

FINITE AUTOMATA

Introduction

Cohen (2001) notes that several children's games fit the following description: Pieces are set up on a playing board; dice are thrown (or a wheel is spun) and a number is generated at random. Based on the generated number, the pieces on the board are rearranged by the rules of the game. Then, another child throws or spins and rearranges the pieces again. There is no skill or choice involved - the entire game is based on the values of the random numbers. Consider all possible positions of the pieces on the board and call them **states**. We begin with the **initial state** of the starting positions of the pieces on the board. The game then changes from one state to another based on the value of the random number. For each possible number, there is one and only one resulting state given the input of the number, and the prior state. This continues until one player wins and the game is over. This is called a **final state**.

Now consider a very simple computer with an input device, a processor, some memory and an output device. We want to calculate $3 + 4$, so we write a simple list of instructions and feed them into the machine one at a time (e.g., STORE 3 TO X; STORE 4 TO Y; LOAD X; ADD Y; WRITE TO OUTPUT). Each instruction is executed as it is read. If all goes well, the machine outputs '7' and terminates execution. This process is similar to the board game. The state of the machine changes after each instruction is executed, and each state is completely determined by the prior state and the input instruction (thus this machine is defined as **deterministic**). No choice or skill is involved; no knowledge of the state of the machine for instructions ago is needed. The machine simply starts at an initial state, changes from state to state based on the instruction and the prior state, and reaches the final state of writing '7'.

Definition of an FA

A Finite Automaton is yet another method for defining languages. This model is said to be finite because the number of possible states and number of letters in the alphabet are both finite. It is called an automaton because the change of states is totally governed by the input. The determination of the next state is automatic (involuntary and mechanical) and not wilful, just as the motion of the hands of a clock is automatic while the motion of a human is presumably the result of desire and thought (Cohen 2001).

Finite Automata are good models for computers with an extremely limited amount of memory, like an automatic door, switches, elevator or digital watches.

Finite automata and their probabilistic counterpart "Markov chains" are useful tools when we are attempting to recognize patterns in data. These devices are used in speech processing and in optical character recognition.

A **finite automaton** has:

- 1) A finite set of states, one of which is designated the initial state or **start state**, and some (maybe none) of which are designated as **final states**.
- 2) An alphabet Σ of possible input symbols.
- 3) A finite set of **transitions** that tell for *each* state and for *each* symbol of the input alphabet, which state to go to next.

A finite automaton can be deterministic or non-deterministic.

Informal definition of a deterministic finite state automaton

A deterministic finite automaton (DFA) is an abstract device that

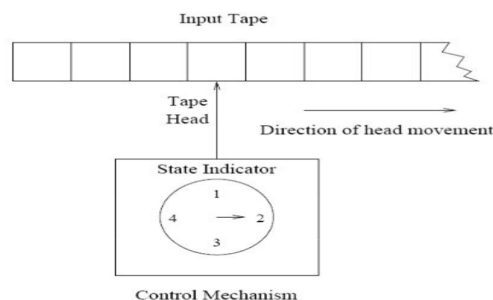
- is always in one of a **finite** number of states
 - has exactly one initial state
 - has at least one accept state
 - is attached to an input stream from which symbols are read sequentially one at a time
 - is capable of switching to a new state (or staying in the current state) which is uniquely **determined** by the current state and the symbol read from the input.
- **State transition** is the process of switching from one state to another.
 - **Control mechanism** is the mechanism which decides, on the basis of the current state and the next input symbol, which new state to move to.

How does an FA work?

The foregoing definitions do not describe how an FA works but instead what it is. The FA works by being presented with an input string of letters that it reads letter by letter starting at the leftmost letter. Beginning at the start state, the letters determine the sequence of states. The sequence ends when the last input letter has been read.

A DFA may be visualised by imagining:

- its input stream to be a tape
 - divided into cells each holding one symbol
 - with a fixed end at the left but extending indefinitely to the right
- its control mechanism to be a clock face with a single hand indicating the current state
- a read head positioned over the tape which reads a symbol, passes it to the control mechanism and is then repositioned over the next cell.



- A deterministic finite automaton **accepts** its input if starting in its initial state with the tape head over the first input symbol it moves to an accept state after reading the last input symbol.

- If the machine reaches the end of its input before reading any symbols then its input is called the **empty string** and is denoted Λ

Note that since Λ contains no symbols, a DFA accepts Λ if and only if its initial state is also an accept state.

Note that the requirement that the next state of a DFA be uniquely determined by the current state and the current input means that in the corresponding transition diagram no two arcs leaving a state can be labelled with the same symbol (otherwise a choice would be necessary and the machine would be **nondeterministic**).

Formal definition of a deterministic finite state automaton

A deterministic finite state automaton is a quintuple $(S, \Sigma, \delta, i, F)$

Where: S - is finite set of states eg $\{s_0, s_1, s_2\}$

Σ - is an alphabet (non-empty finite set of symbols) eg $\{x_0, x_1, x_2\}$

δ - is the transition function or total function, which associate each pair of state and input symbol from alphabet with a state $\delta:(s_i, x_j) \rightarrow s_k$

i - is the initial state

F - is the set of final or accept states, $F \subseteq S$

- The transition function δ for a DFA is interpreted in such a way that $\delta:(s_i, x_j) \rightarrow s_k$ if and only if the machine can move from state s_i to state s_k while reading the symbol x_j .
- A DFA $(S, \Sigma, \delta, i, F)$ accepts a nonempty string $x_1 x_2 \cdot \cdot \cdot x_n$ iff there is a sequence of states $s_0 s_1 \cdot \cdot \cdot s_n$ such that
 1. $s_0 = i$
 2. $s_n \in F$
 3. for each integer j from 1 to n , $\delta(s_{j-1}, x_j) = s_j$

if you are given some input string:

- The DFA begins in start state
- After reading each symbol, the DFA makes state transition with matching label.

After reading last symbol, DFA produces output:

- Accept if DFA is an accepting state.
- Reject otherwise.


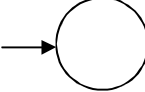


The program within the control mechanism is represented by a transition diagram.

Transition Diagrams

- 1) A transition diagram consists of a finite set of labelled circles called **states** connected by arrows called **arcs**.
- 2) Exactly one state, called the **initial state**, is indicated with a pointer or contains - symbol
- 3) One or more states, called **accept states** or **final states** are indicated with double circles.

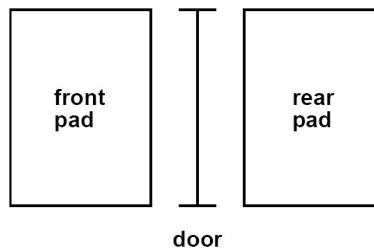
- 4) A string is **accepted** by the transition diagram if there is a sequence of labelled arcs leading from the initial state to an accept state which corresponds to the sequence of symbols (from left to right) which make up the string,

State transition diagram Symbols

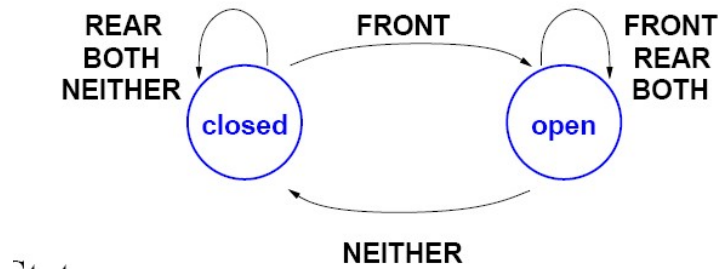
	State
	Start State
	Final State
	Transition

Example of a DFA: Automatic door

The automatic door has two pads as indicated in the diagram below. The controller opens the door when a person approaches, holds the door open until the person clears and does not open when someone is standing behind the door. The controller can be in either of the two states, OPEN or CLOSED. The sensors can be FRONT: person on front pad, REAR- person on rear pad, BOTH- people on both pads and NEITHER - no one on either pad.



Automatic door controller as a DFA

**Door movement:**

- When the door is closed and there is somebody on the rear pad or on both pads or there is no one on the pads, the door remains closed
- When the door is closed and somebody steps on the front pad the door opens
- When the door is open and somebody is on the front pad, rear pad, or on both pads the door stays open
- When the door is open and nobody is on the pads the door closes

Controller movement:

- When controller is in the state closed and the input is rear, both, or neither the controller remains in the state closed
- When controller is in the state closed and the input is front the controller moves to the state open
- When the controller is in the state open and the input is one of front, rear, both, the controller remains in the state open
- When the controller is in the state open and the input is neither the controller moves to the state closed

Transition Tables

Is used to summarise transition rules in instances where the list of rules is very long. Each row of the TT is the name of one of the states in the FA and each column of the TT is a letter of the input alphabet. The entries inside the TT are the new states that the FA moves into-the transition states (Cohen, 2001). A transition table is a two-dimensional array which summarises a transition diagram. If there is no arc labelled leaving a state then the destination state is labelled *error*.

State transition table for automatic door controller

		Input signals			
		NEITHER	FRONT	REAR	BOTH
state	Closed	Closed	Open	Closed	Closed
	Open	Closed	Open	Open	Open

Interpretation: $T(\text{state}, \text{input}) = \text{NewState}$

When the controller receives input signals it moves from one state to another. For example when the controller receives input signals NEITHER or REAR or BOTH and is in closed state it remains in that state.

Note that the door controller has only two states and such it requires less memory than say an elevator controller if the floor on which the elevator is on is regarded as a state. In this example the input may be the signals from the buttons.

Note that this controller is a computer that has just a single bit of memory that is used to record the controller's state

Other examples of computers with limited memory are controllers for dishwashers, electronic thermostats, parts of digital watches and calculators.

The above devices are designed by keeping the methodology and terminology of finite automata in mind.

Example 1 of an FA

1. Suppose $\Sigma = \{a, b\}$
2. There three states namely 1, 2 and 3 and state 1 is the initial state and state 2 is the only final state.
3. Let the following be the rules of the transition
 - a. Rule 1: from state 1 and input a go to state 3
 - b. Rule 2: from state 1 and input b go to state 2
 - c. Rule 3: from state 3 and input a go to state 1
 - d. Rule 4: from state 3 and input b go to state 2
 - e. Rule 5: from state 2 and any input stay at state 2

When $aaaaa$ string is presented to this FA, it does not lead us to the final state 2, so the string is not accepted or is rejected by this FA. The set of all strings that leave us in a final state is called the language defined by the FA.

The above FA accepts all strings that have a letter b in them and no other strings. Therefore the language associated with or accepted by this FA is the one defined by the RE:

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

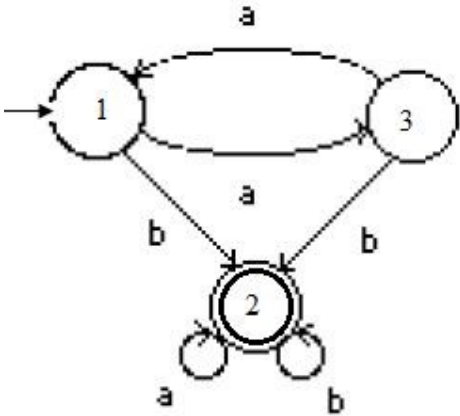
The TT for the FA described above is given below.

		a	b
start	1	3	2
	3	1	2

Transition Diagram

An FA is considered as a machine that has dynamic capabilities. It moves. It processes input. Something goes from state to state as the input is read. One way to represent an FA that feels more like a machine is a transition diagram. This representation makes it much easier to see that this FA accepts only strings with at least one 'b' in them.

FA using a transition diagram.

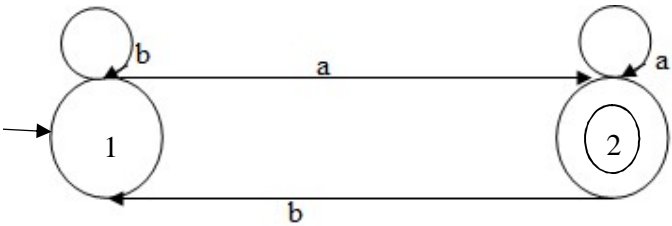


Example 2 of an FA

Question: draw a state transition diagram and a state table for a machine M which is defined as (S, Σ, δ, i, F)

Where $S = \{1, 2\}$ $\Sigma = \{a, b\}$ $\delta: (S \times \Sigma) \rightarrow S: \delta(1,a) \rightarrow 2,$
 $\delta(1,b) \rightarrow 1, \delta(2,a) \rightarrow 2, \delta(2,b) \rightarrow 1$ $i = 1$
 $F = \{2\}$

What language does the machine M accept?



State transition table

		Input signals		
		a	b	#
Start state	1	2	1	R
End state	2	2	1	A

Note that Λ is a special case because it does not appear in the transition function and is not in the alphabet Σ .

For all s an element of S , $\delta(s, \Lambda) = s$

Language: All strings over the alphabet that end in a

Note that when describing a language focus only on strings accepted by the machine rather than those rejected.

Practice Questions

Give the transition graph of a finite state automaton to accept the language;

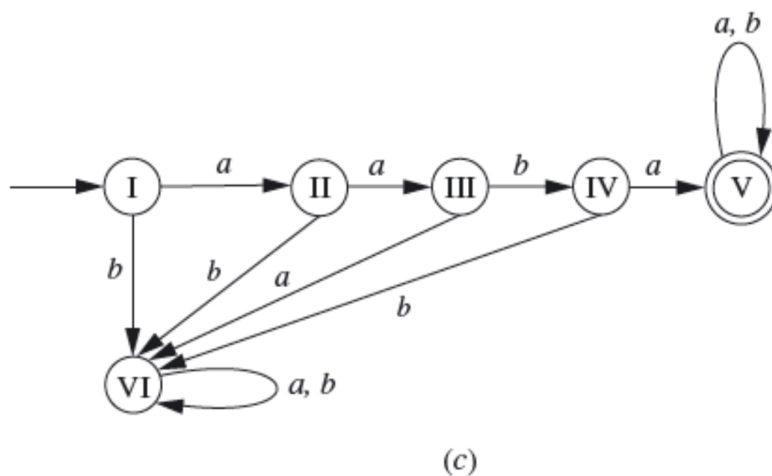
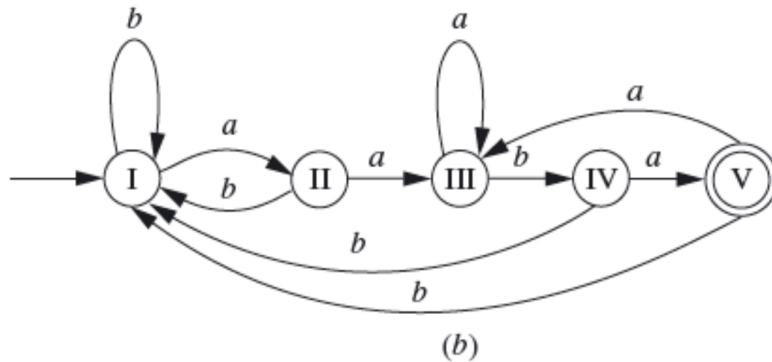
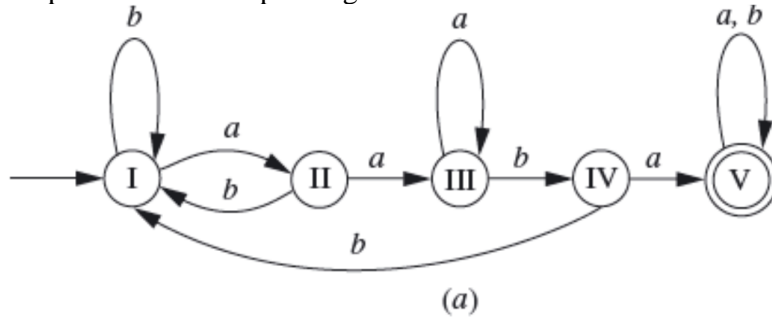
1. $L((a + b)^* a)$ – all strings ending in a and the shortest string is a
2. $L((a + b)^*)$ – all strings over a and b
3. $L(a + b)(a + b)^*$ – only eliminates Λ from $(a + b)^*$
4. $L((a+b)(a+b)(a+b))^*$
5. $L((a + b)(a + b))^*$ – all strings with an even number of symbols
6. $L(a + b)^* a(a + b)^* b(a + b)^*$ strings of $a's$ and $b's$ with an a preceding a b
7. $L(b^* ab^*)$ – all strings over a and b with precisely one a
8. $L(b^* ab^* ab^*)$ strings with exactly 2 $a's$
9. $L((a + b)(a + b)(a + b))$ – strings over a and b with 3 symbols, this can be expressed as $(a + b)^3$ which is just short hand not a RE.
10. $L(a(a + b)^* b + b(a + b)^* a)$ – accepts all strings over a and b with first and last symbols different
11. $L(a^* b^*)$ – all strings over a and b in which all $a's$ (if any) precede all $b's$ (also if any)
12. $L(a^* b^*)^*$ – all strings over a and b
13. $L(a(a+b)^* a + b(a+b)^* b + a + b)$ – all strings over a and b that start and end with the same symbol

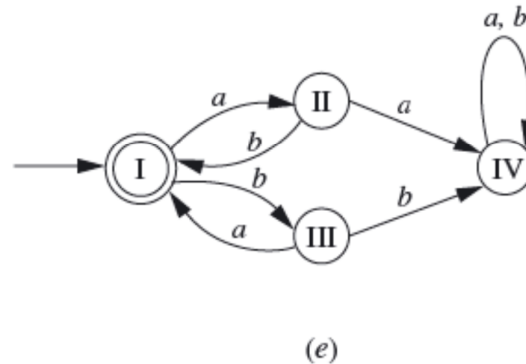
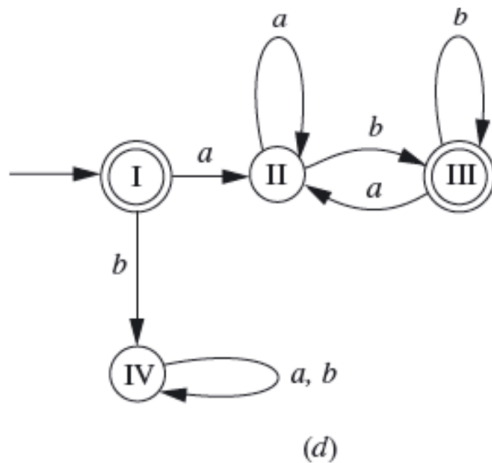
In each part below, draw an FA accepting the indicated language over $\{a, b\}$. Give the RE defining each language

- a. The language of all strings containing exactly two $a's$.
- b. The language of all strings containing at least two $a's$.
- c. The language of all strings that do not end with ab .
- d. The language of all strings that begin or end with aa or bb .
- e. The language of all strings not containing the substring aa .
- f. The language of all strings in which the number of $a's$ is even.
- g. The language of all strings in which both the number of $a's$ and the number of $b's$ are even.

- h. The language of all strings containing no more than one occurrence of the string aa .
(The string aaa contains two occurrences of aa .)
- i. The language of all strings in which every a (if there are any) is followed immediately by bb .
- j. The language of all strings containing both bb and aba as substrings.
- k. The language of all strings containing both aba and bab as substrings.
- l. The language that accepts the string $abaa$ and no other strings

For each of the FAs pictured below, give a simple verbal description of the language it accepts and the corresponding RE





Group Tutorial Questions:

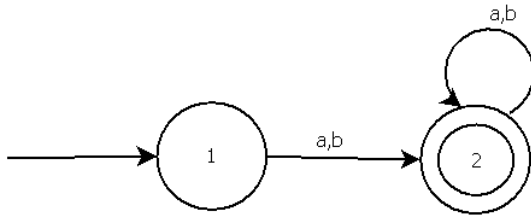
- a) Give the transition graph of a finite state automaton to accept the language; [15 marks]
- $(a+b)^*(aa+bb)(a+b)^*$
 - $a^*(aba^*b^*ba+abb+bbb^*)(a+b)^*$
 - $aa(a+b)^*bb + bb(a+b)^*aa + ab(a+b)^*ab + ba(a+b)^*ba$
 - the set of strings in $\{0, 1\}^*$ that are binary representations of integers divisible by 5.
- b) Let us consider the operation of a soft drink vending machine which charges 8 bond coins for a can. Initially the machine is waiting for a customer to come and put some coins, that is, it is in the waiting-for-customer state. Let us assume that only 1, 2 and 5 bond coins are used for simplicity. When a customer comes and puts in the first coin, say a 5 bond coin, the machine is no longer in the waiting-for-customer state. The machine has received 5 bond coin and is waiting for more coins to come. So we might say the machine is in the 5-bond coin state. If the customer puts in a 2 bond coins, then the machine has now received 7 bond coins. If the customer puts in a 1 bond coin, then the machine has now received 8 and it waits for the customer to select a soft drink. So the machine is in another state, say 8 bond coins. When the customer selects a soft drink, the machine must give the customer a can of soft drink. After that it stays in that state until another coin is put in to start the process anew or the operation may be terminated and start all over from the initial state. It is assumed that the machine terminates its operation when it receives 8 bond coins.

Required:

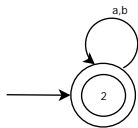
- Give the formal definition of a DFA modelling the solution to the above problem [5]
- Draw the Transition Table [5]
- Draw a DFA with the least number of states [5]
- Come up with a regular expression defining the language accepted by the machine [5]
- Give five strings which are accepted by the machine [5]
- Give five strings which are rejected by the machine. [5]
- Implement your solution using Python [10]

Examples:

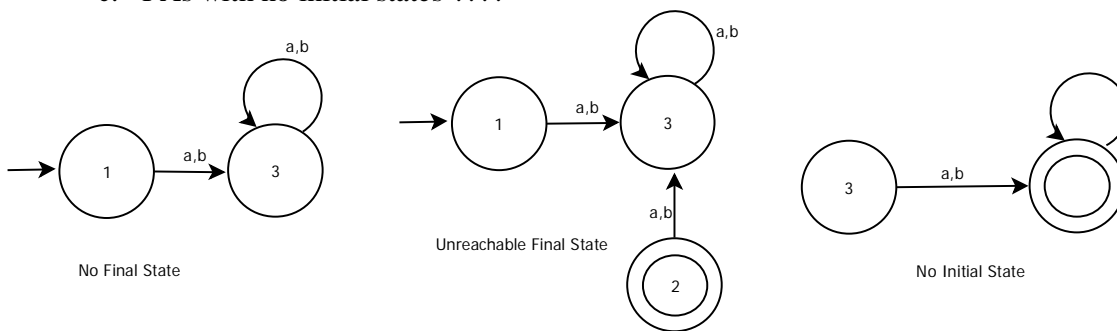
1. Machine that accepts strings over a and b except λ . The regular expression defining the language is $L(a+b)(a+b)^*$.



2. Machine that accepts all strings over a and b and the RE for this language is $L(a+b)^*$.



3. Machine that accepts no language
- FAs with no final states
 - FAs with unreachable final states
 - FAs with no initial states ????



FAs and their language

The world of FAs can be viewed in two ways, either by starting with an FA and try to analyse it to see what language it accepts or starting with the desired language in our mind and try to construct an FA that would act as a language recogniser or definer.

If given a language, one must be able to construct a machine for it and also if given a machine one must be able to deduce its language.

Example:

Machine that accepts the language of all words over the alphabet $\{a,b\}$ with an even number of letters.

Algorithm

Input: input string

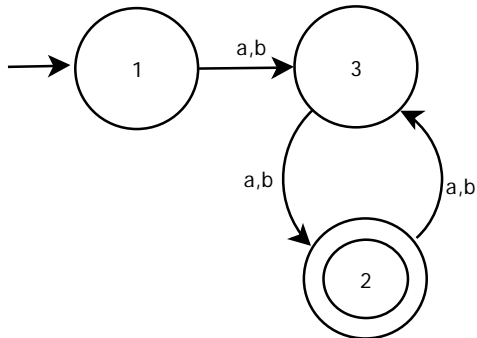
Output: Accept, Reject string

Set EVEN = True

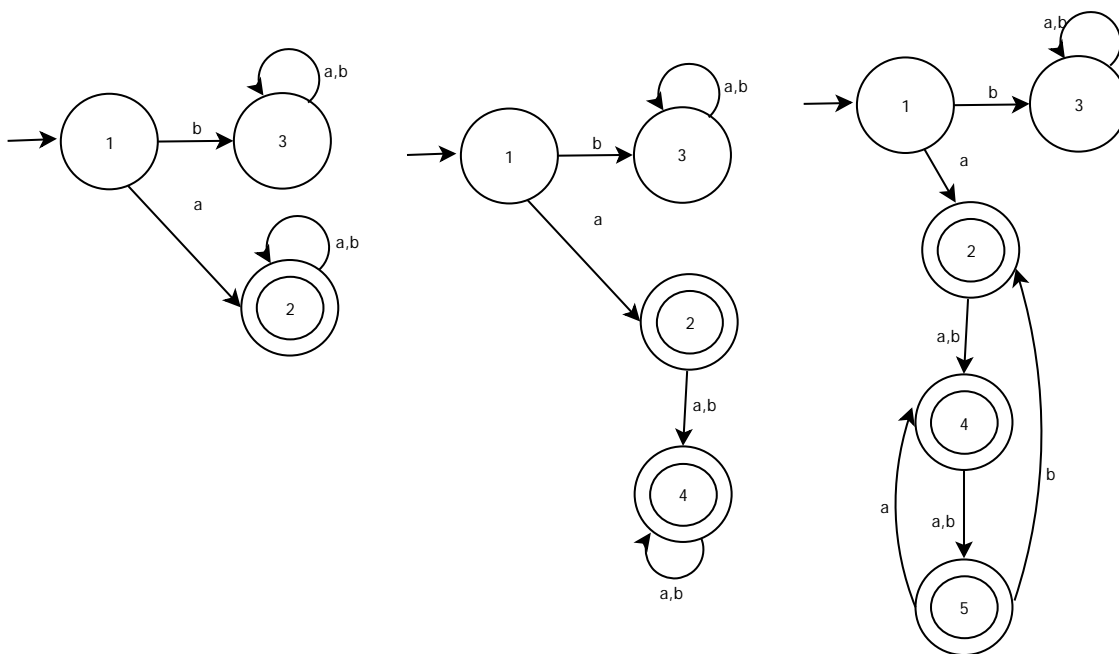
While not out of data do

Read an input letter

EVEN becomes not(EVEN)
If EVEN = True
Accept the string
Else reject string

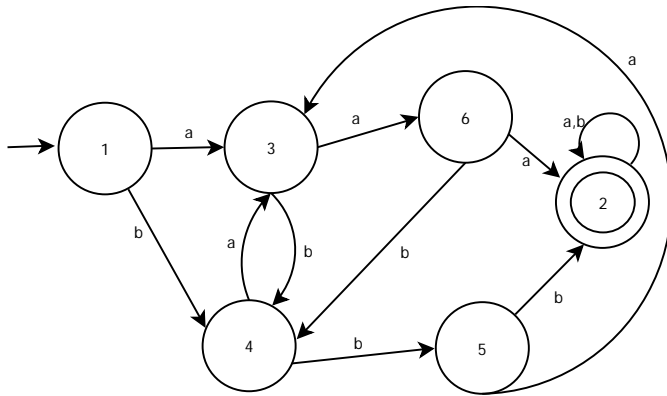


Note that it is possible to have at least one FA that accepts each possible language for example, below are three machines which accept words in the language $L(a(a+b)^*)$.

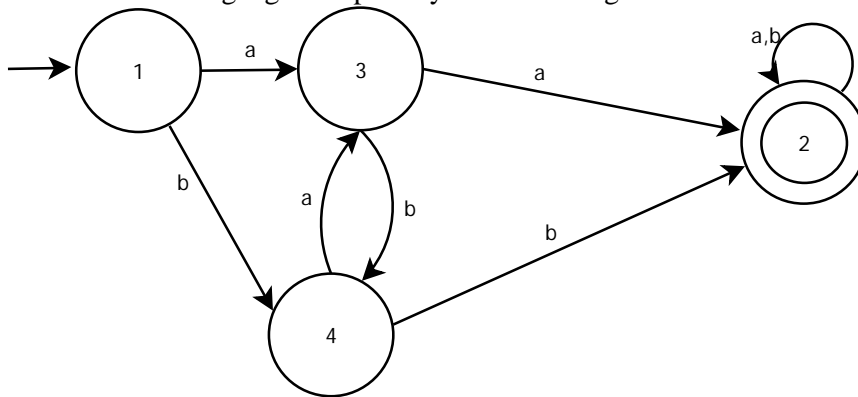


Example:

FA that accepts all words containing a triple letter either aaa or bbb and only those words

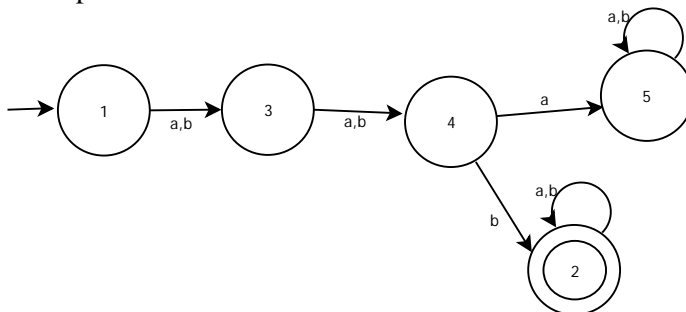
**Question**

Determine the language accepted by the machine given below:



The Machine accepts all string over $\{a,b\}$ that have a double letter in them and the language is defined by the RE $(a+b)^*(aa+bb)(a+b)^*$

Example:

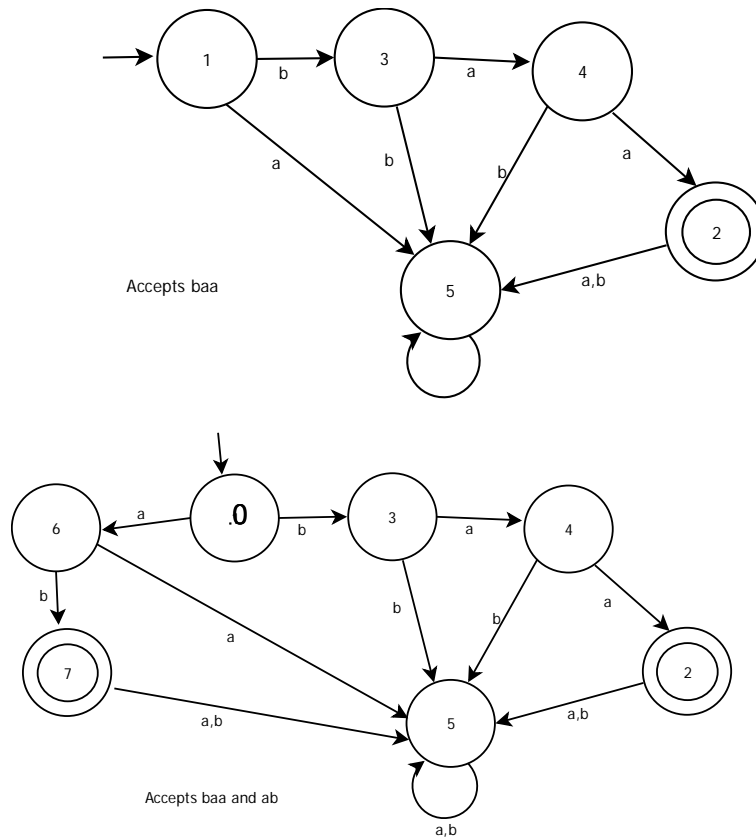


Accepts all strings over $\{a,b\}$ with b as the third letter and rejects all other words

The Res are $(a+b)(a+b)b(a+b)^*$ or $(aab+aba+ bab+ bbb)(a+b)^*$

Draw FAs that accept the following strings over $\{a,b\}$

- i. baa
- ii. baa and ab



Transition Graph

A Transition Graph (TG) is a collection of three things:

- i A finite set of states, at least one of which is designed as the start state and some (may be none) of which are designed as final states.
- ii An alphabet Σ of possible input letters from which input string are formed.
- iii A finite set of transitions that show how to go from one state to another based on reading specified **substrings** of input letters (possibly even the null string Λ).

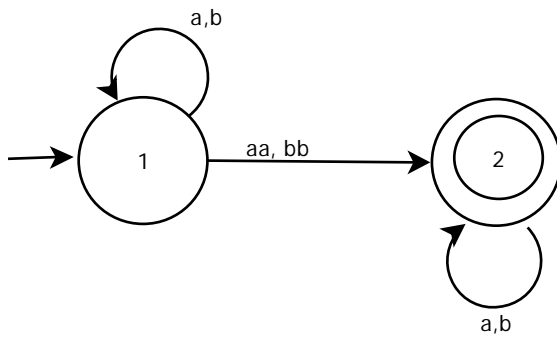
The last clause in the definition means that every edge is labelled by some strings of letters not necessarily only one letter. It is not a requirement that there be any specific number of edges emanating from any state. Some states may have no edge coming out of them at all and some may have thousands.

A successful path through a TG

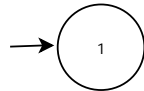
Is a series of edges forming a path beginning at some start state (there may be several) and ending at a final state

Examples:

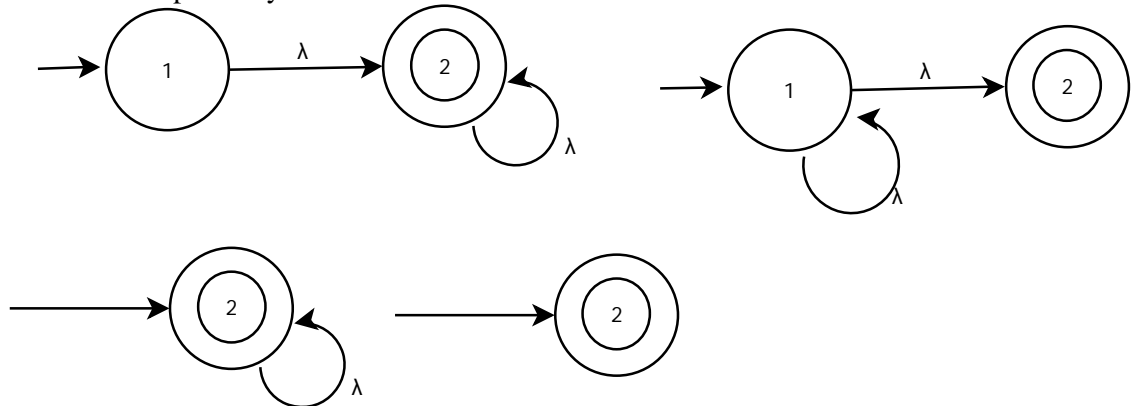
- i. TG that recognises all words that contain a double letter



ii. A TG that accepts nothing

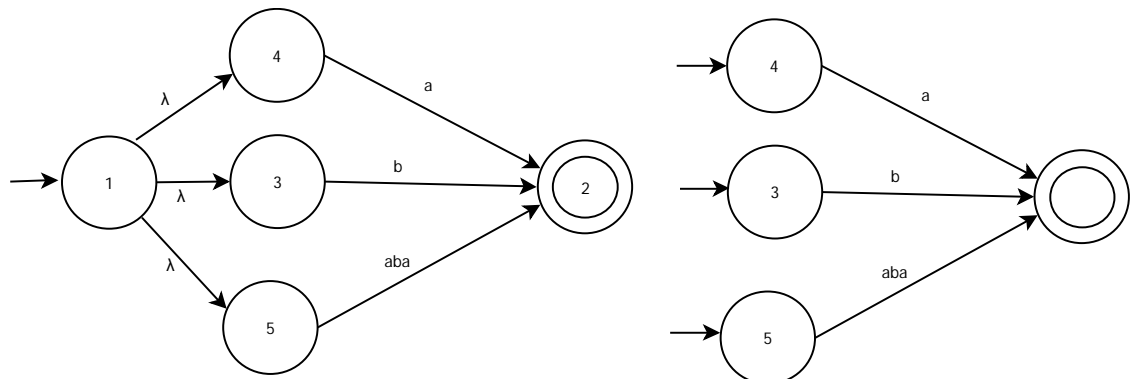


iii. TGs that accept λ only

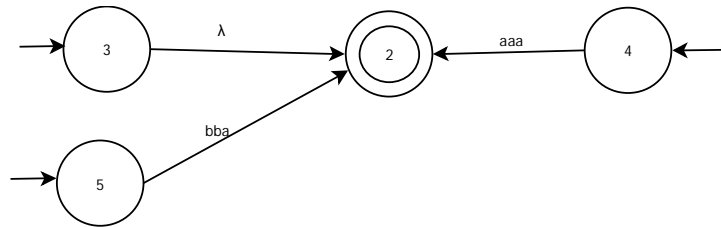


Note that if an edge is labelled with λ , it means we can take the ride it offers without consuming any letters from the input string.

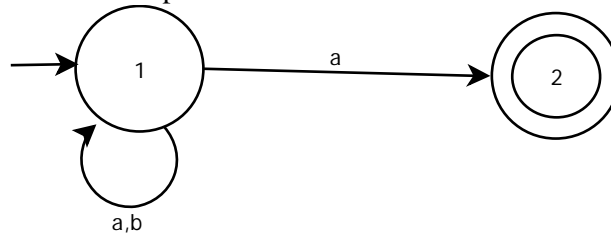
iv. TGs that accept a, b, and aba



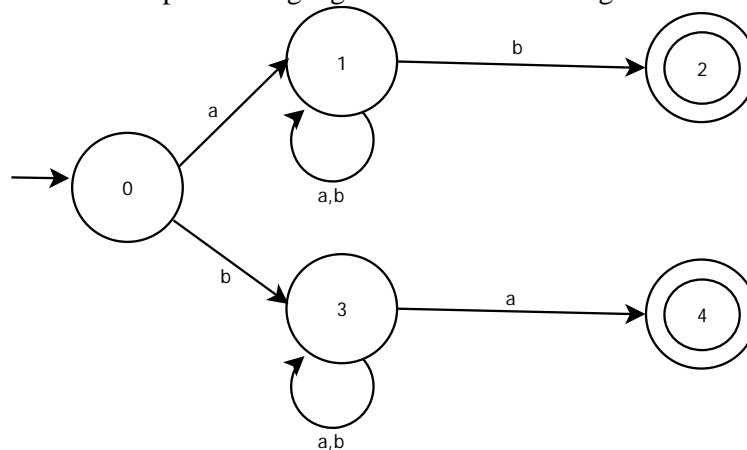
v. Machine that accepts λ , bba, aaa



vi. TG that accepts words that end in a



vii. TG that accepts the language of all words that begin and end with different letters



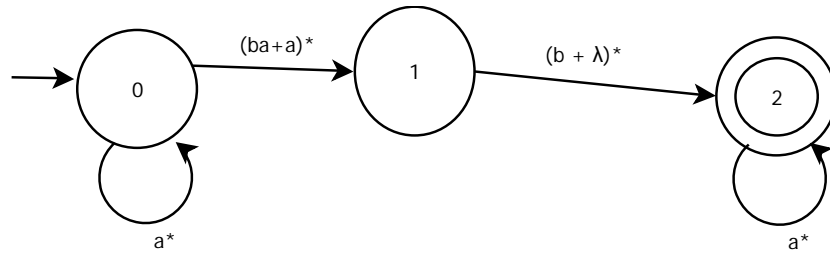
Generalised TG

is a collection of three things

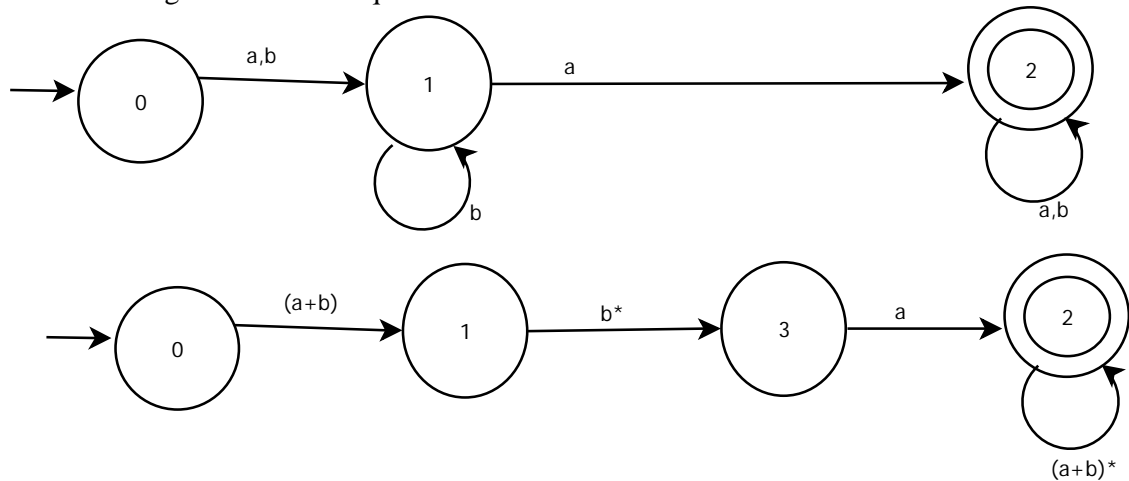
- A finite set of states, at least one of which is designed as the start state and some (may be none) of which are designed as final states.
- An alphabet Σ of possible input letters from which input string are formed.
- Directed edges connecting some pairs of states each labelled with RE

Examples

- Generalised Transition Graph that accepts all strings without a double b



ii The following machines are equivalent



Kleen's Theorem

States that any language that can be defined by:

- i Regular expression or
- ii Finite automata or
- iii Transition graph

Can be defined by all three methods.

Proofs

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

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

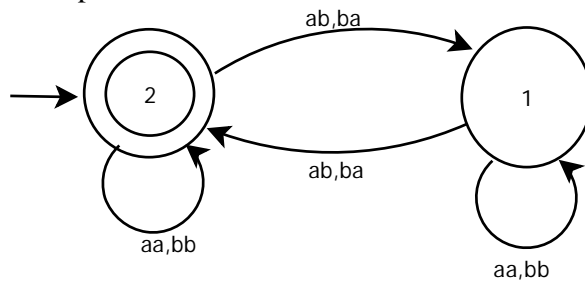
The proof of part2

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.

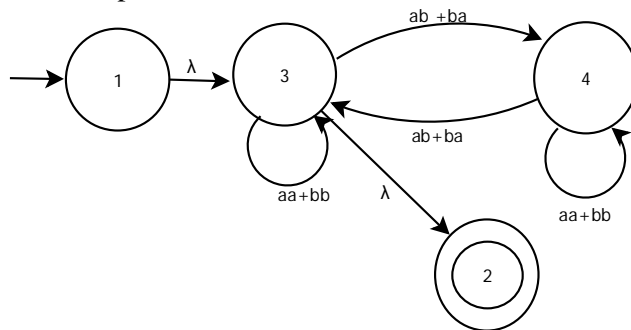
Algorithm:

- i Create a unique unenterable initial state and a unique final unleaveable state
- ii One by one in any order, by-pass and eliminate all the non-final or non start states in the TG. A state is by passed by connecting each incoming edge with each outgoing edge. The label of each resultant edge is the concatenation of the label on the incoming edge with the label on the loop edge if there is any and label on the outgoing edge
- iii When two states are joined by more than one edge going in the same direction unify them by adding their labels
- iv Finally, when all that is left is one edge from start to final, the label on that edge is a regular expression that generates the same language as was recognised by the original machine

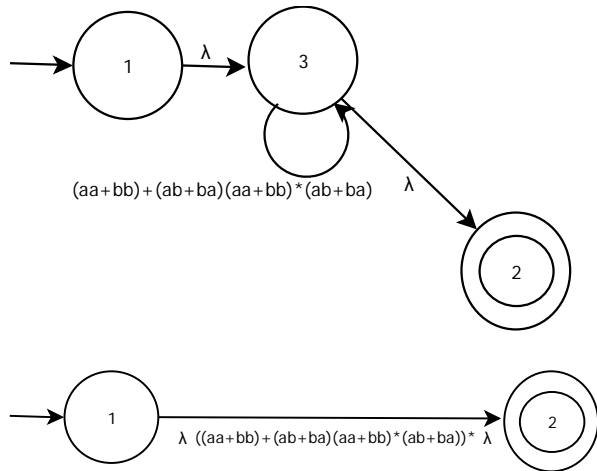
Example:



After Step 1

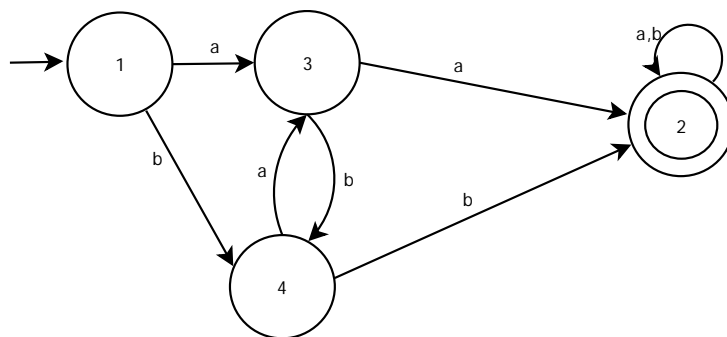


After eliminating state 4 it becomes

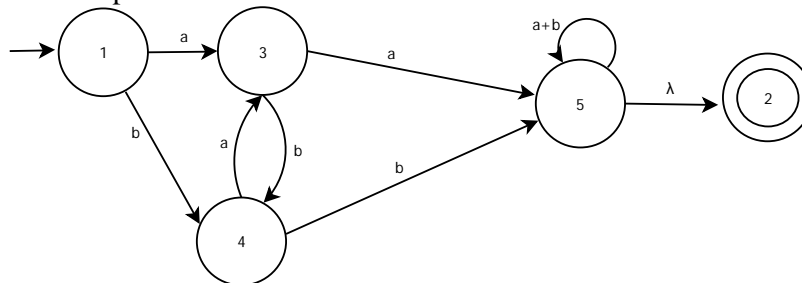


Question

Using the constructive algorithm generate the RE for the language defined by the Transition Graph given below.



After step 1:

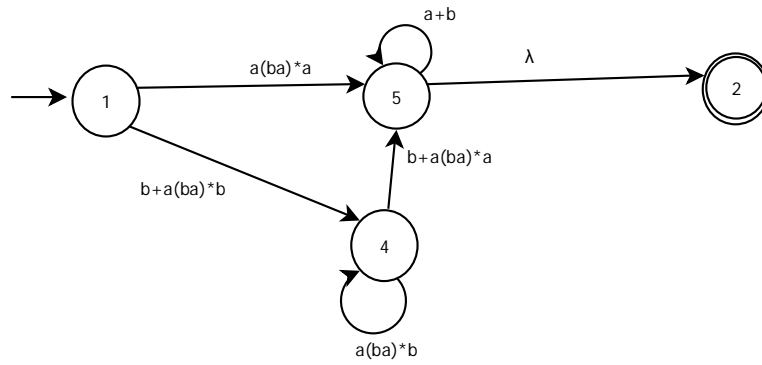


1-3-5 $a(ba)^*a$

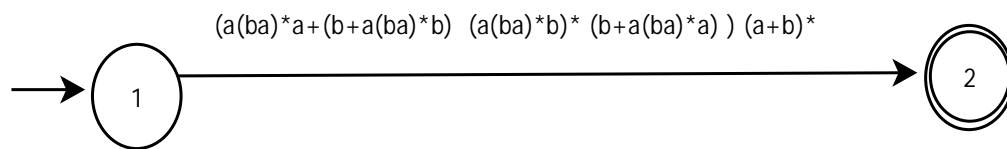
1-3-4 $a(ba)^*b$

4-3-5 $a(ba)^*a$

4-3-4 $a(ba)^*b$



1-4-5 $(b+a(ba)^*b) (a(ba)^*b)^* (b+a(ba)^*a)$



Proof of part 3

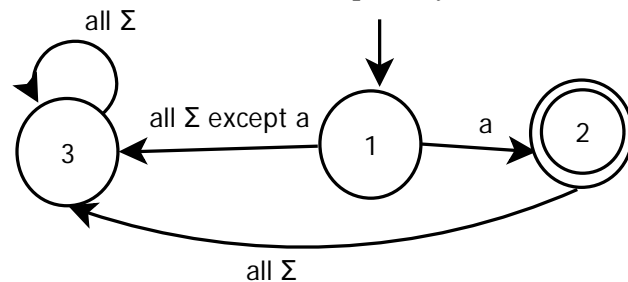
The proof will be by recursive definition and constructive algorithm. It is known that every RE can be built from letters of the alphabet Σ and Λ by repeated application of certain rules: addition, concatenation and closure. We should show that as we are building up a RE, we could at the same time be building up an FA that accepts the same language. Below is the presentation of the algorithm with four rules.

Rule 1

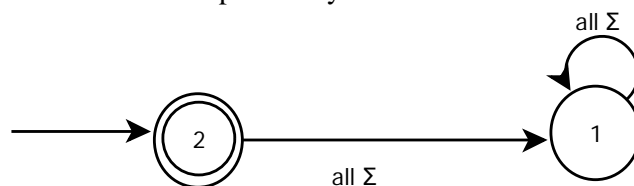
There is an FA that accepts any particular letter of the alphabet. There is an FA that accepts only the word Λ

Proof of Rule 1

If a is in Σ , then the FA accepts only the word a .



One FA that accepts Λ only is



Rule 2

If there is an FA called M_1 that accepts the language defined by the RE r_1 and there is an FA called M_2 that accepts the language defined by the RE r_2 , then there is an FA that we shall call M_3 that accepts the language defined by the RE $(r_1 + r_2)$

Proof of rule 2

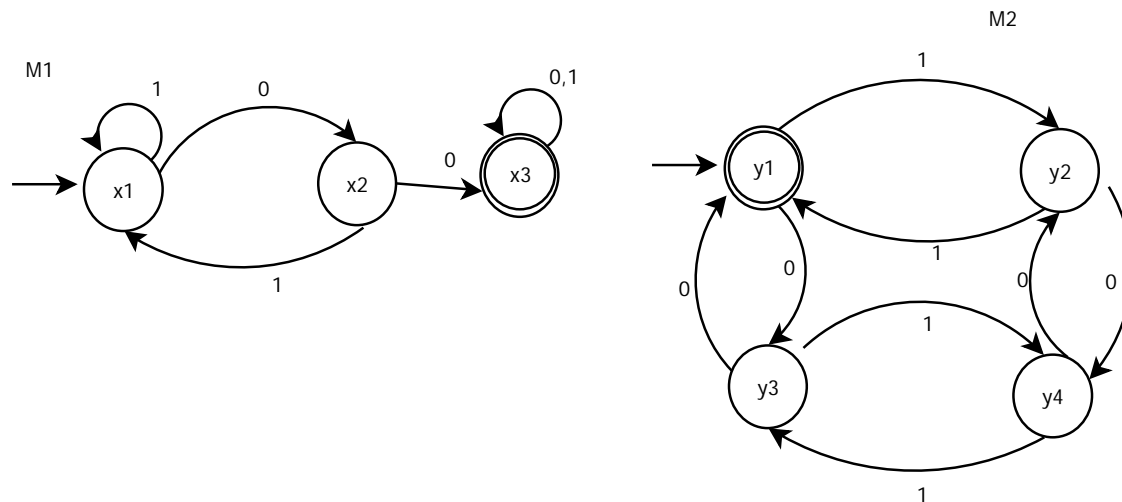
This rule shall be proved by showing how to construct the new machine in the most reasonable way from the two old machines. We shall prove that M_3 exists by showing how to construct it.

Algorithm

- Starting with the two machines M_1 with states $x_1, x_2, x_3 \dots x_n$ and M_2 with states $y_1, y_2, y_3 \dots y_n$ build a new machine M_3 with states $z_1, z_2, z_3 \dots z_n$ where each z is of the form $x_{\text{something}}$ or $y_{\text{something}}$.
- The combination state x_{start} or y_{start} is the start state of the new FA
- 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 look at what happens to the x part and the y part and go to the new z state accordingly. This can be written as a formula:
 $Z_{\text{new after } p} = \{x_{\text{new after } p \text{ on } M_1}\} \text{ or } \{y_{\text{new after } p \text{ on } M_2}\}$
- Useless states may or may not be removed

Example:

M_1 accepts all words with a double 0 in them and M_2 accepts all words that have both an even number of total 0's and an even number of total 1's



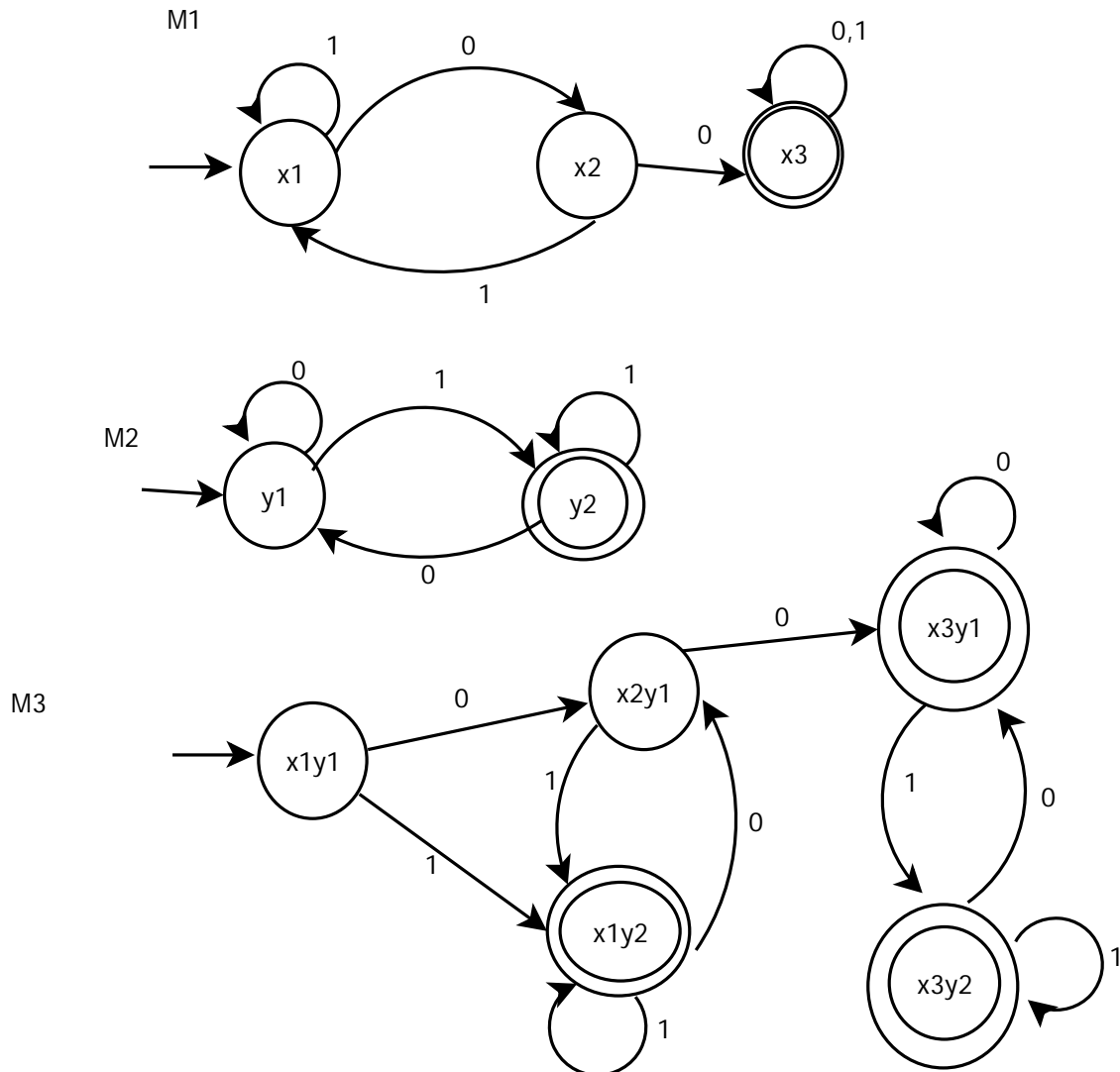
		0	0	1	1
Start/final z_1	x_1 or y_1	z_2	x_2 or y_3	z_3	x_1 or y_2
z_2	x_2 or y_3	z_4	x_3 or y_1	z_5	x_1 or y_4
z_3	x_1 or y_2	z_6	x_2 or y_4	z_1	x_1 or y_1
Final z_4	x_3 or y_1	z_7	x_3 or y_3	z_8	x_3 or y_2

z5	x1 or y4	z9	.	z10	.
z6	x2 or y4	z8	.	z10	.
Final z7	x3 or y3	z4	.	z11	.
Final z8	x3 or y2	z1		z4	
z9	.	z1		z1	
z10	.	z12		z5	
Final z11	.	z8		z7	
Final z12		z7		z3	

Draw the FA from the Transition table

Example

M1 accepts all words with a double 0 in them and M2 accepts all words ending in 1.



Rule 3

If there is an FA that accepts the language defined by a RE r and there is an FA that accepts the language defined by the RE s there is an FA that accepts the language defined by the RE rs .

Algorithm

- i. Make a z-state for every non-final x-state in M_1 reached before ever hitting a final state on M_1
- ii. For each final state in M_1 , we establish a z-state that expresses the options that we are continuing on M_1 or beginning on M_2

Are in $x_{\text{something}}$, which is a final state but still continuing on M_1

Or

Have finished the M_1 part of the string and have jumped to y_1 to continue tracing the remainder of the input string on M_2
- iii. After reaching a jump to M_2 state, any other states we reach have an x and a y possibility like the z states in union machines, with the additional possibility that every time we hit yet another final state on M_1 machine, we may again exercise the option of jumping to y_1 . This implies that every time we pass through a final state while processing the M_1 part of the input string we jettison an alter-ego jumping to y_1 that runs on the M_2 machine

Every z state can have the nature of one and only one x-state, but a whole set of possible y-states. So the full nature of a z-state is:

- Are in $x_{\text{something}}$ continuing on M_1

Or

Are in a set of $y_{\text{something}}$ continuing on M_2
- iv. Note that the transition from one z-state to another is determined by the transition rules in M_1 and M_2
 - v. Any z-state is an accept if it contains a y-machine final state as a possible position for the input

Example

The machine for the product of languages of L_1 and L_2 .

L_1 – all words that start with a 1 and

L_2 – all words that end with a 1

Procedure

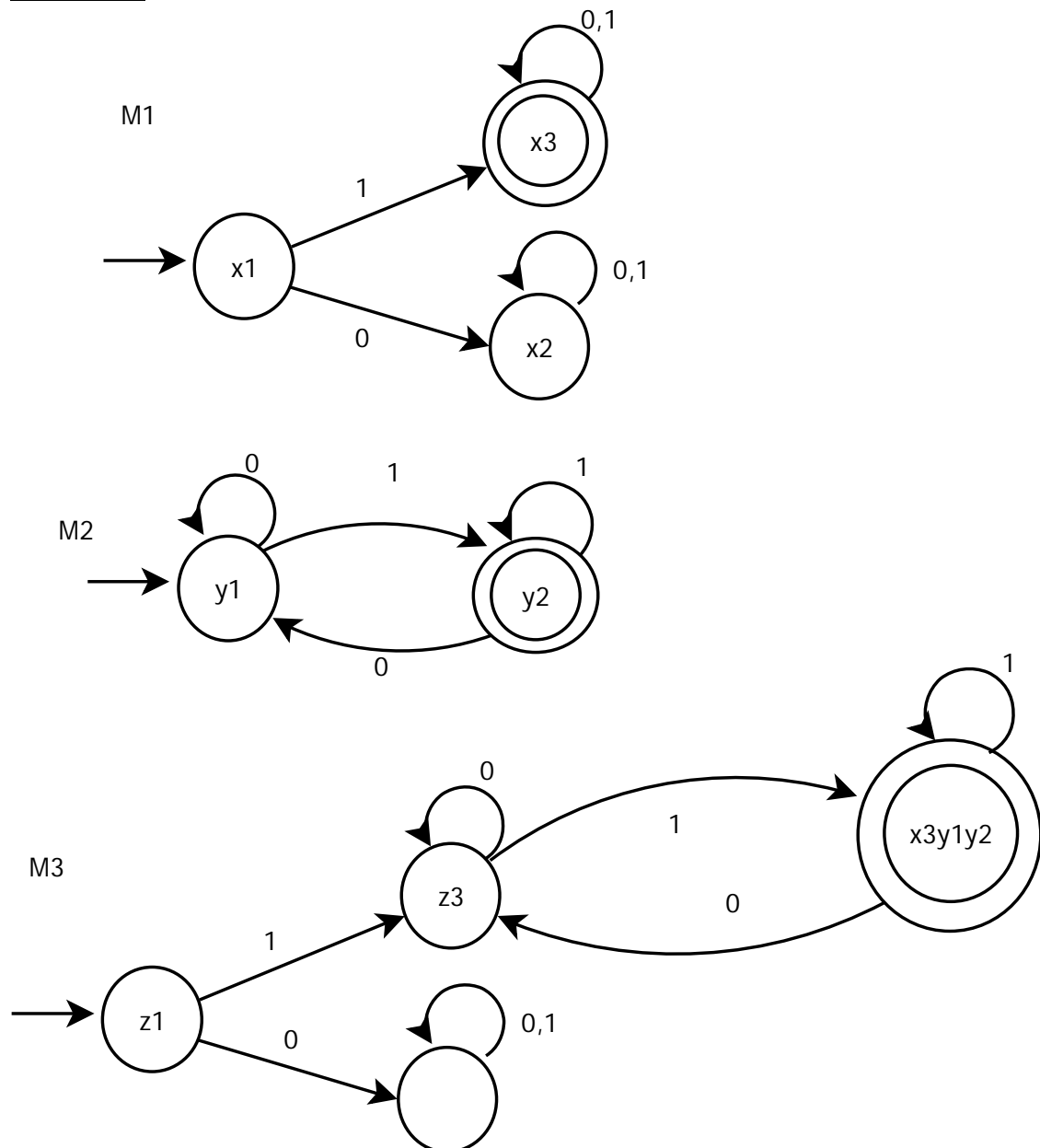
- i. Begin in x_1 which we shall call z_1 , if we read a 0 we go to x_2 which we call z_2 . If we read a 1 we go to x_3 which being a final state means that we have the option of jumping to y_1 . $Z_3 = x_3$ or y_1
- ii. From z_2 , like x_2 both a 0 and 1 takes us to z_2
- iii. In z_3 if we are in x_3 condition and we read a 0, we stay in x_3 or we choose to jump to y_1 . If we were in z_3 in the y_1 condition already and we read a 0 we would loop back to y_1 on the M_2 machine. So if we are in z_3 (x_3 or y_1) and we read a 0 we end up in z_3
- iv. If we are in z_3 (x_3 or y_1) and we read a 1, we stay there or use the occasion to jump to y_1 . If we were in z_3 already in the y_1 condition, then the 1 would take us to y_2 . Therefore z_4 a new state

must be equal to x_3 or y_1 or y_2 . If the input processing ends in this state, then it will be accepted because it may have gotten to the final state on the M2 machine. So z_4 is a final state for m_3

- v. If in z_4 and we read a 0, x_3 goes to x_3 staying on M_1 or x_3 goes to x_3 then jumps to y_1 on M_2 or y_1 stays in y_1 or y_2 goes to y_1 . Therefore from z_4 a 0 takes us to x_3 or y_1 which is z_3 .
If in z_4 and we read the input letter 1, x_3 goes to x_3 staying on M_1 or x_3 goes to x_3 which jumps to y_1 or y_1 goes to y_2 or y_2 loops back to y_2 . So from z_4 a 1 will loop us back to z_4

M3 looks like:

	0	1
z_1	z_2	z_3
z_2	z_2	z_2
z_3	z_3	z_4
z_4	z_3	z_4

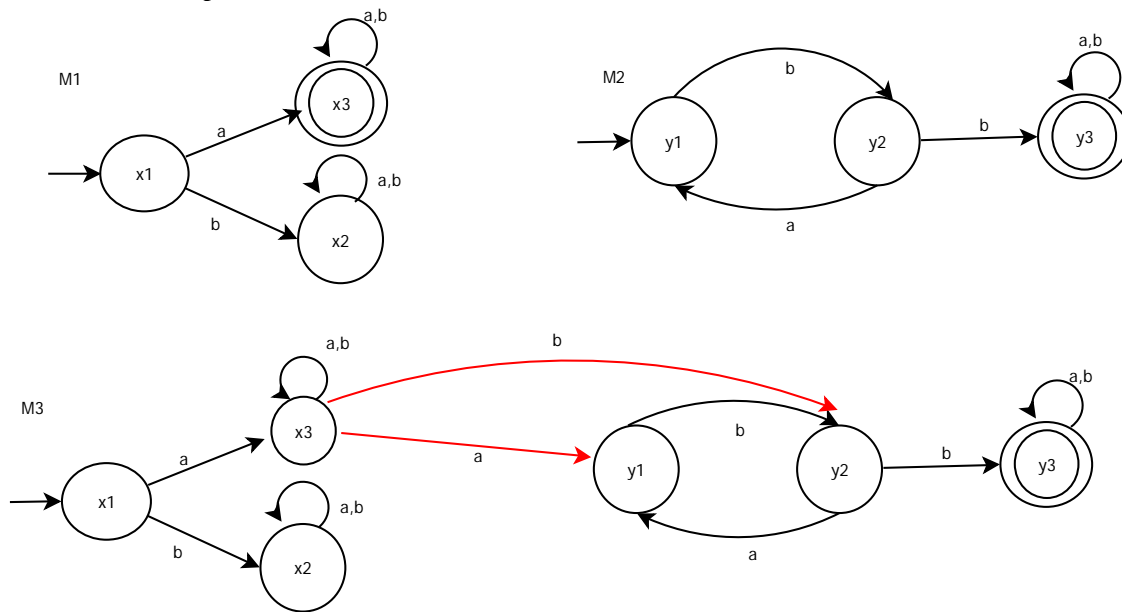


Note that $M_1M_2 \neq M_2M_1$

Another algorithm for finding the product

- i. Connect all final states of the first TG to all final states pointed to the initial state of the second TG and label them with the same symbols
- ii. The final states of the first TG cease to be final states but the final states of the second TG remain final states
- iii. The start state of the first machine remains the start state but that of the second cease to be a start state.

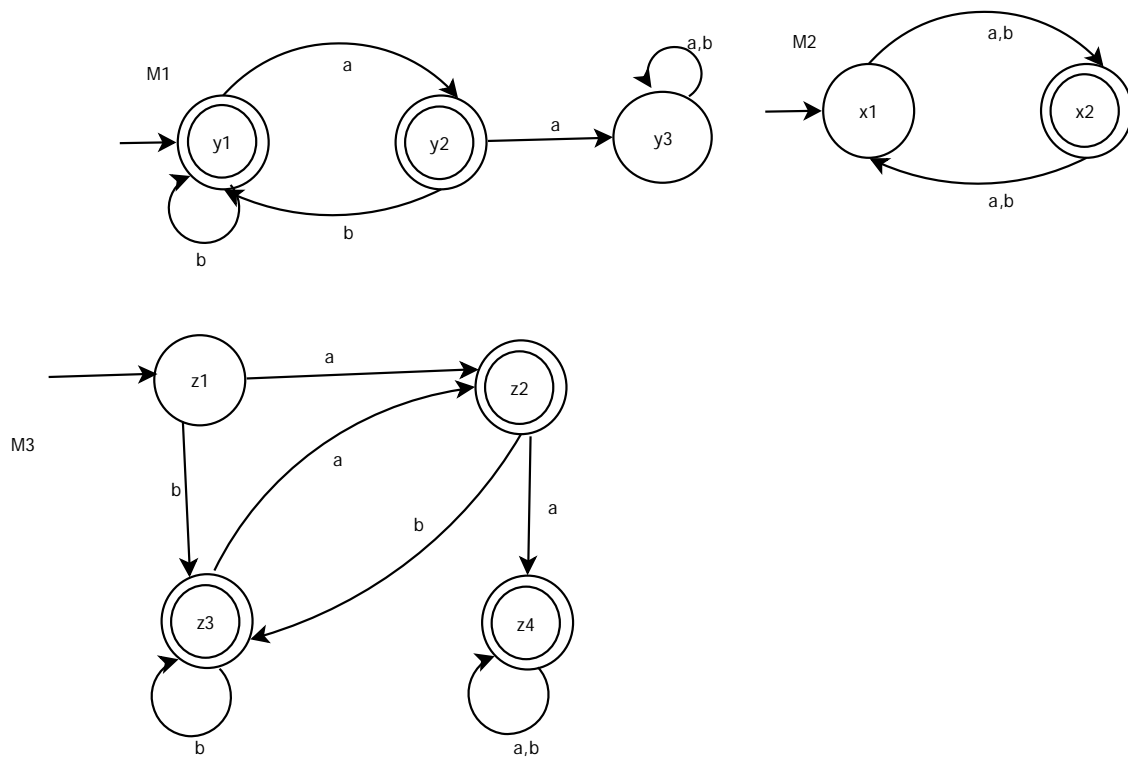
For example



Example

- i. M1- words that do not contain double a
- ii. M2- words with an odd number of letters

Machine that accepts the product language $L_1(M_1) L_2(M_2)$



Note that if a word w has an odd number of letters, we can factor it as $(\Lambda)(w)$
 Where Λ is in L_1 and w is in L_2 while if it has an even number of letters, we factor it as $w =$
 (firstletter)(the rest)

Rule 4

If r is a RE and M_1 is an FA that accepts exactly the language defined by r , then there is an FA called M_2 that will accept exactly the language defined by r^*

Proof of rule 4

The rule shall be proved by a machine that accepts r^* given a machine that accepts r

Algorithm

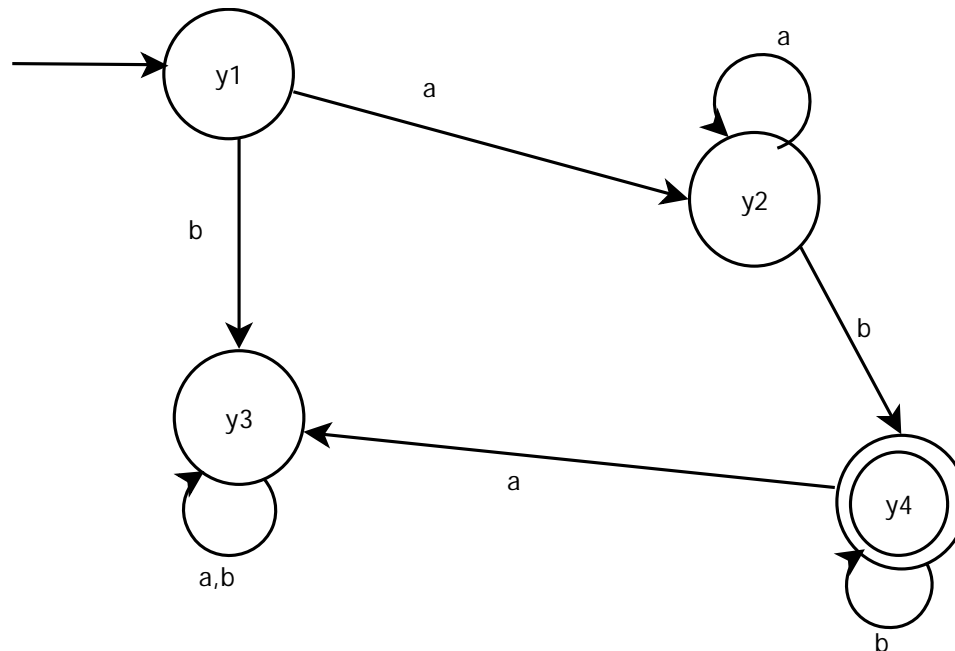
Given an FA whose states are $x_1, x_2, x_3 \dots x_n$, an FA that accepts the Kleen closure of the language of the original machines can be built as follows:

- Create a state for every subset of x 's. cancel any subset that contains a final state x but does not contain the start state
- For all the remaining non-empty states, draw an a edge and a b edge to the collection of x -states reachable in the original FA from the component x 's by a and b edges respectively. If a state is final consider the possibility of going back to the final state
- Call the null subset an initial and final state and connect it to whatever states the original start state is connected to by a and b edges, even possibly the start state itself.
- Put final state signs to every state containing an x -component that is a final component of the original FA

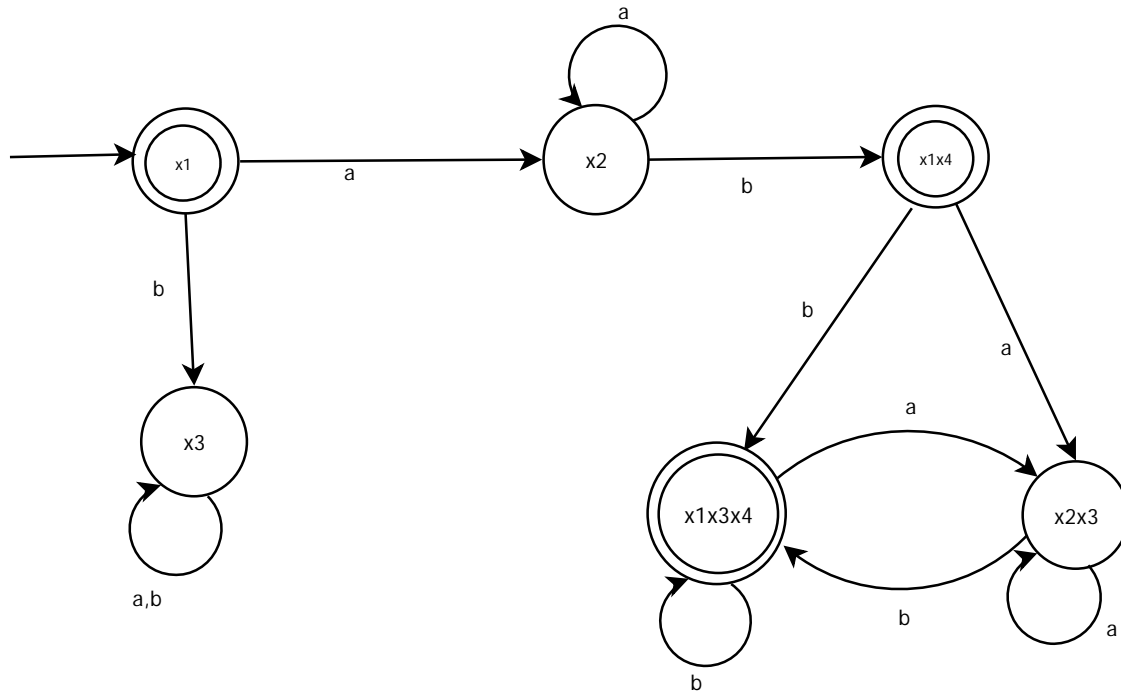
Example

Find the Kleen closure of the machine that accepts all words over a and b where all a 's (at least 1) come before all the b 's (at least 1)

$\{\}, x_1, x_2, x_3, \textcolor{red}{x_4}, x_1x_2, x_1x_3, x_1x_4, x_2x_3, \textcolor{red}{x_2x_4}, \textcolor{red}{x_3x_4}, x_1x_2x_3, x_1x_3x_4, \textcolor{red}{x_2x_3x_4}, x_1x_2x_4, x_1x_2x_3x_4$

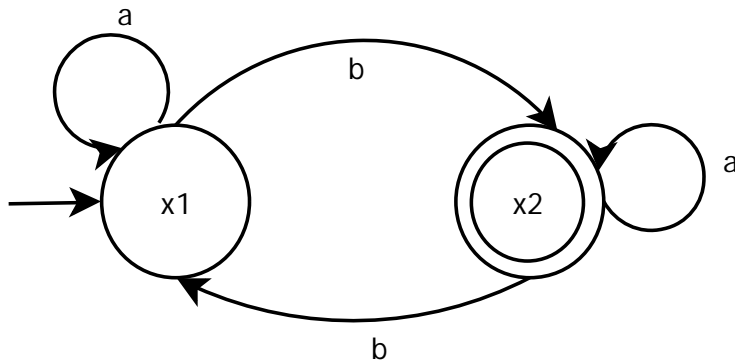


The RE for the machine is aa^*bb^*



Note that if the initial state has a loop two states are required to represent it. When the original start state is converted to a new start state it can either be the original start state and a final state or original start state and a non-final state. $Z_1 = x_1$ and final state and $z_2 = x_1$ and nonfinal state

Find the Kleen closure of the machine given below



The first state is 4 which is like state 1 except that it is a final state if in state 4 and read an a you remain in state 1 but this time in its capacity as a nonfinal state. We have to give a different name to this state lets call it 5. $4 = x_1$ and a final state, $5 = x_1$ and a non-final state.

If in state 4 and we read a b we go to state 2 but since state 2 is a final state we include the possibility of going back to state 1. So state 6 is 1 or 2 and is final state because of 2

If in state 5 and we read an a we stay in state 5 and if in state 5 and we read a b we go to state 6.

If we are in state 6 and we read an a we remain in that state because if we are in state 1 we would stay there and the same is true of state 2

If we are in state 6 and we read a b we remain in that state

