WELL COME TO

Automata Theory and Computability

Course(18CS54)

Course i/c : Dr.S G Gollagi

Module-I

Content to be covered:

- **Why study the Theory of Computation, Languages and Strings:**
  - ✓ Central Concepts of Automata Theory**:** Alphabets, Strings, Languages
  - ✓ A Language Hierarchy
- **Finite State Machines (FSM):**
  - ✓ Introduction
  - ✓ Deterministic FSM,
  - ✓ Regular languages, Designing FSM,
  - ✓ Nondeterministic FSMs,
  - ✓ Simulators for FSMs,
  - ✓ Minimizing FSMs,
  - ✓ Canonical form of Regular languages,
- **Finite State Transducers, Bidirectional Transducers**.

(Textbook 1: Ch 1,2, 3, 5.1 to 5.10)

What is Automata?

Automata Theory is a branch of computer science that deals with designing abstract self-propelled computing devices that follow a predetermined sequence of operations automatically.
An automaton with a finite number of states is called a **Finite Automaton**

Why Study the Theory of Computation or automata theory?

Implementations come and go.

IBM 7090 Programming in the 1950's

```
ENTRY       SXA     4,RETURN
            LDQ     X
            FMP     A
            FAD     B
            XCA
            FMP     X
            FAD     C
            STO     RESULT
RETURN      TRA     0
A           BSS     1
B           BSS     1
C           BSS     1
X           BSS     1
TEMP        BSS     1
STORE       BSS     1
            END
```

$Ax^2 + Bx + C$

Goals of Problem Solving

Principles of Problems:

- Does a solution exist?
  - If not, is there a restricted variation?
- Can solution be implemented in fixed memory?
- Is Solution efficient?
  - Growth of time & memory with problem size?

Applications of the automata Theory

- Used in design of Lexical analyzer of compilers which breaks source program into tokens like identifies, Keywords etc..

- Software for designing and checking the behavior of the Digital circuits.
- FSMs (finite state machines) for vending machines, Traffic signals, communication protocols, & building security devices.
- String Matching: Searching words, phrase and other pattern in large bodies of text(like web pages)
- Interactive Computer games as nondeterministic FSMs.
- Used in Natural languages processing: for speech to text and text to speech conversions.
- Artificial Intelligence: Medical Dignosis,Factory Scheduling etc..

The Central Concepts of Automata Theory

(Alphabets, Strings, Languages etc.)

This is one of MOST important Section.
It includes the TERMINOLOGY required to be successful in this course.

KNOW this section & ALL DEFINITIONS!!

Alphabet - Σ

- An alphabet is a non-empty, finite set of characters/symbols
- Use Σ to denote an alphabet
- Examples

$\Sigma = \{ a, b \}$

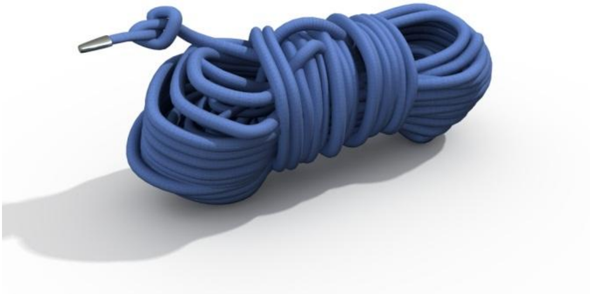
$\Sigma = \{ 0, 1, 2 \}$

$\Sigma = \{ a, b, c, \ldots z, A, B, \ldots Z \}$

$\Sigma = \{ \#, \$, *, @, \& \}$

String

- A ***string*** is a finite sequence, possibly empty, of characters or symbols drawn from some alphabet Σ.
- ε is the empty string
- Σ^* is the set of all possible strings over an alphabet Σ.



Example Alphabets & Strings

Alphabet name	*Alphabet symbols*	*Example strings*
The lower case English alphabet	{a, b, c, …, z}	ε, aabbcg, aaaaa
The binary alphabet	{0, 1}	ε, 0, 001100,11
A star alphabet	{★ , ✪ , ✫ , ✯, ✡, ✰}	ε, ✪✪, ✪✯✯✰★✰
A music alphabet	{𝅝, 𝅗𝅥, 𝅘𝅥, 𝅘𝅥𝅮, 𝅘𝅥𝅯, 𝅘𝅥𝅰, ●}	ε, 𝅘𝅥𝅝,𝅝𝅝𝅘𝅥𝅰

Functions/Operations on String

Length*:*

- $|s|$ is the length of string s
- $|s|$ is the number of characters in string s.

$|\varepsilon| = 0$

$|\texttt{1001101}| = 7$

$\#_c(s)$ is defined as the number of times that c occurs in s.

Ex. : $\#_a(\texttt{abbaaa}) = 4$.

Powers of an alphabet:

Let Σ be an alphabet.

- Σ^k = the set of all strings of length k
- $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots$
- $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \cup \ldots$

For **example**: $\sum^0 = \{ \varepsilon \}$, k =0

$\sum^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}$
k = 3

Σ^*- Kleene closure

- Σ^* is defined as the set of all possible strings of any length that can be formed from the alphabet Σ
  - Σ^* is a language
- Σ^* contains an *infinite* number of strings
  - Σ^* is *countably infinite*

Σ* Example

Let $\Sigma = \{$a, b$\}$

$\Sigma^* = \{\varepsilon,$ a, b, aa, ab, ba, bb, aaa, aab, … $\}$

Σ^+ - Positive closure

- **Definition:** The set Σ^+ is the infinite set of all possible strings of all possible lengths over Σ excluding $\boldsymbol{\varepsilon}$.
- **Representation** – $\Sigma^+ = \Sigma_1 \cup \Sigma_2 \cup \ldots\ldots$

  (or) $\Sigma^+ = \Sigma^* - \{\varepsilon\}$
- **Example** – If $\Sigma = \{a, b\}$,

  then $\Sigma^+ = \{a, b, aa, ab, bb, ba, \ldots\ldots\ldots\}$

Other functions on Strings

Concatenation*:* the ***concatenation*** of 2 strings *s* and *t* is the string formed by appending t to s; written as s||t or more commonly, *st*

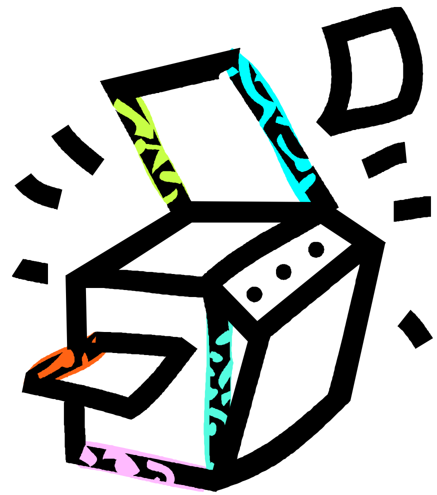
Example:

If x = `good` and y = `bye`, then xy = `goodbye` and yx = `byegood`

- Note that $|xy| = |x| + |y|$
- ε is the identity for concatenation of strings. So,
  $\forall x\ (x\varepsilon = \varepsilon x = x)$
- Concatenation is associative. So,
  $\forall s, t, w\ ((st)w = s(tw))$

Replication: For each string w and each natural number k, the string w^k is:

$$w^0 = \varepsilon$$
$$w^{k+1} = w^k w$$



Examples:

$\mathtt{a}^3 = \mathtt{aaa}$

$(\mathtt{bye})^2 = \mathtt{byebye}$

$\mathtt{a}^0\mathtt{b}^3 = \mathtt{bbb}$

$\mathtt{b}^2\mathtt{y}^2\mathtt{e}^2 = ??$

Reverse: For each string w, w^R is defined as:

if $|w| = 0$ then $w^R = w = \varepsilon$

if $|w| = 1$ then $w^R = w$

if $|w| > 1$ then:

$\exists a \in \Sigma\ (\exists u \in \Sigma^*\ (w = ua))$

So define $w^R = a\,u^R$

A Language

A **language** is a (finite or infinite) set of strings over a (finite) alphabet Σ.

Examples: Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$

Some languages over Σ:

$L_1 = \varnothing = \{\ \}$ // the empty language, no strings

$L_2 = \{\varepsilon\}$ // language contains only the empty string

$L_3 = \{\mathtt{a}, \mathtt{b}\}$

$L_4 = \{\varepsilon, \mathtt{a}, \mathtt{aa}, \mathtt{aaa}, \mathtt{aaaa}, \mathtt{aaaaa}\}$

so on…

Description Languages

Remember we are defining a set

Set Notation:

$$L = \{ w \in \Sigma^* \mid \text{description of } w\}$$

$$L = \{ w \in \{a,b,c\}^* \mid \text{description of } w\}$$

- "Description of w" can take many forms but must be precise
- Notation can vary, but must precisely define

Example Language Definitions

$L = \{\mathtt{w} \in \{\mathtt{a}, \mathtt{b}\}^* \mid \text{all } \mathtt{a}\text{'s precede all } \mathtt{b}\text{'s}\}$

- `aab`, `aaabb`, and `aabbb` are in *L*.
- `aba`, `ba`, and `abc` are not in *L*.

.

Example Language Definitions

Let $\Sigma = \{a, b\}$

- $L = \{ w \in \Sigma^* : |w| < 5\}$
- $L = \{ w \in \Sigma^* \mid$ w begins with b$\}$
- $L = \{ w \in \Sigma^* \mid \#_b(w) = 2\}$
- $L = \{ w \in \Sigma^* \mid$ each a is followed by exactly 2 b's$\}$
- $L = \{ w \in \Sigma^* \mid$ w does not begin with a$\}$

The Membership Problem

Given a string $w \in \Sigma^*$ and a language L over Σ, decide whether or not $w \in L$.

Example:

Let w = 100011

Q) Is $w \in$ the language of strings with equal number of 0s and 1s?

Operation on Languages

- Cardinality of a Language: the number of strings in the language L.
- Denoted as $|L|$
- Smallest language over any Σ is $\emptyset$, with cardinality 0.
- The largest is Σ^*.

Concatenation of Languages

Definition 1. Given languages L_1 and L_2, we define their *concatenation* to be the language $L_1 \circ L_2 = \{xy \mid x \in L_1,\ y \in L_2\}$

Example 2.

- $L_1 = \{\text{hello}\}$ and $L_2 = \{\text{world}\}$ then $L_1 \circ L_2 = \{\text{helloworld}\}$
- $L_1 = \{00, 10\}$; $L_2 = \{0, 1\}$. $L_1 \circ L_2 = \{000, 001, 100, 101\}$
- $L_1 =$ set of strings ending in 0; $L_2 =$ set of strings beginning with 01. $L_1 \circ L_2 =$ set of strings containing 001 as a substring
- $L \circ \{\epsilon\} = L$. $L \circ \emptyset = \emptyset$.

Other Examples:

L_1 = {cat, dog}
L_2 = {apple, pear}

$L_1 L_2$ = {catapple, catpear, dogapple, dogpear}
$L_2 L_1$ = {applecat, appledog, pearcat, peardog}

Others Operations….

Kleene Closure: $L^* = \bigcup_{i=0}^{\infty} L^i = L^0 \cup L^1 \cup L^2 \cup \ldots$

Positive Closure: $L^+ = \bigcup_{i=0}^{\infty} L^i = L^1 \cup L^2 \cup \ldots$

A Language Hierarchy

Generator vs. Recognizer

Reminder…

Given a problem, we can develop a machine (automaton) that

- Generates a solutions

OR

- Recognizes a solution

Generator vs. Recognizer

Example

Given 2 integers A & B, determine the sum.

- Generator: Write a program to accept A & B as input then compute the sum A+B
- Recognizer: Write a program to accept A & B & C as input then determine if A+B = C

We usually write Generators! But when would an Recognizer be an appropriate solution?

Decision Problems

A **decision problem** is simply a problem for which the answer is yes or no (True or False).
A **decision procedure** answers a decision problem.

Example

- Given an integer *n*, does *n* have a pair of consecutive integers as factors?

The language recognition problem: Given a language *L* and a string *w*, is *w* in *L*?

Our focus

Turning Problems into Decision Problems

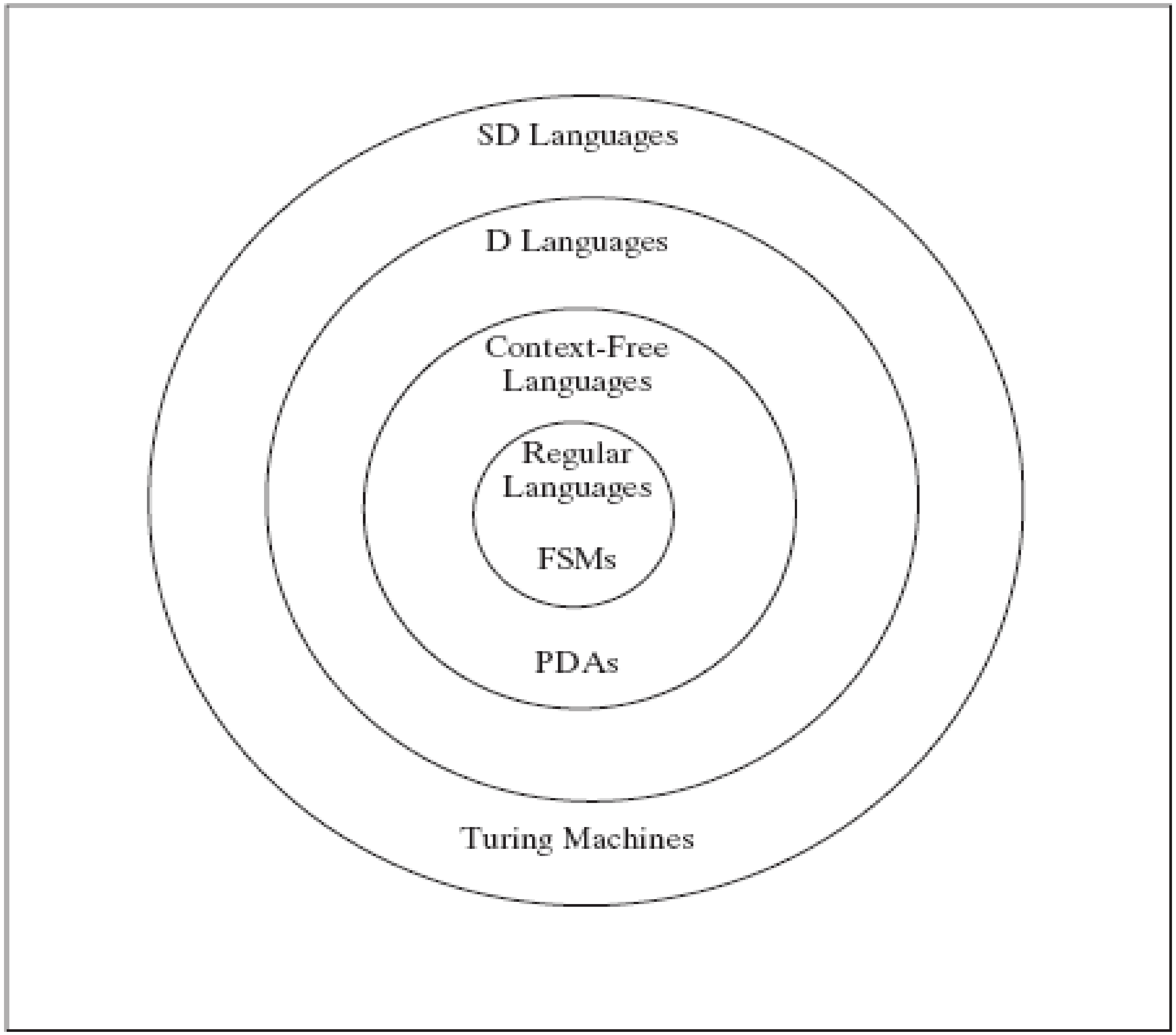
Casting multiplication as decision:

- Problem: Given two nonnegative integers, compute the product.
- Encoding : Transform computing into verification.
- The language:

$L = \{w$ of the form: $\mathit{integer}_1 > \mathtt{x} < \mathit{integer}_2 > = < \mathit{integer}_3 >$, where: $< \mathit{integer}_n >$ is any well formed integer, and $\mathit{integer}_3 = \mathit{integer}_1 * \mathit{integer}_2 \}$

```
12x9=108
12=12
12x8=108
```

A Hierarchy of Languages



D=decidable
SD = Semidecidable

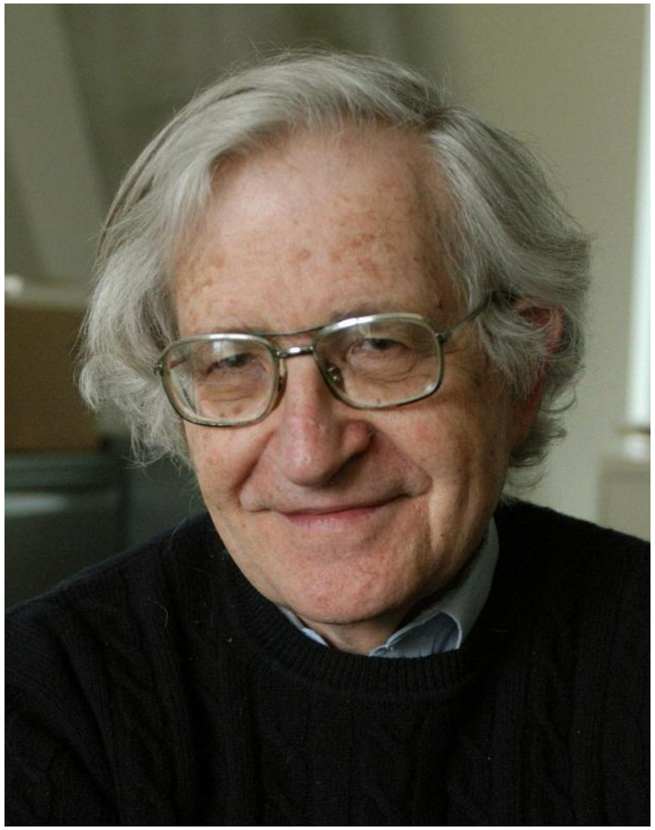
Chomsky Hierarchy of Languages

Languages from "simplest" to "complex"

Each is a subset of the ones below

- Regular
- Context Free
- Context Sensitive
- Recursively Enumerable

Can be defined by the type of Machine that will recognize it.

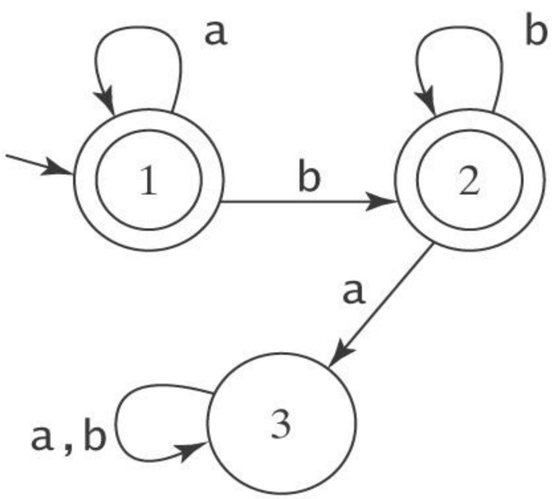


Noam Chomsky

Regular Languages

A Regular Language is one that can be recognized by a Finite State Machine.

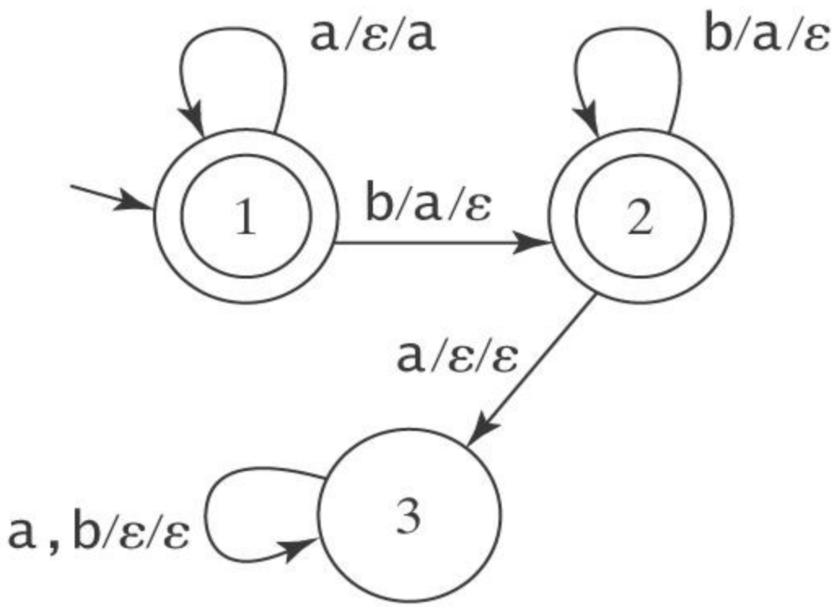
An FSM to accept `a*b*`:



Context Free Language

A Context Free Language is one that can be recognized by a Push Down Automata.

A PDA to accept $A^nB^n = \{a^nb^n : n \geq 0\}$



Decidable & Semidecidable Languages

A Decidable Language is one that is recognized by a Turing Machine which halts on all input strings.

A Semidecidable Language is one that is recognized by a Turing Machine which halts on all input strings which are in the language, but may loop infinitely on some strings which are not in the language.

5. FINITE STATE MACHINE(FSM)

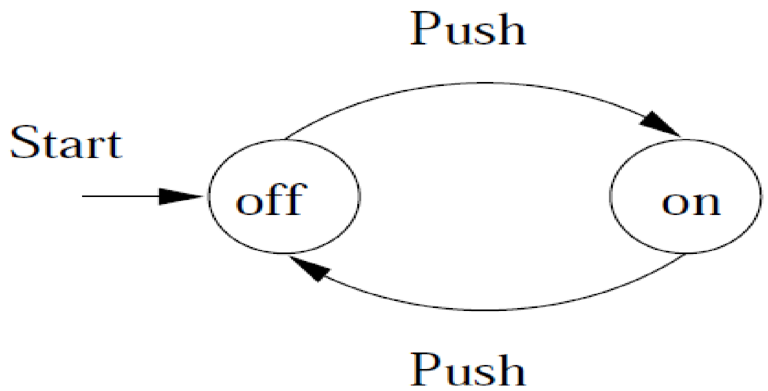
- The simplest and most efficient computational device that we will consider is the Finite State Machine (FSM).

- A Finite State Machine (or FSM) is a computational device whose input is a string and whose output is one of two values that we can call *Accept and Reject.*

(FSMs are also sometimes called finite state automata or FSAs.)

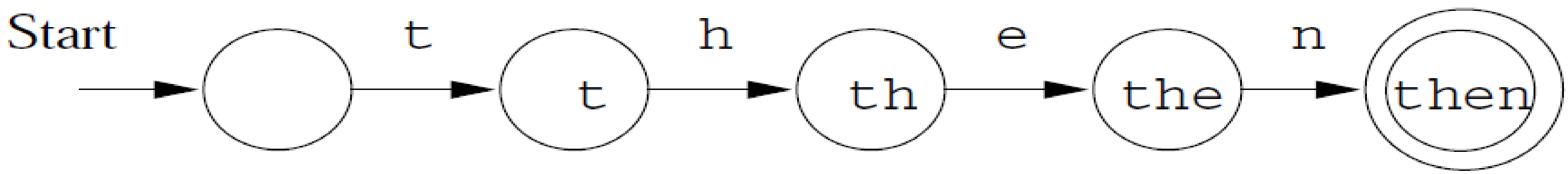
If M is an FSM, the input string is fed to M one character at a time, left to right. Each time it receives a character, M considers its current state and the new character and chooses a next state. One or more of M's states may be marked as accepting states. If M runs out of input and is in an accepting state, it accepts. If, however, M runs out of input and is not in an accepting state, it rejects. The number of steps that M executes on input w is exactly equal to $|w|$, so M always halts and either accepts or rejects.

- Example: Finite Automaton modelling an on/off switch



Model of Computation:
A program

- Example: Finite Automaton recognizing the string `then`



Deterministic FSM(DFSM)

Definition 2.2.1 A *finite automaton* is a 5-tuple $M = (Q, \Sigma, \delta, q, F)$, where

1. Q is a finite set, whose elements are called *states*,
2. Σ is a finite set, called the *alphabet*; the elements of Σ are called *symbols*,
3. $\delta : Q \times \Sigma \to Q$ is a function, called the *transition function*,
4. q is an element of Q; it is called the *start state*,
5. F is a subset of Q; the elements of F are called *accept states*.

Language Accepted by FSM:

Definition 2.2.3 Let $M = (Q, \Sigma, \delta, q, F)$ be a finite automaton. *guage* $L(M)$ *accepted* by M is defined to be the set of all string accepted by M:

$$L(M) = \{w : \ w \text{ is a string over } \Sigma \text{ and } M \text{ accepts } w\}.$$

Definition 2.2.4 A language A is called *regular*, if there exists a finite automaton M such that $A = L(M)$.

Designing of DFSM (Pattern Based Problems)

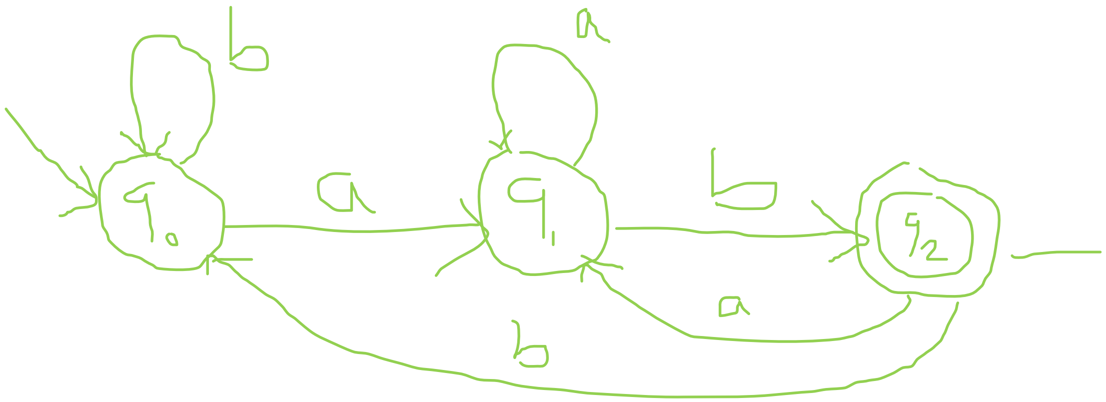
Design DFSM to:

1. Accept all the strings of a’s and b’s
2. Accept all the strings a’s and b’s begin with b
3. Accepts all the strings of a’s and b’s ending with ab
4. Recognize all the strings of 0’s and 1’s having substring 011
5. L = { w : #a(w) $\geq$ 1 , Σ = { a,b} }
6. Obtain a DFSM to accept to accept strings of a’s and b’s starting with the sub-string ab.
7. Obtain a DFSM to accept all strings of a’s and b’s ending with the string abb.
8. Obtain a DFSM to accept all strings of a’s and b’s which do not ending with the string abb

3. Accepts all the strings of a's and b's ending with ab

ab baaab, $\Sigma = \{a, b\}$

$w = abbbab$



$w = bbaba \notin L$

TD

$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{b} q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2$

(Accept)

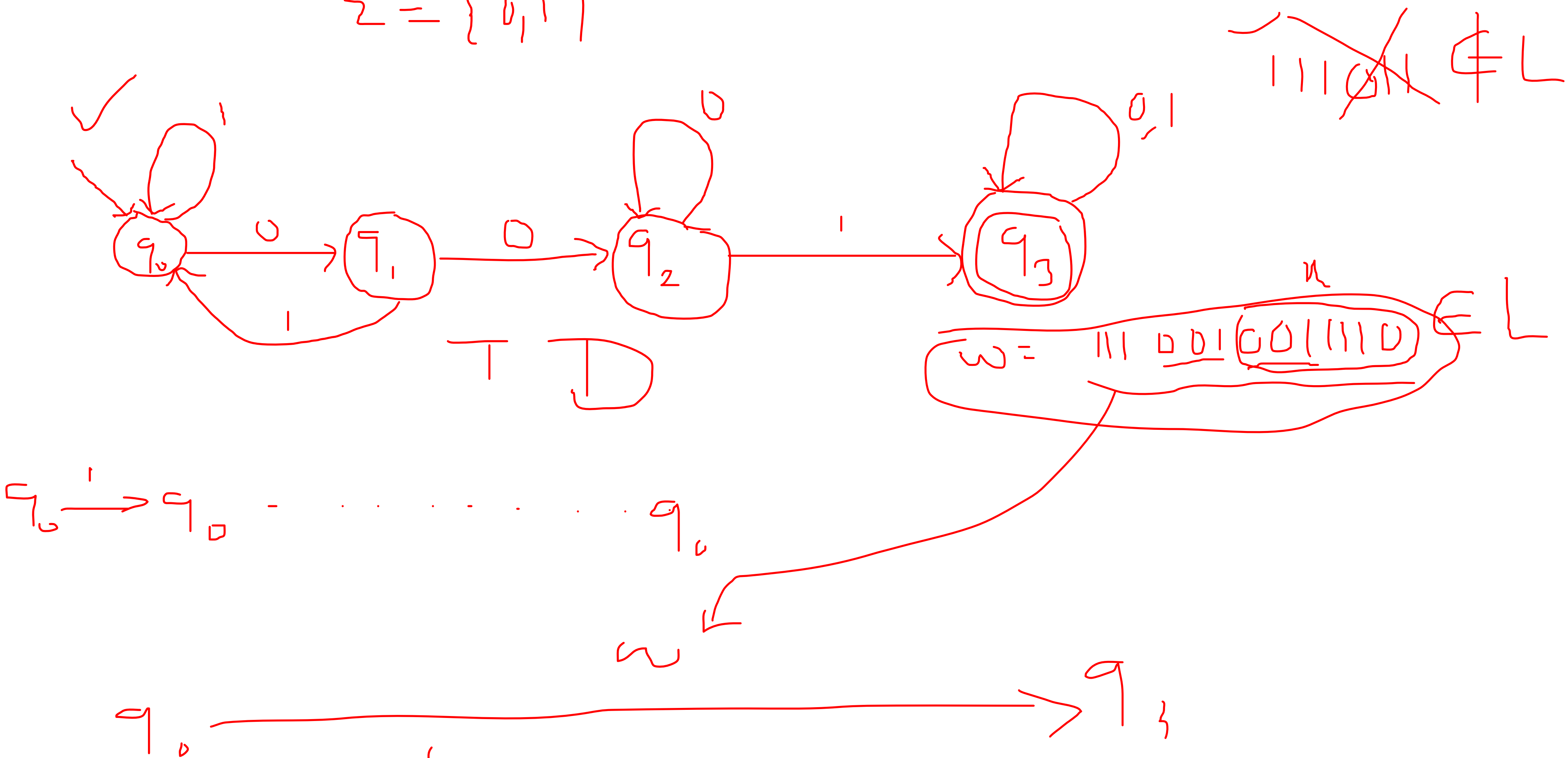
δ	a	b
$\rightarrow q_0$	q_1	q_0
q_1	q_1	q_2
q_2^*	q_1	q_0

TT

4. Recognize all the strings of 0's and 1's having string 011

i.e. L = { w ∈ {0,1}* | each w contains a sub-string 001}

$\Sigma = \{0, 1\}$



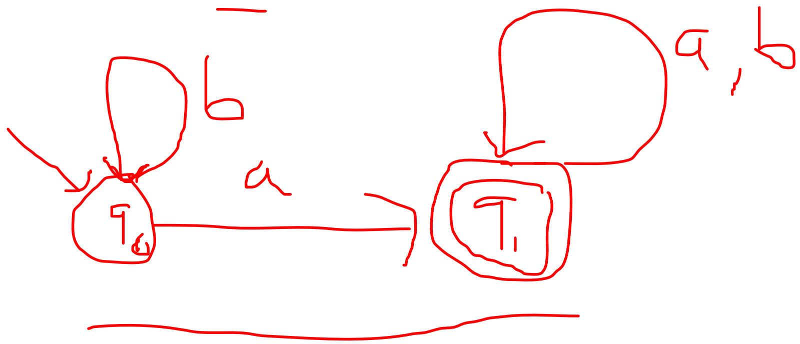
δ	0	1
→ q_0	q_1	q_0
q_1		
q_2		
q_3		

T T

5. L = { w : #a(w) ≥ 1 , Σ = { a,b} }

6. Obtain a DFSM to accept to acc~~ept strings~~ of a's and b's starting with a string ab

5)



DFSM

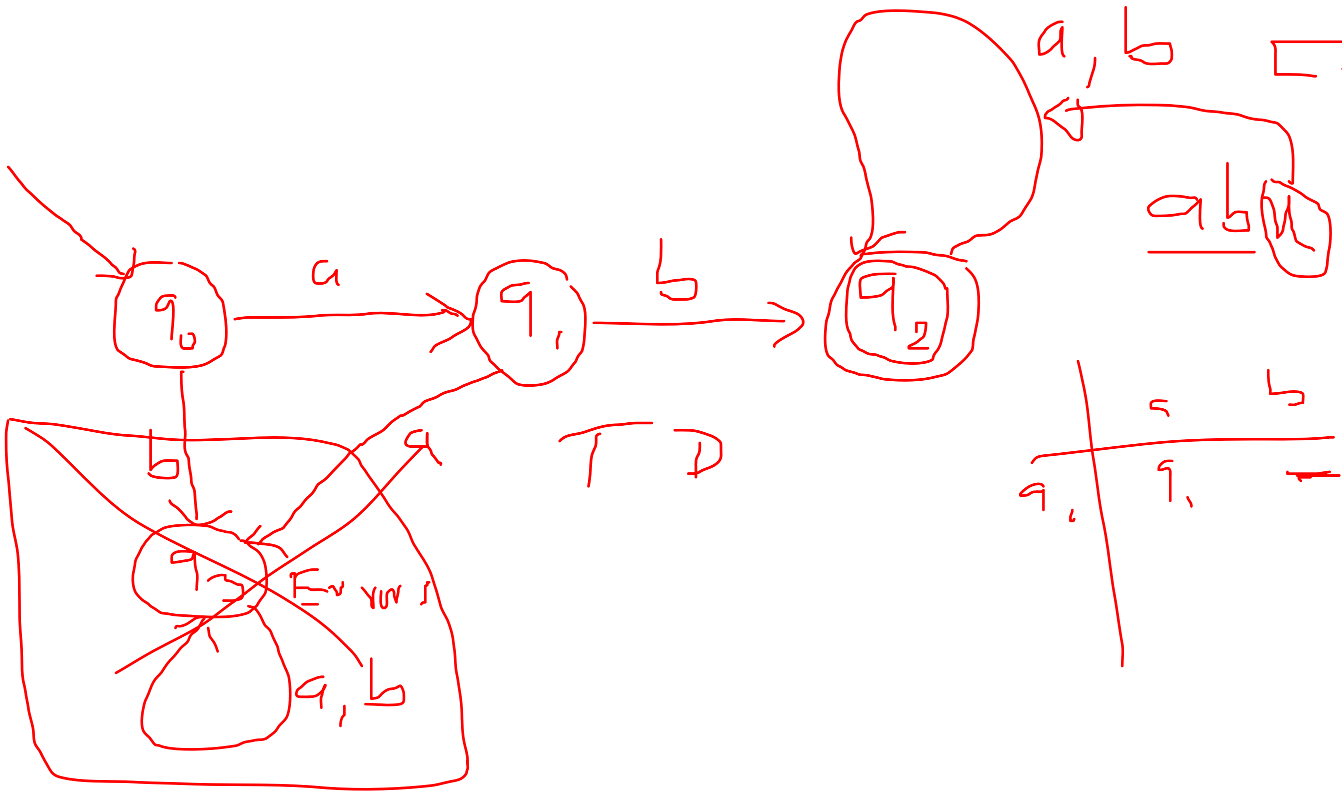
L = { axa | x ∈ {a,b}* }

L = { axb | x ...

L = { bxb | ...

$\bar{L}$ = L = { bxa

6)

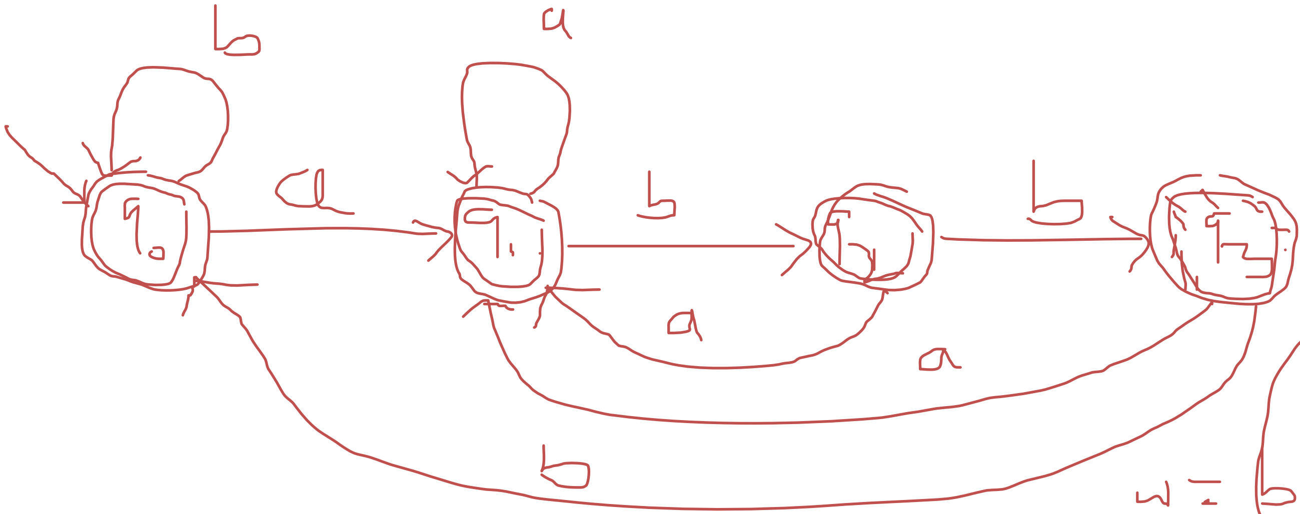


that do not end

7. Obtain a DFSM to accept all strings of a's and b's ~~ending with~~ the substring abb

with

$\Sigma = \{a, b\}$



TD

~~abba~~

w = bb abb bbb abab abb

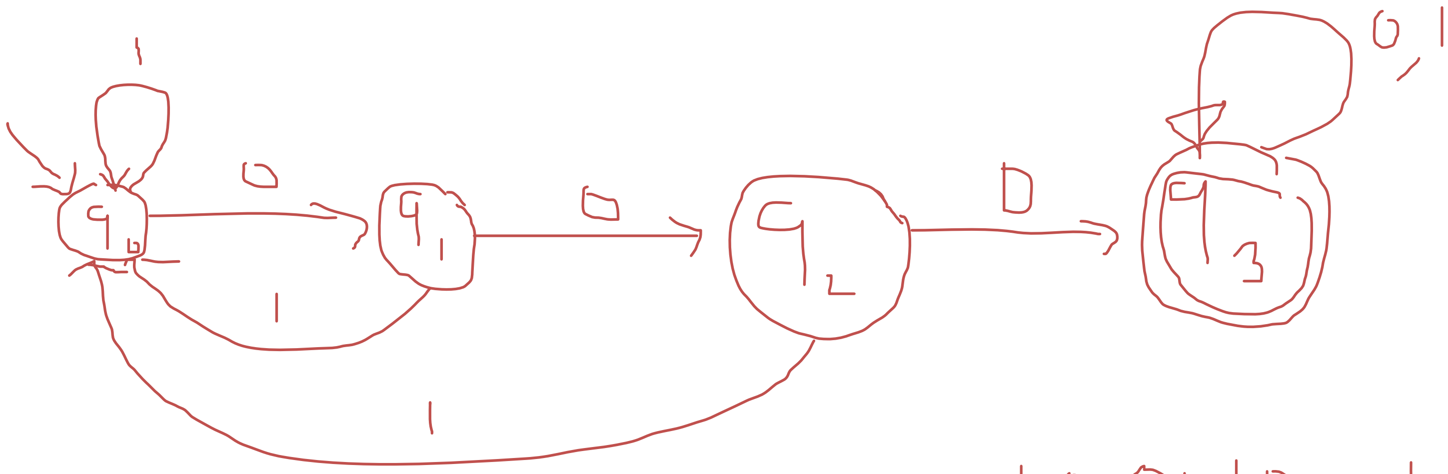
δ	a	b
→ q_0*	q_1	q_0
q_1*		
q_2*		
q_3		

TT

8. Obtain a DFSM to accept all strings of a's and b's which do not ending with the substring abb

9. Obtain DFSM to accept strings of 0's and 1's having three consecutive 0's

000 — $\Sigma = \{0, 1\}$



100010001 $\in L$

10. Construct a DFSM for L= {wbab | w ∈ {a,b}* } and show the moves made machine on: ababab

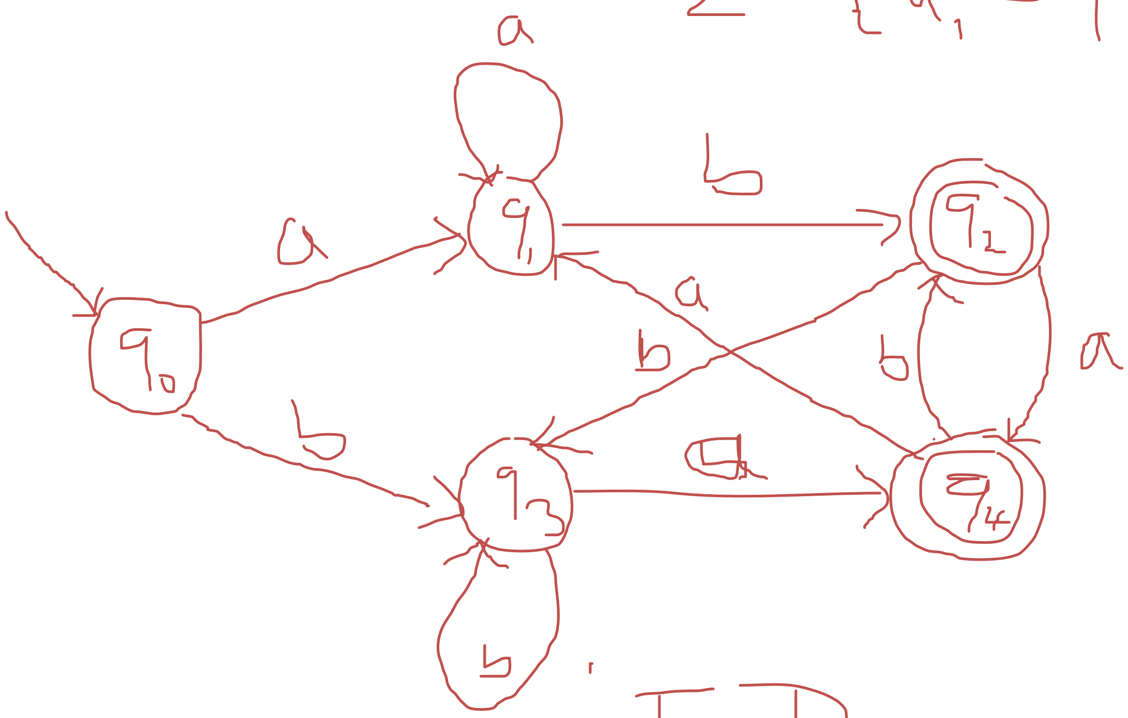
wbab

abb

Hw

11. Obtain a DFSM to accept strings of a's and b's ending with ab or ba

$\Sigma = \{a, b\}$



STD

δ	a	b
$\rightarrow q_0$	q_1	q_3
q_1	q_1	q_2
q_2^*	q_4	q_3
q_3	q_4	q_3
q_4^*	q_1	q_2

TT

MOVn

$w = aabbbab$

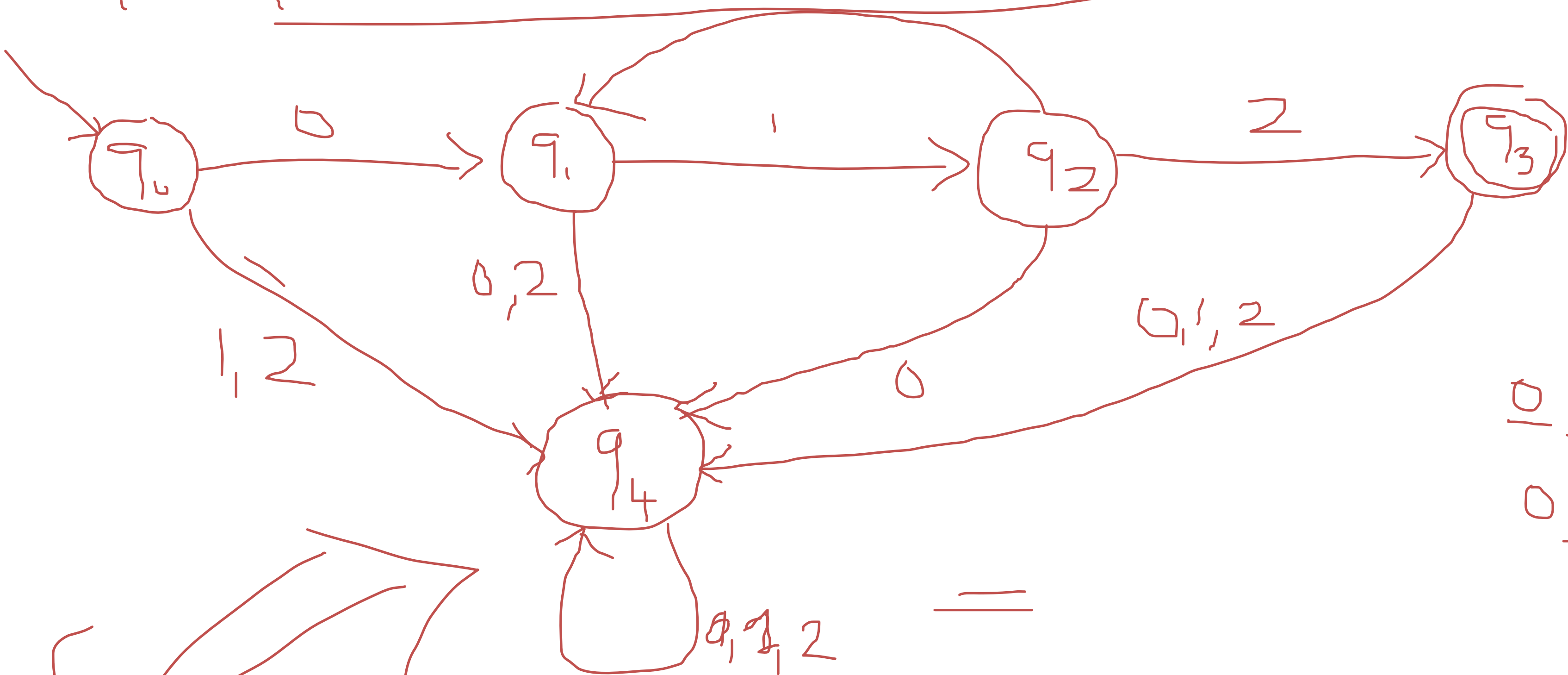
$$q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{b} q_3 \xrightarrow{b} q_3 \xrightarrow{a} q_4 \xrightarrow{b} q_2$$

Accl

*12. Construct a DFSM to accepts all the strings of 0's, 1's and 2's beginning with '0' followed by odd number of 1's & ending with '2'

0111 2   0112 $\Sigma = \{0, 1, 2\}$

$M = (Q, \{0,1,2\}, \delta, q_0, \{q_3\})$



0112 ∈ L

0112 ∉ L

δ q_0 $Q = \{q_0, q_1, \ldots q_4\}$

$F = \{q_3\}$ $\Sigma = \{0, 1, 2\}$

13. Obtain a DFSM to accept strings of a's and b's with at most two consecutive b's

$\Sigma = \{a, b\}$

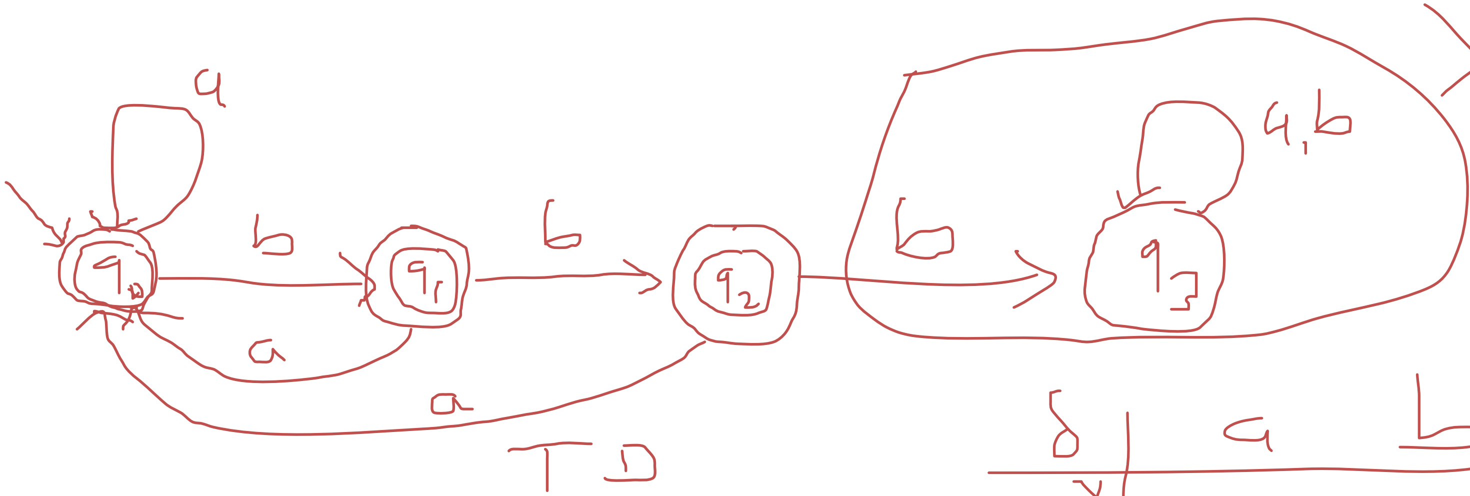
abbabbbaaa

abaa, aaa

ϵ,

abb $\in L$

ababbb



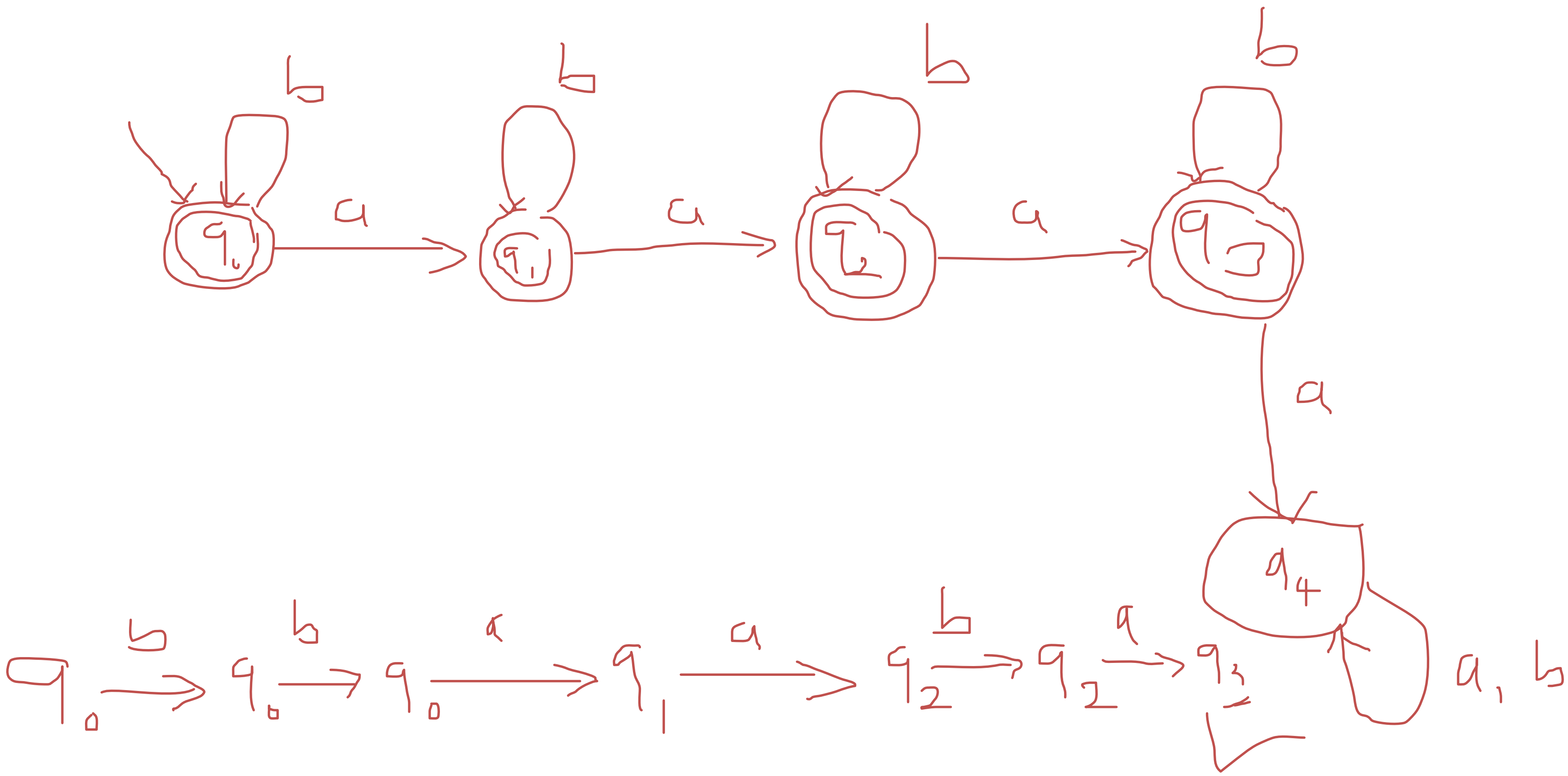
δ	a	b
$\rightarrow q_0^*$	q_0	q_1
q_1^*	q_0	q_2
q_2^*	q_0	q_3
q_3	q_3	q_3

14. Design a DFSM for the L = { w: $\#_a(w) \leq 3$, w ∈ {a , b}* }

4

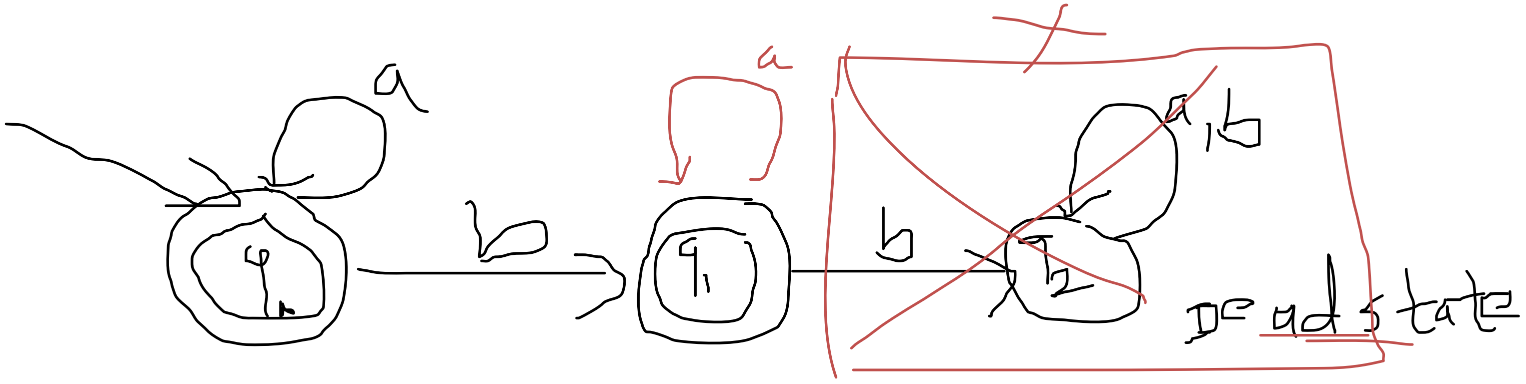
ε, 0, 1, 2, 3

w = bbaabaa



15.Construct DFSM for L = { w ∈ {a,b} | w contains no more than one b }

$\#_b(w) \le 1$



w = aabaa ∈ L

abbb ∉ L

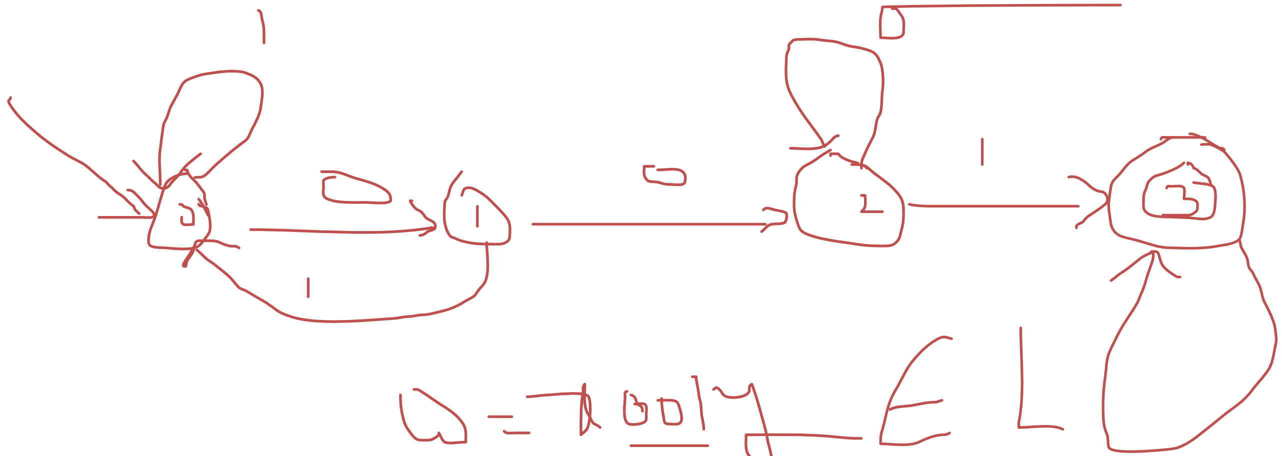
δ	a	b

δ	a	b
→ q_0*	q_0	q_1
q_1*	q_1	—

implicit

* Obtain a DFSM for L = w ∈ {0,1}* | w has 001 as a substring }

ex 001 111 100 101



w = 1001y ∈ L

1110 reject

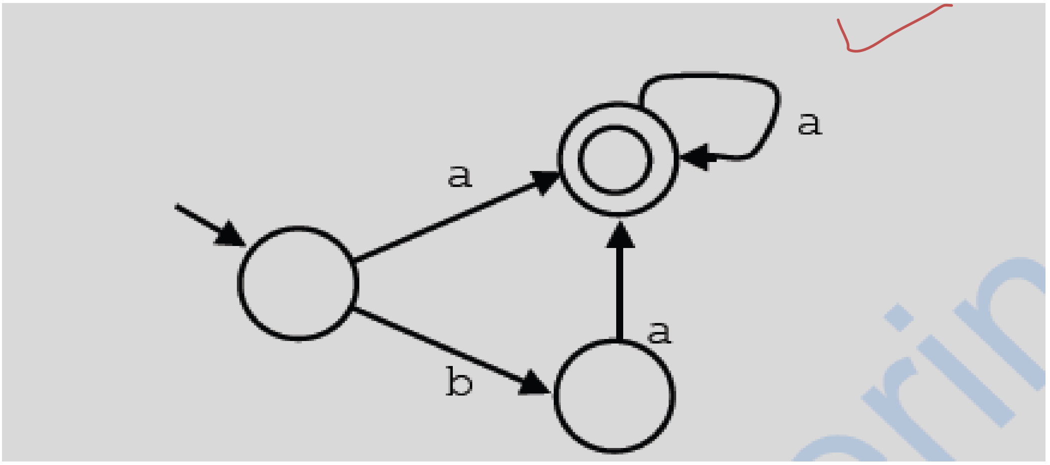
1001 00111

Simulation

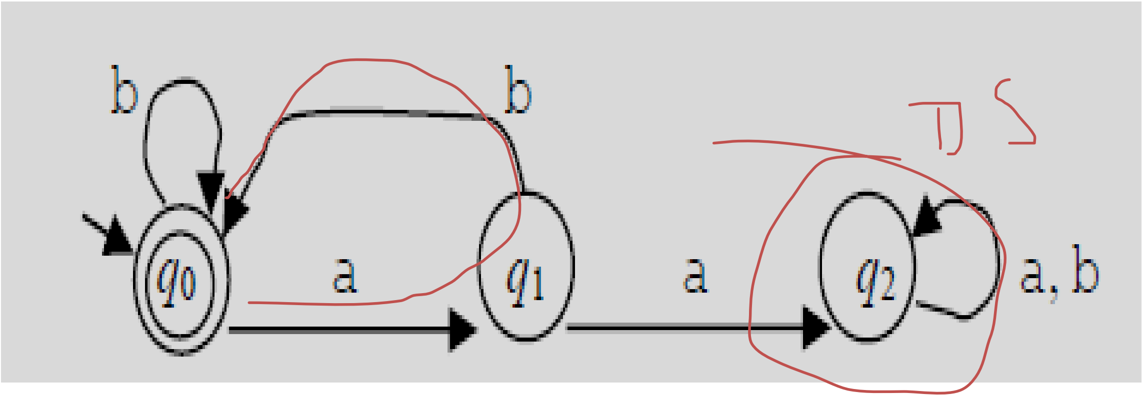
L= $\{w \in \{a, b\}^* : w$ has neither ab nor bb as a substring$\}$.

aa or ba

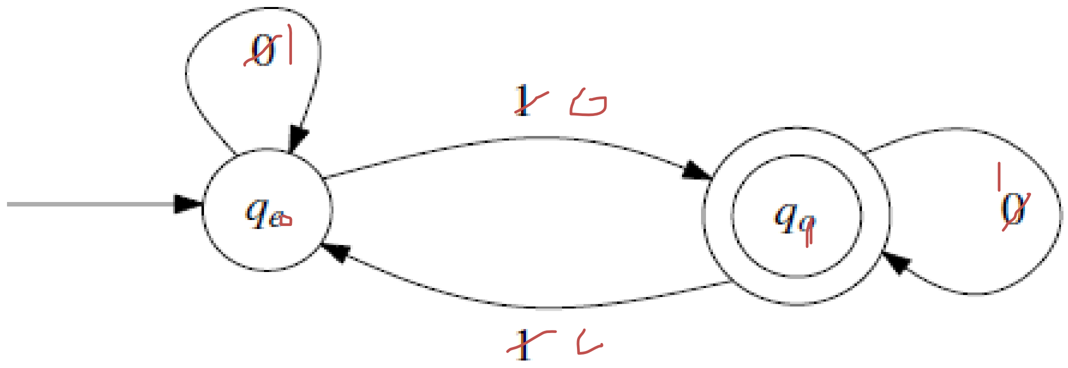
aaa



$L = \{w \in \{a, b\}^* :$ every a is immediately followed by a b$\}$

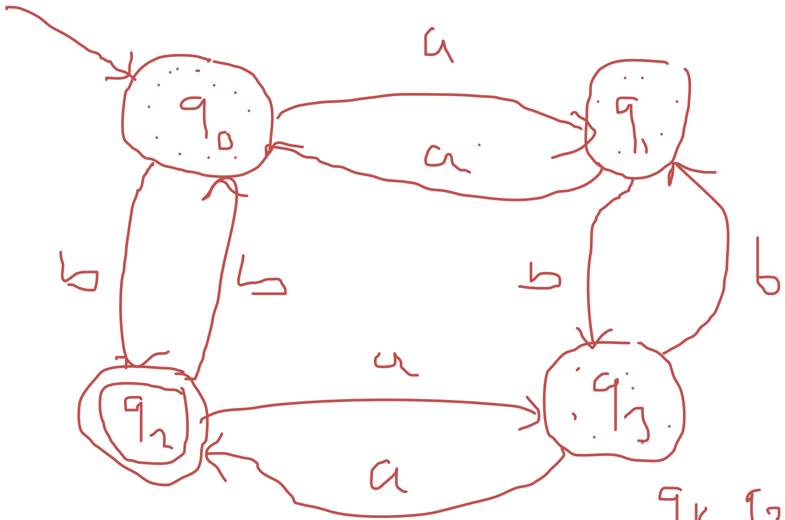


$A = \{w : \ w \text{ is a binary string containing an odd number of 1s}\}.$



Moves made by DFSM: on: 010110

- *Design a DFSM for L = {w ∈ {a,b}* | w contains even number of a's & odd number of b's }
- *Design a DFSM for L = {w ∈ {a,b}* | w contains even number of a's & even number of b's }



even a's even b's
odd a's even b's
even a's odd b's
odd a's odd b's

w = aabbb ∉ L, w = ababb ∈ L

*Draw a DFSM to accept decimal strings which are divisible by 3

< Rem = 0

≠ 4

12 % 3 = 0

$\Sigma = \{0,1,2,3,4,5,6,7,8,9\}$

11 % 3 = 2

0, 1, 2

10 % 3 = 1

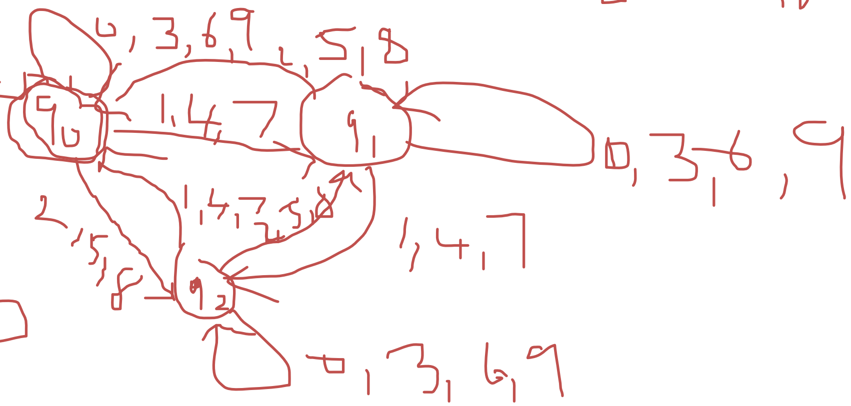
q_0, q_1, q_2

27 % 3 = 2

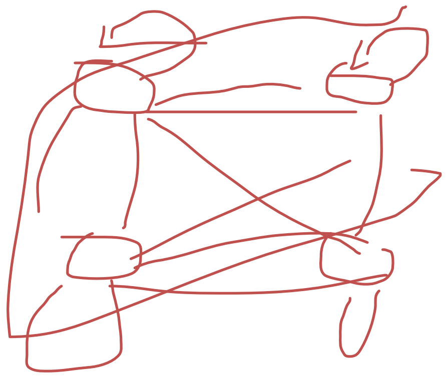
w = 8234 % 3 = 0

δ	0	1	2	3	4	5	6	7	8	9
→ q_0*	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0
q_1	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1
q_2	q_2	q_0	q_1	q_2	q_0	q_1	q_2	q_0	q_1	q_2

TT



TD



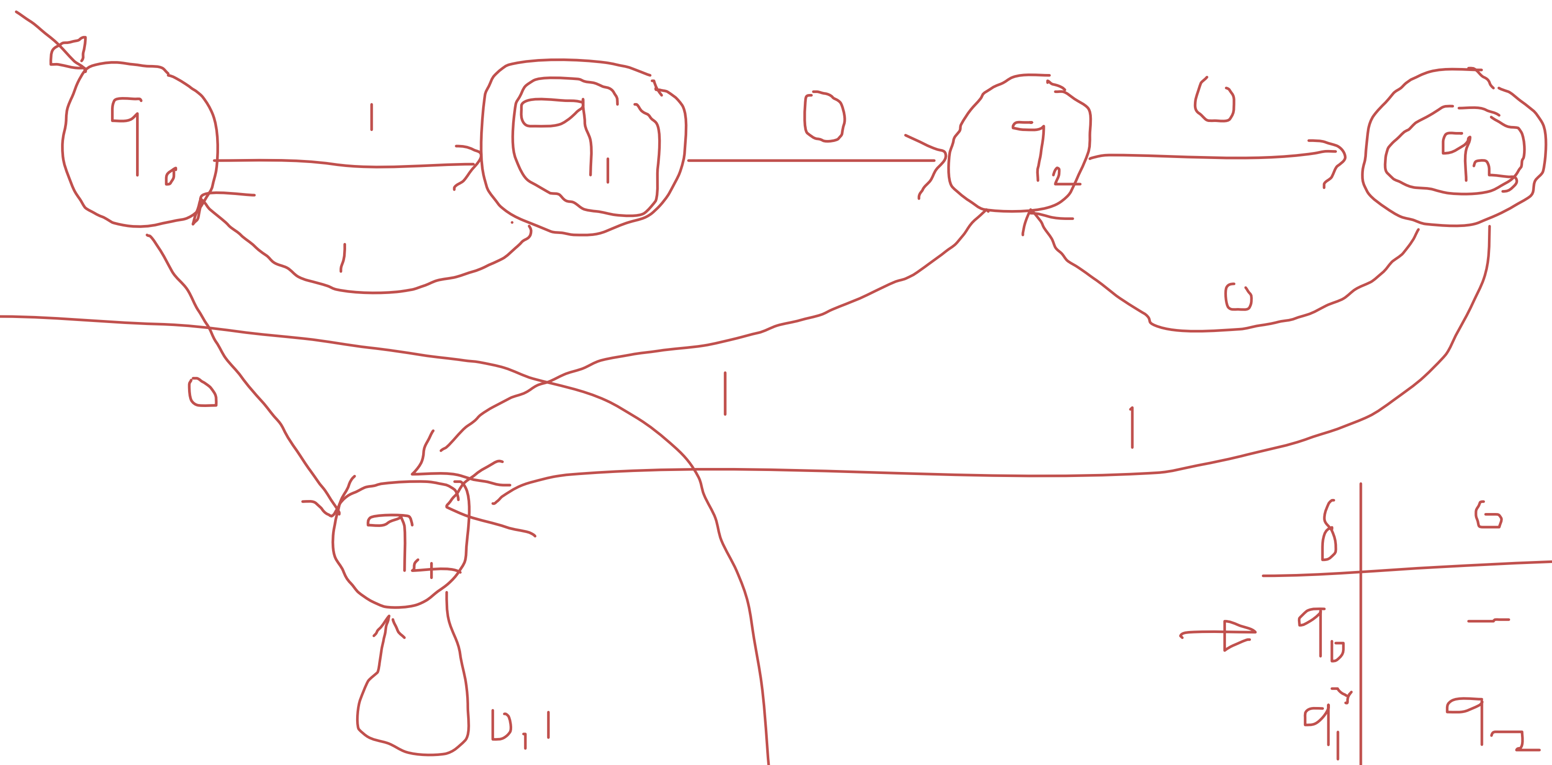
Draw a DFSM to accept the Language L = { w : w has odd number of 1's and followed by even number of 0's }

1

1100

10000

101000 ∉ L



δ	0	1
→ q_0	–	q_1
q_1^*	q_2	q_0
q_2	q_3	–
q_3^*	q_2	–

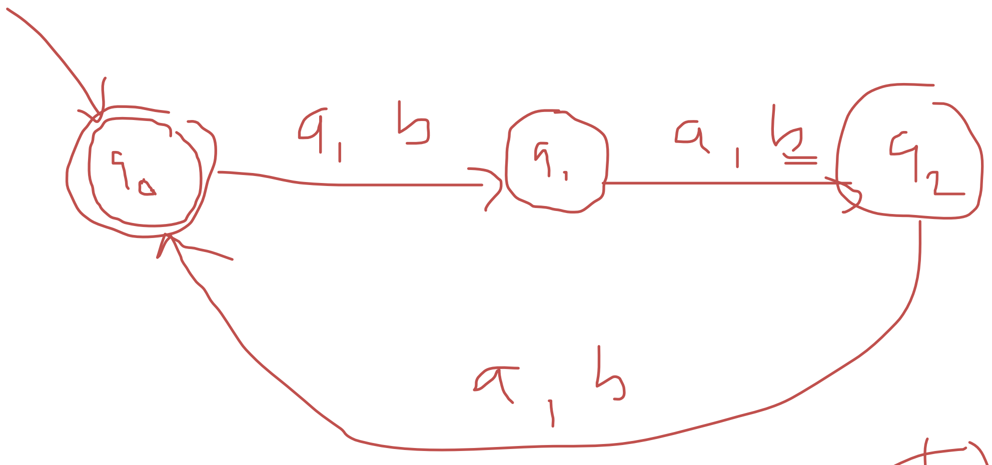
- Obtain a DFSM to accept the Language L={ w : |w| mod 3 = 0} on Σ ={a,b}}
- Obtain a DFSM to accept the Language L={ w : |w| mod 3 ≠ 0} on Σ ={a,b}}

3, 6, 9, 12

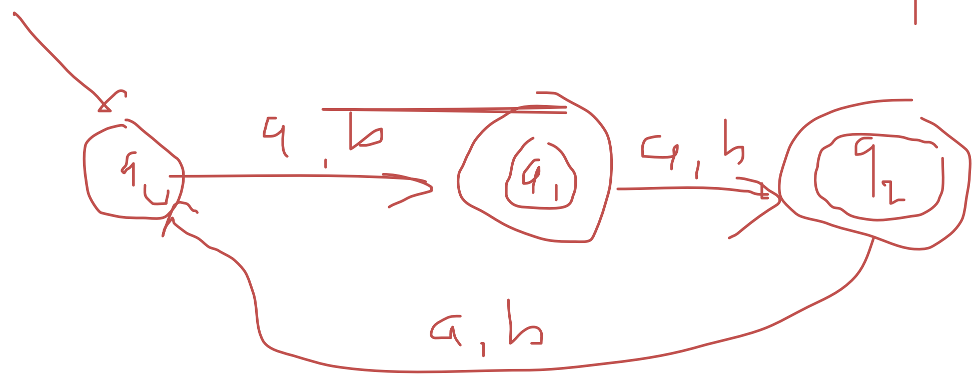
aba ∈ L

aaaabb ∈ L

ab ∉ L



|w| mod 3 = 0



DFSM

Non-Deterministic FSM(NDFSM)

Definition:

$M = (Q, \Sigma, \delta, q_0, F)$, where:

$Q = \{1, 2\}$ $\quad 2^Q = \{\varphi, \{1\}, \{2\}, \{1, 2\}\}$

Q: is a finite set of states

Σ : is an alphabet

$q_0 \in Q$: is the initial state

$F \subseteq Q$: is the set of accepting states, and

δ: is the transition **relation**. $Q\times(\Sigma \cup \{\varepsilon\}) \rightarrow 2^Q$

$\delta(q, \varepsilon) \rightarrow$

$Q \times \Sigma \rightarrow Q$

Accepting by an NDFSM

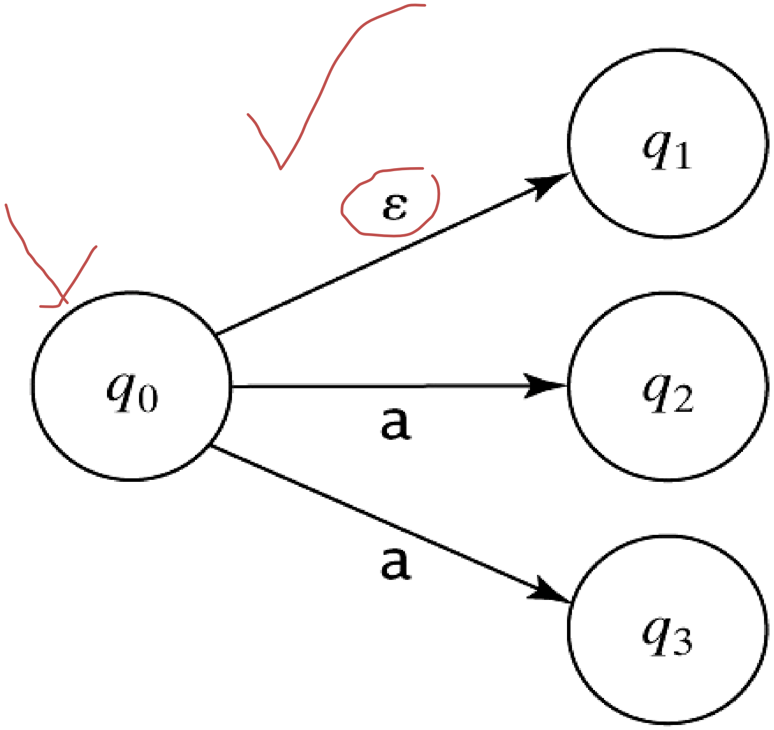
M accepts a string w iff there exists some path along which w drives M to some element of A. F

The language accepted by M, denoted $L(M)$, is the set of all strings accepted by M.

$L(M) = \{w_1, w_2, \ldots\}$

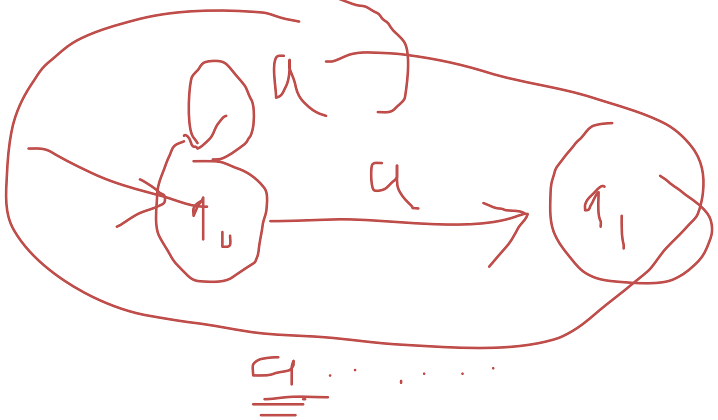
Sources of Non-determinism

What differ from determinism?



NDFSM

NDFSM   ε-NDFSM



Why NDFSM? ≡ DFSM

Impl — d

- Very easy to construct
- Has the ability to guess something about its input
- is more powerful than DFSM
- Has power to be in several states at a time

1. Construct a NDFSM to accept L = { w | w ends with ab, Σ = { a,b} }
Draw the Transition table(TT) and show the moves made by m/c on: baab

$Q = \{q_0, q_1, q_2\}$

(x)ab

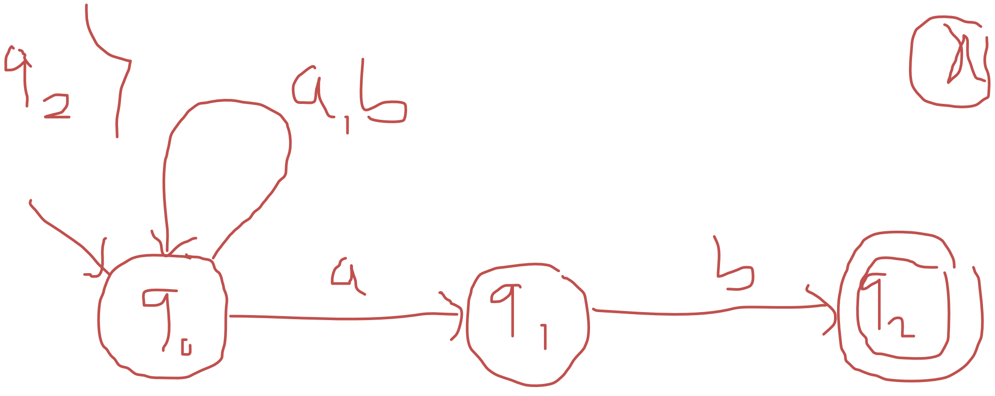
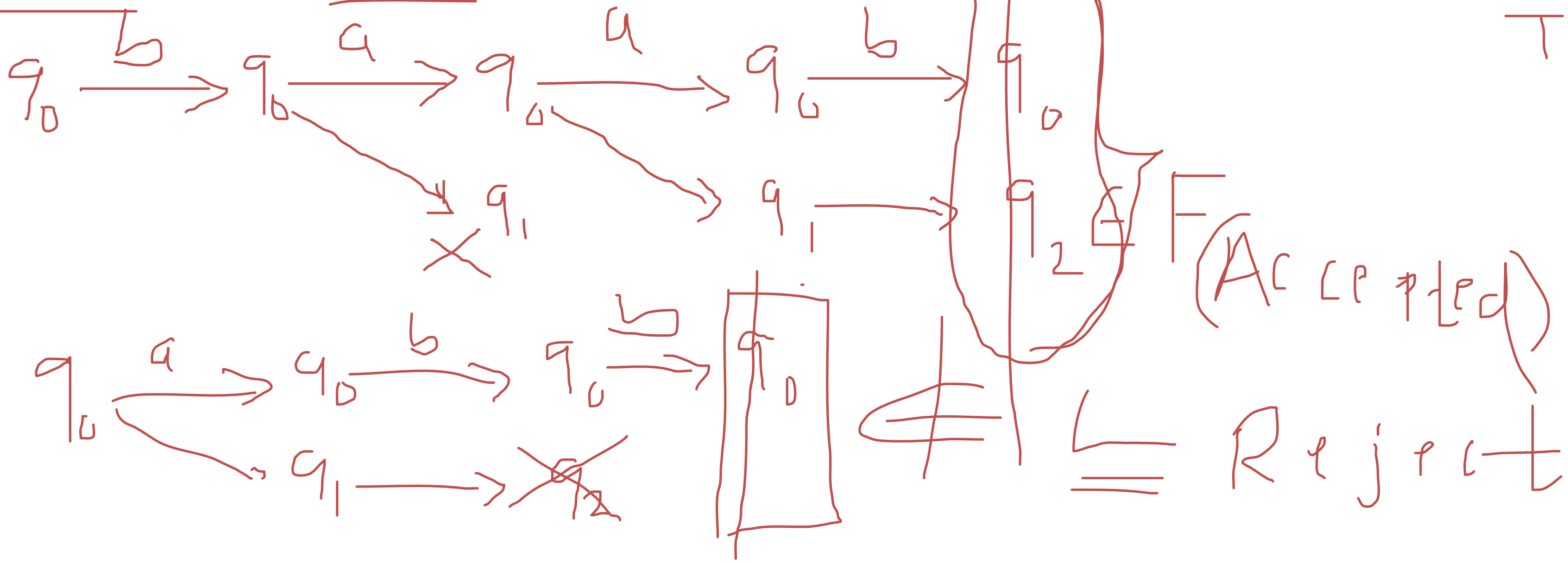


abb ∉ L

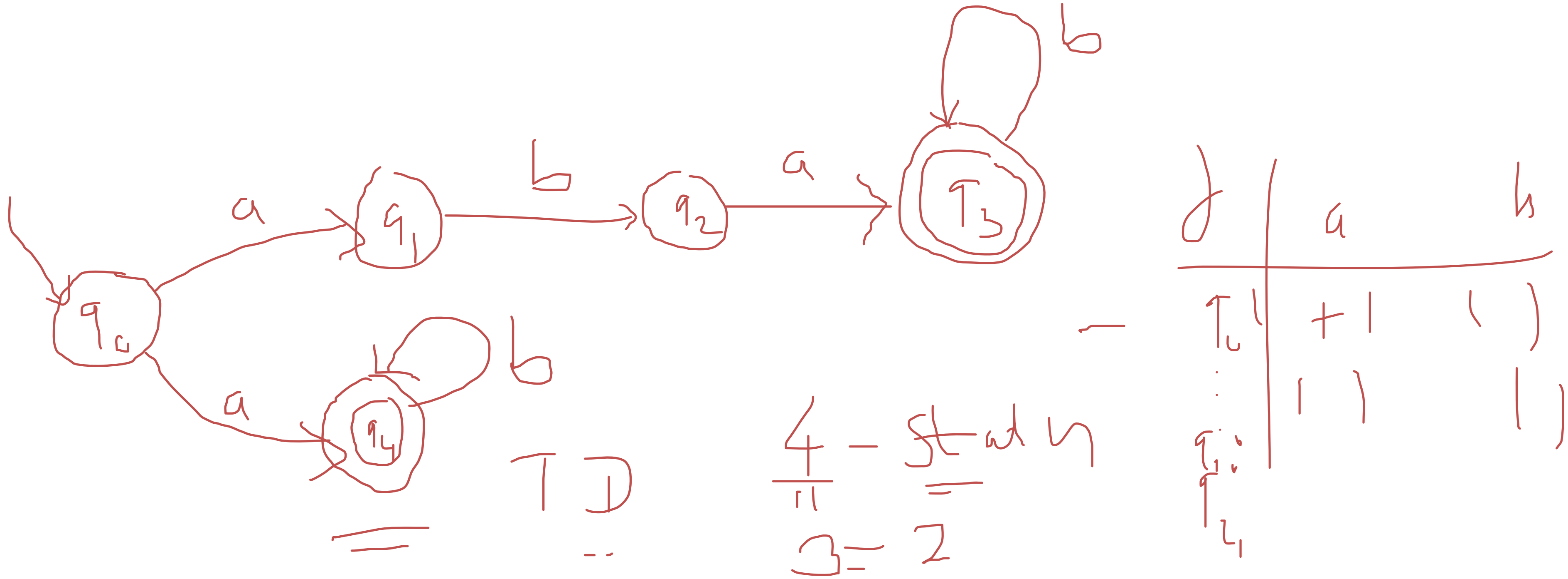
δ	a	b
→ q_0	$\{q_0, q_1\}$	$\{q_0\}$
q_1	ϕ	$\{q_2\}$
* q_2	ϕ	ϕ

TT

Moves by NDFSM (Simulation)



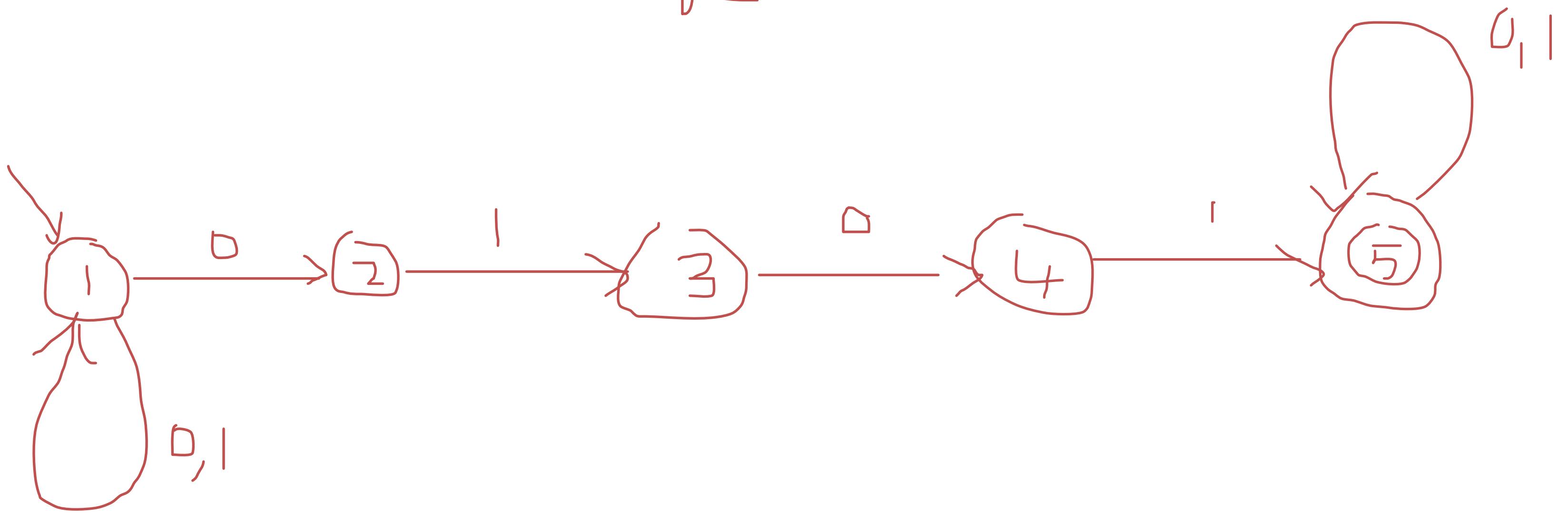
2. Obtain an NDFSM to accept L ={ w | w $\in$ abab^n or ab^n, where n $\geq$ 0}



3.Design an NDFSM for L = { w | w contains the substring 0101 , Σ = { 0,1} }

$x\ \underline{0101}\ y$

$w = \underbrace{101}_{x}\ \underline{0101}\ \underbrace{11}_{y}$

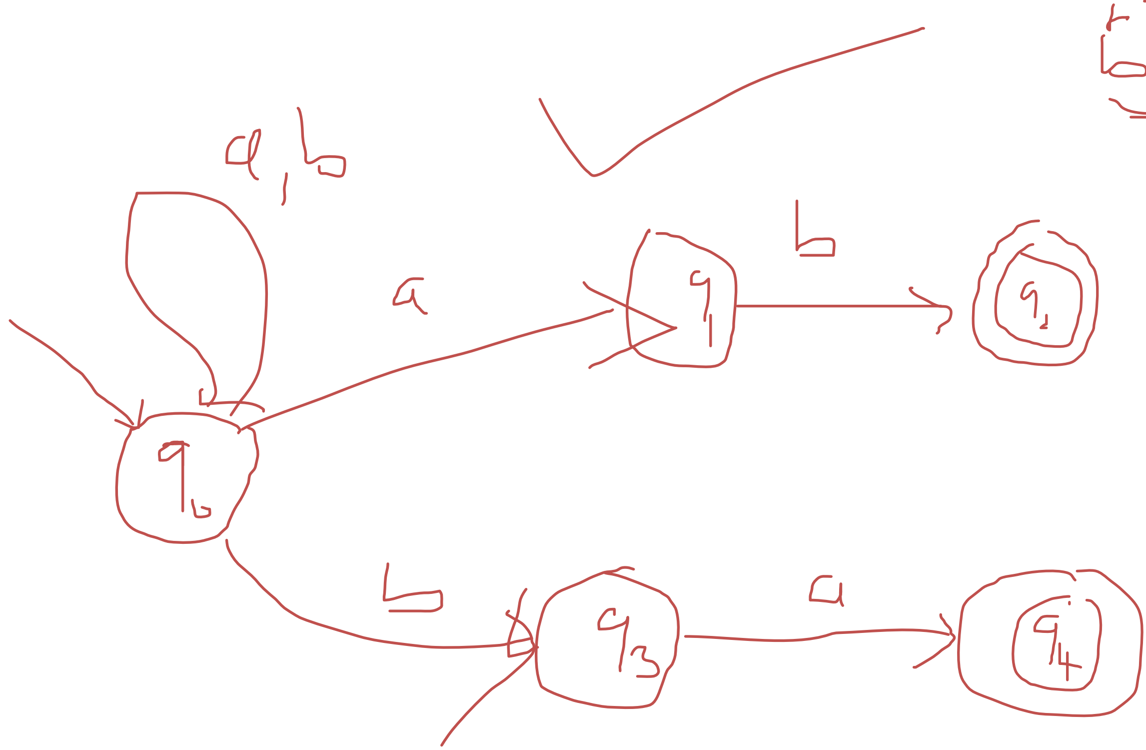


4. Write an NDFSM to accept string of a's and b's ending with ab or ba

bbabbab ab

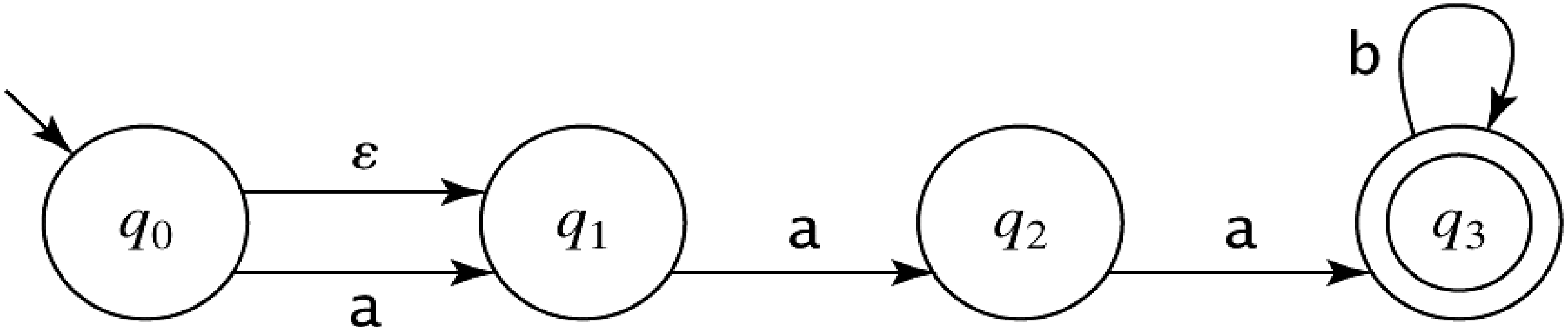
or

aba



δ	a	b
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_3\}$
q_1	—	
q_2^*	—	
q_3	—	
q_4^*	—	

5. $L = \{w \in \{a, b\}^* : w$ is made up of an optional a followed by aa followed by zero or more b's$\}$.

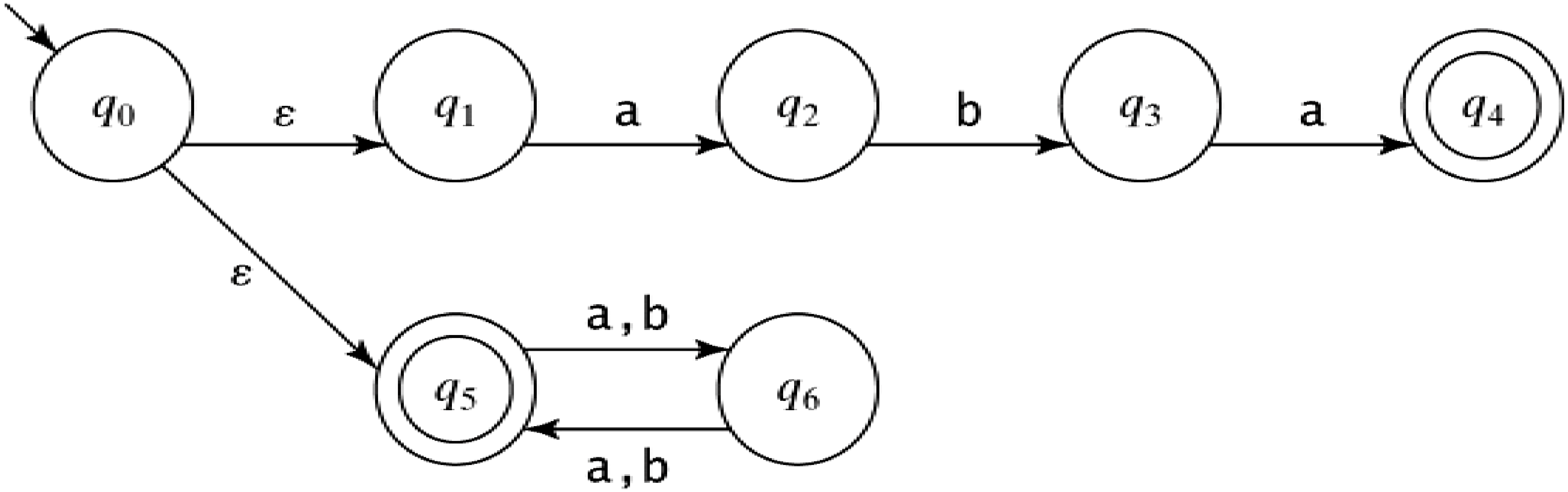


$M = (Q, \Sigma, \delta, q_0, F) = (\{q_0, q_1, q_2, q_3\}, \{a, b\}, \delta, q_0, \{q_3\})$

6. Design $\in$-NDFSM for L = { w | w contains at least two 0's or exactly two 1's }

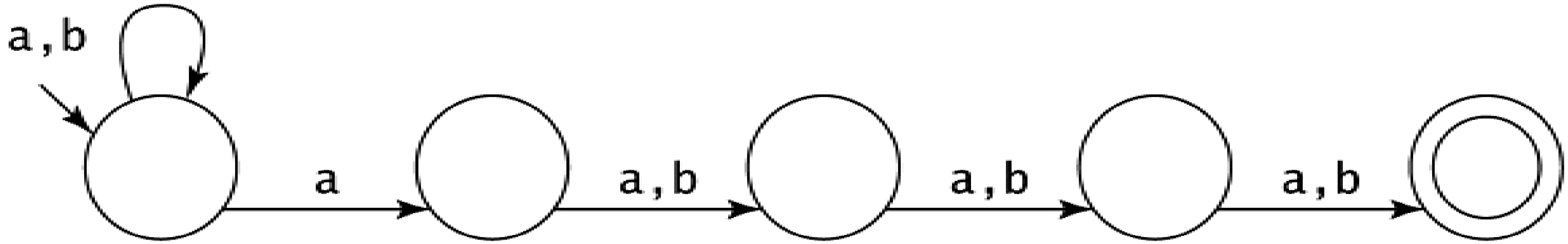
7. Design an $\in$-NDFSM to accept strings of a’s and b’s ending with ab or ba

8. $L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : w = \mathtt{aba}$ or $|w|$ is even$\}$.



Do you start to feel the power of Non-Determinism?

9. $L = \{w \in \{\texttt{a}, \texttt{b}\}^* :$ the fourth to the last character is $\texttt{a}\}$



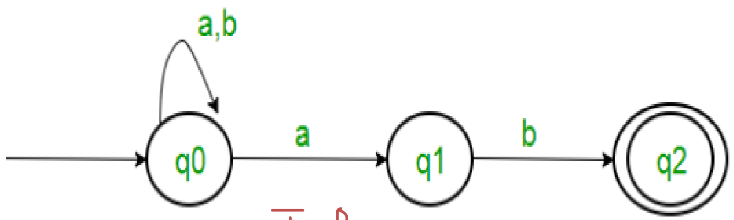
DFSM	NDFSM
$M = (Q, \Sigma, \delta, q_0, F)$ where Q is set of finite states Σ is set of input alphabets $\delta : Q \times \Sigma$ *to* Q q_0 is the initial state $F \subseteq Q$ is set of final states	definition of δ. Here, δ is defined as follows: $\delta : Q \times (\Sigma \cup \epsilon)$ to subset of 2^Q
2. There can be zero or one transition from a state on an input symbol	There can be zero, one or more transitions from a state on an input symbol
3. No $\epsilon -$ transitions exist i.e., there should not be any transition or a transition if exist it should be on an input symbol	$\epsilon -$ transitions can exist i. e., without any input there can be transition from one state to another state.
4. Difficult to construct	Easy to construct
5. Less powerful but easy to implement	More powerful than DFSM, but very difficult to implement

*Converting an NDFSM to DFSM using subset Construction Method

1) NDFSM → DFSM

2) ∈**-NDFSM** → DFSM

Example-1 (NDFSM to DFSM)



$\delta(q_0, a) \rightarrow \{q_0, q_1\}$

$\delta(q_0, b) \rightarrow \{q_0\}$

$\delta(\{q_0, q_1\}, a) = \delta(q_0, a) \cup \delta(q_1, a) = \{q_0, q_1\}$

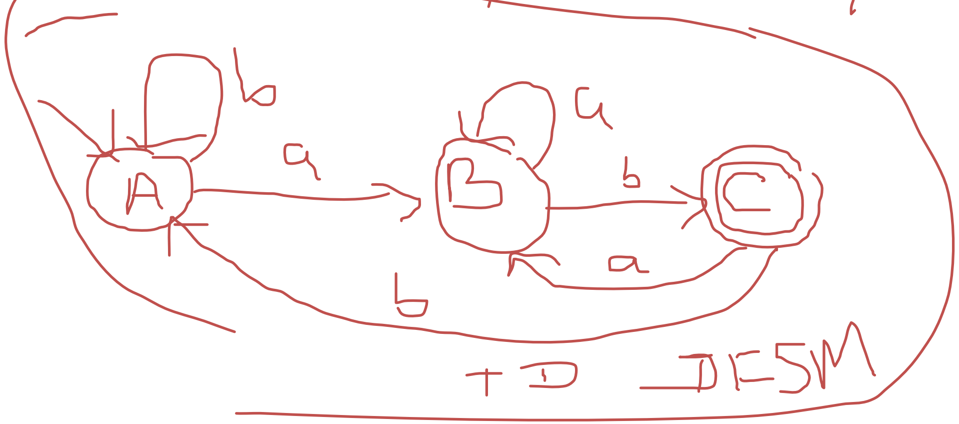
$\delta(\{q_0, q_1\}, b) = \delta(q_0, b) \cup \delta(q_1, b) = \{q_0, q_2\}$

$\delta(q_0, a) \cup \delta(q_2, a) = \{q_0, q_1\}$

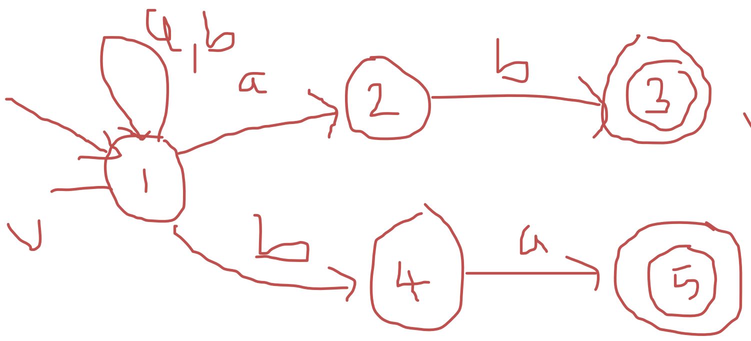
$\delta(q_0, b) \cup \delta(q_2, b) = \{q_0\}$

δ	a	b
$\rightarrow \{q_0\}$	$\{q_0, q_1\}$	$\{q_0\}$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_1\}$	$\{q_0\}$

δ	a	b
$\rightarrow$ A	B	A
B	B	C
C*	B	A



Example-2 : All strings of a's & b's ending ab or ba



NDFSM

$\delta(1,a) \rightarrow 1,2,\quad \delta(1,b) = 1,4$

$\delta(1,a) \cup \delta(2,a) = 1,2$

$\delta(1,b) \cup \delta(2,b) = 1,4,3$

$\delta(1,a) \cup \delta(4,a) \rightarrow 1,2,5$

$\delta(1,b) \cup \delta(4,b) \rightarrow 1,4$

$\delta(1,a) \cup \delta(3,a) \cup \delta(4,a) = 1,2,5$

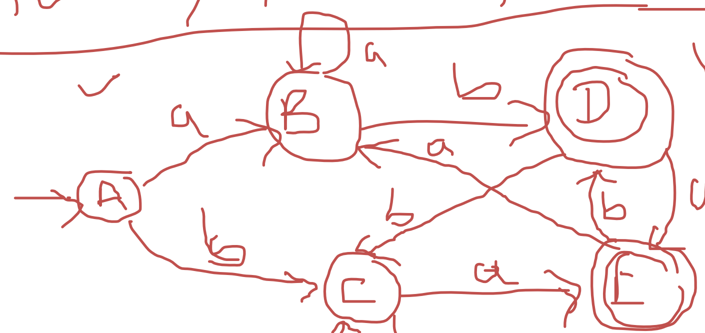
$\delta(1,b) \cup \delta(3,b) \cup \delta(4,b) = 1,4,$

$\delta(1,a) \cup \delta(2,a) \cup \delta(5,a) = 1,2$

$\delta(1,b) \cup \delta(2,b) \cup \delta(5,b) = 1,4,3 = 1,3,4$

δ	a	b
→ A {1}	1,2	1,4
B 1,2	1,2	1,4,3
C 1,4	1,2,5	1,4
D* 1,4,3	1,2,5	1,4
E* 1,2,5	1,2	1,3,4

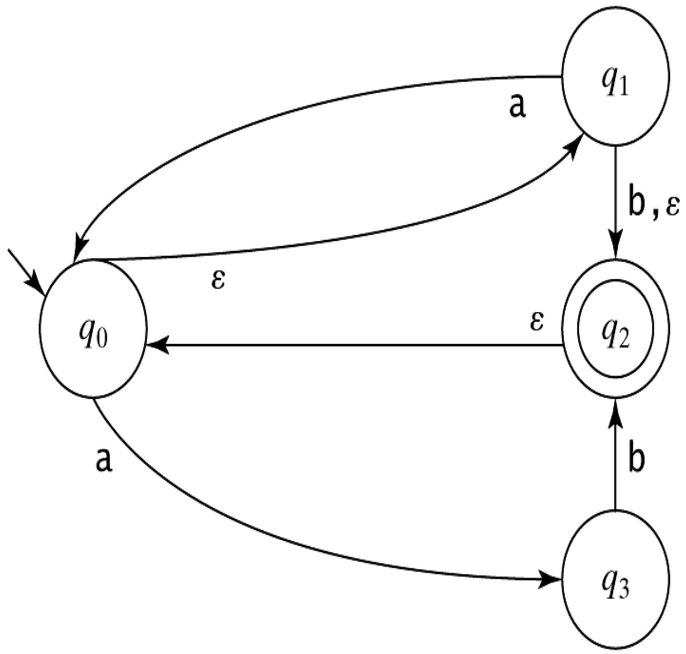
δ	a	b
→ A	B	C
B	B	D
C	E	C
D*	E	C
E*	B	D



DFSM

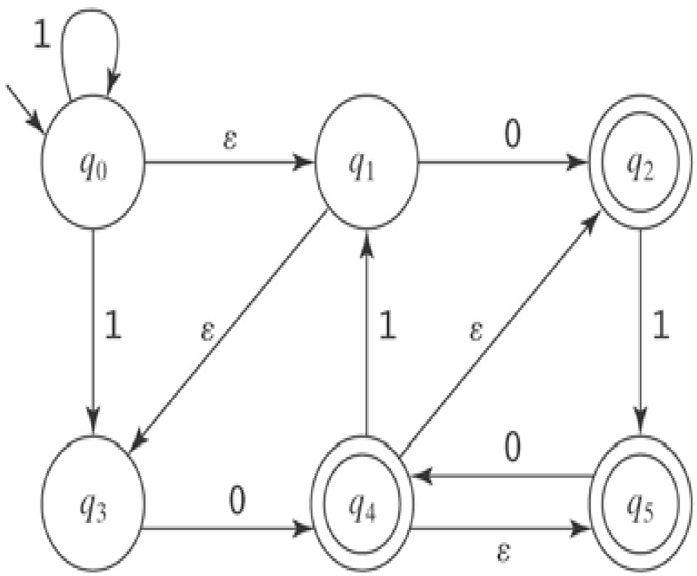
$\in$-NDFSM to DFSM

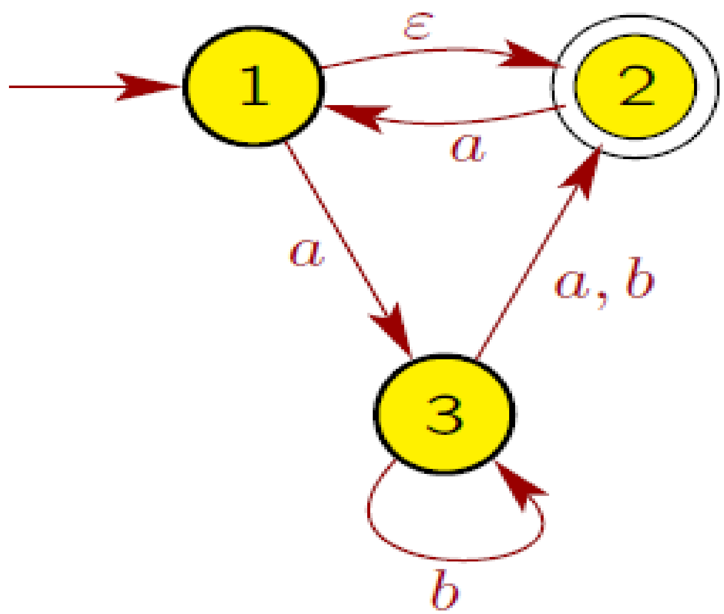
Computing $\in$-Closure of State(eps)



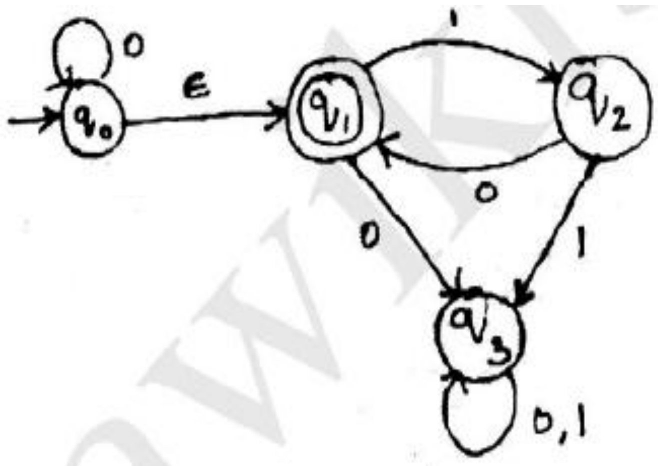
$eps(q_0) =$
$eps(q_1) =$
$eps(q_2) =$
$eps(q_3) =$

Example-1 (∈-NDFSM to DFSM) **Example-2**

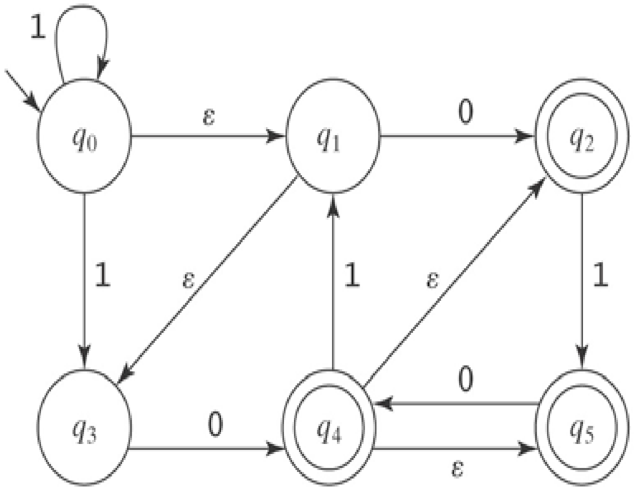


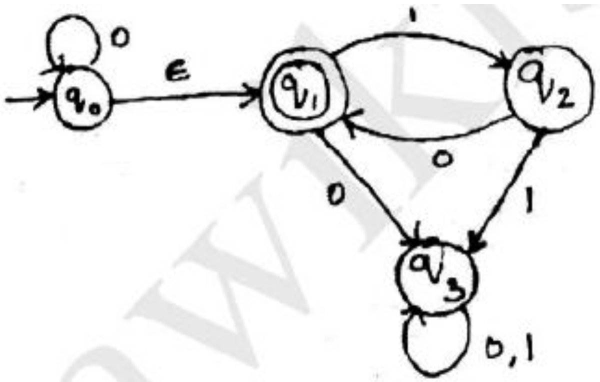


Example-3



Solution:




0
ε
1
0
0
1
0,1

Minimizing a DFSM

-The process of reducing a given DFSM to its minimal form is called as minimization of DFSM

-A DFSM M is minimal iff there is no other DFSM M' such that L(M) = L(M') and M' has fewer states than M does.
-Some states can be redundant
-There Exist a unique Minimal DFSM for ever Regular Language L.

-Most methods involve finding **equivalent states and merging them into single state.**

Equivalence of two states:

Two states ***p*** and ***q*** of a DFSM are equivalent(Indistinguishable) iff:

$$\delta(p, w) \in F \text{ and } \delta(q, w) \in F \quad \underline{\text{or}} \quad \delta(p, w) \notin F \text{ and } \delta(q, w) \notin F$$

for all strings $w \in \Sigma^*$.

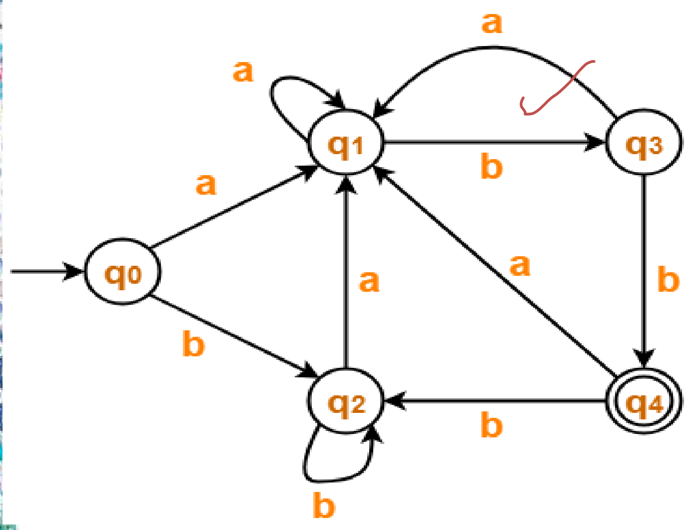
Otherwise states ***p*** and ***q*** are Distinguishable (i.e Distinct)

Method: Minimize DFSM using Partitioning(Spliting)

Procedure(High-Level Description)

1. Eliminate all the dead states and unreachable states from the given DFSM (if any)
2. Let k = 0
3. Divide Q (set of states) into two sets such that one set contains all the non-final states and other set contains all the final states. This Partition is called $\boldsymbol{\pi}_0$
4. k = k+1.
5. Find $\boldsymbol{\pi}_k$ by partitioning the different sets of $\boldsymbol{\pi}_{k-1}$. In each set of $\boldsymbol{\pi}_{k-1}$, consider all the possible pair of states within each set and if the two states are distinguishable, split the set into different sets in $\boldsymbol{\pi}_k$
6. Repeat step 4 and 5 until no change in partition occurs (i.e until $\boldsymbol{\pi}_k \neq \boldsymbol{\pi}_{k-1}$)
7. All those states which belong to the same set are equivalent and Can be merged.

Example-1



Π_0: (q_0, q_1, q_2, q_3) (q_4)

$(q_0, a) \rightarrow (q_0, q_1, q_2, q_3)$
$(q_1, a) \rightarrow (q_0, q_1, q_2, q_3)$
$(q_2, a) \rightarrow (q_0, q_1, q_2, q_3)$
$(q_3, a) \rightarrow (q_0, q_1, q_2, q_3)$
$(q_0, b) \rightarrow (q_0, q_1, q_2, q_3)$
$(q_1, b) \rightarrow (\;\text{''}\;)$
$(q_2, b) \rightarrow (\;\text{''}\;)$
$(q_3, b) \rightarrow (q_4)$

split required

Π_1: (q_0, q_1, q_2) (q_3) (q_4)

$(q_0, a) \rightarrow (q_0, q_1, q_2)$
$(q_1, a) \rightarrow (q_0, q_1, q_2)$
$(q_2, a) \rightarrow (q_0, q_1, q_2)$

NO

$(q_0, b) \rightarrow (q_0, q_1, q_2)$
$(q_1, b) \rightarrow (q_3)$
$(q_2, b) \rightarrow (q_0, q_1, q_2)$

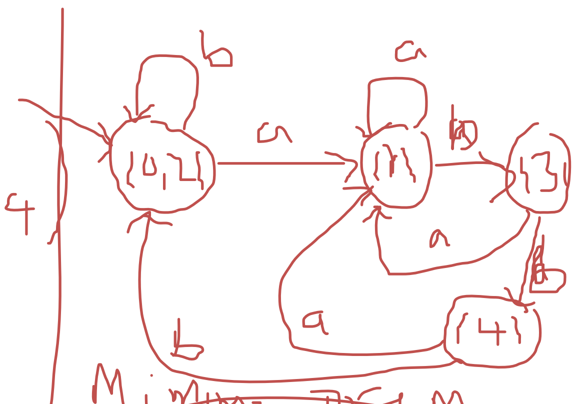
Π_2: (q_0, q_2) (q_1) (q_3) (q_4)

NO
$(q_0, a) \rightarrow (q_1)$
$(q_2, a) \rightarrow (q_1)$

$(q_0, b) \rightarrow (q_0, q_2)$
$(q_2, b) \rightarrow (q_0, q_2)$
NO

Π_3: (q_0, q_2) (q_1) (q_3) (q_4)

$\Pi_k := \Pi_{k-1}$

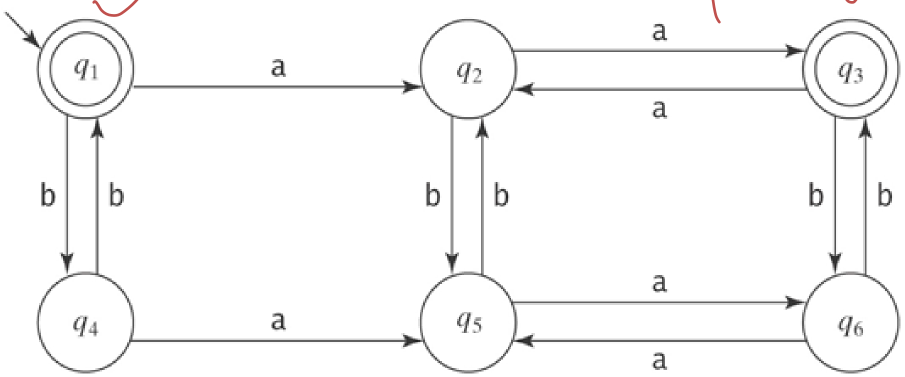


~~Minimal DFA~~

NO split required

Example-2

M



Π_0: (q_1, q_3) (q_2, q_4, q_5, q_6)

$(q_1, a) \to (2,4,5,6)$ $(q_3, a) \to (2,4,5,6)$

$(q_1, b) \to (2,4,5,6)$ $(q_3, b) \to (2,4,5,6)$

Π_1: (q_1, q_3) (q_2, q_4, q_5, q_6)

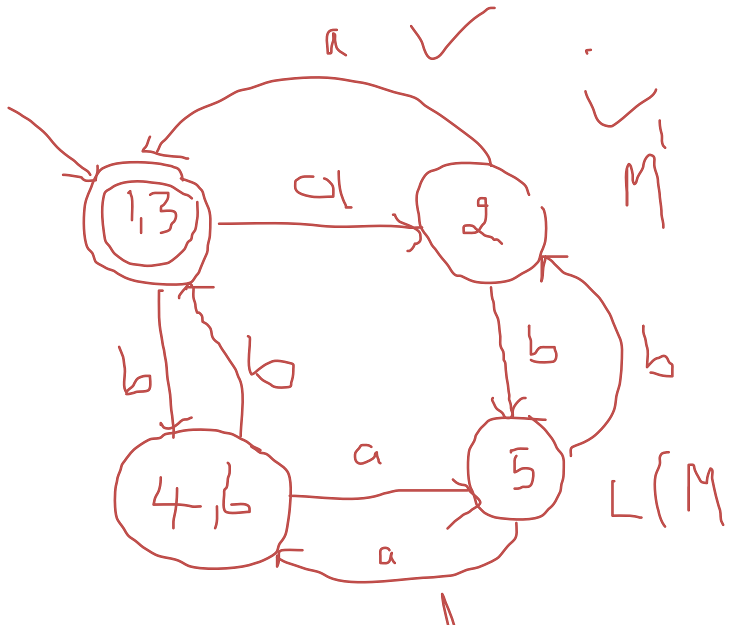
$(2,a) \to (1,3)$ $(4,a) \to (2,4,5,6)$ $(5,a) \to (2,4,5,6)$ $(6,a) \to (2,4,5,6)$

Split q_2

$\Pi_2 =$ (q_1, q_3) (q_2) (q_4, q_5, q_6)

$\Pi_3 =$ (q_1, q_3) (q_2) (q_4, q_6) (q_5)

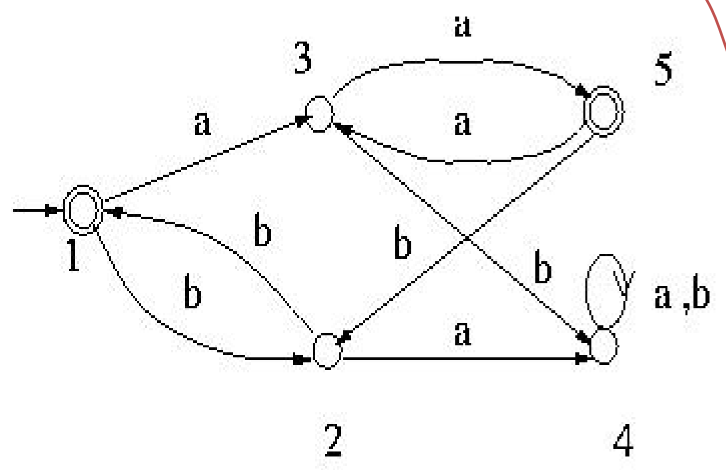
Π_4: (q_1, q_3) (q_2) (q_4, q_6) (q_5)



$L(M) = L(M')$

Minimized DFSM

Example-3



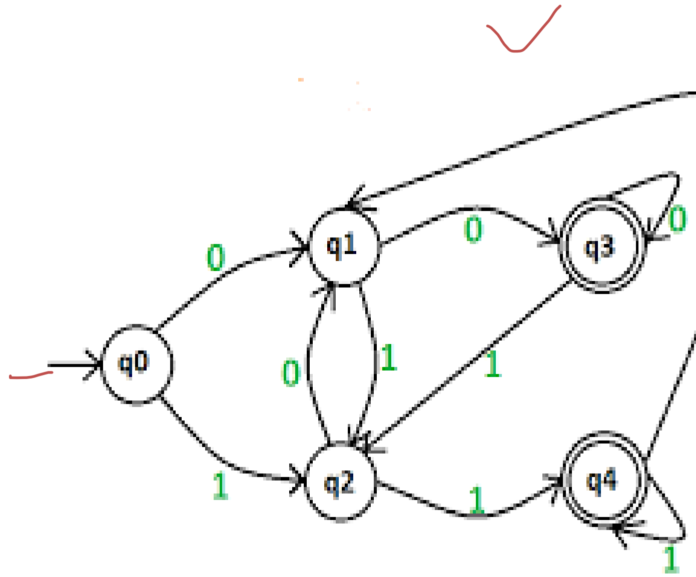
Home Work

Minimize the following DFSMs

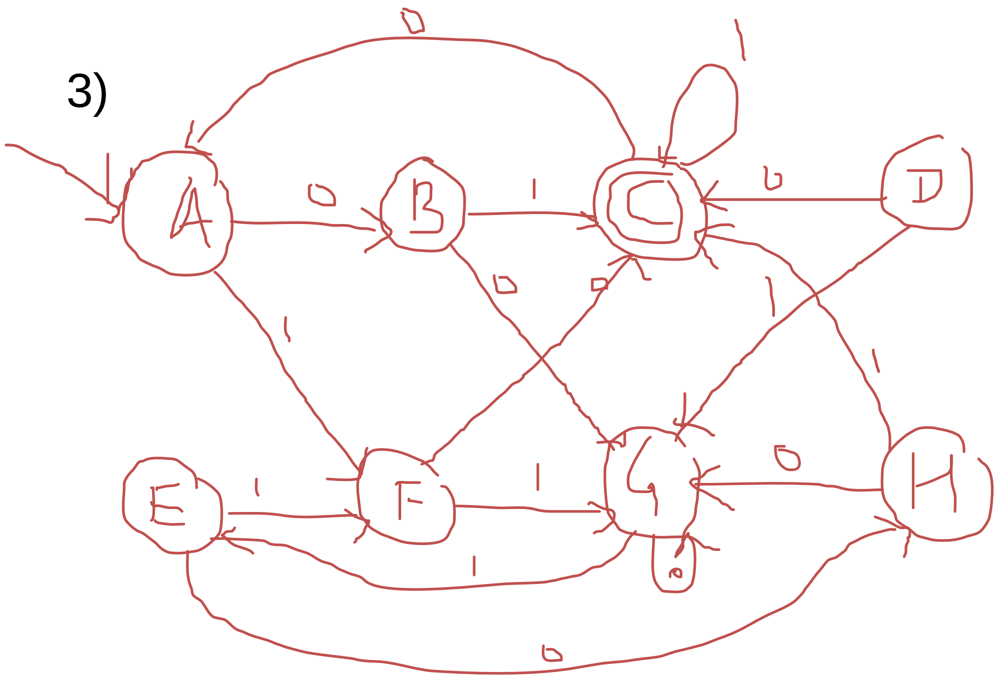
1)

δ	0	1
→A	B	F
B	G	C
C	A	G
D	C	G
E	H	F
F *	C	G
G	G	E
H *	G	C

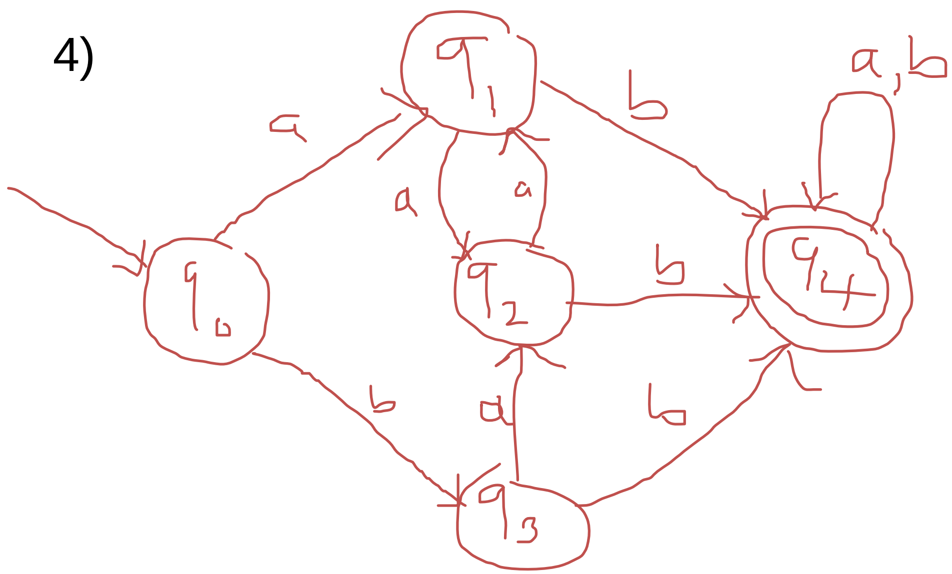
2)

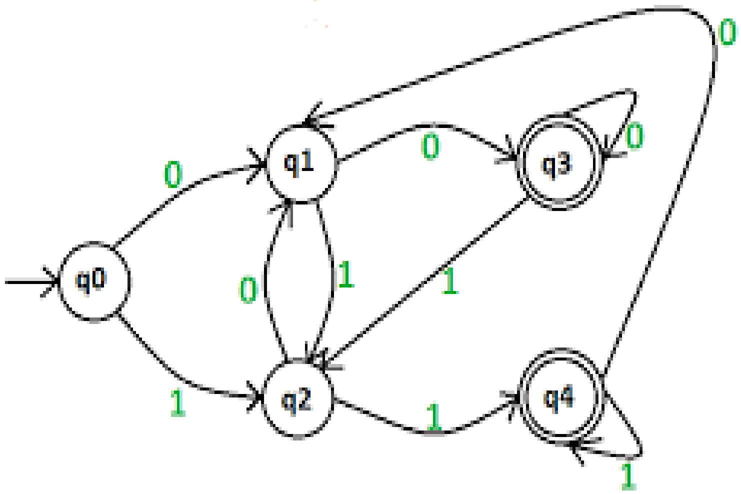


3)



4)




q0
q1
q2
q3
q4
0
1
0
1
0
1
0
1
0
1

A Canonical form for FSM

A ***canonical form*** for some set of objects *C* assigns exactly one representation to each class of "equivalent" objects in *C*.

Further, each such representation is distinct, so two objects in *C* share the same representation iff they are "equivalent" in the sense for which we define the form.

Procedure: To build a Canonical form for FSM

buildFSMcanonicalform(M: FSM) =

1. $M' = ndfsmtodfsm(M)$.
2. $M^* = minDFSM(M')$.
3. Create a unique assignment of names to the states of M^*.
4. Return M^*.

Given two FSMs M_1 and M_2:

$$buildFSMcanonicalform(M_1) = buildFSMcanonicalform(M_2)$$

$$\text{iff } L(M_1) = L(M_2).$$

It provides the basis for a simple way to test whether two FSMs are equivalent or not…

Example

Introduction to Transducers

Introduction to Transducers

What is a transducer?

- Defined as a device that converts one form of energy to another.
- Examples: Motors, Speakers, Microphones, Antennas, Light Bulbs, Potentiometer, Gauges…
- Essentially Sensors and Actuators

Finite-State Transducers (FST)

FSM

Deterministic

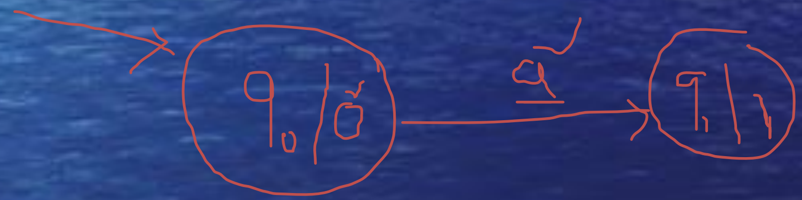
- Two tapes, one for input and one for output.
- Converts a input into an output.
- Mealy and Moore machines.

i/p o/p

010 → aba

Moore Machine

- Invented by Edward F. Moore (1925 -2003)
- Associates an output with each state of the machine.



~~Final state~~

Moore Machine Formal Definition

is a 5-tuples, Denoted by $M = (Q, \Sigma, O, \delta, \lambda, q_0)$ where

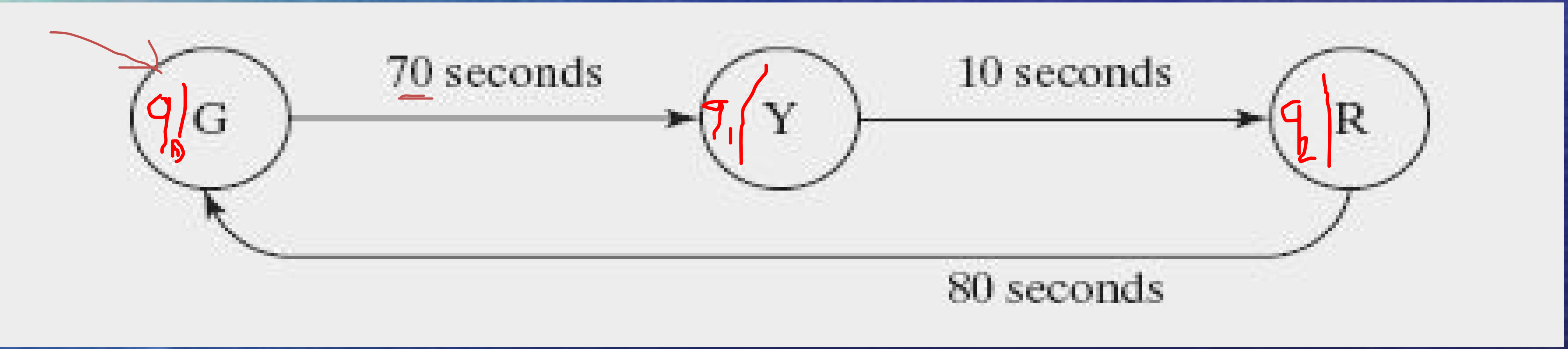
- Q = Finite set of States
- Σ = Input symbols
- O = Output symbols
- δ = Transition Function $(Q \times \Sigma \rightarrow Q)$
- λ = Output Function $(Q \times \Sigma \rightarrow O)$
- q_0 = Initial /Start State

$\Sigma = \{0, 1\}$

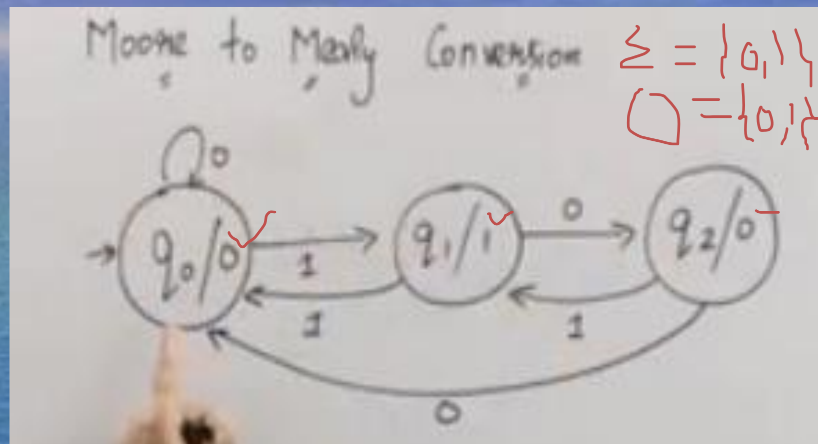
$O = \{a, b\}$

$\lambda(q_0, 0) \rightarrow a \in O$

Street light Example



Example: Moore m/c



1001 0 type 1

m/c

01001 0 type 2

I/p n symbols → O/p $n+1$ symbols

w: 10010

$q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2 \xrightarrow{0} q_0 \xrightarrow{1} q_1 \xrightarrow{0} q_2$

0 1 0 0 1 0

O/p: 010010

Mealy Machine

- Invented by George H. Mealy in 1955
- Associates outputs with transitions.

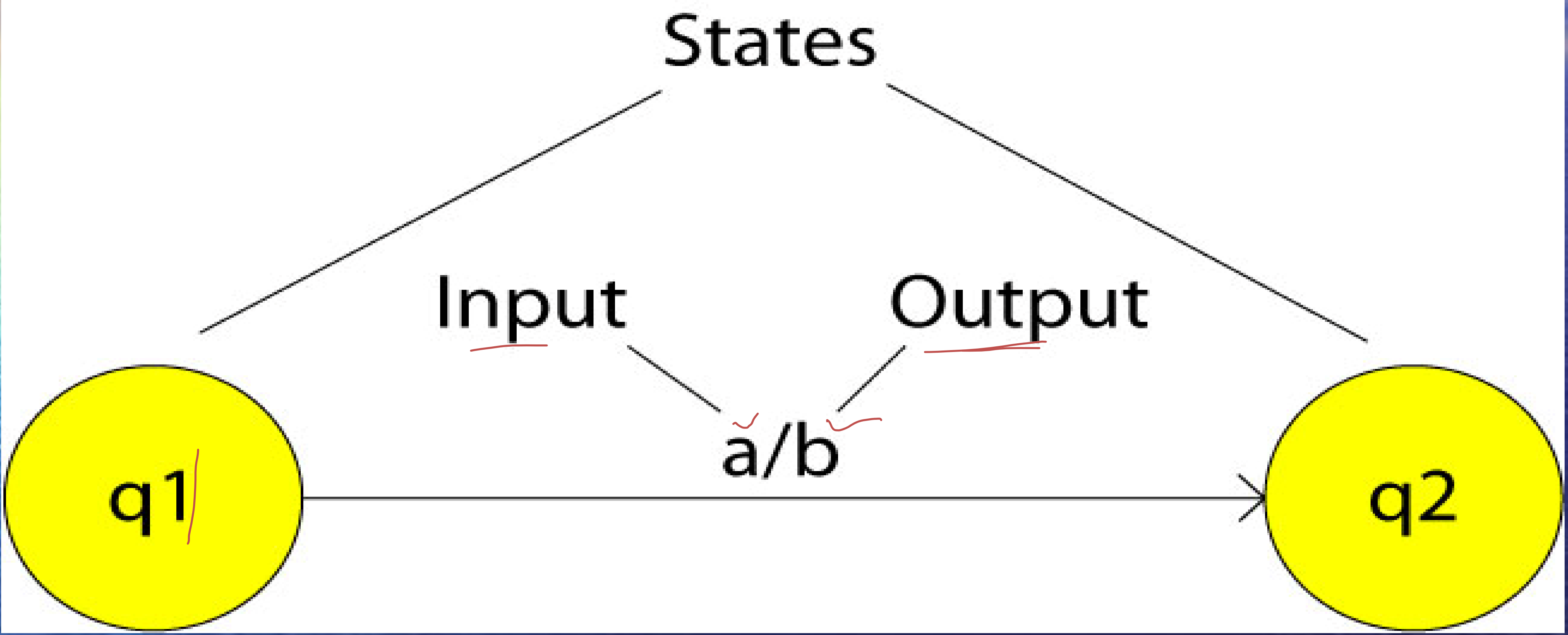
Mealy Machine Formal Definition

Is a 6-tuples , denoted by $M = (Q, \Sigma, O, \delta, \lambda, q_0)$ where

$\lambda(q_0, 0) \rightarrow 1$

- Q = Finite set of States
- Σ = Input symbols
- O = Output symbols
- δ = Transition Function
- λ = output function
- q_0 = Initial State / Start state

Mealy Notation



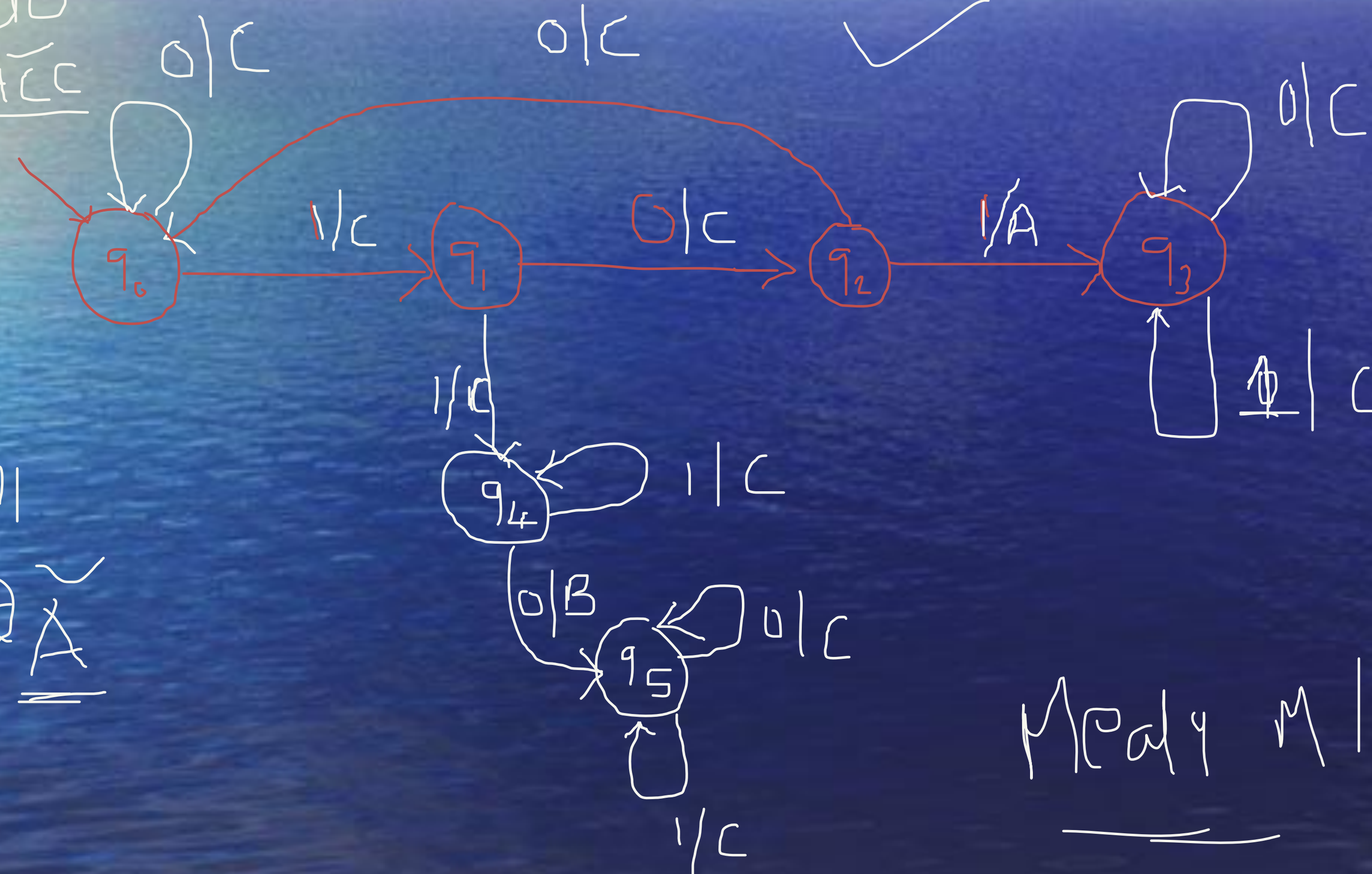
Example -1: Design a Mealy M/c for a binary input sequence, such that , if it has a substring 101, the machine outputs A. If input has substring 110, the machine outputs B. Otherwise it outputs C.

JULY-18

01010110
CCCCCACC

$\Sigma = \{0, 1\}$

$O = \{A, B, C\}$



101

CC A

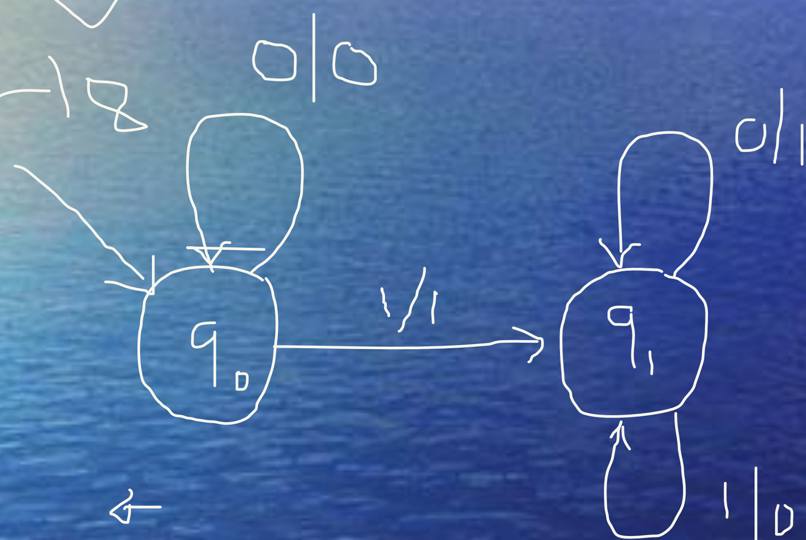
Mealy M/C

Example-2: Design a mealy m/c that takes binary number as input and produces 2's compliment of that number as output . Assume the input is read from LSB to MSB and end carry is discarded.

$\Sigma = \{0, 1\}$

$\Delta = \{0, 1\}$

JULY-18



1010 → 0110

0101 → 1's

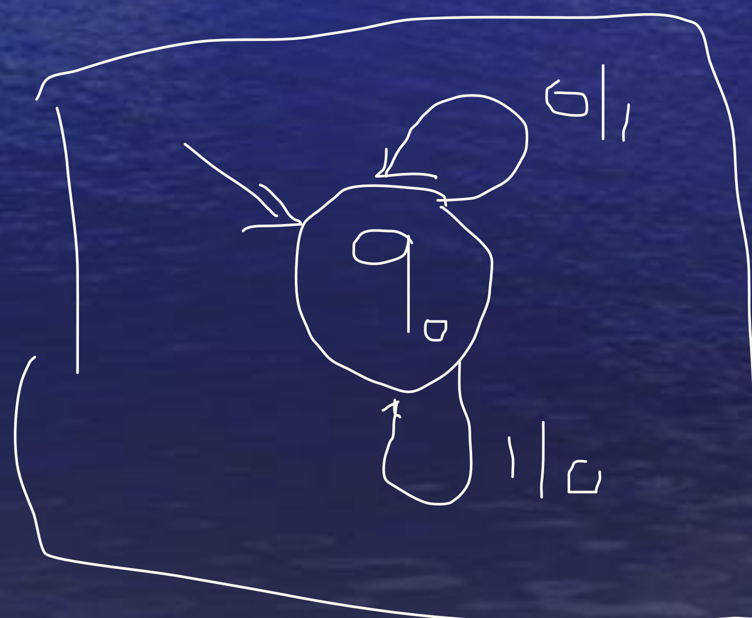
+ 1

0110 → 2's

w = ~~0101011~~

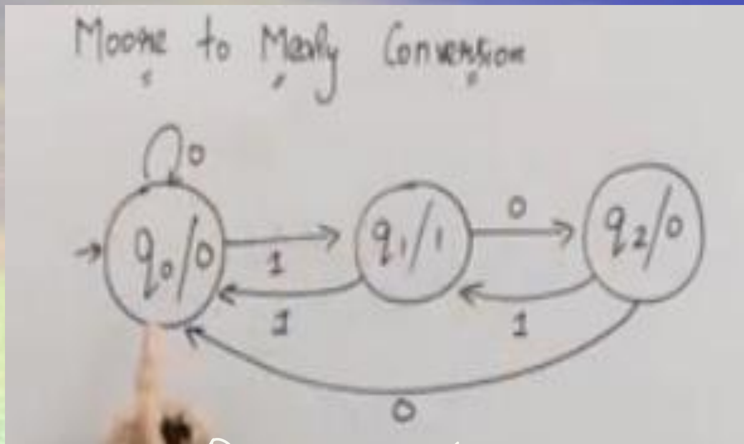
+ 1

2's → 10101100

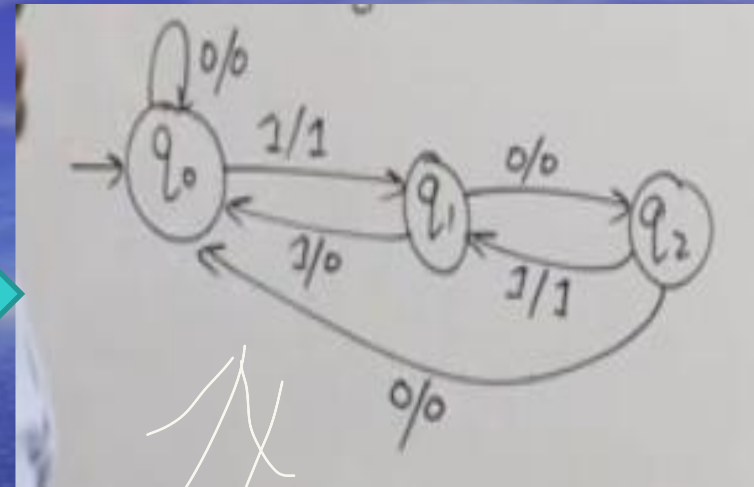


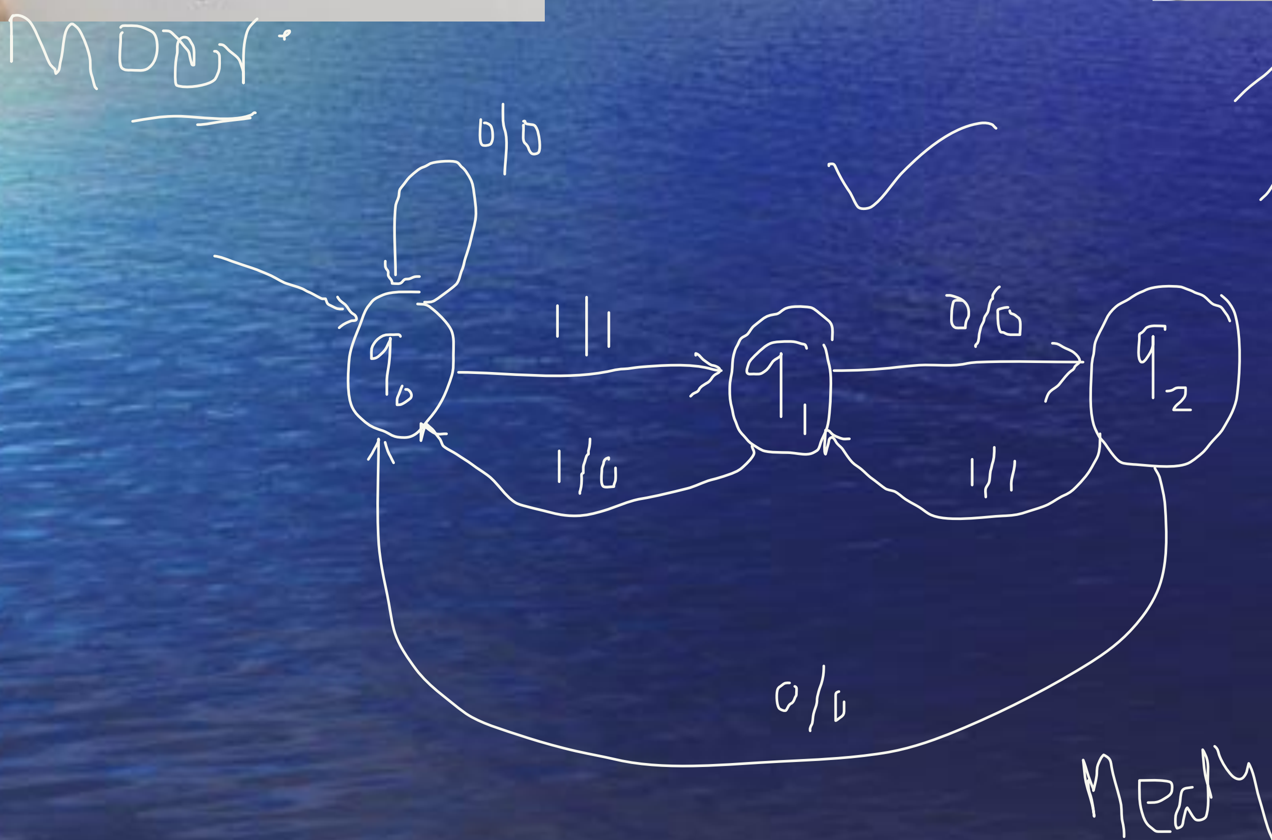
1's

Moore to Mealy Conversion



Moore to Mealy

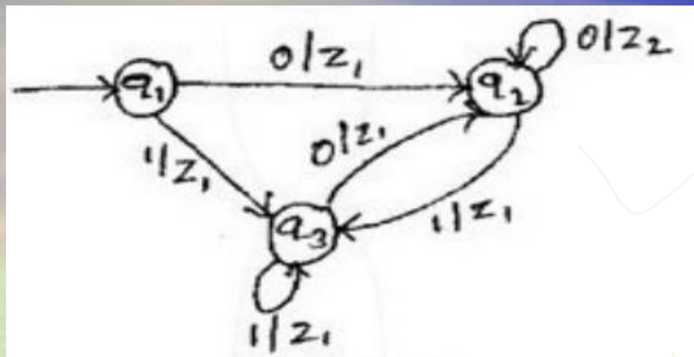




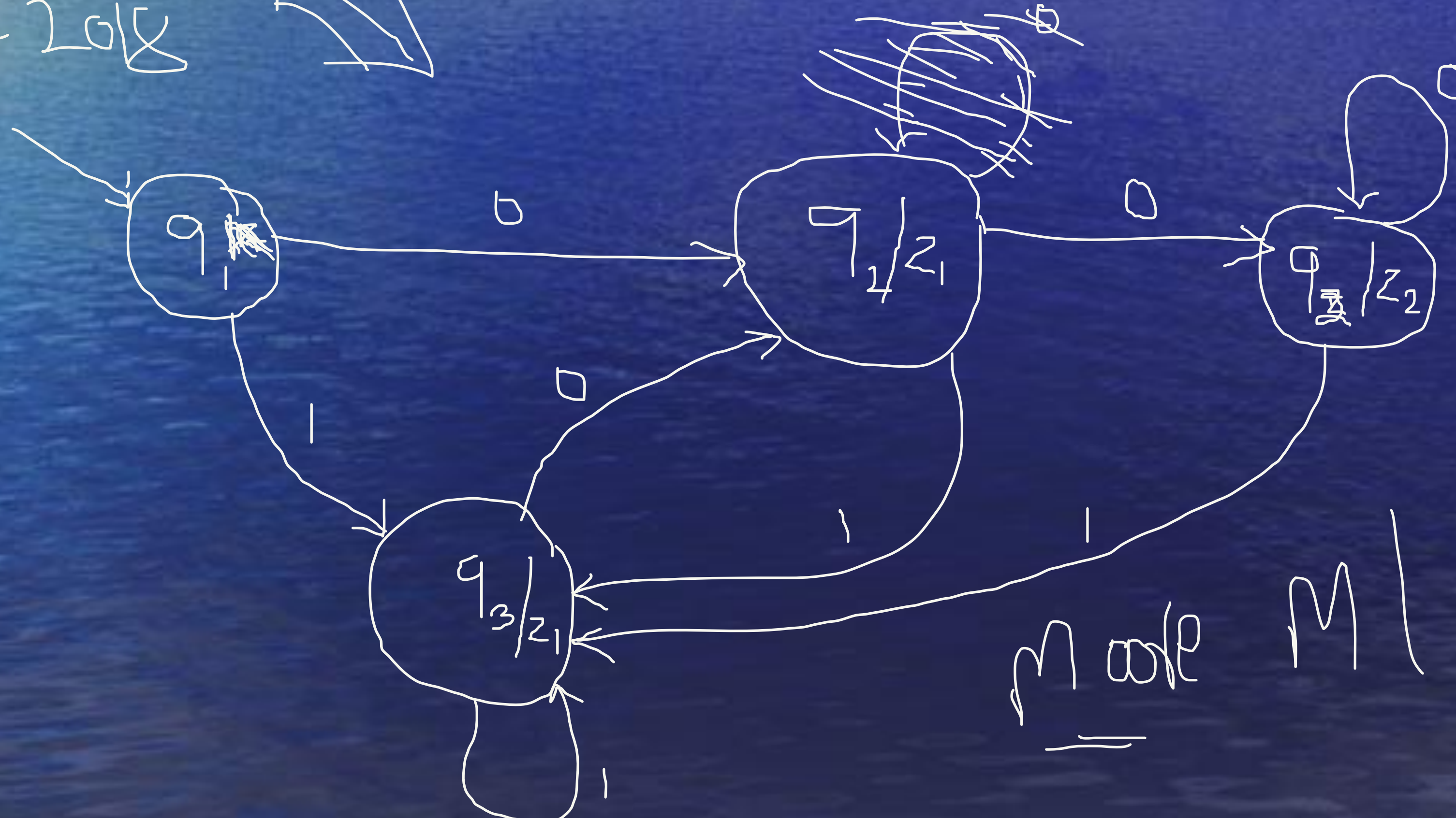
Moore to Mealy Conversion

Mealy to Moore

$\Sigma = \{0, 1\}, \; O = \{z_1, z_2\}$



JULY-2018



Moore M/C

Module-II

Regular Expressions and Regular Grammars

Regular Expressions

-Operators to build REs and their Precedence Levels
-Building Regular expression for RLs.
-Kleene's theorem: Building an FSM from a RE
-Building RE from a FSM using Ripping method
-Applications of REs,

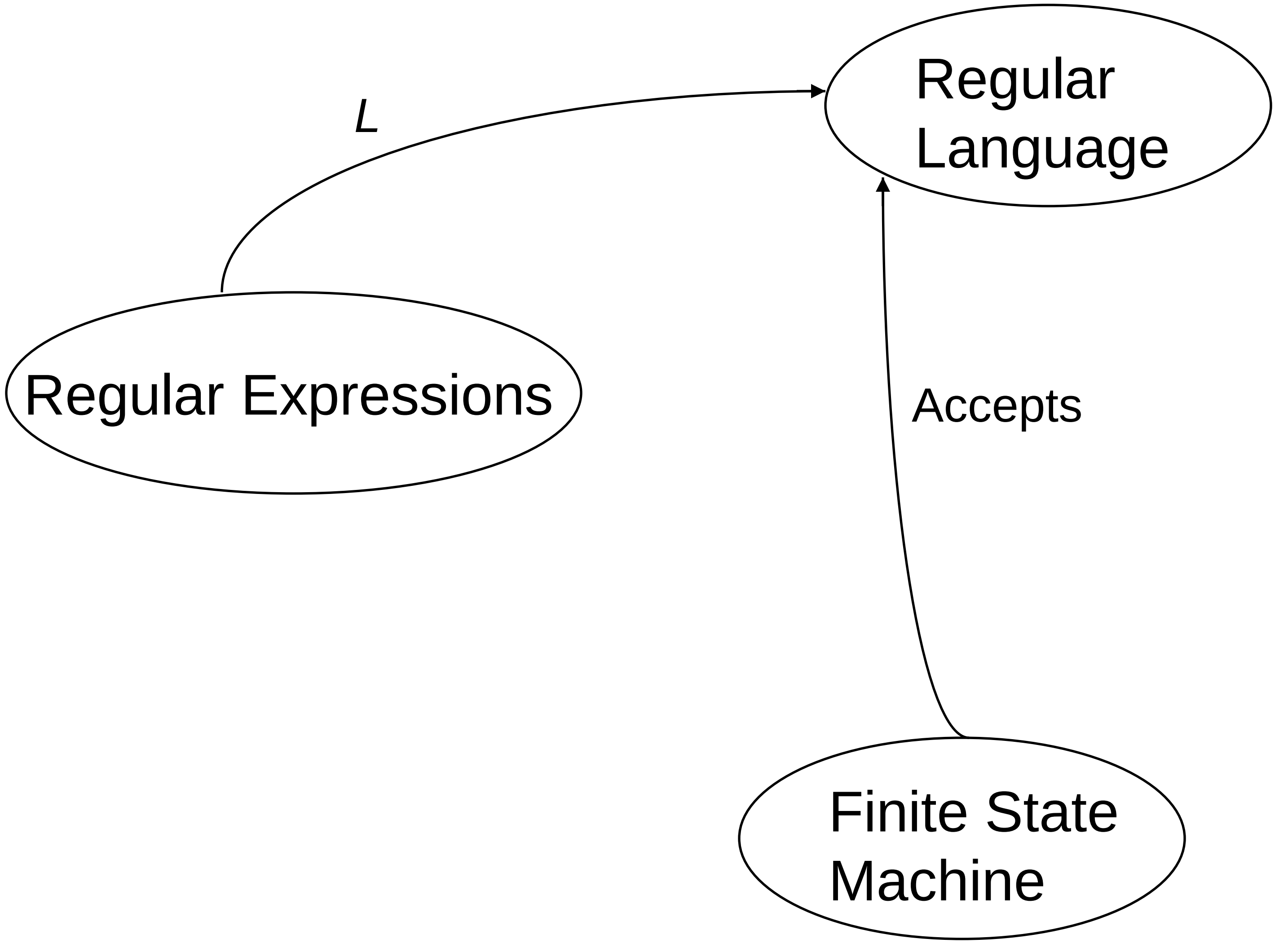
Regular Grammars

-Definition of a Regular Grammar, Examples
-Regular Grammars and Regular languages.

Properties of Regular Languages

-Regular Languages and Non-regular Languages
-Closure properties of RLs
-To show some languages are not RLs using Pumping Lemma

Regular Languages



Definition of Regular Expressions

The regular expressions over an alphabet Σ are all and only the strings that can be obtained as follows:

1. $\varnothing$ is a regular expression, denoting $L(\varnothing) = \varnothing$
2. ε is a regular expression, denoting $L(\varepsilon) = \{\varepsilon\}$
3. Every symbol a belongs to Σ is a regular expression.
4. If α, β are regular expressions, then so is $\alpha\beta$.
5. If α, β are regular expressions, then so is $\alpha|\beta$.
6. If α is a regular expression, then so is α^*.

Operator Precedence in Regular Expressions

Precedence	Operators	
Highest	Kleene star	(*)
	concatenation	(.)
Lowest	union	(|)

Regular Expression Examples

If Σ = {a, b}, the following are regular expressions:

$\varnothing$

ε

a

$(a \cup b)^*$ or $(a|b)^*$

$abba \cup \varepsilon$ or $abba \mid \varepsilon$

Examples: RE

1) a*b*

2) (a | b)*

3) (a |b)*a*b*

4) (a | b)*abba (a|b)*

Examples Contd…

Obtain a Regular Expression to accept all the strings of a's & b's of length ≤ 2

Obtain a RE to accept strings of a's & b's with even number of a's followed by add number of b's

Obtain a RE to accept all the strings of 0's and 1's ending with either 01 or 10

Obtain a RE to accept all the strings of a's and b's having substring abb

Build a RE to accept all strings of a', b's & c's containing atleast one a & atleast on b over $\Sigma = \{ a,b,c\}$

Obtain a RE representing strings of a's and b's having odd length.

RE: $((a \mid b)(a \mid b))^*(a \mid b)$

Obtain a RE to accept a Language consisting of strings of a's and b's with alternate a's and b's

RE: $(\varepsilon \mid b)\ (ab)^*\ (\varepsilon \mid a)$

Build a RE to accept strings of 0's & 1's having no two consecutive 0

RE: $(1 \mid 01)^*\ (0 \mid \varepsilon)$

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*: |w| \text{ is even}\}$

RE: $((\mathtt{a} \mid \mathtt{b})\ (\mathtt{a} \mid \mathtt{b}))^*$

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^*: w \text{ contains an odd number of } \mathtt{a}\text{'s}\}$

RE: $\mathtt{b}^*\ (\mathtt{ab}^*\mathtt{ab}^*)^*\ \mathtt{a}\ \mathtt{b}^*$

$L = \{w \in \{\mathtt{a}, \mathtt{b}\}^* : \text{every } \mathtt{a} \text{ is immediately followed } \mathtt{b}\}$

RE: $(\mathrm{b} \mid \mathrm{ab})^*$

Obtain a RE to recognize all strings of a's & b's whose 3rd symbol from the right is 'a'

RE:

Obtain a RE to accept strings of a's & b's begin and end with same symbol.

RE:

Develop RE for $L = \{ a^{2n}b^{2m} \mid n \geq 0 , m \geq 0 \}$

RE:

Obtain a RE to accept strings of a's & b's containing no more then three a's

RE: $b^*(\varepsilon \mid a)\, b^*(\varepsilon \mid a) b^*(\varepsilon \mid a) b^*$

Obtain a RE for $L = \{ a^n b^m : n \geq 4 , m \leq 3 \}$

RE: $aaaaa^*(\varepsilon \mid b)^3$

Obtain a RE to recognize strings of a's and b's whose length is multiple of 3

or

$L = \{ w : |w| \bmod 3 = 0, w \in \{ a,b\}^* \}$

RE: $((a|b)(a|b)(a|b))^*$

Obtain a RE for L = { $a^n b^m$ | m+n is even }

$n = 1, m = 1, m + n = 2$

case 1: m, n are even ⇒ m + n ⇒ even

$a^3 b^3 = |a^3 b^3| = 3+3 = 6$

RE_1: $(aa)^*(bb)^*$

case 2: m, n are odd, m + n ⇒ even

RE_2: $(aa)^* a (bb)^* b$

RE: $RE_1 | RE_2 = (aa)^*(bb)^* \mid (aa)^* a (bb)^* b$

Obtain a RE for L = { $a^m b^n$ | m ≥1, n ≥ 1, mn ≥ 3 }

case 1: if $m = 1, n \geq 3$, then $mn \geq 3$

RE_1: $abbbb^*$

case 2: if $n = 1, m \geq 3$, then $mn \geq 3$

RE_2: $aaaa^*b$

case 3: if $m \geq 2$, $n \geq 2$ then $mn \geq 3$

RE_3: aaa^*bbb^*

RE: $RE_1 | RE_2 | RE_3$

Obtain a RE for the set of all strings that do not end with 01, over {0,1}*

00, 10, 11

$(0|1)^*01$

RE: $(0|1)^*(00|10|11)$

Keywords

RE: int | float | char | for | . . . . .

Regular expression to recognize variables, signed integer & signed real numbers

RE: $(A|B|\ldots Z|a|b|\ldots|z)(A|B|\ldots Z|a|b|\ldots z|0|1|\ldots|9)^*$

52+1

OR

$\Sigma = \{A-Z, a-Z, 0-9\}$ [A-Za-Z] $[A-Za-z0-9]^*$

Area2

1x

x (-) +286

2 86

-320

RE: $(+|-|\epsilon)(0|1|\ldots|9)(0|1|\ldots|9)^*$

or

$(+|-|\epsilon)(0|1|\ldots|9)^+$ ⇒ $[+|-|\epsilon][0-9]^+$

m.f

10.3 × 10^6

10.3e+6

10.~~20~~ = 10(.20)

RE: $(+|-|\epsilon)(0|1\ldots|9)^+(.(0|1|\ldots|9)^+|\epsilon)(e(+|-|\epsilon)[0|1\ldots 9]|\epsilon)$

120.30

125

Algebraic Laws for Regular expressions

If r,s and t are any arbitrary RE then:

$a + b = b + a$

Law	Description
r | s = s | r	| is commutative
r | (s | t) = (r | s) | t	| is associative
r (st) = (rs)t	Concatenation is associative
r(s|t) = rs | rt; (s|t)r = sr | tr	Concatenation is distributive
Ɛr = rƐ = r	Ɛ is identity for concatenation
r* = (r | Ɛ)*	Ɛ is guaranteed in closure
r** = r*	* is idempotent

$st \neq ts$

$r^{***} = r^{*}$

$(ab|\epsilon)^* = (ab)^*$

$(+|-|\epsilon)^* = (+|-)^*$

$(a|b)^* \cdot \epsilon = \epsilon \cdot (a|b)^* = (a|b)^*$

$(a|b)^* | ab = ab | (a|b)^*$

$m * 1 = 1 * m = m$

$r | s = s | r$

Applications of Regular expressions

X.C

1. Regular expressions in UNIX/Linux operating systems
2. Regular Expressions in Pattern Matching(Search Engines)
3. Regular Expressions in Software Engineering
4. Regular expressions in Programming Languages(Perl, Python etc).
5. Regular Expressions in Lexical Analysis(Compiler Design)

Kleene's Theorem

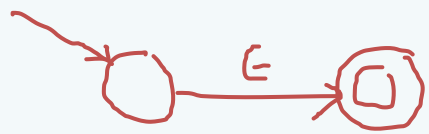
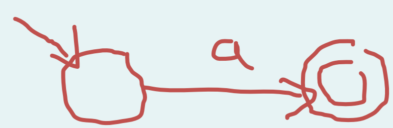
Finite state machines and regular expressions define the same class of languages. To prove this, we must show:

Theorem: Any language that can be defined with a regular expression can be accepted by some FSM and so is regular.

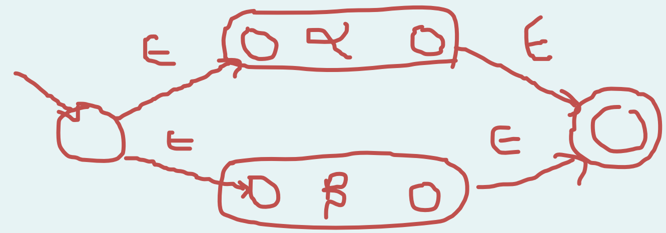
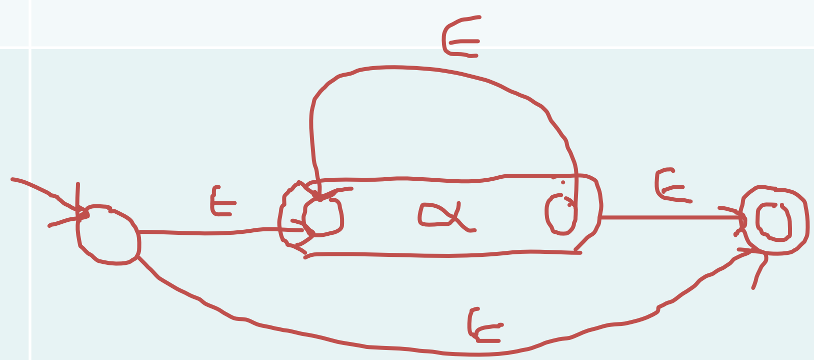
Theorem: Every regular language (i.e., every language that can be accepted by some DFSM) can be defined with a regular expression.

For Every Regular Expression there is a Corresponding FSM

We'll show this by construction. An FSM for:

Regular expression	*FSM*
$\emptyset$ $L(\emptyset) = \emptyset$	
ε $L(\varepsilon) = \{\varepsilon\}$	
for any a ∈ Σ	

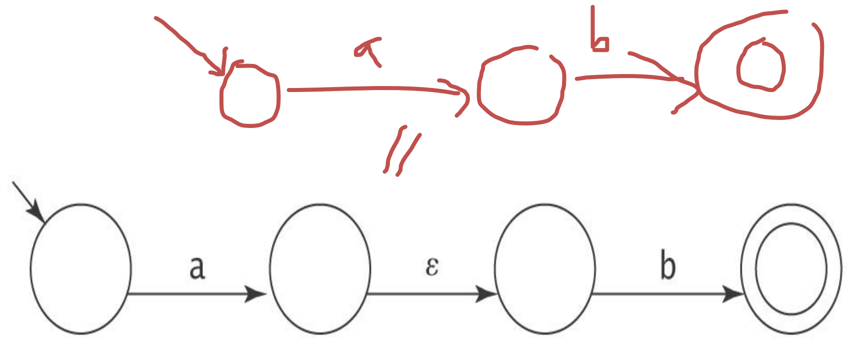
Proof Contd…

Regular expression	***FSM***
$\alpha \mid \beta$	
$\alpha\beta$	 $\alpha \epsilon \alpha \epsilon \alpha = \alpha^3$
α^*	

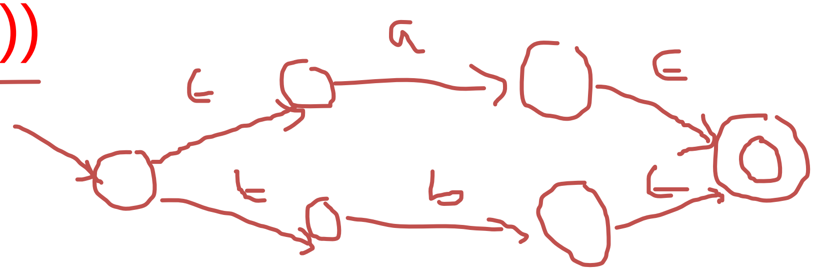
An Example

RE: ab

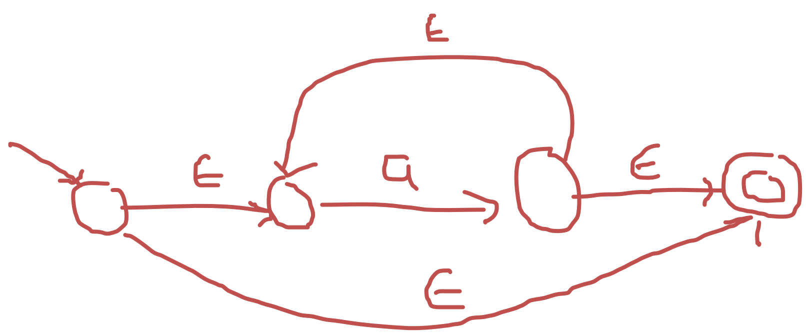
An FSM for `ab`:



RE: (a | b) or L((a | b))

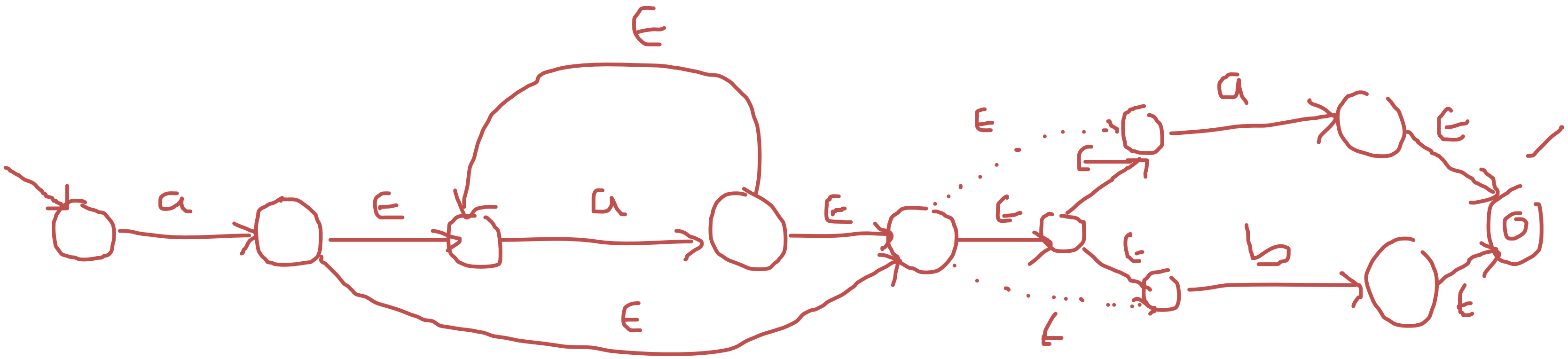


RE: a*

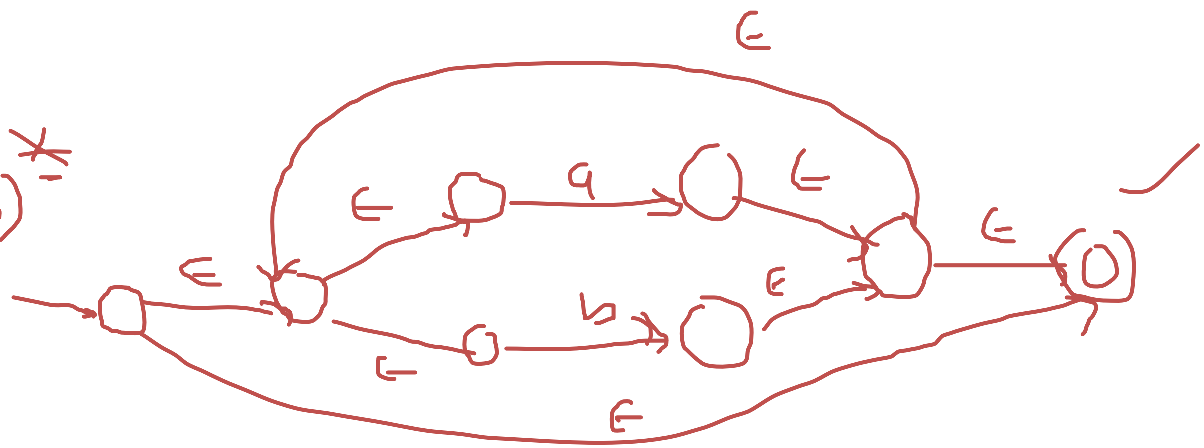


Design an NDFSM that accept the language L(aa*(a+b))

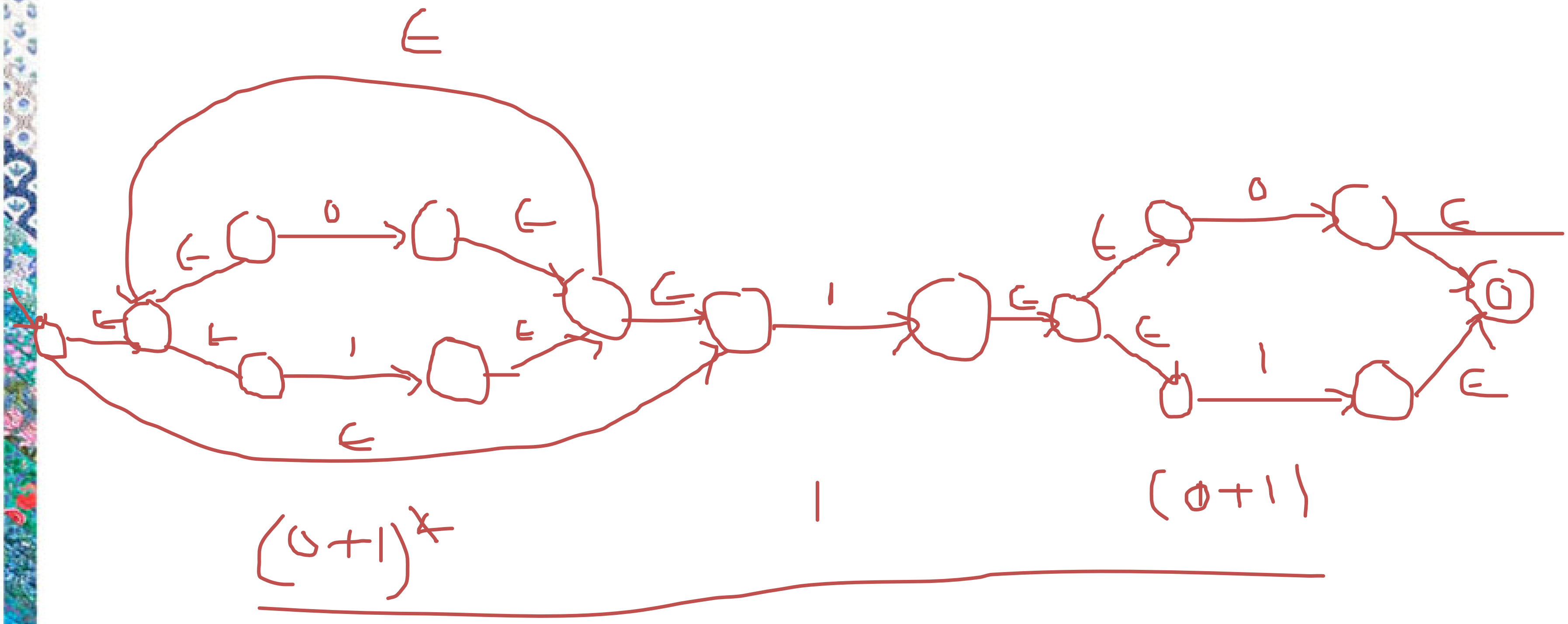
RE: $aa^*(a+b)$



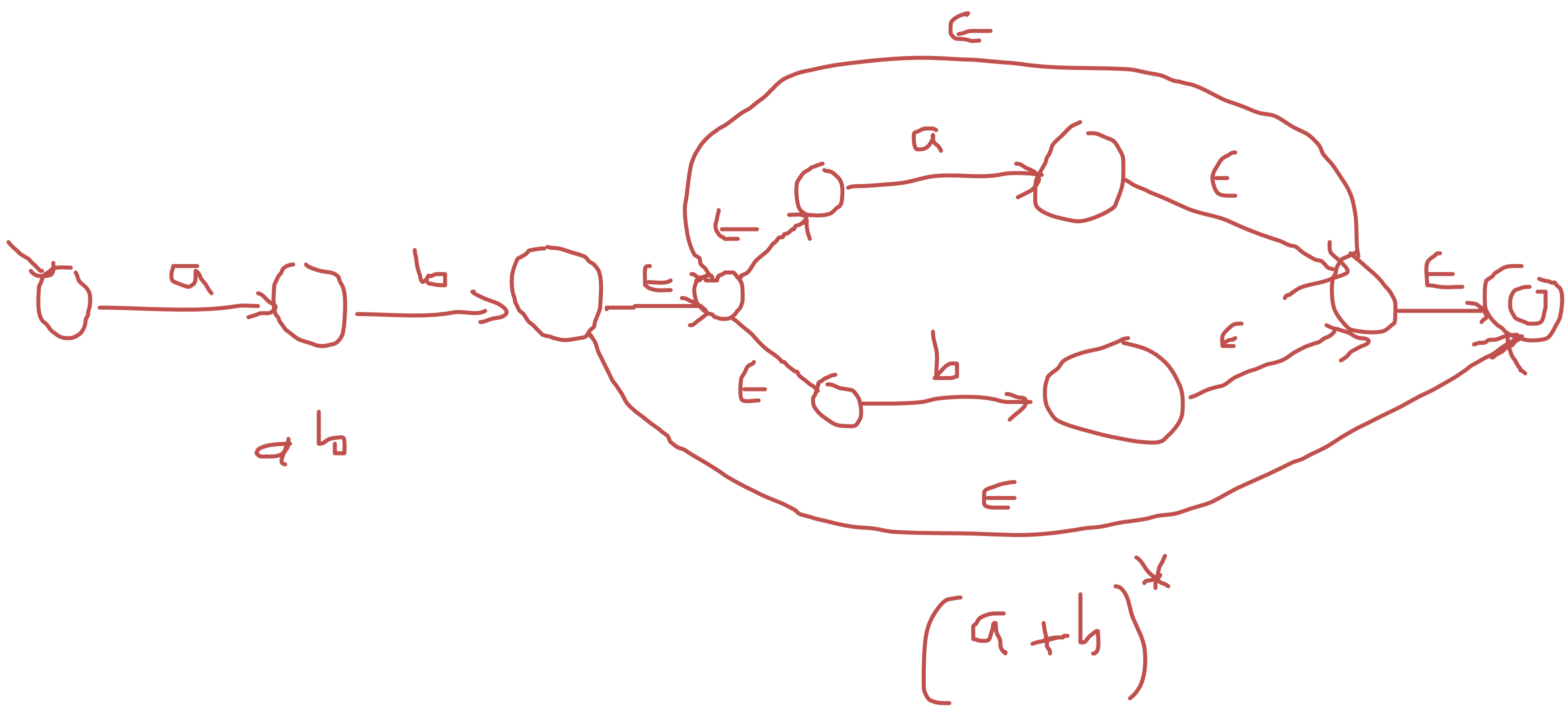
$(a|b)^*$



Convert the regular expression (0 +1)*1(0 + 1) to NDFSM

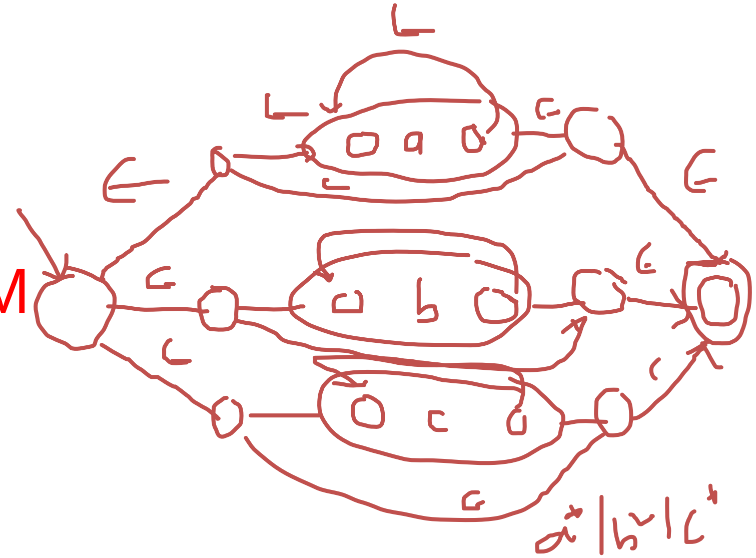


Obtain an NDFSM that accept the Language: L(ab(a+b)*)



Home Work

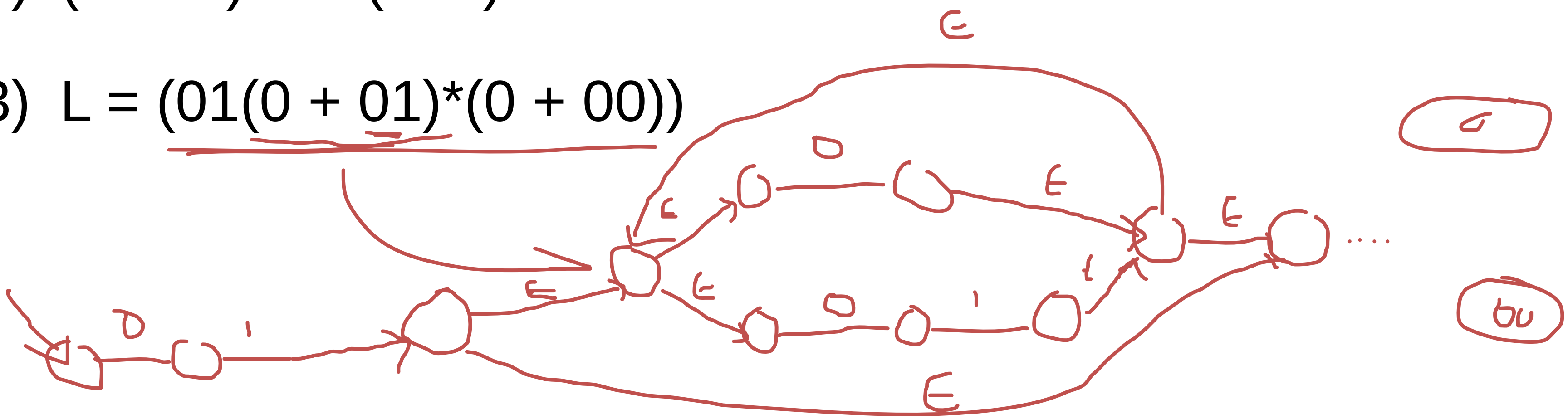
Convert the following Res to NDFSM



1) a* | b*| c* $a^+ | b^+ | c^+$

2) (a + b)* aa (a+b)*

3) L = (01(0 + 01)*(0 + 00))

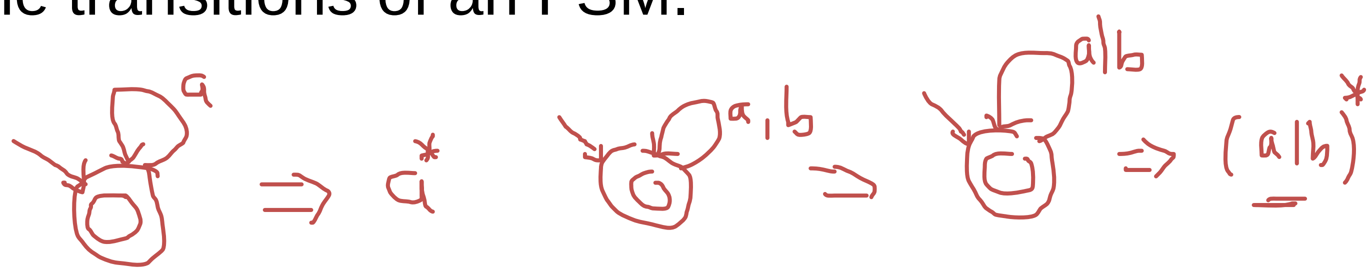


Solution

For Every FSM There is a Corresponding Regular Expression

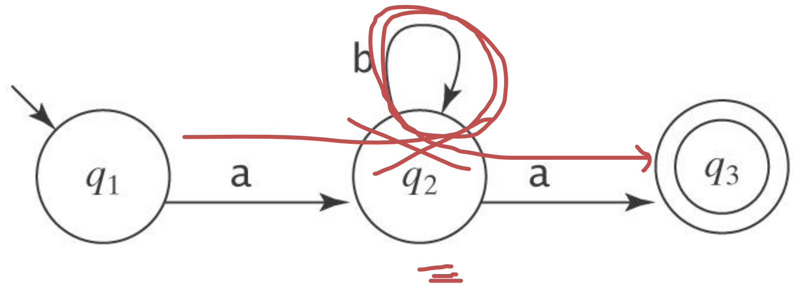
We'll show this by construction(Ripping or State elimination)

The key idea is that we'll allow arbitrary regular expressions to label the transitions of an FSM.



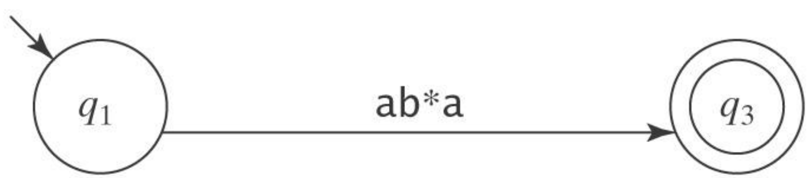
A Simple Example

Let *M* be:

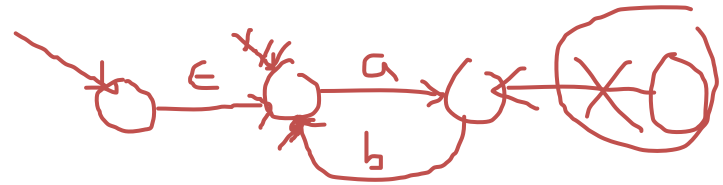


Suppose we rip out state 2:

RE: ab*a

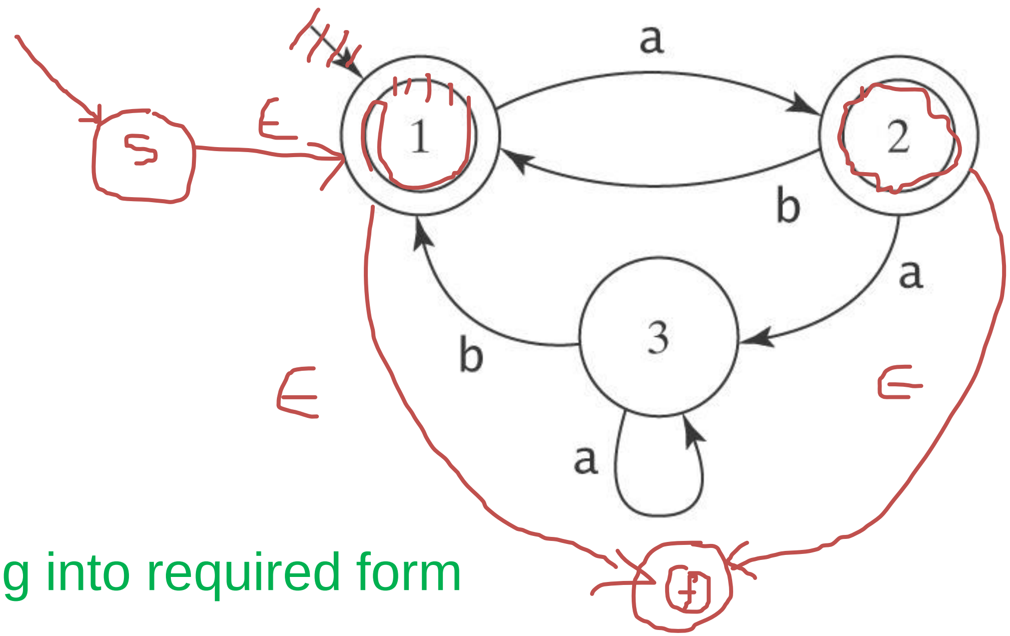


The Algorithm



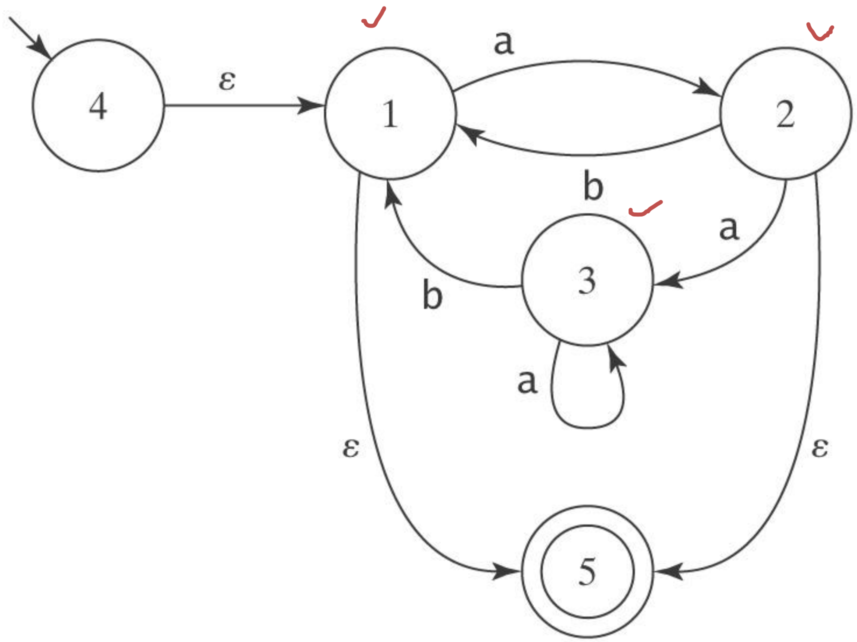
1. Remove unreachable states from *M*.
2. If *M* has no accepting states then return ∅.
3. If the start state of *M* is part of a loop, create a new start state *s* q_0 and connect *s* to *M*'s start state via an ε-transition.
4. If there is more than one accepting state of *M* or there are any transitions out of any of them, create a new accepting state and connect each of *M*'s accepting states to it via an ε-transition. The old accepting states no longer accept.
5. If *M* has only one state then return ε.
6. Until only the **start state** and the **accepting state** remain do:
   - 6.1 Select *rip* (not *start* or an accepting state).
   - 6.2 Remove *rip* from *M*.
   - 6.3 *Modify the transitions among the remaining states so *M* accepts the same strings.
7. Return the regular expression(between start and final state)

Example-1



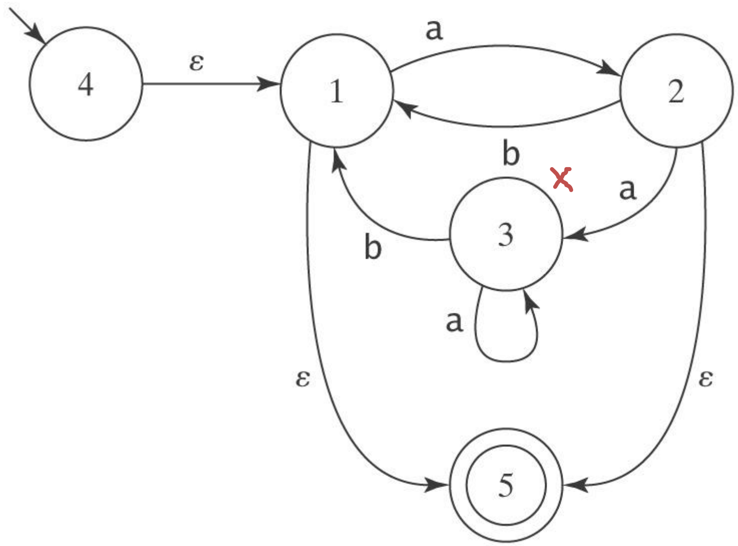
Getting into required form

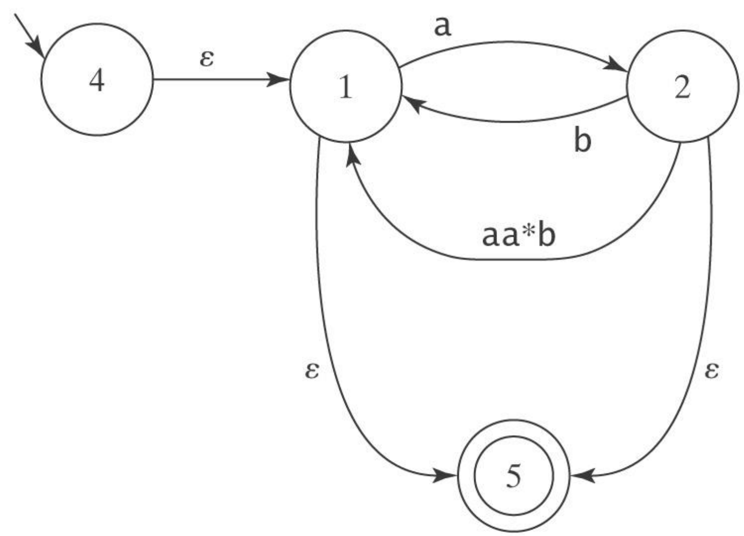
1. Create a new initial state and a new, unique accepting state, neither of which is part of a loop.



2. Remove states and arcs and replace with arcs labelled with larger and larger regular expressions.

An Example, Continued





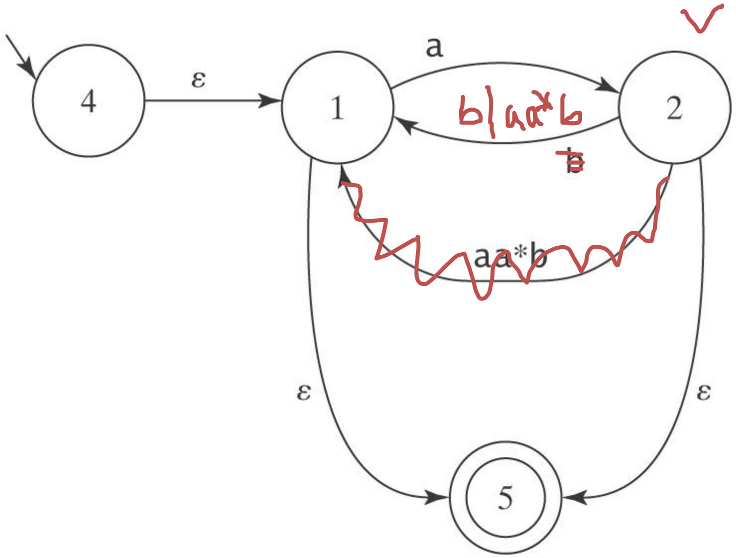
Remove state 3:

An Example, Continued

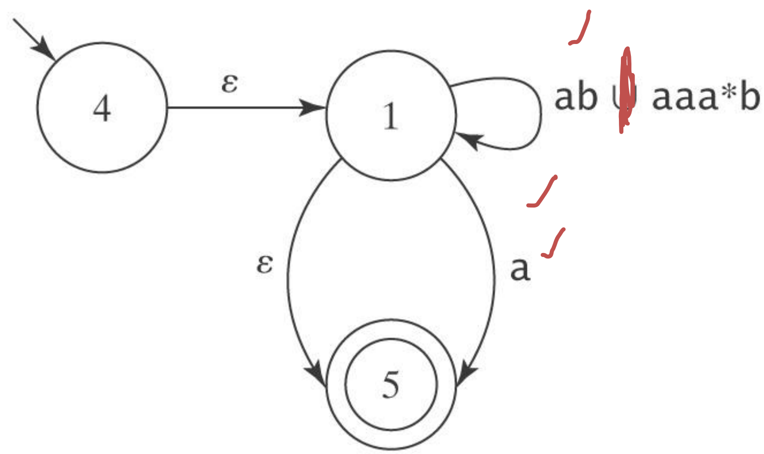
ab|aaa*b

1→2→1 = a(b|aa*b)

1→2→5 = aε = a

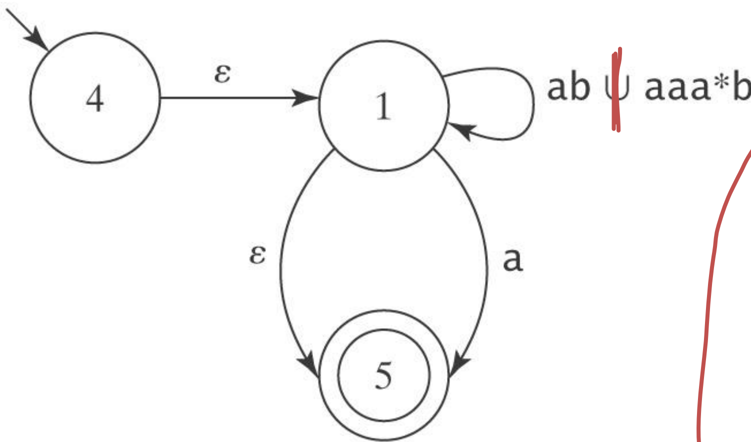


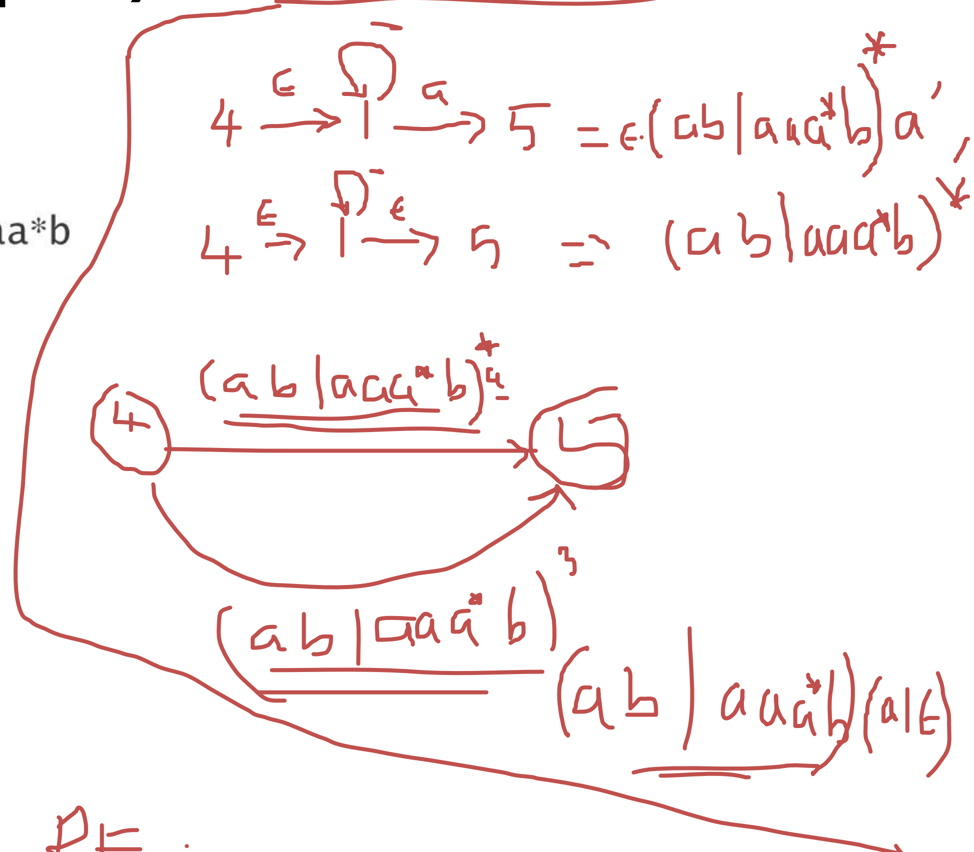
Remove state 2:



An Example, Continued

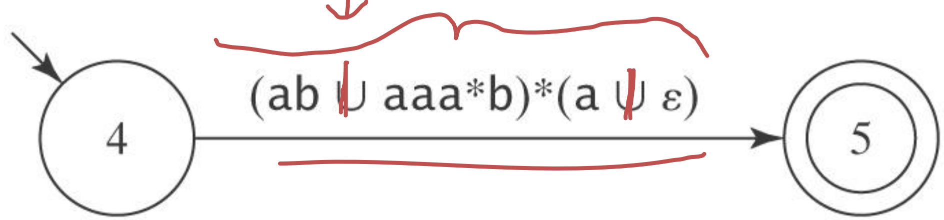
RW



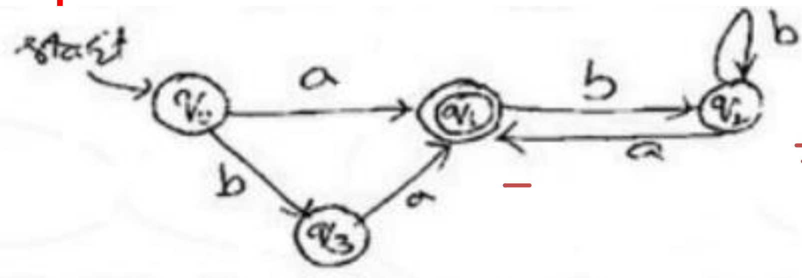


Remove state 1:

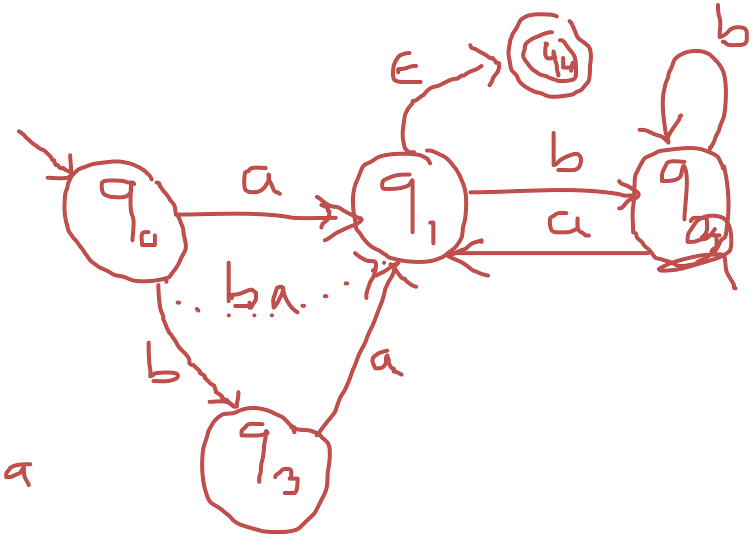
RE..



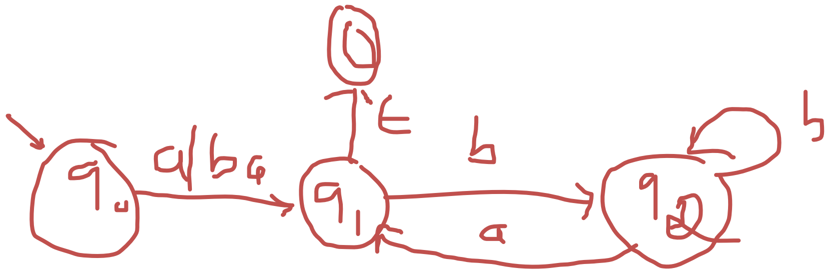
Example-2



$\Rightarrow$



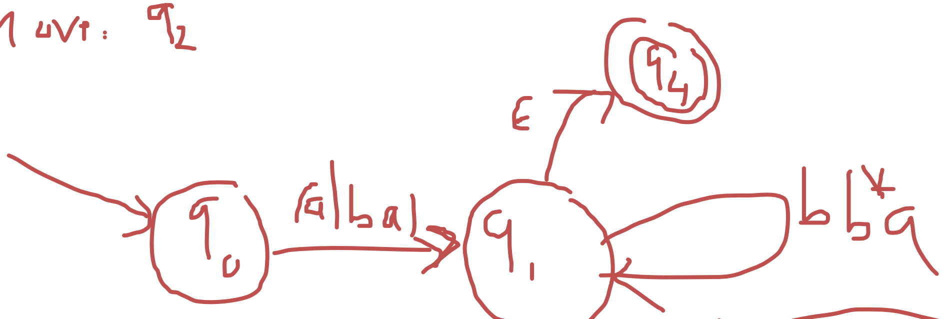
Remove: q_3 $\quad q_0 \xrightarrow{b} q_3 \xrightarrow{a} q_1 : ba$



$q_1 \xrightarrow{b} q_2 \xrightarrow{a} q_1$

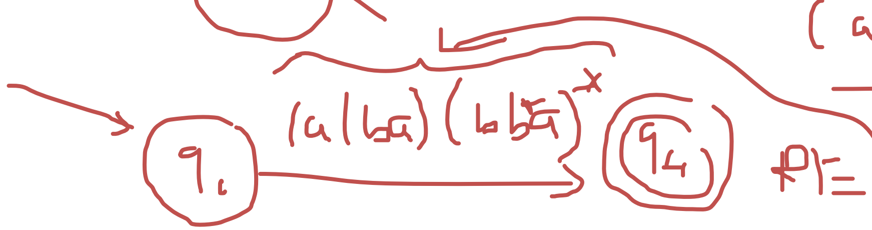
bb^*a

Remove: q_2



$q_0 \xrightarrow{} q_1 \xrightarrow{\epsilon} q_4$

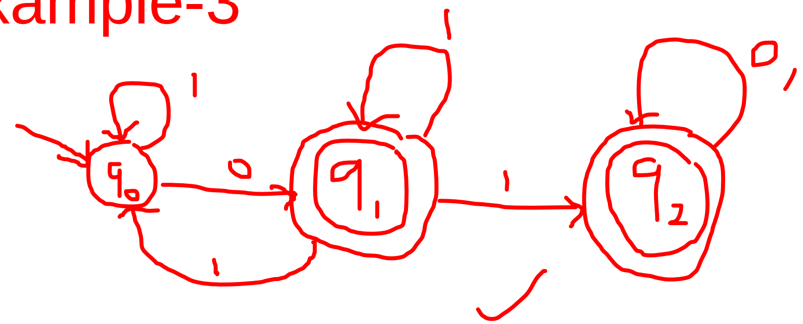
$(a|ba)(bb^*a)^*\epsilon$



Example-3

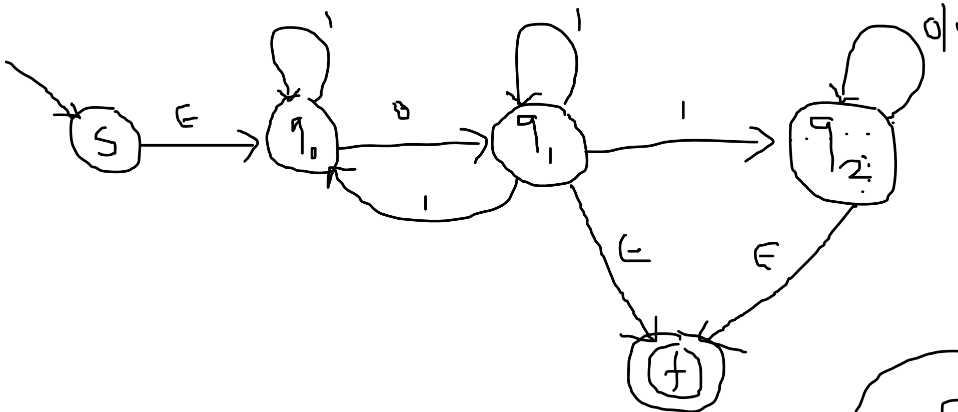
8M FSM → RE

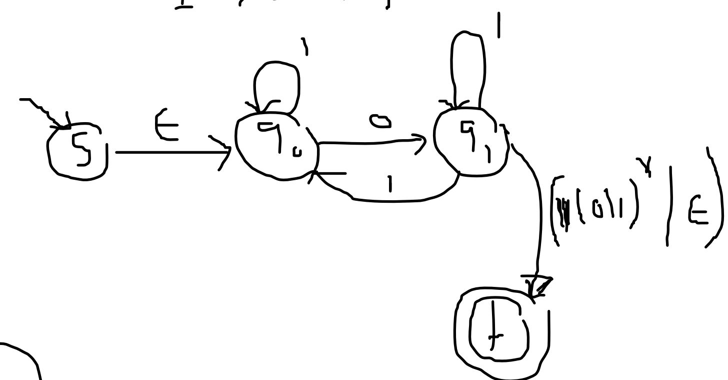
JAN-19



1 Remove q_2:

$1 \xrightarrow{1} 2 \xrightarrow{\epsilon} f$ $1(0|1)^*\epsilon$





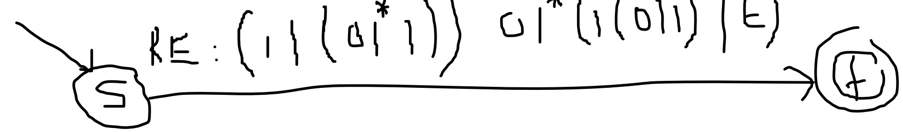
$0 \xrightarrow{0} 1 \xrightarrow{1} 0$: 01^*1

$0 \xrightarrow{0} 1 \longrightarrow f$: $01^*(1(0|1)^* | \epsilon)$

2 Remove q_1 :

3. Remove q_0

RE: $(1|(01^*1))^*\ 01^*(1(0|1)^* | \epsilon)$



Simplifying Regular Expressions

$x + 0 = x$

Regex's describe sets:

- Union is commutative: $\alpha \mid \beta = \beta \mid \alpha$
- Union is associative: $(\alpha \mid \beta) \mid \gamma = \alpha \mid (\beta \mid \gamma)$.
- $\varnothing$ is the identity for union: $\alpha \mid \varnothing = \varnothing \mid \alpha = \alpha$.
- Union is idempotent: $\alpha \mid \alpha = \alpha$.

$\alpha + \emptyset = \alpha$

$\alpha \mid \alpha = \alpha$

Concatenation:

- Concatenation is associative: $(\alpha\beta)\gamma = \alpha(\beta\gamma)$.
- ε is the identity for concatenation: $\alpha\, \varepsilon = \varepsilon\, \alpha = \alpha$.
- $\varnothing$ is a zero for concatenation: $\alpha\, \varnothing = \varnothing\, \alpha = \varnothing$.

$\underbrace{(a \mid b)^*}_{\alpha^*}\ \underbrace{(a \mid b)^*}_{\alpha^*} = \underbrace{(a \mid b)^*}_{\alpha^*}$

Concatenation distributes over union:

- $(\alpha \mid \beta)\, \gamma = (\alpha\, \gamma) \mid (\beta\, \gamma)$.
- $\gamma\, (\alpha \mid \beta) = (\gamma\, \alpha) \mid (\gamma\, \beta)$.

$0^2 = 1$

$0^4 = 1$

$\emptyset^* = \epsilon$

Kleene star:

- $\varnothing^* = \varepsilon$.
- $\varepsilon^* = \varepsilon$.
- $(\alpha^*)^* = \alpha^*$.
- $\alpha^*\alpha^* = \alpha^*$.
- $(\alpha \mid \beta)^* = (\alpha^*\beta^*)^*$.

$\epsilon^4 = \epsilon \cdot \epsilon \cdot \epsilon \cdot \epsilon = \epsilon$

THANK U...

$L = \{ a^n b^m \mid n, m \geq 0 \}$ is RL

$\rightarrow a^* b^*$

Regular and Nonregular Languages

$L = \{ a^n b^n \mid n \geq 0 \}$ is not RL

Module-2

-Showing that a Language is Regular

-Closure Properties of Regular Languages

-Showing that a Language is Not Regular(pumping Lemma)

L, M

$L \cup M$

$\Sigma = \{a, b\}$

FSM

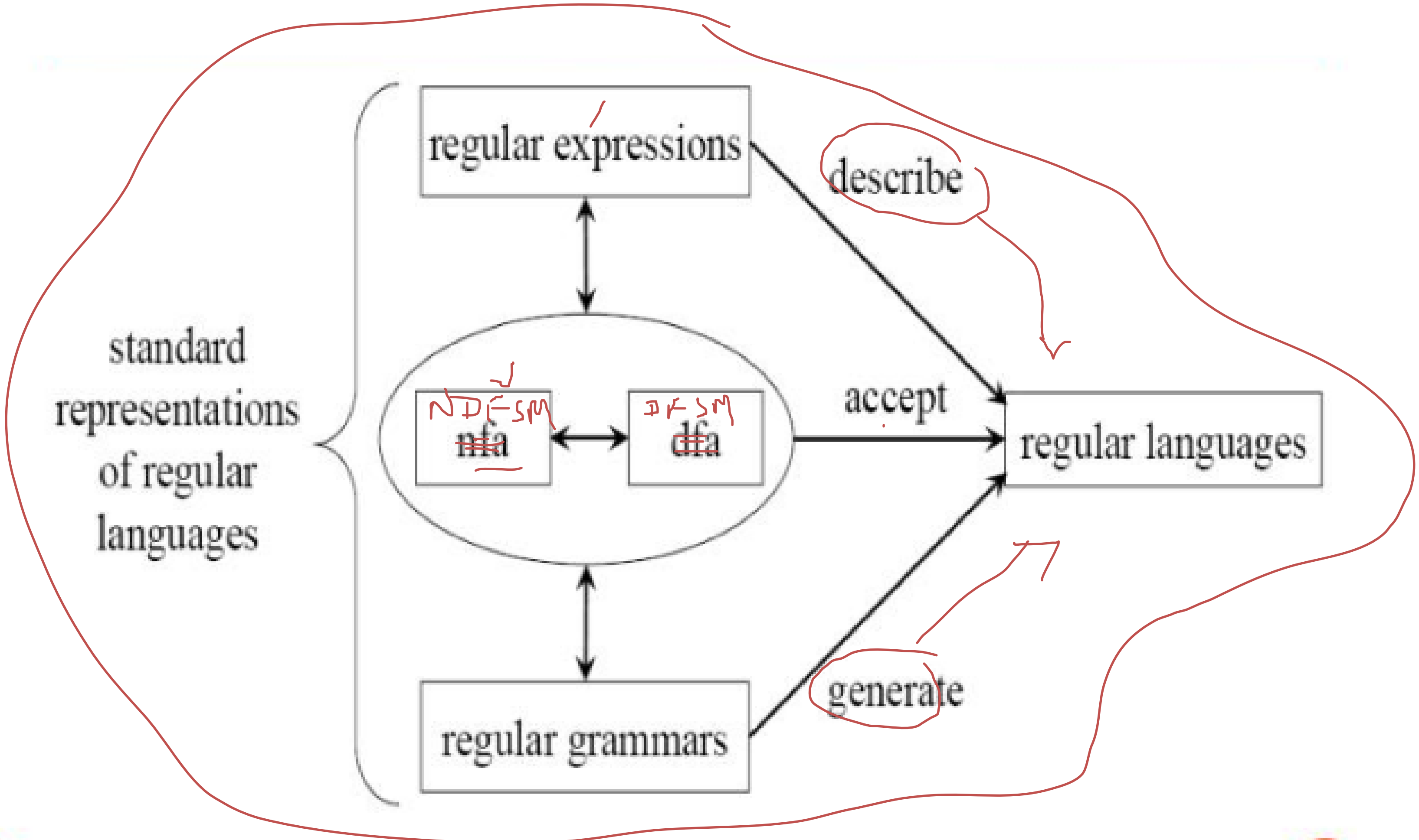
L, L

FSM, RE

RL (L) $\Rightarrow$ FSM | RE | RG

$L \cup M$

$L - M = \emptyset$


regular expressions
describe
standard representations of regular languages
NDFSM
nfa
DFSM
dfa
accept
regular languages
regular grammars
generate

What is a regular Language?

Ex: $L = \{a^n b^m \mid n, m \geq 0\}$

L is Regular → FSM (DFSM, NDFSM), RE, RG

$L = \{s_1, s_2, \ldots, s_n\}$

# of L

Theorem: Every finite language is regular.

Proof: If L is the empty set, then it is defined by the regular expression $\varnothing$ and so is regular. If it is any finite language composed of the strings $s_1, s_2, \ldots s_n$ for some positive integer n, then it is defined by the regular expression:

$$s_1 \cup s_2 \cup \ldots \cup s_n$$

RE: $s_1 | s_2 | \ldots | s_n$

So it too is regular.

Closure Properties of Regular Languages

1. If L and M are regular Languages , so is L U M
2. If L and M are regular Languages, LM is also regular
3. If L is regular, so is L* (Kline star)
4. If L is regular, so is complement of L . $(\overline{L})$
5. If L and M are regular languages , L ∩ M is also regular
6. If L and M are regular languages , L- M is also regular
7. If L is regular, then L^R is also regular(Reversal)
8. if L is regular , so is h(L) < Homomorphism or letter substitution>

L & M

L U M

Set theory

$S = \{a_1, a_2, \dots\}$

$L = \{s_1, s_2, \dots\}$

If L_1 and L_2 are regular , then $L_1 U L_2$, $L_1.L_2$ and L^* also denote the regular Language.

RE, FSM

Proof: It is given that L1 and L2 are regular Languages. So, there exist regular expressions *α* and *β such that*

|, ., * → RE

L1 = L(α)
L2 = L(β)

By the definition of Regular expressions, we have

RL

- α | β is a regular expression denoting the language L1 U L2

RL

- α.β is a regular expression denoting the language L1.L2

RL

- α* is a regular expression denoting the language L1*

so, the regular language are closed under Union, Concatenation and star closure.

proof

$L_1 = \{ s_1, s_2, \ldots \}$ $L_1 \cup L_2 = \{ s_1, s_2 \ldots a_1, a_2 \ldots \}$

~~$L_1 L_2 = \{ s_1 a_1, s_1 a_2, \ldots s_1 a_1 \ldots \}$~~ $L_2 = \{ a_1, a_2, a_3, \ldots \}$

If L is a regular Language , then complement of L is also regular

$\bar{L} = \Sigma^* - L$

DFSM

- Prove that RLs are closed under ***complementation***.
- If L is a regular language over Σ, then $\overline{L} = \Sigma^* - L$ is regular language.
- **Proof:**
  - If L is regular, there exists a DFSM ~~DFA~~ M recognizing L.
  - We can construct a DFA M' for $\overline{L}$ by copying M to M' except that all final states in M are changed to non-final, and all non-final states to final.
- See next slide for a formal proof

$\bar{L} = \{ w \in \{a, b\}^* : w$ do not ab $\}$

$L = \{ w \in \{a, b\}^* \mid w$ end with ab $\}$

L, L

$Q - F = \{0,1,2\} - \{2\} = \{0,1\}$

Let $M_1=(Q, \Sigma, \delta, q_0, F)$ a ~~DFA~~ DFSM that accepts $L(M_1)$

Let $M_2=(Q, \Sigma, \delta, q_0, Q\text{-}F)$ a ~~DFA~~ DFSM that accepts $L(M_2)$

Obviously both languages are regular languages

Per definition 2.2: $L(M) = \{w \in \Sigma^* : \delta(q_0, w) \in F\}$

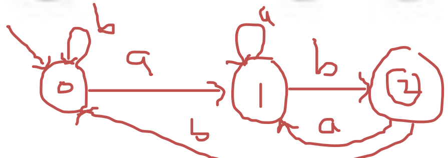
The following are both true

- $w \in \Sigma^* : \delta(q_0, w) \in F \Rightarrow \delta(q_0, w) \notin Q - F$
- $w \in \Sigma^* : \delta(q_0, w) \in Q - F \Rightarrow \delta(q_0, w) \notin F$

Thus $L(M_2) = \overline{L(M_1)}$

$L(M_1)$ is an arbitrary regular language and its complement is also a regular language, therefore the regular languages are closed under complementation

L =



$\bar{L}$ =

$L = \{w \in \{a,b\}^* \mid w \text{ end with } ab\}$

$\bar{L} = \{w \in \{a,b\}^* \mid w \text{ not end in } ab\}$

If L 1 and L2 are regular Languages, then so, is L1 ∩ L2 .

$L_1 = \{aab, babbbab \ldots\}$ $L_2 = \{aab, bba, bab, aaa \ldots\}$
$L_1 \cap L_2 = \{aab, bab \ldots\}$

- Are regular languages closed under **intersection**?
- If L_1 and L_2 are RLs, then $L_1 \cap L_2$ is RL.

DeMorgan's Law

- **Proof:**
  - Since RLs are closed under union and complementation, they are also closed under intersection
  - L_1, L_2 are RLs, so $\overline{L_1}, \overline{L_2}$ are RLs, and $\overline{L_1} \cup \overline{L_2}$ is RL, so $\overline{\overline{L_1} \cup \overline{L_2}}$ is RL.
  - Thus $\overline{\overline{L_1} \cup \overline{L_2}} = L_1 \cap L_2$ is RL.

RL

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$

RL

L_1 FSM$_1$ L_2 FSM$_2$ $L_1 \cap L_2$ FSM$_1 \times$ FSM$_2$

$\therefore L_1 \cap L_2$ is also RL

If L 1 and L2 are regular Languages, then so, is L1 - L2 .

Are regular languages closed under **difference**?
If L_1 and L_2 are RLs, is $L_1 - L_2$ RL? Why?

- $L_1 - L_2 = L_1 \cap \overline{L_2}$
- Since RLs are closed under intersection and complementation, they are also closed under difference.

If L is regular, then L^R is also regular(Reversal)

Proof: We know that L is regular. Let α be a regular expression describing L(α). It is required to prove that there is another regular expression E^R such that:

$$L(\alpha) = (L(\alpha))^R$$

By definition of regular expression, we have:

Regular Expression	(Regular Expression)^R
ε	$\{\varepsilon\}^R = \varepsilon$ → L = {ε}
θ	$\{\theta\}^R = \theta$ → L = { } 0r θ
a (Any i/p symbol)	$\{a\}^R = a$ → L = { a}
α | β	$(\alpha \mid \beta)^R = \alpha^R \mid \beta^R$ → $L(\alpha^R) \cup L(\beta^R)$
α.β	$(\alpha.\beta)^R = \beta^R.\alpha^R$ → $L(\beta^R).L(\alpha^R)$
α*	$(\alpha^*)^R = (\alpha^R)^*$ → $L(\alpha^R)$

From above, its clear that L^R is also regular when L is regular.

If L is regular , so is h(L)

What is homomorphism?

Let Σ and ς are set of alphabets.
The homomorphic function $h: \Sigma \rightarrow \varsigma^*$ is called homomorphism (i.e single letter is replaced by a string)

If $w = a_1a_2a_3....an$ then $h(w) = h(a_1)h(a_2)h(a_3)....$
If $L = \{ w \mid w \in L \}$, then $h(L) = \{ h(w) \mid w \in L \}$

Example:

Let $\Sigma = \{ 0,1\}$, $\varsigma = \{ a,b\}$ and $h(0) = ab$, $h(1) = b$. What is $h(010)$?
If $L = \{ 00, 010\}$ what is $h(L)$?

$h(010) = h(0)h(1)h(0) = abbab$
$h(L) = h(\{00,010\}) = \{ h(00), h(010) \} = \{ h(0)h(0) , h(0)h(1)h(0)\} = \{ abab, abbab\}$

Proof(using Regular Expression):

Let α be the regular expression and $L(\alpha)$ be the corresponding regular Language.

We can easily find $h(\alpha)$ by substituting $h(a)$ for each a in Σ.

By definition of Regular expression, $h(\alpha)$ is a regular expression and $h(L)$ is regular language. So, the regular language is closed under homomorphism.

Example: $\Sigma = \{ a,b\}$, $\varsigma = \{ 0,1)$ and $h(a) = 00$, $h(b) = 10$

Suppose $\alpha = (a|b)^* ab$, describe the $L = \{ w \in \{a,b\}^* \mid w$ ends with $ab\}$

$h(\alpha) = h((a|b)^*ab) = h((a|b)^*)\ h(ab) = (h(a)|h(b))^*h(a)h(b)$

$= (00|10)^*\ 0010 \rightarrow$ describe the Language $h(L)$

Proving Languages Not to Be Regular

Some languages are *not* regular

When is a language is regular?

if we are able to construct one of the following:

DFSM or NDFSM or RE or RG

When is it not regular?

If we can show that no FSM can be built for a Language.

How to prove languages are ***not*** regular?

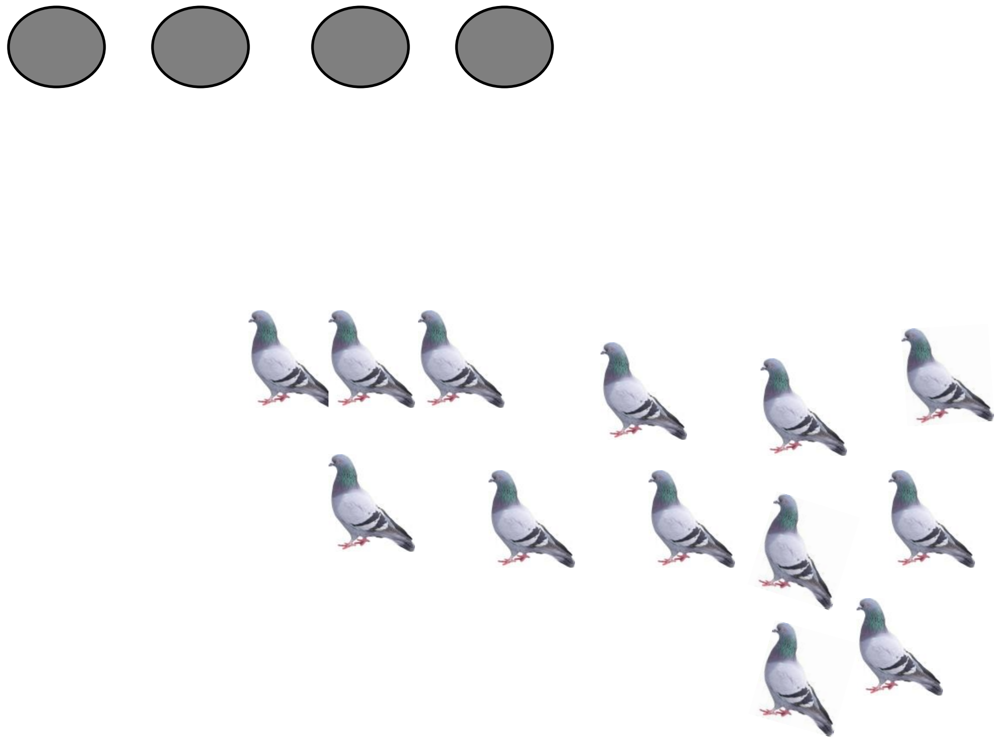
What if we cannot come up with any FSM?

A) Can it be language that is not regular?

B) Or is it that we tried wrong approaches?

How do we *decisively* prove that a language is not regular?

Pigeon Hole Principle



Pumping Lemma for Regular Languages

Statement:

Let L be a regular language. Then there exists a constant n (which depends on L) such that for every string w in L such that $|x| \geq n$, we can break w into three strings $x = uvw$ such that:

1. $v \neq \boldsymbol{\varepsilon}$
2. $|uv| \leq n$
3. For all $k \geq 0$, the string $u(v)^k w$ is also in L.

Pumping Lemma: Proof

- L is regular => it should have a DFSM.
  - Set n := Number of states in the DFSM
- Any string $x \in L$, such that $|x| \geq n$, should have the form: $x = a_1 a_2 \ldots a_m$, where $m \geq n$

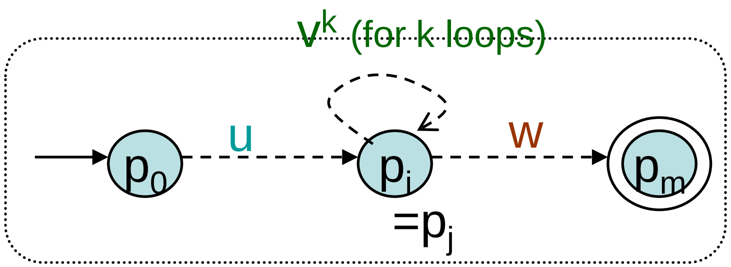
➢ => We should be able to break x = uvw as follows:

  ➢ $u = a_1 a_2 .. a_i$ $v = a_{i+1} a_{i+2} .. a_J$; $w = a_{J+1} a_{J+2} .. a_m$

  ➢ u's path will be $p_0 \ldots p_i$

  ➢ v's path will be $p_i\, p_{i+1} .. p_J$ (but $p_i = p_J$ implying a loop)

  ➢ w's path will be $p_J p_{J+1} .. p_m$



➢ Now consider another string $x_k = u(v)^k w$, where $k \geq 0$

➢ Case: k = 0

  ➢ DFSM will reach the accept state p_m

➢ Case: k > 0

  ➢ DFSM will loop for v^k, and finally reach the accept state p_m for w

➢ In either case, $x \in L$ (This proves the lemma)

Applications of the Pumping Lemma

The pumping lemma is a very powerful tool and has the following applications';

1. It is used to prove that certain Languages are non-regular.

2. It can be used to check whether a language accepted by FSM is finite or infinite

The General Strategy used to prove that Language is Not Regular

Step 1: Assume that the Language L is regular.

Step 2: Select the string x such that $|x| \geq n$ *and break it into 3 substrings u,v and w so that x = uvw with the constraints:* $v \neq \varepsilon$ **&** $|uv| \leq n$.

$|v| \geq 1$

≥ 0

Step 3: Find any k such that $u(v)^k w \notin L$.

(According to pumping lemma, $uv^k w$ is in L for any $k \geq$ *0. so the result is contradiction to our assumption. Hence given L is not regular)*

Examples

Show that $L = \{ a^n b^n \mid n \geq 0 \}$ is not a regular

1) Assume that L is Regular.

2) Select $x = uvw \in L$, split x, such that $|v| \neq \epsilon$, $|uv| \leq n$

Let $x = a^n b^n \in L$, $|x| = 2n \geq n$

we can break it follow:

$\underbrace{aa \dots a}_{n} \underbrace{b \dots b}_{n}$

$\underbrace{a^{n-1}}_{u} \underbrace{a}_{v} \underbrace{b^{n}}_{w} \in L$, $v \neq \epsilon$ $(|v| = 1)$ & $|uv| \leq n$, $|u| = n-1$, $|v| = 1$

3) Choose $k = 0$, resulting string is:

$a^{n-1} (a)^{k=0} b^n = a^{n-1} b^n \in L$ (According to pumping Lemma)

$k = 2$ $\quad n-1+2 = n+1$

$a^{n-1} (a)^2 b^n = a^{n-1} a^2 b^n = a^{n+1} b^n \notin L$

But $a^{n-1} b^n \notin L$

$\therefore$ L is not a Regular //

Prove that $L = \{ a^i b^j \mid i > j \}$ is not a regular.

Step 1: Assume that L is regular and n is some constant integer.

$3 > 2 \in L$

Step 2: Select $x = a^{n+1}b^n$, since $|x| = 2n+1 \geq n$, we can split x into uvw

such that $|uv| \leq n$ and $v \neq \varepsilon$ as shown below.

$x = a^{n+1}b^n = a^n ab^n = \underbrace{a^{n-1}}_{u}\ \underbrace{a}_{v}\ \underbrace{ab^n}_{w}$, where $u = a^{n-1}$, $v = a$, $w = ab^n$

Step 3: According to pumping lemma, $a^{n-1}(a)^k ab^n \in L$ **for any $k \geq 0$**

if we choose k=0,

The resulting string becomes: $a^{n-1} ab^n = a^n b^n \notin L$.

Therefore , L is not regular.

Show that $L = \{ a^n b^l \mid n \neq l \}$ is not regular.

$\overline{n+1} \neq \overline{n}$

$x = a^{n+1} b^n \in L$

-Proof is same as previous Language…

$a^n b^n \notin L$

ababb

{aabbb}

Show that $L = \{ w \mid n_a(w) < n_b(w) \}$ is not regular.

$|a^n| < |b^{n+1}|$

$x = a^n b^{n+1} \in L, \quad n+1 > n$

$a^{n-1} a b^{n+1} \in L$

$\Downarrow$

$a^{n-1}(a)^k b^{n+1} \in L, \; k \geq 0$

$\Downarrow \; k = 2$

$a^{n-1}(a)^2 b^{n+1}$

$a^{n-1+2} b^{n+1}$

$\|$

$a^{n+1} b^{n+1} \notin L$

L is not RL

~~*-Proof same as previous Language*~~*…*

$x = a^{n-1} b^n \in L$

Prove that $L = \{ ww^R \mid w \in \{0,1\}^* \}$ is not a regular.

$w = \underbrace{100}_{w}\underbrace{001}_{w^R}$ ww^R

1. Assume that L is regular and n is some integer constant.

2. Consider the string $x = \underbrace{1^n0^n}_{w}\underbrace{0^n1^n}_{w^R}$, since $|x| = 4n \geq n$, we can split x into uvw such that $|uv| \leq n$ and $v \neq \varepsilon$ as shown below:

$x = \underbrace{1^{n-1}}_{u}\ \underbrace{1}_{v}\ \underbrace{0^n0^n1^n}_{w}$, where $u = 1^{n-1}$, $v = 1$ and $w = 0^n0^n1^n$

x

3. According to pumping lemma, $1^{n-1}(1)^k 0^n0^n1^n \in L$ for $k \geq 0$,

if we can choose k=0, then $ww^R \notin L$

the resulting string become: $1^{n-1}0^n0^n1^n \notin L$.

Therefore , L is not regular

$1111000000000001111 \in L$

$\Sigma = \{a\}$

0! = 1, 1, 1, 2, 6, 24,
3! = 3×2×1 = 6
4! = = 24

Show that $L = \{a^{n!} \mid n \geq 0\}$ *is not a regular.*

1. Assume that L is regular and n is integer constant.

2. Consider the string $x = a^{n!}$ $\in L$, since $|x| = n! \geq n$, we can split x into uvw such that $|uv| \leq n$ and $v \neq \varepsilon$ as shown below:

   i.e. $x = \underbrace{a^i}_{u}\ \underbrace{a^j}_{v}\ \underbrace{a^{n!-i-j}}_{w}$, where $u = a^i$, $v = a^j$ and $w = a^{n!-i-j}$

3. According to pumping lemma, $a^i\ (a^j)^k\ a^{n!-i-j} \in L$ for $k \geq 0$

   n = 3

   if we choose k=0, it means that :

   $$a^i\ (a^j)^k\ a^{n!-i-j} = a^i\, a^{n!-i-j} = a^{n!-j} \in L$$

   It is clear that: $n! > (n!-j) < (n+1)!$ [take j = 1]

   $n! > (n!-1) < (n+1)!$
   $3! > (3!-1) \leq 4!$
   $6 > \boxed{5} < 24$

   Since (n!-1) lies between factorial of n! and (n+1)!

   ➔implies that $\mathbf{a^{n!-1} \notin L}$

Therefore , L is not regular.

aa, aaa, aaaaa, aaaaaaa

12

3x4

2, 3, 5, 7, 11, 13

Show that the language $L = \{a^p \mid p \text{ is a prime number}\}$ is not regular.

1. Assume that L is regular and n is some integer constant

2. Select string $x = a^n \in L$ where n is prime. Since $|x| = n$, so we can break x into x= uvw such that $|v| \neq \varepsilon$ and $|uv| \leq n$ as shown below.

$$x = a^n = \underbrace{a^i}_{u}\ \underbrace{a^j}_{v}\ \underbrace{a^{n-i-j}}_{w} \in L$$

where $|u| = i$, $|v| = j \geq 1$ and $|uv| = i+j \leq n$

3. According to Pumping lemma, $u(v)^k w \in L$ for k =0,1,2….

i.e. $a^i\,(a^j)^k\,a^{n-i-j} \in L$

i.e. $i+jk+n-i-j = n+j(k-1)$ is prime for all $k \geq 0$

$n + j(n+1-1)$

$n + jn = n(j+1)$

Now, if we choose **k = n+1**, then

$n + j(k-1) = n+jn = n(j+1)$ must be a prime.

$a^{n(j+1)} \notin L$

which is a contradiction(because prime number can not be factored)

so, n(j+1) is not a prime.

Therefore , L is not Regular

$n_a(w) < n_b(w)$

~~$n_a(w)$~~ $> n_b(w)$

$n_a(w) = n_b(w)$

$a^n b^n$

Home Work

$n_a(w) = n_b(w)$

$x = 0^n 1^n \in L$

1.Show that L = {w ∈ {0, 1}* : the number of 0s in w equals the number of 1s in w} is not a regular using pumping Lemma.

2. Prove that L = {ww : w ∈ { 0, 1} } is not a regular Language.

100100

$0^n 1^n 0^n 1^n$

$x = 1^n 0^n 1^n 0^n \in L$

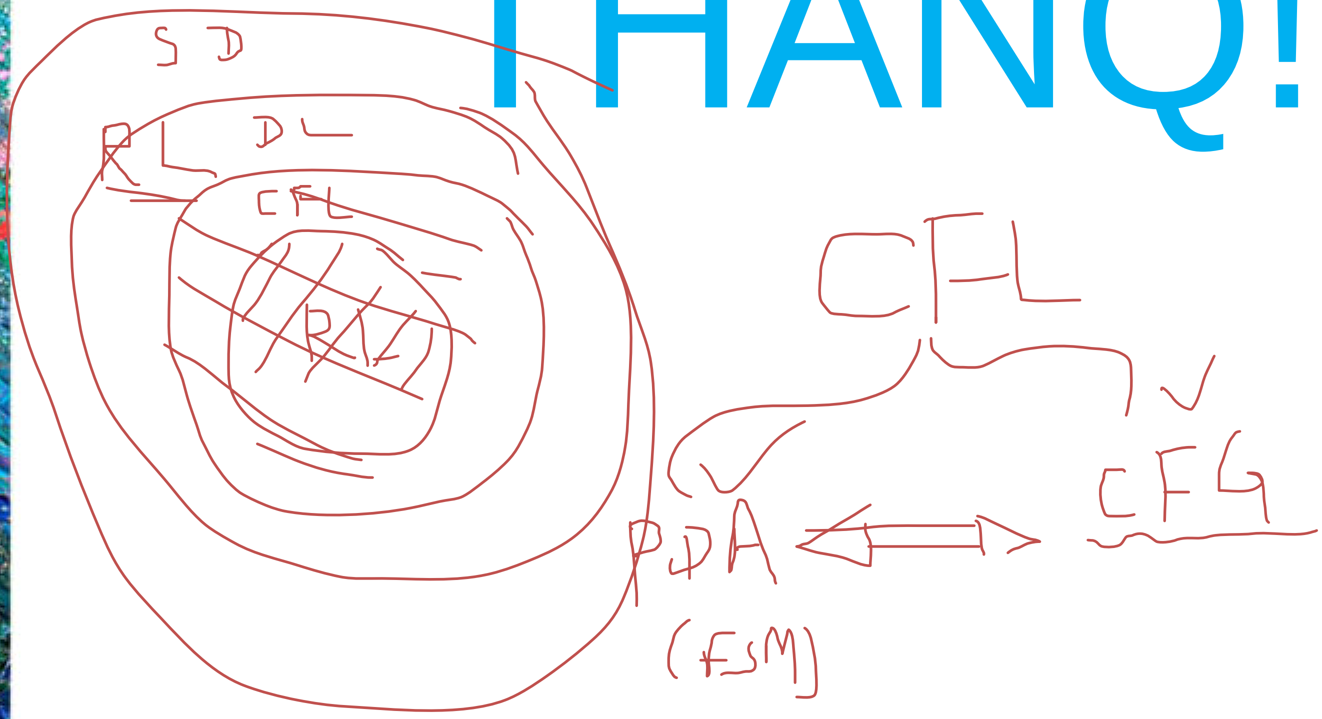
3. Show that L = $\{0^m 1^n : m > n \geq 0\}$ is not a regular.

similar to: L = $\{ a^i b^j \mid i > j \}$ is not regular (ref. : slide No. 23)

$k = 0$

$\notin L$

THANQ!



M-3

FSM
DFSM
NDFSM
~~ε-NDFSM~~
GR
RL
RE →
RG →

Regular Grammars

(also called right-Linear Grammars)

$M = (Q, \Sigma, \delta, q_0, F)$

Definition: A regular grammar G is a 4-tuples

G = (V, T,P,S), where

A B

$S \Rightarrow abbb$

V: Set of Not-terminal symbols also called Variables. $V \cup T$

A, B, ...

G S

T: Set of Terminal symbols (Σ) $T = \Sigma$ $A \to a$

P: Set of productions or rules of the form:

NT — **A → a *or* A → ε *or* A → aB**

where a ∈ T is terminal symbol and A,B ∈ V, are variables

S: The start symbol (Non-terminal)

** Regular grammar G describes a regular Language denoted by L(G)*

Simple Example: Regular Grammar to accept all string with any number of a's

a^*

RG

$w : aaa \in L$

$P : \{ S \to aS \mid \epsilon \}$

$S \Rightarrow aS \Rightarrow aaS$
$\Rightarrow aaaS$
$\Rightarrow aaa \cdot \epsilon = aaa$
$(A \cdot \epsilon = A)$

$G = (V, T, P, S)$

$V = \{ S \}$

$T = \{ a \}$

S – start symbol

$G = (\{S\}, \{a\}, P, S)$

Examples contd..

1.Obtain a Regular grammar to generate all strings of a's and b's including empty string.

$S \to aS \mid bS \mid \varepsilon$

w = aba

$S \Rightarrow aS \Rightarrow abS \Rightarrow abaS$

$\Rightarrow ab a \varepsilon L$

P : $S \to aS$

$S \to bS$

$S \to \varepsilon$

V, T, P, S

Therefore G = ({S}, {a,b}, P,S)

2. Obtain a Regular Grammar to accept all strings of 0's and 1's ending 01

$(0|1)^* 01$

$A \to a \mid \varepsilon \mid aB$

P : $S \to 0S \mid 1S \mid 0A$

$A \to 1$

w : 10101 $\in L$

$S \Rightarrow 1S \Rightarrow 10S \Rightarrow 101S \Rightarrow 1010A$

$\Rightarrow 10101 \in L$

(Accept)

$G = (\{S, A\}, \{0, 1\}, P, S)$

3. Design a Regular grammar $L = \{ w \in \{ a , b\}^* : |w| \text{ is even} \}$

$\Sigma = \{a, b\}$

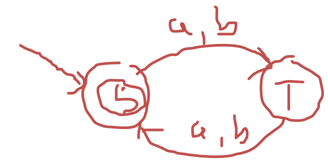
$((a|b)(a|b))^*$

G:

S → aT | bT | ε

T → aS | bS

Generate w = ababbb



$S \Rightarrow aT \Rightarrow abS \Rightarrow abaT$

$\Rightarrow ababS$

$\Rightarrow ababbT$

$\Rightarrow ababbbS \Rightarrow \underbrace{ababbb}_{6} \in L$

$|abbb| = 4$

4. Obtain a Regular grammar to accept all strings of a's and b's that begin with a and end with b.

RE: $a(a|b)^*b$

P:

$S \rightarrow aA$

$A \rightarrow aA \mid bA \mid b$

5. Let $L = \{w \in \{a, b\}^* : w \text{ ends with the pattern } aaaa\}$

$A \to a,\ A \to \epsilon,\ A \to aB$

$(a|b)^*\ aaaa$

Regular Grammar for L:

$S \to aS$	/* An arbitrary number of a's and b's can be generated before the
$S \to bS$	pattern starts.
$S \to aB$	/* Generate the first a of the pattern.
$B \to aC$	/* Generate the second a of the pattern.
$C \to aD$	/* Generate the third a of the pattern.
$D \to a$	/* Generate the last a of the pattern and quit.

$A \to aB$

RLG

$A \to Ba$

LLG

G:

$S \to aS \mid bS \mid aB$

$B \to aC$

$C \to aD$

$D \to a$

$V = \{S, B, C, D\}$

$T = \{a, b\}$

$P =$

S

$G = (V, T, P, S)$

Finite State Machine → Regular Grammars

$L(M) = L(G)$

For every FSM M , there exists a Regular Grammar G, such that L(M) =L(G)

1. Assume that is M DFSM (if not convert it to DFSM)

   S

   Construct G = (V,T,P,S) from M as follows:

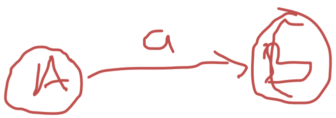
2. Create a Nonterminal for each state in the M.

3. The start state of M becomes the starting Nonterminal for G

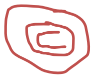
   $S \to aT \mid bT$

4. For each transition δ(A , a) = B, add a production A→aB to G

5. For each accepting state A, add a production A→ ε to G



$\delta(A, a) \to B$



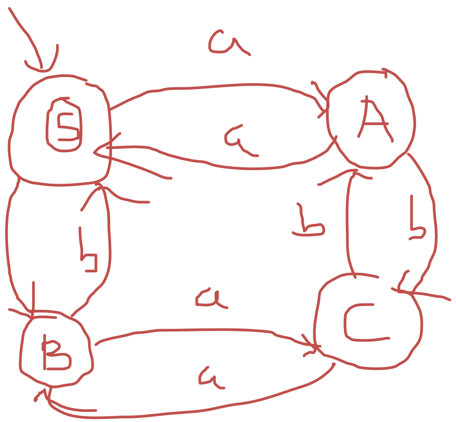
$C \to \varepsilon$

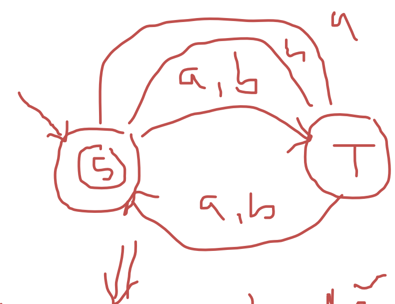
ε-prod

Examples

odd → HW

1. L = {*w* ∈ {*a*, *b*}* : *w* contains an even number of *a*'s and an even number of *b*'s}.

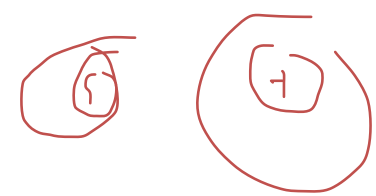




$S \to aT \mid bT \mid \epsilon$ $L = \{w \in \{a,b\}^* : |w|$ is even$\}$

$T \to aS \mid bS$

$$G: \begin{cases} S \to aA \mid bB \mid \epsilon \\ A \to aS \mid bC \\ B \to bS \mid aC \\ C \to bA \mid aB \end{cases}$$

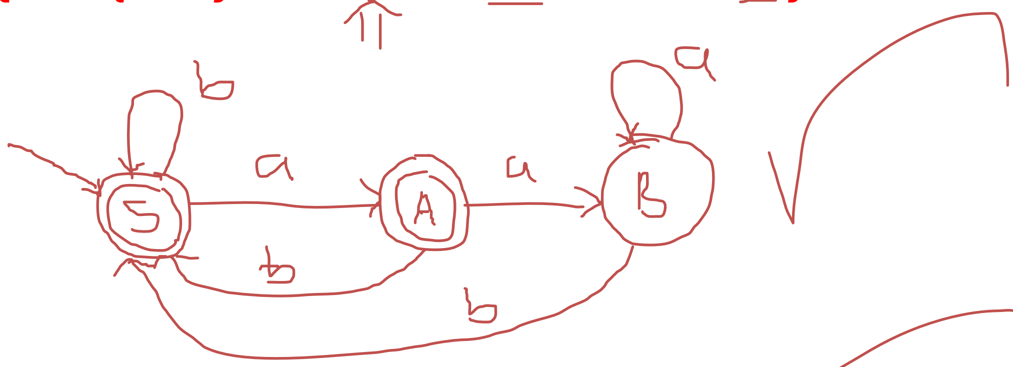


$S \to \epsilon$

$T \to \epsilon$

$G: (\{S, A, B, C\}, \{a, b\}, P, S)$

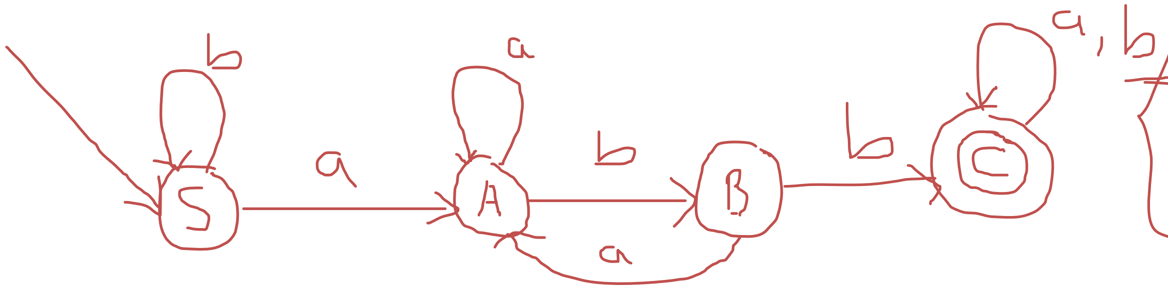
2. L = {*w* ∈ {*a*, *b*}* : *w does not end in aa*}.



$$G: \begin{cases} S \to aA \mid bS \mid \epsilon \\ A \to aB \mid bS \mid \epsilon \\ B \to aB \mid bS \end{cases} P$$

$$G = \{(S, A, B\}, \{a, b\}, P, S\}$$

3. L = {*w* ∈ {*a*, *b*}* : *w* contains the substring abb}.



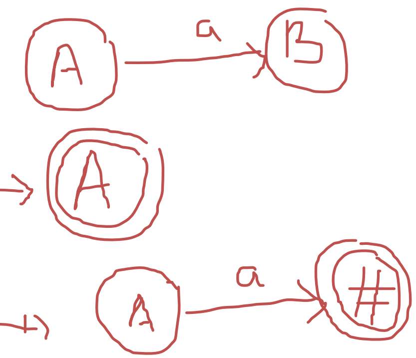
$$\begin{cases} S \to \\ A \to \\ B \to \\ C \to aC \mid bC \mid \varepsilon \end{cases}$$

$$G = (V, T, P, S)$$

Regular Grammars → FSM

For every Regular Grammar G , there exists a FSM M, such that L(G) = L(M)

$A \to a$, $A \to \varepsilon$, $A \to aB$



1.Convert the Following RG to FSM

$S \to aT$

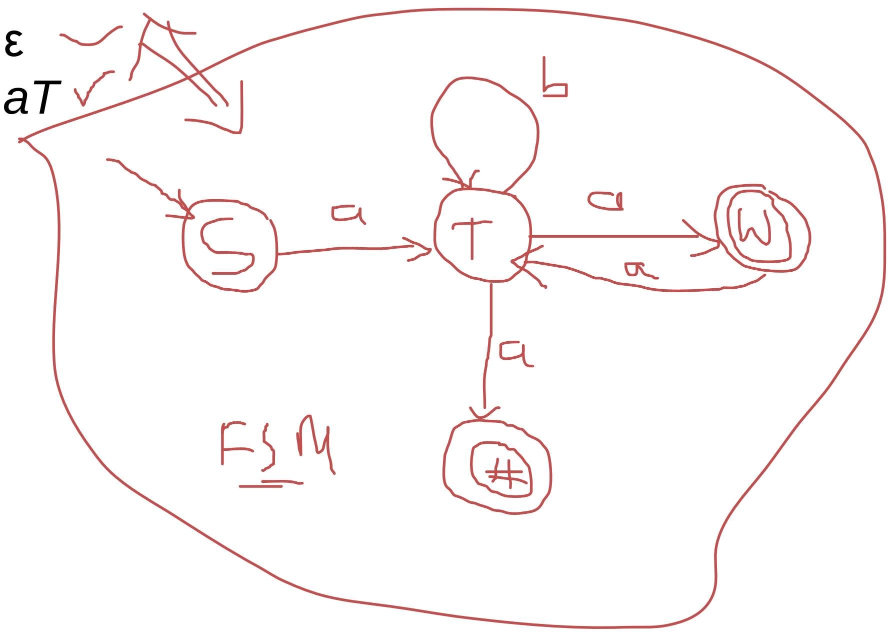
$T \to bT$

$T \to a$

$T \to aW$

$W \to \varepsilon$

$W \to aT$

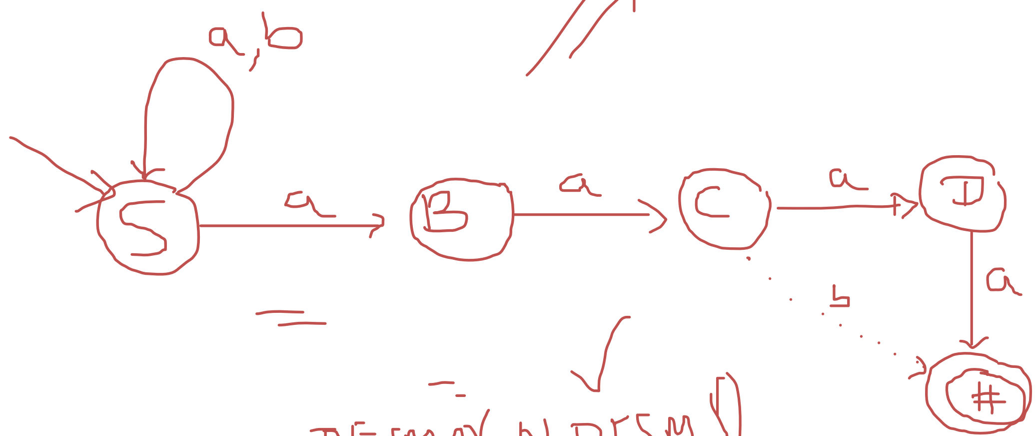


2.Convert the Following RG to FSM

$$S \rightarrow aS$$
$$S \rightarrow bS$$
$$S \rightarrow aB$$
$$B \rightarrow aC$$
$$C \rightarrow aD$$
$$D \rightarrow a$$

$C \rightarrow b$

DFSM



DFSM or NDFSM?

THANQ!

Module-3

Topic: Context-Free Grammars

Content

-Introduction to Grammars, CFGs and languages
-Designing CFGs, simplifying CFGs
-Derivation and Parse trees
-Ambiguity, Examples
-Techniques for reducing ambiguity from Grammars
- Normal Forms(CNF, GNF)

RE + CFG √ = compiler Design CFL = Language

Introduction to CFG: Informal Comments

- A context-free grammar is a notation for describing languages.

Syntax Analysis Syntax Errors:

lexeme

word

FSM

- It is more powerful than finite automata or RE's, but still cannot define all possible languages.
- Useful for nested structures, e.g., parentheses in programming languages.

NT

- Basic idea is to use "variables" to stand for sets of strings (i.e., languages).

L = { .. }

- These variables are defined recursively, in terms of one another.

S → aS | ε RG

TM

Definition of Context-Free Grammars

A context-free grammar (CFG) G, is defined by a 4-tuples as:

G = (V,T,P,S) (V, Σ, P, S) Σ

$\Sigma = \{a, b\}$

$T = \{a, b\}$

Where,

V: is the final set of a Non-terminal(Variables) symbols.

T: is the ~~final~~ finite set of a terminal symbols (Σ)

P: is a set of production rules of the form **A → α**

where A is single *Nonterminal* symbol and α is a string of Zero or more *Terminals* & *Nonterminal* symbols

$\langle head \rangle \rightarrow \langle body \rangle$

$A \rightarrow \alpha$

$A \rightarrow aBbS$

$A \rightarrow \epsilon$

S: is the start symbol which is used to derive/generate the string belongs to the Language and represents the Language being defined.

$A \rightarrow a$ or $A \rightarrow aB$ or

$A \rightarrow \epsilon$

T={0,1}

Example: A Context-Free Grammar for Palindromes

P

S→0S0
S→1S1
S→0
S→1
S→ε

Heart of G

w=0110

w=1001 ∈ L

Formally, the grammar is represented by:

G =({S}, {0,1}, P, S)

V T

0 ∈ L
1 ∈ L
ε ∈ L
w ∈ L

Notation for CFG

a, b, 1 ← T   +, −, ∗, / → T
0, 1, ⋯ 9 ← T,   (,), {, }, if, else → Terminal

1. Lower-case letters near the beginning of the alphabet, a, b, and so on, are terminal symbols. We shall also assume that digits and other characters such as + or parentheses are terminals.

2. Upper-case letters near the beginning of the alphabet, A, B, and so on, are variables (NT)

3. Lower-case letters near the end of the alphabet, such as w or z, are strings of terminals. This convention reminds us that the terminals are analogous to the input symbols of an automaton.

4. Upper-case letters near the end of the alphabet, such as X or Y, are either terminals or variables.

5. Lower-case Greek letters, such as α and β, are strings consisting of terminals and/or variables. (NT)

   Ex: $\alpha = aBbS$ → grammar symbol

6. The productions $A \to \alpha_1$, $A \to \alpha_2, \ldots, A \to \alpha_n$ can be replaced by the notation $A \to \alpha_1 | \alpha_2 | \cdots | \alpha_n$. $(A \to \beta)$

$A \Rightarrow$
$A_1, A_2, \ldots A_n$

$w = 101$
$z = abab$
$w = (\{()\})$

$X = a$
$X = S$
$Y = A$
$Z = 0$
X_1, X_2, X_3

$S \to 0S0$
$S \to 1S1$
$S \to 0$
$S \to 1$
$S \to \epsilon$

$S \to 0S0 \mid 1S1 \mid 0 \mid 1 \mid \epsilon$ (S-Production)

The Language of a Grammar

If $G = (V, T, P, S)$ is a CFG, the *language* of G, denoted $L(G)$, is the set of terminal strings that have derivations from the start symbol. That is,

$$L(G) = \{w \text{ in } T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$$

If a language L is the language of some context-free grammar, then L is said to be a *context-free language*, or CFL.

For Instance, Consider the set of rules or productions below:

$S \rightarrow 0S0$
$S \rightarrow 1S1$
$S \rightarrow 0$
$S \rightarrow 1$
$S \rightarrow \varepsilon$

Above grammar, defined the Language of Palindromes over alphabet {0,1}. Thus, the set of palindromes is a CFL.

Designing of CFG: Problems

Design the Context free grammar for the Language L = { $0^n 1^n$ | n ≥ 0 }

Design the Context free grammar for the Language L = { w∈ { (,)}* : the parentheses are balanced }

P =

S→ (S) | SS | ε

Derivation and Parse trees

Derivation:

A process of obtaining string of terminals and/or Non-Terminals from the start symbol by applying some or productions is called derivation.

Ex.

A parse tree of a derivation is a tree in which:

- Each internal node is labeled with a nonterminal
- If a rule $A \rightarrow A_1A_2\ldots A_n$ occurs in the derivation then A is a parent node of nodes labeled $A_1, A_2, \ldots, A_n$

Leftmost, Rightmost Derivations

Definition. A **left-most derivation(LMD)** of a sentential form is one in which rules transforming the left-most Nonterminal are always applied.

$$
\begin{aligned}
S &\rightarrow A \mid A\,B \\
A &\rightarrow \varepsilon \mid \mathbf{a} \mid A\,\mathbf{b} \mid A\,A \\
B &\rightarrow \mathbf{b} \mid \mathbf{b}\,\mathbf{c} \mid B\,\mathbf{c} \mid \mathbf{b}\,B
\end{aligned}
$$

In LMD, always pick up leftmost Nonterminal

$S \Rightarrow AB \Rightarrow AAB \Rightarrow \mathbf{a}AB \Rightarrow \mathbf{aa}B \Rightarrow \mathbf{aab}B \Rightarrow \mathbf{aabb}$ **(using LMD)**

→ sentence | string

$S \underset{lmd}{\overset{*}{\Rightarrow}} aabb$

S form

Definition. A **right-most derivation(RMD)** of a sentential form is one in which rules transforming the right-most Nonterminal are always applied.

$S \Rightarrow AB \Rightarrow A\mathbf{b}B \Rightarrow A\mathbf{bb} \Rightarrow AA\mathbf{bb} \Rightarrow A\mathbf{abb} \Rightarrow \mathbf{aabb}$

In RMD, always pick up Rightmost Nonterminal

S form

$S \underset{rmd}{\overset{*}{\Rightarrow}} aabb$

Parse Trees(Example)

Derivation(LMD / RMD)

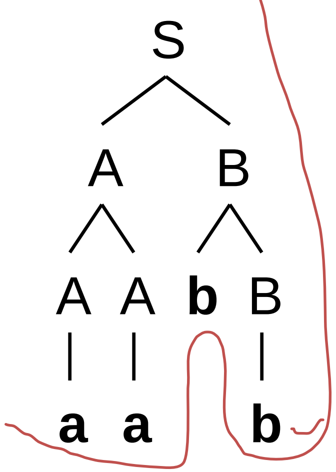
$$
\begin{aligned}
S &\rightarrow A \mid A\,B \\
A &\rightarrow \varepsilon \mid \mathbf{a} \mid A\,\mathbf{b} \mid A\,A \\
B &\rightarrow \mathbf{b} \mid \mathbf{b}\,\mathbf{c} \mid B\,\mathbf{c} \mid \mathbf{b}\,B
\end{aligned}
$$

w = **aabb**

(LMD)

$S \Rightarrow AB \Rightarrow AAB \Rightarrow \mathbf{a}AB \Rightarrow \mathbf{aa}B \Rightarrow \mathbf{aab}B \Rightarrow \mathbf{aabb}$

aabb (Yield)



Parse tree

Consider the grammar G with Productions.

S → AbB

A → aA | **ε**

B → aB | bB | **ε**

Obtain LMD, RMD and Parse tree for the string : aaabab

$x \cdot \epsilon = \epsilon \cdot x = x$

~~S ⇒ AbB ⇒ aAbB~~ ⇒ abB ⇒ abbB ⇒ abb ∈ L

LMD

S ⇒ AbB

⇒ aAbB

⇒ aaAbB

⇒ aaaAbB

⇒ aaabB . {A → ε}

⇒ aaabaB {B → aB}

⇒ aaababB

⇒ aaabab {B → ε}

∈ L

RMD

S ⇒ AbB

⇒ AbaB

⇒ AbabB

⇒ Abab

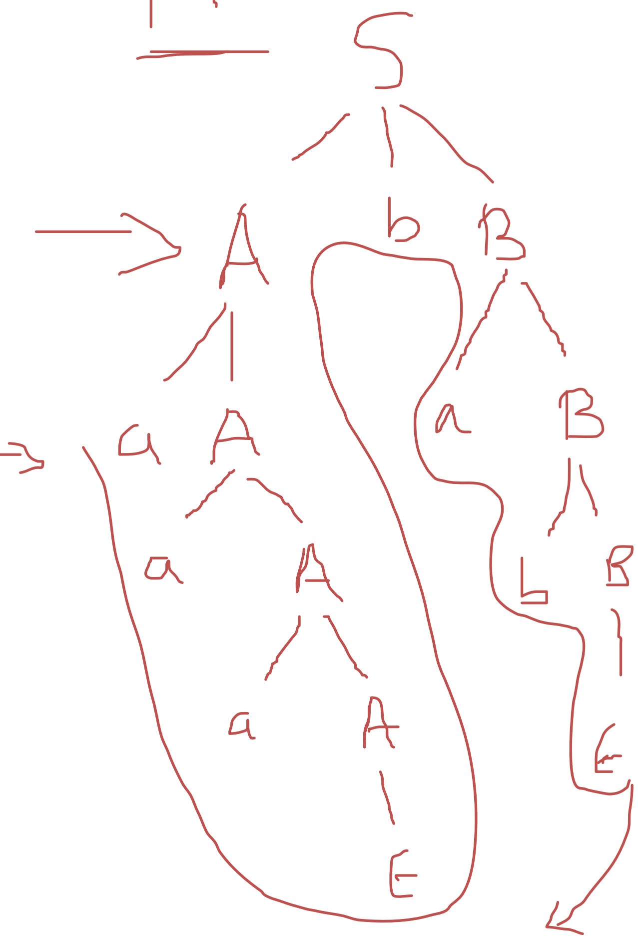
⇒ aAbab

⇒ aaAbab

⇒ aaaAbab

⇒ aaabab ∈ L

PT



Yield

Designing of CFG: Problems contd...

A B ε,1 m=0, n=2

Obtain a grammar for L ={ $0^m 1^m 2^n$ | m ≥ 0 , n ≥ 0 } and also give LMD , RMD and Parse tree for the string w = 0011222.

≥1 , n ≥1 w=012 w= 0011222

$S \to AB$

$A \to 0A1 \mid \epsilon$

$B \to 2B \mid \epsilon$

$S \to AB$

$A \to 0A1 \mid 01$

$B \to 2B \mid 2$

$S \Rightarrow AB$

$\Rightarrow 01B \Rightarrow 012$ ε L

Design a grammar for $L = \{ a^{n+1}b^n : n \geq 0 \}$

$S \rightarrow aSb \mid a$

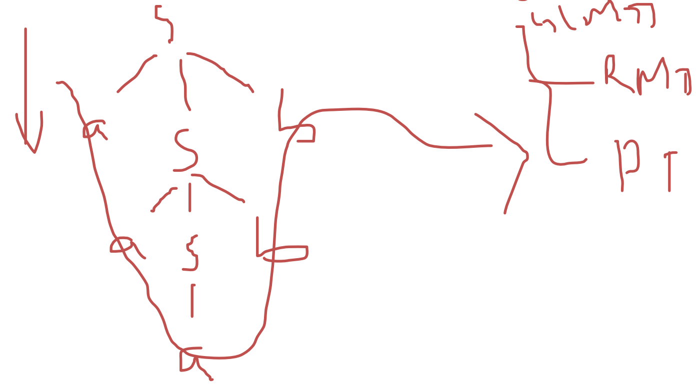
$L = \{ a, aab, aaabb, aaaabbb \ldots \}$

$L = \{ a a^n b^n : n \geq 0 \}$

$L = \{ a^n a b^n : n \geq 0 \}$

$S \Rightarrow a \in L$

aaabb



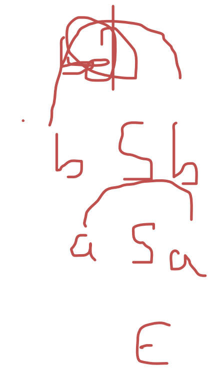
Obtain a grammar for $L = \{ ww^R : w \in \{a,b\}^*$ and where w^R is a reverse of w }

$S \rightarrow aSa \mid bSb \mid \epsilon$

w = baaaab

$\epsilon \cdot \epsilon = \epsilon$

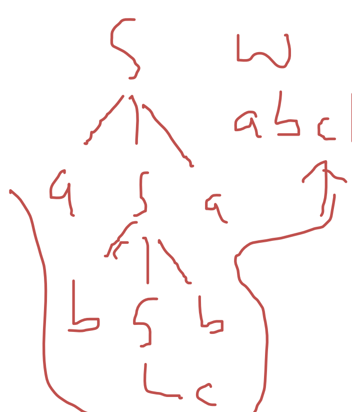
LMD = RMD



$L = \{ w . w^R : w \in \{a, b\}^* \}$

$S \rightarrow aSa \mid bSb \mid \epsilon$

abcba $\notin L \Rightarrow$



Design a CFG for the Language $L = \{ a^i b^j \mid i \neq j, i, j \geq 0 \}$

$i > j$ or $i < j$

$i \neq j$

$L_1 = \{ a^i b^j \mid i > j, \; i, j \geq 0 \}$

OR

$L_2 = \{ a^i b^j \mid i < j, \; i, j \geq 0 \}$

L_1

$S_1 \rightarrow a S_1 b \mid A$

$A \rightarrow aA \mid a$

L_2

$S_2 \rightarrow a S_2 b \mid B$

$B \rightarrow bB \mid b$

aaaabbb

$S_1 \Rightarrow a S_1 b$

$\Rightarrow aa S_1 bb$

$\Rightarrow aa A bb$

$\Rightarrow aaa A bb$

$\Rightarrow aaaabb \in L$

L:

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow a S_1 b \mid A$

$A \rightarrow aA \mid a$

$S_2 \rightarrow a S_2 b \mid B, \quad B \rightarrow bB \mid b$

Design a CFG for the Language $L = \{ a^n b^{n-3} \mid n \geq 3 \}$

$a^3 b^0 \quad n = 3$

$= aaa \in L$

4

$a^4 b^1 = aaaab$

$L = \{ a^{n+3} b^n \mid n \geq 0 \}$

$= \{ aaa\, a^n b^n \mid n \geq 0 \}$

$S \to aSb \mid aaa$

$aaa\, a^n b^n$

$S \to aaaA$

$A \to aAb \mid \epsilon$

For the regular expression $\underbrace{(011 + 1)^*}_{A} \underbrace{(01)^*}_{B}$ obtain the Context – Free Grammar

RE ⇒ CFG

|| ||

RL CFL

RL ⊆ CFL

(1)

$S \to AB$

$A \to 011A \mid 1A \mid \epsilon$

$B \to 01B \mid \epsilon$

(2) RE: $\underbrace{(a|b)^*}_{A} ab \underbrace{(a|b)^*}_{A}$

⇓

$S \to AabA$

$A \to aA \mid bA \mid \epsilon$

(3) $\overbrace{(a|b)^*}^{A} bb$

$S \to Abb$

$A \to aA \mid bA \mid \epsilon$

Obtain a CFG to generate set of all strings with exactly one a over { a,b}

$S \rightarrow bS \mid aB$
$B \rightarrow bB \mid \varepsilon$

b^*ab^*

$S \rightarrow AaA$
$A \rightarrow bA \mid \varepsilon$

b^*ab^*

$S \Rightarrow AaA \Rightarrow aA \Rightarrow a \in L$

Design a grammar for the Language $L = \{ w \mid n_a(w) = n_b(w) \}$

$S \rightarrow aSb \mid bSa \mid SS \mid \varepsilon$

Here S → SS take care of strings that starts and ends with same symbol (a or b)

baab

abba

$w = abbaab \in L$

$n_a(w) = 3$

$n_b(w) = 3$

$w = aababba \in L$

LMD, RMD, PT

$abab \in L$

$aabb \in L$

Design a grammar for the Language $L = \{ w \mid n_a(w) \geq n_b(w) \}$ - Home Work

$S \to aSb \mid bSa \mid SS \mid \epsilon$

$\Downarrow$

$A \to aA \mid a$

$A \to bA \mid b$

$\Rightarrow$
$$\begin{cases} S \to AS \mid SA \mid ASA \\ S \to aSb \mid bSa \mid SS \mid \epsilon \\ A \to aA \mid a \end{cases}$$

Design a CFG to generate the Language $L = \{ a^n b^m : n \geq 0, m > n \}$

$n=0, m \geq 1$

$a^n b^n b^m$ $\quad n \geq 0, m \geq 1$

$S \to aSb \mid B$

$B \to Bb \mid b$

$w = aaabbbbb$

$w = bbb \in L$

$S \Rightarrow B \Rightarrow Bb \Rightarrow Bbb \Rightarrow bbb$

$S \Rightarrow aSb$
$\Rightarrow aaSbb$
$\Rightarrow aaaSbbb$
$\Rightarrow aaaBbbb$
$\Rightarrow aaaBbbbb$
$\Rightarrow aaabbbbb \in L$

Design a CFG for the Language $L = \{ a^n b^m c^k \mid n + 2m = k \ \& \ n,m \geq 0 \}$

$a^n b^m c^n c^{2m}$

$T = \{a, b, c\}$

abc ✗

$aabbc$ ✗

$L = \{ a^n b^m c^{n+2m} \mid m,n \geq 0 \}$

$= \{ a^n b^m c^{2m} c^n \mid n,m \geq 0 \}$

A

S

$abbccccc \in L$

$a^1 b^2 c^5$

$k = 1 + 2 \times 2 = 5$

G:

$$\boxed{\begin{array}{l} S \to aSc \mid A \\ A \to bAcc \mid \epsilon \end{array}}$$

✓

$w = bbcccc \in L$

$S \Rightarrow A \Rightarrow bAcc$

$\Rightarrow bbAcccc$

$\Rightarrow bbcccc \in L$

a^2 c^2

$aabbcccccc \in L$

$aabbcccc \notin L$

Ambiguity

Ambiguous Grammars

-A CFG is ambiguous if it generate more than one parse tree for some (or all) strings. When this happens ,we say that the grammar is ambiguous.

-More precisely, a grammar G is ambiguous iff there is at least one string in L(G) for which G produce more than one parse tree (Obtained by applying either LMD or RMD).

-It is easy to write ambiguous grammars, if we are not careful. But such Grammars undesirable for many applications.

Why care?

-Ambiguity can be a problem in things like programming languages where we want agreement between the programmer and compiler over what happens

Examples-1

Is the following grammar ambiguous? Yes

$T = \{0, 1\}$

$S \rightarrow AS \mid \varepsilon$

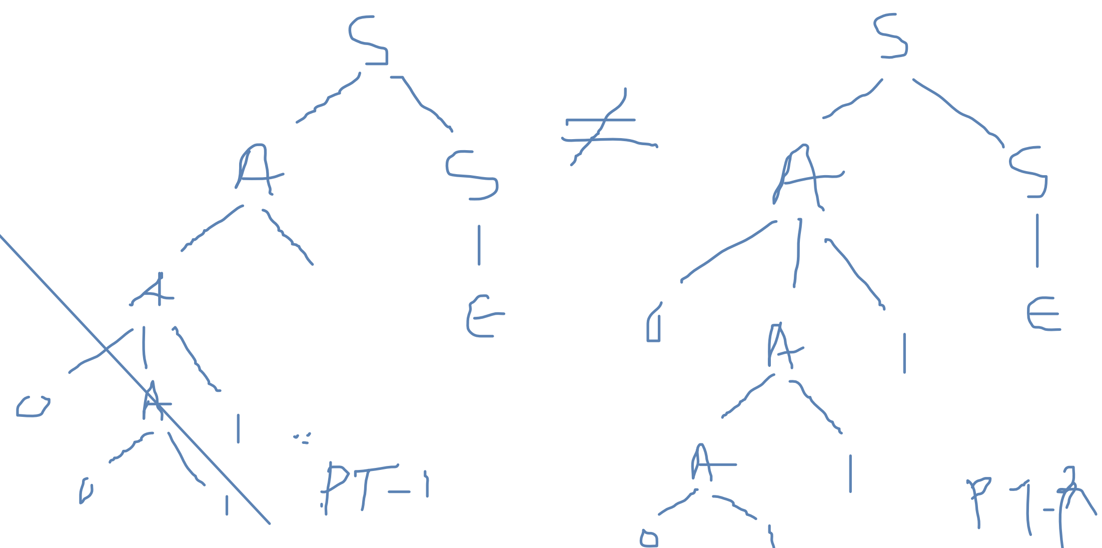
$A \rightarrow A1 \mid 0A1 \mid 01$

$V = \{S, A\}$

Take a string w = 00111

$S \Rightarrow AS \Rightarrow A1S \Rightarrow 0A11S \Rightarrow 00111S \Rightarrow 00111\varepsilon$ ----- (LMD)

$S \Rightarrow AS \Rightarrow 0A1S \Rightarrow 0A11S \Rightarrow 00111S \Rightarrow 00111\varepsilon$ ----- (LMD)



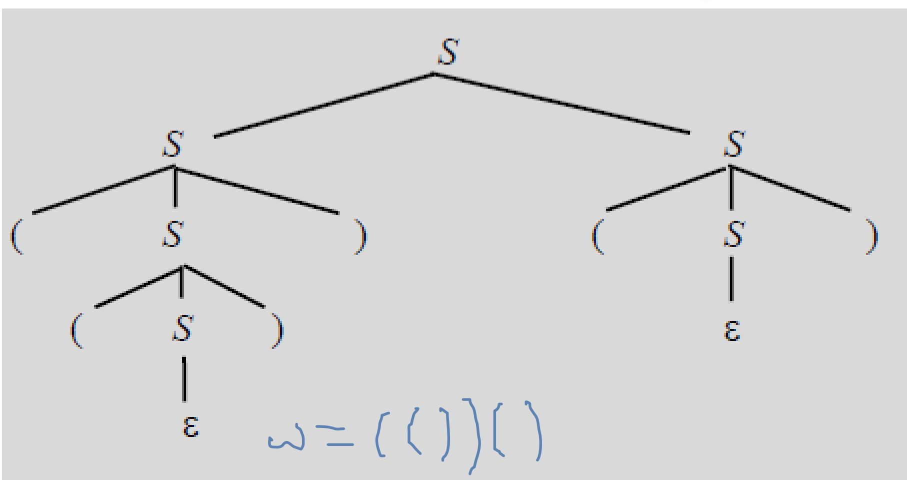
Example-2: The Balanced Parentheses Grammar is Ambiguous

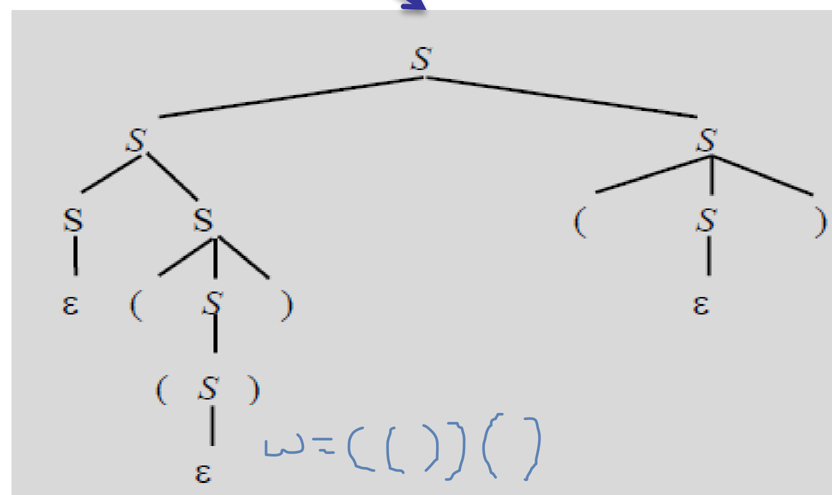
L = {*w* ∈ *{), (}* : the parentheses are balanced} is ambiguous.*

$$G: \quad S \rightarrow (S)$$
$$S \rightarrow SS$$
$$S \rightarrow \varepsilon$$

Take w = (())() ∈ L

Since there exist two parse trees, Hence G is ambiguous





In fact, G can produce an infinite number of parse trees for the string (())().

Example-3: Expression Grammar

T = {+, -, *, /, (,), id}

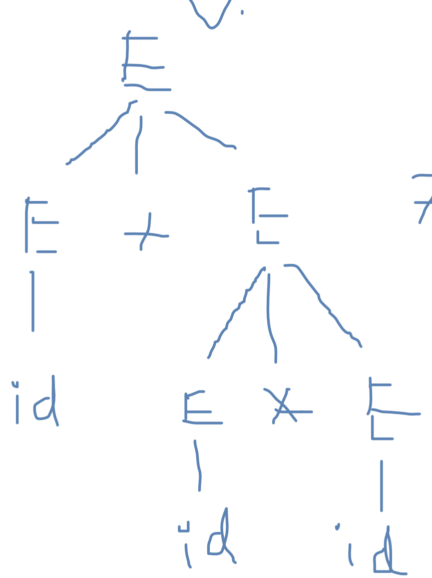
E→ E + E | E – E | E*E | E / E | (E) | id **is ambiguous ?**

I → a | b | c | d

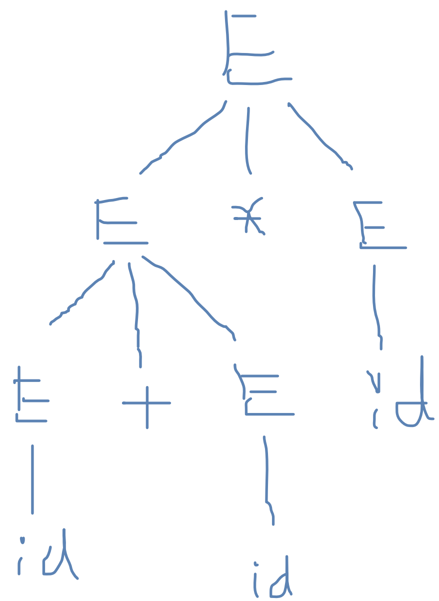
(id + id) * id

let w: id + id * id

id + id *



PT-1



PT-2

Hence Grammar is Ambiguous

Home work

Show that following Grammars are ambiguous:

1. S→ aB | bA
   A→ aS | bAA | a
   B→bS | aBB | b { take w = aabbab }
2. S → iCtS | iCtSeS | a
   C → b { w = ibtibtaea }
3. S → AB | aaB
   A → a | Aa
   B → b { w = aab }

4. S → aSbS | bSaS | ε { w = aababb }

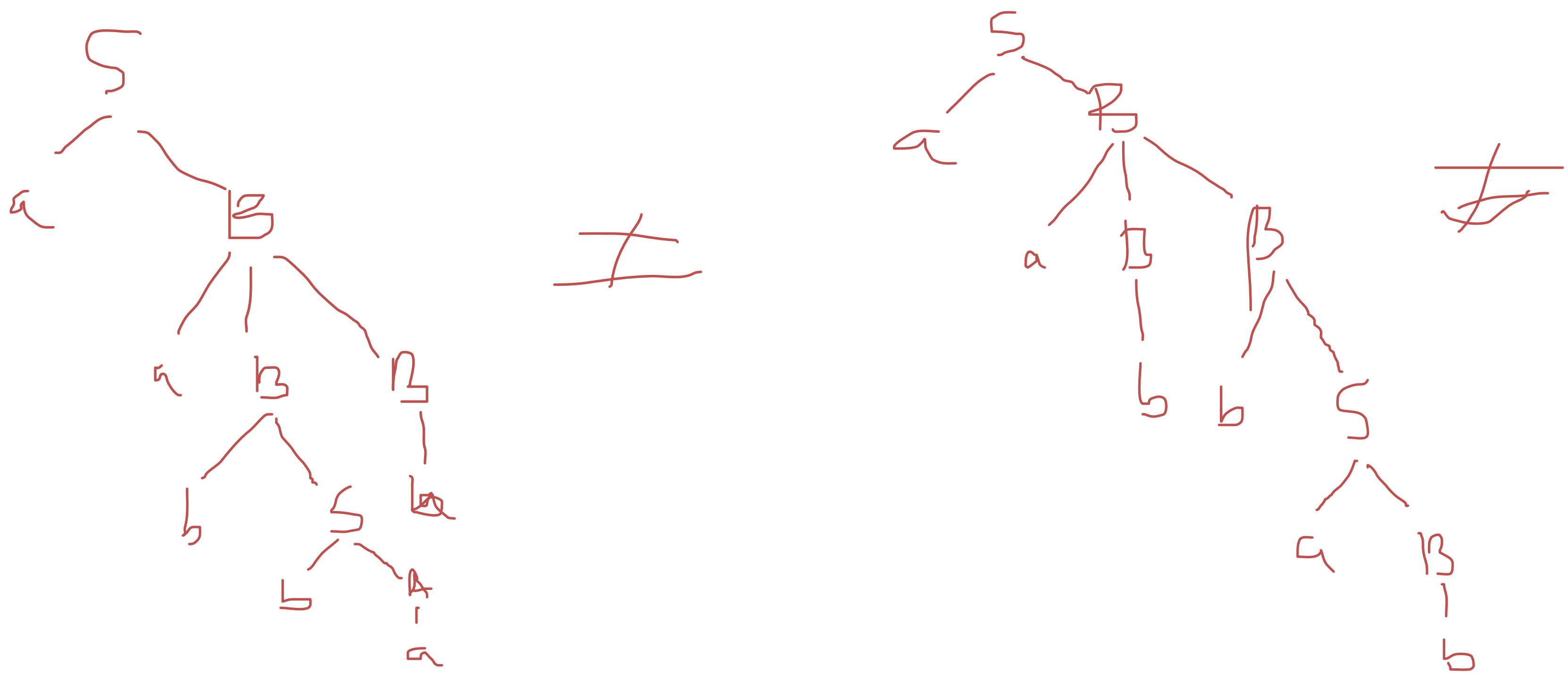
1. S→ aB | bA
   A→ aS | bAA | a
   B→bS | aBB | b   { take w = aabbab }

LMD-1

S ⇒ aB	{ S→ aB }
⇒ aaBB	{ B→ aBB }
⇒ aabSB	{ B → bS }
⇒ aabbAB	{ S → bA }
⇒ aabbaB	{ A→ a }
⇒ aabbab ∈ L	{ B → b }

LMD-2

S ⇒aB	{ S→ aB }
⇒ aaBB	{ B → aBB }
⇒ aabB	{ B → b }
⇒aabbS	{ B→bS }
⇒aabbaB	{ S→ aB }
⇒aabbab ∈ L	{ B→ b }

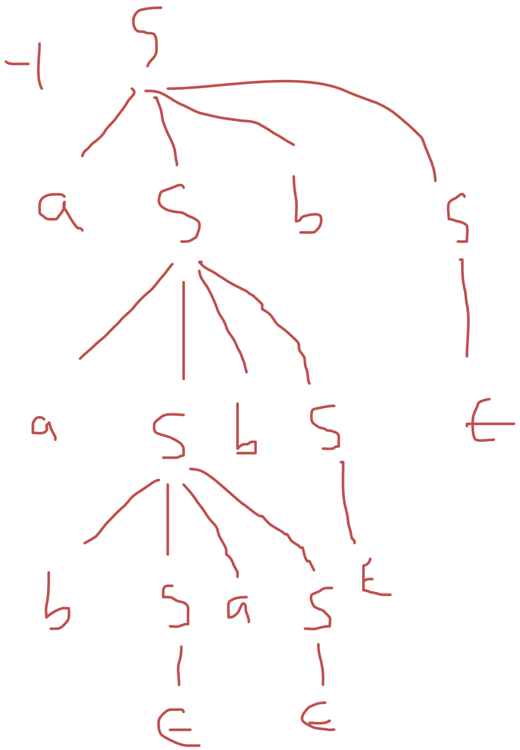


G:

4. S → aSbS | bSaS | ε { w = aababb }

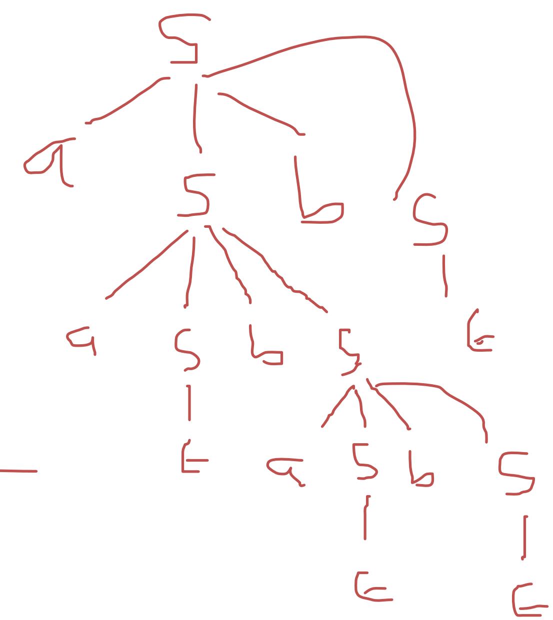
S ⇒ **aSbS** **{S→ aSbS }**
⇒ **aaSbSbS** **{ S→ aSbS }**
⇒ **aabSaSbSbS** **{ S → bSaS }** LMD-1
⇒ **aabaSbSbS** **{ S →** ε }
⇒ **aababSbS** **{ S→** ε }
⇒**aababbS** **{ S→** ε }
⇒ **aababb** ∈ L(G) **{ S→** ε }

PT-1



S ⇒ **aSbS** **{S→ aSbS }**
⇒ **aaSbSbS** **{ S→ aSbS }**
⇒ **aabSbS** **{ S →** ε **}** LMD-2
⇒ **aabaSbSbS** **{ S →** aSbS }
⇒ **aababSbS** **{ S→** ε }
⇒**aababbS** **{ S→** ε }
⇒ **aababb** ∈ L(G) **{ S→** ε }

PT-2



conclusion:

Inherent Ambiguous Language

-In many cases, when confronted with an ambiguous grammar G, it is possible to construct a new grammar G that generates L(G) and that has less (or no) ambiguity. Unfortunately, it is not always possible to do this. There exist context-free languages for which no unambiguous grammar exists. We call such languages **inherently ambiguous.**

$L = \{a^n b^n c^m \mid n, m \geq 0\} \cup \{a^n b^m c^m \mid n, m \geq 0\}$ is inherently ambiguous Language

G: $S \rightarrow S1 \mid S2$

$S1 \rightarrow S1c \mid A$

$A \rightarrow aAb \mid \boldsymbol{\varepsilon}$

$S2 \rightarrow aS2 \mid B$

$B \rightarrow bBc \mid \boldsymbol{\varepsilon}$

Techniques for Reducing Ambiguity

$G = G'$

-No Algorithms available to test for ambiguity in a grammars.

- No Algorithms or methods exist to remove Ambiguity from grammars

-But there do exist heuristics that we can use to find some of the more common source of ambiguity and remove them.

Techniques (heuristics)

$A \to BC$

1. Elimination of **ε**-productions from G
2. Elimination of **Unit** productions from G
3. Elimination of productions like S→SS or E → E + E in G (Symmetric and body contains atleast two copies of the NTs)
4. Elimination of useless symbols/Productions from G.

Eliminating ε-Productions

Definition: Let G = (V,T,P,S) be a CFG . A Production in P of the form A→ ε is called ε-Production or NULL production.

$T = \{a, b, c, d\}$ $G \neq G' = G - \{\varepsilon\}$

$V = \{S, A, B, C, D\}$

Ex. G:

$S \rightarrow ABCa \mid bD$
$A \rightarrow BC \mid b$
$B \rightarrow b \mid \varepsilon$
$C \rightarrow c \mid \varepsilon$
$D \rightarrow d$

In this grammar, the productions:

$B \rightarrow \varepsilon$
$C \rightarrow \varepsilon$

are ε-Productions.

NT/

To eliminate ε-Productions, we have to compute all Nullable variables in the grammar

S → ABCa | bD

A → BC | b

B → b | ε

C → c | ε

D → d

B → ε, C → ε

nullable

A → $B_1 B_2 B_3 B_4 \dots B_n$

a, b, c, d

Nullable variables

NV = {B, C, A} = {A, B, C}

ε ∈ L

S → ABCa | BCa | ACa | ABa | Ca | Aa | Ba | a | bD

A → BC | C | B | b

B → b

C → c

D → d

Eliminate all ε-Productions from the grammar:

S → BAAB
A → 0A2 | 2A0 | ε
B → AB | 1B | ε

NV = {A, B, S}

S → BAAB | AAB | BAB | BAA | AB | AA | BB | BA | B | A

A → 0A2 | 02 | 2A0 | 20

B → AB | B | A | 1B | 1

Eliminating Unit Productions

Definition: Let G = (V,T,P,S) be a CFG . A Production in P of the form A→ B is called unit Production. The Presence of Unit productions in G can be source of ambiguity.

Ex.: Consider the Grammar

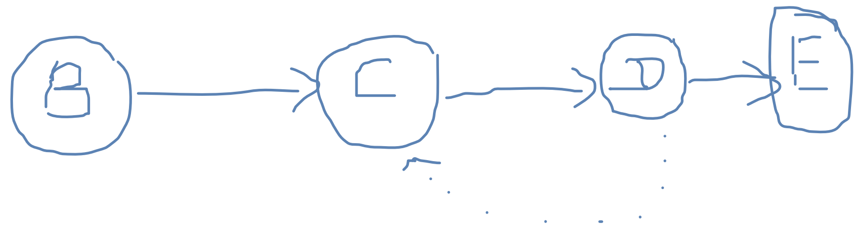
A→B | C
B→ aB | b | D

B → D

Here: A → B and A→ C are Unit Productions. B→aB and B → b are non–Unit productions.

Example:

using Dependency graph



Eliminate all Unit productions from the grammar:

S→ AB
A→ a
B→ C |b
C→D
D→E | bC
E→d | Ab

unit
B → C
C → D
D → E

before

non-unit
S → AB
A → a
B → b
D → bC
E → d | Ab

⟹

S → AB
A → a
B → b | bC | d | Ab
C → bC | d | Ab
D → bC | d | Ab
E → d | Ab

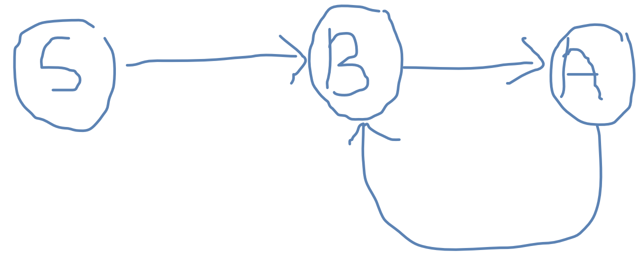
After Elimination

Eliminate unit productions from the grammar:

S→ A0 | B
B→ A | 11
A→ 0 | 12 | B

UP

S → B
B → A
A → B



Before

NUP

S → A0
B → 11
A → 0 | 12

U

S → A0 | 11 | 0 | 12
B → 11 | 0 | 12
A → 0 | 12 | 11

After

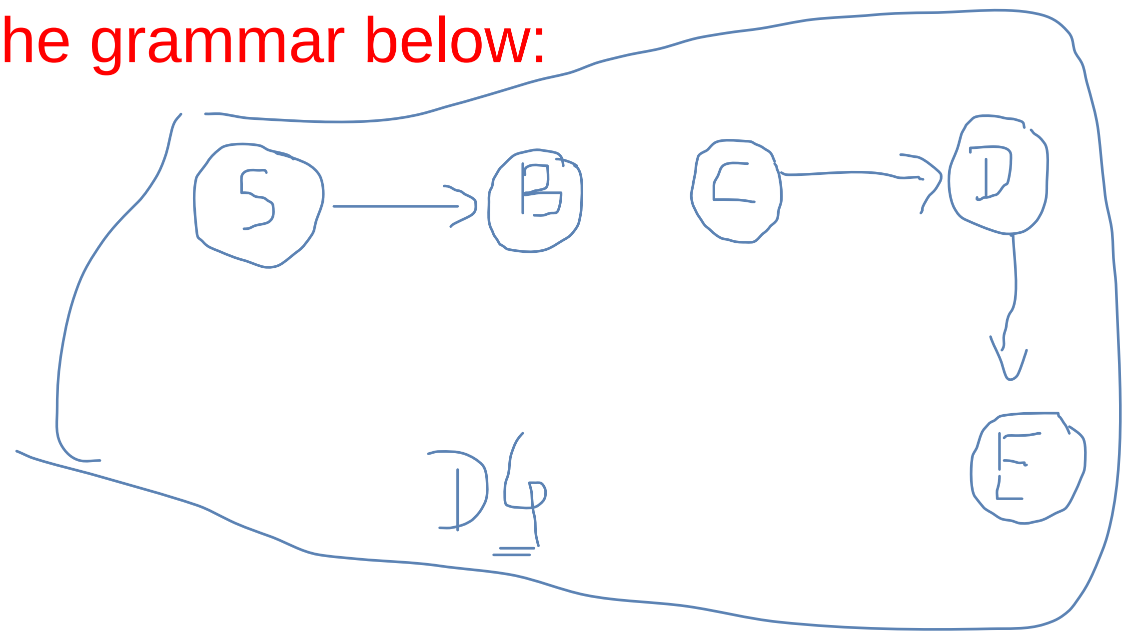
Practice Examples

1. Eliminate all ε-Productions from the grammar:

$S \rightarrow aSbS \mid bSaS \mid \varepsilon$

2. Eliminate Unit production from the grammar below:

$S \rightarrow Aa \mid B \mid Ca$
$B \rightarrow aB \mid b$
$C \rightarrow Db \mid D$
$D \rightarrow E \mid d$
$E \rightarrow ab$



Eliminating Symmetric Recursive Productions

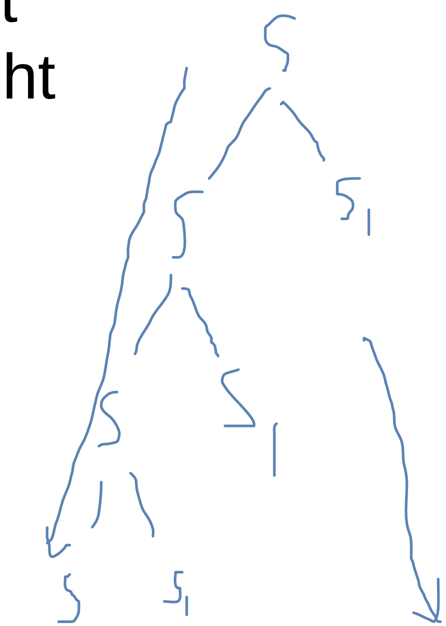
(S→SS , E → E + E etc. forms)

-Rewrite the grammar so that there is no longer choice
-Replace the production S → SS with one of the following production:

OR

$S \rightarrow SS_1$ /* force branching to the left
$S \rightarrow S_1S$ /* force branching to the right

then we add the production $S \rightarrow S_1$



Example: Consider the grammar

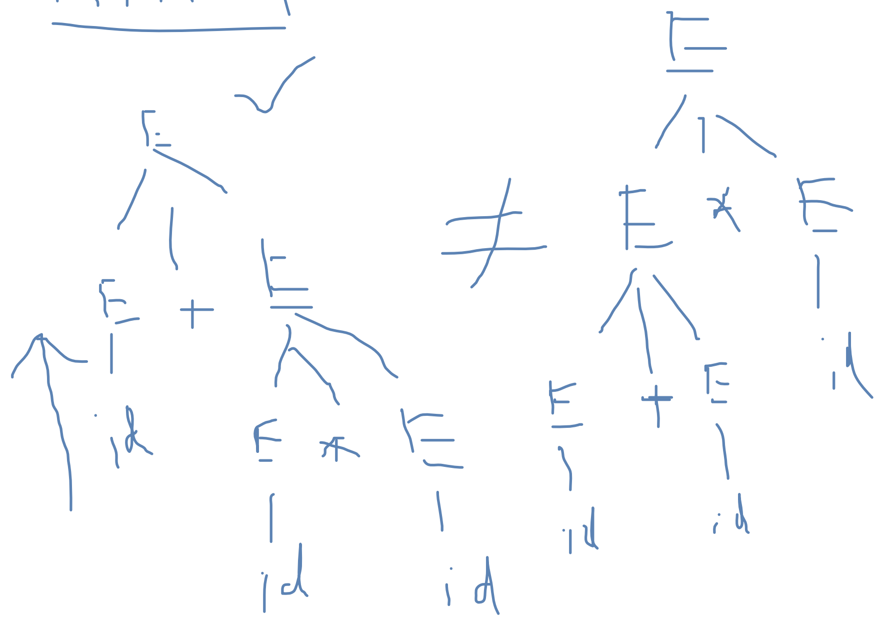
$$\left\{ \begin{array}{l} S \rightarrow SS \\ S \rightarrow (S) \\ S \rightarrow \boldsymbol{\varepsilon} \end{array} \right. \quad \Longrightarrow \quad \left\{ \begin{array}{l} S \rightarrow SS_1 \\ S \rightarrow S_1 \\ S_1 \rightarrow (S) \\ S_1 \rightarrow \boldsymbol{\varepsilon} \end{array} \right.$$

Example:

Ambiguous Grammar

E→ E + E | E-E | E*E | E/E | (E) | id

$\omega = id + id * id$



Unambiguous Grammar

E → E + T | E-T | T

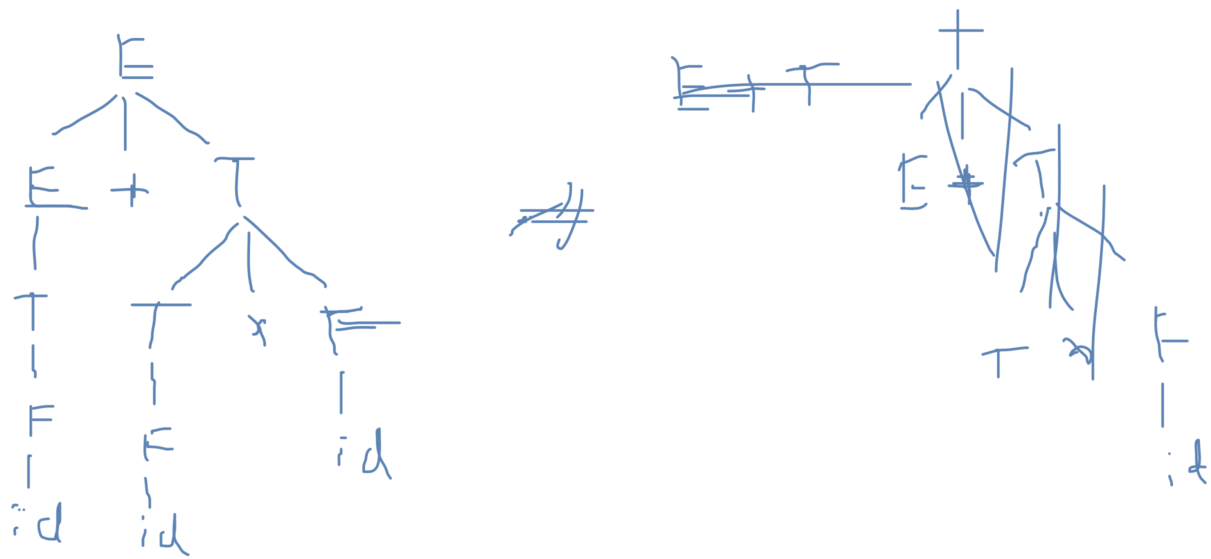
T → T * F | ~~E/E~~ | F | T/F

F → (E) | id

+, - } E

*, / } T

( ), id } F



Eliminating useless Symbol

Definition: A symbol X is useful , if there is a derivation of the form:

$$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w \in L$$

Otherwise , the symbol X is useless.

Useless Production

Example: Consider the grammar

S → aA | bB
A → aA | a
B → bB
D → ab | Ea
E → aC | d

Procedure:

Step1: Compute Non-Generating symbols in G and eliminate them.

Step2: Computing Un-Reachable symbols in G and eliminate them.

Example: 1. Eliminate useless symbols in the grammar

$A^{5} \rightarrow X_1 X_2 X_3 \cdots X_n$

$A \rightarrow a$

S → aA | bB
A → aA | a
B → bB
D → ab | Ea
E → aC | d

$G_S = \{a, b, d, A, D, S, E\}$

$NG = \{B\}$ non-Generating

$U_S = \{D, E\}$ unReachable symbols

Step 1

$S \rightarrow aA$

$A \rightarrow aA \mid a$

$D \rightarrow ab \mid Ea$

$E \rightarrow aC \mid d$

$R_S = \{a, b, d, S, A\}$

$$\boxed{\begin{aligned} S &\rightarrow aA \\ A &\rightarrow aA \mid a \end{aligned}}$$

2. Eliminate Useless symbols in the grammar below

G :

S → aA |a | Bb |cC
A → aB
B → a | Aa
C → cCD
D → ddd

HW

step-1 G'

step-2 G''

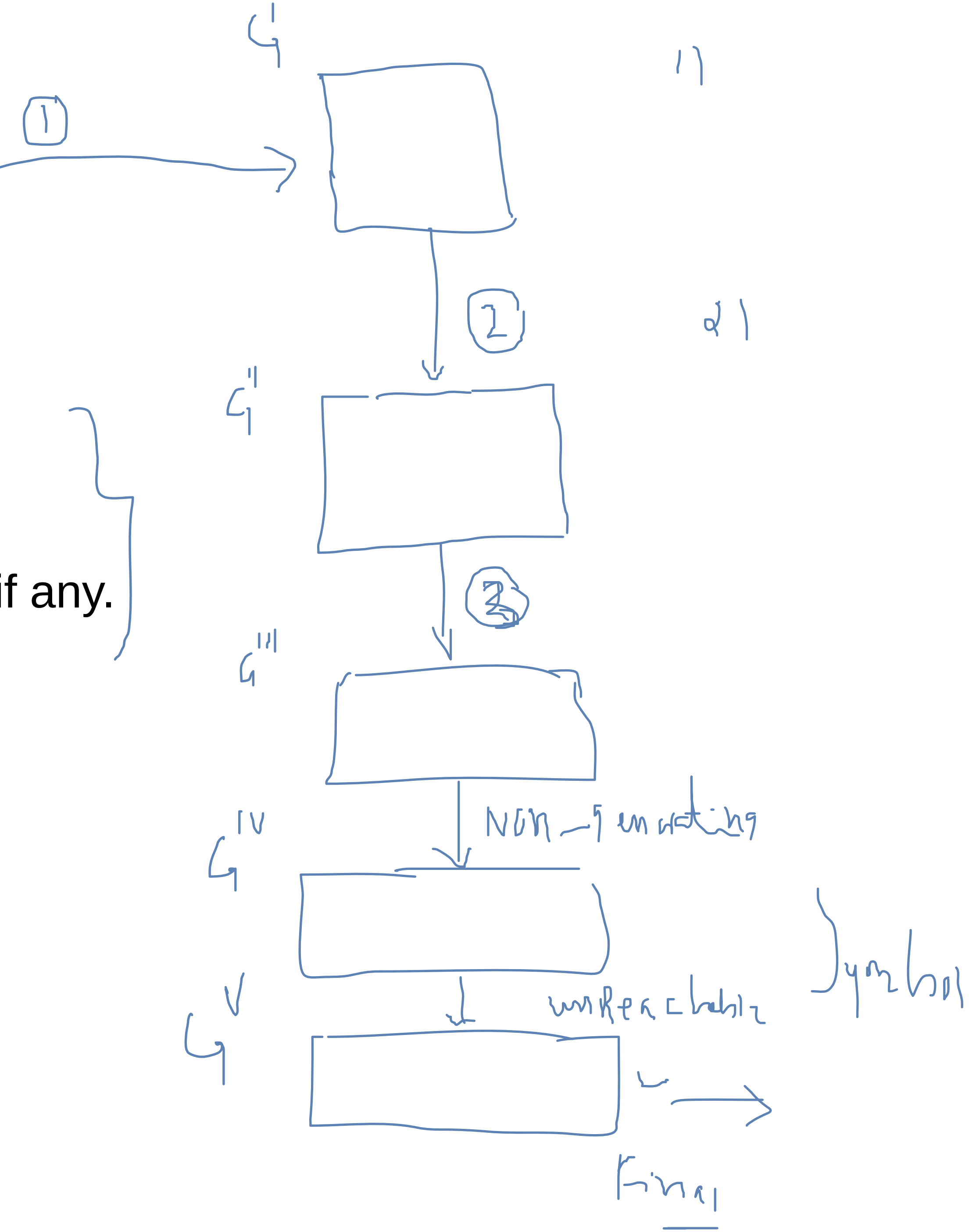
$G = G' = G''$

HW

Consider the grammar G:

A → bA | Bba | aa
B → aBa | b | D
C → CA | AC | B
D → a | **ε**

1) Eliminate any ε-Productions.
2) Eliminate any unit productions
3) Eliminate useless productions, if any.



Consider the grammar G:

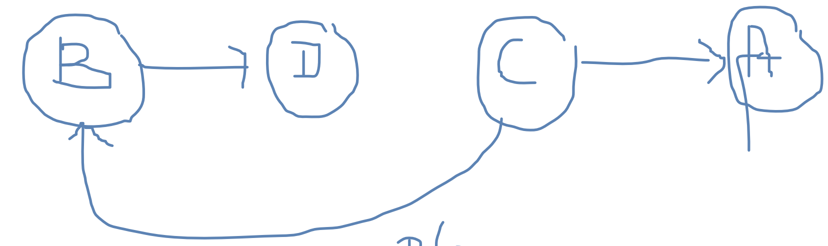
$A \rightarrow bA \mid Bba \mid aa$
$B \rightarrow aBa \mid b \mid D$
$C \rightarrow CA \mid AC \mid B$
$D \rightarrow a \mid \boldsymbol{\varepsilon}$

1) Eliminate any ε-Productions.
2) Eliminate any unit productions
3) Eliminate useless productions, if any.

1) $NV = \{D, B, C\}$

$A \rightarrow bA \mid Bba \mid ba \mid aa$
$B \rightarrow aBa \mid aa \mid b \mid D$
$C \rightarrow CA \mid A \mid AC \mid B$
$D \rightarrow a$

1)



DG

$A \rightarrow bA \mid Bba \mid ba \mid aa$
$B \rightarrow aBa \mid aa \mid b \mid a$
$C \rightarrow CA \mid AC \mid aBa \mid aa \mid b \mid a \mid bA \mid Bba \mid ba$
$D \rightarrow a$

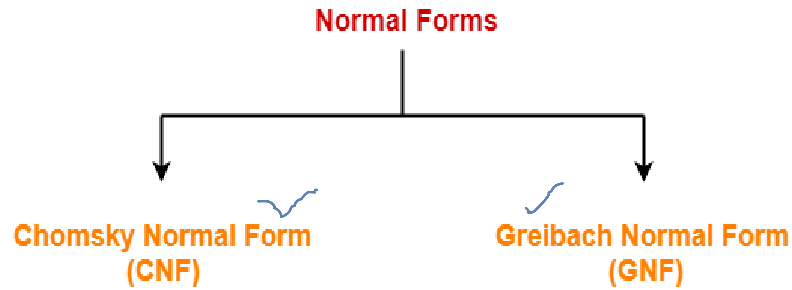
3) $G_S = \{a, b, D, C, B, A\}$ $N_G = \{\Phi\}$
$U_S = \{C, D\}$

$A \rightarrow bA \mid Bba \mid ba \mid aa$
$B \rightarrow aBa \mid aa \mid b \mid a$

Normal form

A → [a] →

The restriction can be imposed on the right hand side of productions in a CFG resulting in various normal forms.



1. Chomsky Normal Form (CNF)
2. Greibach Normal Form (GNF)

Definition(CNF): Let G = (V,T,P,S) be a CFG. The grammar G is said to be in CNF, if all productions are of the form:

$$A \rightarrow BC$$

or

$$A \rightarrow a$$

where A,B and C ∈ V and a ∈ T.

Ex. $S \rightarrow AB$
$A \rightarrow a$
$B \rightarrow b$

CNF

This context free grammar is in Chomsky normal form.

Definition(GNF): Let G = (V,T,P,S) be a CFG. The grammar G is said to be in GNF, if all productions are of the form:

$$A \rightarrow a\alpha$$

where α ∈ V^* and a ∈ T.

Ex: S → bABC
A → a
B → bB

~~S → aSb~~
~~S → ε~~

Problems:

Convert the Following grammar to Chomsky Normal Form:

S→ 0A | 1B
A → 0AA | 1S | 1
B → 1BB | 0S | 1

Sm -- 10M

$A \to BC$
$A \to a$

$S \to ABBC$
$S \to A_1 A_2$
$A_1 \to AB$
$A_2 \to BC$

$A \to 1$, $B \to 1 \to$ ①

CNF

$S \to 0A$	$S \to A_0 A$, $A_0 \to 0$
$S \to 1B$	$S \to A_1 B$, $A_1 \to 1$
$A \to 0AA$	$A \to A_0 A A$ ⇓ $A \to A_2 A$, $A_2 \to A_0 A$
$A \to 1S$	$A \to A_1 S$
$B \to 1BB$	$B \to A_1 BB$ ⇓ $B \to A_3 B$, $A_3 \to A_1 B$
$B \to 0S$	$B \to A_0 S$

$S \to \underbrace{ABB}_{A_1}\underbrace{CD}_{A_2}$
$S \to A_1 A_2$
$A_1 \to ABB$
$A_2 \to CD$
$A_1 \to A_3 B$
$A_3 \to AB$

CNF

$S \to A_0 A \mid A_1 B$
$A \to A_2 A \mid A_1 S$
$B \to A_3 B \mid A_0 S$
$A \to 1$
$B \to 1$
$A_0 \to 0$
$A_1 \to 1$
$A_2 \to A_0 A$
$A_3 \to A_1 B$

Convert the Following grammar to Chomsky Normal Form:

$S \rightarrow aAD$

$A \rightarrow aB \mid bAB$

$B \rightarrow b$

$D \rightarrow d$

$S \rightarrow A_1 A D \Rightarrow S \rightarrow A_2 D, \; A_2 \rightarrow A_1 A$

$A_1 \rightarrow a$

Convert the Following grammar to CNF:

$S \rightarrow aAD$
$A \rightarrow aB \mid bAB$
$B \rightarrow b \mid D$
$D \rightarrow d$

$B \rightarrow D$

~~$B \rightarrow DA$~~
~~$A \rightarrow CX$~~

Eliminating Unit Pro

$$\boxed{\begin{array}{l} S \rightarrow aAD \\ A \rightarrow aB \mid bAB \\ B \rightarrow b \mid d \\ D \rightarrow d \end{array}} \longrightarrow \text{CNF}$$

✓

Convert the Following grammar to CNF:

$S \rightarrow aACa$
$A \rightarrow B \mid a$
$B \rightarrow C \mid c$
$C \rightarrow cC \mid \varepsilon$

← productions
↓
unit productions

$N_V = \{C, B, A\}$ ✓

①

$S \rightarrow aACa \mid aCa \mid aAa \mid aa$
$A \rightarrow B \mid a$
$B \rightarrow C \mid c$
$C \rightarrow cC \mid c$

② $A \rightarrow B \rightarrow C$

16M - 2om

✓

$S \rightarrow \ldots$
$A \rightarrow c \mid a \mid cC$
$B \rightarrow cC \mid c$
$C \rightarrow cC \mid c$

$\Downarrow$

CNF =

Practice Problems

Convert the Following grammars to CNF:

1)
$$\begin{aligned} S &\rightarrow 0A0 \mid 1B1 \mid BB \\ A &\rightarrow C \\ B &\rightarrow S \mid A \\ C &\rightarrow S \mid \epsilon \end{aligned}$$

2)
$$\begin{aligned} S &\rightarrow AAA \mid B \\ A &\rightarrow aA \mid B \\ B &\rightarrow \epsilon \end{aligned}$$

3)
$$\begin{aligned} S &\rightarrow AB \mid CA \\ A &\rightarrow a \\ B &\rightarrow BC \mid AB \\ C &\rightarrow \cancel{aB \mid b} \end{aligned}$$

$A_1B \mid b$

$A_1 \rightarrow a$

4)

Aug-2010,

Obtain the following grammar in CNF

10M 12m 2

$S \rightarrow ABC$
$A \rightarrow aC / D$
$B \rightarrow bB / E / A$
$C \rightarrow Ac / E / Cc$
$D \rightarrow aa$

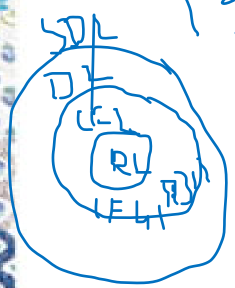
CFh

THANQ...

THANQ...

FSM → ND FSM D FSM 5-tuple, 7-tuple

2-D CFL → CFG PDA → 3-D



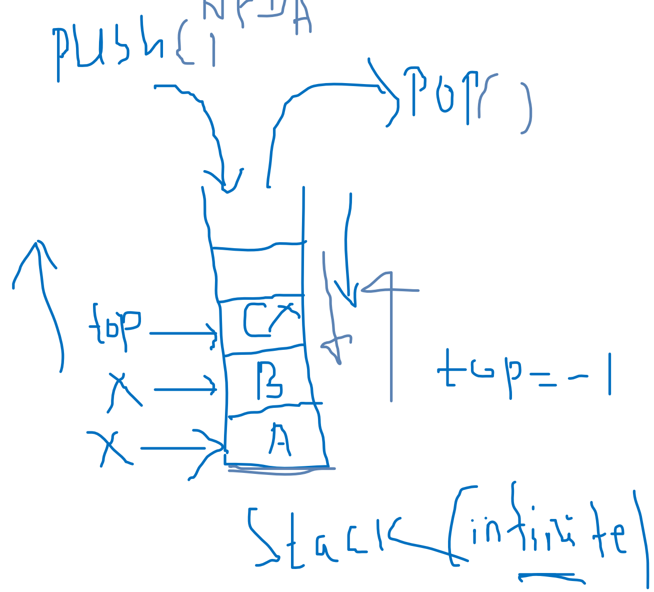
Module-3

Automaton = FSM + Memory (stack)

Pushdown Automata(PDA)

push()
pop()

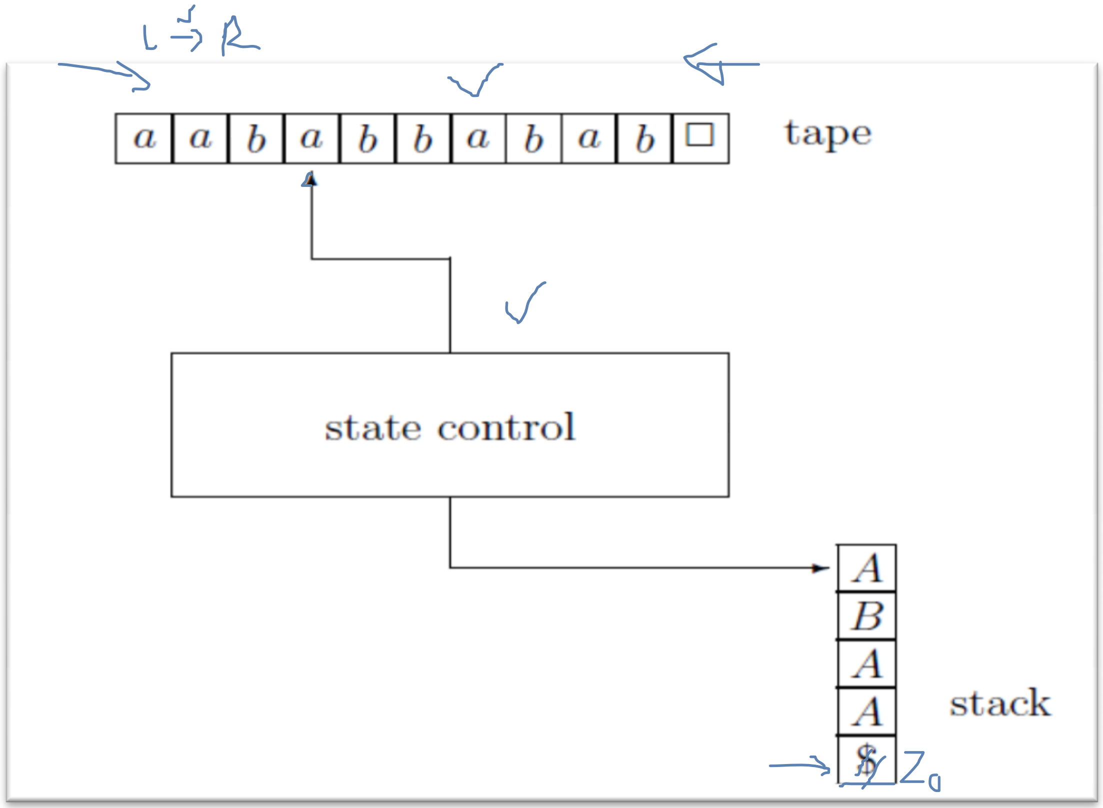
DPDA NPDA



Content

-An introduction to PDA, Languages of the PDA
-Designing PDA
-Deterministic and Non-deterministic PDAs
-Alternative equivalent definitions of a PDA

Introduction: An informal description of a pushdown automaton is shown in the diagram below. Such an automaton consists of the following:



100 years old

RAM

FSM →

-There is a tape which is divided into cells.

-There is a tape head which can move along the tape, one cell to the right per move.

-There is a stack containing symbols and special symbol $ or Z_0

-There is a state control, which can be in any one of a finite number of states.

In one transition, the pushdown automaton:

1. Consumes from the input the symbol that it uses in the transition. If ϵ is used for the input, then no input symbol is consumed.

2. Goes to a new state, which may or may not be the same as the previous state.

push

3. Replaces the symbol at the top of the stack by any string. The string could be ϵ, which corresponds to a pop of the stack. It could be the same symbol that appeared at the top of the stack previously; i.e., no change to the stack is made. It could also replace the top stack symbol by one other symbol, which in effect changes the top of the stack but does not push or pop it. Finally, the top stack symbol could be replaced by two or more symbols, which has the effect of (possibly) changing the top stack symbol, and then pushing one or more new symbols onto the stack.

push → one or more symbols
pop → only one symbol at time
replace →
unchanged

Definition:

NPDA, DPDA

S Δ

A PDA can be formally defined as a 7-tuple: $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$

The components have the following meanings:

Q: A finite set of *states*, like the states of a ~~finite automaton~~. FSM

Σ: A finite set of *input symbols*, also analogous to the corresponding component of a ~~finite automaton~~. FSM

Γ: A finite set of stack symbols

δ : The transition function: Maps $Q \times \{\Sigma \cup \varepsilon\} \times \Gamma \rightarrow Q \times \Gamma^*$

q_0 : The start state

Z_0 : The stack initial(bottom marker) symbol.

F: The set of Accepting states or final states

aa

aab Z_0

top

$\delta(q, a, b) = (q, ab)$

$= (q, \varepsilon)$

$= (q, b)$

$= (q, B)$

Final

ε

b

Z_0

Example: A pushdown automata(PDA) for accepting L = { a^nb^n : $n \geq 1$ }

$a, Z_0 | aZ_0$

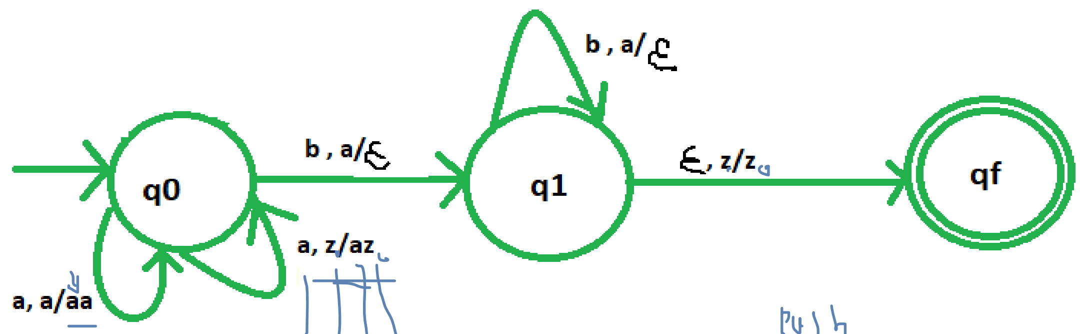
$a, a | aa$

$\exists$ PDA

ϵ, Z_0

$= \{aabb, ab, aaabbb, \ldots$

a^3b^3



Required PDA

push

PDA

No-change

$w = aaabbb \in L$

$aabbb \notin L$

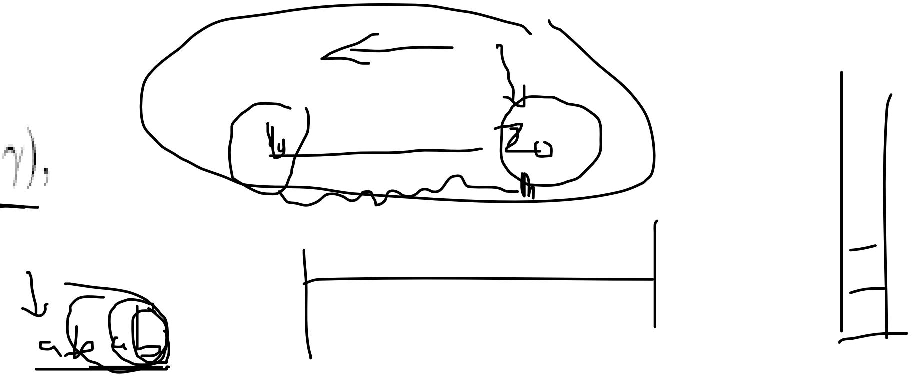
$\delta(q_0, a, Z_0) = (q_0, aZ_0)$

$a, Z_0 / aZ_0$

Z_0

$w = aaabbb\,\epsilon \in L$

Instantaneous Descriptions of a PDA (ID)



we shall represent the configuration of a PDA by a triple (q, w, γ),

where

1. q is the state,
2. w is the remaining input, and
3. γ is the stack contents.

Conventionally, we show the top of the stack at the left end of γ and the bottom at the right end. Such a triple is called an *instantaneous description*, or ID, of the pushdown automaton.

$\vdash$ $\quad \vdash_P$

Moves of a PDA

Let $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. Define $\underset{P}{\vdash}$, or just $\vdash$ when P is understood, as follows. Suppose $\delta(q, a, X)$ contains (p, α). Then for all strings w in Σ^* and β in Γ^*:

$$\underbrace{(q, aw, X\beta) \vdash (p, w, \alpha\beta)}_{\text{ID} \qquad \text{ID}} \vdash \cdots$$

$\vdash^*$

We also use the symbol $\vdash^*$ to represent zero or more moves of the PDA

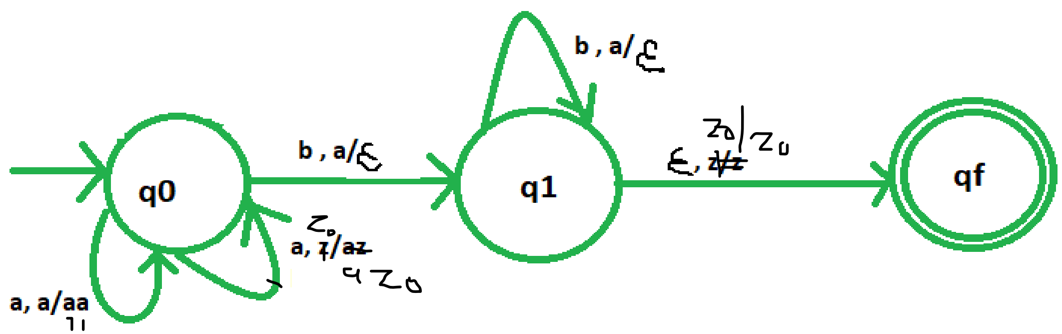
ID $\vdash^*$ ID

Example: Show the moves made by PDA for the string "aaabbb"

Or

Give the sequence of IDs the PDA is in for the string "aaabbb"

$a^3b^3 \in L$



DPDA

Required PDA

ID ID ID ID

$(q_0, aaabbb, Z_0) \vdash (q_0, aabbb, aZ_0) \vdash (q_0, abbb, aaZ_0) \vdash (q_0, bbb, aaaZ_0)$

$\vdash (q_1, bb, aaZ_0) \vdash (q_1, b, aZ_0) \vdash (q_1, \epsilon, Z_0) \vdash (q_f, \epsilon, Z_0)$

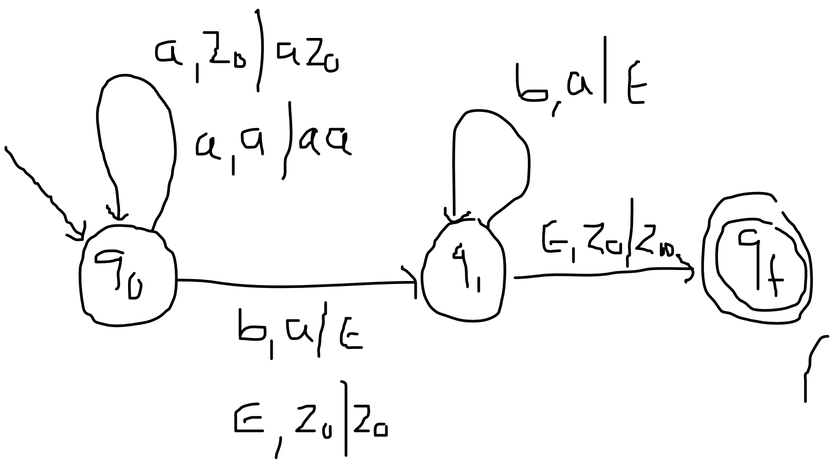
Accept

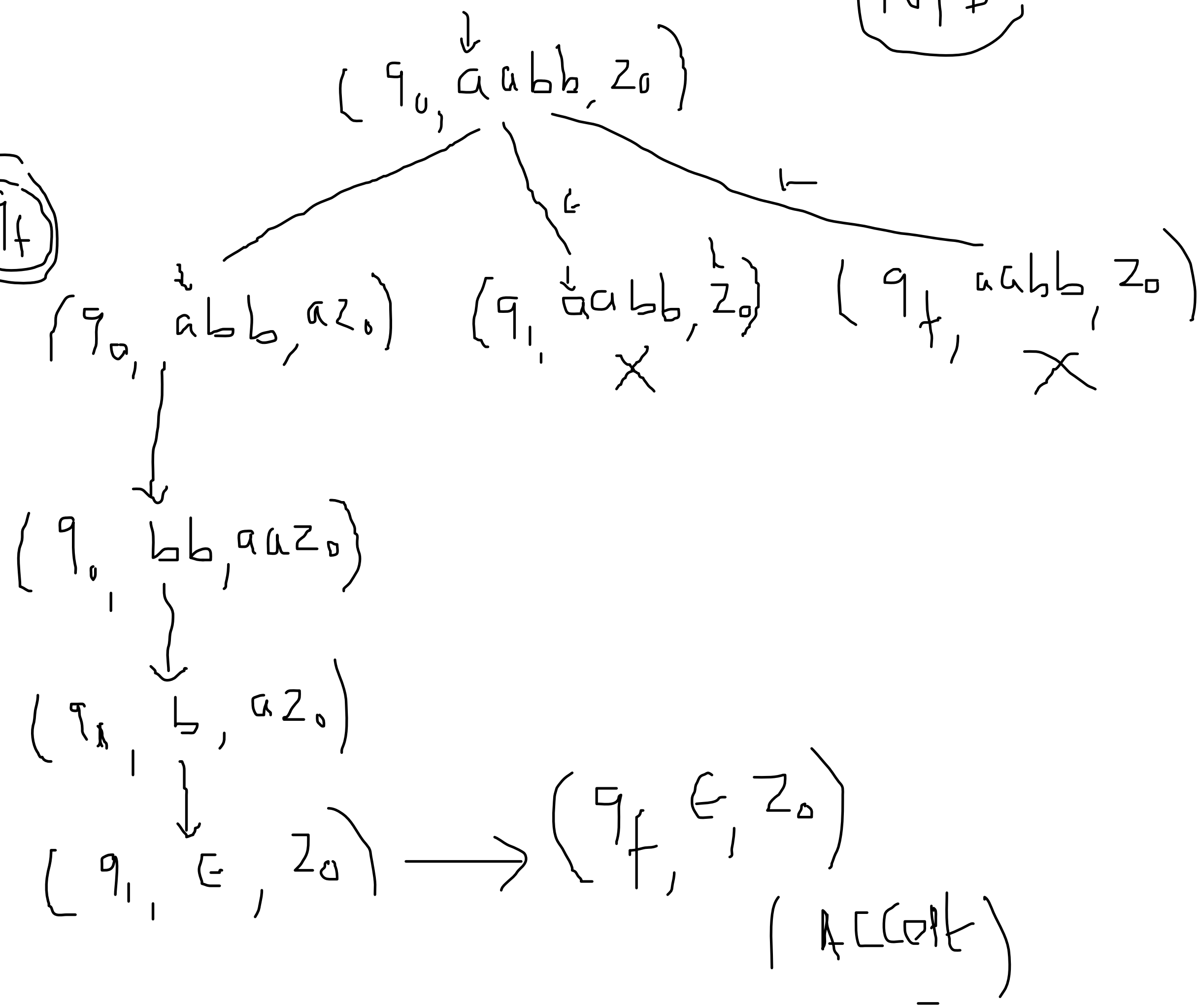
$$\boxed{(q_0, aaabbb, Z_0) \vdash^{*} (q_f, \epsilon, Z_0)}$$

Example: Design a Pushdown automata(PDA) for accepting $L = \{ a^n b^n : n \geq 0 \}$

& also show the sequence of IDs for the string "aabb" .

NPDA





"aaabbb"

Language of PDA *(Alternate definition of PDA)*

A language can be accepted by PDA using two approaches:

1. **Acceptance by Final State:** The PDA is said to accept its input by the final state if it enters any final state in zero or more moves after reading the entire input.

Let P =(Q, ∑, Γ, δ, q_0, Z_0, F) be a PDA. The language acceptable by the ***final state*** can be defined as:

ID *ID*

L(P) = $\{w \mid (q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (q, \epsilon, \alpha)\}$ for some state q in F and any stack string α.

2. **Acceptance by Empty Stack**

On reading the input string from the initial configuration, the stack of PDA gets empty.

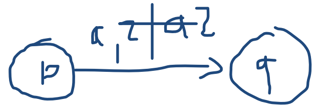
Let P =(Q, ∑, Γ, δ, q_0, Z_0, F) be a PDA. The language acceptable by ***empty stack*** can be defined as:

L(P) $= \{w \mid (q_0, w, Z_0) \overset{*}{\vdash} (q, \epsilon, \epsilon)\}$ for any state q.

Since set of accepting states are irrelevant, We shall sometimes leave off , seventh component from P.

What does each of the following transitions represent?

1. $\delta(p, a, Z) = (q, aZ)$

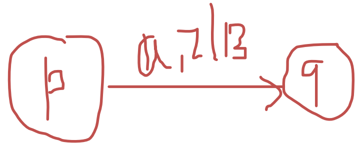


push( )

2. $\delta(p, a, Z) = (q, \varepsilon)$

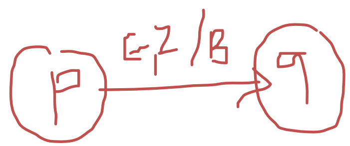
pop( )

3. $\delta(p, a, Z) = (q, B)$



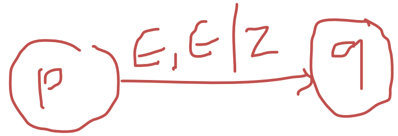
Replace( )

4. $\delta(p, \varepsilon, Z) = (q, B)$



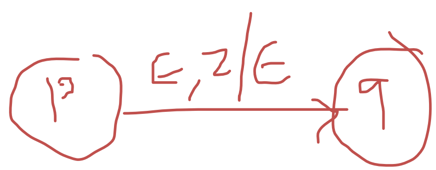
Replace( )

5. $\delta(p, \varepsilon, \varepsilon) = (q, Z)$



Replace( )

6. $\delta(p, \varepsilon, Z) = (q, \varepsilon)$

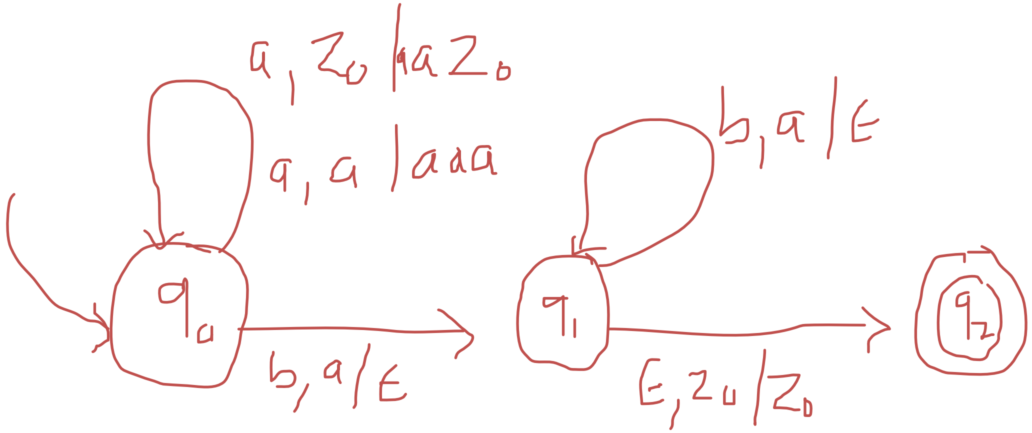


pop( )

Designing PDA

$\epsilon, abb \leftarrow n \geq 0$

Design a Pushdown automata(PDA) for L = { $a^n b^{2n}$: n ≥ 1 }



Moves by PDA: $w = aabbbb$ 1st

$(q_0, aabbb, Z_0) \vdash (q_0, abbbb, aaZ_0) \vdash (q_0, bbbb, aaaaZ_0)$

$\vdash (q_1, bbb, aaaZ_0)$

$\vdash (q_1, bb, aaZ_0)$

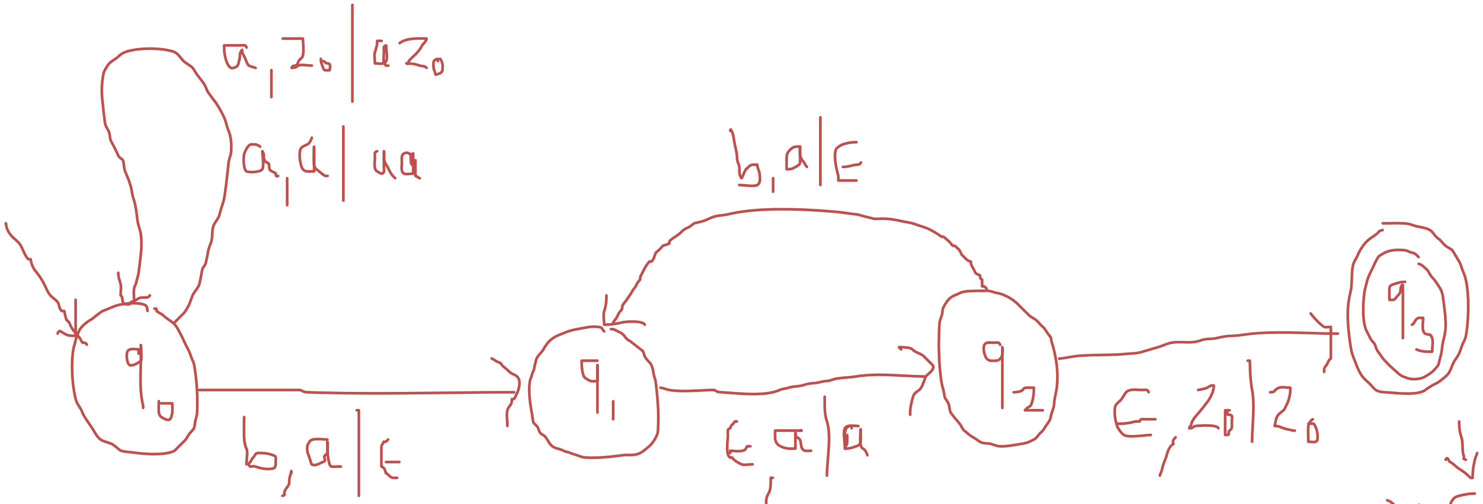
$\vdash (q_1, b, aZ_0)$

$\vdash (q_1, \epsilon, Z_0) \vdash (q_2, \epsilon, Z_0)$ accept

Initial ID

(Final ID)

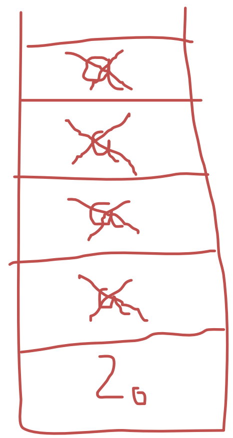
Design a Pushdown automata(PDA) for $L = \{ a^{2n}b^n : n \geq 1 \}$



w : aaaabb

ID

$\Gamma = \{a, Z_0\}$



ID

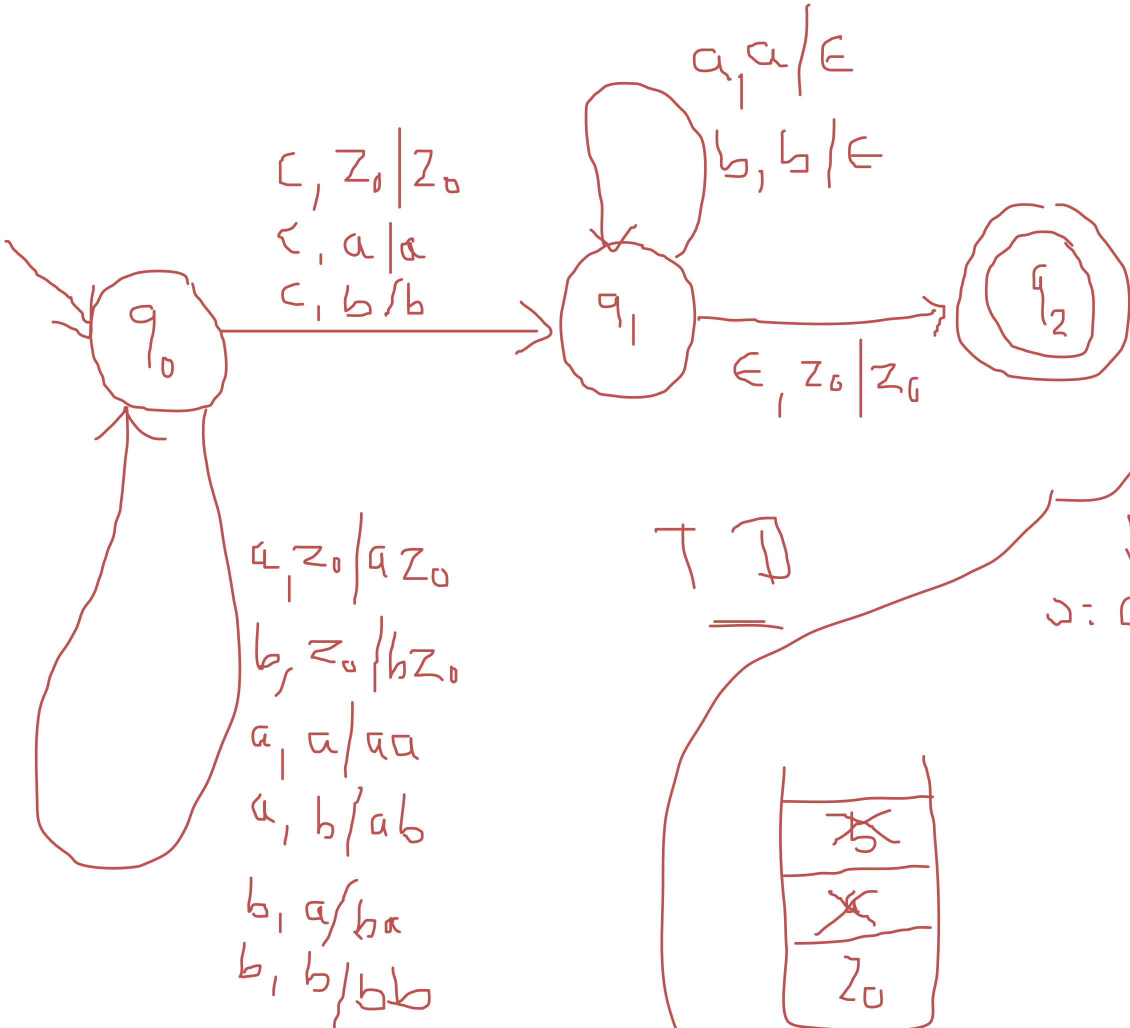
$(q_0, aaaabb, Z_0) \vdash (q_0, aaabb, aZ_0) \vdash \ldots\ldots \vdash (q_3, \epsilon, Z_0)$

Construct a PDA to accept the language $L = \{ wcw^R \mid w \in \{a,b\}^*$ **and** w^R is reverse of $w \}$ by a final state

DPDA

$c \in L$ $\quad$ $c \in L$

$S \to aSa \mid bSb \mid \epsilon$



TD

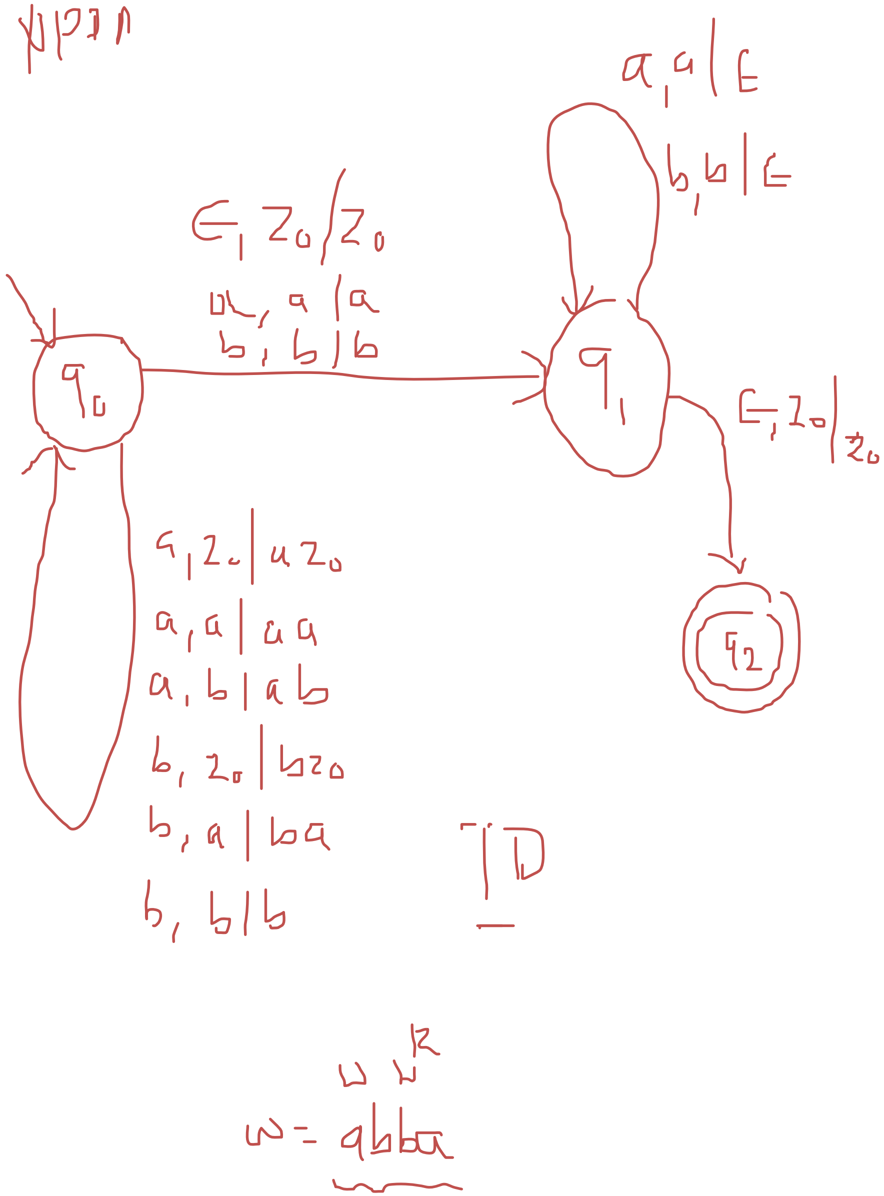
$w \quad w^R$

$abbcbba \in L$

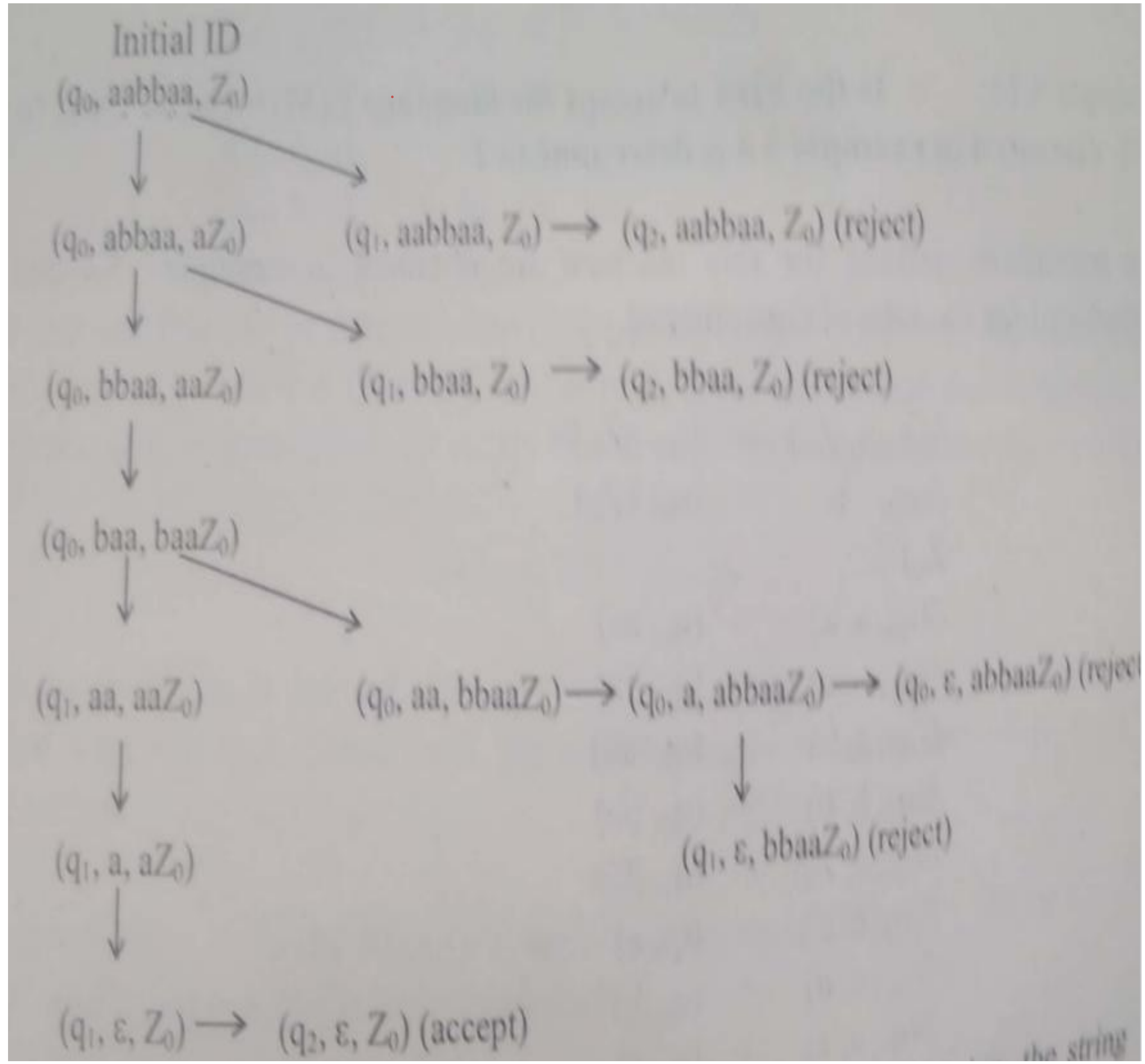
$w: abcba \in L$

$bacbb \notin L$

Design a PDA to accept the language L = { ww^R | w ∈ **{a,b} * and** w^R is reverse of w }



Moves by PDA: on “ aabbaa”



Practice Problems

$w = abba$

$|\epsilon| = 0$

1. Obtain a PDA to accept the language $L = \{ w \in \{a,b\}^* \mid N_a(w) = N_b(w) \}$

ex: abbaa

2. Obtain a PDA to accept the language $L = \{ w \in \{a,b\}^* : N_a(w) > N_b(w) \}$ by final state.

3. Obtain a PDA to accept the language $L = \{ w \in \{a,b\}^* : N_a(w) < N_b(w) \}$ by final state.

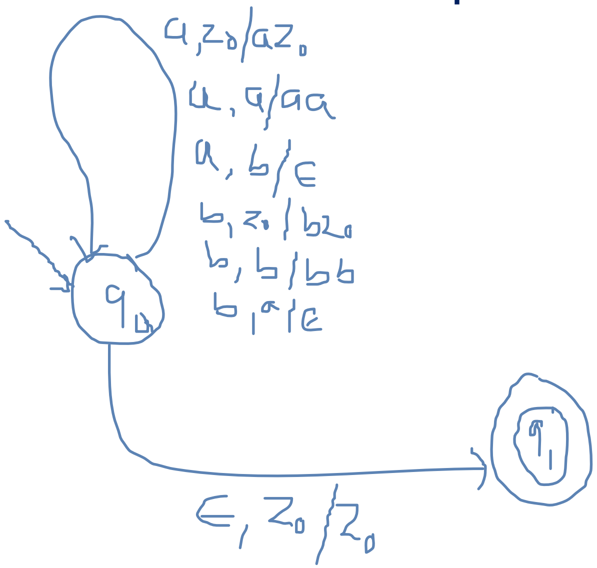
4. Design a PDA to accept all the strings of 0's and 1's having substring 001.

5. Construct a PDA to accept strings of a's and b's ending with ab or ba.

RL

1)

NPDA



$w = abbaba$

$\vdash_P$

$(q_0, abba, Z_0)$

(q_0, bba, aZ_0) $\quad$ $(q_1, abba, Z_0)$ reject

$(q_0, ba, Z_0) \rightarrow (q_0, a, bZ_0) \rightarrow (q_0, \epsilon, Z_0)$

(q_1, ϵ, Z_0) Accept

Practice Problems

1. Obtain a PDA to accept the language $L = \{ w \in \{a,b\}^* \mid N_a(w) = N_b(w) \}$

2. Obtain a PDA to accept the language $L = \{ w \in \{a,b\}^* : N_a(w) > N_b(w) \}$ by final state.

3. Obtain a PDA to accept the language $L = \{ w \in \{a,b\}^* : N_a(w) < N_b(w) \}$ by final state.

4. Design a PDA to accept all the strings of 0's and 1's having substring 001.

5. Construct a PDA to accept strings of a's and b's ending with ab or ba.

w = abba

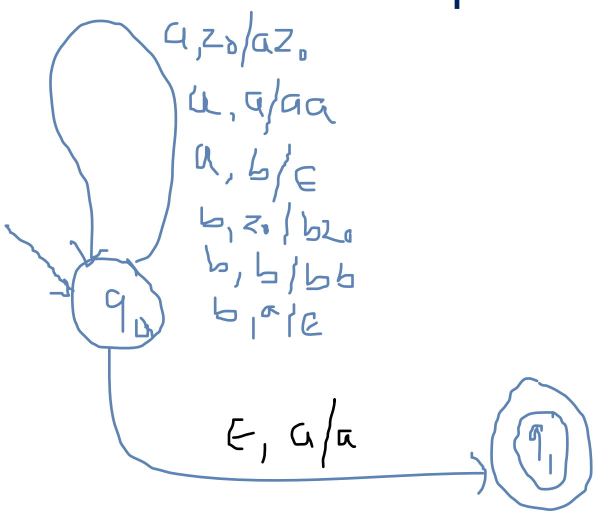
$|\epsilon| = 0$

Ex: aabbaaa

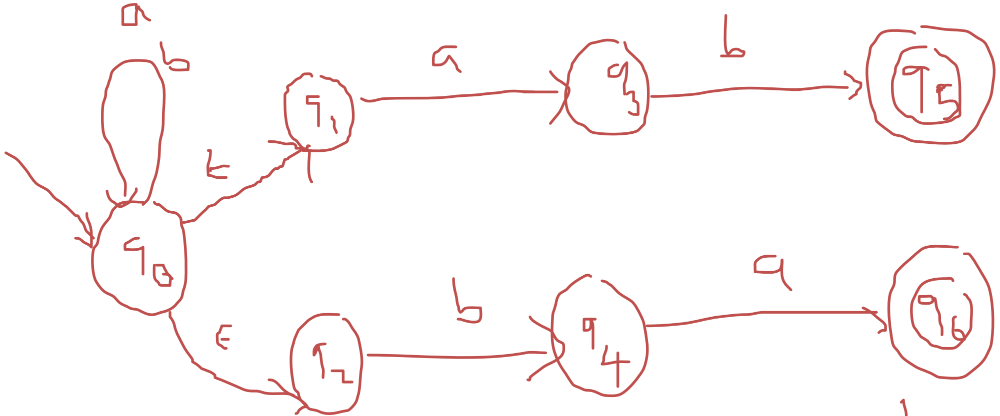
RL

w = abbabaa

NPDA

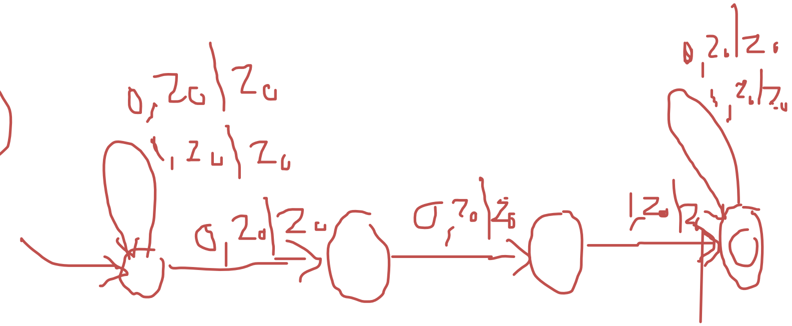


ab or ba



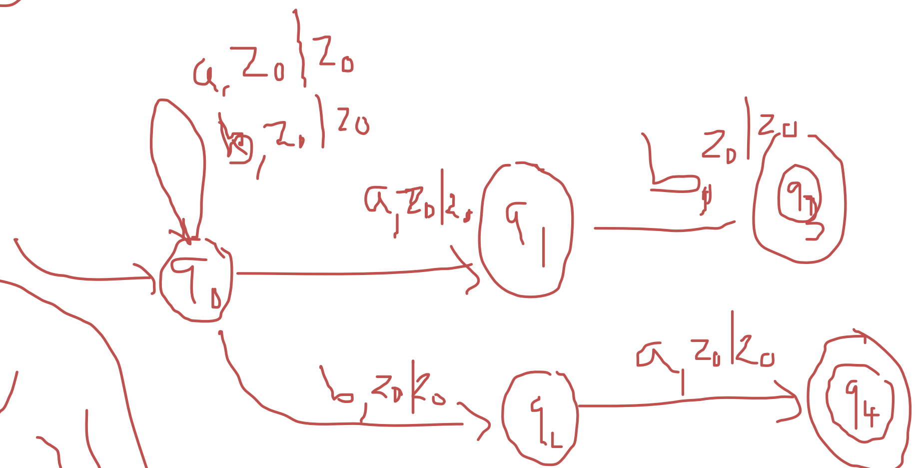
ε-NFA

6



PDA

5



$\delta(q_0, a, Z_0) = (q_0, Z_0)$

$\delta(q_0, b, Z_0) = \{(q_0, Z_0), (q_1, Z_0)\}$

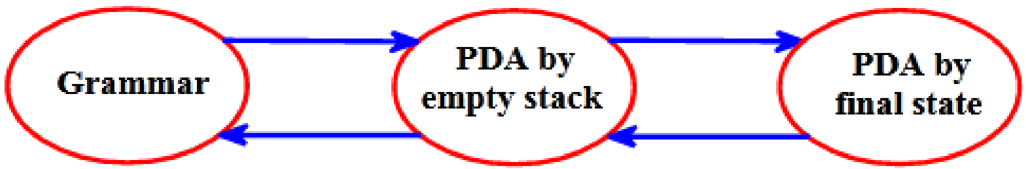
$\delta(q_0, a, Z_0) = (q_1, Z_0)$

$\delta(q_1, b, Z_0) = (q_3, Z_0)$

$\delta(q_2, a, Z_0) = (q_4, Z_0)$

PDA

Equivalence of PDA and CFG



From Grammar to Pushdown Automata

LMP

-Given a CFG G, we Can construct a PDA that simulates the leftmost derivations of G.

Let G= (V, T, P, S) be a CFG. Construct the PDA P that accepts L(G) by **empty stack**

such that L(G) = L(P), where P= ({ q }, T, V∪T, δ, q, S)

$P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$

CFG ⇒ PDA

Method:

1. For each variable A, define transitions:

   δ(q, ε, A) = { (q, β) | A→β is a production of P }

2. For each terminal a, define transition:

   δ(q, a, a) = {(q, ε)}

Convert the following grammar into Equivalent PDA.

S → aABC
A → aB | a
B → bA | b
C → a

⇒ PDA

$V = \{S, A, B, C\}$, $T = \{a, b\}$

$\delta(q, \epsilon, S) = (q, aABC)$

$\delta(q, \epsilon, A) = \{(q, aB), (q, a)\}$

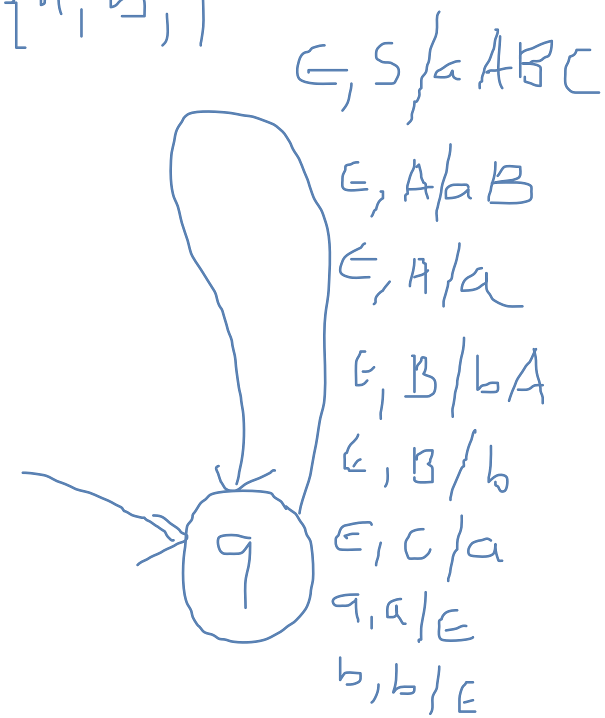
$\delta(q, \epsilon, B) = \{(q, bA), (q, b)\}$

$\delta(q, \epsilon, C) = (q, a)$

$\delta(q, a, a) = (q, \epsilon)$

$\delta(q, b, b) = (q, \epsilon)$

⟺



Practice Examples

1) For the grammar:

S→ aABB | aAA | **ε**
A→ aBB | a
B→ bBB | A
C → a

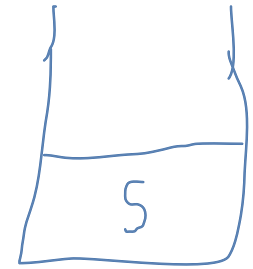
Obtain the corresponding PDA

a, a | ε
b, b | ε
ε, B | A
ε, C | a

ε, S | aABB
ε, S | aAA
ε, S | ε
ε, A | aBB
ε, A | a
ε, B | bBB

2) Convert the following CFG to PDA

S → aSa | bSb |aa | bb



THANQ….

Module-4

Content

- **Algorithms and Decision Procedures for CFLs:**
  - -Decidable questions
  - -Un-decidable questions.

CFL

- **Turing Machine:**
  - -Turing machine model, Representation,
  - -Language acceptability by TM,
  - -Design of TM,
  - -Techniques for TM construction. Variants of Turing Machines (TM),
  - -The model of Linear Bounded automata. (LBA)

TextBook-1: 14.1, 14.2 TextBook-2: 9.1 to 9.8

The Decidable Questions

$w \in L$ $L =$

-Membership

Given a CFL L and a string w, is w in L ? -- Can be answered

$L = \{ \quad \}$

-Emptiness

Given a CFL L , is L = ⦰ ? -- Can be answered

-Finiteness

Given a Context- Free Language L, is L infinite ? -- Can be answered

$L = \{a^n b^n : n \geq 0\}$ $L = \{w_1, w_2, \ldots \infty\}$

Membership

Algorithm: Using Grammar

decideCFLusingGrammar(L: CFL, w: string) =

1. if L is specified as a grammar G, simply use G.
2. if (w = ε) then if (S is nullable) then accept else reject.
3. if (w ≠ ε) then

   3.1. From G, construct G' such that L (G') = L(G)-{ε} and G' is in CNF.

   3.2. if G' derives w in $(2 * |w| - 1)$ steps then accept else reject.

Example: Suppose $L = \{ a^n b^n : n >= 1 \}$

$L = \{ ab, aabb, \ldots \}$

$S \rightarrow aSb \mid ab$

$w = aaabbb \in L$ $\quad$ $a + b - 1$ step

1 step

$A \rightarrow BC$

or

$A \rightarrow a$

$S \rightarrow ASB \mid AB$

$\Downarrow$

$S \rightarrow A_1B \mid AB$

$A_1 \rightarrow AS$

$A \rightarrow a$

$B \rightarrow b$

CNF

$S \Rightarrow A_1B \Rightarrow ASB$
$\Rightarrow aSB$
$\Rightarrow aA_1BB$
$\Rightarrow aASBB$
$\Rightarrow aaSBB$
$\Rightarrow aaABBB$
$\Rightarrow aaaBBB \Rightarrow aaabBB$
$\Rightarrow aaabbB \Rightarrow aaabbb$

$w = aabb$

$S \Rightarrow A_1B$
$\Rightarrow ASB$
$\Rightarrow aSB$
$\Rightarrow aABB$
$\Rightarrow aaBB$
$\Rightarrow aabB$
$\Rightarrow aabb \in L$

7 - st

Emptiness

Algorithm

Let *G = (V, T, P, S) be a context-free grammar that generates L. L(G) = Φ iff S is unproductive (i.e., not able to* generate any terminal strings). The following algorithm exploits the procedure *remove unproductive(non generating) symbols* to remove all unproductive non-terminals from *G. It answers the question, "Given a context-free* language *L, is L = Φ ?"*.

decideCFLempty*(G: context-free grammar) =*

1. Let *G' = removeunproductive(G).*
2. If *S is not present in G'*

   then return True

   else

   return False.

Example:

1) $S \rightarrow AB \mid Bb$
   $A \rightarrow a$

$G_s = \{a, b, A\}$

$N_G = \{S, B\}$

$\Rightarrow$

$G' =$

$A \rightarrow a$

$L(G) = \phi$

$A \rightarrow \alpha$

$A \rightarrow aBCd \mid BD$

2) $S \rightarrow AB \mid B$
   $A \rightarrow a \mid \varepsilon$
   $B \rightarrow b$

$G_s = \{a, b, A, B, S\}$

$L(G) \neq \phi$

Finiteness

is L finite or infine ?

Let G = (V, T, P, S) be a context-free grammar that generates L. is L infinite? There exist an algorithm to decide whether L is finite or infinite.

Algorithm

decideCFLinfinite*(G: CFG) =*

1. *Let G' = G with ∈, Unit and Useless productions removed.*
2. *Draw a directed graph whose nodes are variables of the G'.*
3. *if (graph contains a cycle)*

   then

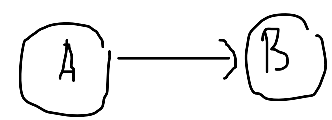
   return true;   // L is infinite

   else

   return false;   // L is finite

G'

$A \to aB$



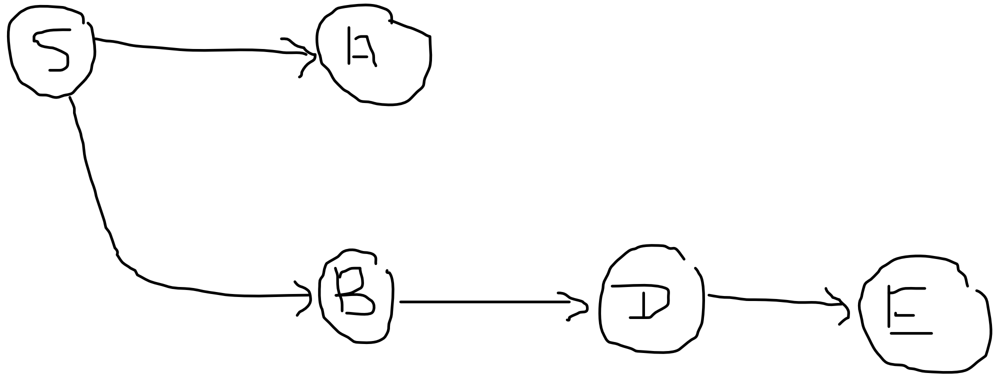
Example:

G:

S→ AB | ab
A → a
B→ aD | b
D→ bE
E → e

G ≡ G':

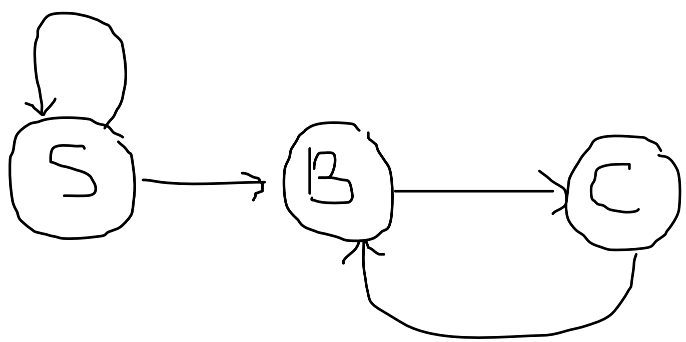
L(G) is finite



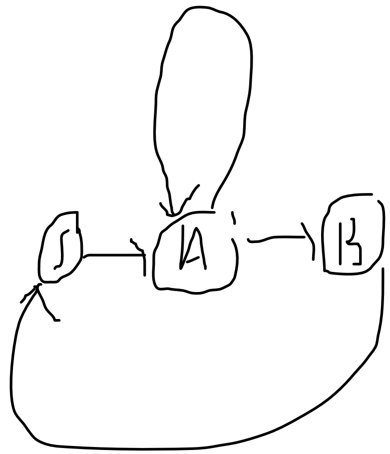
Directed Graph G

G:

S → aS | Bb
B → b | C
C → Bb



DG (cycle)



L(G) is infinite

∞

The Undecidable questions

- Given a Context-free language L, is $L = \Sigma^*$?
- Given a CFL L, is the complement of L context-free?
- Given two context-free languages L_1 and L_2 is $L_1 = L_2$?
- Given two context-free languages L_1 and L_2, is $L_1 \subseteq L_2$?
- Given two context-free languages $L_1$and L_2, is $L_1 \cap L_2 = \emptyset$?
- Given a context-free language L, is L inherently ambiguous?
- Given a context-free grammar G, is G ambiguous?

Note: No algorithms or Procedures exist for all the above Questions as of now!

Turing Machine(TM)

FSM
PDA
RE

In the early 1930s, mathematicians were trying to define effective computation.

Alan Turing in 1936, gave various models using the concept of Turing machines.

It is interesting to note that these were formulated much before the electro-mechanical/electronic computers were devised.

TM
↓

It has been universally accepted by computer scientists that the Turing machine provides an ideal theoretical model of a computer.

0 — REL
1 — CSL
2 — CFL
3 — RL

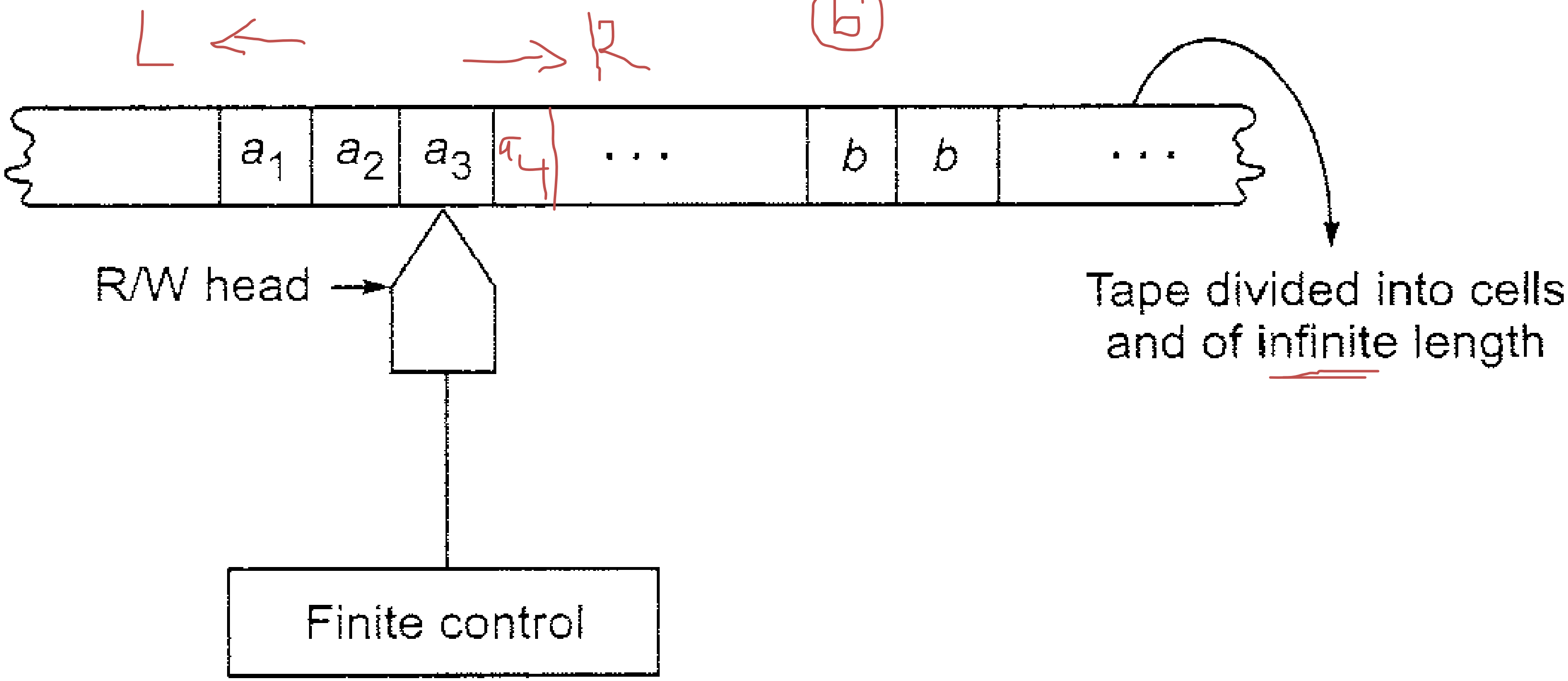
Turing machines are useful in several ways:

-As an automaton, the Turing machine is the most general model for accepting type-0 Languages
-It can also be used for computing functions etc…

x y
0101
± ALU

BASIC Model of TURING MACHINE(TM)

DTM
NTM



1936

RTM

Fig. Turing machine model

The Turing machine can be thought of as finite control connected to a R/W (read/write) head.

It has one tape which is divided into a number of cells. The block diagram of the basic model for the Turing machine is given in Fig.

Each cell can store only one symbol. The input to and the output from the finite state automaton are effected by the R/W head which can examine one cell at a time.

BASIC Model of TURING MACHINE(TM) Contd…

In one move, the machine examines the present symbol under the R/W head on the tape and the present state of an automaton to determine:

(i) a new symbol to be written on the tape in the cell under the R/W head,

(ii) a motion of the R/W head along the tape: either the head moves one cell left (L) or one cell right (R),

(iii) The next state of the automaton, and

(iv) whether to halt or not.

q_1

Definition:

q_1

$\Sigma = \{a, b\}$

$\Sigma = \{0, 1\}$

A Turing machine *M is a 7-tuple, namely* $(Q, \Sigma, \Gamma, \delta, q_o, b, F)$, where;

1. Q is a finite nonempty set of states.
2. Γ is a finite nonempty set of tape symbols.
3. $b \in \Gamma$ is the blank.
4. Σ is a nonempty set of input symbols and is a subset of Γ and $b \notin \Sigma$.
5. δ is the transition function mapping (q, x) onto (q', y, D) where D denotes the direction of movement of R/W head: $D = L$ or R according as the movement is to the left or right.
6. $q_0 \in Q$ is the initial state, and
7. $F \subseteq Q$ is the set of final states.

$\Sigma \subseteq \Gamma$ $\quad b \notin b$

D

$(q, x) \rightarrow (q', y, L/R)$

REPRESENTATION OF TURING MACHINES

IDs

We can describe a Turing machine by employing:

1. INSTANTANEOUS DESCRIPTIONS(IDs) Moves TM
2. TRANSITION DIAGRAM(TD)
3. TRANSITION TABLE (TT)

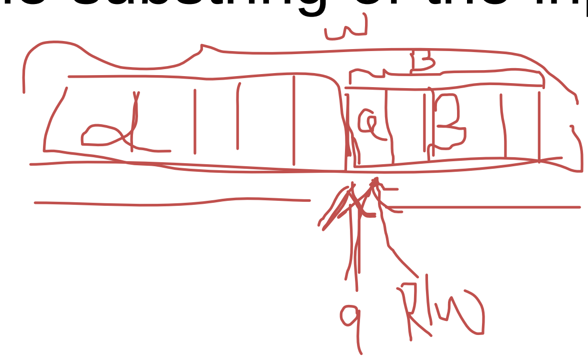
REPRESENTATION BY INSTANTANEOUS DESCRIPTIONS(IDs)

Snapshots' of a Turing machine in action can be used to describe a Turing machine. These give 'instantaneous descriptions' of a Turing machine.

w, q,

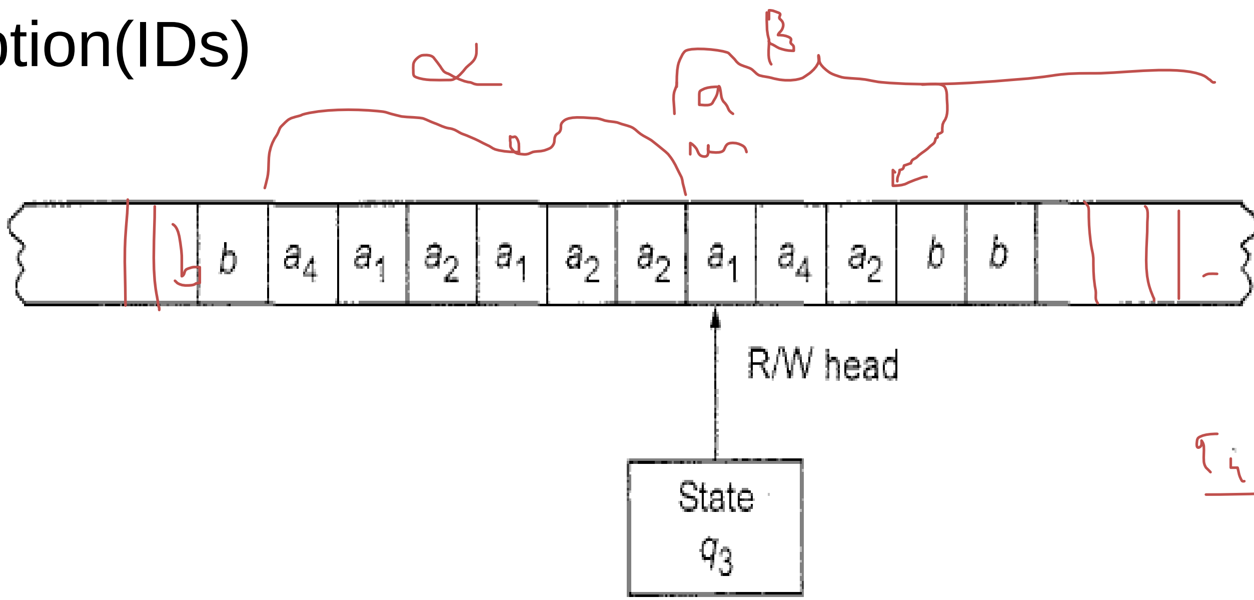
An ID of a Turing machine is defined in terms of the entire input string and the current state.

An ID of a Turing machine M is a string **αqβ**, where q is the present state of M, the entire input string is split as **αβ** , the first symbol of **β** is the current a symbol under the R/W head and **β** has all the subsequent symbols of the input string and the string **α** is the substring of the input string formed by all the symbols to the left of a.



EXAMPLE:

A snapshot of Turing machine is shown in Fig., obtain the instantaneous description(IDs)



$\alpha q \beta$

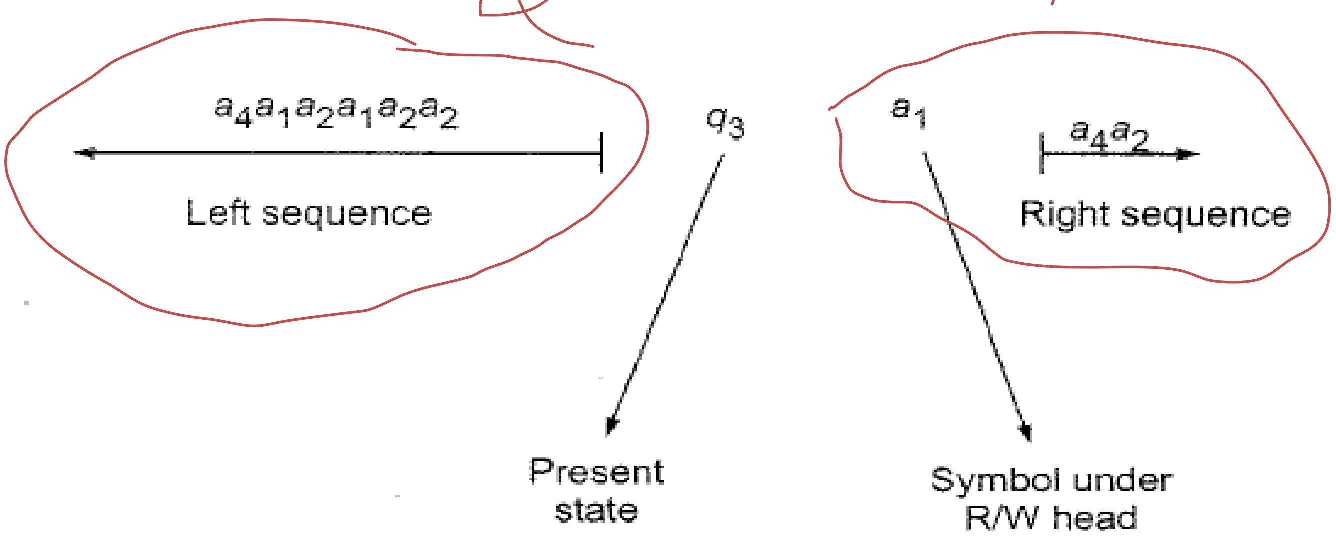
R/W

$\underbrace{a_4 a_1 a_2 a_1 a_2 a_2}_{\alpha} q_3 \underbrace{a_1 a_4 a_2}_{\beta}$

Solution

The present symbol under the R/W head is a_1. The present state is q_3. So a_1 is written to the right of q_3. The nonblank symbols to the left of a_1 form the string $a_4a_1a_2a_1a_2a_2$, which is written to the left of q_3. The sequence of nonblank symbols to the right of a_1 is a_4a_2. Thus the ID is as given in Fig.

Representation of ID



Notes: (1) For constructing the ID, we simply insert the current state in the input string to the left of the symbol under the R/W head.

(2) We observe that the blank symbol may occur as part of the left or right substring.

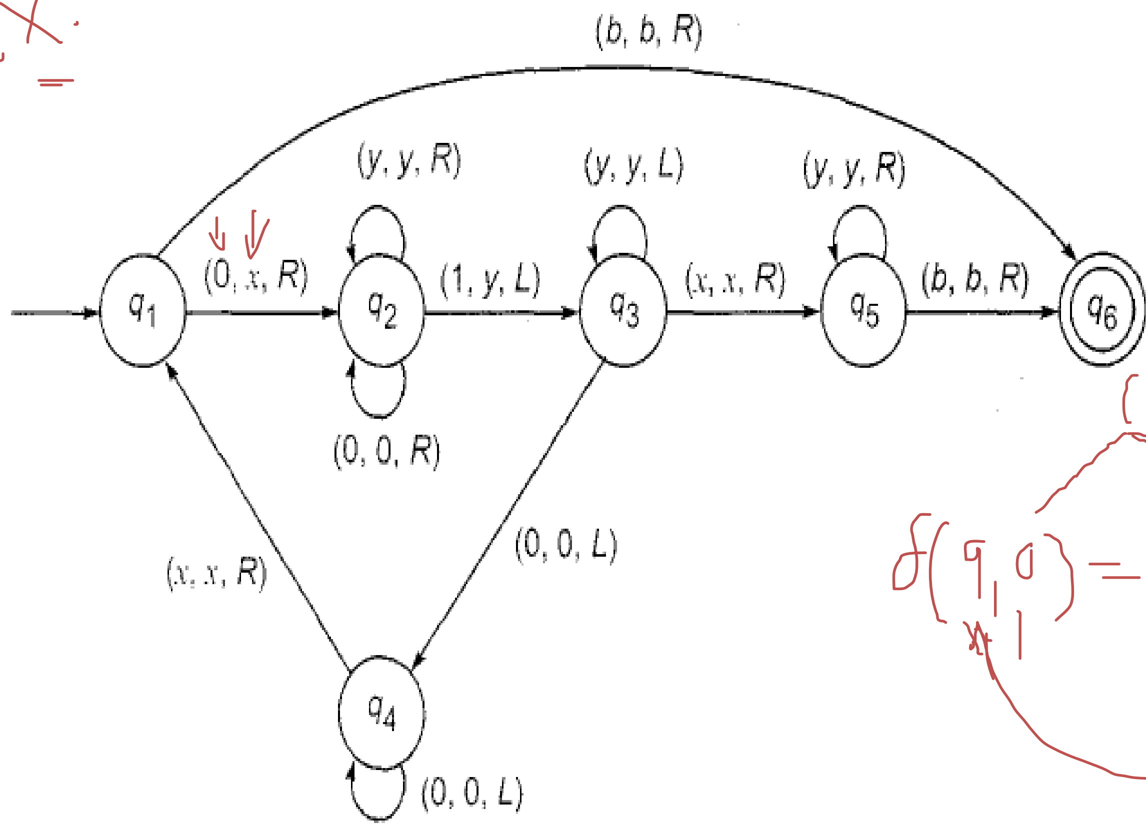
REPRESENTATION BY TRANSITION DIAGRAM

We can use the transition diagram to represent Turing machines.

The states are represented by vertices. Directed edges are used to represent transition of states. Each edge has label described by triple (x, y, D).

$$\delta(p, x) = (y, q, R)$$

Ex:



Transition Diagram

(q, x, R)

$\delta(q_1, 0) = (x, q_2, R)$

tape symbol

δ	b	0	1	y	x
q_1	bRq_6	xRq_2	–	–	–
q_2	–	$0Rq_2$	yLq_3	yRq_2	–

REPRESENTATION BY TRANSITION TABLE

We give the definition of **δ** in the form of a table called the transition table

Consider, for example, a Turing machine with five states q_1, ..., q_5, where q_1 is the initial state and q_5 is the (only) final state. The tape symbols are 0, 1 and b. The transition table given in table below describes δ.

$\Sigma \subseteq \Gamma$

$\Sigma = \{0, 1\}$

Present state	Tape symbol b	Tape symbol 0	Tape symbol 1
$\rightarrow q_1$	$1Lq_2$	$0Rq_1$	–
q_2	bRq_3	$0Lq_2$	$1Lq_2$
q_3	–	bRq_4	bRq_5
q_4	$0Rq_5$	$0Rq_4$	$1Rq_4$
q_5	$0Lq_2$	–	–

$q_1 \xrightarrow{(b,1,L)} q_2$ $\qquad$ $q_2 \xrightarrow{(1,1,L)} q_2$ $\qquad$ $q_1 \xrightarrow{(0,0,R)} q_1$

Transition Table

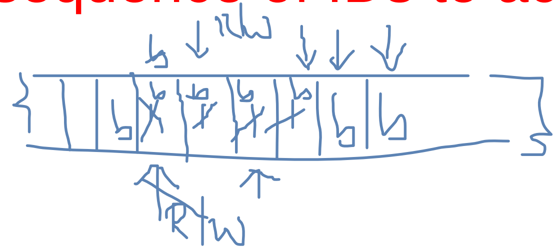
The initial state is marked with → and the final state ◯

DESIGN OF TURING MACHINES

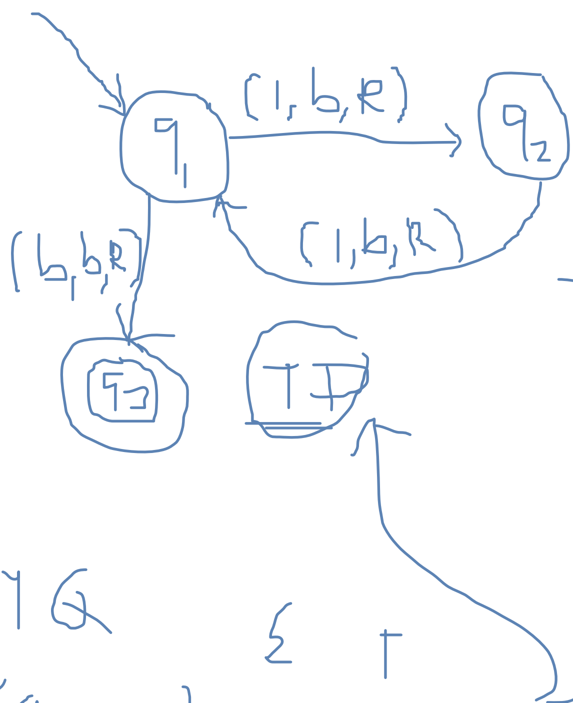
RL

Design a Turing Machine to recognize all strings consisting of even number of 1's. Obtain the sequence of IDs to accept string : 1111

$M = (Q, \Sigma, \Gamma, \delta, q_o, b, F)$



$\Sigma = \{1\}$



	b	1
$\rightarrow q_1$	bRq_3	bRq_2
q_2	—	bRq_1
q_3	—	—

$q_1 1111 \vdash bq_2 111 \vdash bbq_1 11 \vdash bbbq_2 1 \vdash bbbbq_1 b \vdash bbbbbq_3$

$\therefore 1111 \in L$

formally

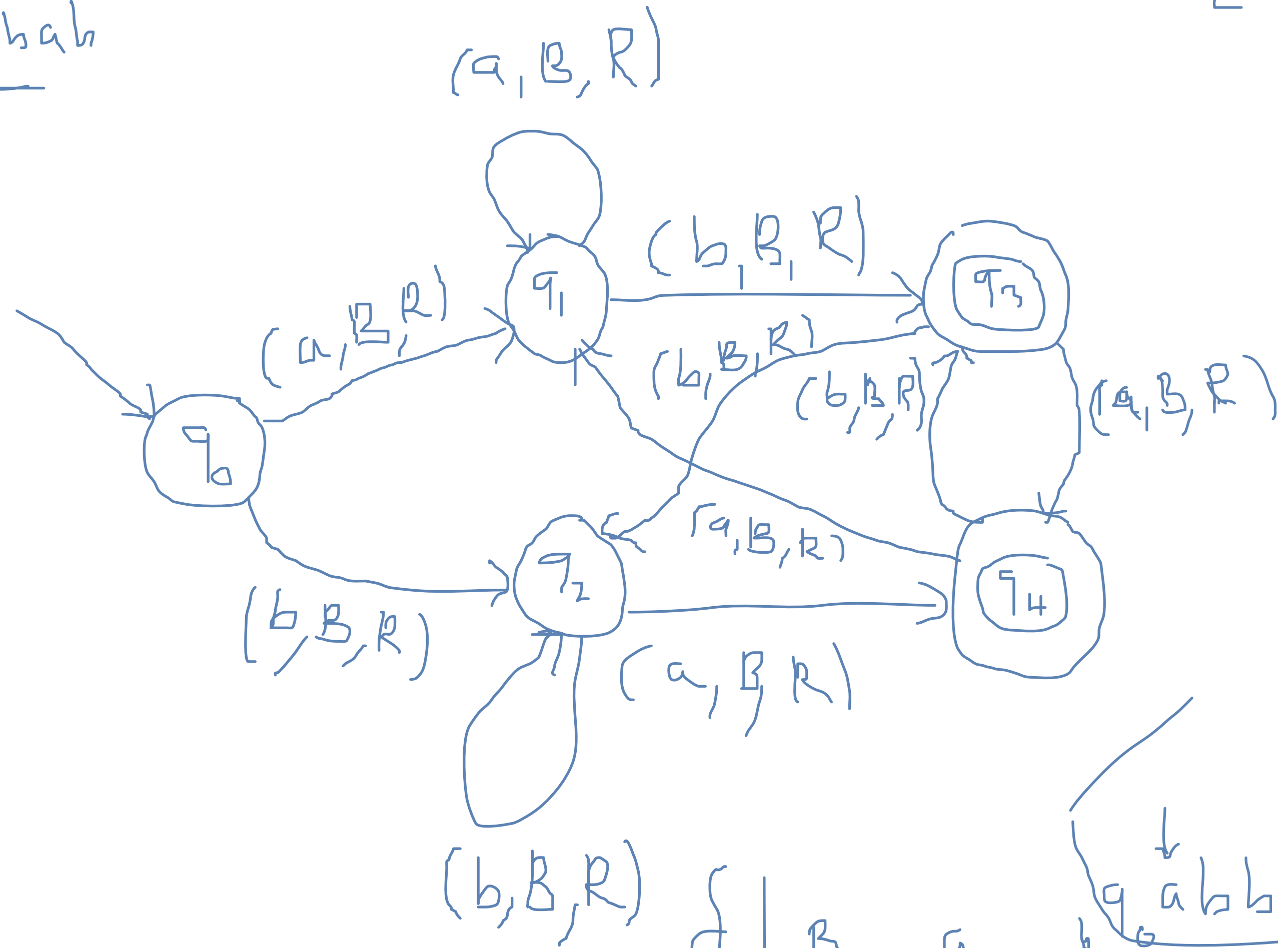
$M = (\{q_1, q_2, q_3\}, \{1\}, \{1, b\}, \delta, q_1, b, \{q_3\})$

BM

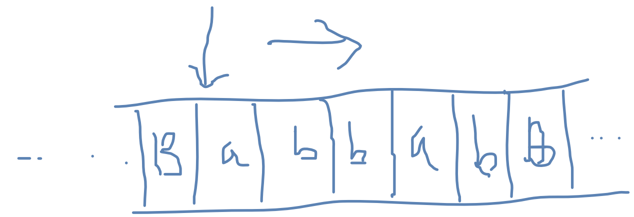
Design a Turing Machine to accept strings of a's and b's ending with ab or ba

$\Sigma = \{a, b\}$ B → blank

abbab



DFSM



q_0 abbab

δ	B	a	b
→ q_0	—	B R q_1	B R q_2
q_1	—	B R q_1	B R q_3
q_2	—	. .	. .
q_3	—	.	.
q_4	—	.	.

TT

Practice Problems

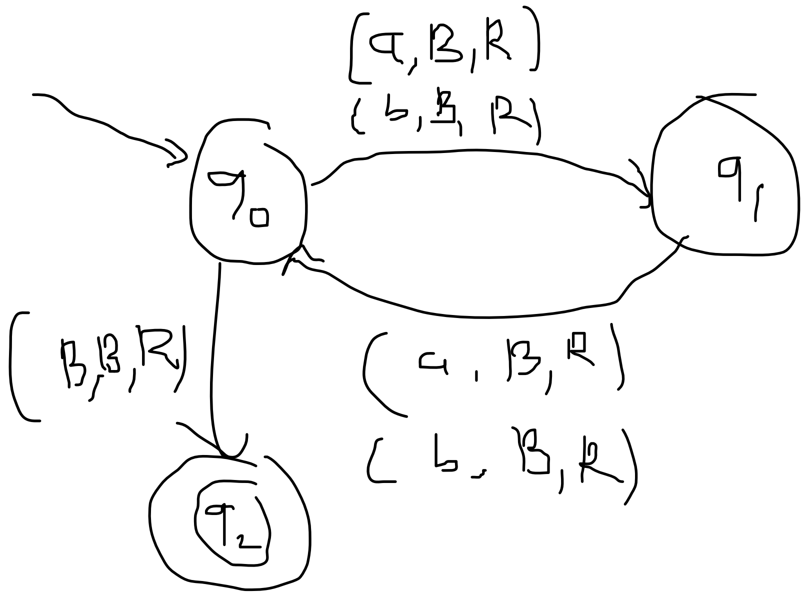
1. Design a Turing Machine to accept the L={ w : |w| is even and w consisting of a's and b's }

2. Design a Turing Machine to accept the language containing strings of 0's and 1's ending with 011}

RL

3. Design a Turing Machine to accept the language L = { w | w ∈ {0,1}* } containing the substring 001

1)



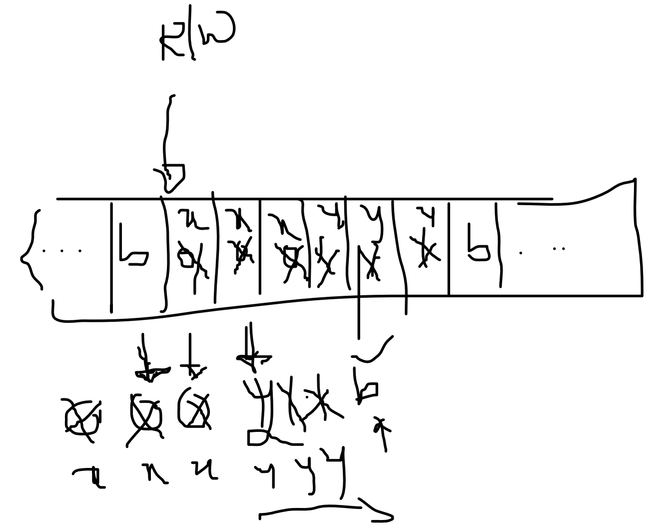
Design a TM that accept $L = \{ 0^n1^n \mid n \geq 1 \}$

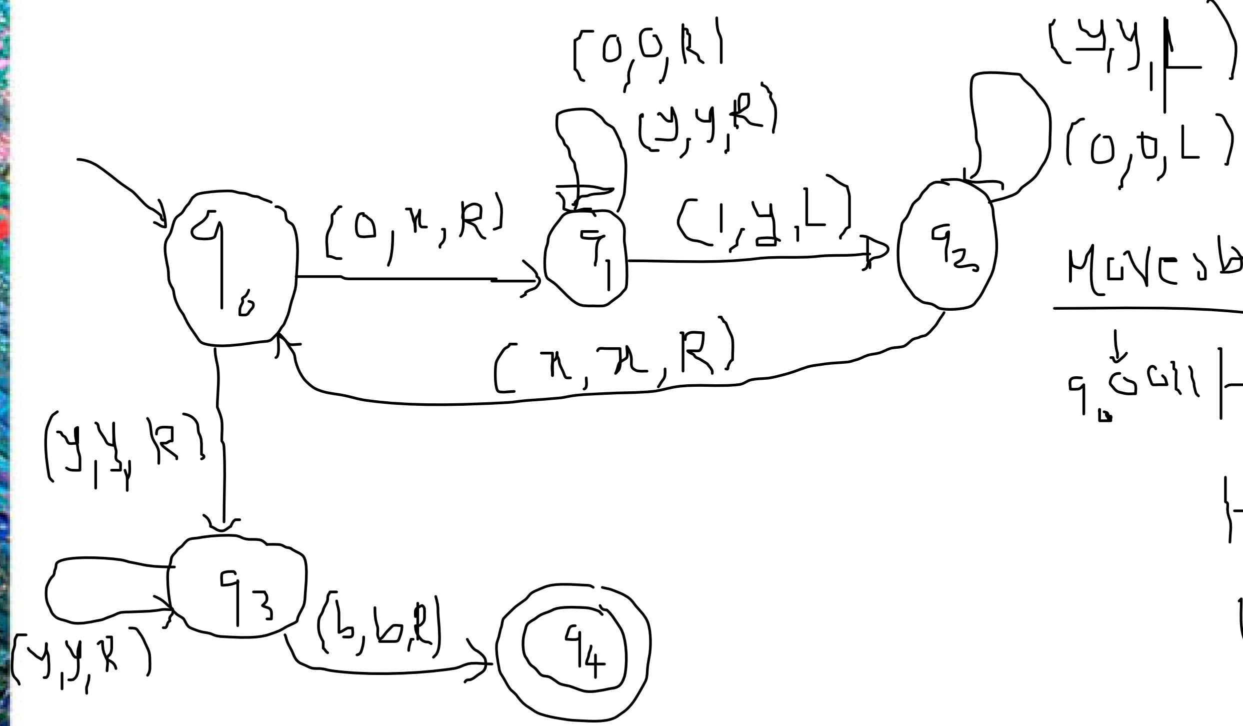
TM

$n \geq 0$

$w = 000111$

$\alpha q \beta$





TP

Moves by TM : w:0011

$q_0 0011 \vdash x q_1 011 \vdash x0 q_1 11 \vdash x q_2 0y1$

$\vdash q_2 x0y1 \vdash x q_0 0y1 \vdash xx q_1 y1$

$\vdash xxy q_1 1 \vdash xx q_2 yy \vdash x q_2 xyy$

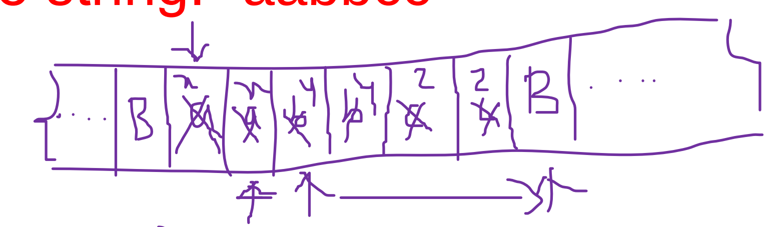
$\vdash xx q_0 yy \vdash xxy q_3 y$

$\vdash xxyy q_3 b \vdash xxyyb q_4$

(accept)

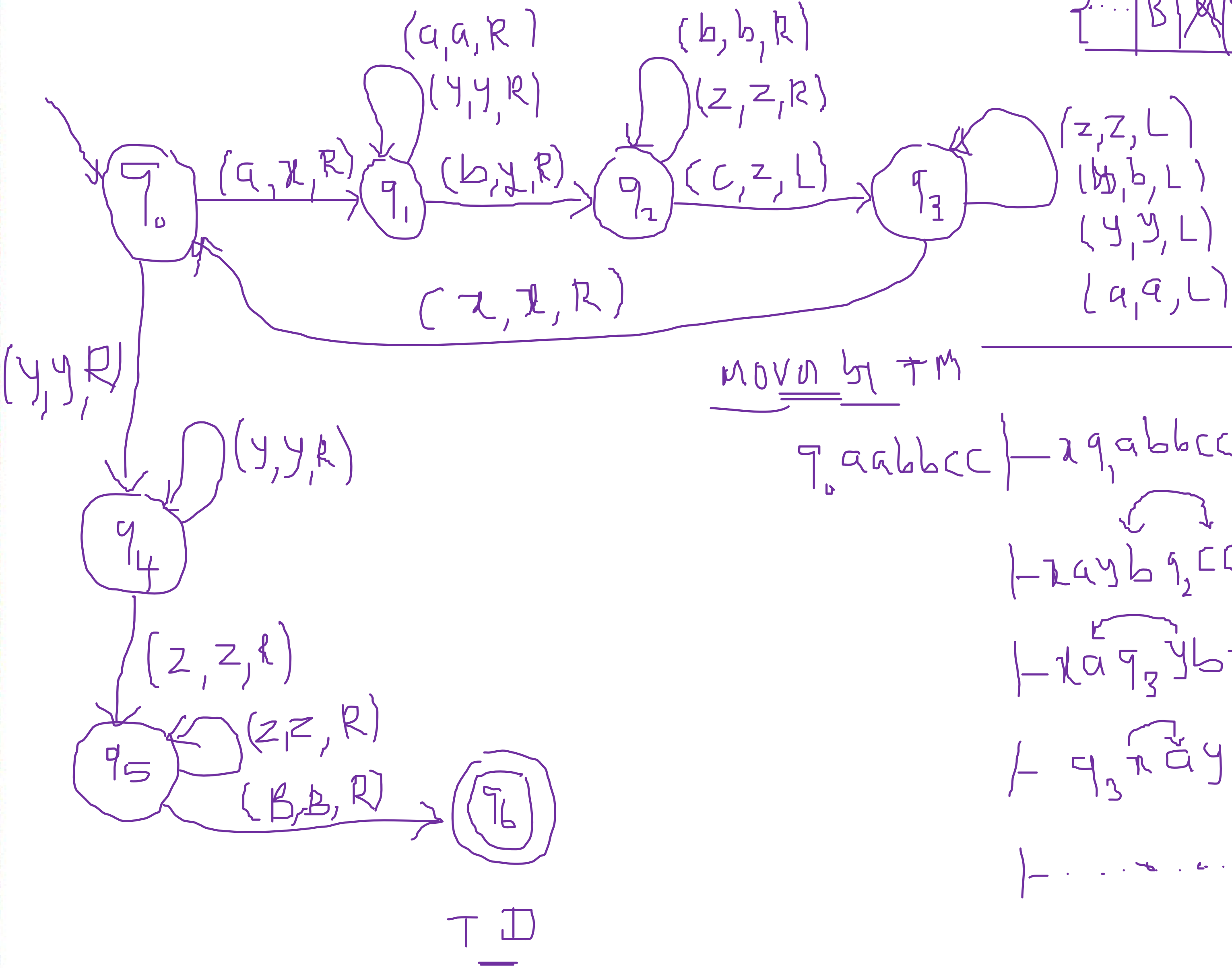
$n \geq 0$

Design a TM to accept the language $L = \{ a^n b^n c^n \mid n \geq 1 \}$. Draw the Transition diagram and show the moves made by TM for the string: "aabbcc"



$\Sigma = \{a, b, c\}$

$\Gamma = \{a, b, c, x, y, z, B\}$



Moves by TM

$q_0 aabbcc \vdash x q_1 abbcc \vdash x a q_1 bbcc \vdash x a y q_2 bcc$

$\vdash x a y b q_2 cc \vdash x a y q_3 b z c$

$\vdash x a q_3 y b z c \vdash x q_3 a y b z c$

$\vdash q_3 x a y b z c \vdash x q_0 a y b z c$

$\vdash \ldots\ldots$

$\vdash x x y y z z q_5 B$

$\vdash x x y y z z B q_6$ ✓

Design a Turing Machine to accept the language $L = \{ 1^n 2^n 3^n \mid n \geq 1 \}$

$0^n 1^n 2^n \mid n \geq 1$

$4^n 5^n 6^n \mid n \geq 1$

HW

$\omega = 112233$

Design a TM to accept the following Languages:

1) $L = \{ w_c w^R \mid w \in \{ a,b \}$ and w^R reverse of w$\}$

LANGUAGE ACCEPTABILITY BY TM

The Language accepted by TM is defined as follows.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, b, F)$ be a Turing Machine. The Language L(M) accepted by M is defined as:

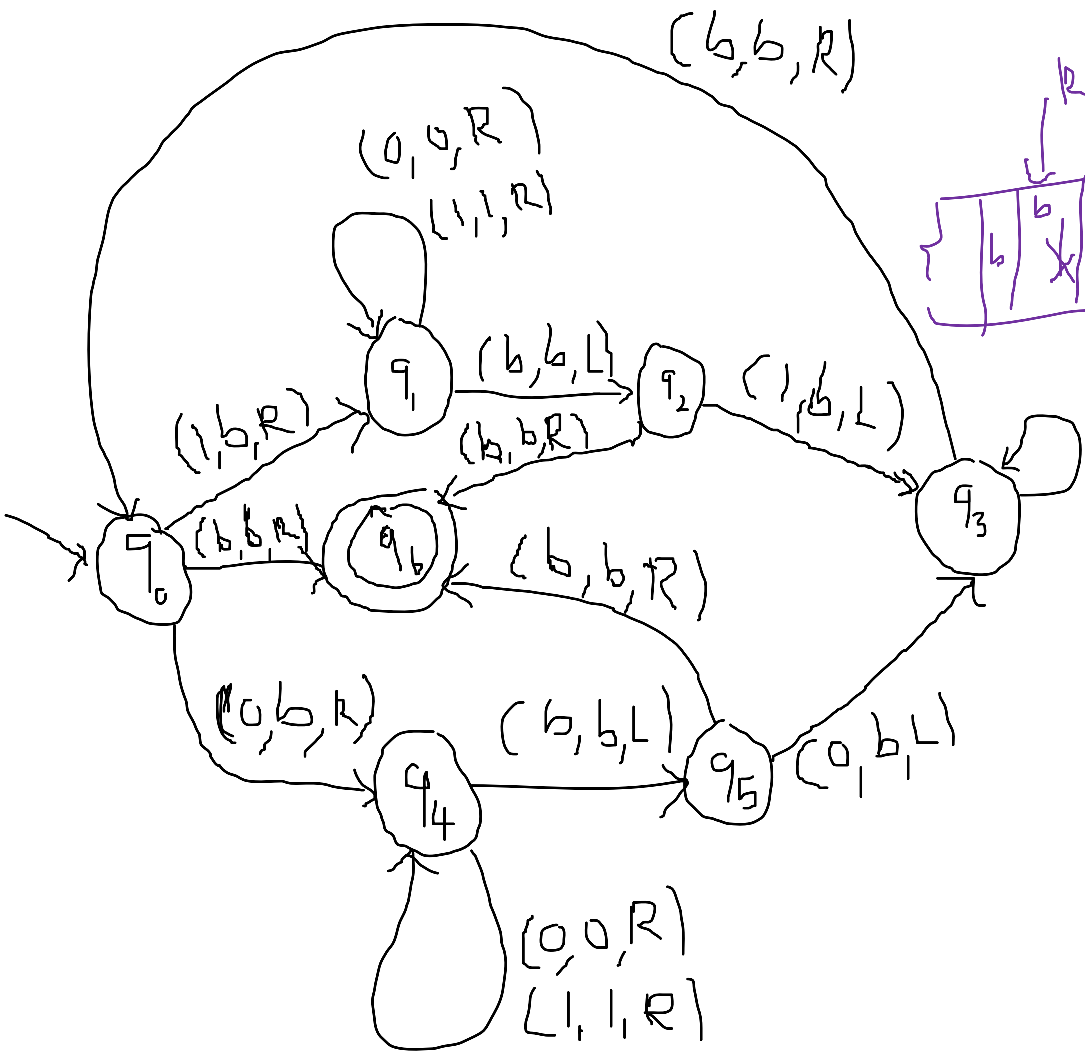
$L(M) = \{ w \mid q_0 w \vdash^* \alpha p \beta$, where $w \in \Sigma^*$, $p \in F$ and $\alpha, \beta \in \Gamma$ $\}$

Note: The TM can do one of the following:

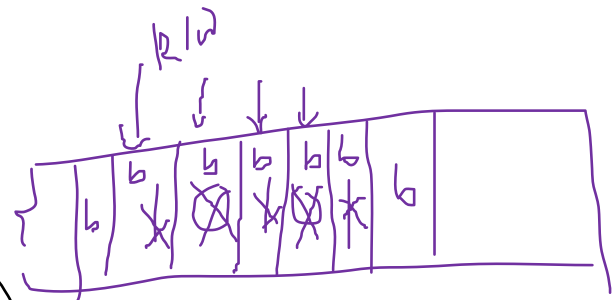
1. **Halt and accept by entering into final state**
2. **Halt and reject . This is possible if the transition is not defined.**
3. **TM will never halt and enters into an infinite loop.**

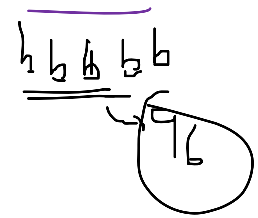
No algorithms exist to determine and tell whether TM always halts → Undecidable Problem

Design a TM to accept all set of palindromes over { 0,1}. Also draw the transition diagram and Instantaneous Description(IDs) on "10101".



TD





IDs

$q_0 10101 \vdash b q_1 0101 \vdash b0 q_1 101b \vdash b01 q_1 01b$

$\vdash b010 q_1 1b \vdash b0101 q_1 b \vdash b010 q_2 1b$

$\vdash b01 q_3 0b \vdash b0 q_3 10b \vdash b q_3 010b$

$\vdash q_3 b010b \vdash b q_0 010b \vdash \dots$

$\vdash \dots \vdash bb q_0 1 bb \vdash \dots$

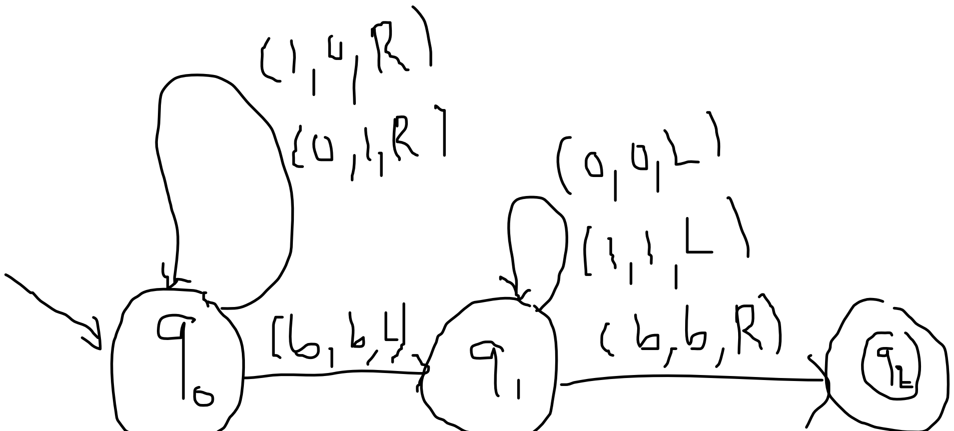
TM to Compute a function

Transducers

Add
Sub
Comp

Examples:

Obtain a Turing Machine to compute 1's Complement of a given binary number.



TD

101 ⇒ 010

b1010b ⊢ b0010b

q_0 q_0

⊢ b0110b ⊢ b0100b ⊢ b0101b

q_0 q_0 q_0

⊢ b0101b ⊢ b0101b ⊢ b0101b ⊢ b0101b

q_1 q_1 q_1 q_1

⊢ b0101b ⊢ 0101

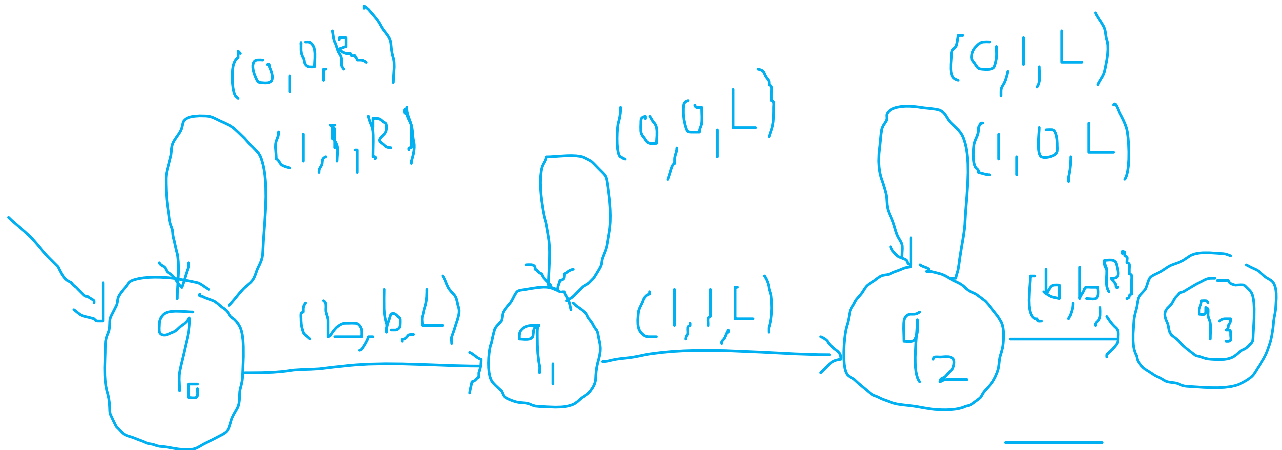
q_1 q_2

Obtain a Turing Machine to compute 2's Complement of a given binary number.

0110010

⇓

$\boxed{1001110}$



T.D

Move by TM:

IP

$q_0\,0110010 \vdash 0\,q_0\,110010 \vdash 01\,q_0\,10010$

$\vdash 011\,q_0\,0010 \vdash 0110\,q_0\,010 \vdash 01100\,q_0\,10$

$\vdash 011001\,q_0\,0 \vdash 0110010\,q_0\,b \vdash 011001\,q_1\,0$

$\vdash 01100\,q_1\,10 \vdash 0110\,q_2\,010 \vdash 011\,q_2\,0110$

$\vdash 01\,q_2\,1110 \vdash 0\,q_2\,10110 \vdash q_2\,0\,00110$

$\vdash q_2\,b\,1001110 \vdash b\,q_3\,1001110$ Accept

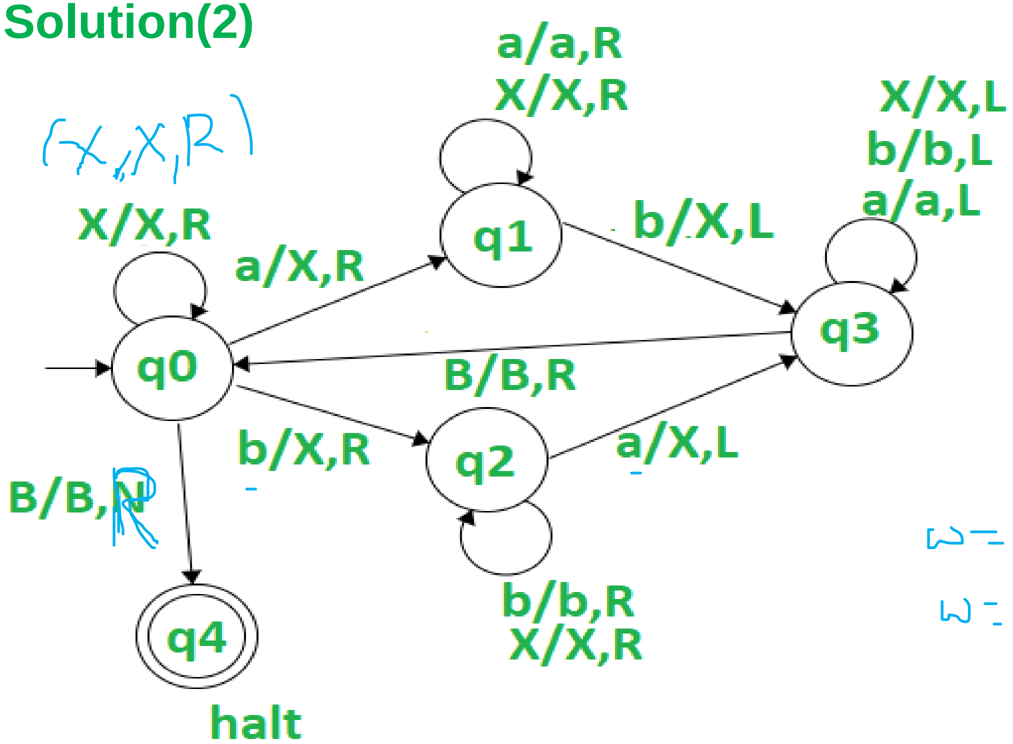
$M = (\{q_0, q_1, q_2, q_3\}, \{0, 1\}, \{b, 0, 1\}, \delta, q_0, b, \{q_3\})$

SUPPLEMENTARY EXAMPLES(Languages)

$a^n b^n$

1. Design a Turing Machine to accept the language $L = \{ a^n b^{2n} \mid n \geq 1 \}$
2. Design a Turing Machine to accept the language $L = \{ w \mid n_a(w) = n_b(w) \}$
3. Design a TM that reads a string in $\{0, 1\}^*$ and erases the rightmost symbol.

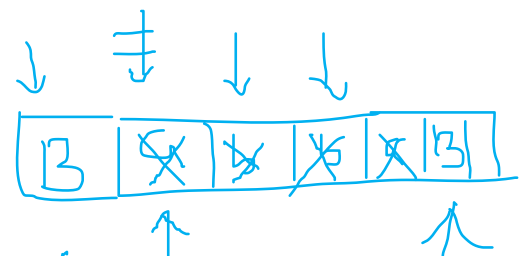
Solution(2)



abba

$n_a(abba)=2$

$n_b(abba)=2$



$w = abaabb \in L$

$w = aba \notin L$

Acceptance by TM

Accept Input String ✓

If machine halts
in an Final/accept state

Reject Input String ✗

If machine halts
in a non-Final state
or
If machine enters
an *infinite loop*

TECHNIQUES FOR TM CONSTRUCTION

-In this section: some high-level conceptual tools to make the construction of TMs easier for addressing simple/complex problems.

BASIC

-The TM defined & studied till now is called the standard TM(single Tape)

There are ~~4~~ 5 Techniques:

1. TURING MACHINE WITH STATIONARY HEAD
2. STORAGE IN THE STATE
3. MULTIPLE TRACK TURING MACHINE
4. SUBROUTINES
5. CHECKING OF SYMBOLS

RE
type-0
CSL-1
CFL-2
RL-3

A standard TM is capable of accepting some of the languages, called Recursively Enumerable(RE) language. But by doing some kind of modifications, we can increase the number of languages accepted by Turing Machine.

1. Stationary Head

$q \xrightarrow{(a,y,L|R)} q'$

- In the definition of a TM we defined $\delta(q, a)$ as (q', y, D) where $D = L$ or R.

S

D = L, R or S

- Suppose, we want to include the option that the head can continue to be in the same cell for some input symbol. Then we define $\delta(q, a)$ as (q', y, S). $\delta(q, a) = (q', y, S)$
- This means that the TM, on reading the input symbol a, changes the state to q' and writes y in the current cell in place of a and continues to remain in the same cell.

$q \xrightarrow{(a,y,S)} q'$

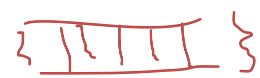
- In terms of IDs,

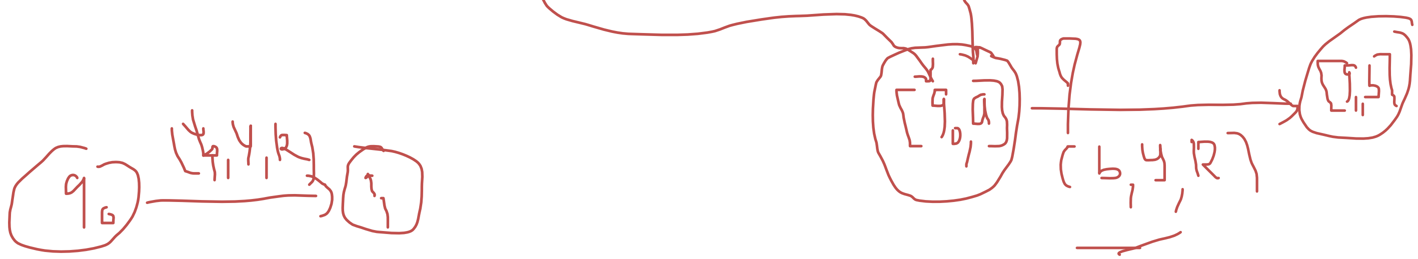
$$\underbrace{w}_{\alpha}q\underbrace{ax}_{\beta} \vdash wq'yx$$

Thus in this model $\delta(q, a) = (q', y, D)$ where $D = L, R$ or S.

2.Storage in the State

- State is used in FA or PDA or TM, to 'remember' things.
- We can use a state to store a symbol as well. So the state becomes a pair (q, a) where q is the state and a is the tape symbol stored in (q, a). So the new set of states becomes Q x Γ.





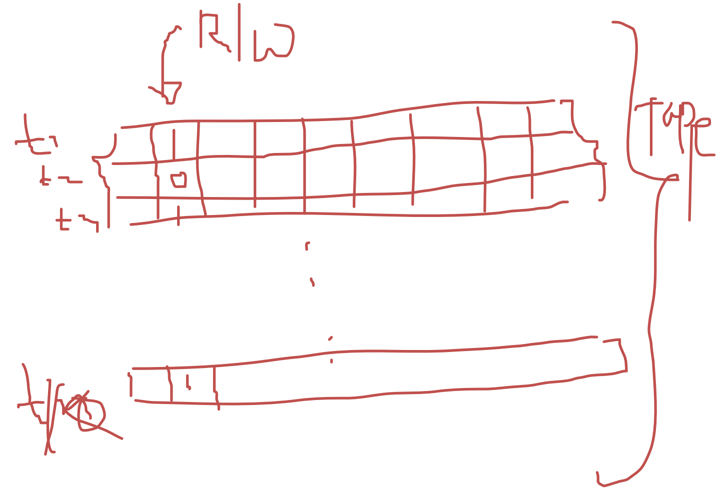
3. MULTIPLE TRACK TURING MACHINE

- In a multiple track TM, a single tape is assumed to be divided into several tracks.
- So, tape alphabet Γ is required to consist of k-tuples of tape symbols, *k being the number of tracks*.
- Hence the only difference between the standard TM and the TM with multiple tracks is the set of tape symbols.
- Standard Turing machine:

  Tape symbols are - elements of Γ;

- TM with multiple track:

  Tape symbols are- Γ^k =1, 10

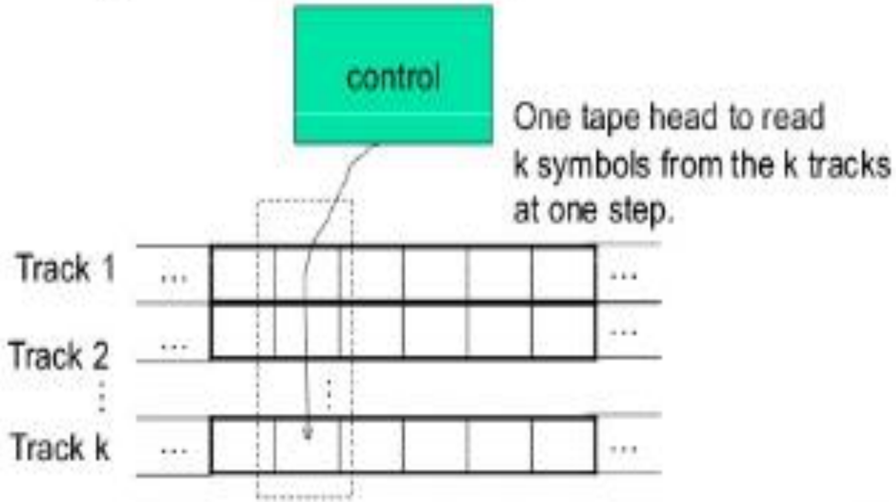


$\Gamma^1 = \{ \quad \}$

$\Gamma^2 = \{ \quad \}$

$\Gamma^3 = \{ \quad \}$

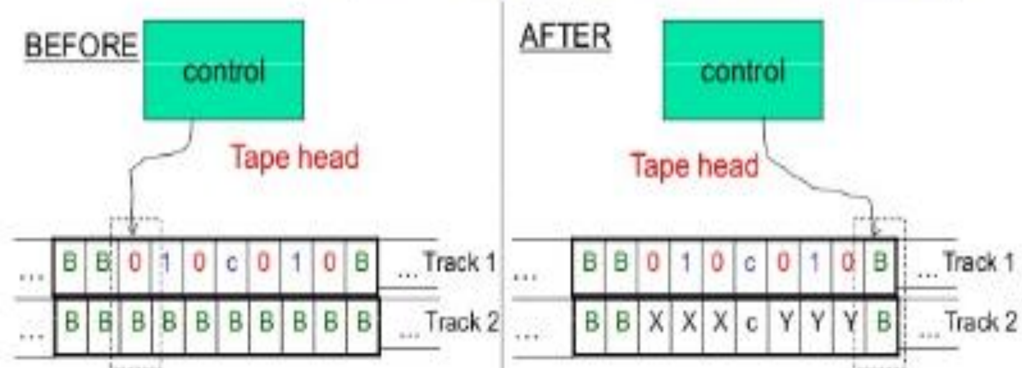
- TM with multiple tracks,
  but just one unified tape head



$\{wcw \mid \in \{0,1\}^*\}$

- TM with multiple "tracks" but just one head

E.g., TM for {wcw | w∈ {0,1}* }
but w/o modifying original input string



Second track mainly used as a scratch space for marking

Function
Mem.

D.N-Q

4. SUBROUTINES

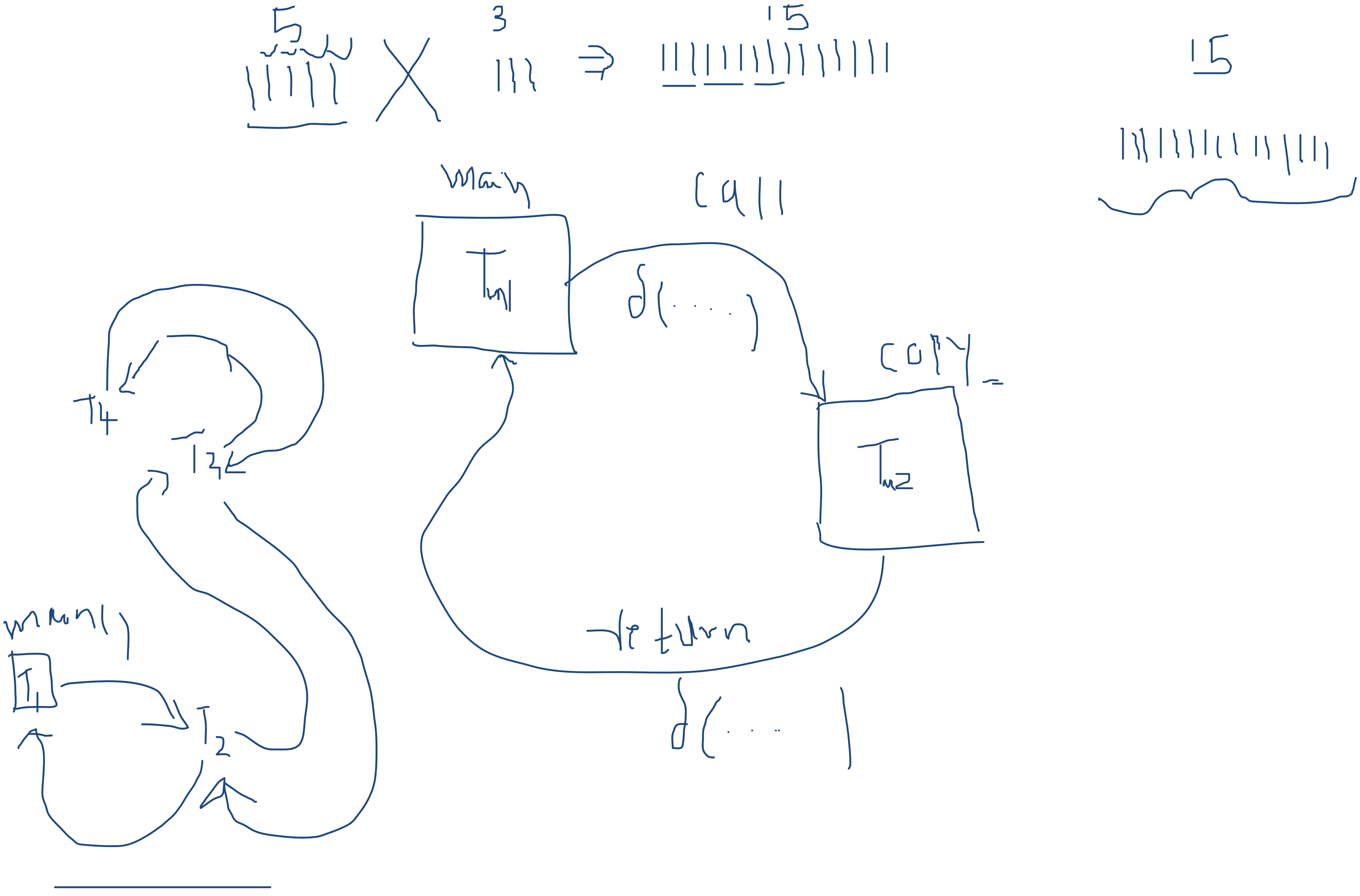
- Subroutine- some task has to be done repeatedly.
- TM program for the subroutine has an initial state and a 'return' state. After reaching the return state, there is a temporary halt.
- For using a subroutine, new states are introduced. When there is a need for calling the subroutine, moves are effected to enter the initial state for the subroutine (when the return state of the subroutine is reached) and to return to the main program of TM.

Ex: Design a TM for performing multiplication of two positive integers.

$T_1 \quad T_2$

$\delta(q, x) = (q_0, x, D)$

Explanation with Example: TM to perform multiplication of two positive integers



Variants of Turing machines

DTM
NTM

The Turing machine we have introduced has a single tape. **δ**(*q, a*) *is either a* single triple *(p, a, D), where D = R or L, or is not defined.*

In this section, we introduce two new models of TM:

$q \xrightarrow{(q_1, a, D)} q_1$

(i) a TM with more than one tape.

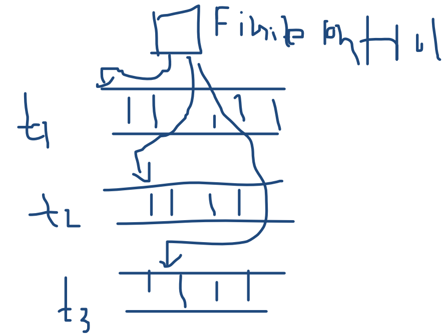
(ii) a TM where $\boldsymbol{\delta}(q, x) = \{(q_1, y_1, D_1), \{(q_2, y_2, D_2), \ldots ,(q_n, a_n, D_n) \}$

The first model is called a Multitape Turing Machine and the second a Nondeterministic Turing Machine. (NTM)

i) Multitape Turing machines

D = R, L, or S

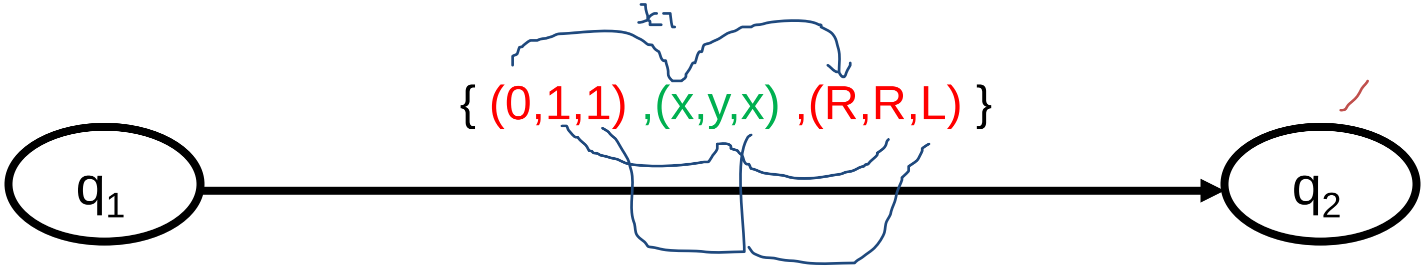
A multitape TM has k tapes, each with its own read/write head. Initially, the input is written on the first tape, and all the other tapes are blank, with each head at the beginning of the corresponding tape.



For a 3-tape TM, a transition will look like

(0, 1, R)

t1

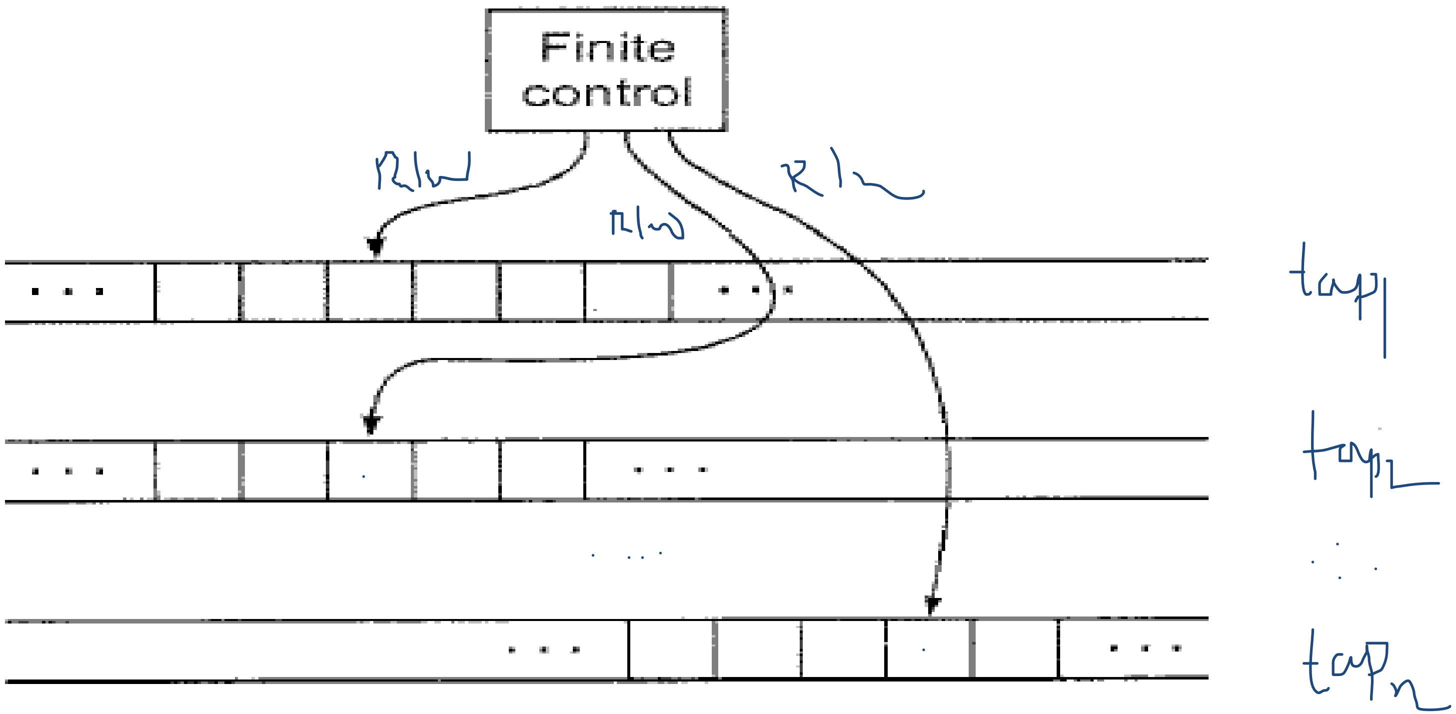


If you are in state q1 and see 0 on $Tape_1$, 1 on $Tape_2$ and 1 on $Tape_3$,

- replace x on $Tape_1$, y on $Tape_2$ and x on $Tape_3$;
- move $Head_1$ right, $Head_2$ right and $Head_3$ left;
- go to state q2.

Structure of multitape TM

A move depends on the current state and k tape symbols under k tape heads.



In a typical move:

(i) *M enters a new state.*

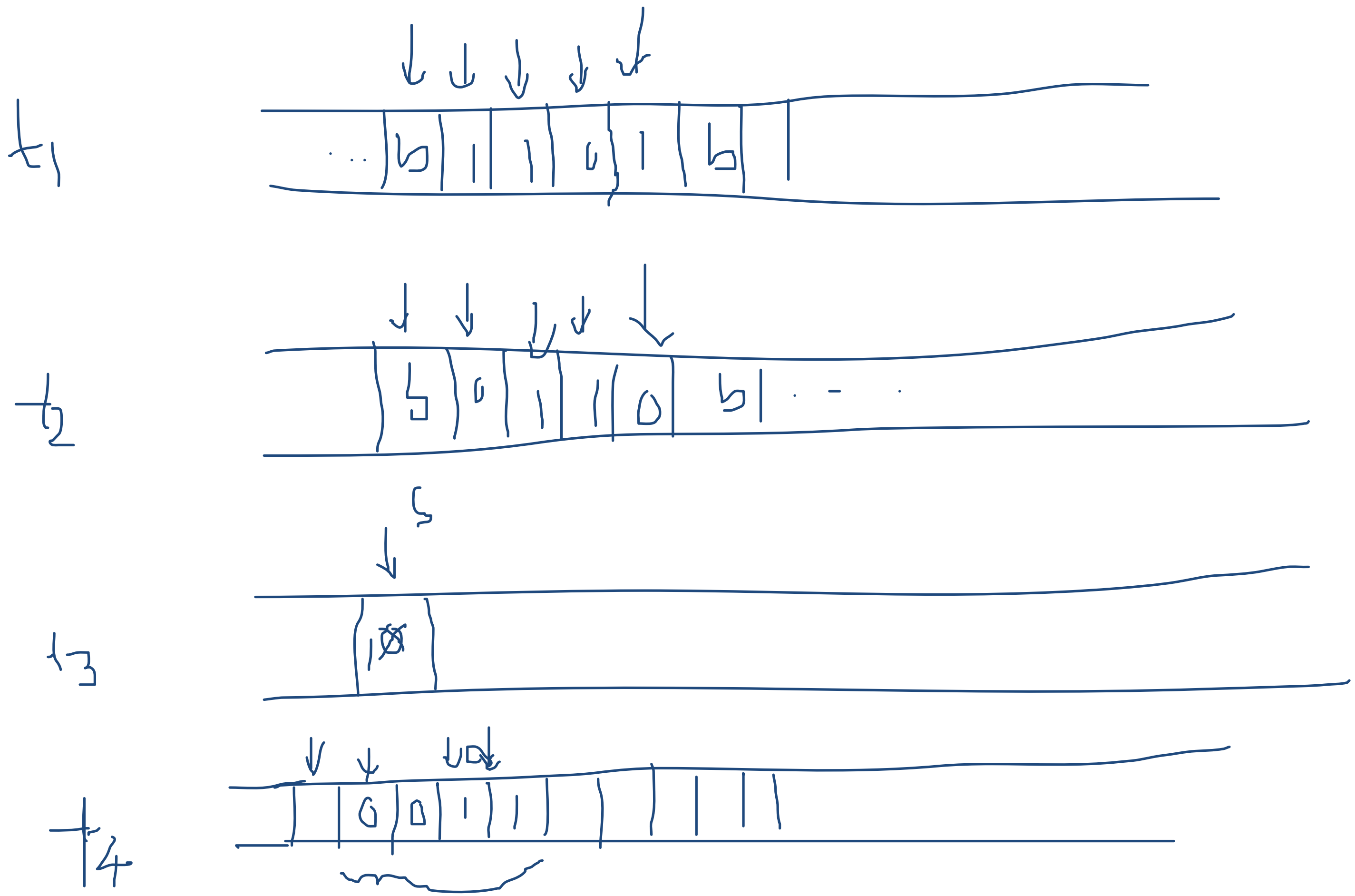
(ii) On each tape, a new symbol is written in the cell under the head.

(iii) Each tape head moves to the left or right or remains stationary. The heads move independently: some move to the left, some to the right and the remaining heads do not move.

Ex. : Addition of two Binary Numbers(discard final carry, if any)

```
   1101  — A
+  0110  — B
   1          ← C
   ----
   0011
```

$\{(1,0,0,b) = (1,0,0,1)/(L,L,S,L)\}$



Note:

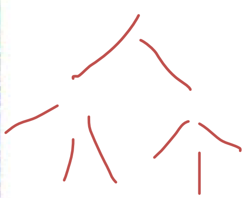
Every language accepted by a multitape TM is acceptable by some single-tape TM (that is, the standard TM).   M, M,   L(M) = L(M')

Definitions:

Running time: Let M be a TM and w an input string. The running time of M on input w, is the number of steps that M takes before halting. If M does not halt on an input string w, then the running time of M on w is infinite. ∞

Time complexity: The time complexity of TM M, is the function T(n), n being the input size, where T(n) is defined as the maximum of the running time of M over all inputs w of size n.

T(n)



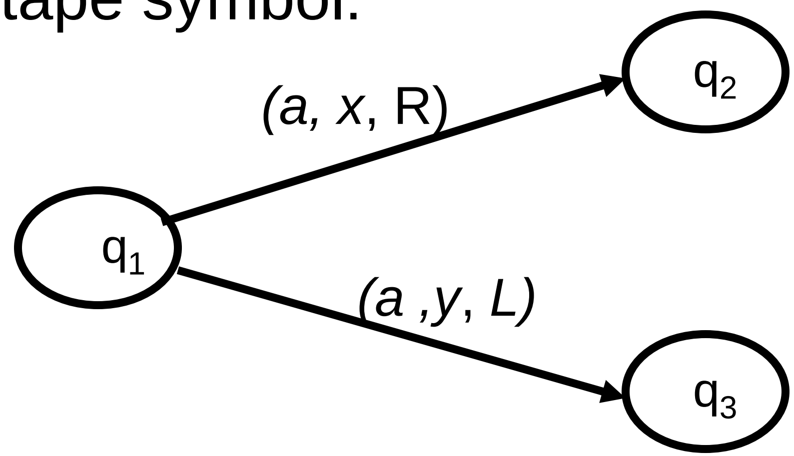
NDFSM
DFSM

ii) Non-Deterministic Turing machines(NTM)

A *Nondeterministic TM* is allowed to have more than 1 transition for a given tape symbol:

$\Sigma = \{0,1\}$ tape symbol

$\Gamma = \{x, y, 0, 1, b\}$



A string is accepted, if one of the branches of computation takes us to the accept state.

DTM

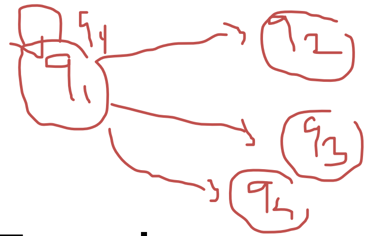
Note: Every Nondeterministic TM has an equivalent Deterministic TM (i.e. Standard TM).

NTM → DTM

Definition:

$Q = \{q_1, q_2, q_3\}$

$2^Q = \{\{\}, \{q_1\}, \{q_2\}, \{q_3\}, \{\ \}, \{\ \}, \{\ \}$



$\{q_1, q_2, q_3, q_4\}$

A Nondeterministic Turing machine is a 7-tuple *(Q, Σ , Γ , δ, q_o, b, F)*,

Where:

1. Q is a finite nonempty set of states
2. Γ is a finite nonempty set of tape symbols
3. $b \in \Gamma$ is called the blank symbol
4. Σ is a nonempty subset of Γ, called the set of input symbols. We assume that $b \notin \Sigma$.
5. q_0 is the initial state
6. $F \subseteq Q$ is the set of final states
7. δ is a partial function from $(Q \times \Gamma)$ into the power set of $Q \times \Gamma \times \{L, R\}$.

$Q \times \Gamma \times \{L, R\}$

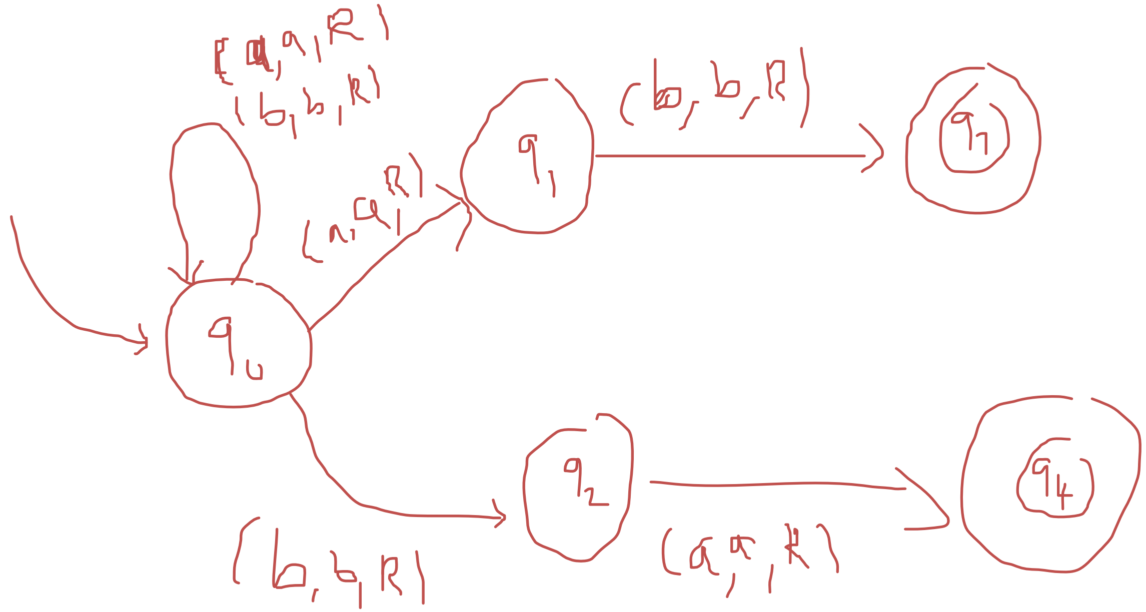
$2^S = \{\{\ \}, \{\ \}, \{\ \}, \{\ \}, \{\ \},$

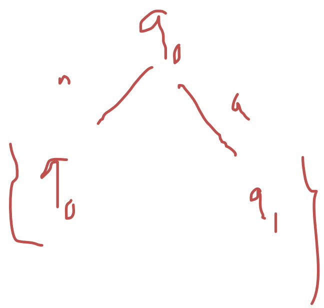
2^Q

Note: If $q \in Q$ and $x \in \Gamma$ and $\delta(q, x) = \{(q_1\ y_1, D_1), \{(q_2, y_2, D_2), \ldots, (q_n\ y_n, D_n)\}$

then the NTM can chose any one of the actions defined by (q_i, y_i, D_i) for $i = 1. 2..... n$.

Example: NTM to accept all the strings of a's and b's ending with ab or ba





NTM

THE LINEAR BOUNDED AUTOMATON(LBA)

(A restricted form of Turing Machine)

A Linear Bounded Automaton(LBA) is a Non-Deterministic Turing machine which has a single tape whose length is not infinite but bounded by a linear function of the length of the input string.

This model is important because: (1) the set of context-sensitive languages is accepted by the model. and (2) the infinite storage is restricted in size.

It is called the linear bounded automaton (LBA) because a linear function is used to restrict the length of the tape.

LBAs are not as powerful as TM…

Definition: The LBA can be described formally by the following set format:

$M = (Q, \Sigma, \Gamma, \boldsymbol{\delta}, q_o, b, ¢, \$, F)$ *where*

$\Sigma = \{0, 1, ¢, \$\}$

Q is finite nonempty set of states

Γ is a finite set of tape symbols

$b \in \Gamma$ is called the blank symbol

Σ is a nonempty set of input symbol

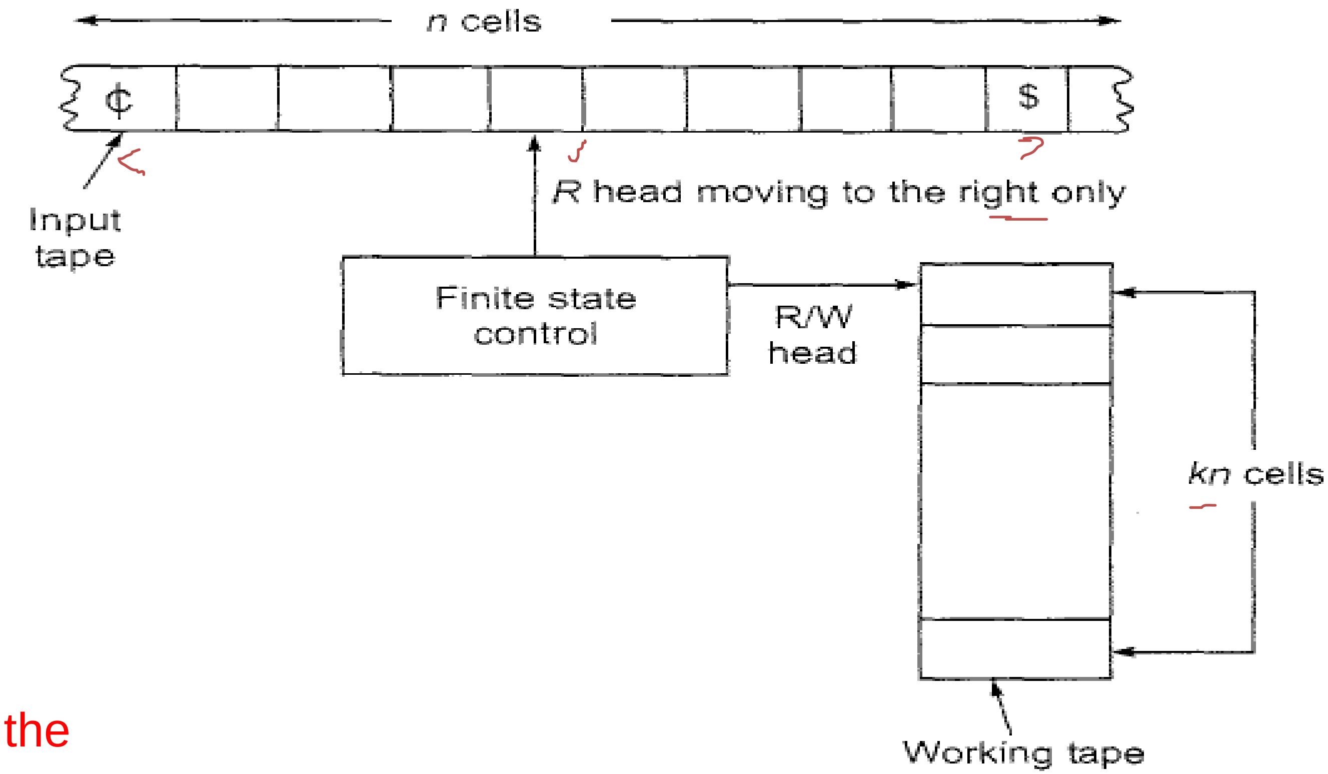
q_o is the initial state.

$F \subseteq Q$ is the set of final states

δ is a transition function.

¢ , $\$ \in \Sigma$ are input left-end and right-end marker on tape respectively and are special symbols.

Model of Linear Bounded Automata(LBA): Block diagram



There are two tapes: one is called the input tape, and the other, working tape.
On the input tape the head never writes and never moves to the left.

On the working tape the head can modify the contents in any way, without any restriction.
Where k is a constant specified in the description of LBA.

Whenever we process any string in LBA, we shall assume that the input string is enclosed within the end markers ¢ and $.

$ is called the right-end marker which is entered in the rightmost cell of the input tape and prevents the R head from getting off the right end of the tape.

¢ is called the left-end marker which is entered in the leftmost cell of the input tape and prevents the R head from getting off the left end of the tape.

The language accepted by LBA M is defined as:

¢10101$

ID

K

CSL

$$L(M) = \{ w \in (\Sigma - \{¢, \$\})^* \mid (q_0, ¢w\$, 1) \overset{*}{\vdash} (q, \alpha, i) \}$$

for some $q \in F$ and for some integer i between 1 and n.

In the case of LBA, an ID is denoted by (q, w, k), where $q \in Q$, $w \in \Gamma$ and k is some integer between 1 and n. The transition of IDs is similar except that k changes to $k - 1$ if the R/W head moves to the left and to $k + 1$ if the head moves to the right.

The Class of Languages accepted by LBA is called Context-Sensitive Language.

THANQ...

M-4

I=4

Presented(online) by : Dr. S G Gollagi

BE, MTech,PhD., LMISTE,MCSI, MIEEE

(for University of Toronto, Canada)

Topic: Decidability and Complexity

Decidability:

1. The definition of an algorithm, Decidability.
2. Decidable languages & Undecidable languages.
3. Halting problem of TM.
4. Post Correspondence Problem(PCP).

Complexity:

1. Growth rate of functions, The classes of P and NP problems.
2. Quantum Computation:Quantum computers.
3. Church -Turing thesis.

Textbook - 2: 10.1 to 10.7, 12.1, 12.2,12.8.1, 12.8.2

1. The definition of an algorithm, Decidability

-An Algorithm is a finite, well defined procedural steps to solve a given task. The algorithm is terminated after finite number of steps for any input.

a = 3 3+1+1+1+1 = 7
b = 4 c = a+b

1. —
2
3
Stop

⇒ TM

Algo, Procedure, Protocols
OS
Communication
Computing

Ex.: *Algorithm to add two numbers.*

Euclidean algorithm for computing GCD of two natural numbers.

The formal definition of algorithm emerged after the works of Alan Turing and Alanzo Church in 1936.

The Church-Turing thesis states that any algorithmic procedure that can be carried out by a human or a computer, can also be carried out by a Turing machine.

<algo> ⇄ TM

Thus the Turing machine arose as an ideal theoretical model for an algorithm.

Decidability:

As an algorithm terminates eventually, the TM also terminates. The TM halts in following two situations:

1. When a TM reaches a final state, it halts (accepting string)
2. When a TM is in some state q & next input symbol is 'a' and if the transition **δ(q, a)** is not specified, it halts(rejecting string) .

But, there are some TM that never halt on some inputs in any of the above situations.

So, we have to make a distinction between the language that are recognized by TM & Halts on all input strings and a TM that never halts on some input strings.

Recursively enumerable(RE) Language: A Language L ⊆Σ* is RE iff there exist a TM such that L = T(M), where T(M) is the language accepted by Turing Machine.

RE V/S RL

Decidable undecidable → Decidable

Recursive Language: A language L ⊆Σ* is recursive if f there exist a Turing Machine M that satisfy the following two conditions:

(i) if w ∈ L then w is accepted by M on reaching the accepting state & M halts.

(ii) if w ∉ L then M eventually halts, without reaching an accepting state.

Note: The Conditions (i) and (ii) assure us that the TM always halts, accepting w under condition (i) and rejecting under condition (ii).

HTM

Decidable Language

A problem/Language with two answers (Yes/No) is decidable if the corresponding language is recursive. In this case, the language L is also called decidable.

Undecidable Language

$w \in L$ $w \notin L$

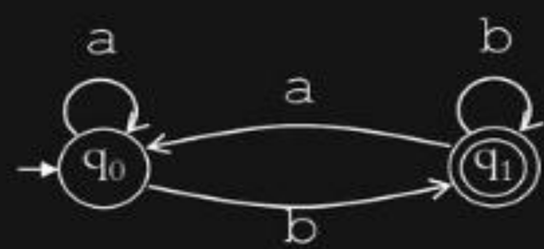
valid invalid

A problem / Language is undecidable if it is not decidable.

Note: A decidable problem is called a <u>solvable problem</u> and an undecidable problem an <u>unsolvable problem</u>.

2. Decidable and undecidable Language

In this section, we consider the decidability of regular and context-free languages.

Does 

accept input abb?

- We can formulate this question as a language:

$$A_{\text{DFA}} = \{\langle B, w\rangle : B \text{ is a DFA that accepts input } w\}$$

Is A_{DFA} decidable?

((q0,q1)(a,b)((q0,a,q0)(q0,b,q1)(q1,a,q0)(q1,b,q1)(q0)(q1))(abb)

$B = (Q, \Sigma, \delta, q_0, F)$ $\qquad w$

Theorem: if A_{DFA} = { < B, w > | B is DFA that accept w }, then A_{DFA} is decidable.

Proof: Let B = (Q, Σ, δ, q_0,F) be a DFA. We have to construct a TM M that always halts & accepts L(B). We know that DFA always ends in some state after reading the string w.

Now, we can construct a TM *M* that simulate DFA as follows:

1) Let B be a DFA & w an input string. *<B, w>* is an input for the Turing machine M.

2) Simulate B and input w in the TM M. Here, TM M checks whether input <B,w> is valid input. If <B,w> is invalid then TM M rejects and halts. If <B,w> is valid input, M writes the initial state q_0 and leftmost input symbol of w. It updates the state using δ & reads the next symbol in w.

3) If simulation ends in an accepting state of M, then TM accept <B,w>. Otherwise, M rejects <B,w>.

It is evident that M accepts <B, w> iff w is accepted by the DFA B.

Hence, A_{DFA} is decidable Language.

Is Language A_{DFA} decidable?

Does there exist a TM that accepts *all members of A_{DFA} and rejects all other inputs?*
(i.e. does it always halt)

Algorithm: Input <B,w> where B is a DFA & w an input string

1. Start
2. TM M , simulate B on string w
3. If simulated B ends in accept state
   then accept <B,w> and halt.
4. If simulated B ends in non-accepting state
   then reject <B,w> and halt.
5. Stop

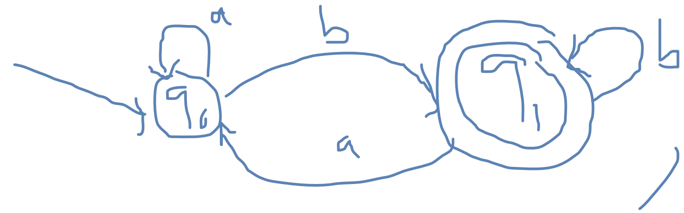
Since , there exist an algorithm to answer the problem.

Hence , A_{DFA} is decidable.

is A_{NFA} is decidable?

NDFSM

B



w = abb

Perform simulation:

state | input symbol

((q0,q1)(a,b)((q0,a,q0)(q0,b,q1)(q1,a,q0)(q1,b,q1)(q0)(q1))(abb)

$\longrightarrow q_{acc}$

B#w <B, w>

w = abba

$q_0 \xrightarrow{a} q_0 \xrightarrow{b} q_1 \xrightarrow{b} q_1 \xrightarrow{a} q_0$ (reject)

A_{DFA} is decidable

Definition: $A_{CFG} = \{ < G, w > \mid$ the Context-Free Grammar G accepts the input string w $\}$

Prove that CFL is decidable Language

Or

Is A_{CFG} decidable Language?

Every context-free language is decidable.

Proof. **We convert a CFG into CNF. Then any derivation of w of length n requires 2n-1 steps, if the grammar is in CNF. So, for checking whether the input string w of length n is in L(G), it is enough to check derivation in 2n-1 steps. We know that there are only finitely many derivations in 2n-1 steps.**
Now, we design a TM that halts as follows:

1) Let G be a CFG in CNF & <G, w> is an input string for TM M.
2) if n = 0, list all the single-step derivations.
3) if n $\neq$ 0 list all the derivations with 2n-1 steps.
4) if any of the derivations in step 2 or 3 generates the string w, then M accepts <G,w > & halts.
else
M rejects <G, w> & halts.

<G, w> is represented by representing the four components (V,T,P,S) of G and input string w. The next step of the derivation is got by the production to be applied.

Example: Consider the CFG for L = { $a^n b^n$ | n >=0 }

8-1 = 7 steps

$S \rightarrow aSb \mid \varepsilon$ and w = 'aabb'

$n = |w| = |aabb| = 4$

$A \rightarrow BC$
or
$A \rightarrow a$

⇓ ε-production

$S \rightarrow aSb \mid ab$

Acc_CFG = < G, w >

⇓ CNF

$S \rightarrow ASB \mid AB$
$A \rightarrow a$
$B \rightarrow b$

⇒

$S \rightarrow CB \mid AB$
$C \rightarrow AS$
$A \rightarrow a$
$B \rightarrow b$

$S \overset{1}{\Rightarrow} CB \overset{2}{\Rightarrow} ASB$
$\overset{3}{\Rightarrow} aSB$
$\overset{4}{\Rightarrow} aABB$
$\overset{5}{\Rightarrow} aabB$
$\overset{6}{\Rightarrow} aabB$
$\overset{7}{\Rightarrow} aabb \in L$

✓

{ (S, C, A, B)(a, b)(S → CB, S → AB, C → AS, A → a, B → b)(S)(aabb) }

Encoding

T R/W

Undecidable Problems…

1. The Post Correspondence Problem(PCP)
2. Halting Problem of Turing Machine

1.THE POST CORRESPONDENCE PROBLEM

The Post Correspondence Problem (PCP) was first introduced by Emil Post in 1946. Later, the problem was found to have many applications in the theory of formal languages.

The problem over an alphabet **Σ** belongs to a class of yes/no problems and is stated as follows:

$|x_1 x_2 \ldots x_n| = |y_1 y_2 \ldots y_n|$

1) Df:

Consider the two lists $x = (x_1, x_2 \ldots x_n)$, $y = (y_1, y_2, \ldots y_n)$ of nonempty strings over an alphabet **Σ.** The PCP is to determine whether or not there exist $i_1, i_2 \ldots i_m$ where $1 \leq i_j \leq n$, such that $x_{i1} \ldots . x_{im} = y_{i1} \ldots . y_{im}$

Note: if there exists a solution to PCP, there may exist infinitely many solutions.

https://www.youtube.com/watch?v=VZNN1OGoqr8

The Post Correspondence Problem

The Post correspondence problem is an undecidable decision problem that was introduced by Emil Post in 1946

Emil Leon Post

The Post Correspondence Problem

Dominos:

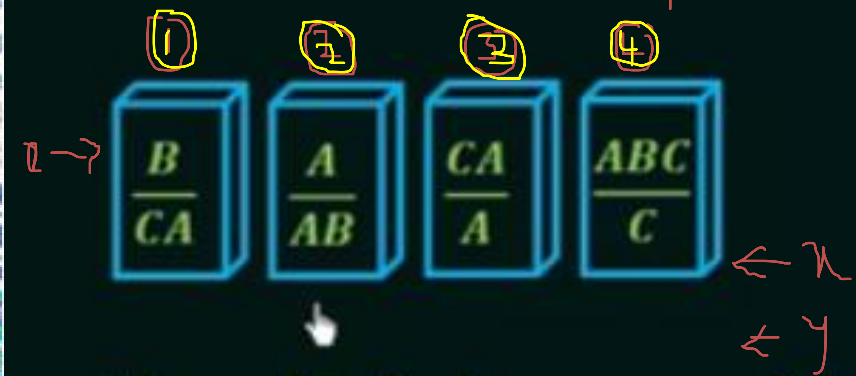
$\Sigma = \{A, B\}$

$x = \{B, A, CA, ABC\}$

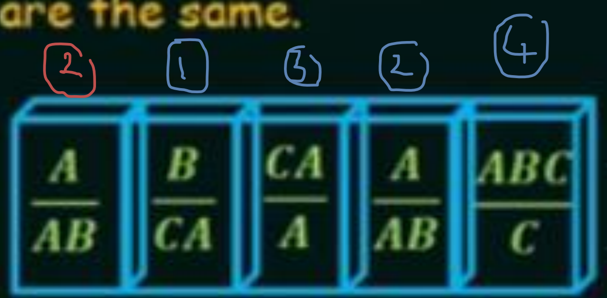
$y = \{CA, AB, A, C\}$

$|x| = 7$

$|y| = 6$



We need to find a sequence of dominos such that the top and bottom strings are the same.



→ A B C A A A B C

→ A B C A A A B C

Soln-I

A	B	C	A	A	A	B	C
A	B	C	A	A	A	B	C

Practice Problems:

x = (b, babbb, ba) y = (bbb, ba, a)

1. Does the PCP with two lists *x = (b, bab*^{3}*, ba) and y = (b*^{3}*, ba, a) have a* solution?

2. Find at least two solutions to PCP defined by the dominoes:

1	2	3
1	10	10111
111	0	10

2. HALTING PROBLEM OF TURING MACHINE

Given a TM M and an input string w with the initial configuration q_0w, after some(or all) computations, does the machine M halts on w ?

Alan Turing ,in 1990 proved that the halting problem of TM on input w is undecidale.

A reduction Techniques may be used to prove the undecidabilty of halting problem of a TM. Using this techniques , a problem A is reducible to problem B if a solution to the problem B can be used to solve the problem A.

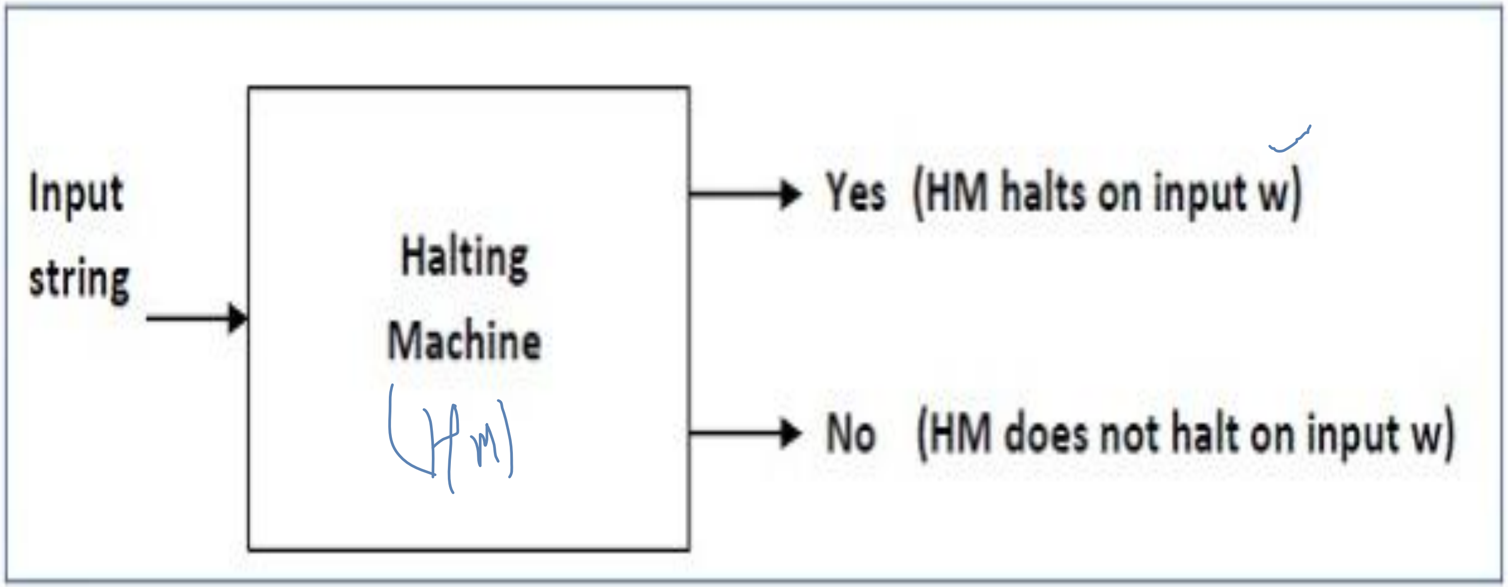
$A \Rightarrow B$

Thus,

-if A is reducible to B and B is decidable , then A is decidable.
-if A is reducible to B & B is undecidable, then A is undecidable.

Block diagram of a Halting machine:

$\langle M, w \rangle$



Key point: Turing machine can be encoded as string, and other Turing machines can read those strings to perform “simulations”

Theorem: The Language $HALT_{TM}$ = { <M ,w> | the TM M halts on input w } is undecidable.

Proof: We assume that $HALT_{TM}$ is decidable and get a contradiction. Let M_1, be the TM such that $T(M_1) = HALT_{TM}$ & let M_1 halts eventually on all <M,w>. We construct a TM M_2 as follows:

1) For M_2, <M,w> is an input.
2) The TM M_1 acts on <M, w>
3) if M_1 rejects <M,w> then M_2 rejects <M,w>
4) If M_1 accepts <M,w>, simulate the TM M on the input string w until M halts.
5) If M has accepted w, M_2 accepts <M,w>; otherwise M_2 rejects <M,w>.

When M_1 accepts <M,w> (in step 4), the TM M halts on w.

In this case either accepting state q or a state q' such that $\delta(q',a)$ undefined on 'a' is reached.

In the first case (the first alternative of step 5) M_2 accepts <M,w>.

In the second case (the second alternative of step 5) M_2 rejects <M,w>.

It follows from the definition of M_2 that M_2 halts eventually.

But, $T(M_2)$ = { <M,w> | The TM accepts w} is undecidable which is a contradiction.

Therefore, the Language $HALT_{TM}$ is undecidable.

Important Questions on: Module-5

1. Define an algorithm and Explain with example.
2. Write short notes on: i) Recursively enumerable Language ii) Decidable Language.
3. if $A_{DFA} = \{ < B, w > \mid B \text{ is DFA that accept } w \}$, then show that A_{DFA} is decidable.
4. What is Post Correspondence problem? Explain with example.
5. What is Halting problem of Turing Machine?
6. Show that $HALT_{TM} = \{ <M ,w> \mid \text{the TM } M \text{ halts on input } w \}$ is undecidable.
7. Define the following: i) Quantum Computers ii) Class P and NP problems
8. Explain Church -Turing Thesis.

Topic: Complexity

When a problem/language is decidable, it simply means that the problem is computationally solvable in principle.

It may not be solvable in practice in the sense that it may require enormous amount of computation time and memory.

P-stands for polynomial time: this class of problems that can be solved by a deterministic algorithm in a polynomial time.

DTM

NP-stands for Non-deterministic problem: this class of problems that can be solved by a nondeterministic algorithm in a polynomial time.

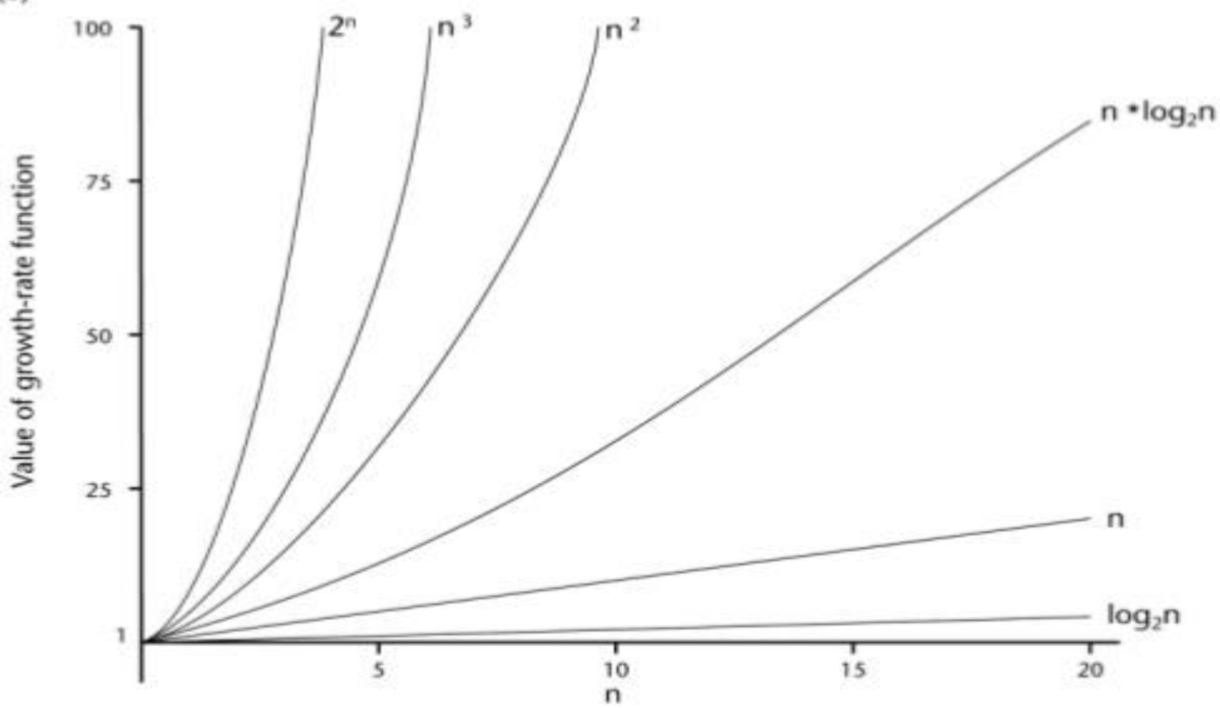
NTM

1. GROWTH RATE OF FUNCTIONS

O, Ω, Θ, o

A Comparison of Growth-Rate Functions (cont.)

(b)



P

A_1 A_2 A_3

$O(2^n)$ $O(n^2)$ $O(n)$

when we have two algorithms for the same problem, we may require a comparison between the running time of these two algorithms.

O(n)

Definition: Let f, g : N → R^+ (R^+ being the set of all positive real numbers). We say that $f(n) = O(g(n))$ if there exist positive integers C and N_0 such that $f(n) \leq Cg(n)$ for all $n \geq N_0$.
In this case we say f is of the order of g (or f is 'big oh' of g)

O(1)

Ex.: Let $f(n) = 4n^3 + 5n^2 + 7n + 3$. Prove that $f(n) = O(n^3)$

In order to prove that $f(n) = O(n^3)$, Take C= 5 & N_0 = 10.
Then $f(n) = 4n^3 + 5n^2 + 7n + 3 \leq 5n^3$ for $n \geq 10$.

1
2
3
4
5
6
7

Then, $f(n) = O(n^3)$

$g(n) = n^3$

$n^3\left(4 + \frac{5}{n} + \frac{7}{n^2} + \frac{3}{n^3}\right)$

$f(n) = n^3\left(4 + \frac{5}{n} + \frac{7}{n^2} + \frac{3}{n^3}\right) \leq 5n^3$

$\leq 5n^3$

$f(n) \leq cn^3 \leq cg(n)$

$f(n) \leq cg(n) \leq 5n^3 \approx O(n^3)$ $n \geq N_0$

$f(n) = O(n^3)$

Theorm

Ex.: If $p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots .. a_1 n^1 + a_0$ with $a_k > 0$.

Then $p(n) = O(n^k)$

Soln:

It is given that:

$$p(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots a_1 n^1 + a_0$$

Each term in summation is of the form: $a_i n^i$,

Since, n is non-negative, a particular term will be negative only if $a_i < 0$.

So

$$|p(n)| = |a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n^1 + a_0|$$

$$= n^k \left(|a_k| + \frac{|a_{k-1}|}{n} + \ldots + \frac{|a_1|}{n^{k-1}} + \frac{|a_0|}{n^k} \right)$$

$$\le n^k (c)$$

$$\le c n^k, \quad \text{where } c =$$

ie $p(n) \le c\, g(n)$ for $n \ge N_0$, where $g(n) = n^k$

$\therefore$ $p(n) = O(g(n)) = O(n^k)$

Ex. Obtain a time complexity of a TM which accepts $L = \{ a^n b^n \mid n \geq 1 \}$

→ replace
→ moves

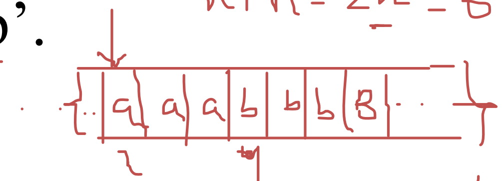
Solution:

Step1: Consist of going through the input string $a^n b^n$ forward and backward and replacing leftmost 'a' by x & the leftmost 'b' by y.

$w = aaabbb$

So, we require at most 2n moves to match 'a' with a 'b'.

$n + n = 2n = 6$



(i.e. Total number of moves = 2n)

$3 \times 6 = 18$

Step2: The *step1* is to be repeated for n times for each 'a'.

Hence the number of moves for accepting $a^n b^n$ is at most (2n)(n)

For strings not of the form $a^n b^n$, TM halts with less than $2n^2$ moves.

Hence , the time (running time) complexity is given by $O(n^2)$.

i.e. $T(M) = O(n^2)$

$C = 2, \; g(n) = n^2$

$T(M) \leq c\,g(n)$

$T(M) = O(n^2)$

2. The Class P & NP Languages

1 — x
0 — y
Move
T(n) ≈ No of Moves

In this section we introduce the classes P and NP of languages.

Definition: A Turing machine M is said to be of time complexity *T*(n) if the following holds: Given an input w of length n. M halts after making at most *T*(n) moves.

***Note: In this case. M eventually halts. Recall that the standard TM is called* a deterministic TM.**

Definition: A language L is in class P if there exists some polynomial *T*(n) such that L = L(M) for some Deterministic TM M of time complexity *T*(n).

DTM → L
NTM

D P-class

Obtain a time complexity of a TM which accepts L = { $1^n2^n3^n$ | n ≥ 1}

DTM

x y z

... | 1 | 1 | 1 | 2 | 2 | 2 | 3 | 3 | 3 | b | ...

Solution:

Step1: Consist of going through the input string $1^n2^n3^n$ forward and backward and replacing leftmost '1' by x , the leftmost '2' by y and leftmost '3' by z . So, we require at most 4n moves.

$12 \times 3 = 36$

$4n \times n = 4n^2 = O(n^2)$

(F: 2n + B: 2n)

(i.e. Total number of moves = 4n)

Step2: The *step1* is to be repeated for n times for each '1'.

Hence the number of moves for accepting $1^n2^n3^n$ is at most (4n)(n)

For strings not of the form $1^n2^n3^n$, TM halts with less than $4n^2$ moves.

Hence, the time (running time) complexity is given by $O(n^2)$.

i.e. $T(M) = O(n^2)$

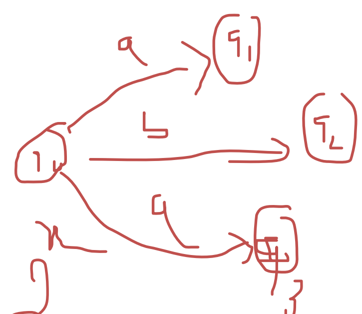
Non-Deterministic Polynomial (NP)

Definition: A language L is in class **NP** if there is a Non-Deterministic TM M and a polynomial time complexity $T(n)$ and M executes in at most $T(n)$ moves for every input w of length n.

NTM

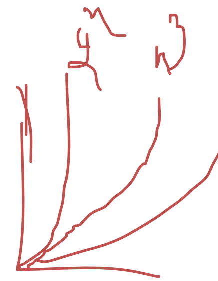
We have seen that a Deterministic TM M_1 simulating a Non-Deterministic TM M exists. If $T(n)$ is the complexity of M, then the time complexity of the equivalent Deterministic TM M_1 is: $2^{O(T(n))}$

NDTM



NTM = 2^n

DTM $\approx n^k$



NTM $\approx$ DTM

NDFSM = DFSM

Polynomial: $T(n) = n^k$

300 pages

3. Church-Turing Thesis (1930)

Models

Church-Turing's thesis is stated as : **Any " effective computation" or "any algorithmic" procedure that can be carried out by a human being or a team of human being or a computing machine can be carried out by Turing Machine.**

1980,

In other words , there is an effective procedure to solve a decision problem P if and only if there is a Turing Machine that answer YES on input w ∈ P and NO on input w ∉ P.

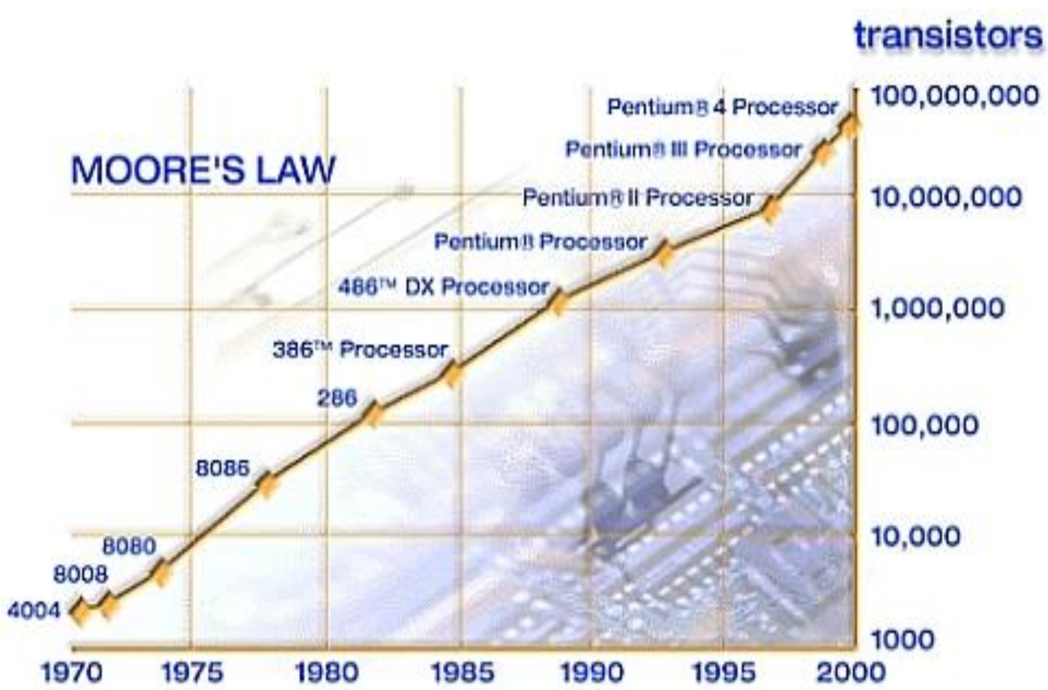
The Church-Turing thesis predicts that it is able to construct models of computations more powerful then the existing once.

Thesis also states that we cannot go beyond Turing Machines or their equivalent. Since there is no precise definition for "effective computation", or "Algorithmic procedure" Church's thesis is not a mathematically precise statement today.

Any algorithmic process can be simulated efficiently by a Turing machine.
But a challenge to the strong Church-Turing thesis arose from analog computation. Certain types of analog computers solved some problems efficiently whereas these problems had no efficient solution on a Turing machine. This led to the modification of the Church thesis.

→ S/W
→ H/W

Quantum Computers
Classical Computers



The compactness of chip has increased the power of the Computer. The growth of Computer power is described by Moore's law, which states that the Computer power will double for constant cost once in every 1.5 years.

H/W
S/W →

4. Quantum Computer

Referee text book or internet for theoretical note of Quantum Computer…