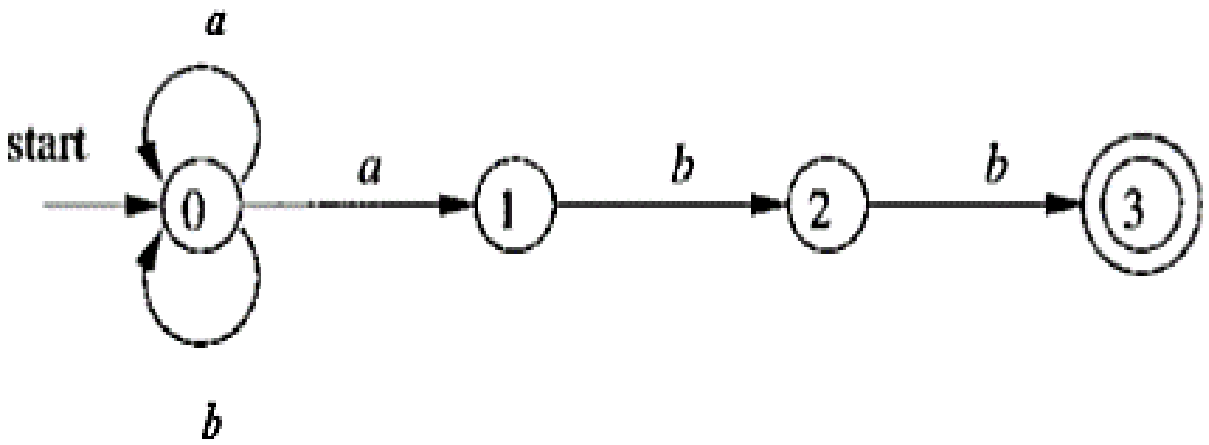
Transition Table

Transition Tables

- We can also represent an FSA by a **transition table**, whose **rows** correspond to **states**, and whose **columns** correspond to the input **symbols** and **ε**
- The entry for a given state and input is the value of the transition function applied to those arguments.
- If the transition function has no information about that state-input pair, we put 0 or φ in the table for the pair.

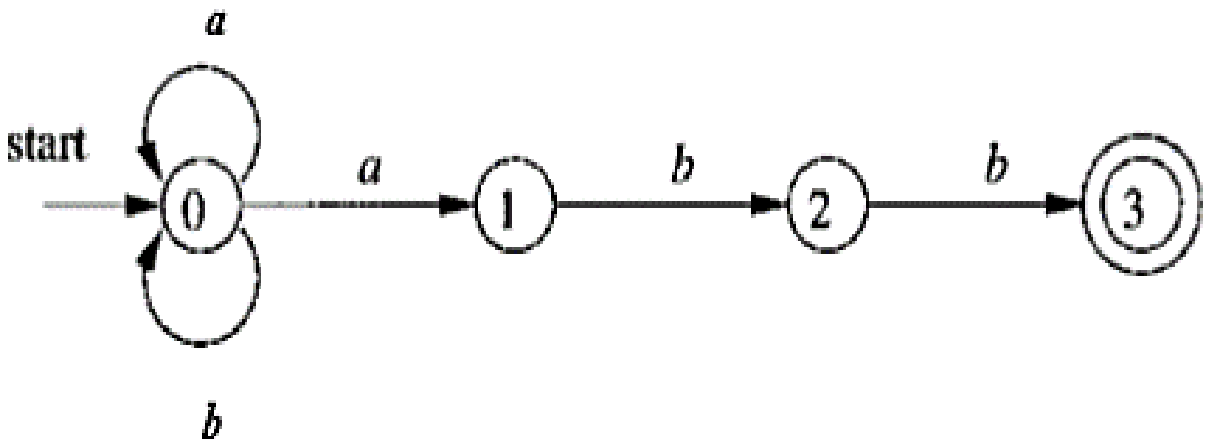
Example

- The transition graph for an FSA recognizing the language of regular expression **(a|b)*abb**



Example

- The transition graph for an FSA recognizing the language of regular expression **(a|b)*abb**



STATE	*a*	*b*	*e*
0	{0, 1}	{0}	0
1	0	{2}	0
2	0	{3}	0
3	0	0	0


Q
A

3 NFA to DFA

NFA to DFA

- A DFA is like an NFA, but with tighter restrictions
  - Every state must have exactly one transition defined for every letter.
  - ε-moves are not allowed.

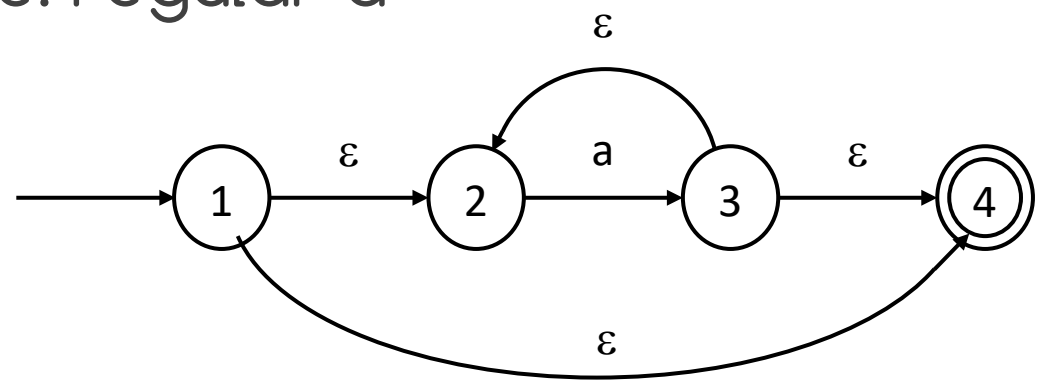
Goal and Methods

- Goal
  - Given an arbitrary NFA, construct an equivalent DFA. (i.e., one that accepts precisely the same strings)
- Some methods
  1. Eliminating ε-transitions
     - **ε-closure**: the set of all states reachable by ε-transitions from a state or states
  2. Eliminating multiple transitions from a state on a single input character.
     - Keeping track of the set of states that are reachable by matching a single character
  - Both these processes lead us to *consider sets of states instead of single states.* Thus, it is not surprising that the DFA we construct has sets of states of the original NFA as its states.

Subset Construction Algorithm

- The ε-closure of a Set of states:
  - The ε-closure of a single state ***s*** is the set of states reachable by a series of zero or more ε-transitions, and we write this set as ***ś***.

- Example: regular a*



The *Subset Construction* Algorithm

(1) Compute the ε-closure of the start state of *M;* to obtain new state $\overline{M}$.

(2) For this set, and for each subsequent set, compute transitions on characters *a* as follows.

Given a set *S* of states and a character *a* in the alphabet,

Compute the set

$$S'_a = \{\, t \mid \text{for some } s \text{ in S there is a transition from } s \text{ to } t \text{ on } a \,\}.$$

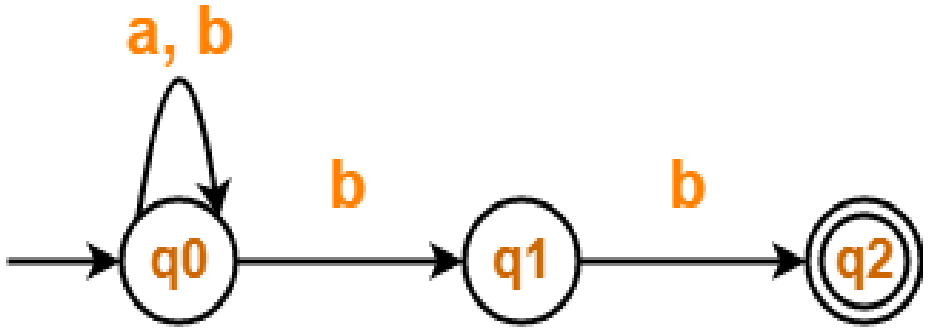
Then, compute $\overline{S_a}'$, the ε-closure of S'_a.

This defines a new state in the subset construction, together with a new transition S→ $\overline{S_a}'$.

(3) Continue with this process until no new states or transitions are created.

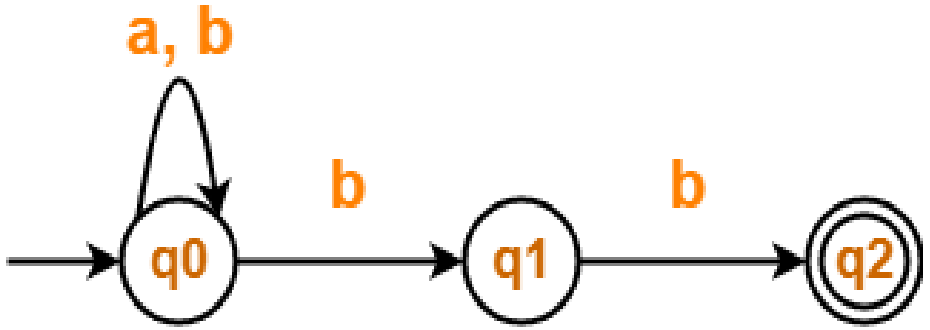
(4) Mark as accepting those states constructed in this manner that contain an accepting state of M.

Example 1

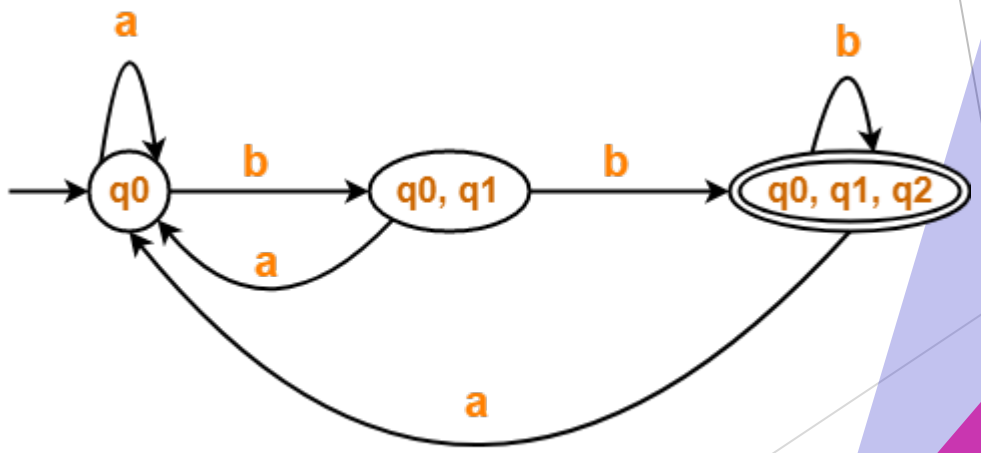


Example 1

State / Alphabet	a	b
→q0	q0	q0, q1
q1	–	*q2
*q2	–	–

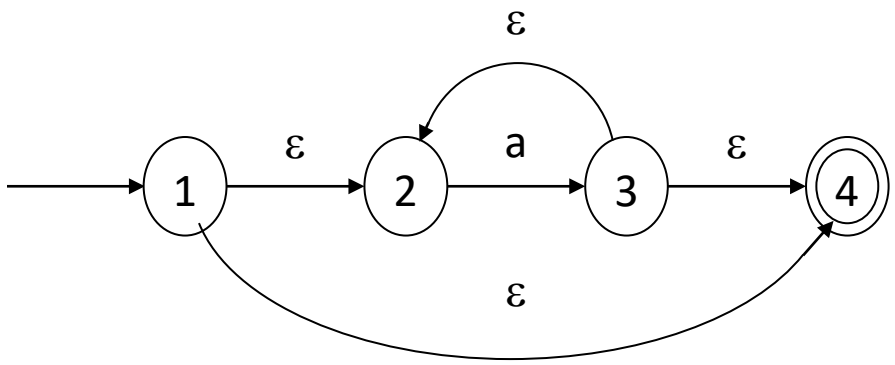


DFA Table		
State / Alphabet	a	b
→q0	q0	{q0, q1}
{q0, q1}	q0	*{q0, q1, q2}
*{q0, q1, q2}	q0	*{q0, q1, q2}

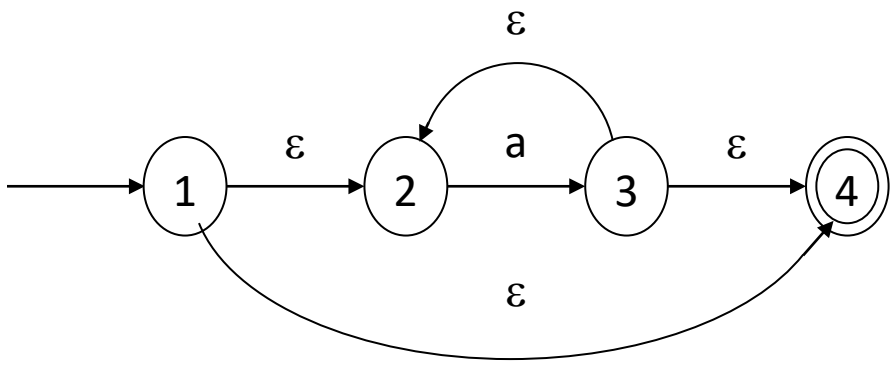


Deterministic Finite Automata (DFA)

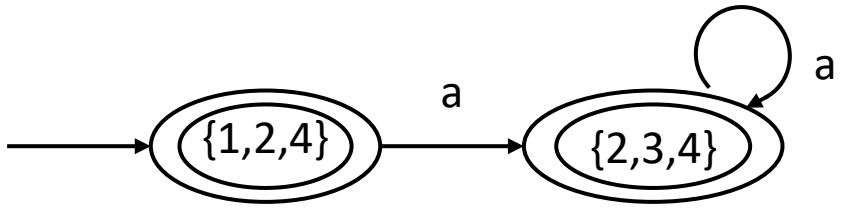
Example 2



Example 2

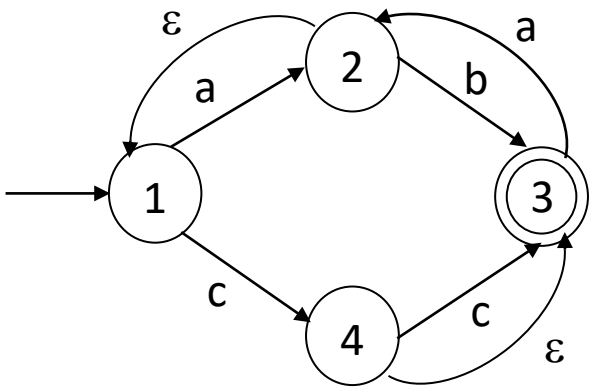


State / Alphabet	a	ε
→1	-	{1, 2, 4}
2	3	{2}
3	-	{2, 3, 4}
*4	-	{4}

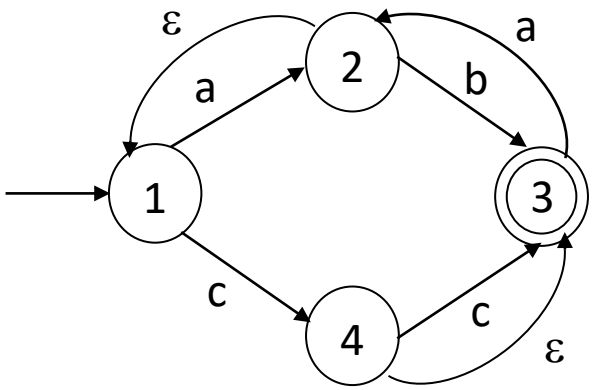


DFA Table	
State / Alphabet	a
→{1,2,4}	{2,3,4}
{2,3,4}	{2,3,4}

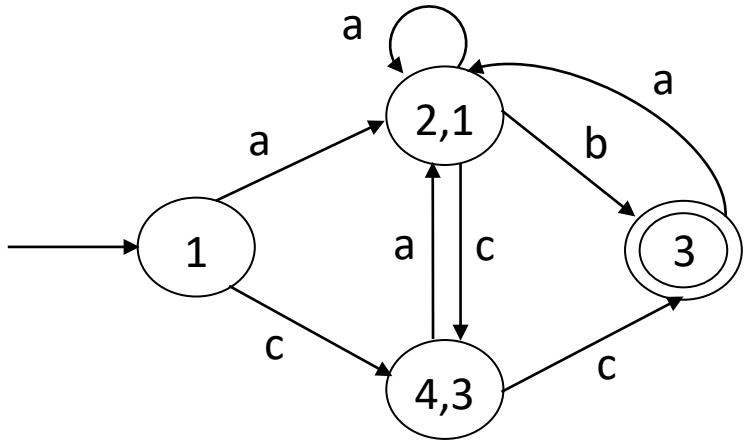
Example 3



Example 3



State	a	b	c	ε
→1	2	-	4	1
2	-	3	-	2,1
3	2	-	-	3
4	-	-	3	4,3



State	a	b	c
→1	2,1	-	4,3
2,1	2,1	3	4,3
4,3	2,1	-	3
3	2,1	-	-

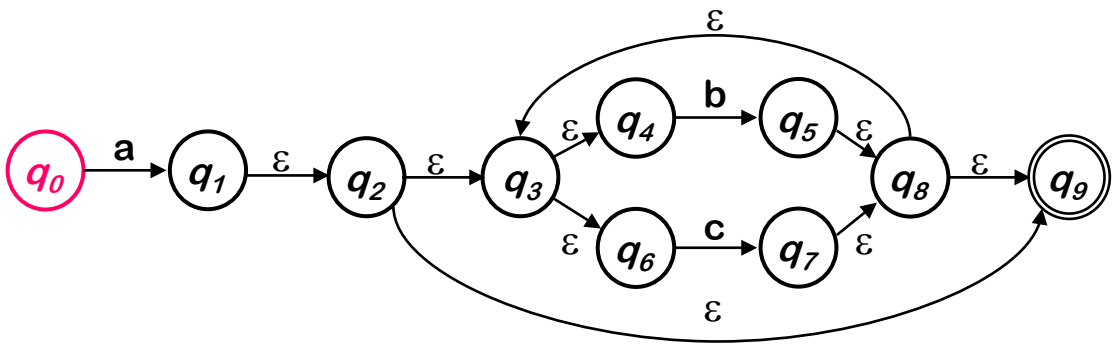
DFA Table

Summary of Conversion Steps

1. Find e-closure of all states from the NFA
2. Draw the NFA transition table
3. Start computing the DFA table from the first state and take the resulting states as the next state in each step
4. Draw the DFA from generated DFA table

NFA →DFA with Subset Construction

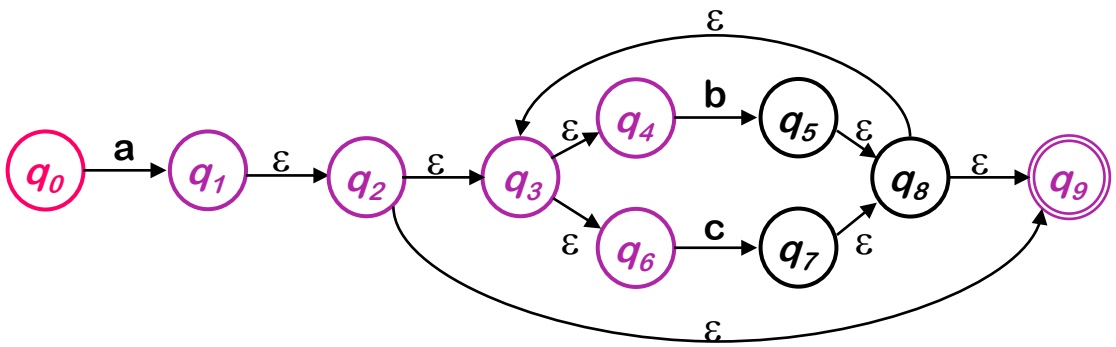
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0			

NFA →DFA with Subset Construction

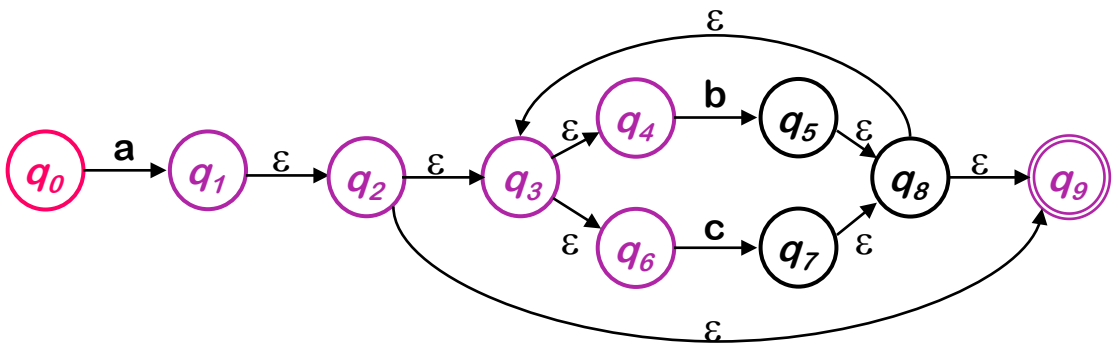
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$		

NFA →DFA with Subset Construction

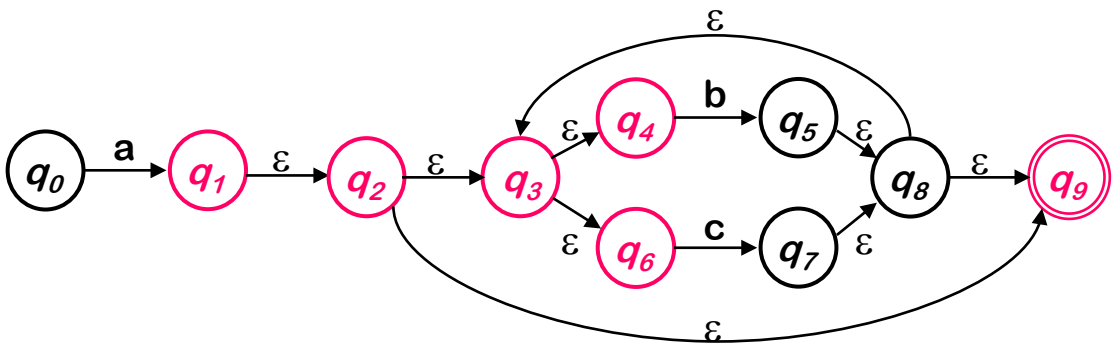
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none

NFA →DFA with Subset Construction

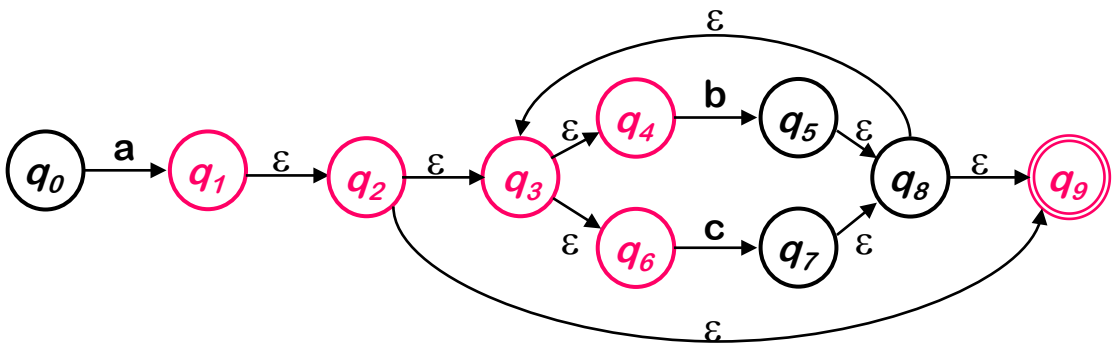
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$			

NFA →DFA with Subset Construction

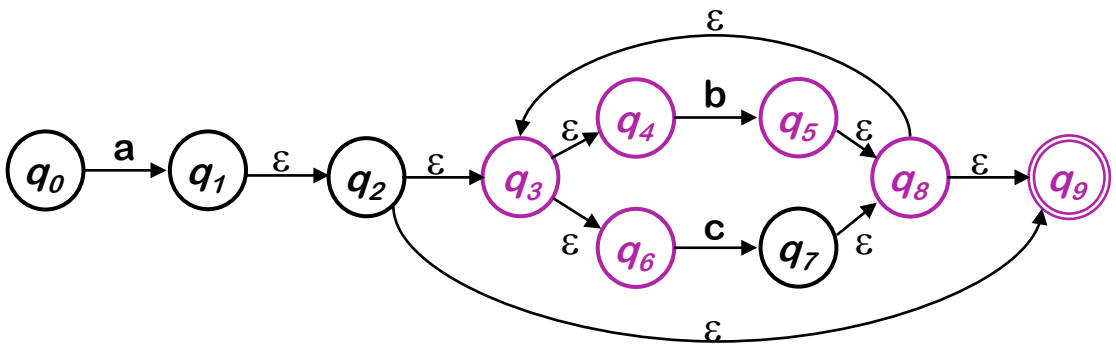
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none		

NFA →DFA with Subset Construction

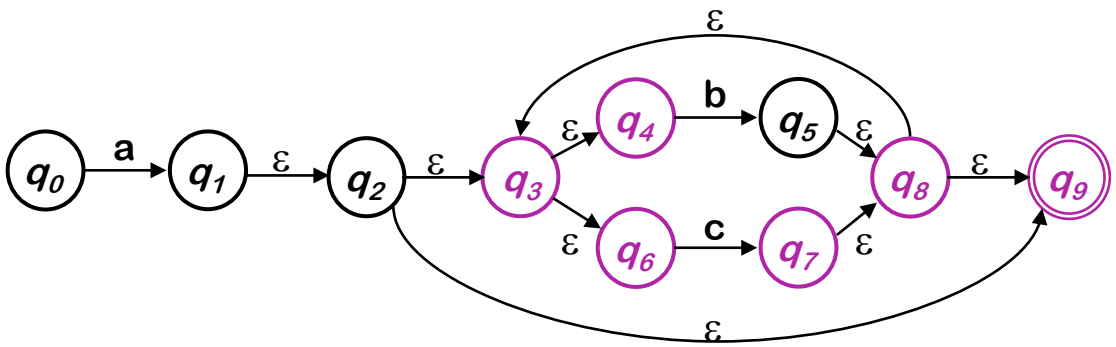
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	$q_5, q_8, q_9, q_3, q_4, q_6$	

NFA →DFA with Subset Construction

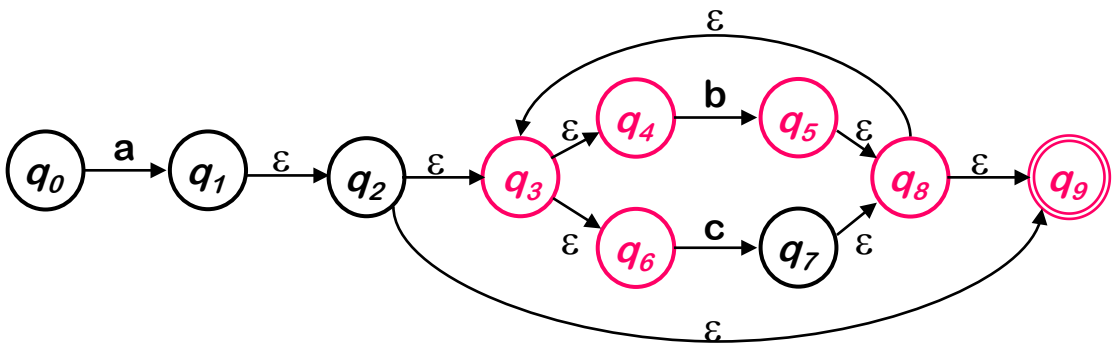
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$

NFA →DFA with Subset Construction

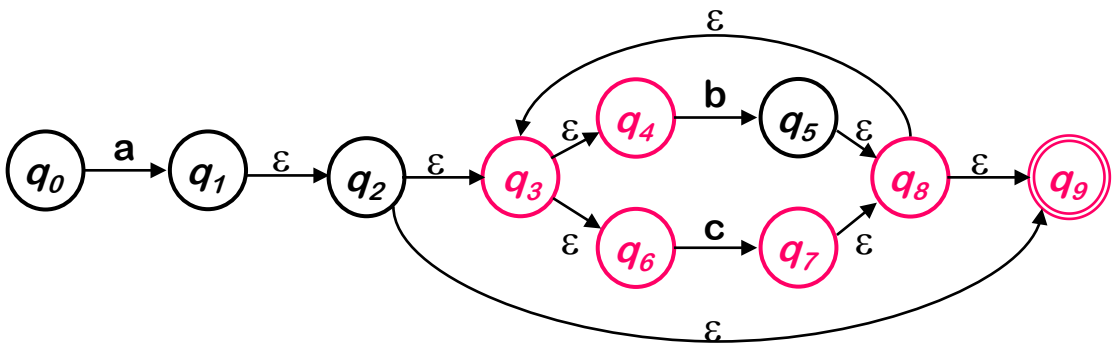
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$			

NFA →DFA with Subset Construction

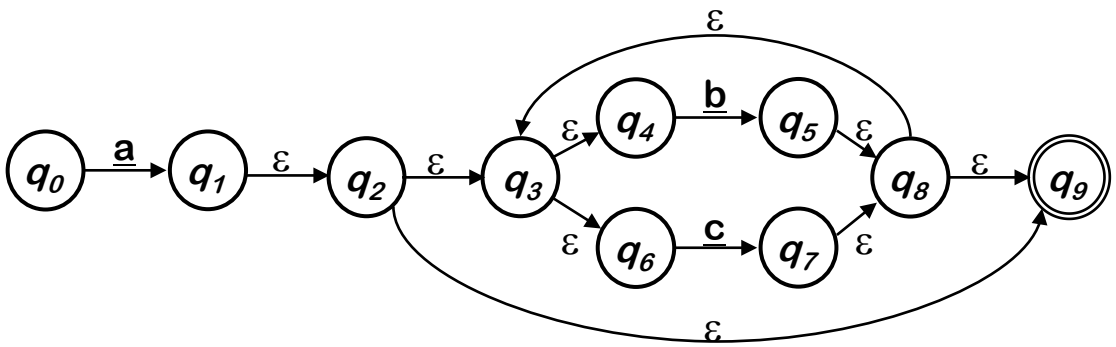
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$			
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$			

NFA →DFA with Subset Construction

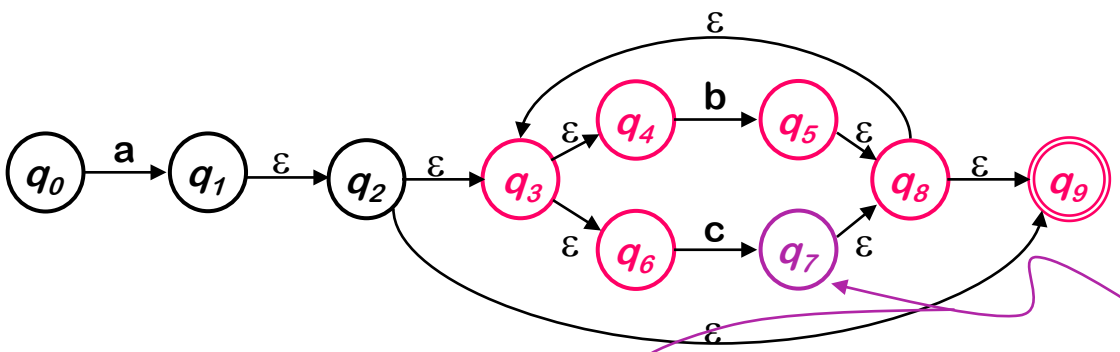
a (b | c)* :



States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	none		
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	none		

NFA →DFA with Subset Construction

a (b | c)* :

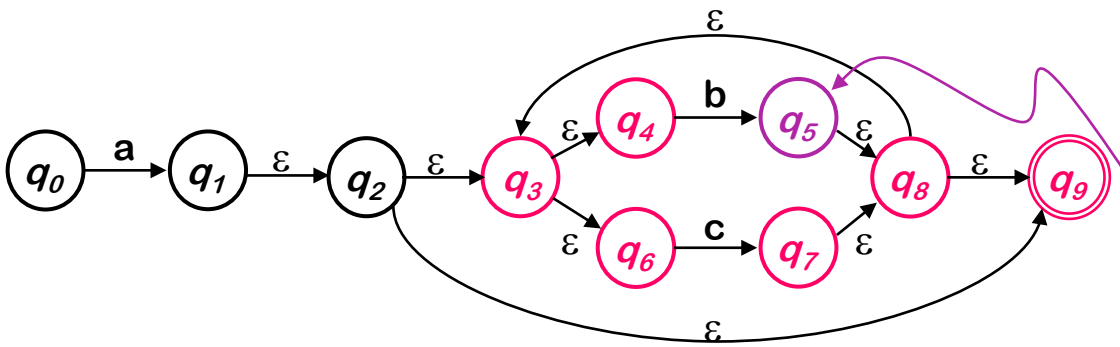


q_7 is the core state of s_3

States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	none	s_2	s_3
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	none		

NFA →DFA with Subset Construction

a (b | c)* :

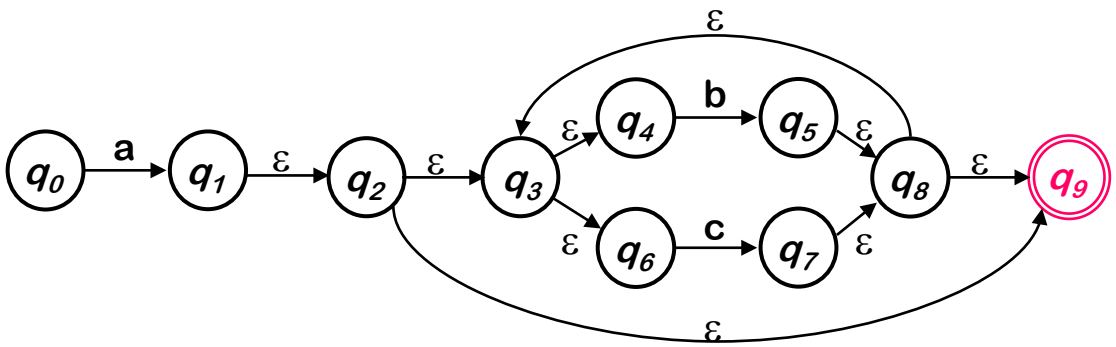


q_5 is the core state of s_2

States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	none	s_2	s_3
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	none	s_2	s_3

NFA →DFA with Subset Construction

a (b | c)* :

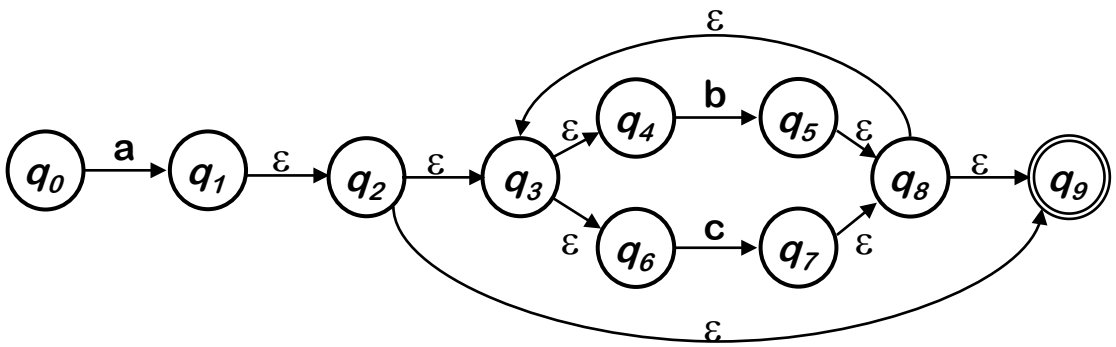


States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	$q_1, q_2, q_3, q_4, q_6, q_9$	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	$q_5, q_8, q_9, q_3, q_4, q_6$	$q_7, q_8, q_9, q_3, q_4, q_6$
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	none	s_2	s_3
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	none	s_2	s_3

Final states because of q_9

NFA →DFA with Subset Construction

a (b | c)* :

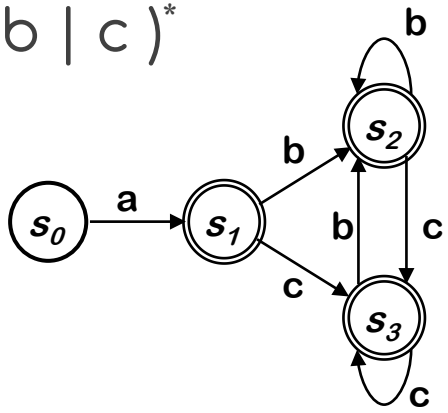


States		ε-closure(Move(s,*))		
DFA	NFA	a	b	c
s_0	q_0	s_1	none	none
s_1	$q_1, q_2, q_3, q_4, q_6, q_9$	none	s_2	s_3
s_2	$q_5, q_8, q_9, q_3, q_4, q_6$	none	s_2	s_3
s_3	$q_7, q_8, q_9, q_3, q_4, q_6$	none	s_2	s_3

Transition table for the DFA

NFA →DFA with Subset Construction

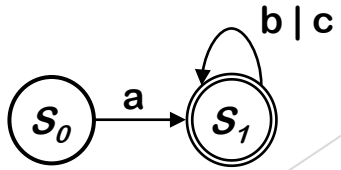
The DFA for a (b | c)*



	a	b	c
s_0	s_1	none	none
s_1	none	s_2	s_3
s_2	none	s_2	s_3
s_3	none	s_2	s_3

- Much smaller than the NFA (no ε-transitions)
- All transitions are deterministic
- Use same code skeleton as before

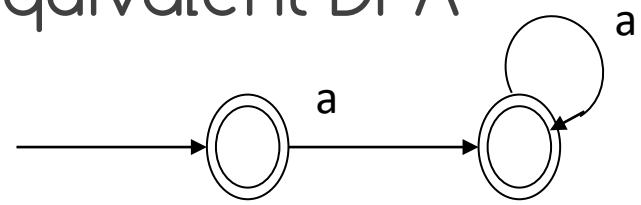
DFA Can be minimized to

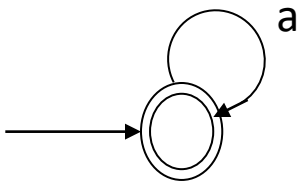


Minimizing the Number of States in a DFA

Why need Minimizing?

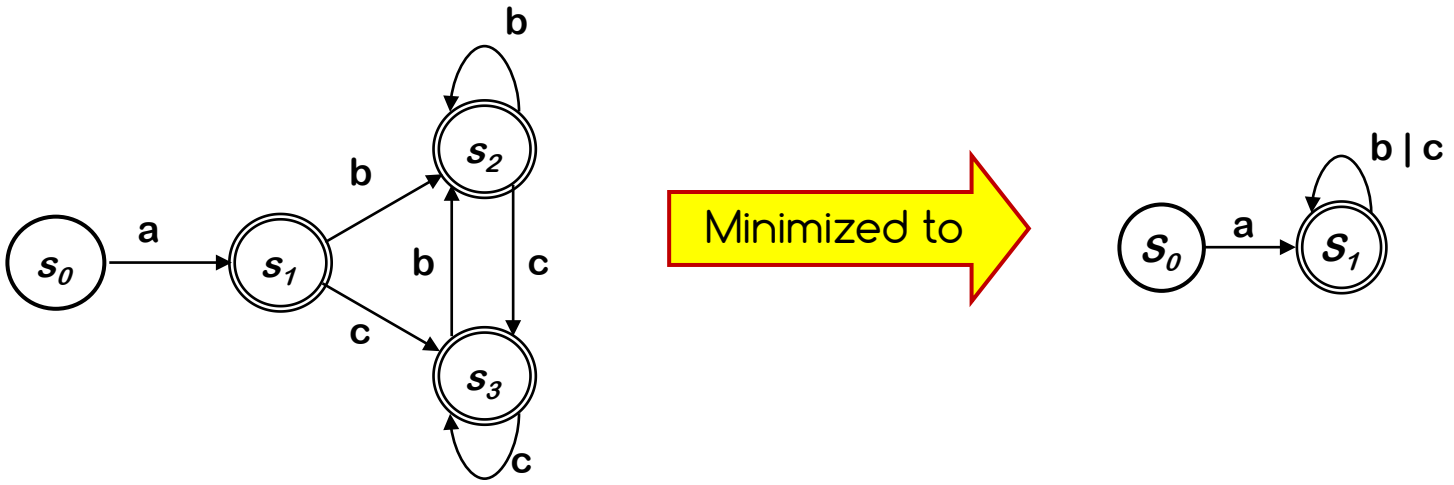
- DFA minimization/optimization stands for converting a given DFA to its equivalent **DFA with minimum number of states**.
- The process of deriving a DFA algorithmically from a regular expression has the unfortunate property that the resulting DFA may be more complex than necessary.
- The derived DFA for the regular expression a* and an equivalent DFA





An Important Result from Automata Theory for Minimizing

- Given any DFA, there is an equivalent DFA containing a minimum number of states, and, that this minimum-state DFA is unique (except for renaming of states)

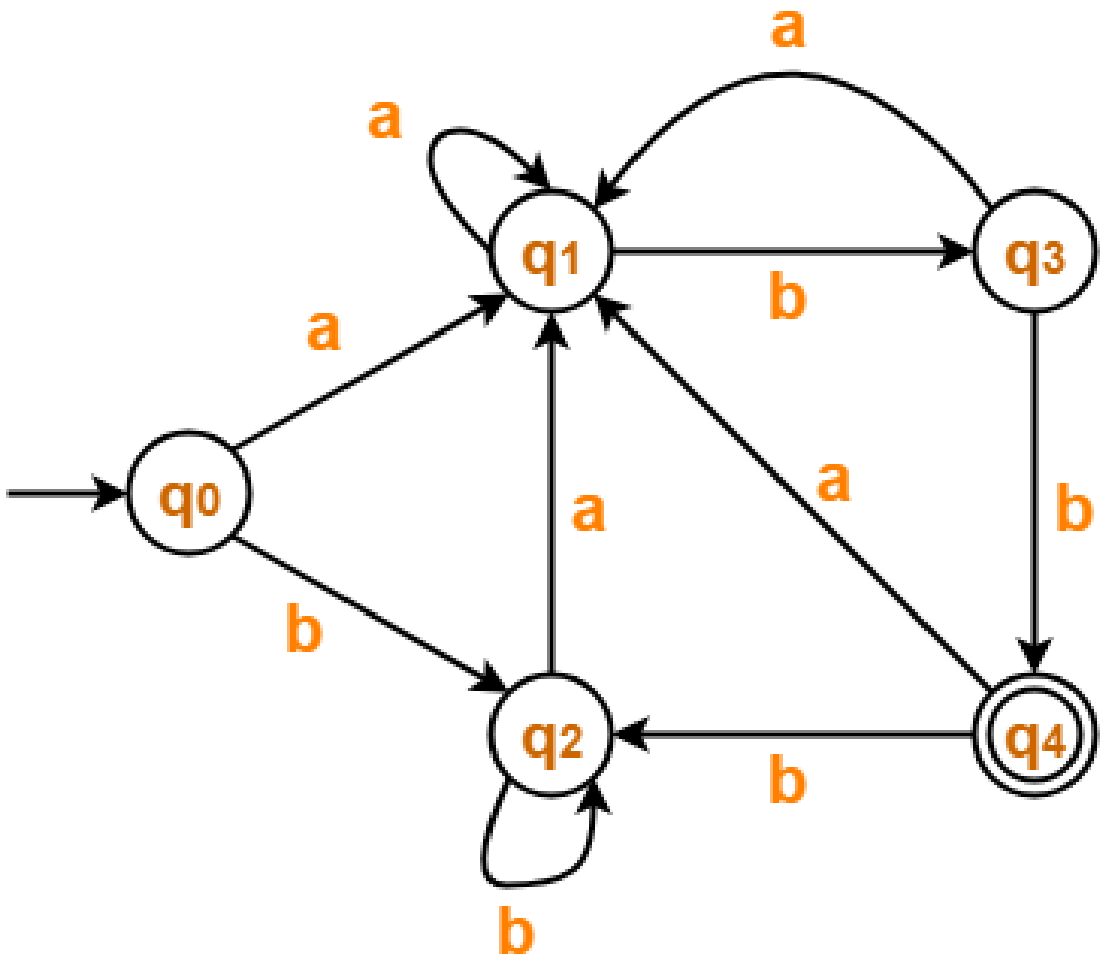


DFA Minimization using Equivalence Theorem

- **Step 1** – All the states **Q** are divided in two partitions – **final states** and **non-final states** and are denoted by P_0. All the states in a partition are 0^{th} equivalent. Take a counter **k** and initialize it with 0.
- **Step 2** – Increment k by 1. For each partition in P_k, divide the states in P_k into two partitions if they are k-distinguishable. Two states within this partition X and Y are k-distinguishable if there is an input **S** such that **δ(X, S)** and **δ(Y, S)** are (k-1)-distinguishable.
- **Step 3** – If $P_k \neq P_{k-1}$, repeat Step 2, otherwise go to Step 4.
- **Step 4** – Combine k^{th} equivalent sets and make them the new states of the reduced DFA.

Example 1

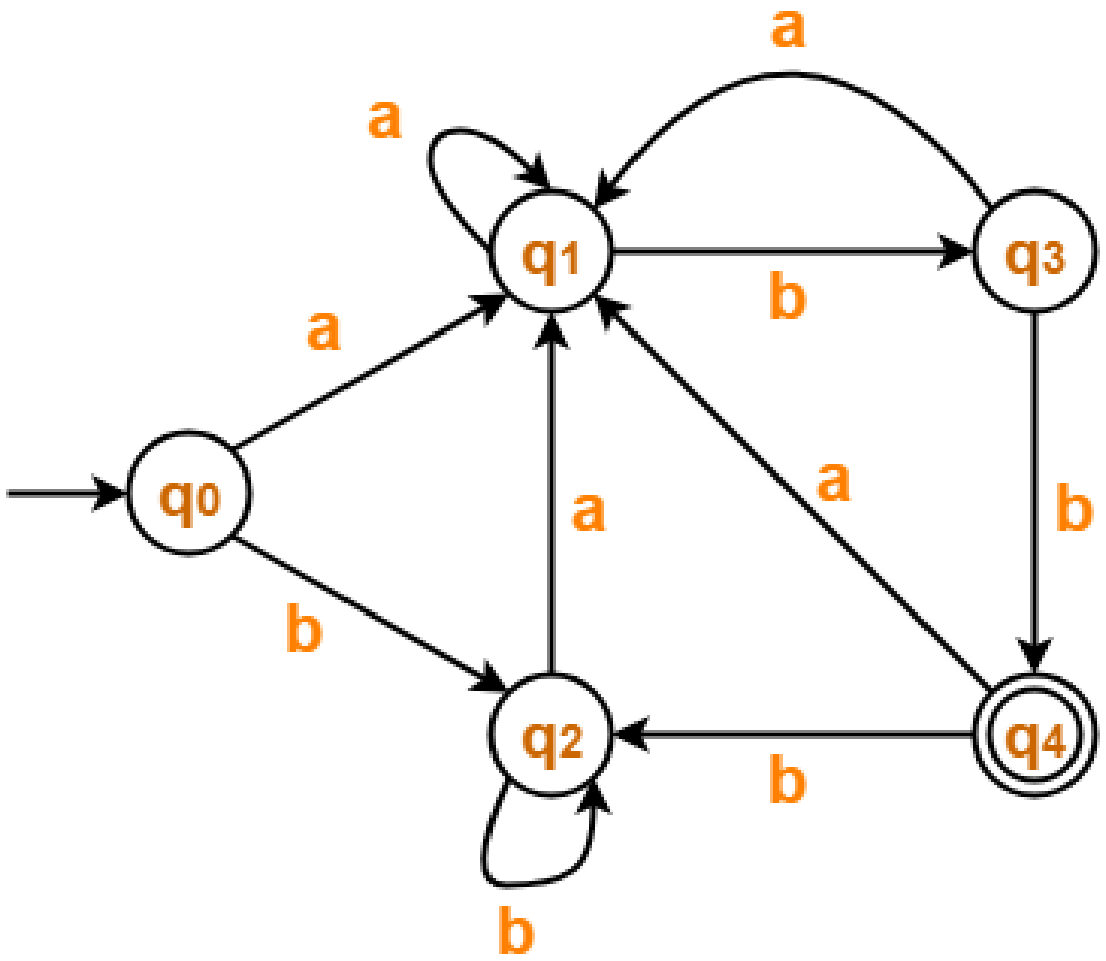
Draw a state transition table for the given DFA



Example 1

Draw a state transition table for the given DFA

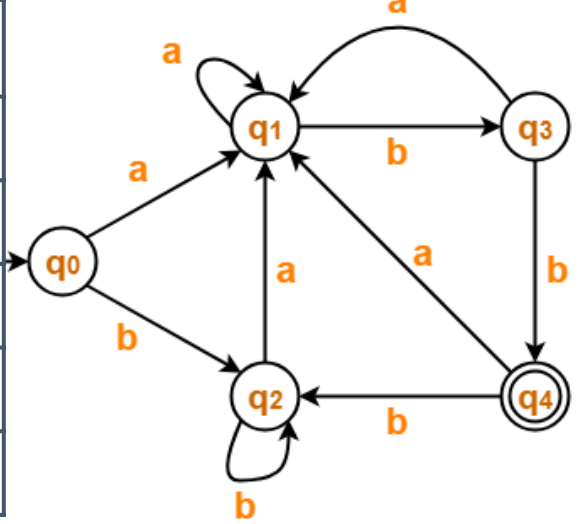
State	a	b
→q0	q1	q2
q1	q1	q3
q2	q1	q2
q3	q1	*q4
*q4	q1	q2



Example 1

Start applying equivalence theorem

State	a	b
→q0	q1	q2
q1	q1	q3
q2	q1	q2
q3	q1	*q4
*q4	q1	q2

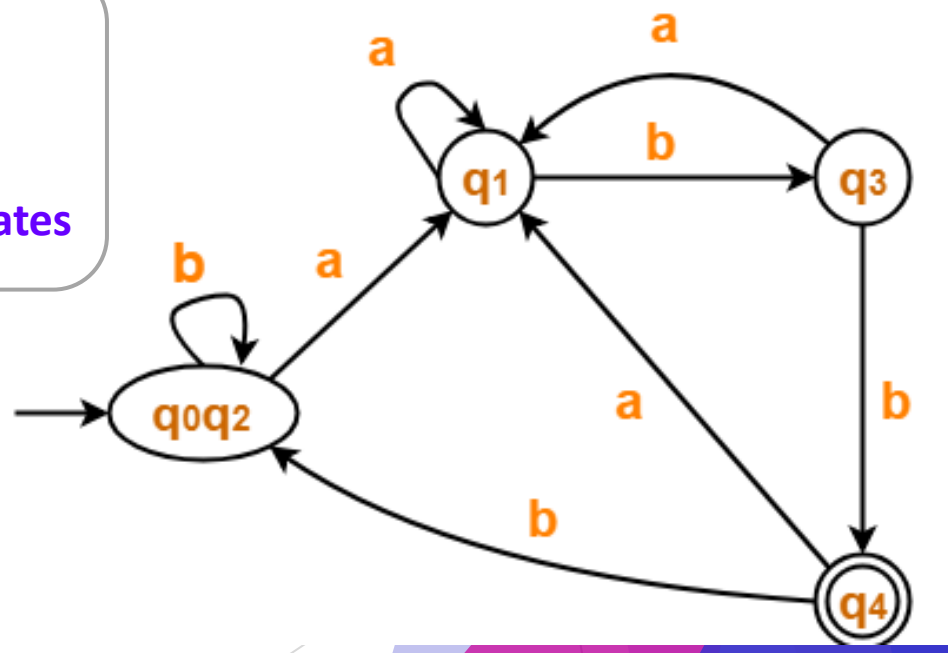


- P0 = { q0 , q1 , q2 , q3 } { q4 }
- P1 = { q0 , q1 , q2 } { q3 } { q4 }
- P2 = { q0 , q2 } { q1 } { q3 } { q4 }
- P3 = { q0 , q2 } { q1 } { q3 } { q4 }
- Since $P_3 = P_2$, so we stop
- From P_3, we infer that states q_0 and q_2 are equivalent and can be merged together.
- So, Our minimal DFA

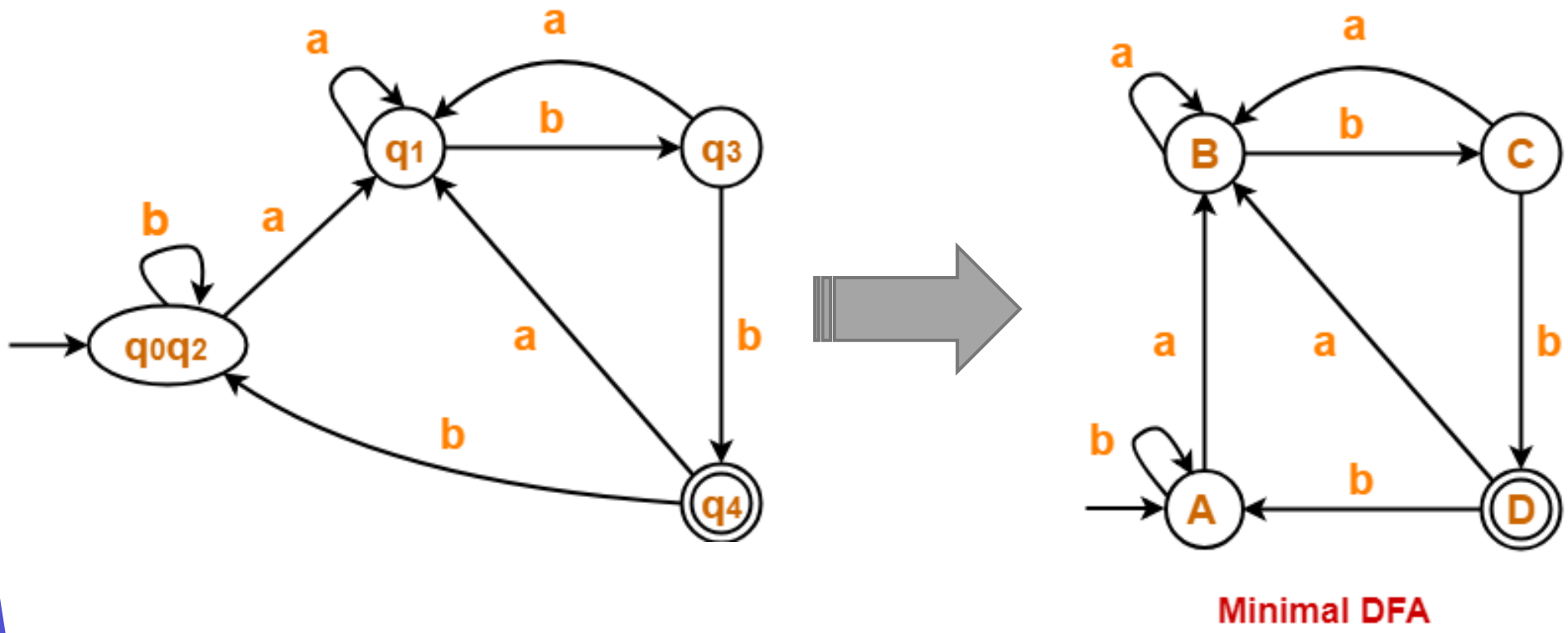
Start with P0

Two partitions

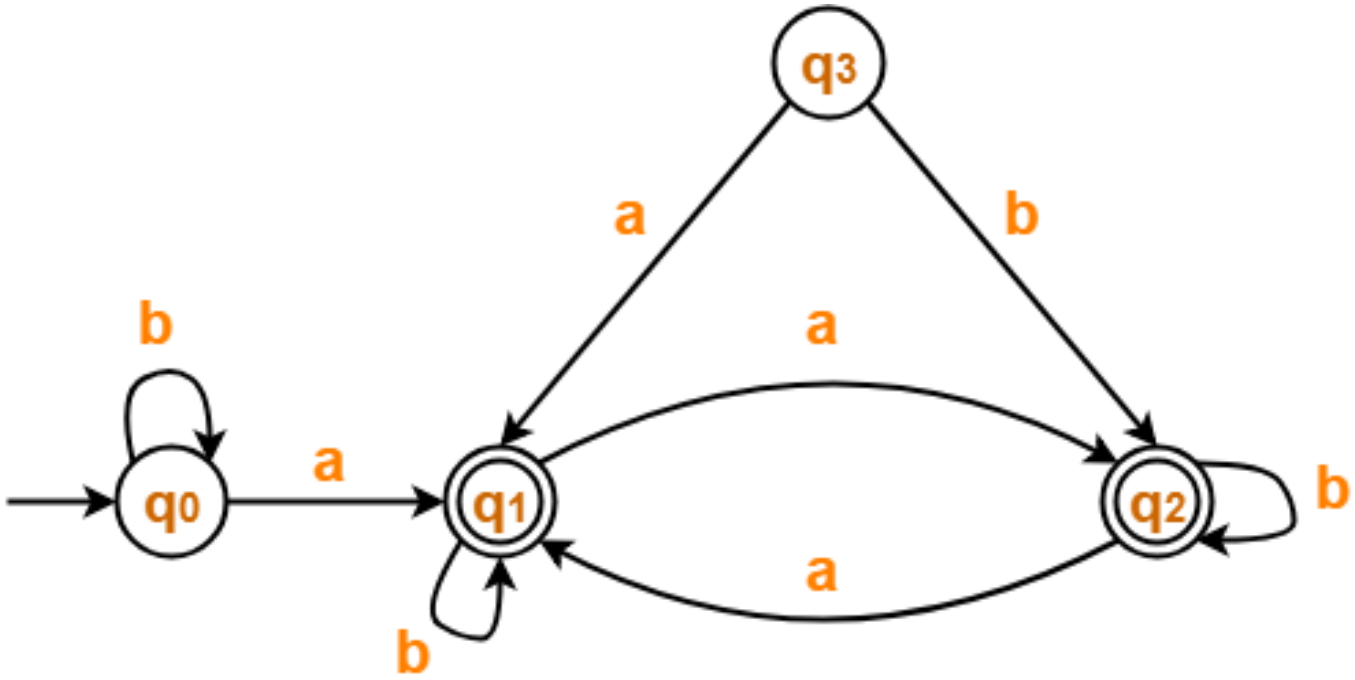
- **final states**
- **non-final states**



Example 1

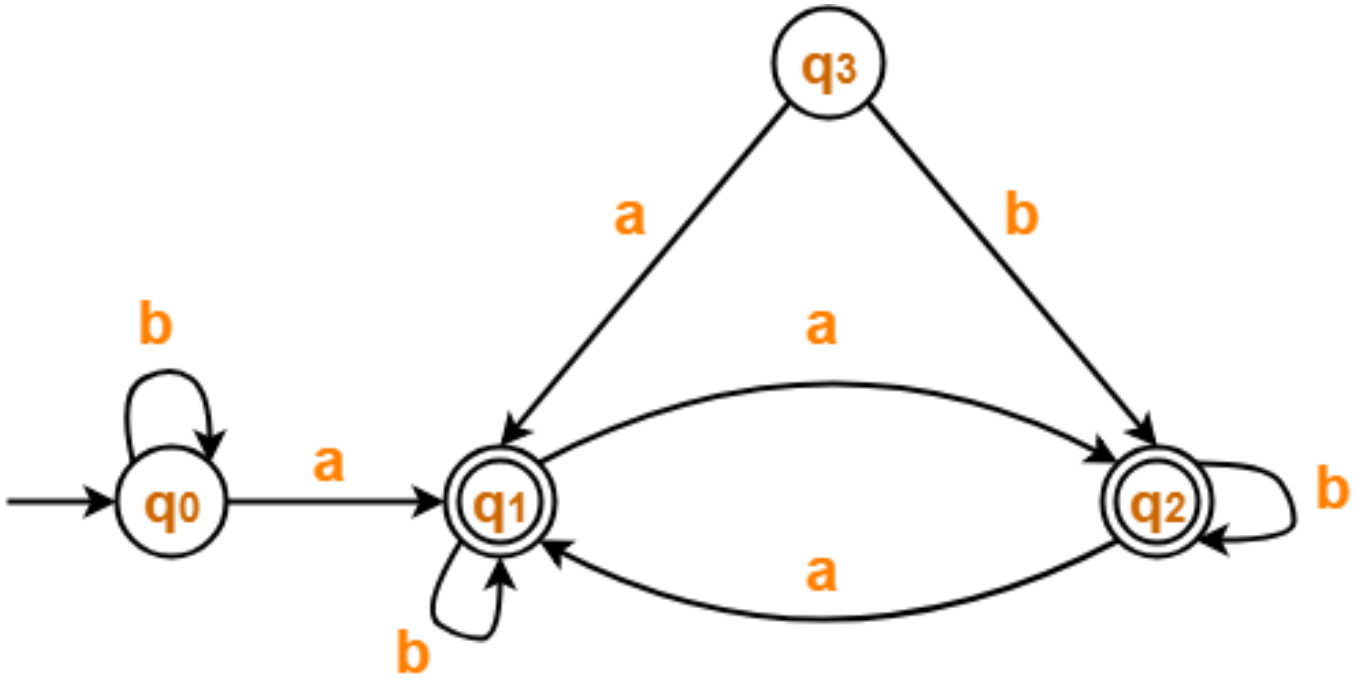


Example 2



- Remove dead and inaccessible states

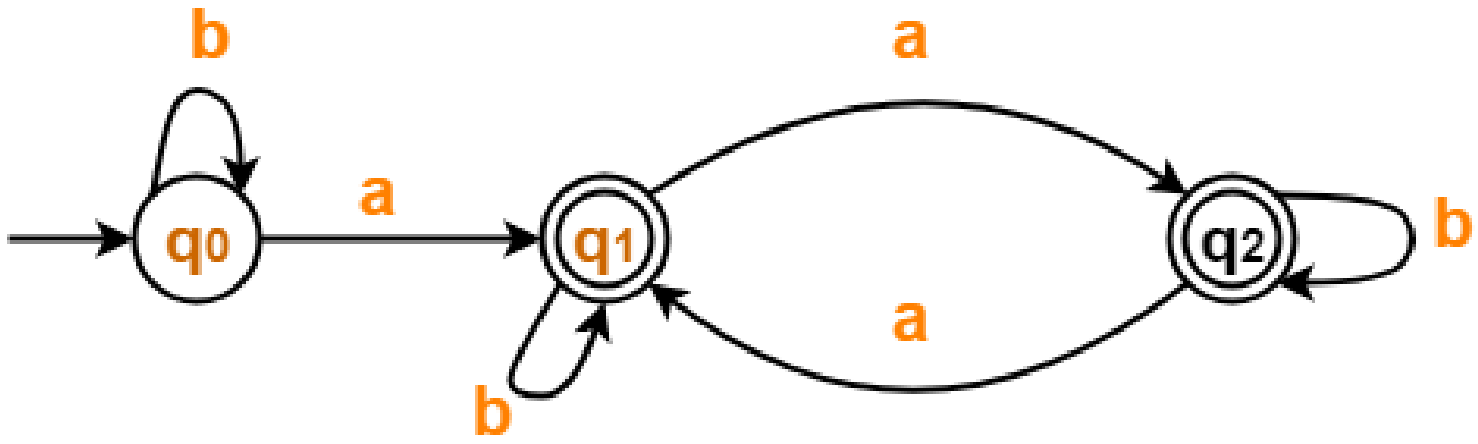
Example 2



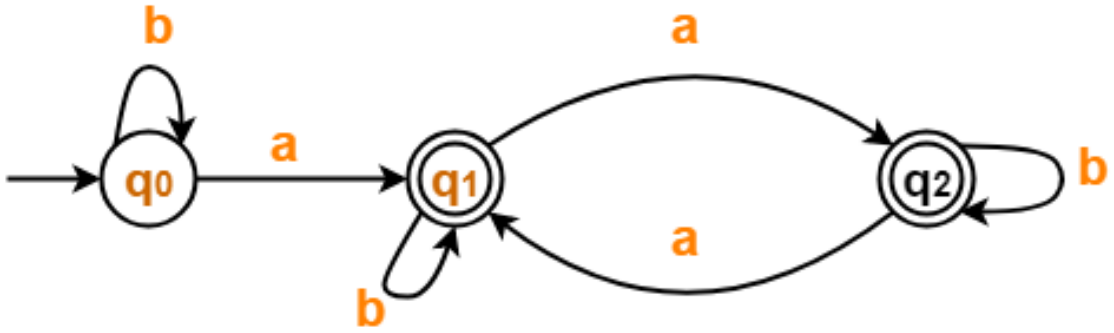
- State q_3 is inaccessible from the initial state.

Example 2

- So, we eliminate it and its associated edges from the DFA.



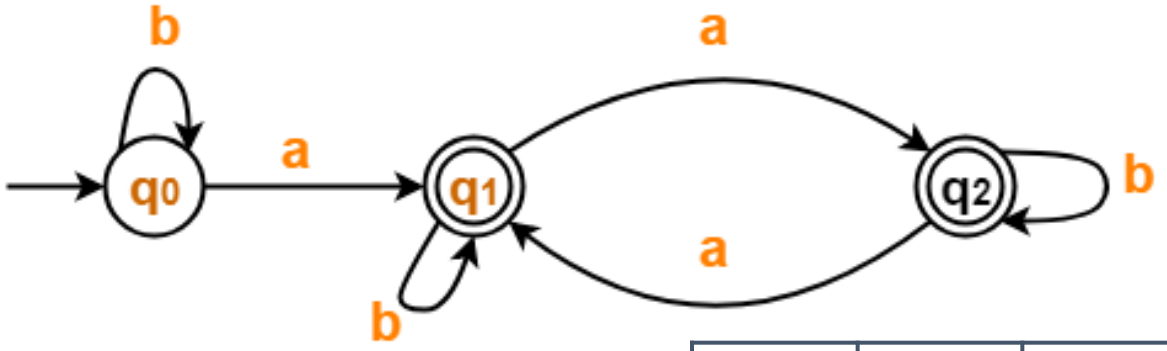
Example 2



- Draw a state transition table

State	a	b
→q0	*q1	q0
*q1	*q2	*q1
*q2	*q1	*q2

Example 2

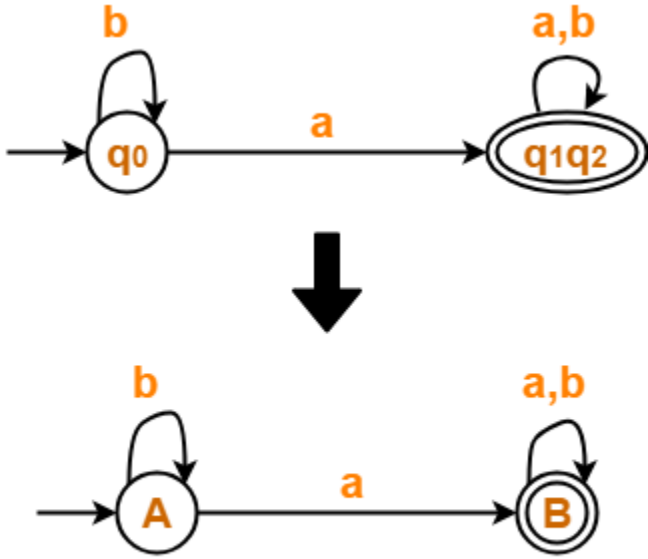


Now using Equivalence Theorem, we have-

- $P_0 = \{ q_0 \} \{ q_1 , q_2 \}$
- $P_1 = \{ q_0 \} \{ q_1 , q_2 \}$

State	a	b
→q0	*q1	q0
*q1	*q2	*q1
*q2	*q1	*q2

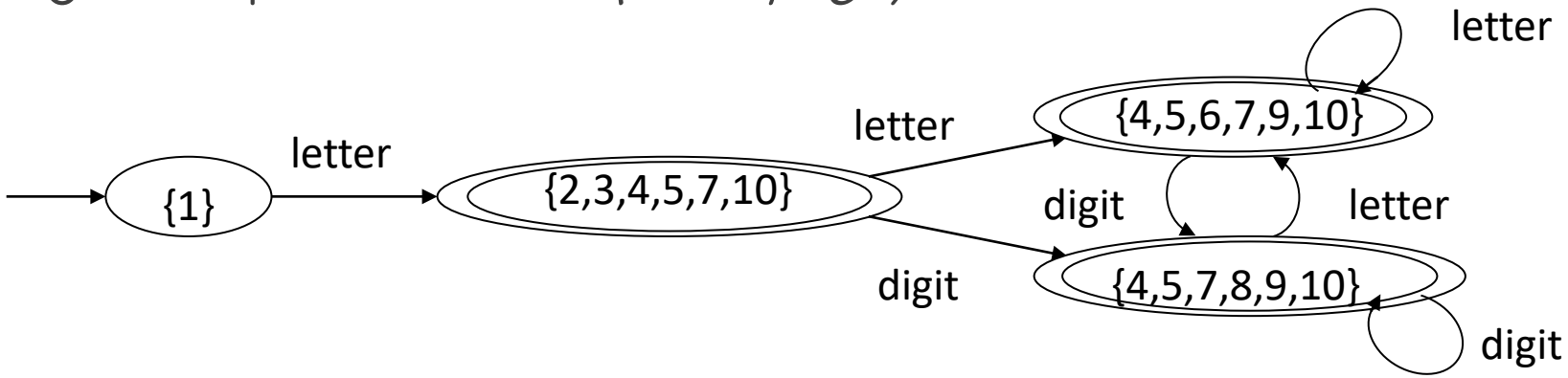
- Since $P_1 = P_0$, so we stop.
- From P_1, we infer that states q_1 and q_2 are equivalent and can be merged together.
- So, Our minimal DFA is-



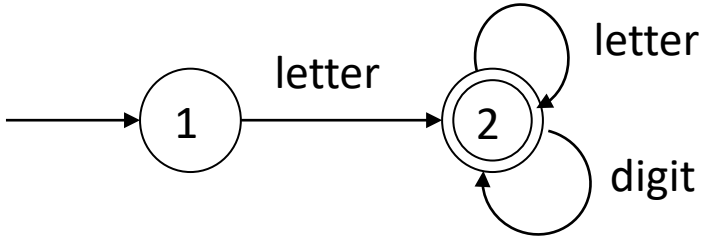
Minimal DFA

Example 3

The regular expression *letter(letter|digit)**

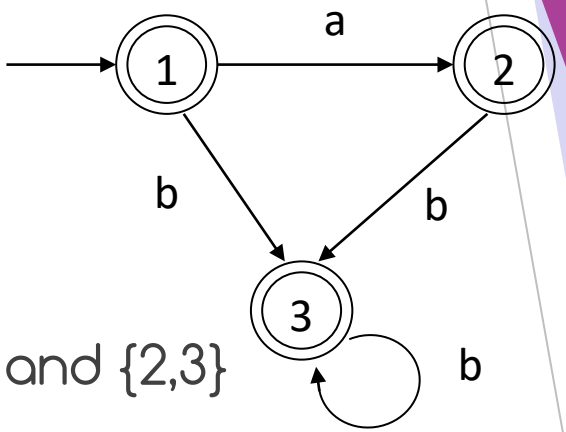


The accepting sets	{2,3,4,5,7,10},{4,5,6,7,9,10},{4,5,7,8,9,10}
The nonaccepting sets	{1}



Example 4

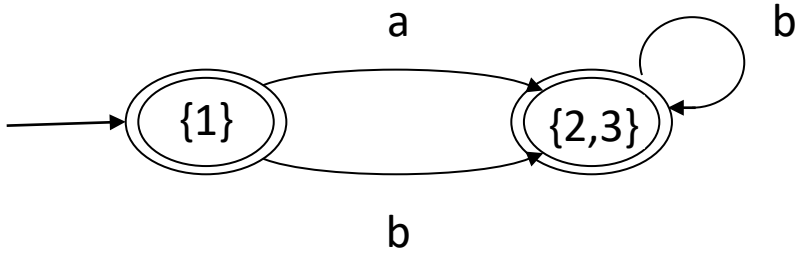
The regular expression (a| ε) b*



a distinguishes state 1 from states 2 and 3,

and we must repartition the states into the sets {1} and {2,3}

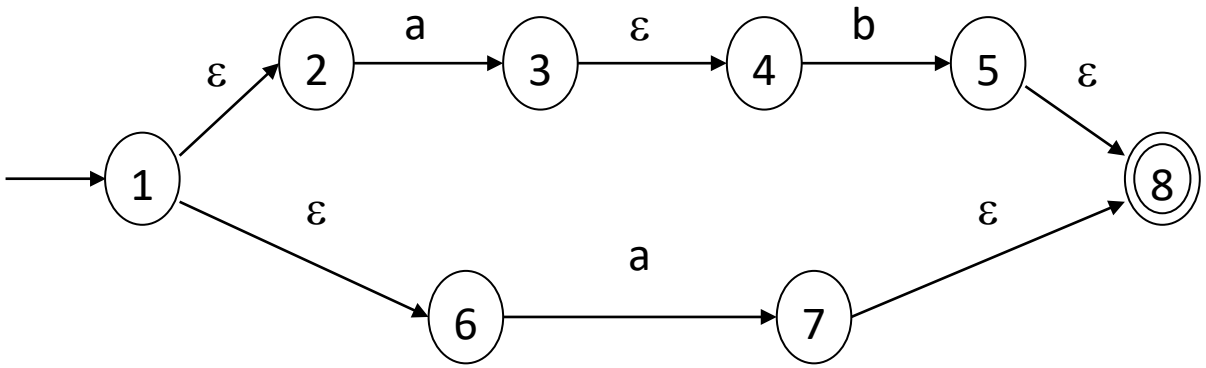
The accepting sets	{1,2,3}
The non-accepting sets	



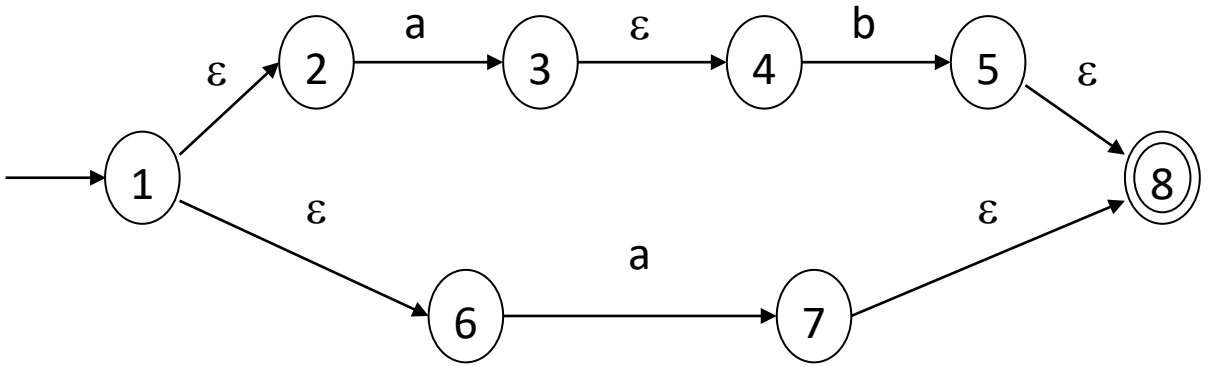

Q
A

Review Questions

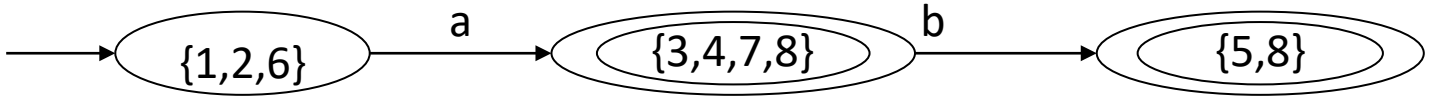
NFA to DFA Question 1



Solution of Question 1



M	ε-closure of M (S)	S'_a	S'_b
1	1,2,6	3,7	
3,7	3,4,7,8		5
5	5,8		

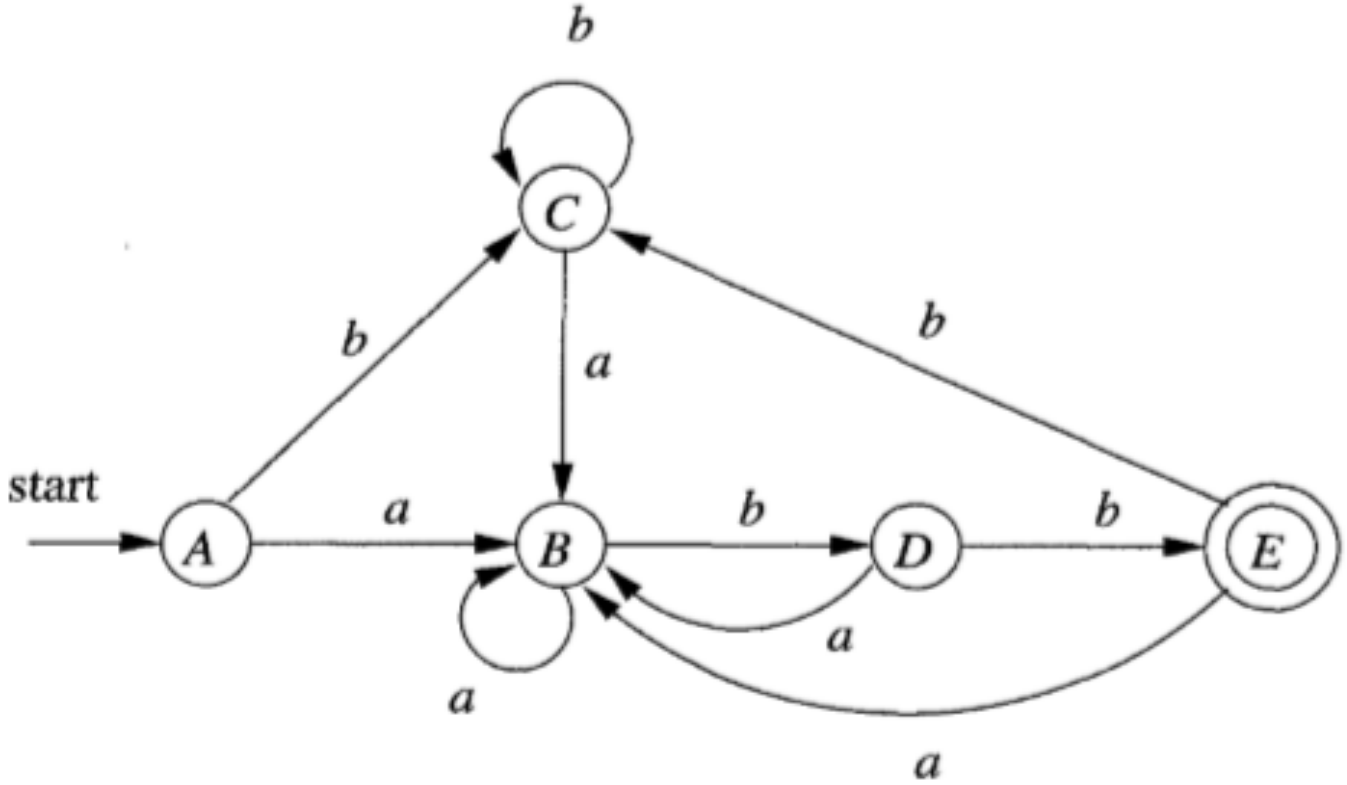


NFA to DFA Question 2

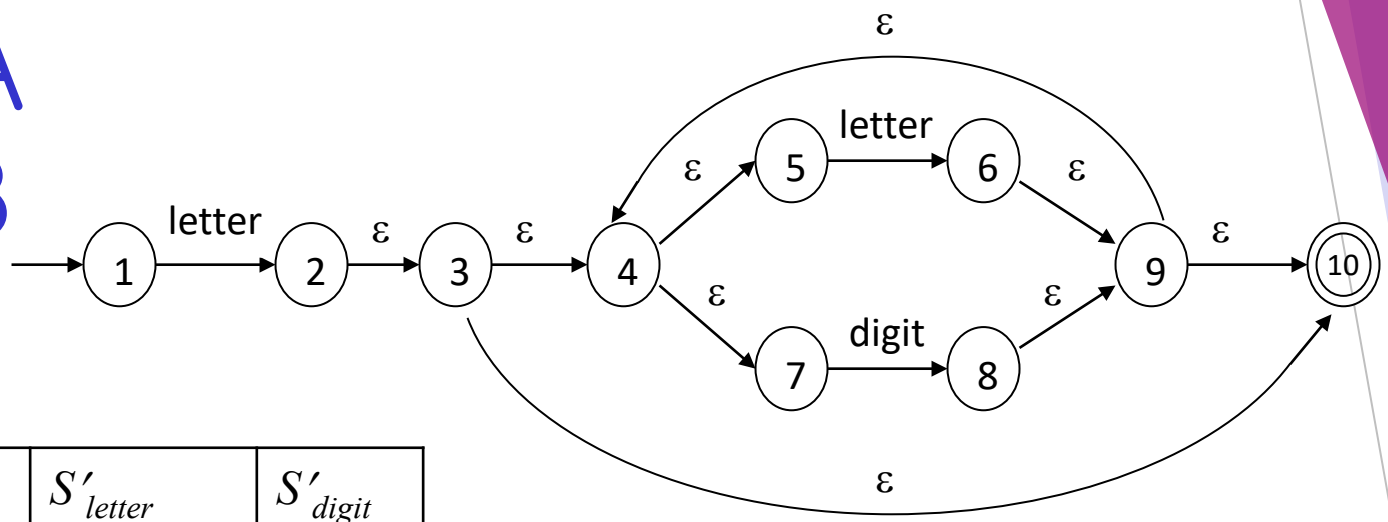
- the DFA for (a | b)* abb

NFA to DFA Question 2

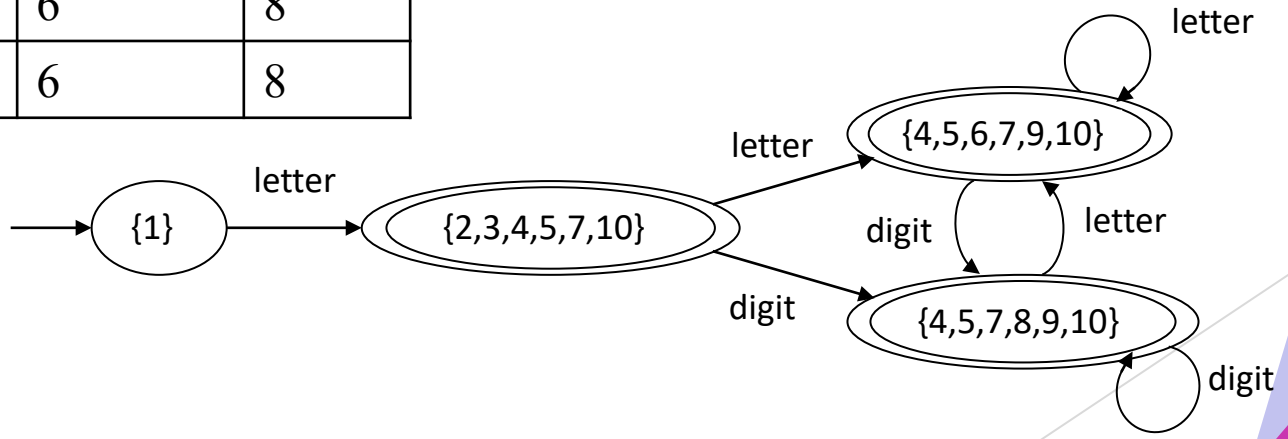
- the DFA for (a | b)* abb



NFA to DFA Question 3

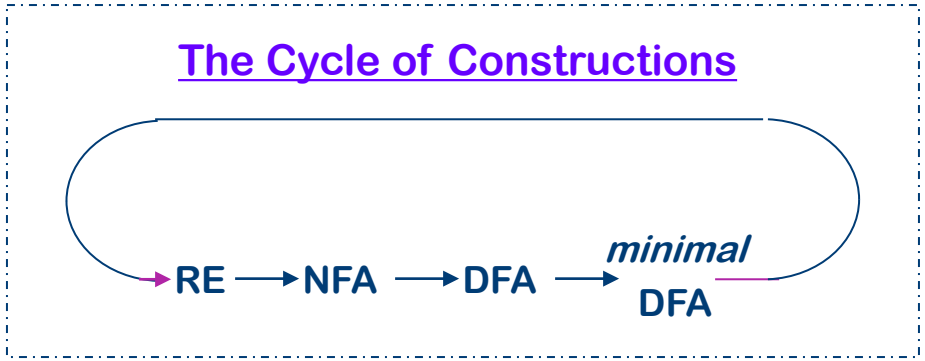


M	ε-closure of M (S)	S'_{letter}	S'_{digit}
1	1	2	
2	2,3,4,5,7,10	6	8
6	4,5,6,7,9,10	6	8
8	4,5,7,8,9,10	6	8



Summary of this lecture

- RE to NFA: Thompson's construction
  - Core insight: inductively build up NFA using "templates"
  - Core concept: use null transitions to build NFA quickly
- NFA to DFA: Subset construction
  - Core insight: DFA nodes represent subsets of NFA nodes
  - Core concept: use null closure to calculate subsets
- DFA minimization: Hopcroft's algorithm
  - Core insight: create partitions, then keep splitting



Supplementary Material

- NFA to DFA
  - https://www.youtube.com/watch?v=taClnxU-nao
  - https://www.youtube.com/watch?v=U71roXRlNlg
- Regular Expressions
  - https://www.youtube.com/watch?v=upu_TeZImN0&list=PLBlnK6fEyqRgp46KUv4ZY69yXmpwKOIev&index=45
  - https://www.youtube.com/watch?v=paOPoZyjzdg&list=PLBlnK6fEyqRgp46KUv4ZY69yXmpwKOIev&index=46

References of this lecture

- Presentation slides of the book: COMPILER CONSTRUCTION, Principles and Practice, by Kenneth C. Louden
- Presentation slides of Introduction to the Theory of Computation, Michael Sipser book

Prepared by:

- Shachar Lovett, University of California
- Ananth Kalyanaraman, Washington State University

See you next lecture