Kleene's Theorem

Any language that can be defined by:

1. Regular expression

or

2. Finite automata

or

3. Transition graph Can be defined by all three methods.

Proof

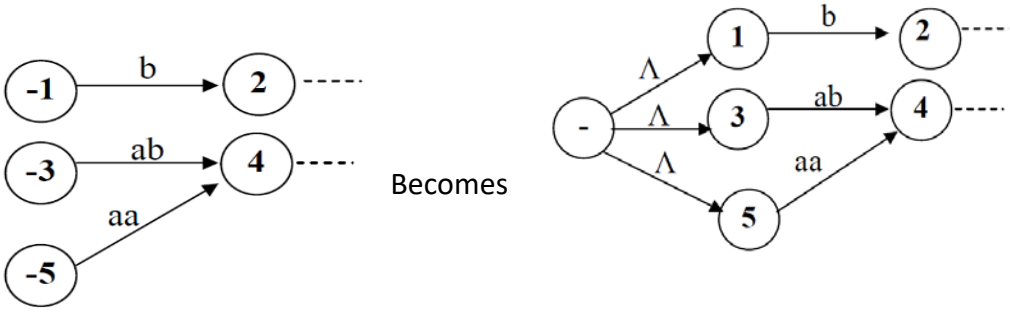
The three sections of our proof will be:
Part1: every language that can be defined by a FA can also be defined by a TG.
Part2: every language that can be defined by a TG can also be defined by a RE.
Part3: every language that can be defined by a RE can also be defined by a FA.

The proof of part1

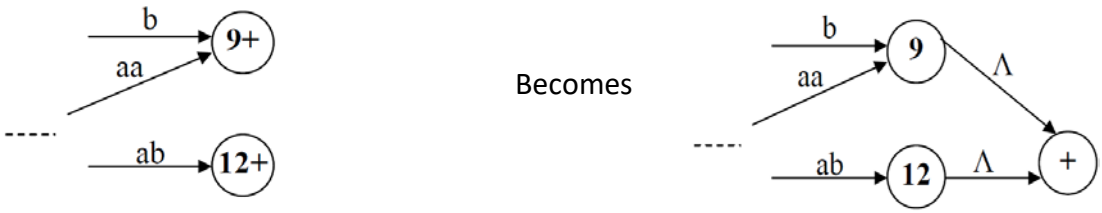
Every FA is itself a TG. Therefore, any language that has been defined by a FA has already been defined by a TG.

The proof of part2

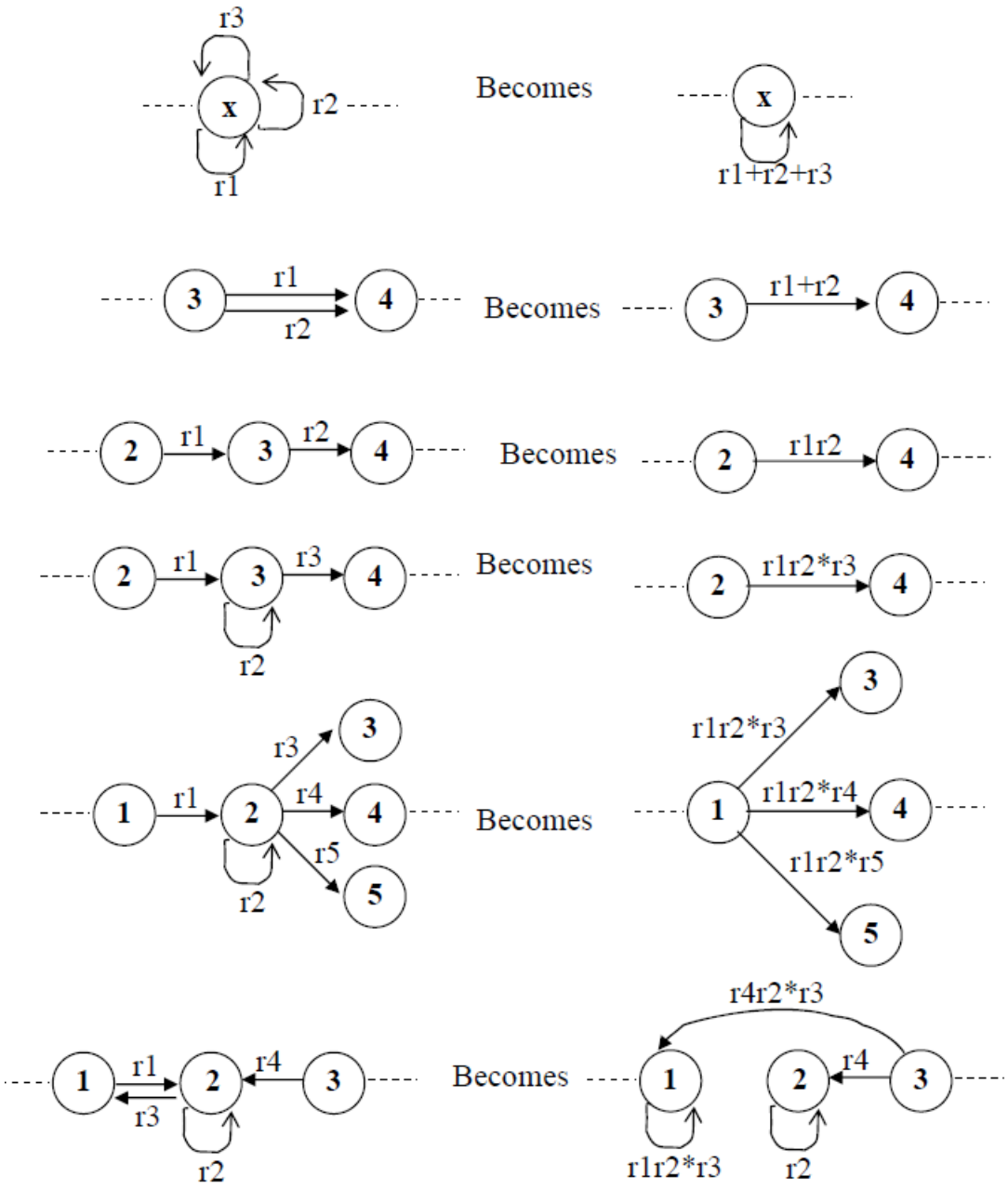
The proof of this part will be by constructive algorithm. This means that we present a procedure that starts out with a TG and ends up with a RE that defines the same language.
Let the start states be only one.



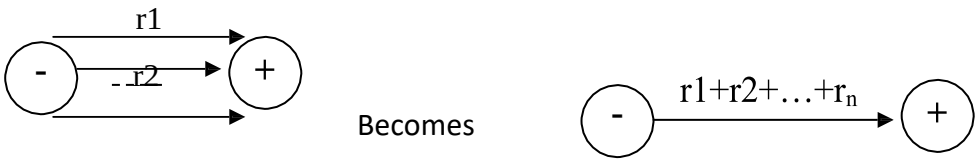
Let the final states be only one.



Allow the edges to be labeled with regular expressions (reduce the number of edges or states in each time).

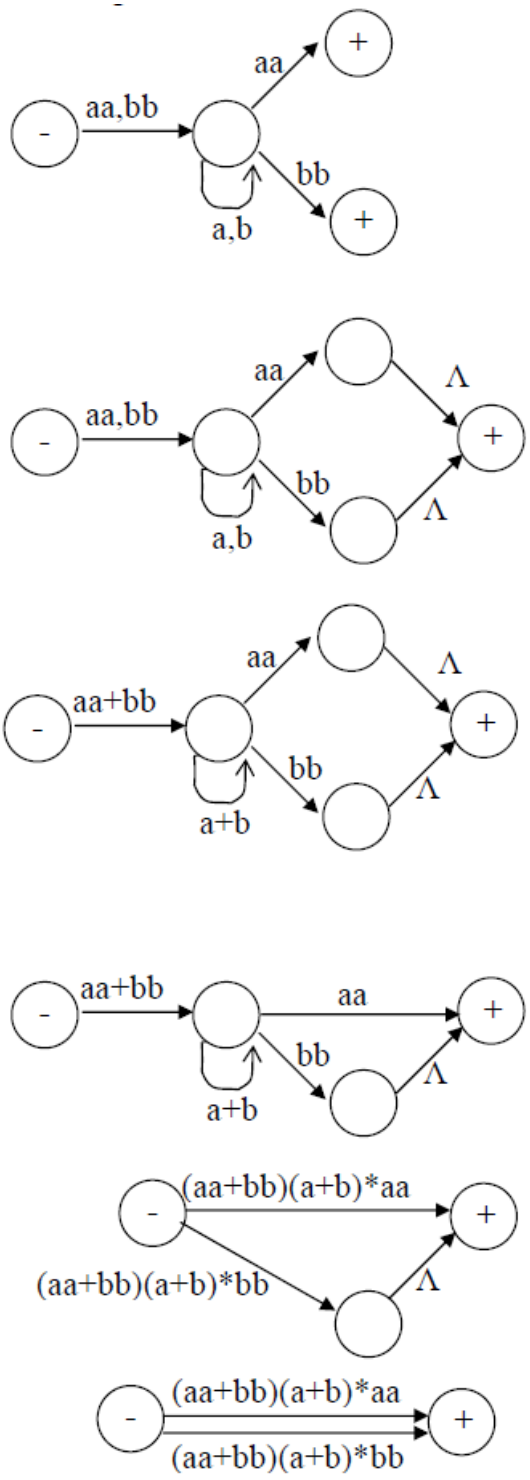


Repeat the last step again and again until we eliminate all the states from TG except the unique start state and the unique final state.



Example:

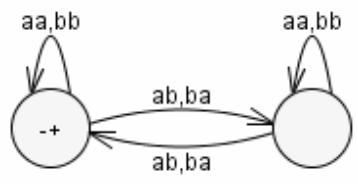
Find the RE that defines the same language accepted by the following TG using Kleen's theorem.



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

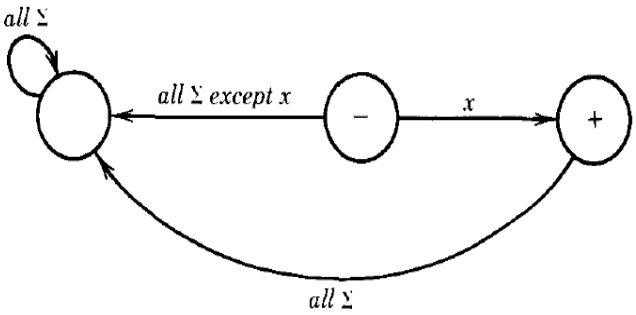
Home Work

Find the RE that defines the same language accepted by the following TG using Kleen's theorem.

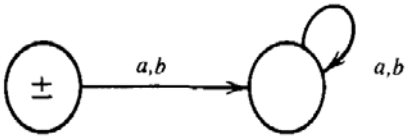


The proof of part3

Rule1: there is an FA that accepts any particular letter of the alphabet. There is an FA that accepts only the word Λ. For example, if x is in ∑, then the FA accepts only the word x



One FA accepts only Λ will be:

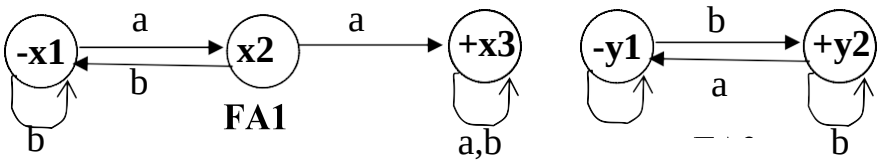


Rule2: if there is an FA called FA1, that accepts the language defined by the regular expression r1 and there is an FA called FA2, that accepts the language defined by the regular expression r2, then there is an FA called FA3 that accepts the language defined by the regular expression (r1+r2). We can 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 "xsomething or ysomething". 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:

znew after letter p=[xnew after letter p]or[ynew 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.



The transition table for FA1

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

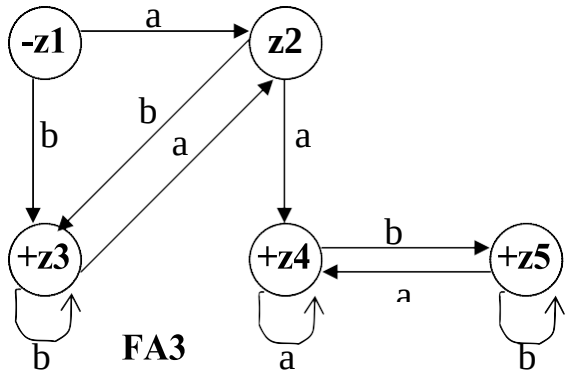
The transition table for FA2

	a	b
-y1	y1	y2
+y2	y1	y2

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

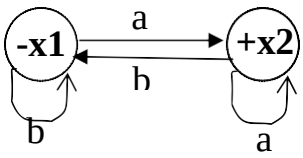
The transition table for FA3

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

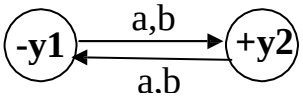


Home Work:

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 Kleen's theorem.



FA1



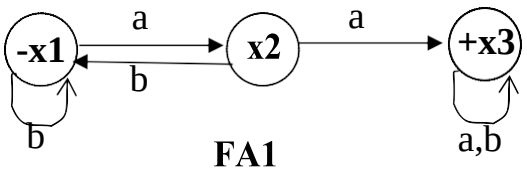
FA2

Rule3: 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 r1r2. We can describe the algorithm for forming FA3 as follows:

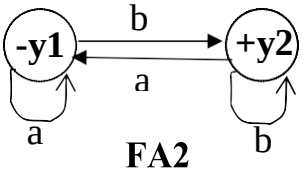
We make a z state for each none final x state in FA1. And for each final state in FA1 we establish a z state that expresses the option that we are continuing on FA1 or are beginning on FA2. From there we establish z states for all situations of the form:

Are in x something continuing on FA1 or Have just started y1 about to continue on FA2 or Are in y something continuing on FA2

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**.



FA1



FA2

The transition table for FA1

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

The transition table for FA2

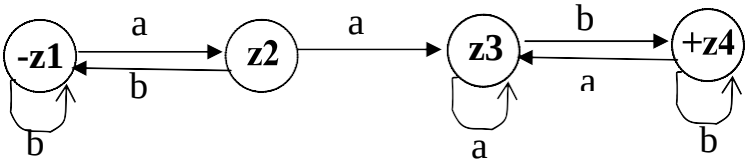
	a	b
-y1	y1	y2
+y2	y1	y2

z1=x1 z2= x2

z3=x3 or y1 z4=x3 or y2 or y1

The transition table for FA3

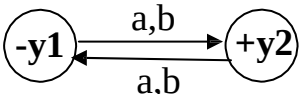
	a	b
-z1	z2	z1
z2	z3	z1
z3	z3	z4
+z4	z3	z4



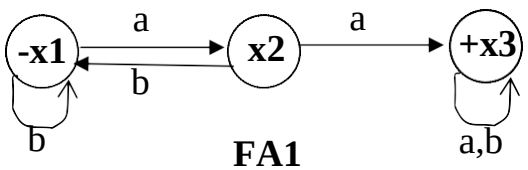
FA3

Home Work:

Let FA1 accepts all words with a double a in them, and let FA2accepts 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.



FA2



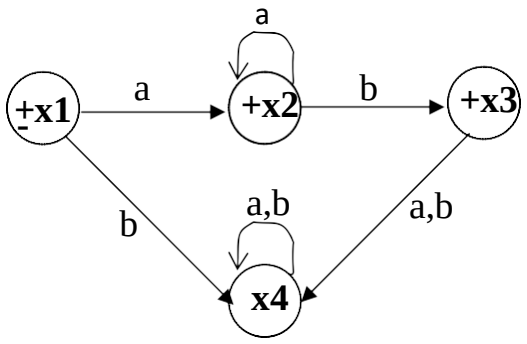
FA1

Rule4: 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

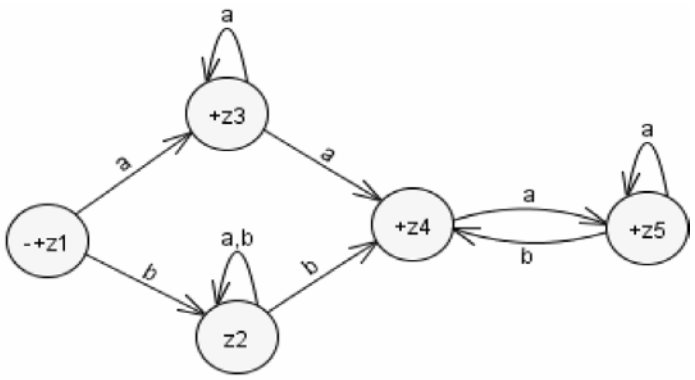
If we have FA1 that accepts the language defined by the regular expression: r=a*+aa*b
We want to build FA2 that accept the language defined by r*. Note: We will try to connect the final state with start state.



The transition table for FA1

	a	b
-,+ x1	x2	x4
+x2	x2	x3
+x3	x4	x4
x4	x4	x4

z1=x1
z2=x4
z3=x2 or x1
z4=x3 or x4 or x1 z5=x4 or x2 or x1



The transition table for FA2

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

University of Basrah

Faculty of Education for Pure Sciences

Computer Department

Home Work

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