Computational Theory

Lecturer:- Samer Al-khazraji

Reference:- Introduction to Computational Theory by Daniel I.A. Cohen.

Background

- Transfer computer from calculating device to reasonable device.
- Computer theory considers the computer parts (logic circuits, operating systems, instruction sets, data structure, artificial intelligence, data structure, and so on).
- Computer theory studies performance enhancement of existing machine components.

Background

- Our study research a better use of the current and future computers and optimality will be not considered.
- Our study will investigate the accepted data structure and rejected.
- Mathematica logic is one of computer theory parts.

Background

- Mathematicians faced a number of barriers to define a number of math terms such as infinity.

- The **Universal-algorithm machine** is a theoretical concept was developed by “Alan Mathison Turing” that research what kind of work can be accepted by the machine.

Background

- The universal-algorithm machine led to the invention of the computer.
- Vacuum tube invention assisted researchers to build automatic electronic calculators which attracted engineers to build computer for:
  - Storing input data in the computer.
  - Storing the software in the computer.
  - Has a central processing unit to process the input data based on their type.

Background

- The Linguistic ability of computers was achieved after the development of Noam Chomsky theory that led to developing computer languages.

- Our study will focus on software that can be performed using computers and it is called **computational theory.**

Languages

- Each language in the linguistic field consists of three entities; letters, words, and sentences.
- Set of characters shape word, group of words collect sentences which form paragraph and etc.
- Not every set of letters can shape valid words and not every collection of words can make up a valid sentence.

Languages

- Similarly, in computer languages, a certain set of characters form a word (e.g. while, for, and so on). Certain collection of words shape commands (e.g. for (i > 0; i < 100; i++), and certain set of commands compose program.

- **Theory of Formal Languages** is an interesting set of string of symbols that obey a set computer language rules. The set of symbols does not focus on the meaning.

Languages

- **Null** Λ is an empty string (word) that do not have a letter.
- Two words that have the same order of letters are equals.
- The English language will be an example in our study. The Greek letter Σ will represent the whole alphabet.

Languages

- Formal language theory should include a basic unit called **alphabet** which consists of a finite number of symbols.
- Formal language theory concentrates on syntax not on semantics (i.e. focuses on word spelling, not on the word meaning).

Languages

- To investigate a word whether it is valid in a language, two types of rules can be setup: test the alphabet letters or construct all words from the language.
- **Concatenation** is an operation that can be applied to the formal language theory to write letters side by side.
- **Length** is a function which defines the number letters in the string.

Languages

- **Reverse** function spells word reversely, for instance, the reverse of the word a = (123) is (321).
- **PALINDROME** is a language that includes Λ and strings which are x and revers(x).
- **Closure (*) of the alphabet**, is a language that contains a set of finite length of strings (including Λ). Each string is shaped from concatenation of the alphabet elements. Closure (*) sometimes is called Kleene star.

Regular Expressions

- Regular Expression (RE) is a phrase which defines a language and the resulted language is called Regular Language.

Example:-

From the alphabet Σ = {a, b}, it is possible to define the following L_5.

L_5 = {a, ab, abb, abbb, abbb, . . .}

The RE which describes L_5 is:

L_5 = language (ab*)

Regular Expressions

- Kleene star (*) represents to the infinite language, however, (+) define a finite language.
- Plus sign (+) has been employed to give a choice (i.e. either, or) in determining a language using RE.
- The RE for a language which consists of strings that start with a and end with b and a combination of a and b in the middle is:

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

Regular Expressions

Language associated with any RE should obey the following rules:

1. It may contain only one letter such as (Λ)
2. r_1 is RE_1 and r_2 is RE_2, r_1 is associated with L_1 and r_2 associated with L_2 then:
   - $r_3 = r_1 r_2$, the language defined by r_3 is associated with L_1 many times L_2.

     $$\text{Language } (r_1 r_2) = L_1 L_2$$
   - $r_3 = r_1 + r2$, the language defined by r_3 is associated with a language L_1 union L_2.

     $$\text{Language } (r_1 + r_2) = L_1 + L_2$$
   - The language associated with $(r_1)^*$ is $L_1{}^*$, then L_1 is a set of all words.

Finite Automata

- Finite automaton FA is a combination of three items:
  - ❖A group of states, one is designed as an initial state and one or more as the final state.
  - ❖A collection of input letters to be read by the machine one after another.
  - ❖A set of transitions which guide the transition from one state to another depending on the input letter.

Finite Automata

- The start state can be recognised by the signs *minus* (-) or *arrow* or *start word* as a start state and the signs *plus* (+) or *inside* or *outside circle* and *final as a final word state respectively.*
- The letters of the input string will form path in FA machine starting from the begin state and traverse machine states reaching a particular state.

Finite Automata

- If the state is a final state, the path has ended successfully and the input string is accepted.
- Otherwise, the input string will be rejected.
- The travelling across FA will be ended when the input string is out of letters.

Finite Automata

- Some aspects of directed graph for example arrow and circle are adopted to draw FA:
  - directed edge or edge is employed to represent a path from one state to another.
  - Some states have one or more outgoing edges and some states have no incoming edges (e.g. start state).

Finite Automata

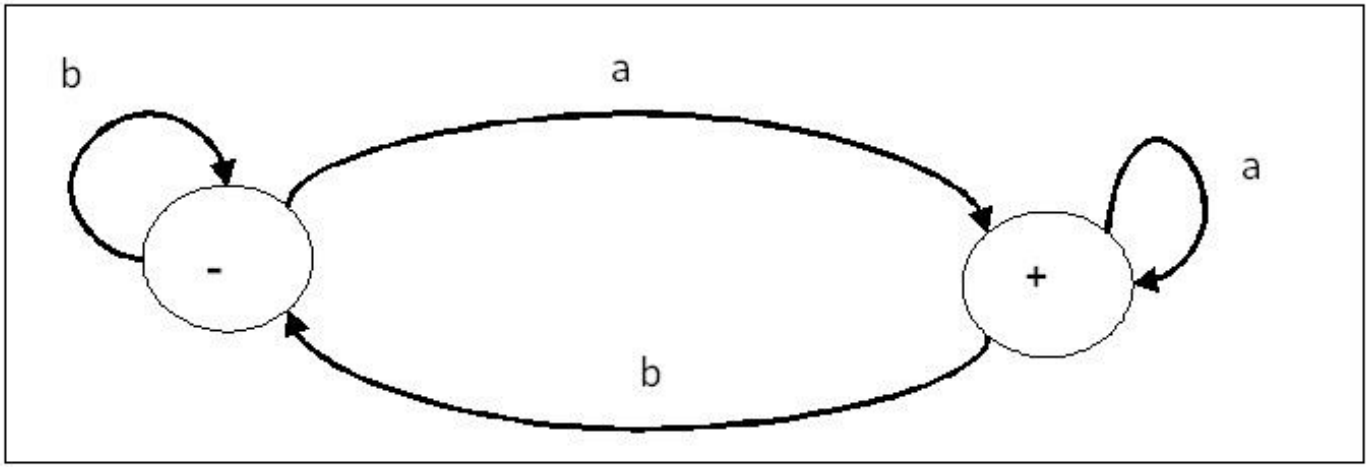
- FA read letters of input string one each time beginning from the leftmost letter.
- The process starts from the start state and ends with reading of the last letter.
- The sequence of letters determines the sequence of states.
- FA sometimes call finite accepter because its role only accepts the input string or reject it.

Finite Automata

Example:

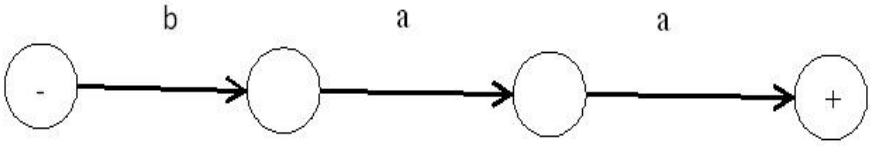
The following FA accepts the language which is generated by the following RE:

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

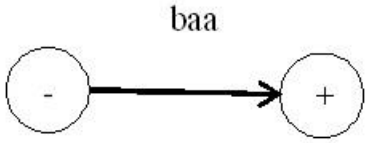


Transition Graph TG

- FA consumes one input letter at a time to travel from one state to another as shown in the following figure.



- The need for a machine which accepts strings by few numbers of edges is important as demonstrated in the following graph.



Transition Graph TG

- Transition graph TG is a collection of three items:
  - A collection of sets; at least one start state and one or more (may be none) final state.
  - A set of input letters (alphabet) which form strings.
  - A set of edges (travels) which examine the input string.

Kleene's Theorem

- Kleene confirmed in 1956 a theorem which states that; *A language which is determined by RE or FA or TG can be defined by the three methods*.

The prof of the theorem consists of three parts;

- Every language which is accepted by FA can be defined by TG.
- Every language which is accepted by TG can be defined by RE.
- Every language which is accepted by RE can be also accepted by FA.

Kleene's Theorem

- **The Proof of part 1**:

Every FA is TG, so that, any language which is defined by FA is defined by TG.

- **The Proof of Part 2**:

The proof consists of steps which start with TG and end with RE that defines the same language.

---------- Continued----------

Kleene's Theorem

- **First**: simplifying the TG, this can be done by creating a new start state which is labelled by '-' and connect it with other start states by edges which are labelled by the word Λ. Remove '-' signs from the old start states.

- **Second**: apply the same steps in first to unify the final states in one final state.

---------- Continued ----------

Kleene's Theorem

- **Third :** one by one, in any order, bypass and eliminate all not – and + states in the TG. A state is bypassed by connecting each incoming edge with each outgoing edge, the label of each resultant edge is the concatenation of the label on the incoming edge with the label on the loop edge if there is one and the label on the outgoing edge.

- ---------- Continued ----------

Kleene's Theorem

- **Fourth**: When two states are joined by more than one edge going in the same direction, unify (+) them by adding their labels.

- **finally**, when all that is left is one edge from – to +, the label on that edge is a RE that generates the same language as was recognized by the original machine.

Kleene's Theorem

- **<u>The Proof of Part 3</u>**:

- **Rule1:** A machine accepts a specific letter of an alphabet, there is a machine accepts Λ string.

- **Rule2**: FA_1 accepts a language which is defined by RE_1 (r_1). FA_2 accepts a language which is determined by RE_2 (r_2). FA_3 ($FA_1 + FA_2$) accepts the language which is defined by ($r_1 + r_2$).

Kleene's Theorem

- **Rule3**: FA_1 accepts a language which is defined by RE_1 (r_1). FA_2 accepts a language which is determined by RE_2 (r_2). FA_3 (FA_1 + FA_2) accepts the language which is defined by (r_1 r_2).

- **Rule 4:** If "r" is a regular expression and FA_1 is a finite automaton that accepts exactly the language defined by "r", then there is an FA called FA_2 that will accept exactly the language defined by " r^* ".

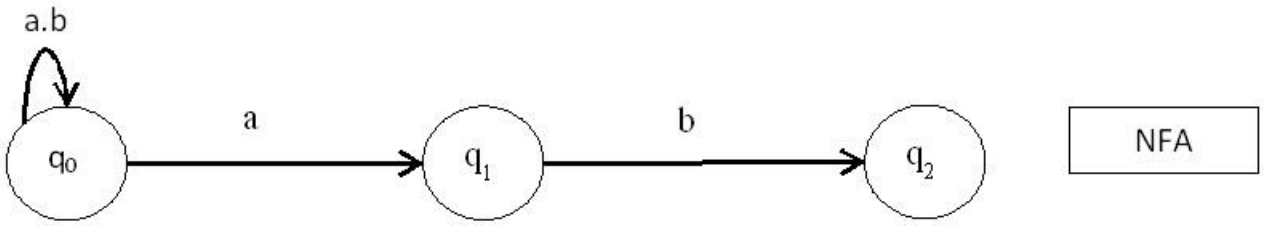
Nondeterministic Finite Automata NFA

NFA is a group of three items:

- A finite collection of states, one can act as a start state and set of final states.
- Input letters of alphabet Σ.
- A finite group of edges which explain the transitions from start state toward the final states. The transition can be done using more than one edge with the same label.

Nondeterministic Finite Automata NFA

Example of machine conversion from NFA to DFA.



Nondeterministic Finite Automata NFA

- States of the NFA are S= $\{q_0, q_1, q_2\}$ and the transition table is:

	a	b
q_0	q_0, q_1	q_0
q_1		q_2
q_2		

- First state of DFA = $\{q_0\}$

Nondeterministic Finite Automata NFA

- $q_{0,}\ q_1$ is the second state in DFA and q_0 is already exist in DFA. The transition table for the new state is:

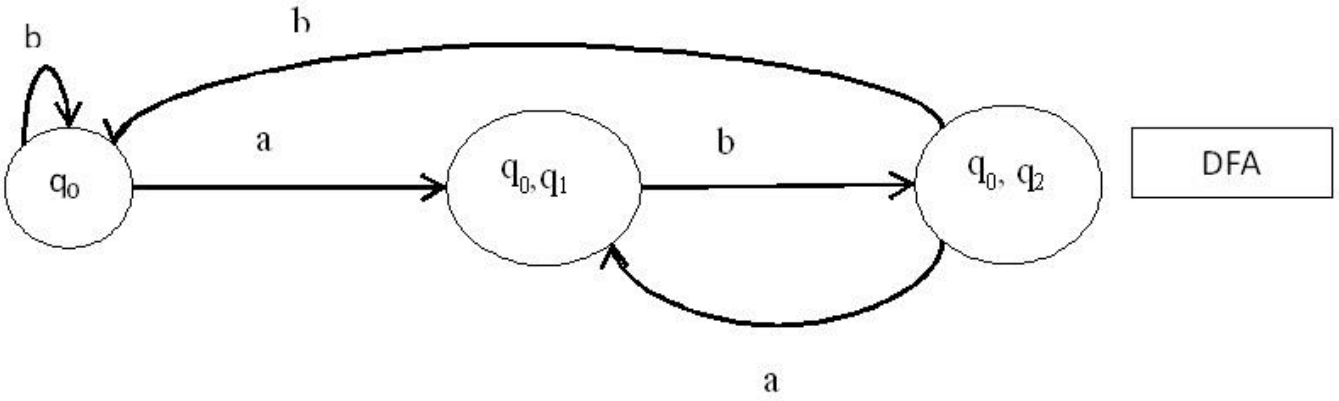
	a	b
$q_{0,}\ q_1$	$q_{0,}\ q_1$	$q_{0,}\ q_2$

- The next state of the DFA is: $q_{0,}\ q_2$ and the transition table of the new state is:

	a	b
$q_{0,}\ q_2$	$q_{0,}\ q_1$	q_0

Nondeterministic Finite Automata NFA

- There is no new state for the new machine which will be as follows:



Regular Languages

- It is a language that is defined by a regular expression and also by FA based on Kleene's theorem.

- **Theorem**:-

  If L1 and L2 are regular languages, then L1 + L2, L1L2, and L* are also regular language.

Regular Languages

- **Proof-1** (by regular expression) :

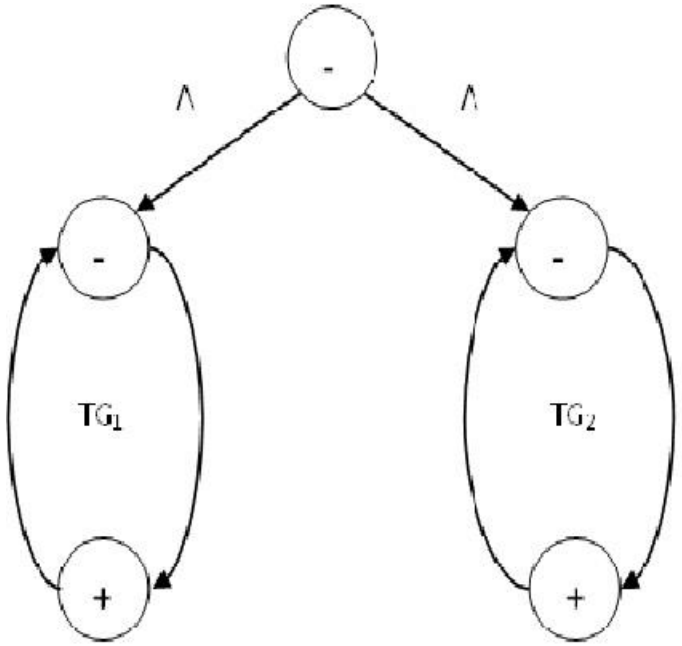
The regular expressions r_1 and r_2 generate the languages L_1 and L_2, then the language which is generated by L_1+L_2, L_1L_2 , and L_1^* is also regular language.

Regular Languages

- **Proof-2** (by machine):-

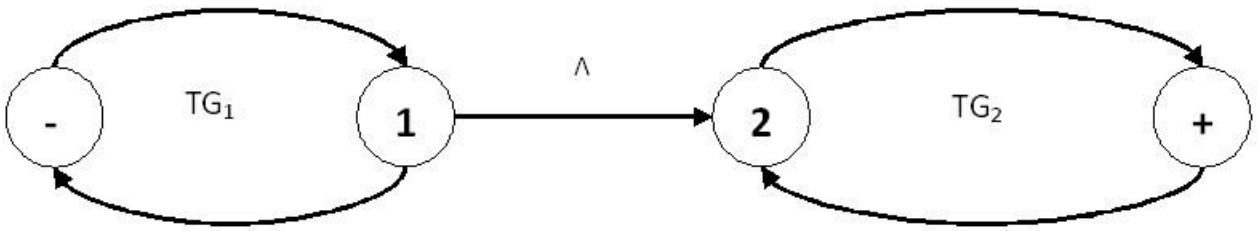
Let L_1 and L_2 are accepted by TG_1 and TG_2 respectively. Then,

$L_1 + L_2$:



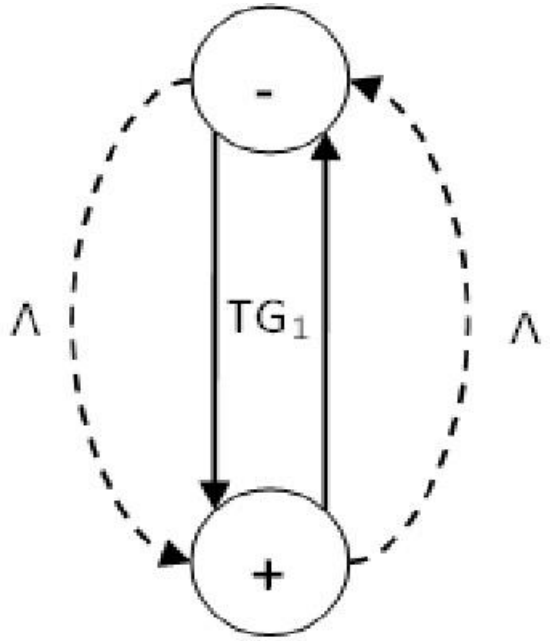
Regular Languages

- L_1L_2



Regular Languages

- L^*



Non-Regular Languages

- A language that cannot be defined by a regular expression is called a non-regular language.

Example:-

$L = \{\Lambda \;\; ab \;\; aabb \;\; aaabbb \;\; . \;\; . \;\; . \;\; \}$

$L = \{a^n b^n \text{ for } n = 0\ 1\ 2\ 3\ . \ . \ . \ )$

$RE = a^n b^n$

---------- Continued ----------

Non-Regular Languages

- For every RL there is some that FA accepts it, then we can construct FA machine for the language L, the machine will accept the string "$a^{98}b^{96}$ ", while this string is not in the language L, in other words, the machine accepts words not in L, therefore, the language L is not Regular.

Context-Free Grammar CFG

- A group of rules which is called grammar is used to validate sentence in the linguistic field.
- The validation process comprises inspection of the syntax (structure) of the input sentence, not in the semantics (meaning).
- A set of grammatical rules is dubbed *productions.*

Context-Free Grammar CFG

- The structure of the rules known as *Context-Free Grammar* CFG.
- The process of generating of final string of leaves starting from the beginning of a sequence of rules is known as *derivation*.
- The language which is generated by CFG is called *Context-free Language*.

Context-Free Grammar CFG

- Context-Free Grammar is a collection of three items:
  - An alphabet of letters called *terminals* and r.
  - A group of symbols and known as non-t*erminals* one of them act as a start symbol.
  - A sequence of *productions* of the form of:

Non-terminal → string of terminals and/or non-terminals

Context-Free Grammar CFG

Example:

Let us consider the following CFG:

$$S \rightarrow aS$$

$$S \rightarrow \Lambda$$

If the production-1 is applied 6 times then production-2 only once, then the derivation procedure will take the following sequence:

---------- Continued ----------

Context-Free Grammar CFG

S => aS
=> aaS
=> aaaS
=> aaaaS
=> aaaaaS
=> aaaaaaS
=> aaaaaaΛ
= aaaaaa

The RE of this CFG is a^*.

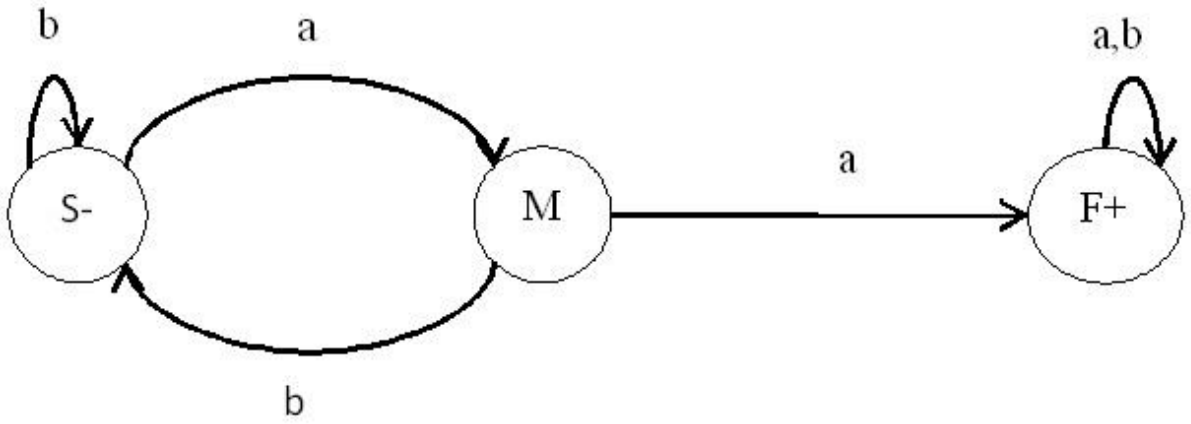
Regular Grammars

- Some languages are generated by CFG and defined by RE and is called *Regular Languages.*
- However, some languages such as $a^n b^n$ are not regular languages but can be generated by CFG.
- To investigate that the CFG is a regular grammar, it needs to proof that there is a compatible FA accepts the strings generated by the CFG.

Regular Grammars

Example:

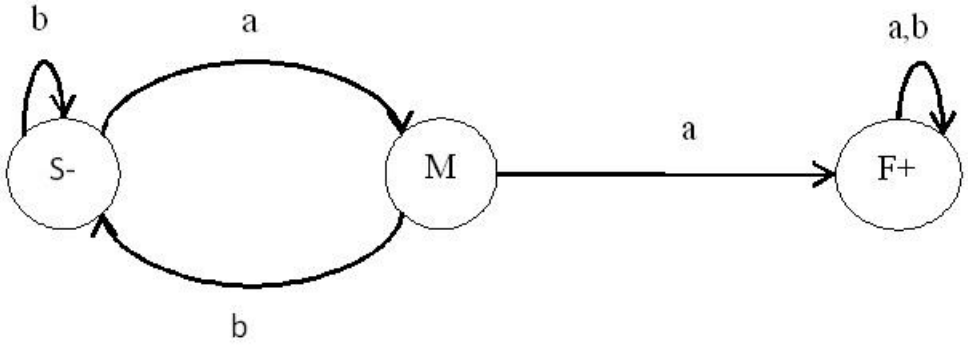
The following FA accepts strings that have a double ‘a’.



Regular Grammar

Example:

let us walk through the below machine to confirm that the string *abbaab* which is defined by the following FA, is accepted by a CFG. The procedure starts from the start state to the final state based on the edge and state labels.



---------- Continued ----------

Regular Grammar

S (We begin in S)

aM (We take an a-edge to M)

abS (We take an a-edge then a b-edge and we are in S)

abbS (An a-edge, a b-edge, and a b-loop back to S)

abbaM (Another a-edge and we are in M)

abbaaF (Another a-edge and we are in F)

abbaabF (A b-loop back to F)

abbaab (The finished path: an a-edge a b-edge . . .)

Regular Grammar

- The path development of the string *abbaab* is similar to the derivation of the string in a CFG:

$$S \rightarrow aM$$
$$S \rightarrow bS$$
$$M \rightarrow aF$$
$$M \rightarrow bS$$
$$F \rightarrow aF$$
$$F \rightarrow bF$$
$$F \rightarrow \Lambda$$

- Therefore, this CFG is regular grammar.

Chomsky Normal Form CNF

Some of CFGs' can produce the same string in several different ways and this CFG is called *ambiguous* and the case is known as *ambiguity*.

Example:

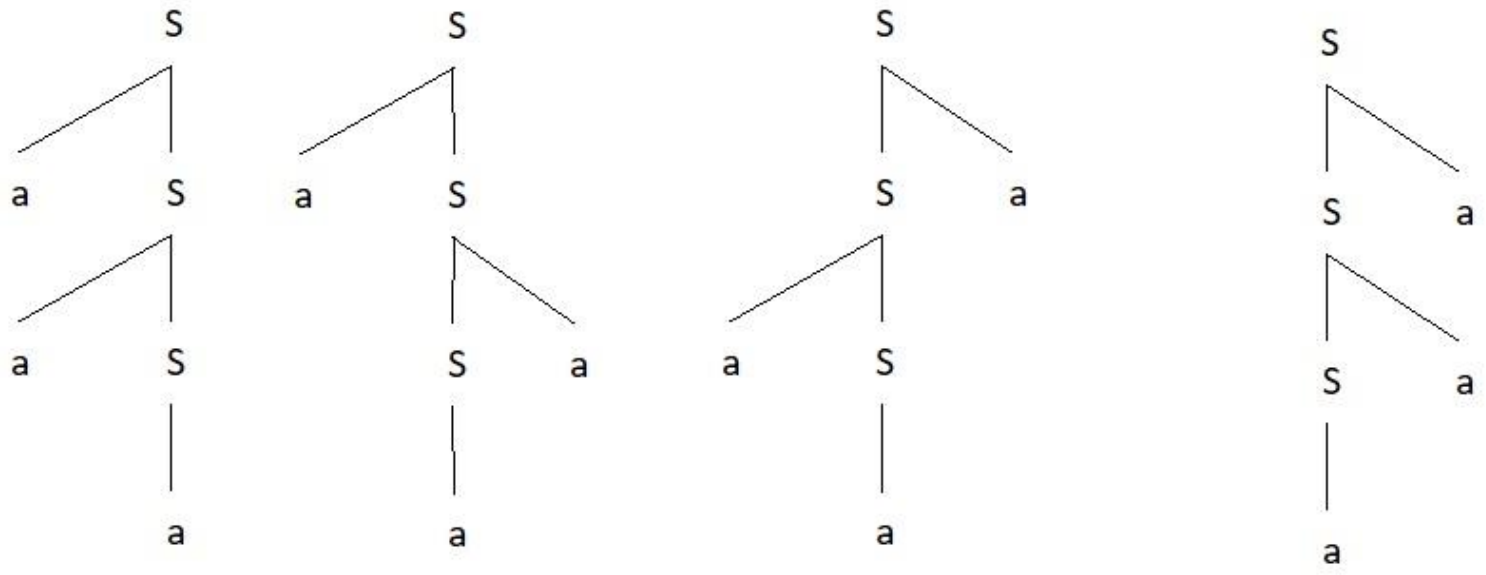
The following CFG defines a stings of a's:

$$S \rightarrow aS \mid Sa \mid a$$

---------- Continued ----------

Chomsky Normal Form CNF

To drive a string of three as from the CFG, there different ways to do that as shows in the following trees:



Chomsky Normal Form CNF

- Chomsky Normal Form was developed to cover this issue.
- A CFG said to be CNF when its productions in the following form:

Non-terminal → two non-terminals

or

Non-terminal → one terminal

Chomsky Normal Form CNF

- To perform this transferring, some steps are needed:
  - Remove NULL-production.
  - The semi-word which is a string of non-terminal, and terminals should be represented as a string of couple of non-terminals.

Chomsky Normal Form CNF

Example of removing NULL-production.

The following CFG generates a language which is defined by the RE = $a(a+b)^*$.

$S \rightarrow aX \mid a$

$X \rightarrow aX \mid bX \mid \Lambda$

The equivalent CFG without Λ-production is:

$S \rightarrow aX \mid a$

$X \rightarrow aX \mid bX \mid a \mid b$

Chomsky Normal Form CNF

Example of handling semi-word.

The following CFG for a language that their strings must end with *a*

$S \rightarrow Xa$

$X \rightarrow a \mid b$

The modified CFG is:

$S \rightarrow Xa$

$X \rightarrow a \mid b$

$A \rightarrow a$

Chomsky Normal Form CNF

Example of transferring CFG into CNF.

The following CFG produces EVENPALINDROM language.

$S \rightarrow ASA$
$S \rightarrow BSB$
$S \rightarrow AA$
$S \rightarrow BB$
$A \rightarrow a$
$B \rightarrow b$

---------- Continued ----------

Chomsky Normal Form CNF

The CNF of the original CFG is:

$S \rightarrow AR_1$

$S \rightarrow BR_2$

$S \rightarrow AA$

$S \rightarrow BB$

$R_1 \rightarrow SA$

$R_2 \rightarrow BA$

$A \rightarrow a$

$B \rightarrow b$

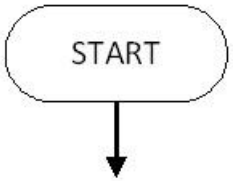
Pushdown Automata PDA

- Pushdown Automata was designed to examine the CFG whether its regular or not.
- It consists of 8 items:
  - An input alphabet Σ.
  - The input string will be stored in an input TAPE which is labelled starting from cell-I and ended by blank Δ.
  - A pushdown stack is used to store characters Γ and it is initially filled with blank Δ.

---------- Continued ----------

Pushdown Automata PDA

- One START state that has only out-edge and no in-edge.



- Halt state of two things, some ACCEPT and some REJECT. They have in-edge and no out-edge.

----------Continued ----------

Pushdown Automata PDA

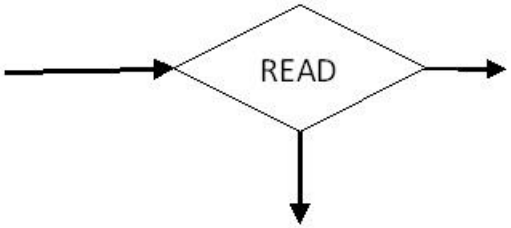
- Finitely many non-branching PUSH states that introduce characters onto the top of the STACK, they are of the form:-

Where, X is any letter in Γ.

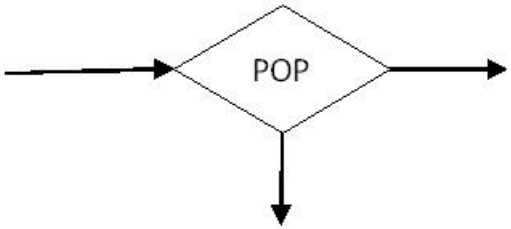
---------- Continued ----------

Pushdown Automata PDA

- Finitely many branching states of two things:-
  - States that read the next unused letter from the TAPE:-



  - State that read the top character of the STACK:-



Pushdown Automata PDA

- The procedure of PDA will be as follows:

  ❖Starting from START state and follow the un-labelled edge to generate a path through the graph.

  ❖The path will be ended either at the halt state or crash in a state when there is no edge corresponding to the input letter.

---------- Continued ----------

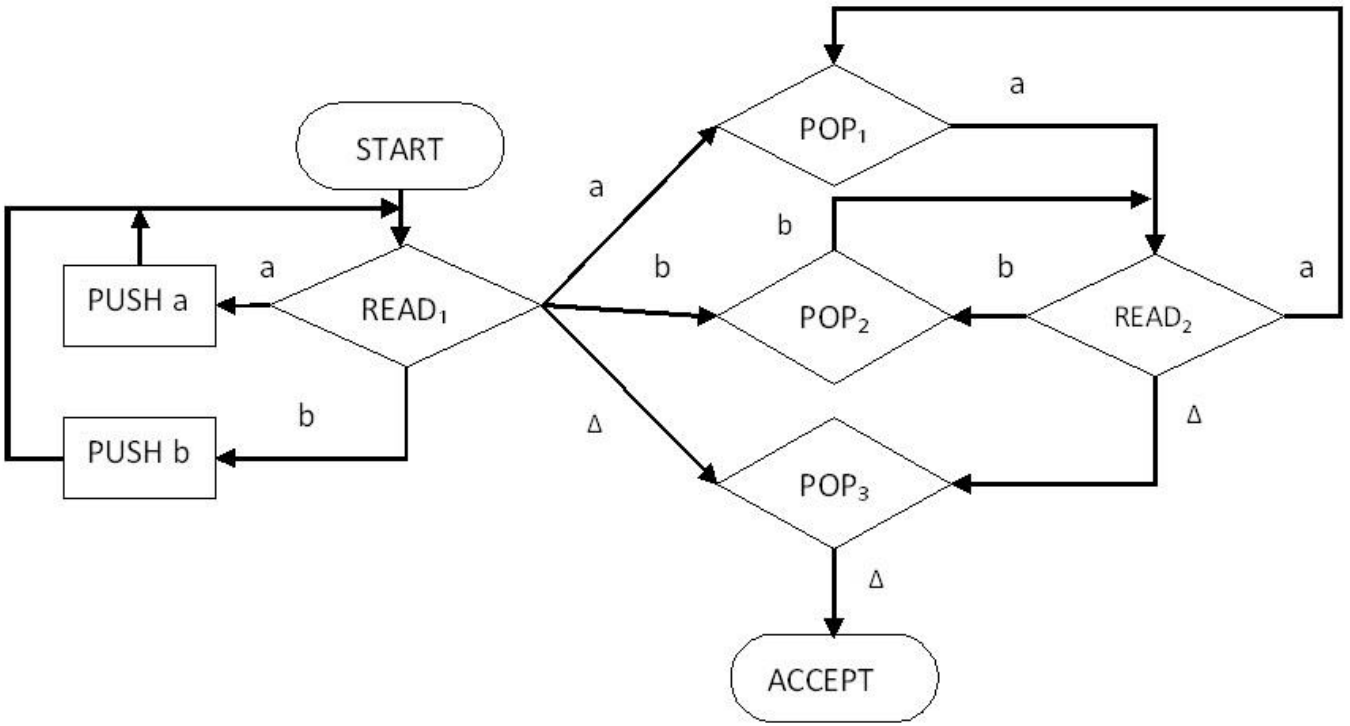
Pushdown Automata PDA

- The letter will be vanished when it is read from the TAPE or popped from the STACK.

- A path of string which is ended in ACCEPT it is accepted and the set of strings which are accepted by PDA it called *the language accepted by the PDA.*

Pushdown Automata PDA

Example: the following PDA defines EVENPALINDROM language.



---------- Continued ----------

Pushdown Automata PDA

- Let us examine the string *babbab whither* it is accepted by the PDA or not. This can be done by tracing the string through the machine.
- The input string will be stored in the TAPE.

TAPE	b	a	b	b	a	b	Δ	

- The following table explains the tracing process of the string through the PDA.

---------- Continued ----------

Pushdown Automata PDA

STATE	TAPE	STACK
START	babbabΔ	Δ
READ1	~~b~~abbabΔ	Δ
PUSH b	babbabΔ	bΔ
READ1	baabbabΔ	bΔ
PSUH a	baabbabΔ	abΔ
READ1	babbabΔ	abΔ
PUSH b	babbabΔ	babΔ
READ1	babbabΔ	babΔ
POP2	babbabΔ	abΔ
READ2	babbabΔ	abΔ
POP1	babbabΔ	bΔ
READ2	babbabΔ	bΔ
POP1	babbabΔ	Δ
READ2	babbabΔ	Δ
POP3	babbabΔ	Δ
ACCEPT	babbabΔ	Δ

Turing Machine TM

- It is a mathematical representation of computer which developed by *Alan Mathison Turing*.

- The importance of TM is that it has an output which conveys people of the operation results.

Turing Machine TM

TM has the following items:

- An alphabet Σ of input letters that do not contain the blank symbol Δ for clarity purpose.

- The input string will be stored in a TAPE that consists of a sequence of cells. Each cell contains a single character of the input string. The initial values of the TAPE are blanks Δ.

Turing Machine TM

- A HAED read the content of the TAPE cells and can be moved to the left or right. The initial location of the HEAD is the first cell (cell i) and the HEAD has to be relocated to the right or the machine will be crashed.
- The input string is written on the TAPE by HEAD. If the HEAD writes blank Δ in a cell, it means removing the content of the cell not writing a blank as a character in the cell.

Turing Machine TM

- A machine which consists of a set of states; one START state where the execution begins and HALT states where the execution is finished, a number of states which do not have a function it has only names.
- A program which is a set of rules that labels the machine edges by three sections: (Letteri, Lettero, direction).
- Letteri, the input character is read by the HEAD.

---------- Continued ----------

Turing Machine TM

- Lettero, the character which is written by the HEAD.
- direction, move the HEAD to a specific direction.
- Turing Machine is deterministic, nor more than one leaving edge have the same Letteri.

Turing Machine TM

- TM will be crashed when:

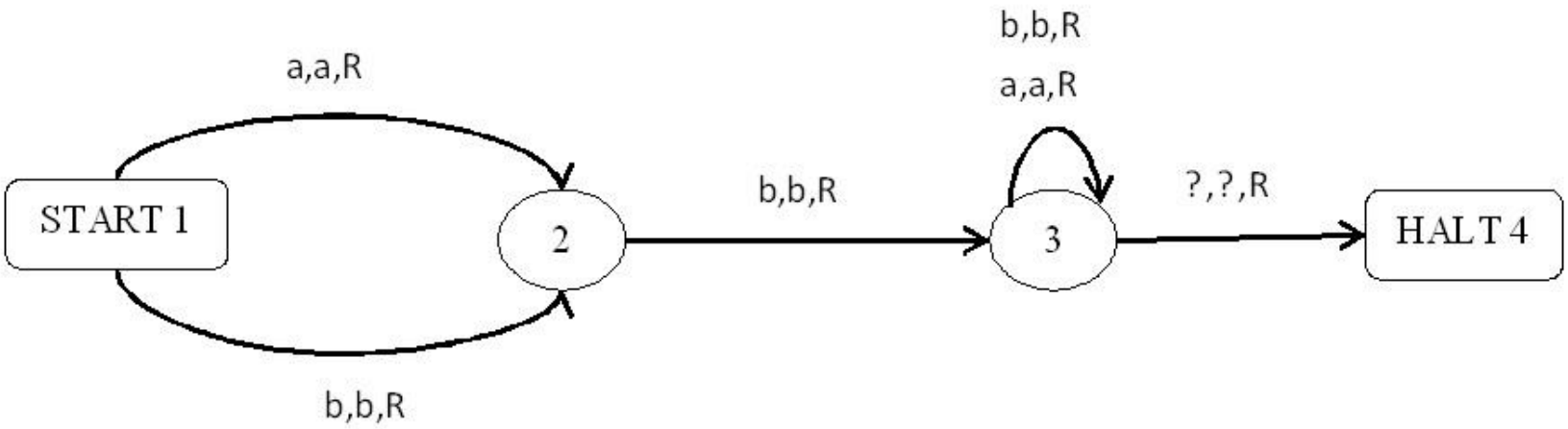
  1. The HEAD is moved to the right while it is in the first cell.

  2. The path of the input string will not arrive at the HALT state.

Turing Machine TM

Example:

Let us trace the input string *aba* on the following TM:



Turing Machine TM

STATE	TAPE & TAPE HEAD
START1	a̲ba
2	ab̲a
3	aba̲
3	aba$\underline{\Delta}$
HALT 4	aba$\Delta\underline{\Delta}$