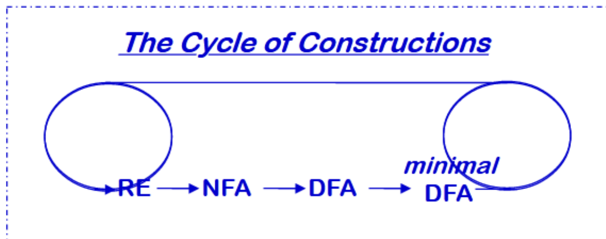
Kleen's Theorem (Lecture 5)

► **If a language can be expressed by**:

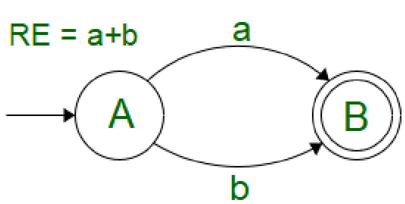
1. Regular expression (RE) or
2. Finite automata (FA) or
3. Transition graph (TG)

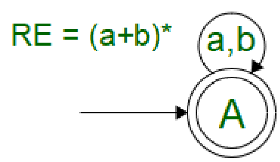
- **Then it can also be expressed by other two as well.**

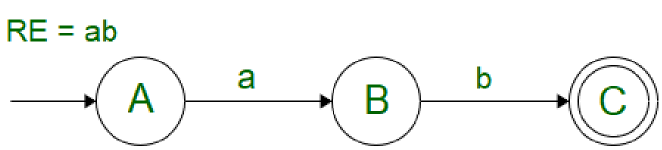


Theorem For every language L (over a finite alphabet Σ), the following statements are equivalent:

1. L is defined by some regular expression E.
2. L is accepted by some nondeterministic finite automaton N.
3. L is accepted by some deterministic finite automaton D.







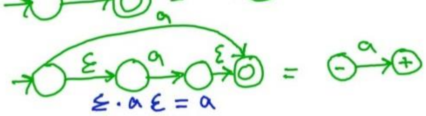
Thompson's Construction Algo:

$\Phi, \varepsilon, \forall a \in \Sigma, \Sigma = \{a, b\}$

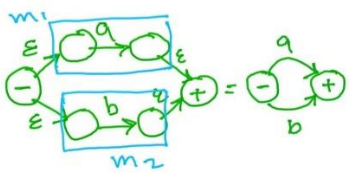
ϕ

ε

a

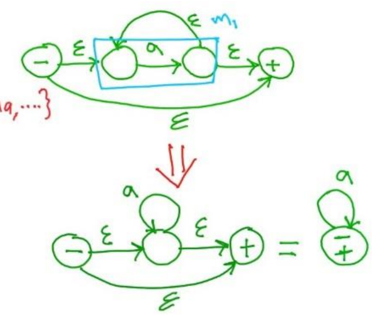


$a + b$

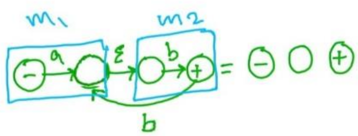


a^*

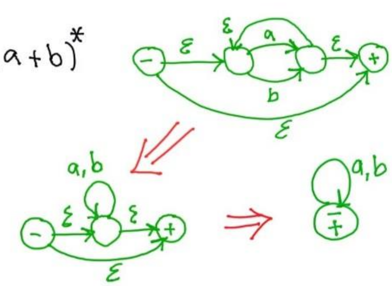
$= \{\varepsilon, a, aa, \ldots\}$



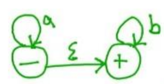
$a \cdot b$



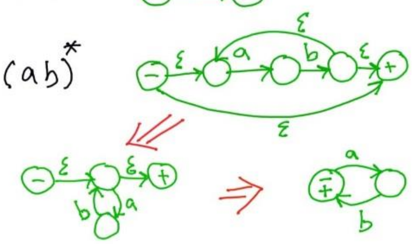
$(a+b)^*$



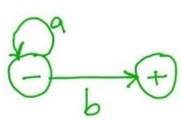
$a^* \cdot b^*$



$(ab)^*$



$a^* \cdot b$



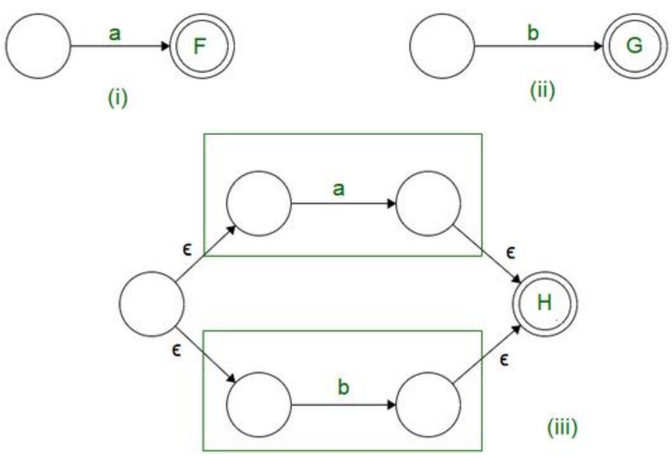
Let's, **r_1** and **r_2** be two regular expressions. Then,

1. **r1+ r2** is a regular expression too, whose corresponding language is **L(r1) U L(r2)**
2. **r1. r2** is a regular expression too, whose corresponding language is **L(r1).L(r2)**
3. **r1*** is a regular expression too, whose corresponding language is **L(r1)***

We can further use this definition in association with Null Transitions to give rise to a **FA** by the combination of two or more smaller Finite Automata (each corresponding to a Regular Expression).

Let **S** accept **L = {a}** and T accept **L = {b}**, then **R** can be represented as a combination of **S** and **T** using the provided operations as:

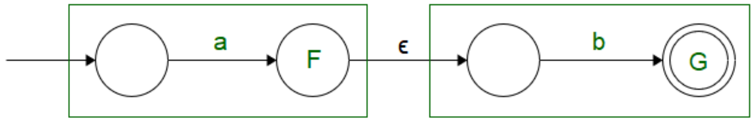
R = S + T



We observe that,

1. In case of union operation we can have a new start state, from which, null transition proceeds to the starting state of both the Finite State Machines.
2. The final states of both the Finite Automata's are converted to intermediate states. The final state is unified into one that can be traversed by null transitions.

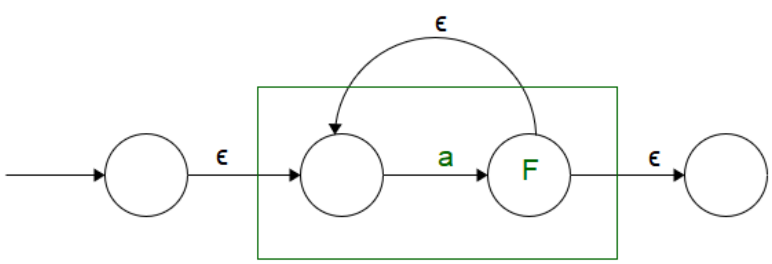
R = S.T



We observe that,

1. In case of concatenation operation, we can have the same starting state as that of **S**, the only change occurs in the end state of **S**, which is converted to an intermediate state followed by a Null Transition.
2. The Null transition is followed by the starting state of **T**; the final state of **T** is used as the end state of **R**.

R = S*

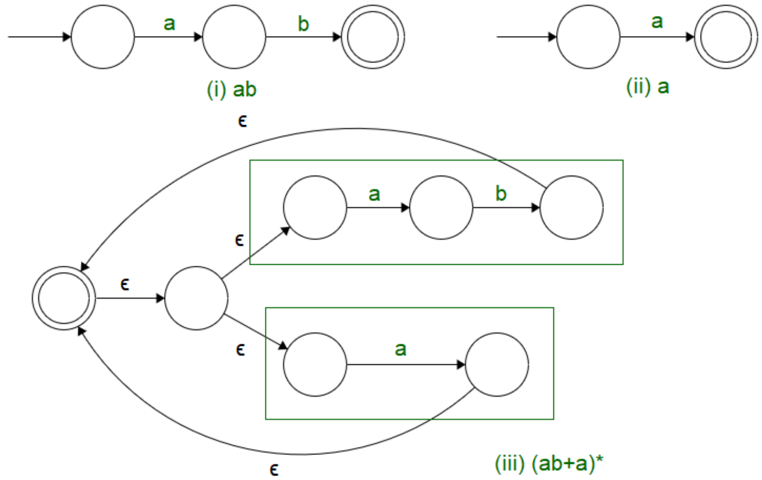


We observe that,

1. A new starting state is added, and S has been put as an intermediate state so that self-looping condition could be incorporated.
2. Starting and Ending states have been defined separately so that the self-looping condition is not disturbed.

Example

Make a Finite Automata for the expression **(ab+a)***



Proof

The three sections of our proof will be:

Kleene's Theorem Part1:

- Every language that can be defined (accepted) by a **FA** can also be defined (accepted) by a **TG**.

Kleene's Theorem Part2:

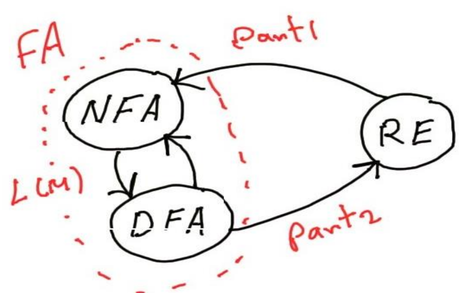
- Every language that can be defined by a **TG** can also be defined by a **RE**.

Kleene's Theorem Part3:

- Every language that can be defined (expressed) by a **RE** can also be defined by a **FA**. (We will break part 3 in to 4 rules).

Part # 3

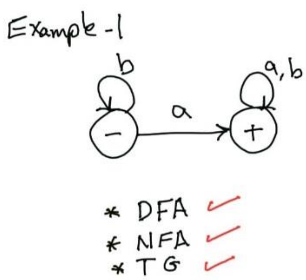
1. Rule #1
2. Rule #2 (Union of two FAs)
3. Rule #3 (Concatenation of two FAs)
4. Rule #4 (Kleen's Closure (Star) of a FAs)

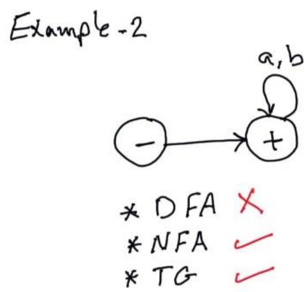


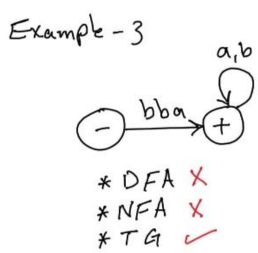
Kleene's Theorem Part I

Proof

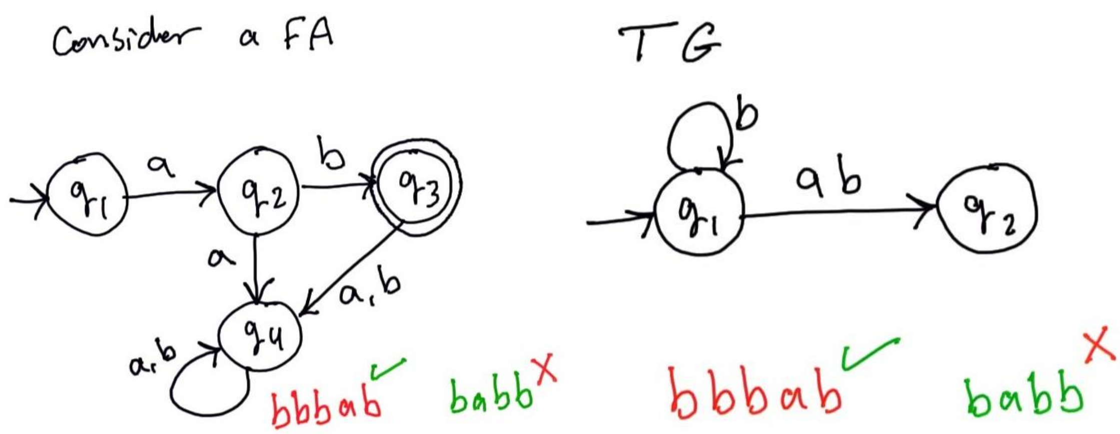
- Every FA can be considered to be a TG as well.
- Any language that has been defined by a FA has already been defined by a TG.
    - So, **there is nothing to prove**.







Example



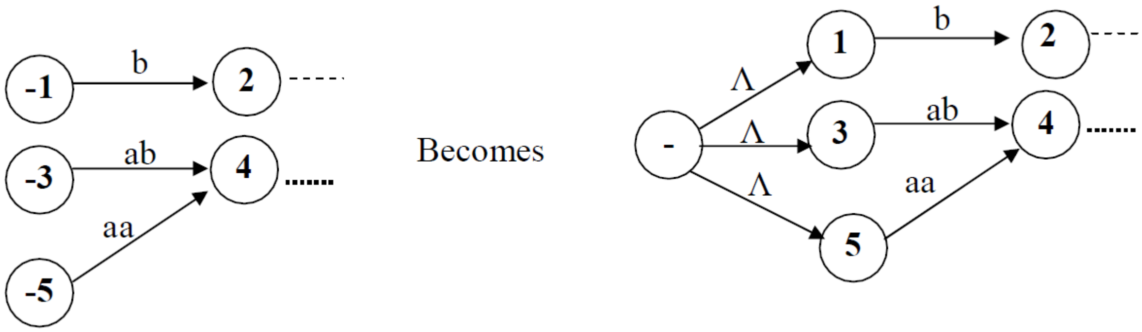
Kleene's Theorem Part II

Proof

The proof of this part will be by constructive algorithm. This means that we present a procedure that starts out with a TG and ends up with a RE that defines the same language.

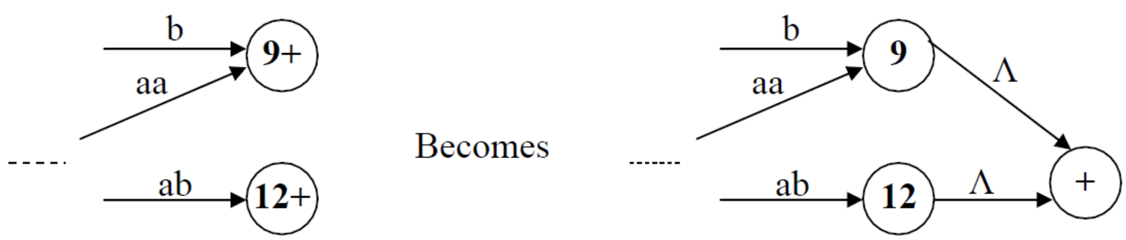
- **Step 1:** (Let the start states be only one)
  - If a TG has more than one start states, then introduce a new start state connecting the new state to the old start states by the transitions labeled by and Λ (ε) make the old start states the non-start states. This step can be shown by the following example

Example



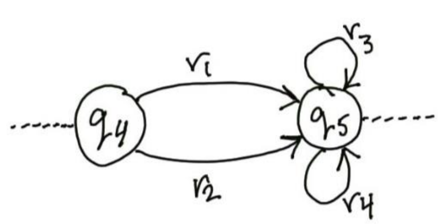
- **Step 2:** (Let the final states be only one)
- If a TG has more than one final states, then introduce a new final state, connecting the old final states to the new final state by the transitions labeled by Λ.
- This step can be shown by the previous example of TG, where the step 1 has already been processed.

Example

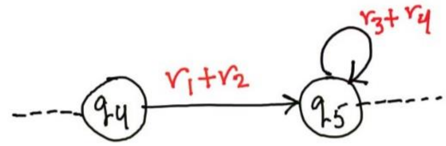


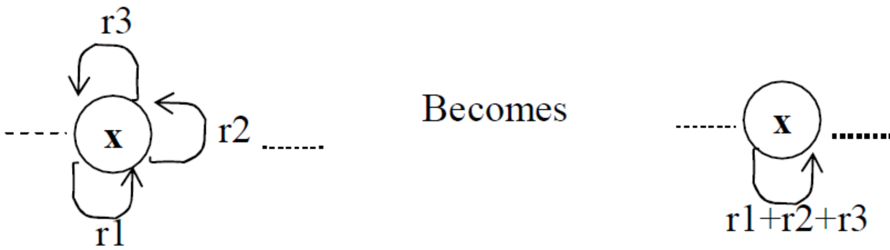
- **Step 3:** (Reduce the number of edges)
- If a state has two (more than one) incoming transition edges labeled by the corresponding REs, from the same state (including the possibility of loops at a state), then replace all these transition edges with a single transition edge labeled by the sum of corresponding REs.
- This step can be shown by a part of TG in the following example.

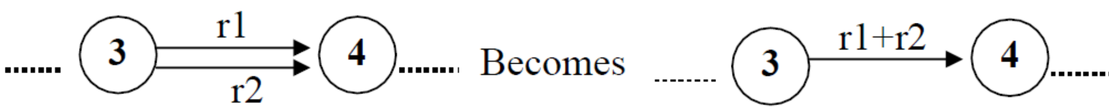
Example

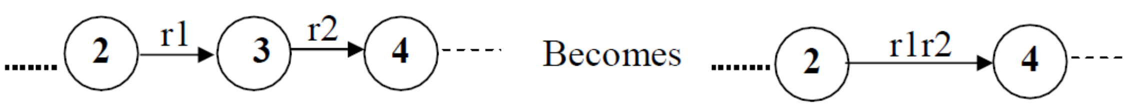


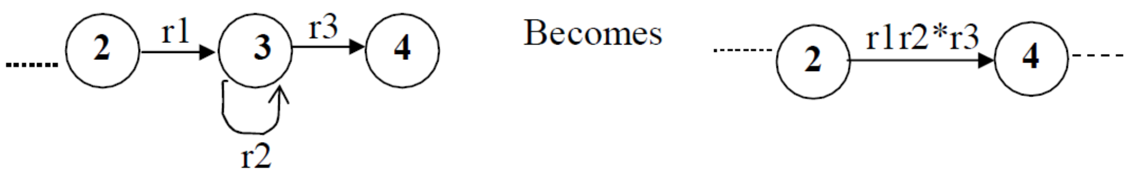
The above TG can be reduced to

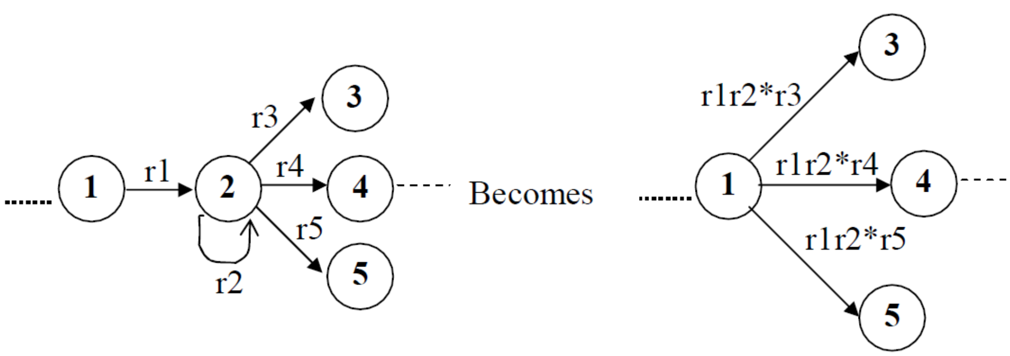


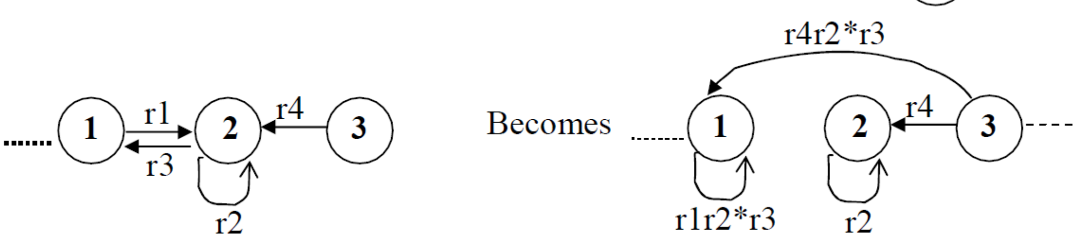




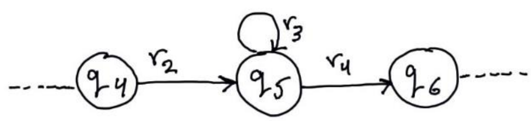




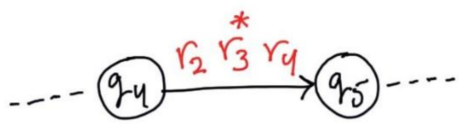


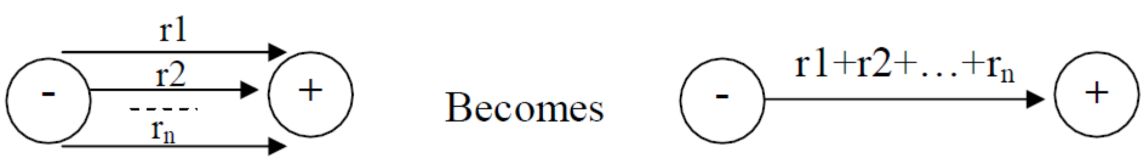


- ► Repeat the last step repeatedly until we eliminate all the states from TG except the unique start state and the unique final state.
- **Step 4:** (Eliminate states in each time)
  - – If three states in a TG, are connected in sequence then eliminate the middle state and connect the first state with the third by a single transition (include the possibility of circuit as well) labeled by the RE which is the concatenation of corresponding two REs in the existing sequence.
- – This step can be shown by a part of TG in the following example.



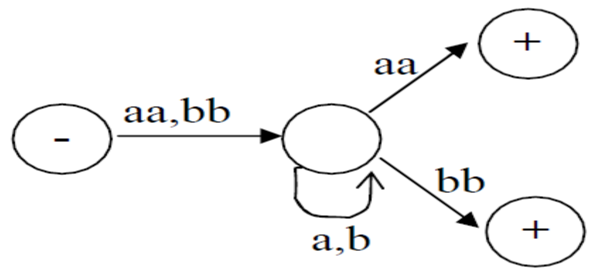
To eliminate state 5 the above can be reduced to

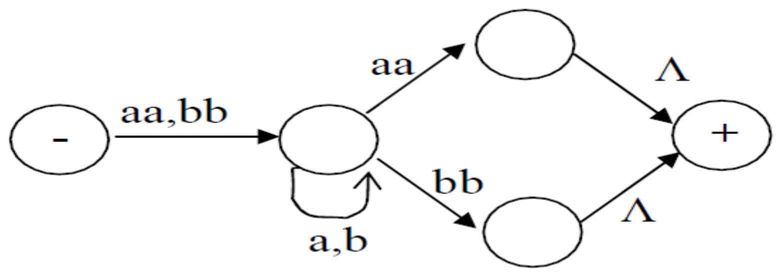


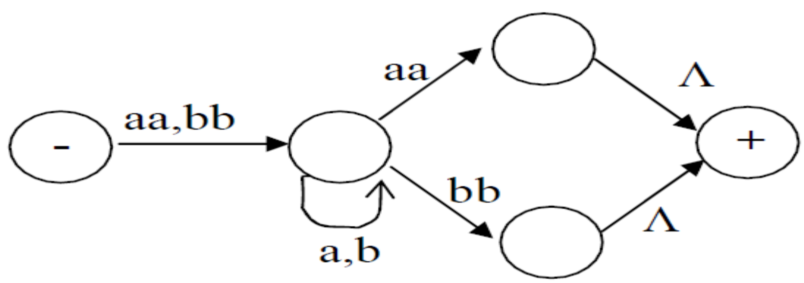


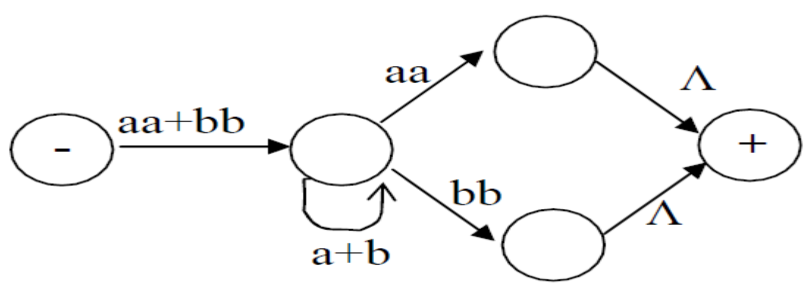
Example

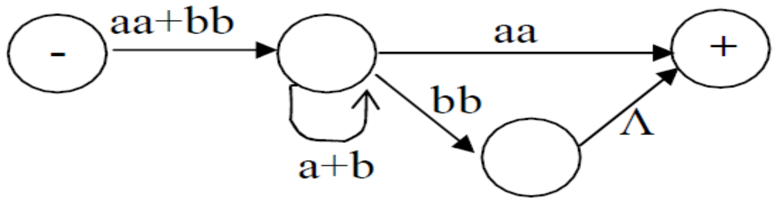
Find the RE that defines the same language accepted by the following TG using Kleenes theorem.

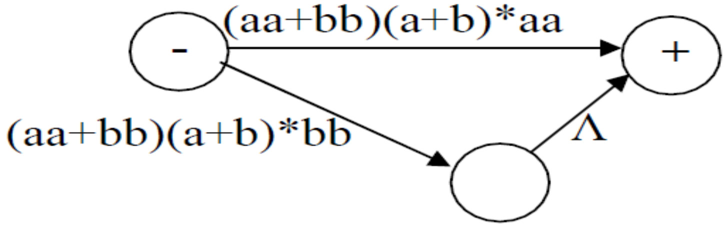

aa
-
aa,bb
+
bb
+
a,b

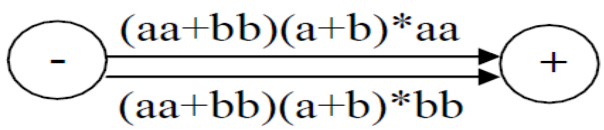

aa
Λ
-
aa,bb
+
bb
Λ
a,b

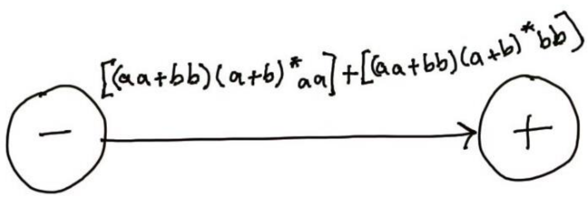

aa
Λ
-
aa,bb
+
bb
Λ
a,b


aa
Λ
-
aa+bb
+
bb
Λ
a+b





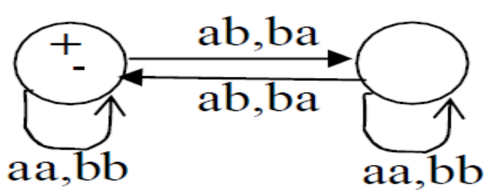




RE= (aa+bb) (a+b)*(aa+bb)

H.W

Find the RE that defines the same language accepted by the following TG using Kleenes theorem.

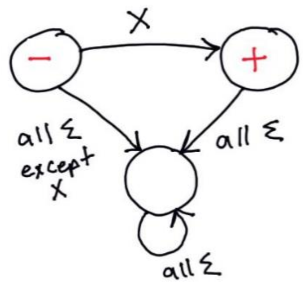


Kleene's Theorem Part III

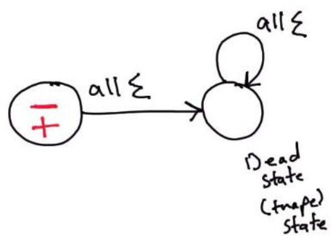
The proof of part3

Rule1: there is an **FA** that accepts any particular letter of the alphabet. There is an **FA** that accepts only the word **Λ**.

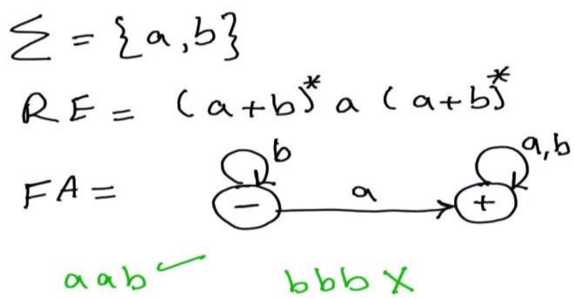
If **RE** is **x** then the **FA** will be:



If **RE** is **Λ** then the **FA** will be:



Example



Rule2: (**Union of two FA's**)

If there is an **FA** called **FA1**, that accepts the language defined by the regular expression **r1** and there is an **FA** called **FA2**, that accepts the language defined by the regular expression **r2**, then there is an **FA** called **FA3** that accepts the language defined by the regular expression **(r1+r2)**.

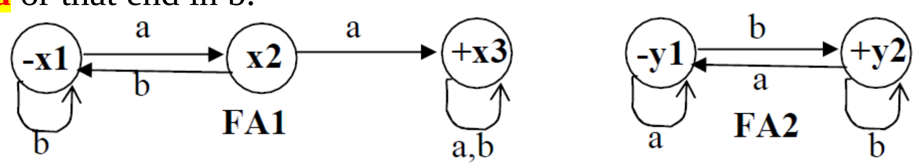
We can describe the algorithm for forming FA3 as follows:

- Starting with two machines **FA1**, with states **x1, x2, x3**,…. And **FA2** with states **y1,y2,y3**,…,build a new machine **FA3** with states **z1,z2,z3**,… where each **z** is of the form "$x_{something}$ or $y_{something}$". If either the **x** part or the **y** part is a final state, then the corresponding **z** is a final state.
- To go from one **z** to another by reading a letter from the input string, we see what happens to the **x** part and to the **y** part and go to the new **z** accordingly. We could write this as a formula:

$$z_{new} \text{ after letter } p = [x_{new} \text{ after letter } p] \text{ or } [y_{new} \text{ after letter } p]$$

Example

We have **FA1** accepts all words with a double a in them, and **FA2** accepts all words ending in **b**. we need to build **FA3** that accepts all words that have double **a** or that end in b.



	a	b
-x1	x2	x1
x2	x3	x1
+x3	x3	x3

The transition table for FA1

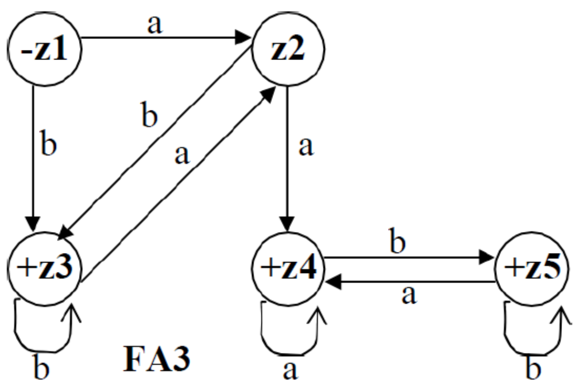
	a	b
-y1	y1	y2
+y2	y1	y2

The transition table for FA2

z1=x1 or y1
z2= x2 or y1
z3=x1 or y2
z4=x3 or y1
z5=x3 or y2

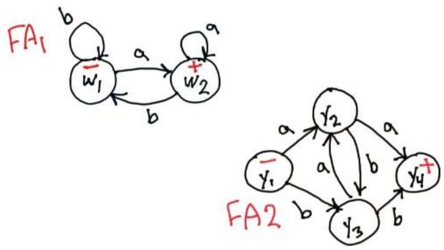
	a	b
-z1	z2	z3
z2	z4	z3
+z3	z2	z3
+z4	z4	z5
+z5	z4	z5

The transition table for FA3



FA3

Example

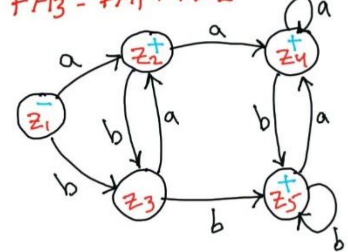


FA1	a	b
$-w_1$	w_2^+	w_1
$+w_2$	w_2^+	w_1

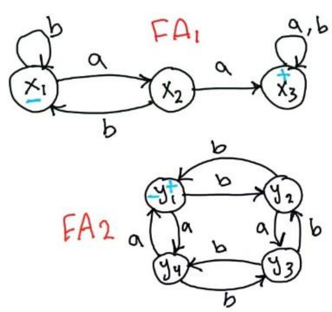
FA2	a	b
$-y_1$	y_2	y_3
y_2	y_4^+	y_3
y_3	y_2	y_4^+
$+y_4$	y_4^+	y_4^+

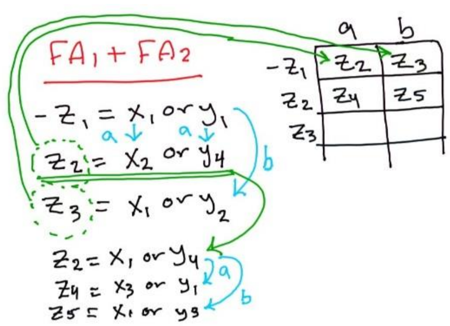
FA1+FA2	a	b
$-z_1 = w_1 + y_1$	$w_2^+ + y_2 = z_2^+$	$w_1 + y_3 = z_3$
$+z_2 = w_2 + y_2$	$w_2^+ + y_4^+ = z_4^+$	$w_1 + y_3 = z_3$
$z_3 = w_1 + y_3$	$w_2^+ + y_2 = z_2^+$	$w_1 + y_4^+ = z_5^+$
$+z_4 = w_2 + y_4$	$w_2^+ + y_4^+ = z_4^+$	$w_1 + y_4^+ = z_5^+$
$+z_5 = w_1 + y_4$	$w_2^+ + y_4^+ = z_4^+$	$w_1 + y_4^+ = z_5^+$

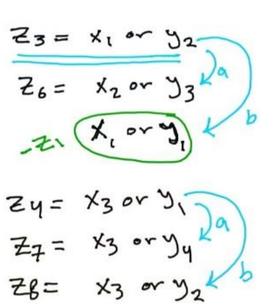
$FA_3 = FA_1 + FA_2$



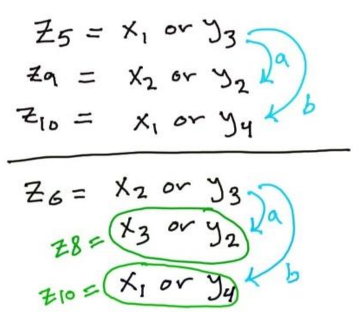
Example







	a	b
$-z_1$	z_2	z_3
z_2	z_4	z_5
z_3	z_6	z_1
z_4	z_7	z_8



$z_7 = x_3$ or y_4
- a: $z_4 = (x_3$ or $y_1)$
- b: $z_{11} = x_3$ or y_3

$z_8 = x_3$ or y_2
- a: $z_{11} = (x_3$ or $y_3)$
- b: $z_4 = (x_3$ or $y_1)$

* $z_{11} = x_3$ or y_3
- a: $z_8 = (x_3$ or $y_2)$
- b: $z_7 = (x_3$ or $y_4)$

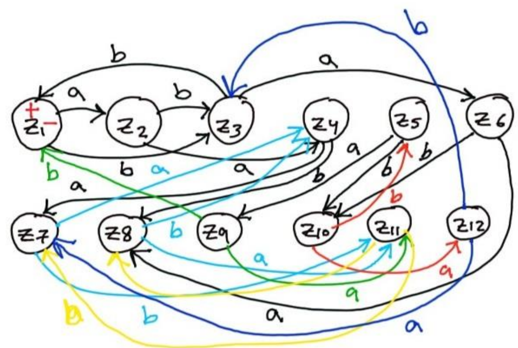
* $z_{12} = x_2$ or y_1
- $z_7 = (x_3$ or $y_4)$
- $z_3 = (x_1$ or $y_2)$

* $z_9 = x_2$ or y_2
- a: $z_{11} = (x_3$ or $y_3)$
- b: $z_1 = (x_1$ or $y_1)$

* $z_{10} = x_1$ or y_4
- a: $z_{12} = x_2$ or y_1
- b: $z_5 = (x_1$ or $y_3)$

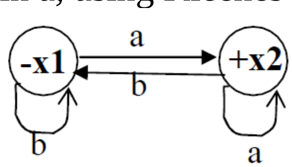
	a	b
$-z_1$	z_2	z_3
z_2	z_4	z_5
z_3	z_6	z_1
z_4	z_7	z_8
z_5	z_9	z_{10}
z_6	z_8	z_{10}
z_7	z_4	z_{11}
z_8	z_{11}	z_4
z_9	z_{11}	z_1
z_{10}	z_{12}	z_5
z_{11}	z_8	z_7
z_{12}	z_7	z_3

	a	b
$\pm z_1$	z_2	z_3
z_2	z_4	z_5
z_3	z_6	z_1
$+z_4$	z_7	z_8
z_5	z_9	z_{10}
z_6	z_8	z_{10}
$+z_7$	z_4	z_{11}
$+z_8$	z_{11}	z_4
z_9	z_{11}	z_1
z_{10}	z_{12}	z_5
$+z_{11}$	z_8	z_7
$+z_{12}$	z_7	z_3

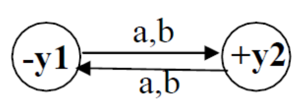


H.W.

Let FA1 accepts all words ending in a, and let FA2 accepts all words with an odd number of letters (odd length). Build FA3 that accepts all words with odd length or end in a, using Kleenes theorem.



FA1



FA2

Rule3: (**Concatenation of two FA's**)

If there is an **FA1** that accepts the language defined by the regular expression **r1** and an **FA2** that accepts the language defined by the regular expression **r2**, then there is an **FA3** that accepts the language defined by the concatenation **r1 r2**.

We can describe the algorithm for forming FA3 as follows:

We make a **z** state for each none final **x** state in **FA1**. And for each final state in **FA1** we establish a **z** state that expresses the option that we are continuing on **FA1** or are beginning on **FA2**. From there we establish **z** states for all situations of the form:

Are in $\mathbf{x}_{\text{something}}$ continuing on **FA1**

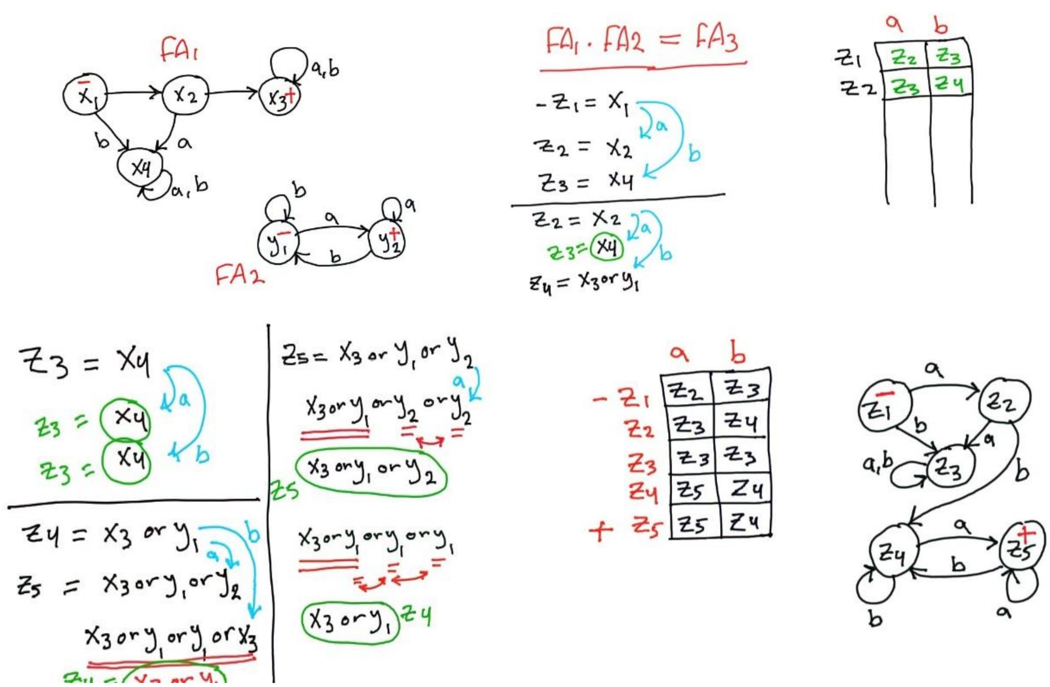
Or

Have just started y1 about to continue on **FA2**

Or

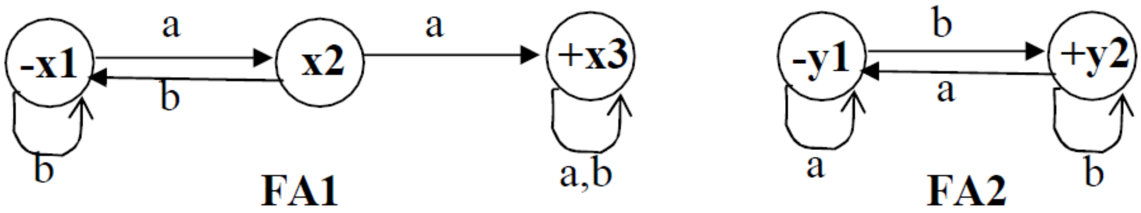
Are in $\mathbf{y}_{\text{something}}$ continuing on **FA2**

Example



Example

We have **FA1** accepts all words with a double **a** in them, and **FA2** accepts all words ending in **b**. we need to build **FA3** that accepts all words that have double **a** and end in **b**.



	a	b
-x1	x2	x1
x2	x3	x1
+x3	x3	x3

The transition table for FA1

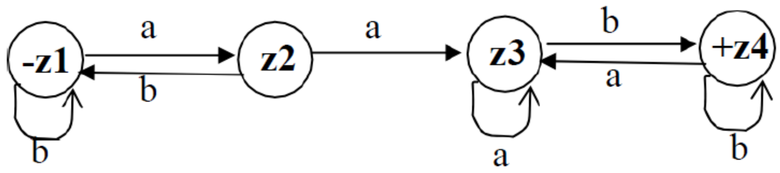
	a	b
-y1	y1	y2
+y2	y1	y2

The transition table for FA2

z1=x1
z2= x2
z3=x3 or y1
z4=x3 or y2 or y1

	a	b
-z1	z2	z1
z2	z3	z1
z3	z3	z4
+z4	z3	z4

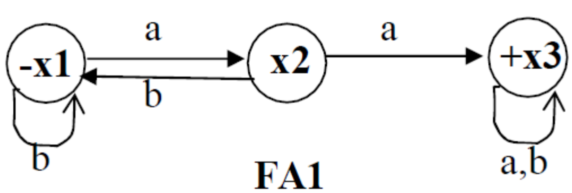
The transition table for FA3



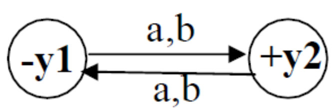
FA3

H.W.

Let FA1 accepts all words with a double a in them, and let FA2 accepts all words with an odd number of letters (odd length). Build FA3 that accepts all words with odd length and have double a in them using Kleen's theorem.



FA1



FA2

Rule4: (**Kleene's closure of FA's**)

If **r** is a regular expression and **FA1** accepts exactly the language defined by **r**, then there is an **FA2** that will accept exactly the language defined by **r***.

We can describe the algorithm for forming FA2 as follows:

- Each **z** state corresponds to some collection of **x** states.
- We must remember each time we reach a final state it is possible that we have to start over again at **x1**.
- Remember that the **start state** must be the **final state** also.

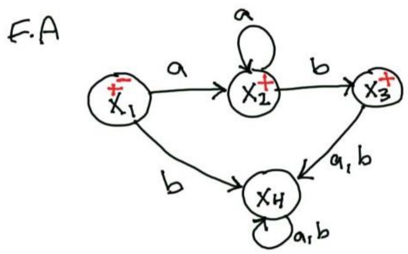
Example

If we have **FA1** that accepts, the language defined by the regular expression:

r=a*+aa*b

We want to build **FA2** that accept the language defined by r*.

Part3 (Rule #04)

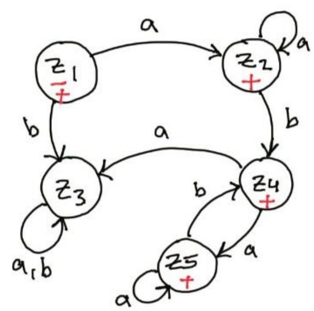


old state	a	b
∓ $X_1 = Z_1$	$X_2X_1 = Z_2$	$X_4 = Z_3$
+ $X_2X_1 = Z_2$	$X_2X_1 = Z_2$	$X_3X_1X_4 = Z_4$
$X_4 = Z_3$	$X_4 = Z_3$	$X_4 = Z_3$
+ $X_3X_1X_4 = Z_4$	$X_4X_2X_1 = Z_5$	$X_4 = Z_3$
+ $X_4X_2X_1 = Z_5$	$X_4X_2X_1 = Z_5$	$X_3X_1X_4 = Z_4$

- With final state, always write final state.
- Repeat initial state a second time with new "Z" state.
- Initial state will always be final state.

The transition table and diagram of **FA2** will be:

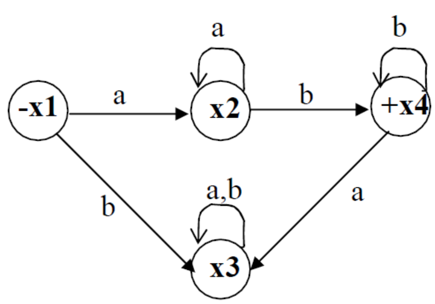
	a	b
∓ Z_1	Z_2	Z_3
+ Z_2	Z_2	Z_4
Z_3	Z_3	Z_3
+ Z_4	Z_5	Z_3
+ Z_5	Z_5	Z_4



H.W.

Let FA1 accept the language defined by r1, find FA2 that accept the language defined by r1* using Kleene's theorem.

r1= aa*bb*



FA1