* Grammar

→ Grammar can be defined as a finite set of formal rules that are generating syntactically correct sentences.

→ A grammar 'G' can be defined as a quadruple / four tuples, i.e.

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

G: is the grammar, that consists of a set of production rules, that is used to generate the strings of a language.

V: Variables (or not terminal symbols). Denoted by capital letters.

T: set of terminal symbols

P: is the production rule, used for terminals and the variables (non-terminals). Production rule helps in the generation of the language.

S: Start symbol.

Example $S \to aSb \mid \varepsilon$

S Minimum string can be produced will be ε, because $S \to \varepsilon$ Production rule.

Sol

$\varepsilon, aSb, aaSbb, aaaSbbb$

$\Rightarrow \varepsilon, ab, a^2b^2, a^3b^3, \ldots a^nb^n; n \geq 0$

ab ab X

Example

$S \to SS$

$S \to asb$; ε, asbbsa, bsabsa

$S \to bSa$ $\Rightarrow \varepsilon$, abba, baba

$S \to \varepsilon$

$$L = n_a(w) = n_b(w)$$

Note:- Equivalent Grammar

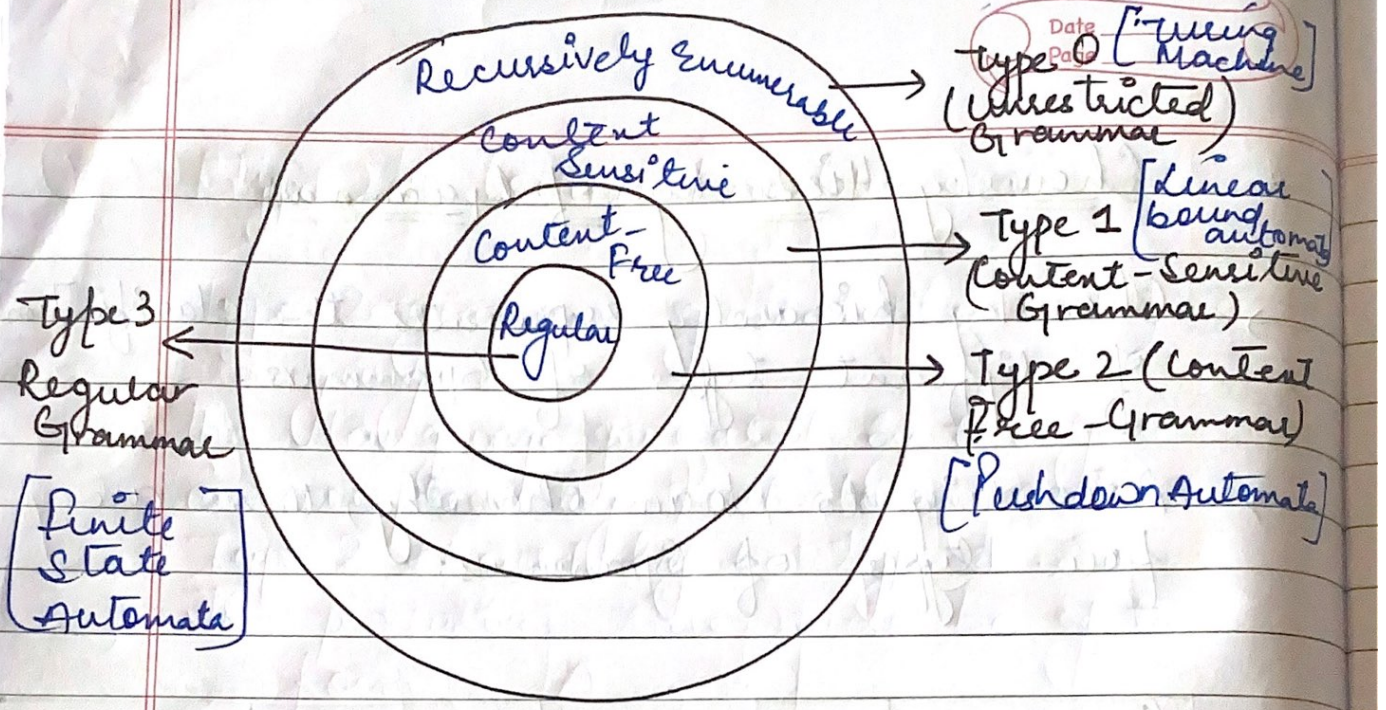
Grammars are said to be equivalent if they produce the same language.

* Chomsky Hierarchy of Grammar

→ Chomsky hierarchy represents the classification of different types of Grammars.

→ According to Noam Chomsky, there are four types of Grammars:

Grammar Type	Grammar Accepted	Language Accepted	Automaton
Type 0	Unrestricted Grammar	Recursively enumerable Language	Turing Machine
Type 1	Context-Sensitive Grammar	Context Sensitive Language	Linear bounded automata
Type 2	Context free Grammar	Context Free L	Pushdown Automata
Type 3	Regular Grammar	Regular Language	Finite state Automata



Type 0: Unrestricted Grammar

→ Type 0 Grammar generate recursively enumerable languages.

→ These are recognised by Turing Machines.

→ Type 0 grammars include formale grammars

→ There is no restriction for the productions. In type 0, there must be atleast one variable on the left side of production.

→ The productions can be of the form $\alpha \rightarrow \beta$ where α is a string of terminals and non terminals (variables) with atleast one non-terminal and α can't be null. β is a string of terminals & variables

Example

$S \rightarrow abA$

$A \rightarrow bb$

Type 1: Context Sensitive Grammar

→ Type 1 grammar generates Context sensitive languages.

→ The language generated by Type 1 grammar is recognised by Linear Bound automata.

→ This grammar may have more than one symbol on the left hand side of their production rules.

→ The no. of symbols on LHS must not exceed the no. of symbols on R.H.S

→ The rule $S \rightarrow \varepsilon$ is not allowed unless S is the start symbol and it does not occur on the RHS.

→ The production is of the form $V \rightarrow T$, where the count of symbol in V is less than or equal to T.

→ Type 1 grammar should be in type 0.

Example

$S \rightarrow AT$

$T \rightarrow xy$

$A \rightarrow a$

Type 2 : Context Free Grammar

→ Type 2 grammars generate Context free languages.

→ The language generated by the grammar is accepted by Pushdown automata.

→ Type 2 should be type 1.

→ The production rule must be of the form $\boxed{A \rightarrow \alpha}$ where A is any non-terminal and α is any combination of terminals and non terminals.

→ The LHS of production rule can have only one variable and there is no restriction on α.

Example

$S \rightarrow Xa$

$X \rightarrow a$

$X \rightarrow aX$

$X \rightarrow abc$

$X \rightarrow \varepsilon$

Type 3: Regular Grammar

→ Type 3 grammars generate regular language
→ Regular languages are those that are described using regular expressions
→ These are the languages that are accepted by a finite state automata.
→ Type 3 is the most restricted form of a grammar.
→ These must have a single non terminal on the left hand side and a right hand side consisting of a single terminal or a single terminal followed by a single non-terminal.
→ The rule $S \to \varepsilon$ is allowed if S does not appear on the right side of any rule.

Example

$$X \to \varepsilon$$
$$X \to a \mid aY$$
$$Y \to b$$

There are two types of regular grammar:
→ Left Linear Grammar
→ Right Linear Grammar

Right Linear Grammar :- A grammar is said to be in right linear if the productions are of the form

$A \rightarrow xB$

$A \rightarrow x$, where $A, B \in V$ & $x \in T$,

i.e. if the non terminal symbol lies to the right of the terminal symbol, it is said to be right linear grammar.

Left Linear Grammar A grammar is said to be left linear, if the non-terminal symbol lies to the left of the terminal symbol in the production, i.e.

$A \rightarrow Bx$

$A \rightarrow x$

where $A, B \in V$ and $x \in T$.

* Derivations from a Grammar

The set of all the strings that can be derived from a grammar is said to be the Language generated from that Grammar.

Example: $G_1 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to a, B \to b\})$

$S \to AB$
$\to ab$
$L(G_1) = \{a, b\}$

Example: $G1 = (\{S, A, B\}, \{a, b\}, S, \{S \to AB, A \to aA \mid a, B \to bB \mid b\})$

$S \to AB$ $\quad$ $S \to AB$ $\quad$ $S \to AB$ $\quad$ $S \to AB$
$\to aB$ $\quad$ $\to aAbB$ $\quad$ $\to aAb$ $\quad$ $\to abB$
$\to aabb$ $\quad$ $\to aab$ $\quad$ $\to abb$

$L(G1) = \{ab, a^2b^2, a^2b, ab^2 \text{---}\}$
$= \{a^m b^n \mid m \geq 0 \text{ and } n \geq 0\}$

Formal Language: In automata theory, a formal language is a set of string of symbols drawn from a finite alphabet. A formal language can be specified either by a set of rules, (such as regular expression or context free grammar) that generates the language, or by a formal machine that accepts (recognizes) the language.

Formal languages can be grouped into a series of successively larger classes k/as the Chomsky Hierarchy.

* Recursive Language

The language L is recursive (decidable) if L is the set of strings accepted by some Turing Machine that halts on every input

Example

When a Turing Machine reaches a final state, it halts. It can also be concluded that a turing machine M halts when M reaches a state q and current symbol 'a' to be scanned so that $\delta(q, a)$ is undefined

There may be some TMs that do not halt on some inputs in any ways. So, we can differentiate between the languages accepted by a TM that halts on all input strings and a TM that never halts on some input strings.

Two states: Halt and Accept
Halt and reject

Note: Recursive Languages are also called as Turing decidable languages.

Recursive Enumerable Languages

A recursive enumerable language can be accepted by the turing machine, which means it will enter into the final state for the strings of the language and may or may not enter the rejecting state for the strings that do not belong to the language.

This means that the TM can enter into the loop state forever for all the strings which are not a part of the language.

RE languages are also called as Turing recognizable languages.

→ Halt and Accept
→ Halt and Reject
→ Never halt

Explanation

The language L is recursively enumerable if L is the set of strings accepted by some TM

If L is a RE language then:

If $w \in L$, then TM halts in final state
If $w \notin L$, then TM halts in a non-final state or loops forever.

If L is recursive language, then -
If $w \in L$, then TM halts in a final state
If $w \notin L$, then TM halts in a non-final state.

Recursive Languages are also RE Language

Proof: If L is a recursive language, then there is TM which decides a member in language; then

→ M accepts x, if x is in language L
→ M rejects on x, if x is not in language L.

According to the definition, M can recognize the strings in language that are accepted on those strings.

Decidability

A problem is called decidable, when there is a solution to that problem and also can construct algorithms corresponding to it

For example : Given a DFA, does it accept a given word (Acceptance problem for DFA)

Given two DFAs, do they accept the same language ? (equivalence problem for DFA)

Undecidable language : A decision problem P is said to be undecidable if the language L of all yes instances to P is not decidable.

An undecidable language may be partially decidable but not decidable.

If a language is not even partially decidable, then there exist no TM for that language.

Partially Decidable / Semi-Decidable :- A decision problem P is said to be partially decidable, if

the language L of all yes instances to P is RE (Recursively enumerable)

In terms of FA, decidable refers to the problem of testing whether a DFA accepts an input string. A decidable language corresponds to algorithmically solvable decision problems.

Decidability Theorem

A language L over Σ is called decidable if,

→ There exist a TM 'M' that accepts language L.

* Decidability for Recursive language

For Recursive language:

→ A language 'L' is said to be recursive if there exist a TM which will accept the strings in 'L' and reject all the strings not in 'L'.

→ The TM will halt everytime and give an answer either accepted or rejected for

each and every input

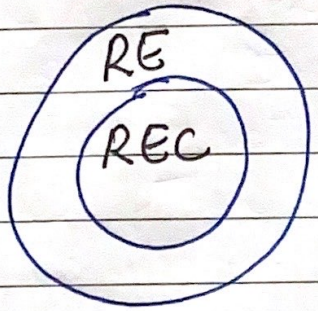
→ A language L is decidable if it is a recursive language.

→ All decidable languages are recursive languages and vice-versa.

For Recursively Enumerable languages

A language L is said to be RE language if there exist a Turing Machine which will accept (and hence halt) for all the input strings included in L but may or may not halt for the strings ∉ L.

Therefore, all recursive languages are also RE languages but not all RE languages are Recursive languages.

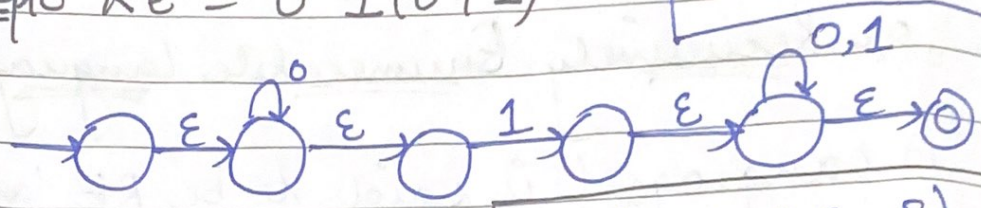


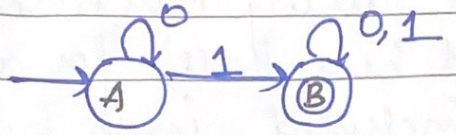
* Regular Expression to Regular Grammar

→ Convert RE to FA
→ Convert FA to RG.

$L(G) = L(M)$

Example $RE = 0^*1(0+1)^*$





V: A, B
T: 0, 1

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

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

Q → set of non-terminals (V)
Σ → Terminal symbols (T)
δ → P (Production Rule)
q_0 → start symbol (S)

Production Rules

$A \rightarrow 0A$
$A \rightarrow 1B$
$A \rightarrow 1$
$B \rightarrow 0B$
$B \rightarrow 0$
$B \rightarrow 1B$
$B \rightarrow 1$

Start Symbol

$S = \{A\}$

For Production Rule, P:
If $\delta(q_0, a) = P$, then $q \rightarrow aP$ in case if P is non-final state

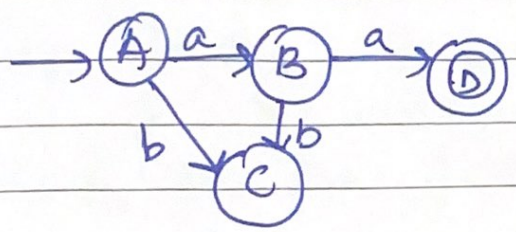
If $\delta(q_0, a) = P$, then $q \rightarrow aP$ when P is final state. $q \rightarrow a$

* Regular Grammar to Regular Expression

a) Convert RG to FA
b) FA to RE (using Arden's Theorem)

Example

$G = (\{A, B, C\}, \{a, b\}, \{A \to aB; A \to bC; B \to a, B \to bC\}, \{A\})$



{convert into RE}

{Arden's Theorem can be used}

$A = \varepsilon$ — (1)

$B = Aa$ — (2)

$C = Ab + Bb$ — (3)

$D = Ba$ — (4)

Substitute (1) in (2)

$B = Aa$

$B = \varepsilon a$

$\boxed{B = a}$ — (5) (Identities)

$D = aq$

Regular expression for G is 'aa'.