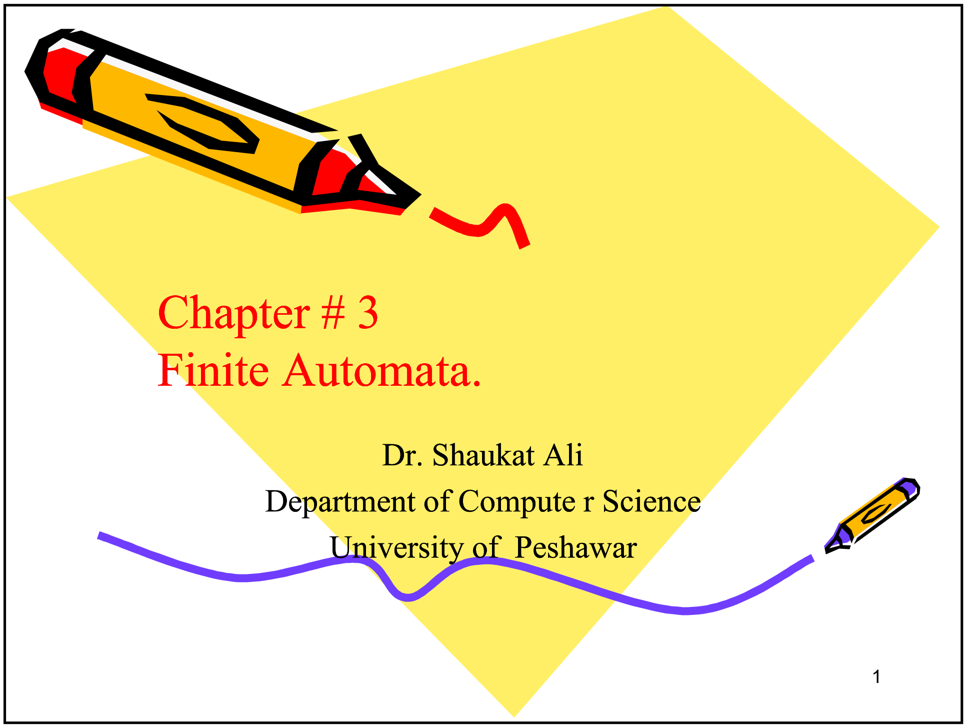
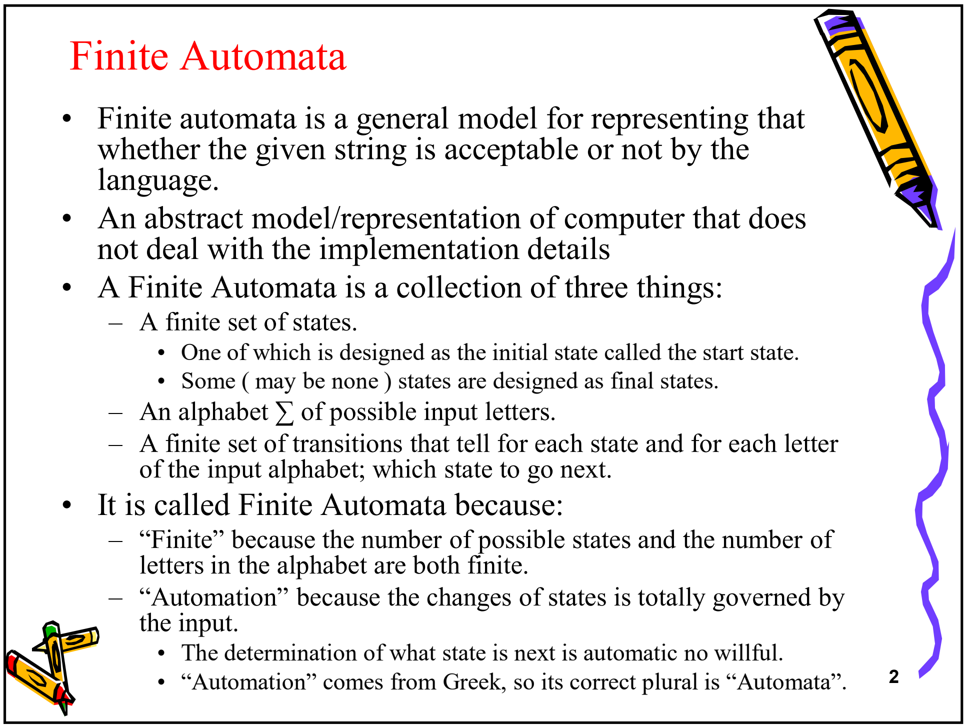


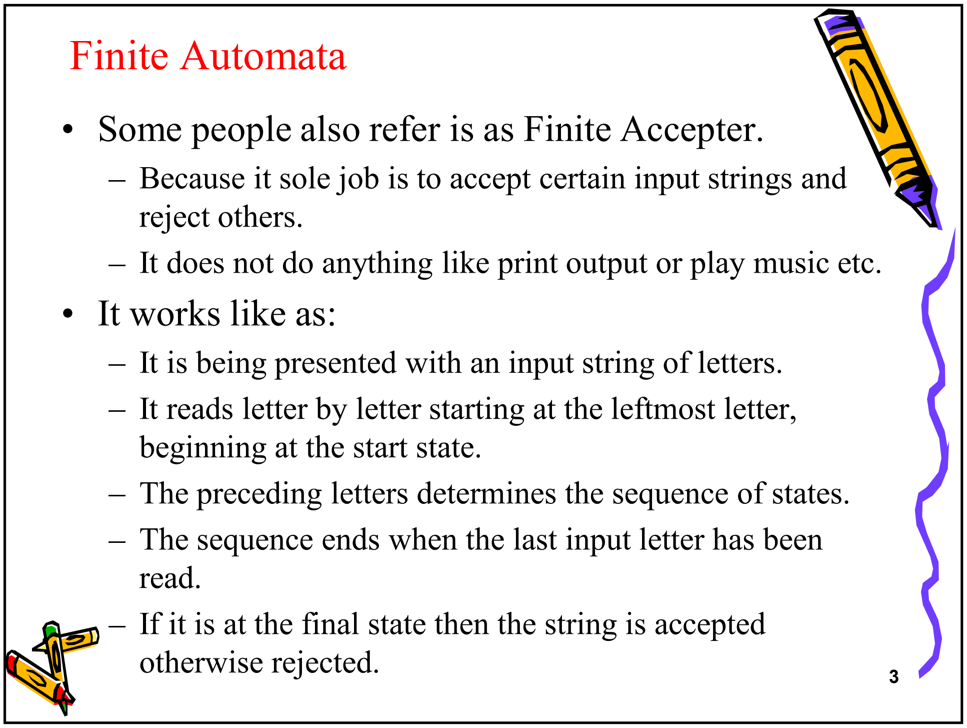
Chapter # 3 Finite Automata.

Dr. Shaukat Ali
Department of Compute r Science
University of Peshawar



Finite Automata

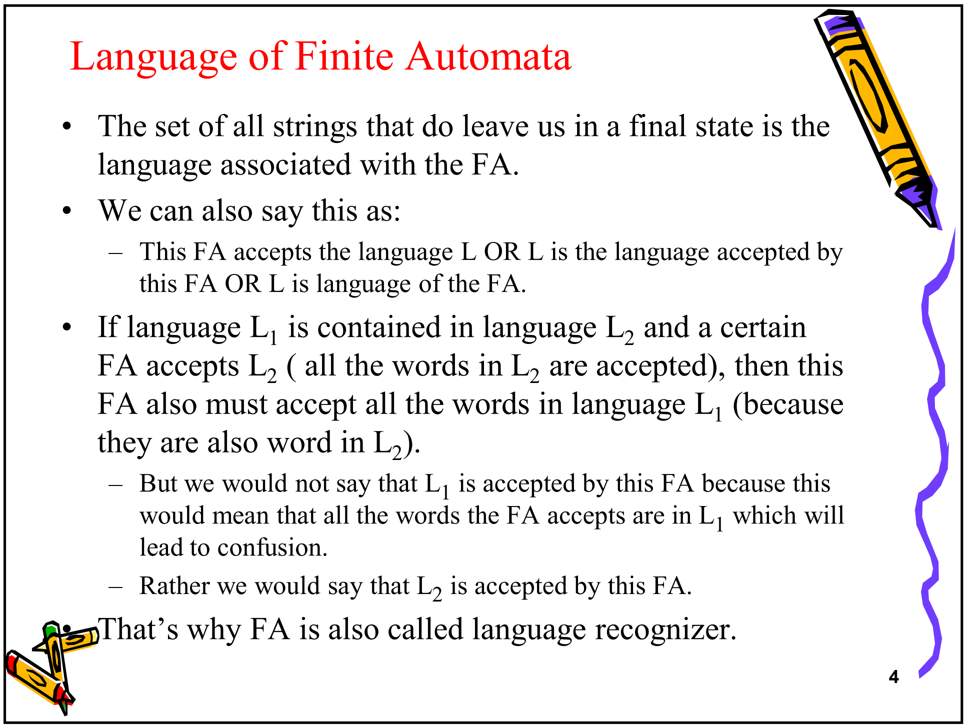
- Finite automata is a general model for representing that whether the given string is acceptable or not by the language.
- An abstract model/representation of computer that does not deal with the implementation details
- A Finite Automata is a collection of three things:
  - A finite set of states.
    - One of which is designed as the initial state called the start state.
    - Some (may be none) states are designed as final states.
  - An alphabet ∑ of possible input letters.
  - A finite set of transitions that tell for each state and for each letter of the input alphabet; which state to go next.
- It is called Finite Automata because:
  - "Finite" because the number of possible states and the number of letters in the alphabet are both finite.
  - "Automation" because the changes of states is totally governed by the input.
    - The determination of what state is next is automatic no willful.
    - "Automation" comes from Greek, so its correct plural is "Automata".



Finite Automata

- Some people also refer is as Finite Accepter.
  - Because it sole job is to accept certain input strings and reject others.
  - It does not do anything like print output or play music etc.
- It works like as:
  - It is being presented with an input string of letters.
  - It reads letter by letter starting at the leftmost letter, beginning at the start state.
  - The preceding letters determines the sequence of states.
  - The sequence ends when the last input letter has been read.
  - If it is at the final state then the string is accepted otherwise rejected.

3

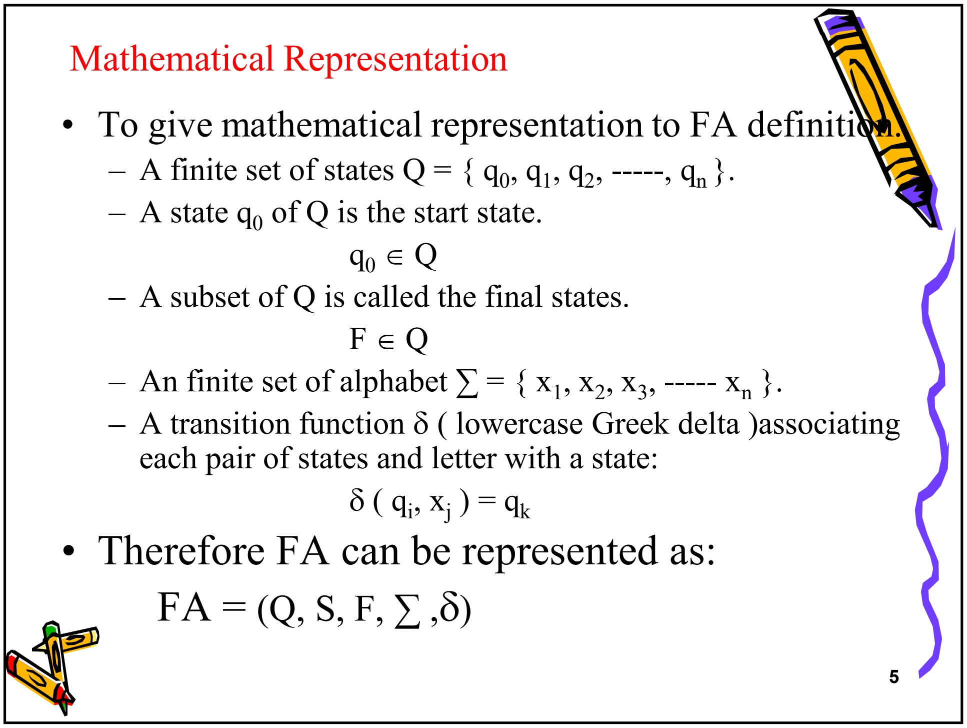


Language of Finite Automata

- The set of all strings that do leave us in a final state is the language associated with the FA.
- We can also say this as:
  - This FA accepts the language L OR L is the language accepted by this FA OR L is language of the FA.
- If language L_1 is contained in language L_2 and a certain FA accepts L_2 (all the words in L_2 are accepted), then this FA also must accept all the words in language L_1 (because they are also word in L_2).
  - But we would not say that L_1 is accepted by this FA because this would mean that all the words the FA accepts are in L_1 which will lead to confusion.
  - Rather we would say that L_2 is accepted by this FA.

That’s why FA is also called language recognizer.

4



Mathematical Representation

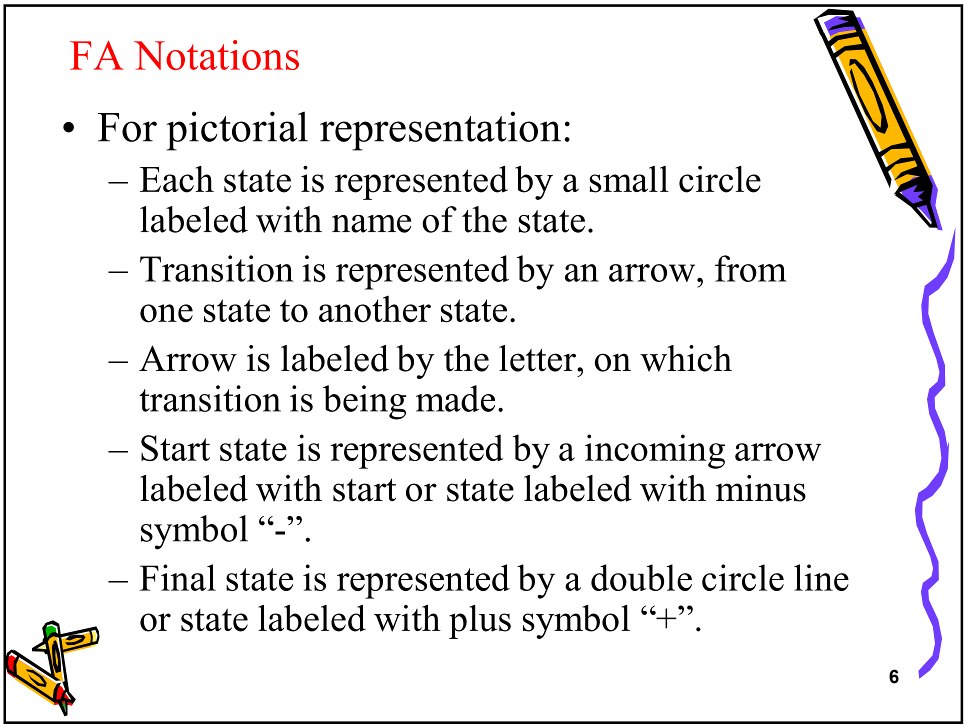
- To give mathematical representation to FA definition.
  - A finite set of states Q = { q_0, q_1, q_2, -----, q_n }.
  - A state q_0 of Q is the start state.

    $q_0 \in Q$
  - A subset of Q is called the final states.

    $F \in Q$
  - An finite set of alphabet ∑ = { x_1, x_2, x_3, ----- x_n }.
  - A transition function δ (lowercase Greek delta)associating each pair of states and letter with a state:

    $\delta (q_i, x_j) = q_k$
- Therefore FA can be represented as:

  FA = (Q, S, F, ∑ ,δ)

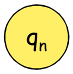


FA Notations

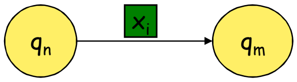
- For pictorial representation:
  - Each state is represented by a small circle labeled with name of the state.
  - Transition is represented by an arrow, from one state to another state.
  - Arrow is labeled by the letter, on which transition is being made.
  - Start state is represented by a incoming arrow labeled with start or state labeled with minus symbol “-”.
  - Final state is represented by a double circle line or state labeled with plus symbol “+”.

Notations.

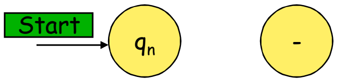
- State



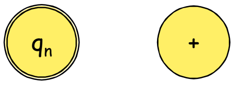
- Transition:



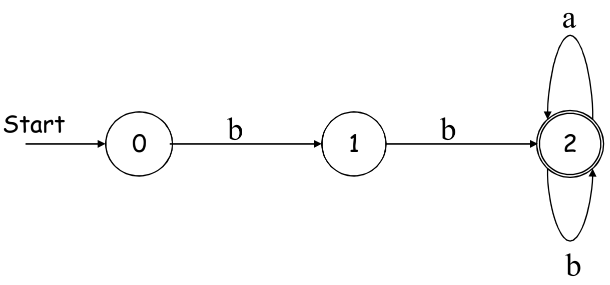
- Start State:



- Final State



Example.

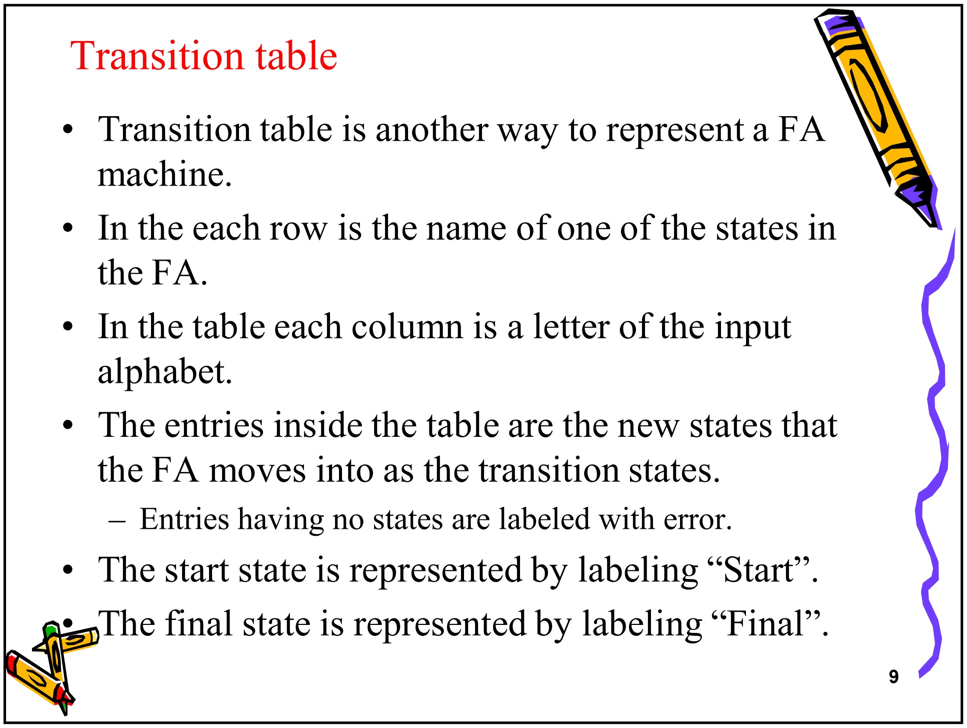


1. The words that can be accepted by this FA machine are:

   L = { bb, bba, bbaa,bbaba, bbbb, bbaabba, ---- }

2. Therefore the RE for this FA machine will be:

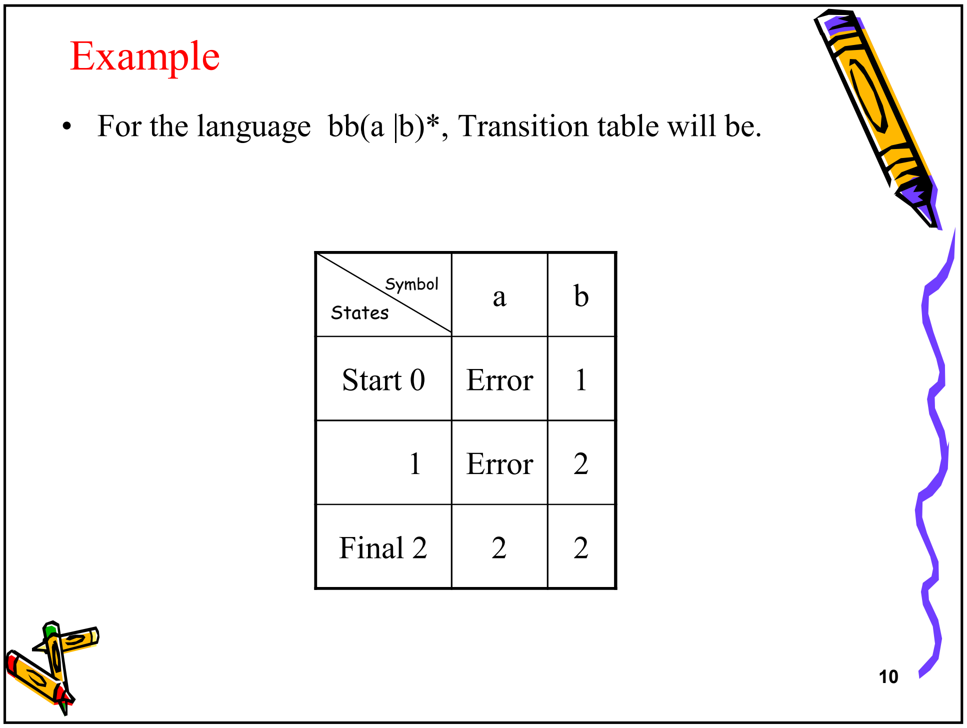
   bb (a | b)*



Transition table

- Transition table is another way to represent a FA machine.
- In the each row is the name of one of the states in the FA.
- In the table each column is a letter of the input alphabet.
- The entries inside the table are the new states that the FA moves into as the transition states.
  - Entries having no states are labeled with error.
- The start state is represented by labeling "Start".
- The final state is represented by labeling "Final".

9

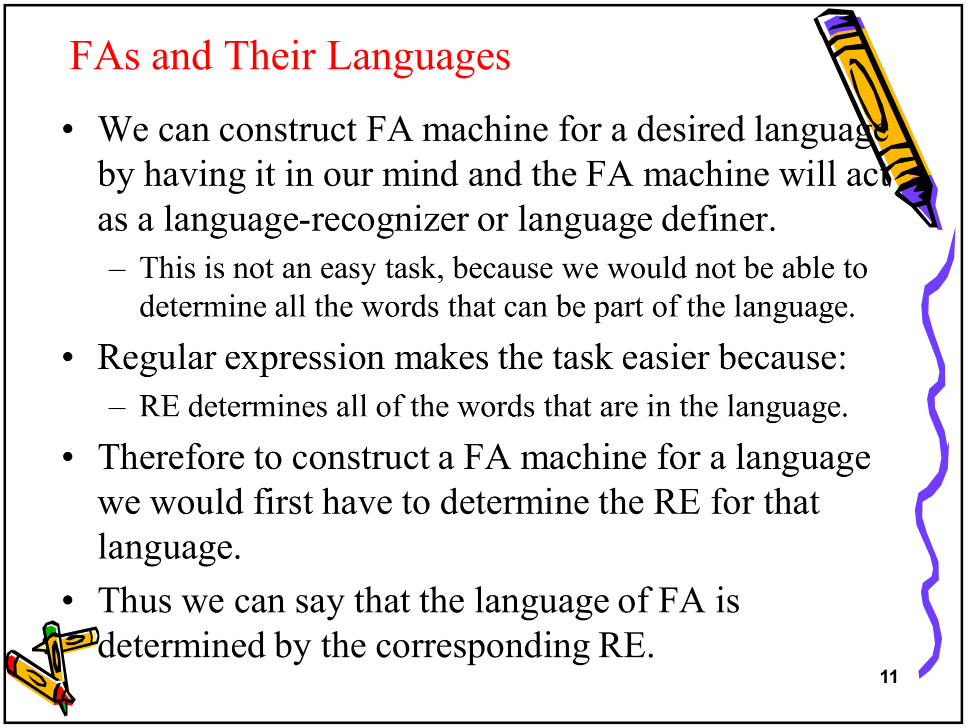


Example

- For the language bb(a |b)*, Transition table will be.

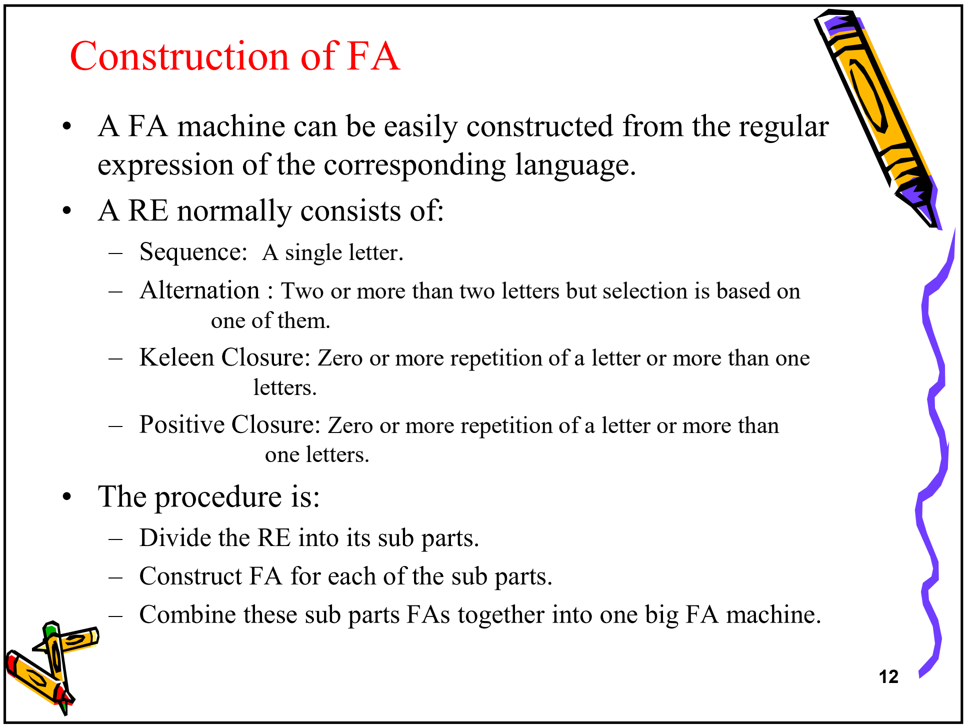
Symbol States	a	b
Start 0	Error	1
1	Error	2
Final 2	2	2

10



FAs and Their Languages

- We can construct FA machine for a desired language by having it in our mind and the FA machine will act as a language-recognizer or language definer.
  - This is not an easy task, because we would not be able to determine all the words that can be part of the language.
- Regular expression makes the task easier because:
  - RE determines all of the words that are in the language.
- Therefore to construct a FA machine for a language we would first have to determine the RE for that language.
- Thus we can say that the language of FA is determined by the corresponding RE.

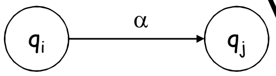


Construction of FA

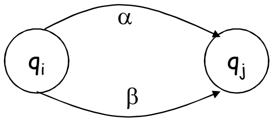
- A FA machine can be easily constructed from the regular expression of the corresponding language.
- A RE normally consists of:
  - Sequence: A single letter.
  - Alternation : Two or more than two letters but selection is based on one of them.
  - Keleen Closure: Zero or more repetition of a letter or more than one letters.
  - Positive Closure: Zero or more repetition of a letter or more than one letters.
- The procedure is:
  - Divide the RE into its sub parts.
  - Construct FA for each of the sub parts.
  - Combine these sub parts FAs together into one big FA machine.

Example

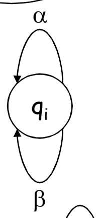
- Sequence (a single letter):



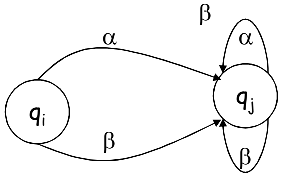
- Alternation (α | β):



- Keleen Closure (α | β)* :

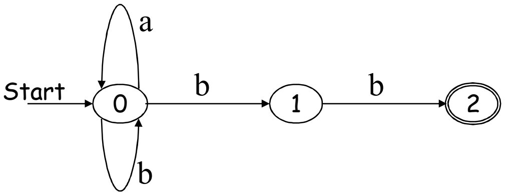


- Positive Closure (α | β)+



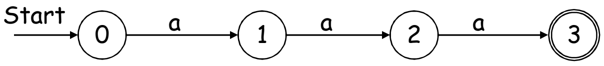
Example

- Construct FA for the RE (a | b)* bb.
  - This RE can be divided into three parts.
    - First is the keleen Closure (a | b)*.
    - Second is the sequence (single b).
    - Third is the sequence (single b).
  - Construct FA for each of the sub part and then combine them in the order as they are occurring in the RE.
  - The resulting FA machine will be:

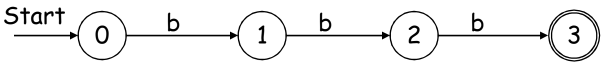


Example

- Build a FA that accepts all words containing a triple letter either aaa or bbb (only those words).
  - RE will be (aaa | bbb).
  - This RE can be divided into two sub parts that aaa and bbb.
  - First part (aaa) can be further divided into three sub parts (sequences) and similarly the second part.
  - For aaa it will be as:

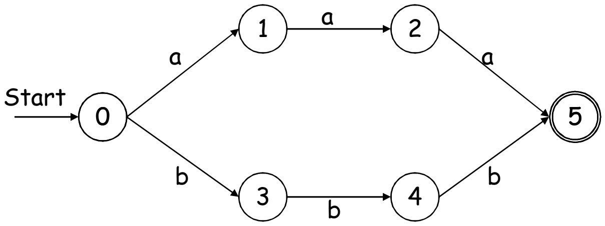


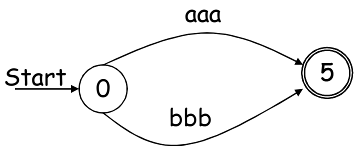
  - For bbb it will be as:



Contd.

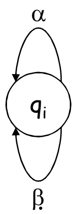
- Now by combining these parts we get.





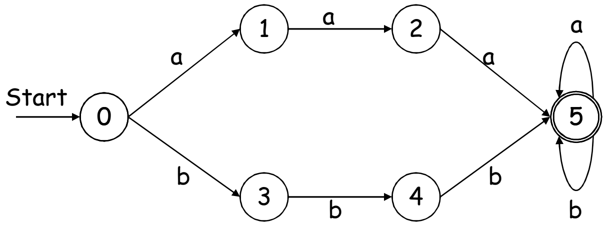
Example

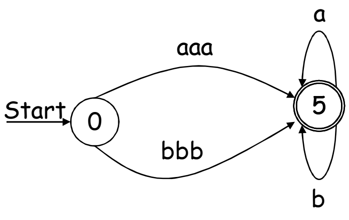
- Build FA for a language that strats with either triple (aaa) or triple (bbb) followed by any combination a’s and b’s.
  - RE will be (aaa | bbb) (a | b)*.
  - This is similar to the previous example but we have one more keleen closure.
    - (a | b)*



  - Combining this with the previous example we will get the require FA machine.

Contd.



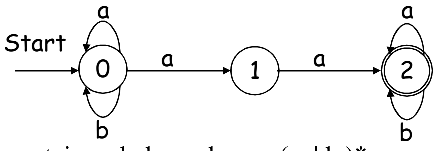


Language from FA Machine

- In the similar way we can also extract language from a FA machine.
- This proces is reverse of the building FA machine from RE.
- It works as:
  - Write RE for each of the sub part of the FA machine.
  - Combine REs in the order occurring in the FA machine.
  - Describe language on the basis of resultant RE.

Example

- Consider the FA machine.



  - State 0 contains a keleen closure (a | b)*.
  - State 0 contains a transition to state 1 on input a; which is sequence (a).
  - State 1 contains a transition to state 2 on input a; which is sequence (a).
  - State 2 contains a keleen closure (a | b)*.
  - By combining them we get:

    (a | b)* a a (a | b)*
  - Therefore the language is all words that contain a double letter (aa) somewhere.

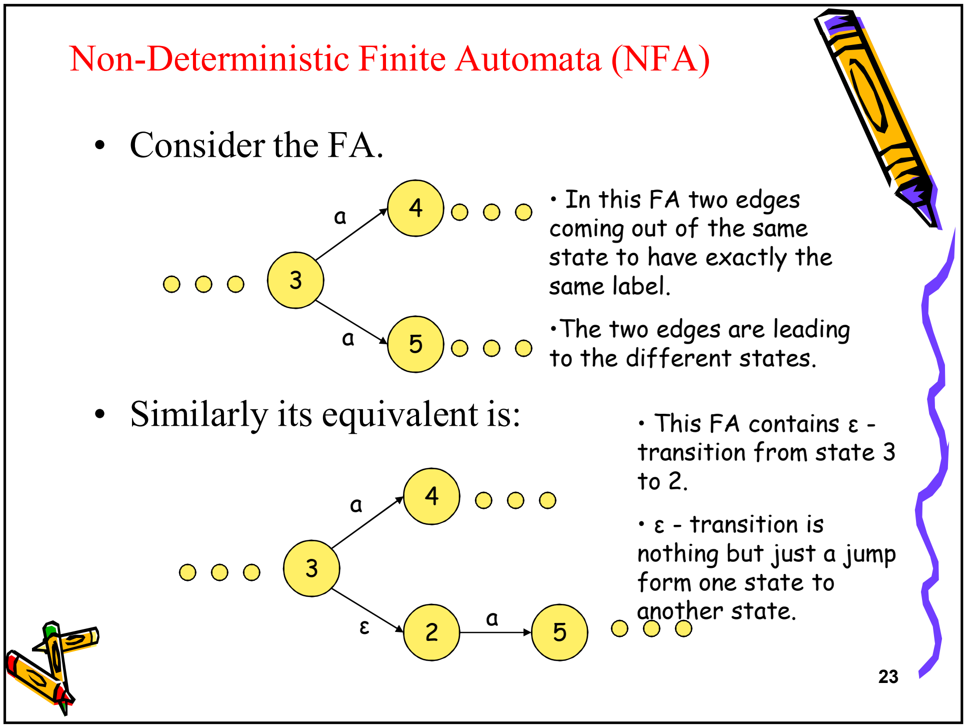
Example

- Build FA machine for the RE:

  (a | b)* (aa | bb) (a | b)*
- Build FA machine that accepts only the words baa, ab, abb only and no other strings.
- Build FA machine that accepts only those words that do not end with ba.
- Build FA machine that accepts only those words that begin or end with double letter (aa or bb).
- Write out the transition table for the FA machine on slide number 20.

Types of FA

- There are TWO types of FA.
  - Non-Deterministic Finite Automata (NFA).
  - Deterministic Finite Automata (DFA).



Non-Deterministic Finite Automata (NFA)

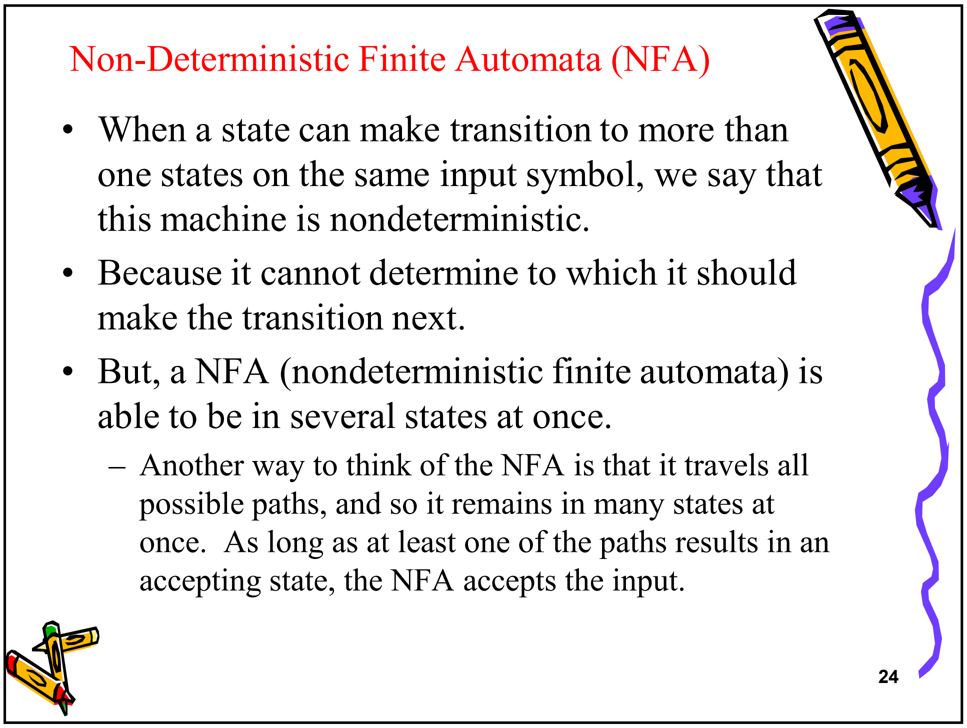
- Consider the FA.

- In this FA two edges coming out of the same state to have exactly the same label.
- The two edges are leading to the different states.

- Similarly its equivalent is:

- This FA contains ε - transition from state 3 to 2.
- ε - transition is nothing but just a jump form one state to another state.

23



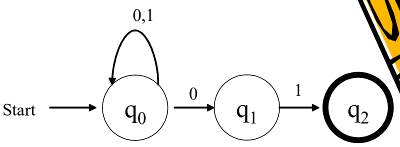
Non-Deterministic Finite Automata (NFA)

- When a state can make transition to more than one states on the same input symbol, we say that this machine is nondeterministic.
- Because it cannot determine to which it should make the transition next.
- But, a NFA (nondeterministic finite automata) is able to be in several states at once.
  - Another way to think of the NFA is that it travels all possible paths, and so it remains in many states at once. As long as at least one of the paths results in an accepting state, the NFA accepts the input.

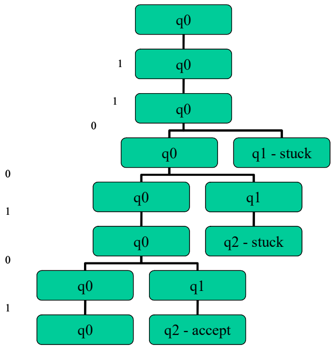
24

NFA Example

- This NFA accepts only those strings that end in 01



- Running in “parallel threads” for string 1100101



Formal Definition of FA

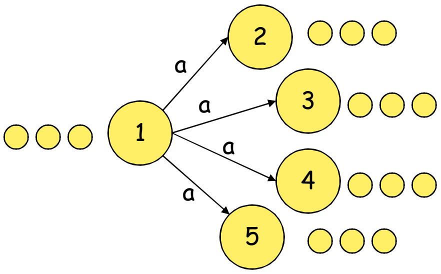
- Similar to FA a Nondeterministic Finite Automata (NFA) is a collection of the following things:
  - A finite set of states, typically Q.
  - One state is the start/initial state, typically q_0 such that:

    $$q_0 \in Q$$
  - A subset of Q is called the final states.

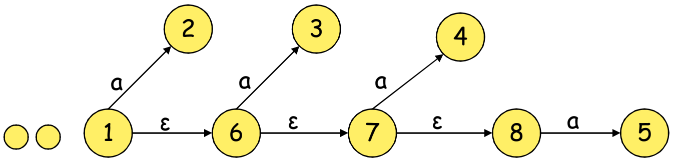
    $$F \in Q$$
  - An finite set of alphabet $\sum = \{ x_1, x_2, x_3, \text{-----} \ x_n \}$.
  - A finite set of transitions that describes how to proceed from each state to other states along edges labeled with letters of the alphabet.
    - There is the possibility of more than one edges with the same label from any state.
    - Some of the edges can be labeled with the ε – transition (empty).
- Therefore NFA can be represented as:

$$\text{NFA} = (Q, S, F, \sum, \delta)$$

Example

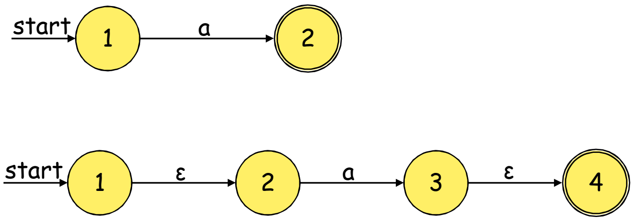


Is equivalent to.



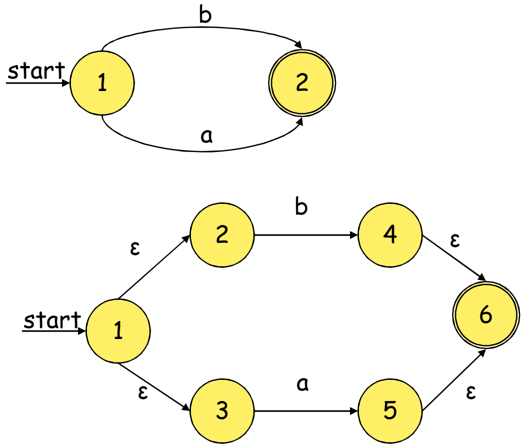
NFA for Sequence

- Sequence is a single alphabet "a" etc.



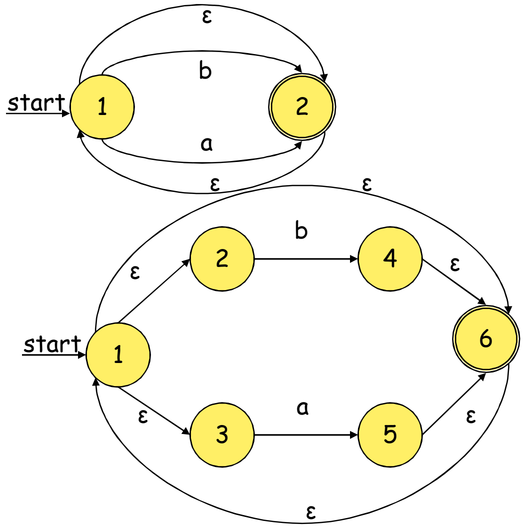
NFA for Alternation

- Alternation is (a | b).



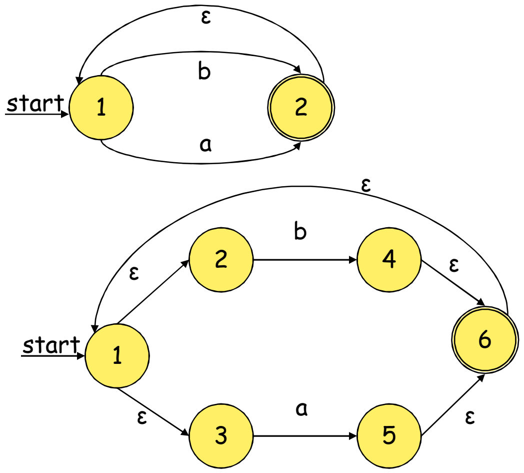
NFA for Kleene Closure

- Keleen Closure is (a | b)*.



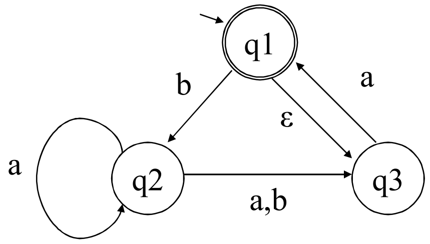
NFA for Positive Closure

- Positive Closure is (a | b)+.



NFA Example

- Practice with the following NFA to satisfy yourself that it accepts ε, a, baba, baa, and aa, but that it doesn't accept b, bb, and babba.



Theorem

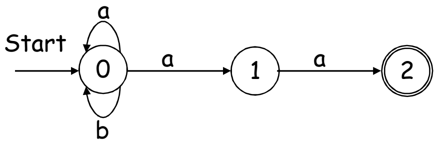
- The Theorem:

  FA = NFA

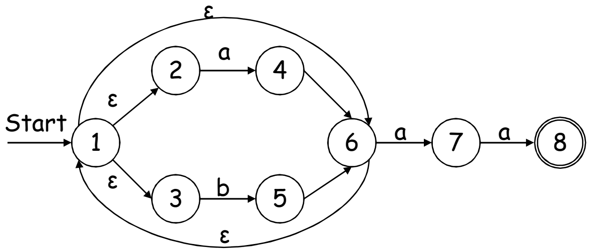
- By which we mean that any language definable by a nondeterministic finite automation is also definable by a finite automation and vice versa.
- If we take meaning that every NFA is itself an FA. This is not true and is a mistake.
- Only that for every FA there is a some NFA that is equivalent to it as a language accepter.

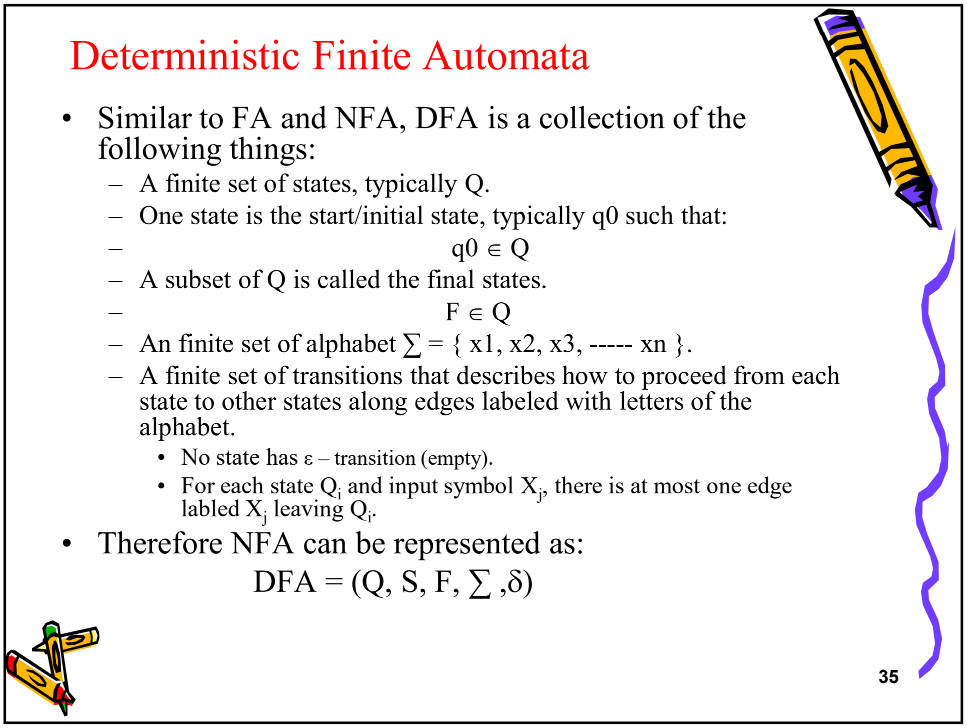
Proof

- Consider the following FA.



- Here state Zero (0) has more than one transition on the same input symbol "a".

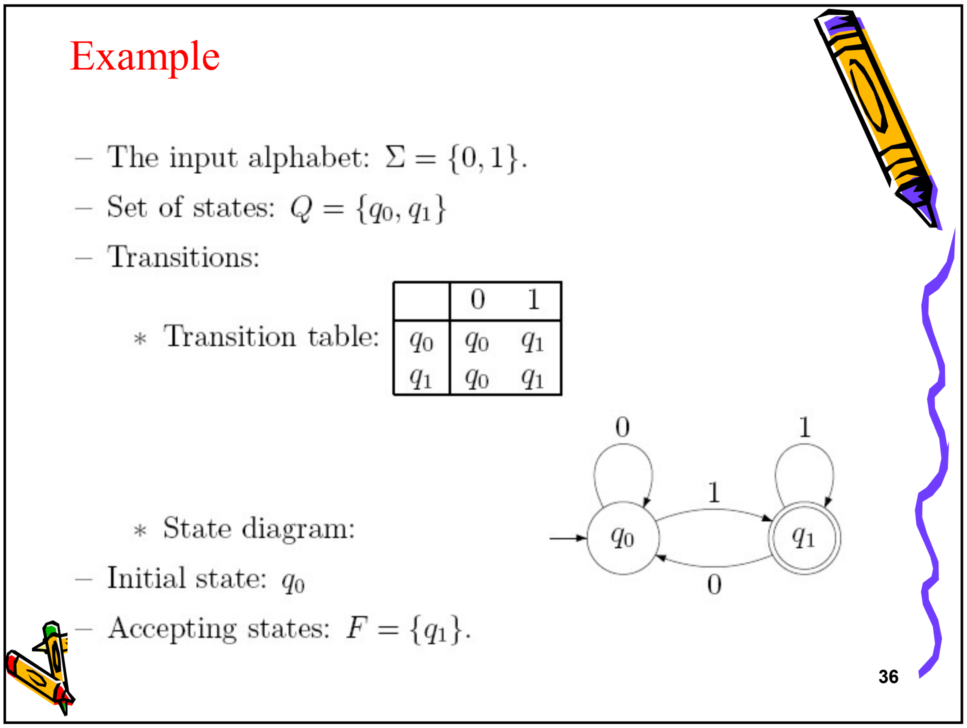




Deterministic Finite Automata

- Similar to FA and NFA, DFA is a collection of the following things:
  - A finite set of states, typically Q.
  - One state is the start/initial state, typically q0 such that:
  - $q0 \in Q$
  - A subset of Q is called the final states.
  - $F \in Q$
  - An finite set of alphabet $\sum = \{ x1, x2, x3,$ ----- $xn \}$.
  - A finite set of transitions that describes how to proceed from each state to other states along edges labeled with letters of the alphabet.
    - No state has ε – transition (empty).
    - For each state Q_i and input symbol X_j, there is at most one edge labled X_j leaving Q_i.
- Therefore NFA can be represented as:

$$DFA = (Q, S, F, \sum, \delta)$$



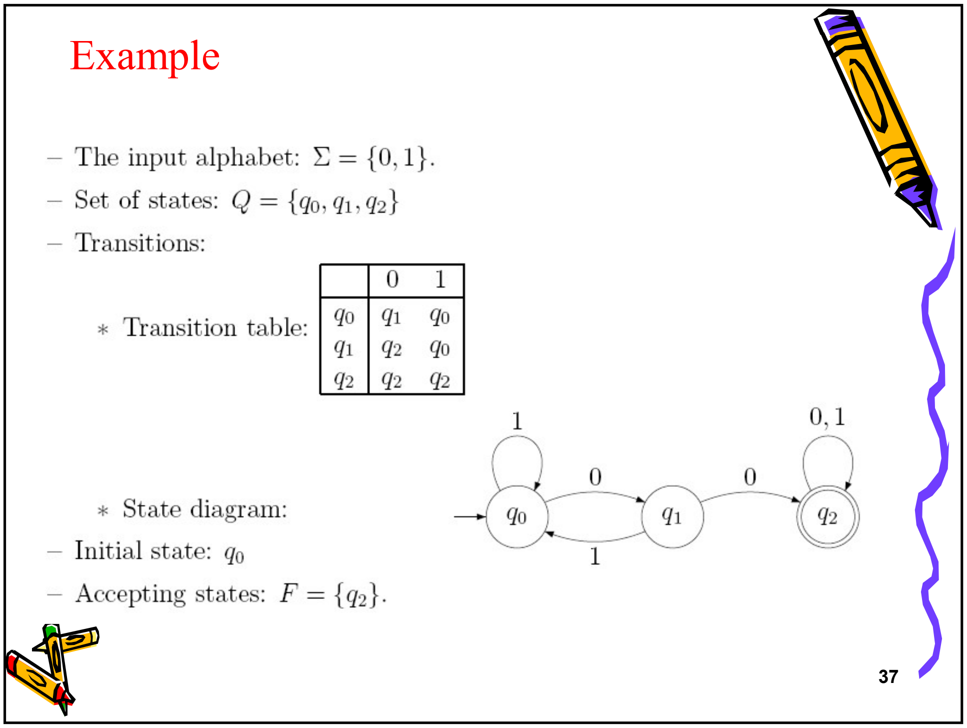
Example

- The input alphabet: $\Sigma = \{0, 1\}$.
- Set of states: $Q = \{q_0, q_1\}$
- Transitions:
  - ∗ Transition table:

	0	1
q_0	q_0	q_1
q_1	q_0	q_1

  - ∗ State diagram:

- Initial state: q_0
- Accepting states: $F = \{q_1\}$.



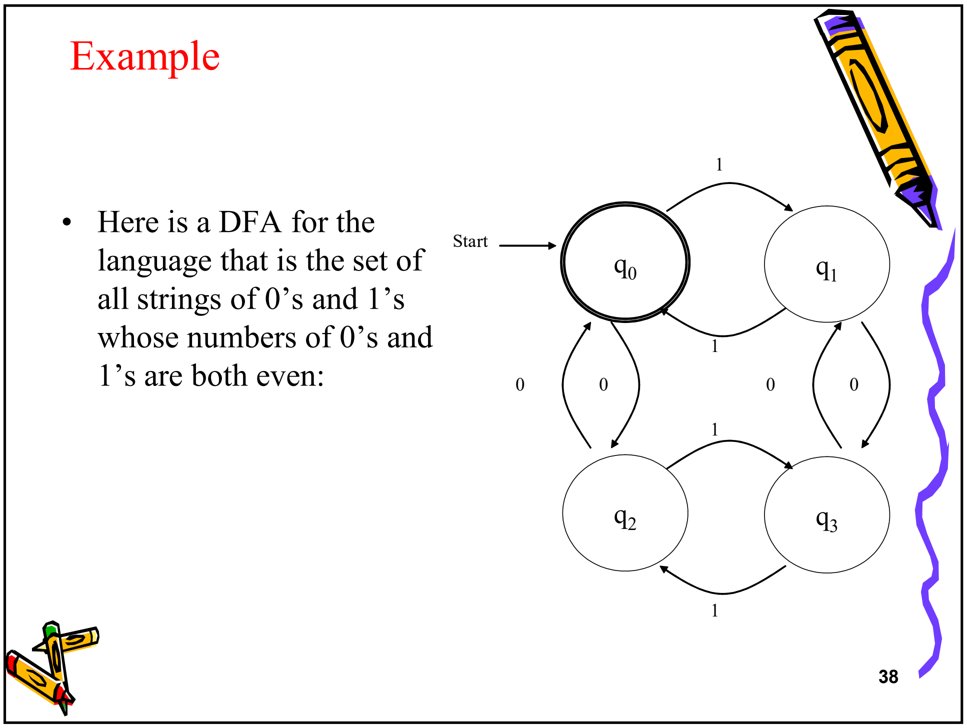
Example

- The input alphabet: $\Sigma = \{0, 1\}$.
- Set of states: $Q = \{q_0, q_1, q_2\}$
- Transitions:
  - $*$ Transition table:

	0	1
q_0	q_1	q_0
q_1	q_2	q_0
q_2	q_2	q_2

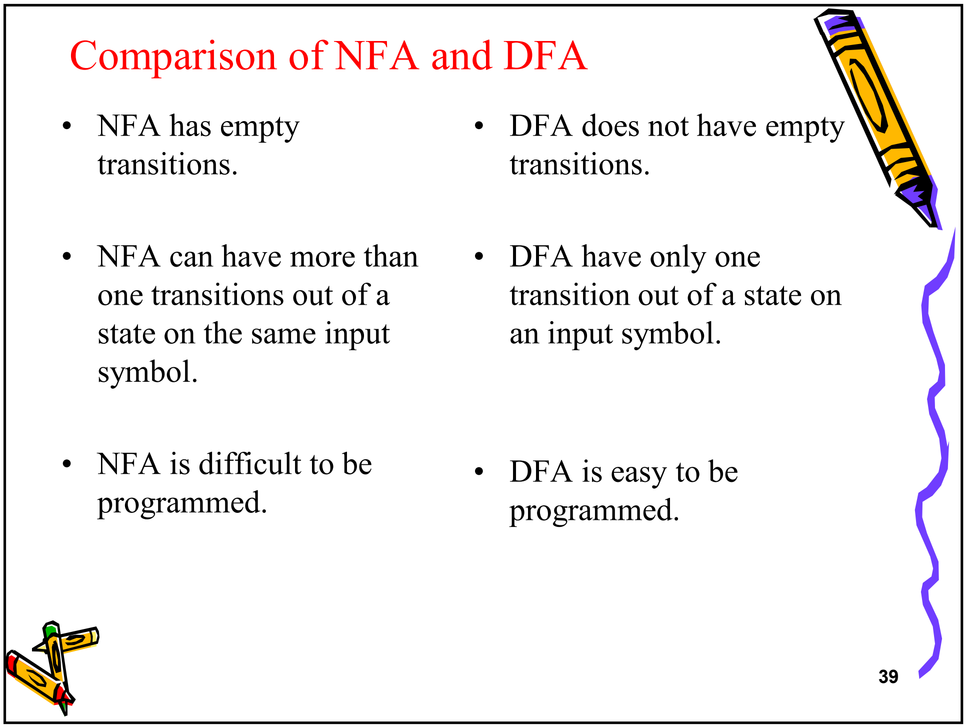
  - $*$ State diagram:

- Initial state: q_0
- Accepting states: $F = \{q_2\}$.



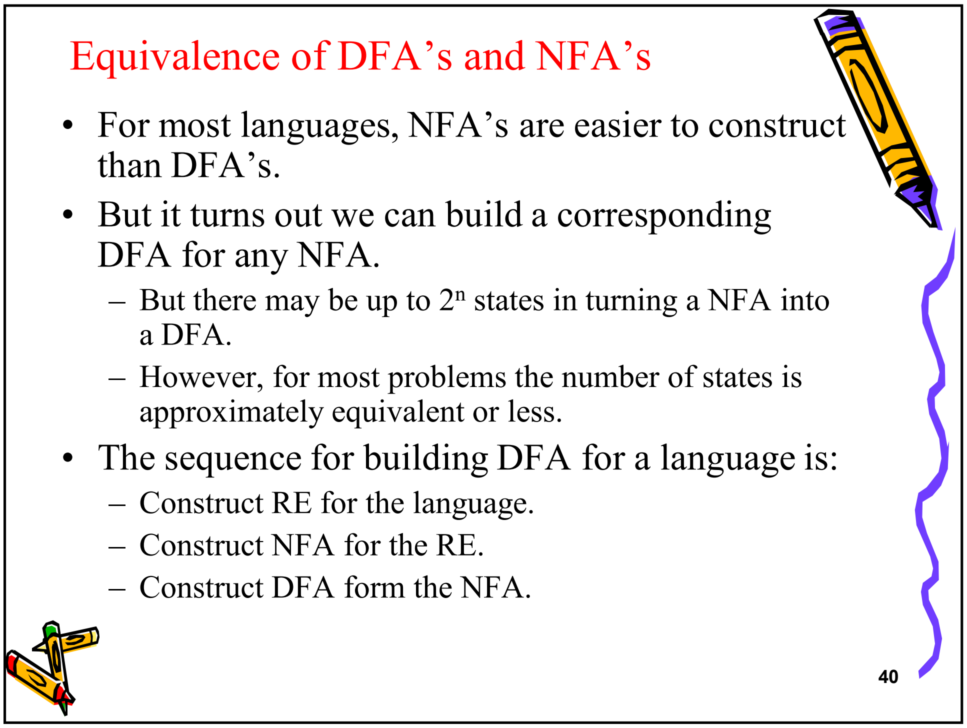
Example

- Here is a DFA for the language that is the set of all strings of 0's and 1's whose numbers of 0's and 1's are both even:



Comparison of NFA and DFA

• NFA has empty transitions.	• DFA does not have empty transitions.
• NFA can have more than one transitions out of a state on the same input symbol.	• DFA have only one transition out of a state on an input symbol.
• NFA is difficult to be programmed.	• DFA is easy to be programmed.



Equivalence of DFA’s and NFA’s

- For most languages, NFA’s are easier to construct than DFA’s.
- But it turns out we can build a corresponding DFA for any NFA.
  – But there may be up to 2^n states in turning a NFA into a DFA.
  – However, for most problems the number of states is approximately equivalent or less.
- The sequence for building DFA for a language is:
  – Construct RE for the language.
  – Construct NFA for the RE.
  – Construct DFA form the NFA.

Theorem

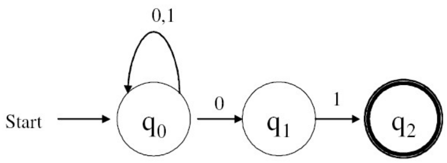
- The theorem is:
  - A language L is accepted by some DFA if and only if L is accepted by some NFA.

$$L(DFA) = L(NFA)$$

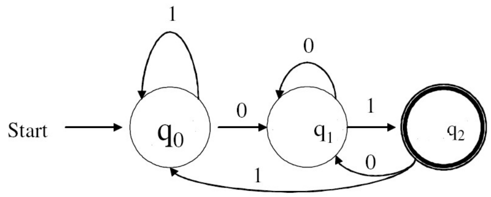
- Proof:
  - This assumption is true if we can construct DFA from the NFA.
  - We can construct DFA from NFA.
  - But it is very difficult to construct NFA from DFA.

Proof

- Consider the NFA machine.



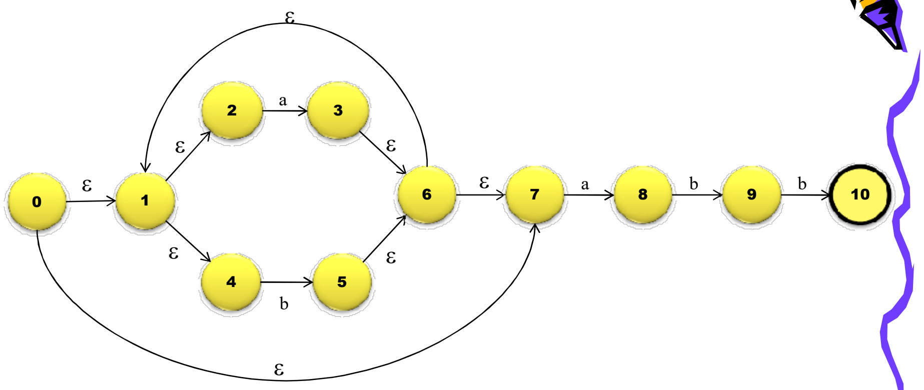
- Its equivalent DFA is.



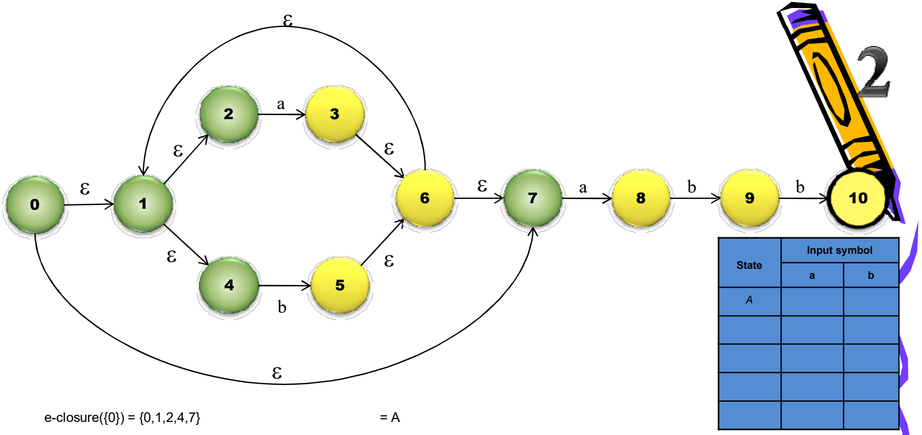
Conversion of NFA to DFA

- It is very difficult and error prone to write directly DFA for a language.
- Therefore we have to first write NFA for the language and then convert it into corresponding DFA.
- There are two methods to convert NFA to DFA.
  - □ ε-Closure Method.
  - – Sub-Set Method.

ε-Closure Method.



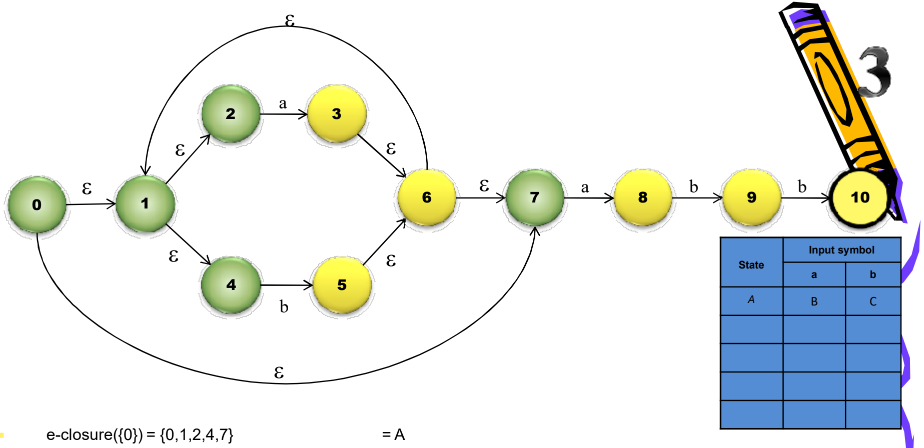
*NFA for (a | b)*abb*



State	Input symbol	
	a	b
A		

- e-closure({0}) = {0,1,2,4,7} = A

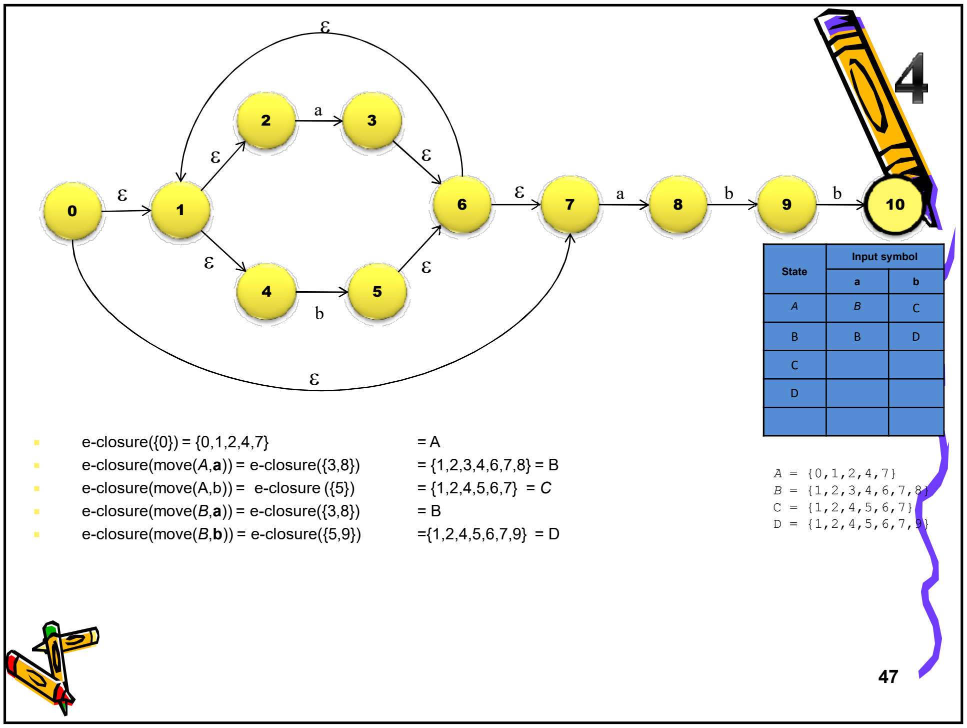
A = {0,1,2,4,7}



State	Input symbol	
	a	b
A	B	C

- e-closure({0}) = {0,1,2,4,7} = A
- e-closure(move(*A*,**a**)) = e-closure({3,8}) = {1,2,3,4,6,7,8} = B
- e-closure(move(A,b)) = e-closure ({5}) = {1,2,4,5,6,7} = *C*

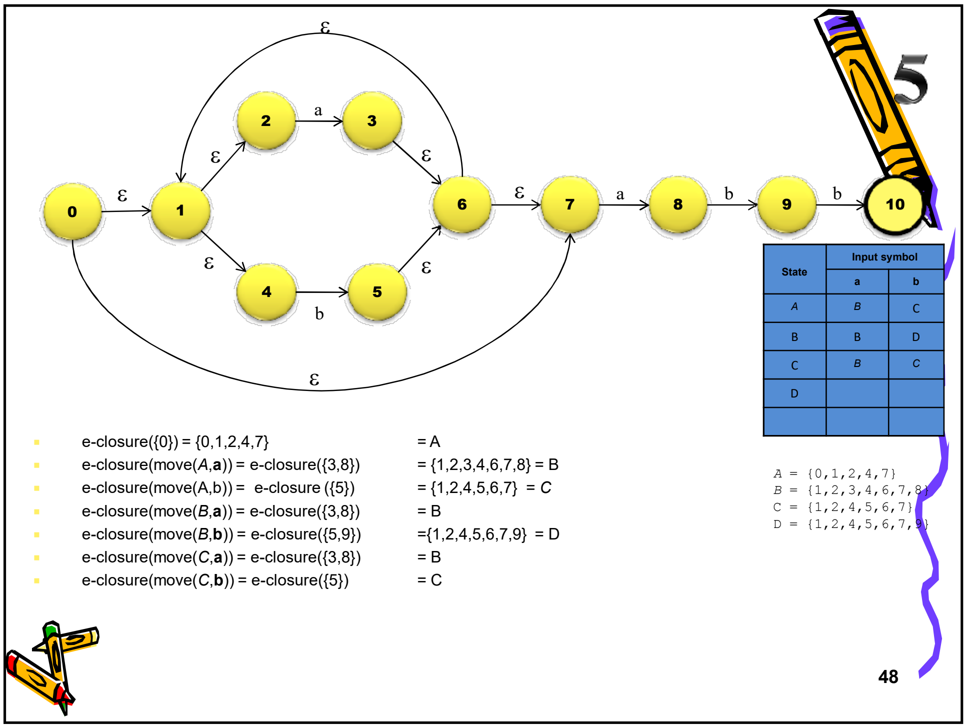
A = {0,1,2,4,7}
B = {1,2,3,4,6,7,8}
C = {1,2,4,5,6,7}



State	Input symbol	
	a	b
A	*B*	C
B	B	D
C		
D		

- e-closure({0}) = {0,1,2,4,7} = A
- e-closure(move(*A*,**a**)) = e-closure({3,8}) = {1,2,3,4,6,7,8} = B
- e-closure(move(A,b)) = e-closure ({5}) = {1,2,4,5,6,7} = *C*
- e-closure(move(*B*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*B*,**b**)) = e-closure({5,9}) ={1,2,4,5,6,7,9} = D

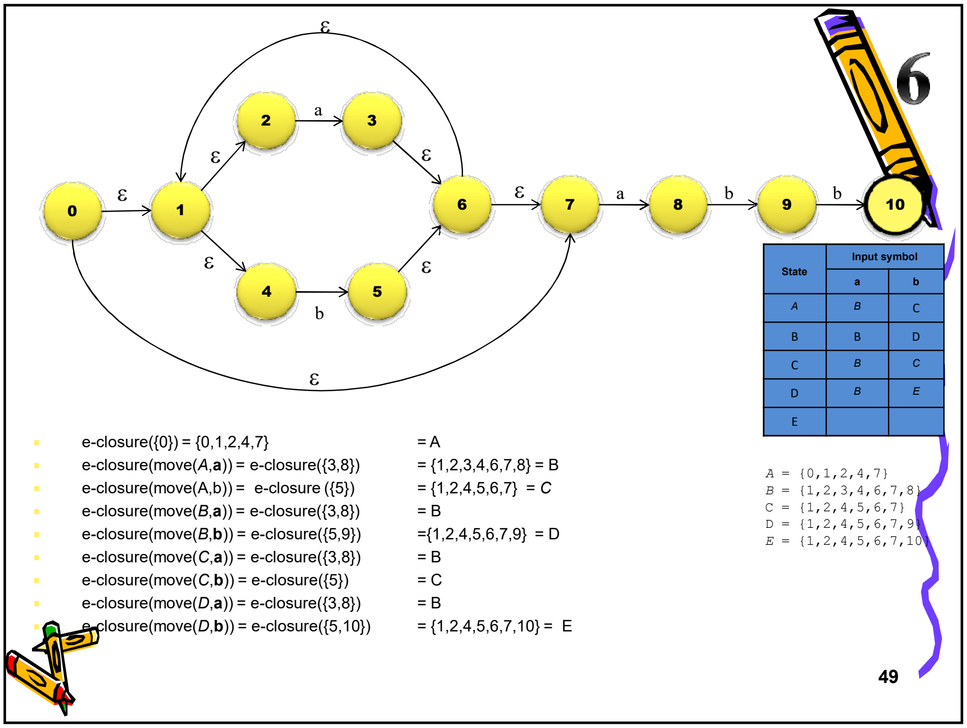
```
A = {0,1,2,4,7}
B = {1,2,3,4,6,7,8}
C = {1,2,4,5,6,7}
D = {1,2,4,5,6,7,9}
```



State	Input symbol	
	a	b
A	*B*	C
B	B	D
C	*B*	*C*
D		

- e-closure({0}) = {0,1,2,4,7} = A
- e-closure(move(*A*,**a**)) = e-closure({3,8}) = {1,2,3,4,6,7,8} = B
- e-closure(move(A,b)) = e-closure ({5}) = {1,2,4,5,6,7} = *C*
- e-closure(move(*B*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*B*,**b**)) = e-closure({5,9}) ={1,2,4,5,6,7,9} = D
- e-closure(move(*C*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*C*,**b**)) = e-closure({5}) = C

```
A = {0,1,2,4,7}
B = {1,2,3,4,6,7,8}
C = {1,2,4,5,6,7}
D = {1,2,4,5,6,7,9}
```

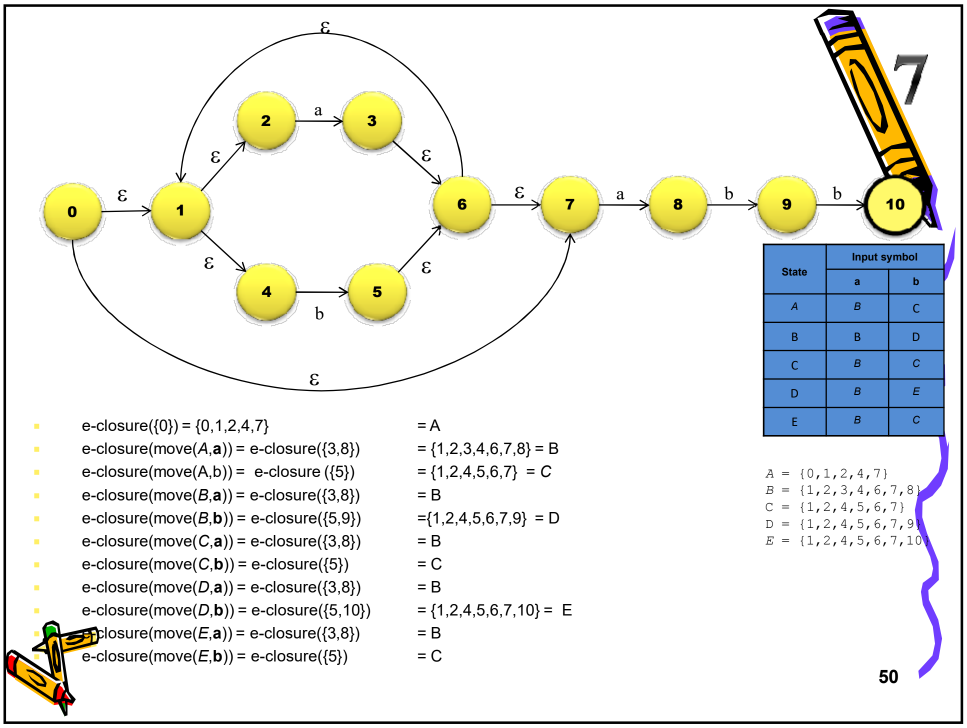


State	Input symbol	
	a	b
A	*B*	C
B	B	D
C	*B*	*C*
D	*B*	*E*
E		

```
A = {0,1,2,4,7}
B = {1,2,3,4,6,7,8}
C = {1,2,4,5,6,7}
D = {1,2,4,5,6,7,9}
E = {1,2,4,5,6,7,10}
```

- e-closure({0}) = {0,1,2,4,7} = A
- e-closure(move(*A*,**a**)) = e-closure({3,8}) = {1,2,3,4,6,7,8} = B
- e-closure(move(A,b)) = e-closure ({5}) = {1,2,4,5,6,7} = *C*
- e-closure(move(*B*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*B*,**b**)) = e-closure({5,9}) ={1,2,4,5,6,7,9} = D
- e-closure(move(*C*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*C*,**b**)) = e-closure({5}) = C
- e-closure(move(*D*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*D*,**b**)) = e-closure({5,10}) = {1,2,4,5,6,7,10} = E

49



State	Input symbol	
	a	b
A	*B*	C
B	B	D
C	*B*	*C*
D	*B*	*E*
E	*B*	*C*

```
A = {0,1,2,4,7}
B = {1,2,3,4,6,7,8}
C = {1,2,4,5,6,7}
D = {1,2,4,5,6,7,9}
E = {1,2,4,5,6,7,10}
```

- e-closure({0}) = {0,1,2,4,7} = A
- e-closure(move(*A*,**a**)) = e-closure({3,8}) = {1,2,3,4,6,7,8} = B
- e-closure(move(A,b)) = e-closure ({5}) = {1,2,4,5,6,7} = *C*
- e-closure(move(*B*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*B*,**b**)) = e-closure({5,9}) ={1,2,4,5,6,7,9} = D
- e-closure(move(*C*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*C*,**b**)) = e-closure({5}) = C
- e-closure(move(*D*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*D*,**b**)) = e-closure({5,10}) = {1,2,4,5,6,7,10} = E
- e-closure(move(*E*,**a**)) = e-closure({3,8}) = B
- e-closure(move(*E*,**b**)) = e-closure({5}) = C

50

Subset Construction

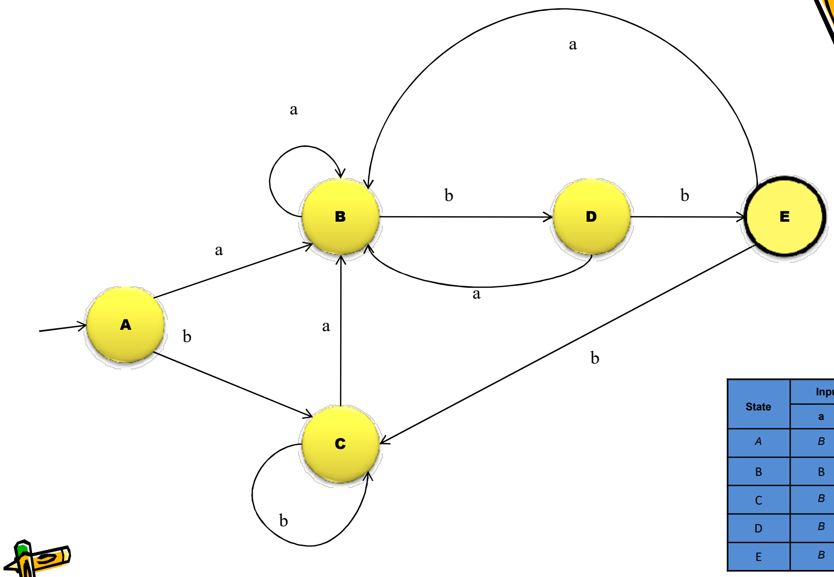
Eventually, the 5 sets are:

A={0,1,2,4,7}
B={1,2,3,4,6,7,8}
C={1,2,4,5,6,7}
D={1,2,4,5,6,7,9}
E={1,2,4,5,6,7,10}

A is start state
E is accepting state

State	Input symbol	
	a	b
A	*B*	*C*
B	*B*	*D*
C	*B*	*C*
D	*B*	*E*
E	*B*	*C*

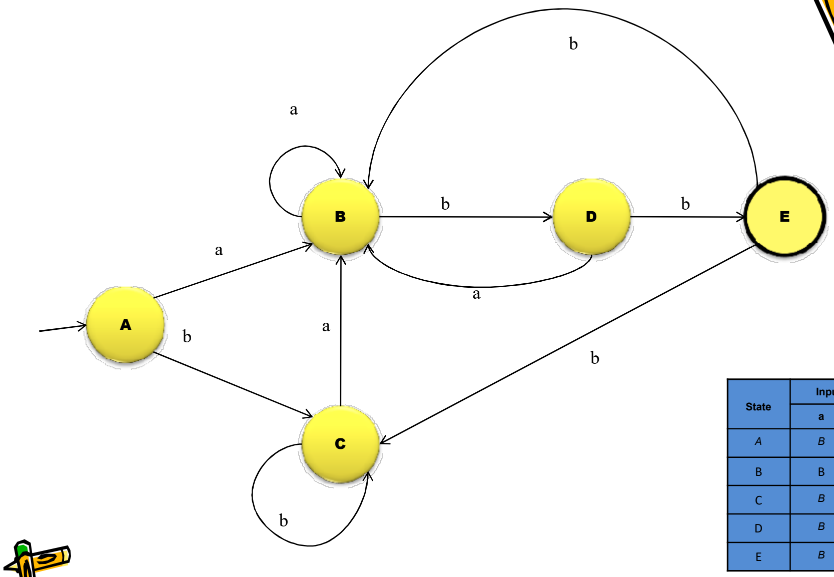
Final Transition Table



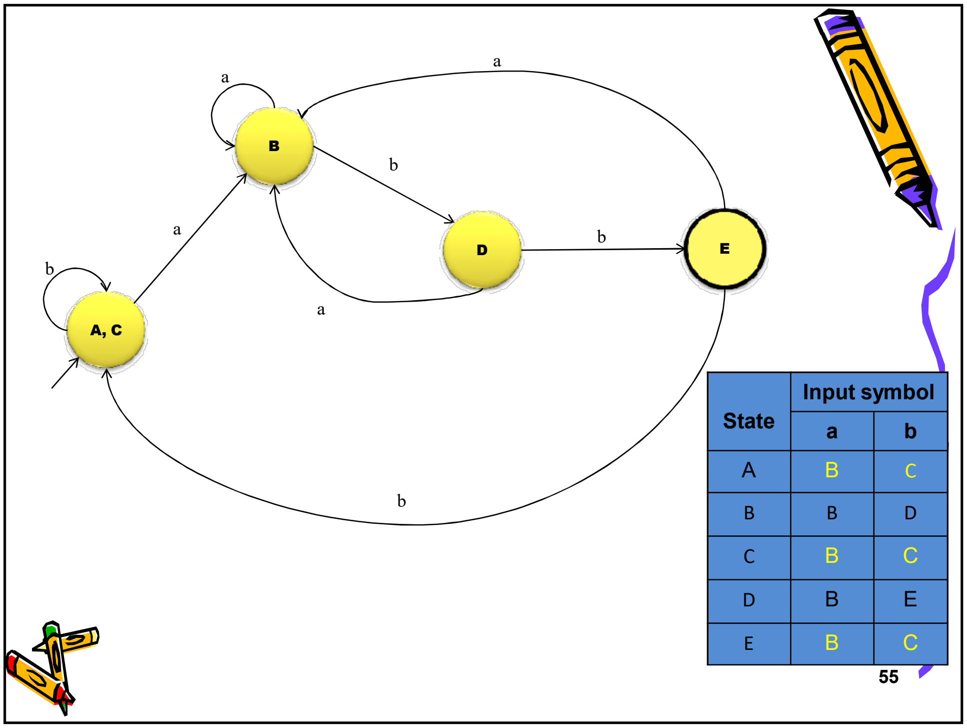
State	Input symbol	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

DFA Minimization

- The generated DFA may have a large number of states.
- Hopcroft’s algorithm: minimizes DFA states
- Idea: find groups of *equivalent states.*
- All transitions from states in one group G_1 go to states in the same group G_2
- Construct the minimized DFA such that there is one state for each group of states from the initial DFA.

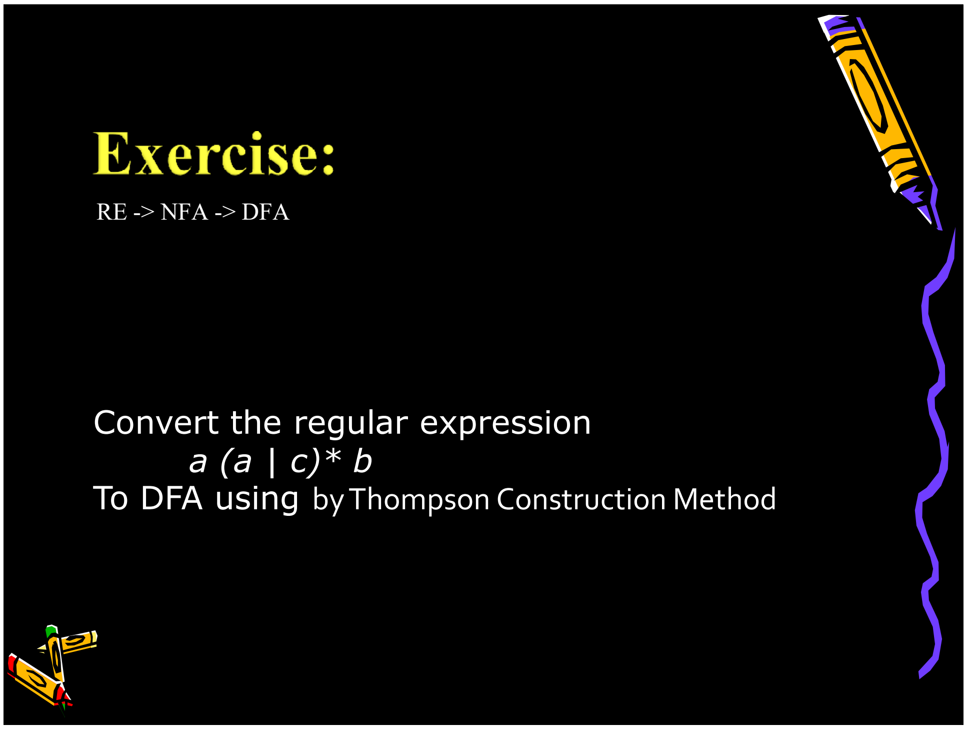


State	Input symbol	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C



State	Input symbol	
	a	b
A	B	C
B	B	D
C	B	C
D	B	E
E	B	C

55



Exercise:

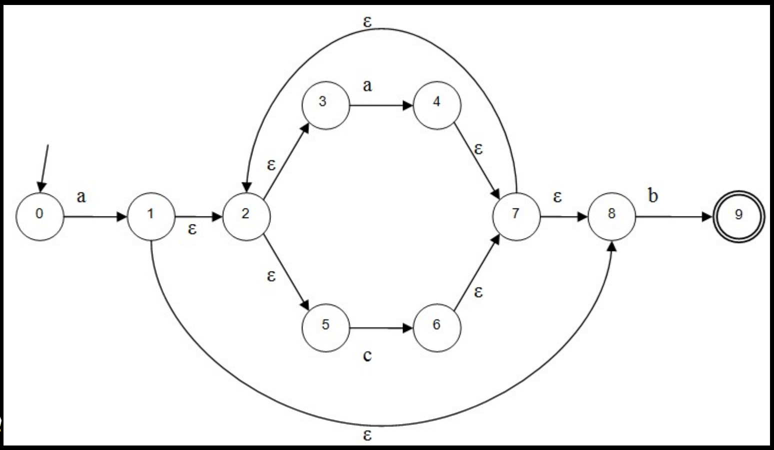
RE -> NFA -> DFA

Convert the regular expression

a (a | c) b*

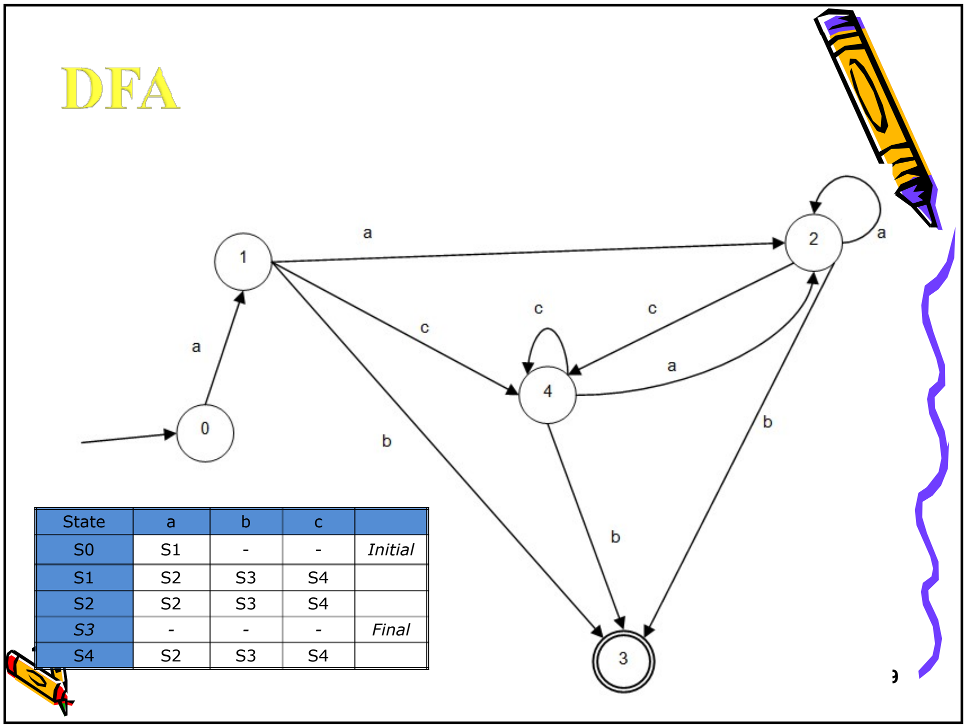
To DFA using by Thompson Construction Method

a (a | c)* b



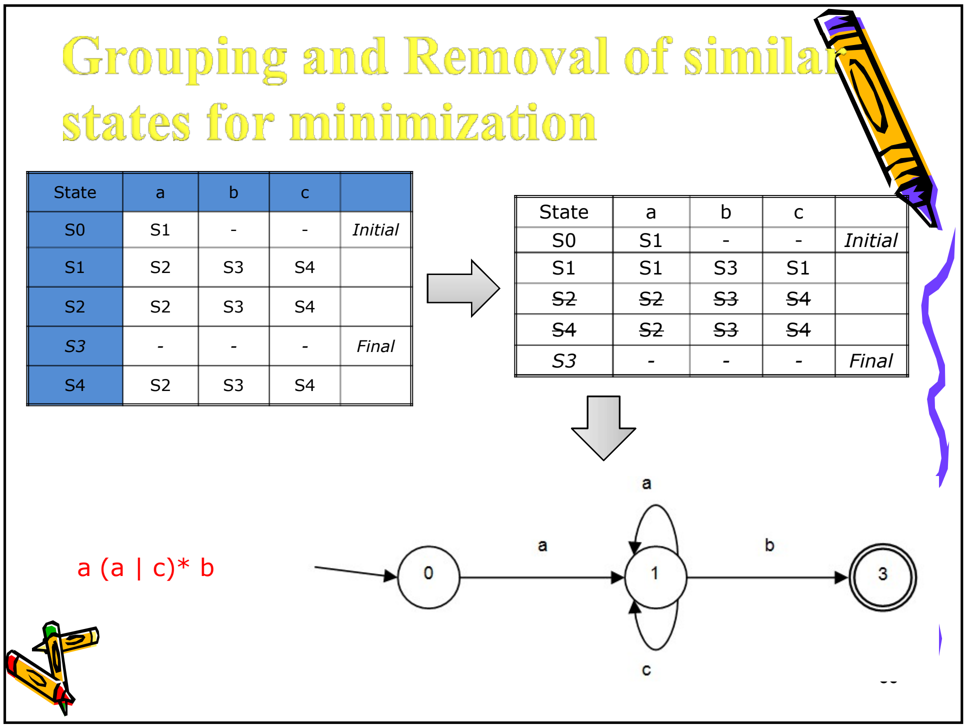
ε-closure Computation

ε-closure(0) = {0}	= S0
ε-closure(move(S0, a)) = ε-closure({1}) = {1,2,3,5,8}	= S1
ε-closure(move(S0, b)) = ε-closure({})	= no state
ε-closure(move(S0, c)) = ε-closure({})	= no state
ε-closure(move(S1, a)) = ε-closure({4}) = {4,7,8,2,3,5}	= S2
ε-closure(move(S1, b)) = ε-closure({9}) = {9}	= S3
ε-closure(move(S1, c)) = ε-closure({6}) = {6,7,8,2,3,5}	= S4
ε-closure(move(S2, a)) = ε-closure({4})	= S2
ε-closure(move(S2, b)) = ε-closure({9})	= S3
ε-closure(move(S2, c)) = ε-closure({6})	= S4
ε-closure(move(S3, a)) = ε-closure({})	= no state
ε-closure(move(S3, b)) = ε-closure({})	= no state
ε-closure(move(S3, c)) = ε-closure({})	= no state
ε-closure(move(S4, a)) = ε-closure({4})	= S2
ε-closure(move(S4, b)) = ε-closure({9})	= S3
ε-closure(move(S4, c)) = ε-closure({6})	= S4



DFA

State	a	b	c	
S0	S1	-	-	*Initial*
S1	S2	S3	S4	
S2	S2	S3	S4	
S3	-	-	-	*Final*
S4	S2	S3	S4	

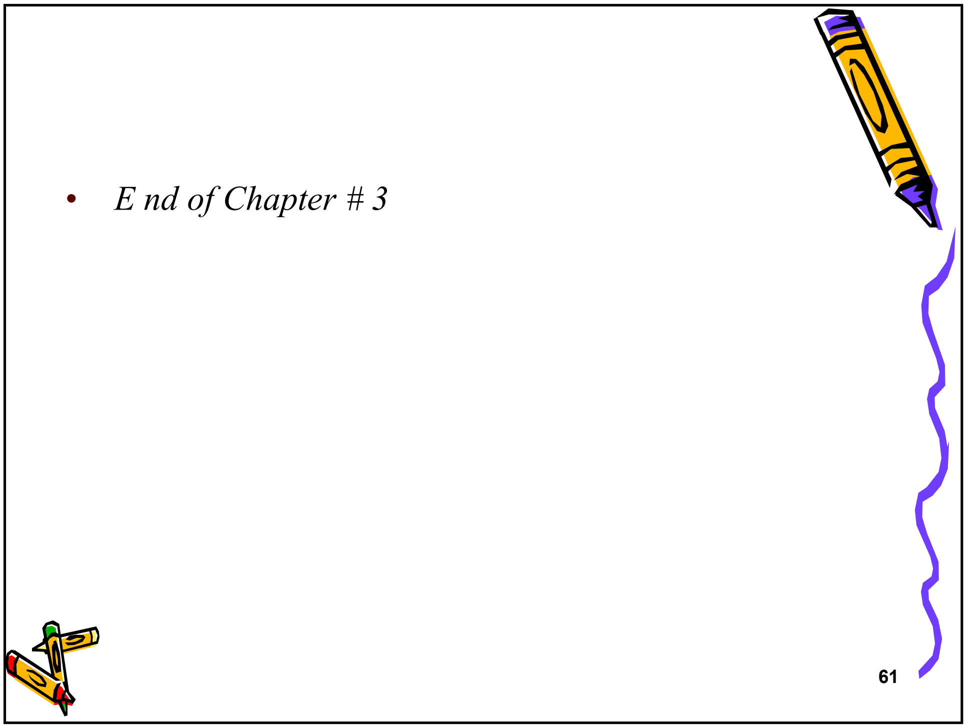


Grouping and Removal of similar states for minimization

State	a	b	c	
S0	S1	-	-	*Initial*
S1	S2	S3	S4	
S2	S2	S3	S4	
S3	-	-	-	*Final*
S4	S2	S3	S4	

State	a	b	c	
S0	S1	-	-	*Initial*
S1	S1	S3	S1	
~~S2~~	~~S2~~	~~S3~~	~~S4~~	
~~S4~~	~~S2~~	~~S3~~	~~S4~~	
S3	-	-	-	*Final*

a (a | c)* b



- *E nd of Chapter # 3*