

# *Biçimsel Diller ve Otomata Teorisi*

## *Sunu IX Düzenli Diller*

İZZET FATİH ŞENTÜRK



# *Closure Properties*

- A language that can be defined by a RE is called a **regular language**
- Are all languages regular?
  - The answer is no
- Before proving this fact, let us discuss some of the properties of regular languages

# *Properties of Regular Languages*

- If  $L_1$  and  $L_2$  are regular languages then
  - $L_1 + L_2$  (The language of all words either in  $L_1$  or  $L_2$ )
  - $L_1L_2$  (The language of all words formed by concatenating a word from  $L_1$  with a word from  $L_2$ )
  - $L_1^*$  are also regular languages (Strings that are concatenation of arbitrarily many factors from  $L_1$ )
- The set of regular languages is *closed* under union, concatenation, and Kleene closure

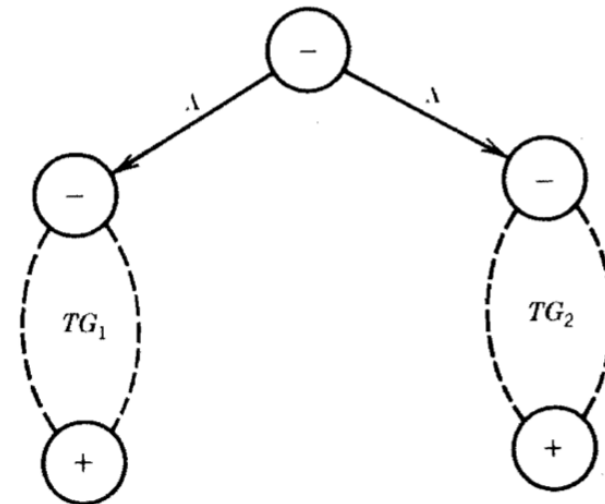
## *Proof 1 – By Regular Expressions*

- If  $L_1$  and  $L_2$  are regular languages, there are regular expressions  $r_1$  and  $r_2$  that define these languages. Then  $(r_1 + r_2)$  is a RE that defines the language  $L_1 + L_2$ . The language  $L_1L_2$  can be defined by the RE  $r_1r_2$ . The language  $L_1^*$  can be defined by the RE  $(r_1)^*$ 
  - All three of these sets of words are definable by regular expressions and so are themselves regular languages
- Regular languages can also be defined in terms of machines

## Proof 2 – By Machines

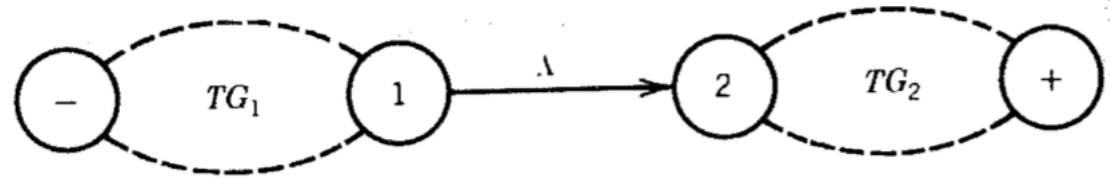
- $L_1$  and  $L_2$  are regular languages
  - There must be TGs that accept them
  - Let  $TG_1$  accept  $L_1$  and  $TG_2$  accept  $L_2$
  - $TG_1$  and  $TG_2$  each have a unique start state and a unique separate final state (If this is not the case originally, we can modify TGs so that it becomes true)

The TG accepts the language  $L_1 + L_2$   
This machines prove that  $L_1 + L_2$  is regular

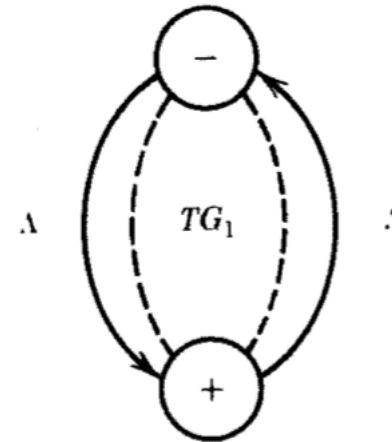


## Proof 2 – By Machines

The TG accepts the language  $L_1L_2$   
1 is the former + of  $TG_1$  and 2 is the former – of  $TG_2$

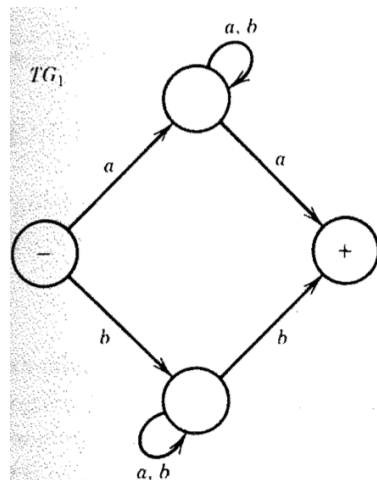


The TG accepts the language  $L_1^*$   
We begin at the – of  $TG_1$  and trace a path to the + of  $TG_1$



# Example

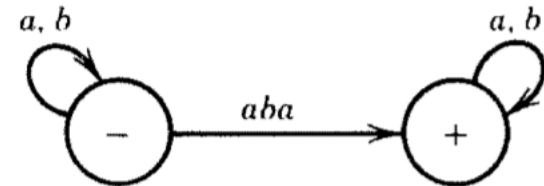
- Let the alphabet be  $\Sigma = \{a, b\}$  and
  - $L_1$  = all words of two or more letters that begin and end with the same letter
  - $L_2$  = all words that contain the substring  $aba$
- We will use the following TGs and REs



$$\mathbf{r_1}$$
$$\mathbf{a(a + b)^*a + b(a + b)^*b}$$

$$\mathbf{r_2}$$
$$\mathbf{(a + b)^*aba(a + b)^*}$$

$TG_2$



## Example

- The language  $L_1 + L_2$  is regular because it can be defined by the RE

$$[a(a + b)^*a + b(a + b)^*b] + [(a + b)^*aba(a + b)^*]$$

For the purpose of clarity, we have employed brackets instead of nested parantheses

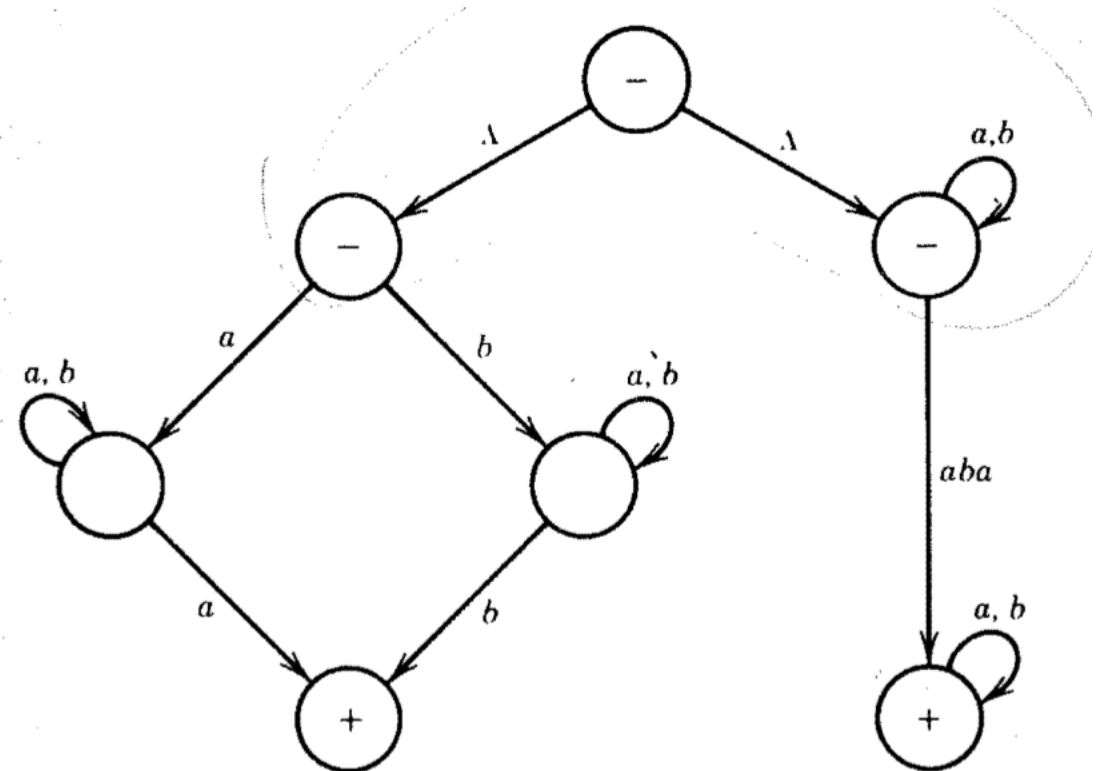
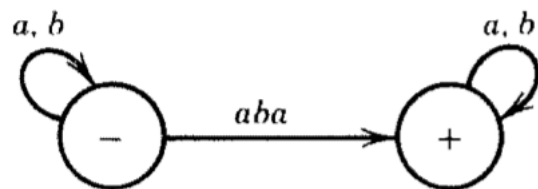
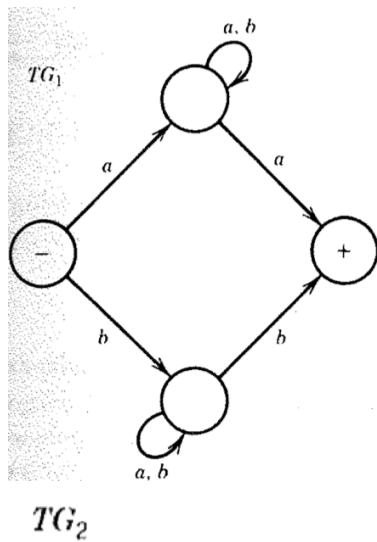
$$r_1$$
$$a(a + b)^*a + b(a + b)^*b$$

$$r_2$$
$$(a + b)^*aba(a + b)^*$$



# Example

$L_1 + L_2$  is accepted by the following TG



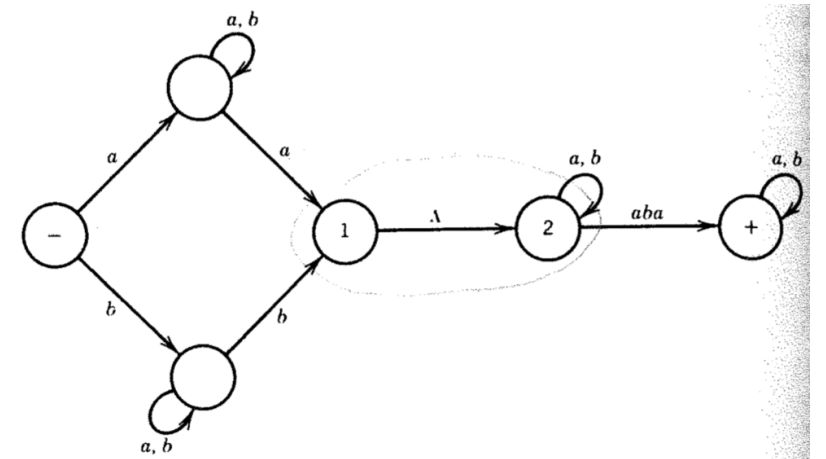
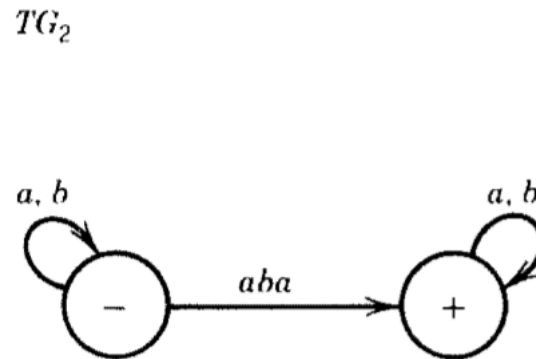
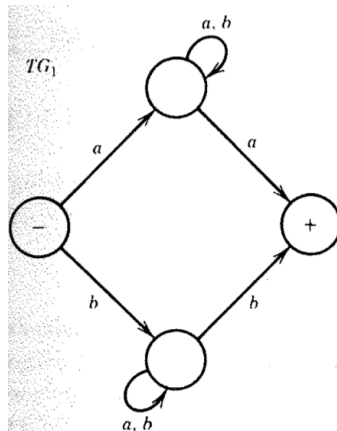
# Example

- The language  $L_1L_2$  is regular because it can be defined by the RE

$[a(a + b)^*a + b(a + b)^*b] \quad [(a + b)^*aba(a + b)^*]$

$\mathbf{r_1}$   
 $\mathbf{a(a + b)^*a + b(a + b)^*b}$   
 $\mathbf{r_2}$   
 $\mathbf{(a + b)^*aba(a + b)^*}$

- The language is accepted by the TG

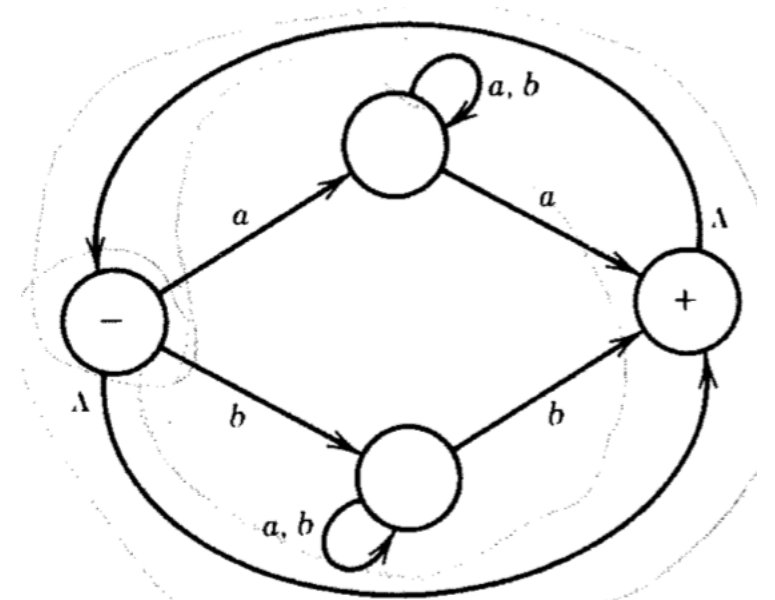
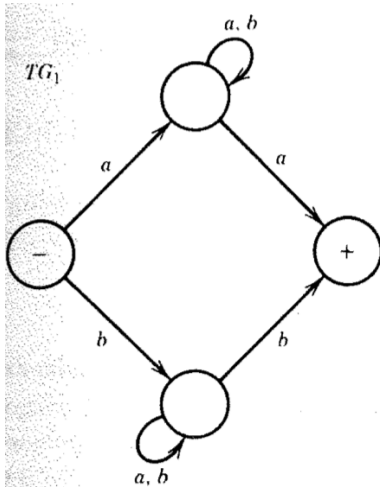


# Example

- The language  $L_1^*$  is regular because it can be defined by the RE

$$[a(a + b)^*a + b(a + b)^*b]^*$$

- The language is accepted by the TG



# *Complements and Intersections*

- If  $L$  is a language over the alphabet  $\Sigma$ , we define its **complement**,  $L'$ , to be the language of all strings of letters from  $\Sigma$  that are not words in  $L$
- Example: If  $L$  is the language over the alphabet  $\Sigma = \{a, b\}$  of all words that have a double  $a$  in them, then  $L'$  is the language of all words that do not have a double  $a$
- The complement of the language  $L'$  is the language  $L$ 
  - $(L')' = L$
- If  $L$  is a regular language, then  $L'$  is also a regular language
  - The set of regular languages is closed under complementation

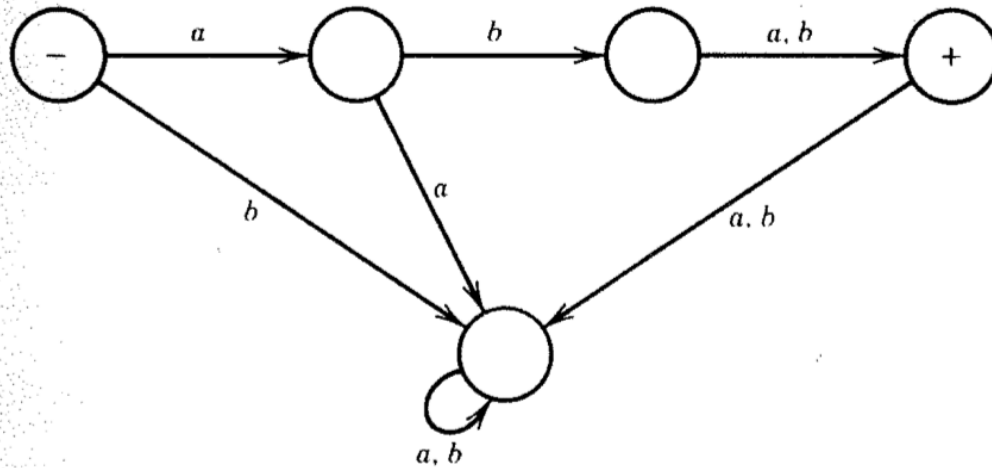
# Complements

- If  $L$  is a regular language, then  $L'$  is also a regular language
- Proof: If  $L$  is a regular language, there is some FA that accepts the language  $L$ 
  - Some of the states of this FA are final states and, most likely, some are not
  - Let us reverse the final status of each state; that is, if it was a final state, make it a nonfinal state, and if it was a nonfinal state, make it a final state
- If an input string formerly ended in a nonfinal state, it now ends in a final state and vice versa
- This new machine we have built accepts all input strings that were not accepted by the original FA and rejects all the input strings that the FA used to accept

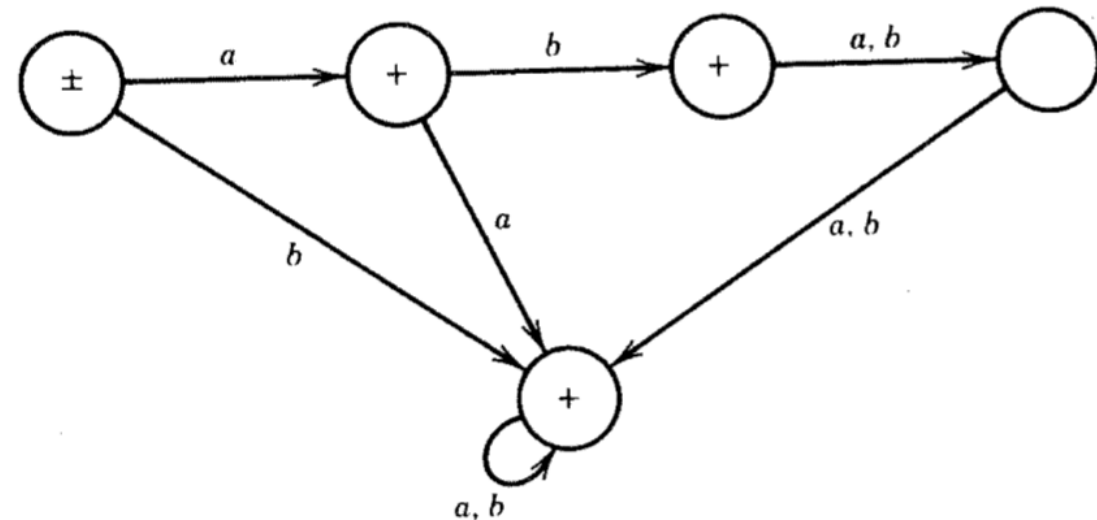
$$- \leftrightarrow \pm$$

# Example

- An FA that accepts only the strings *aba* and *abb*



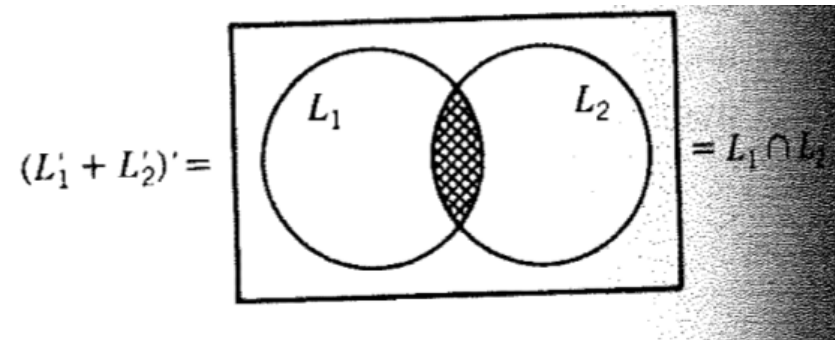
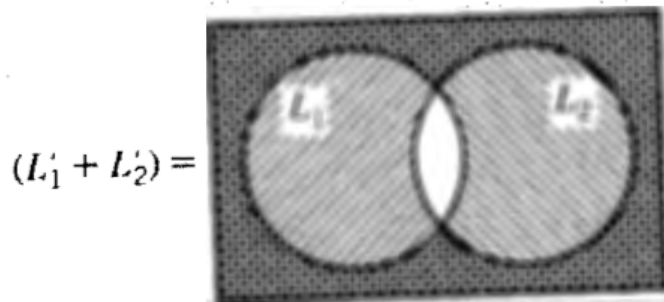
- An FA that accepts all strings other than *aba* and *abb*



# Intersections

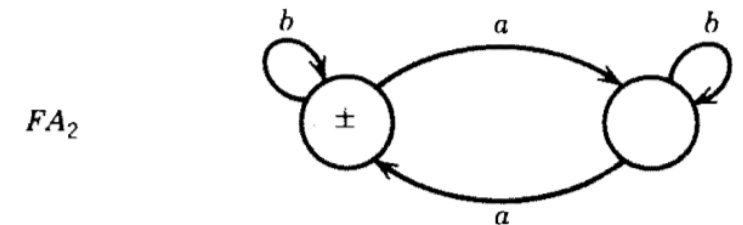
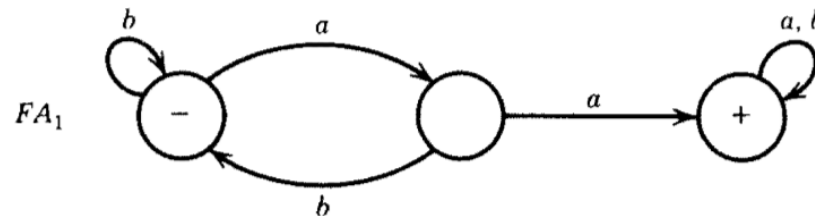
- If  $L_1$  and  $L_2$  are regular languages, then  $L_1 \cap L_2$  is also a regular language
  - The set of regular languages is closed under intersection
- Proof: By DeMorgan's law for sets of any kind (regular languages or not)
  - $L_1 \cap L_2 = (L_1' + L_2')'$

$L_1'$  and  $L_2'$  are regular  
 $L_1' + L_2'$  is regular



# Example

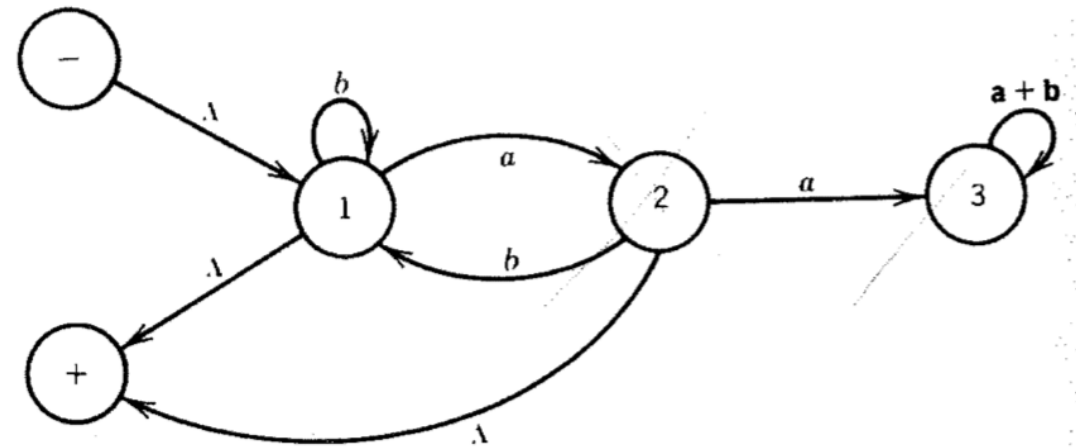
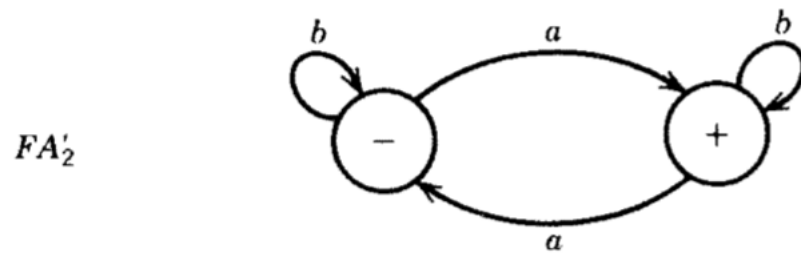
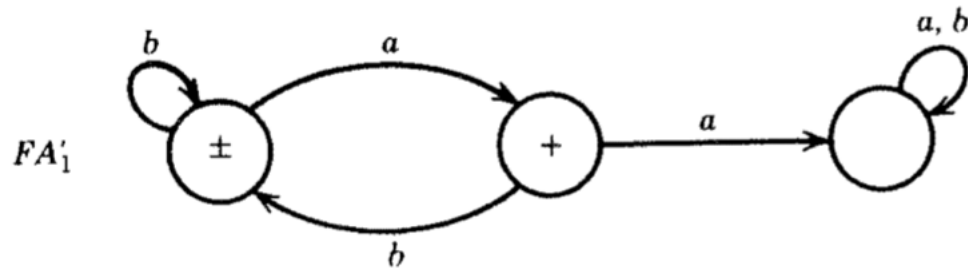
- Two languages,  $\Sigma = \{a, b\}$ 
  - $L_1$  = all strings with a double  $a$
  - $L_2$  = all strings with an even number of  $a$ 's
- These languages are not the same
  - $aaa$  is in  $L_1$  but not in  $L_2$
  - $aba$  is in  $L_2$  but not in  $L_1$
- They are both regular languages
  - $r_1 = (a + b)^*aa(a + b)^*$
  - $r_2 = b^*(ab^*ab^*)^*$





# Example

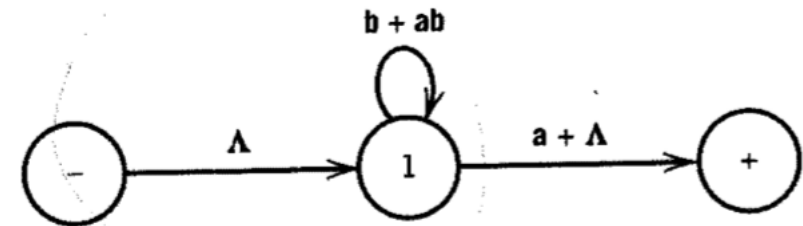
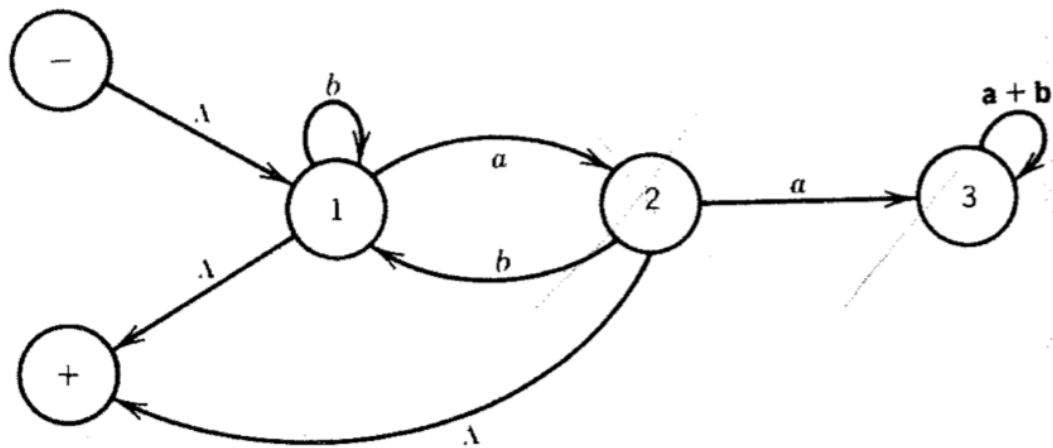
- The first step for  $L_1 \cap L_2$  is to find  $L_1'$  and  $L_2'$



State 3 is part of no path from  $-$  to  $+$ , it can be dropped

# Example

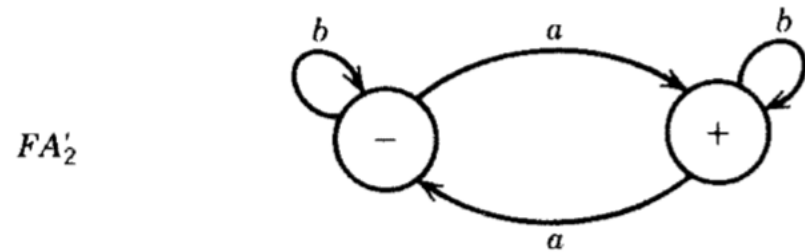
- To bypass state 2, we need to join the incoming a-edge with both outgoing edges (b-edge to 2 and  $\Lambda$ -edge to +)
  - When we add the two loops, we get  **$b + ab$**  and the sum of the two edges from 1 to + is  $a + \Lambda$



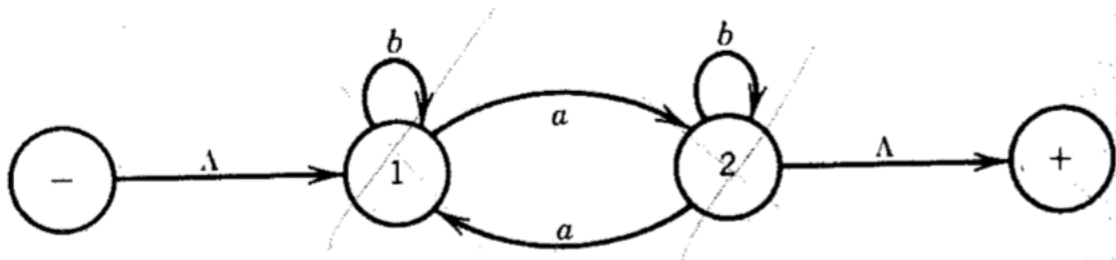
$$r_1' = (b + ab)^*(a + \Lambda)$$

## Example

- Let us do the same thing for  $L_2'$

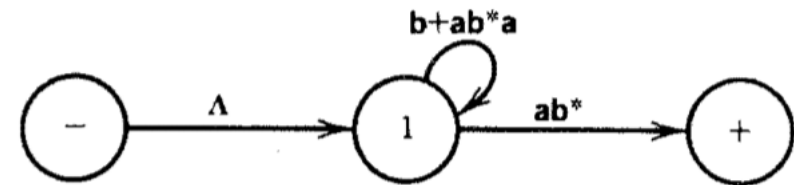
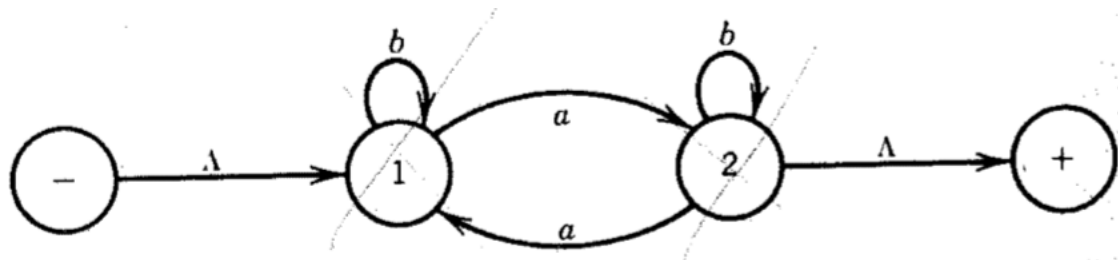


- $FA_2'$  becomes



# Example

- Start simplification by eliminating state 2
  - There is one incoming edge, a loop, and two outgoing edges
  - We need to replace them with only two edges:
    - The path 1-2-2-1 becomes a loop at 1
    - The path 1-2-2-+ becomes an edge from 1 to +
- After bypassing state 2 and adding the two loop labels, we have



$$r'_2 = (b + ab^*a)^*ab^*$$

## Example

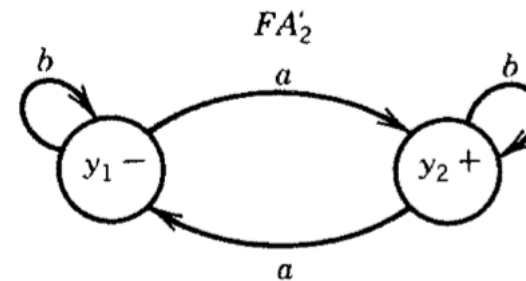
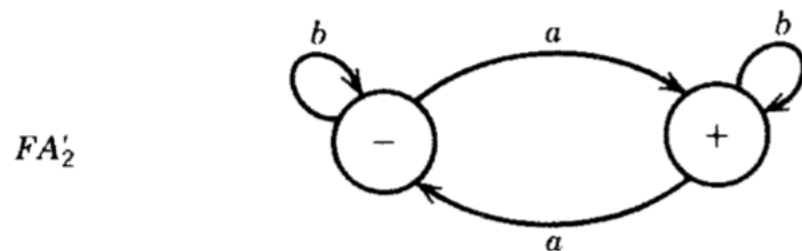
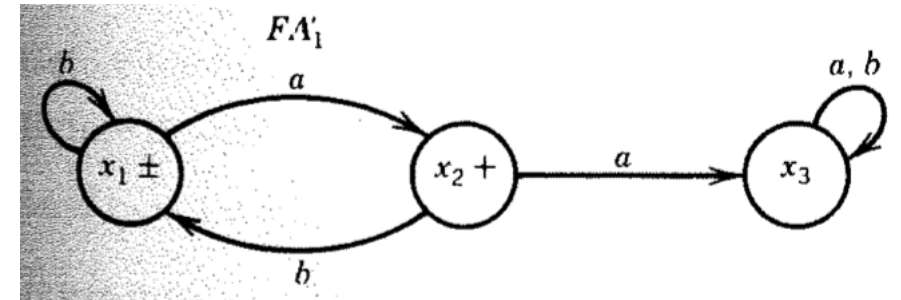
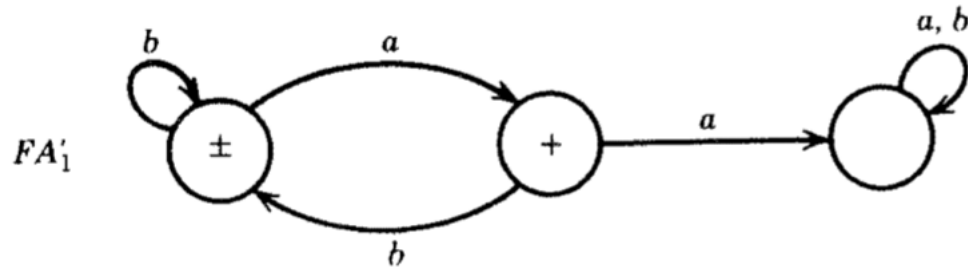
- We now have regular expressions for  $L_1'$  and  $L_2'$  and we can write  $L_1' + L_2'$

$$r_1' + r_2' = (b + ab)^*(\Lambda + a) + (b + ab^*a)^*ab^*$$

- We must now go in the other direction and make this RE into an FA so that we can take its complement to get the FA that defines  $L_1 \cap L_2$
- To build the FA that corresponds to a complicated RE is no picnic, but it can be done. However, we can find a better way

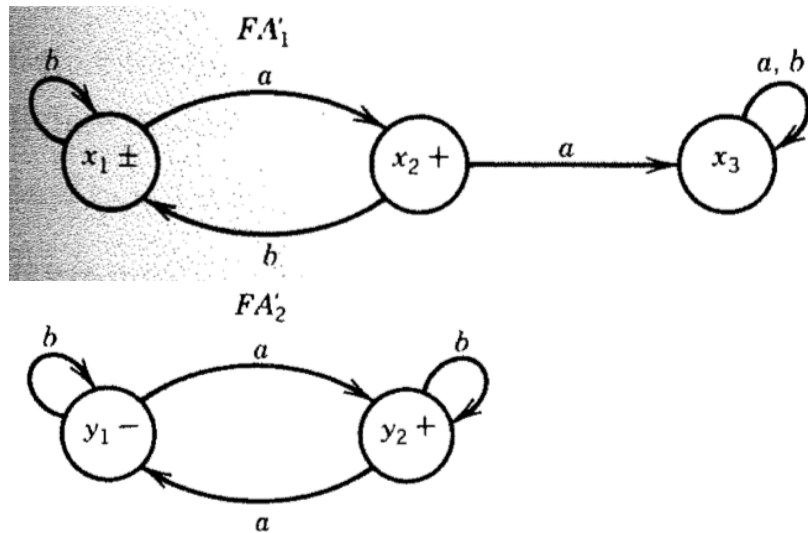
# Example

- An alternative approach is to make the machine for  $L_1' + L_2'$  directly from the machines for  $L_1'$  and  $L_2'$  without resorting to regular expressions



# Example

- The start states are  $x_1$  and  $y_1$  and the final states are  $x_1$ ,  $x_2$ , and  $y_2$ . The six possible combination states are



- The transition table for this machine is

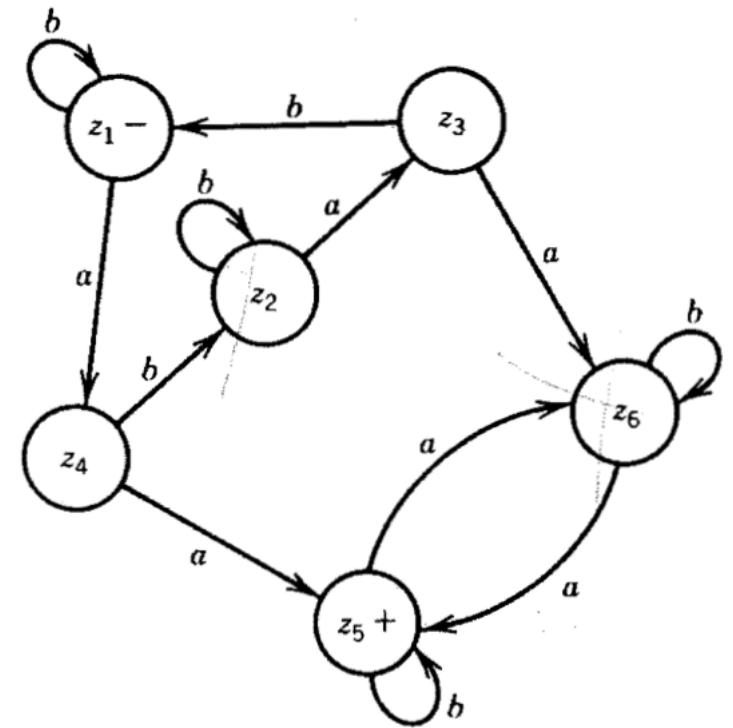
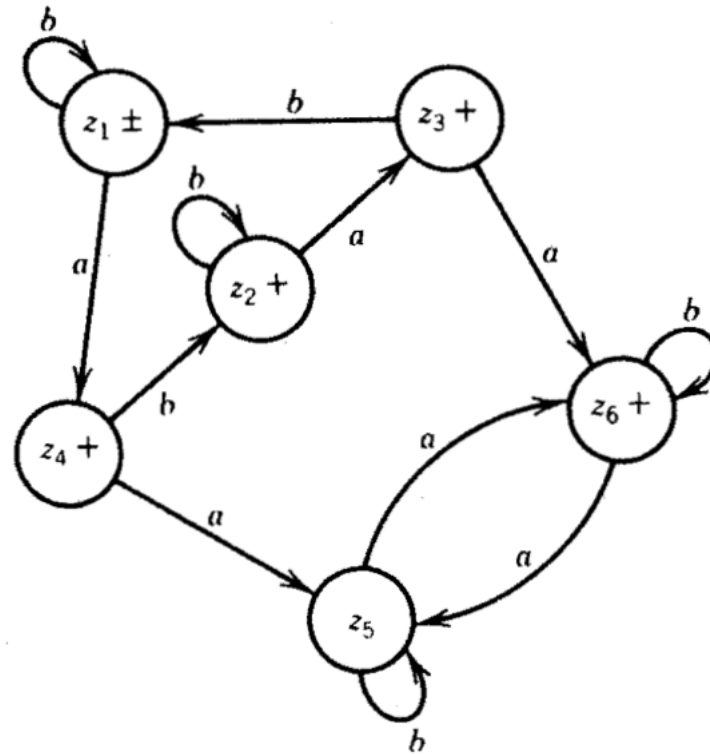
$z_1 = x_1$  or  $y_1$  start, final (words ending here are accepted in  $FA'_1$ )  
 $z_2 = x_1$  or  $y_2$  final (words ending here are accepted on  $FA'_1$  and  $FA'_2$ )  
 $z_3 = x_2$  or  $y_1$  final (words ending here are accepted on  $FA'_1$ )  
 $z_4 = x_2$  or  $y_2$  final (words ending here are accepted on  $FA'_1$  and  $FA'_2$ )  
 $z_5 = x_3$  or  $y_1$  not final on either machine  
 $z_6 = x_3$  or  $y_2$  final (words ending here are accepted on  $FA'_2$ )

	$a$	$b$
$\pm z_1$	$z_4$	$z_1$
$+ z_2$	$z_3$	$z_2$
$+ z_3$	$z_6$	$z_1$
$+ z_4$	$z_5$	$z_2$
$z_5$	$z_6$	$z_5$
$+ z_6$	$z_5$	$z_6$

# Example

- The union machine can be pictured as

	$a$	$b$
$\pm z_1$	$z_4$	$z_1$
$+ z_2$	$z_3$	$z_2$
$+ z_3$	$z_6$	$z_1$
$+ z_4$	$z_5$	$z_2$
$z_5$	$z_6$	$z_5$
$+ z_6$	$z_5$	$z_6$

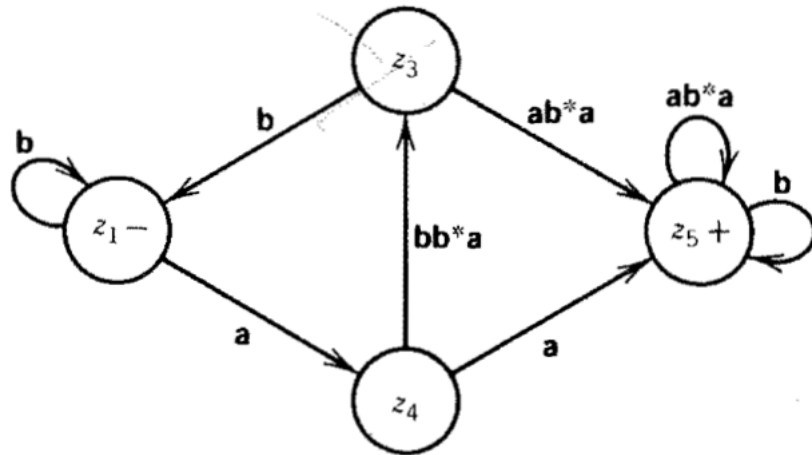


- This is an FA that accepts the language  $L_1' + L_2'$ . We produce an FA for the language  $L_1' + L_2'$  when we reverse the status of each state from final to nonfinal and vice versa

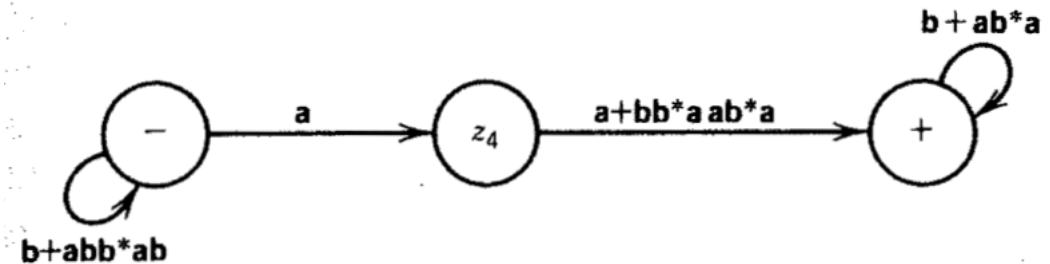


# Example

- Bypassing  $z_2$  and  $z_6$  gives



- Then bypassing  $z_3$  gives



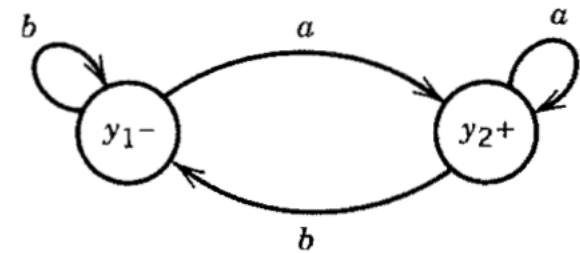
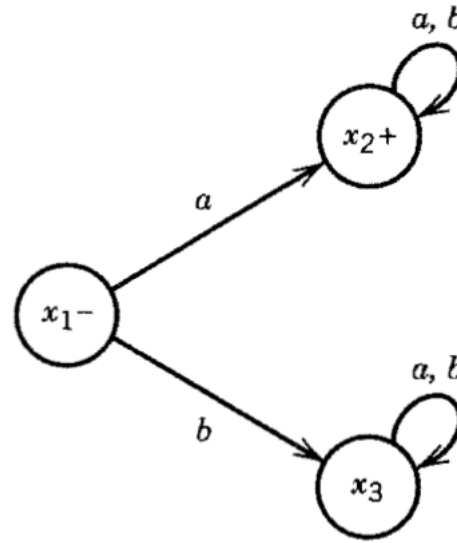
$$(b + abb^*ab)^*a(a + bb^*aab^*a)(b + ab^*a)^*$$

## Example

- $L_1$  = all words that begin with an a
- $L_2$  = all words that end with an a
- $r_1 = \mathbf{a(a + b)^*}$
- $r_2 = \mathbf{(a + b)^*a}$
- The intersection language will be
  - $L_1 \cap L_2$  = all words that begin and end with the letter a
  - Obviously regular, it can be defined by the RE:  $\mathbf{a(a + b)^*a + a}$

# Example

- $r_1 = \mathbf{a(a + b)^*}$
- $r_2 = (\mathbf{a + b})^*\mathbf{a}$



We now build the transition table of the machine that runs its input strings on FA<sub>1</sub> and FA<sub>2</sub> simultaneously

	State	Read <i>a</i>	Read <i>b</i>
$-z_1$	$x_1$ or $y_1$	$x_2$ or $y_2$	$x_3$ or $y_1$
$*z_2$	$x_2$ or $y_2$	$x_2$ or $y_2$	$x_2$ or $y_1$
$z_3$	$x_3$ or $y_1$	$x_3$ or $y_2$	$x_3$ or $y_1$
$z_4$	$x_2$ or $y_1$	$x_2$ or $y_2$	$x_2$ or $y_1$
$z_5$	$x_3$ or $y_2$	$x_3$ or $y_2$	$x_3$ or $y_1$

We construct the machine for  $L_1 \cap L_2 =$  all words in both  $L_1$  and  $L_2$  we put + only in the state that represents acceptance by both machines at once

