2020-21 Onwards(MR-20)	**MALLA REDDY ENGINEERING COLLEGE (Autonomous)**	**B.Tech. V Semester**		
Code: A0523	**FORMAL LANGUAGES AND AUTOMATATHEORY**	**L**	**T**	**P**
Credits: 4		**3**	**1**	**-**

Prerequisites: NIL

Course Objectives:

This course enables the students to define basic properties of formal languages, explain the Regular languages and grammars, inter conversion, Normalizing CFG, describe the context free grammars, minimization of CNF, GNF and PDA, designing Turing Machines and types of Turing Machines, church's hypothesis counter machines, LBA, P & NP problems and LR grammar.

MODULE I: **Introduction** [13 Periods]

Basics of Formal Languages - Alphabet, Strings, Language, Operations, Chomsky hierarchy of languages, Finite state machine Definitions, finite automation model, acceptance languages.

Nondeterministic Finite Automata: Formal Definition, an application, Text Search, Finite Automata with Epsilon-Transitions.

Deterministic Finite Automata: Definition of DFA, How A DFA Process Strings, The language of DFA, Conversion of NFA with €-transitions to NFA without €-transitions. Conversion of NFA to DFA, Moore and Melay machines

MODULE II: **Regular Languages** [13 Periods]

Representation of Regular Expressions - Regular Sets, Regular Expressions, identity Rules, Constructing Finite automata for the given regular expressions, Conversion of Finite automata to regular expressions.

Pumping Lemma - **Pumping lemma of regular sets, closure properties of regular sets (proofs not required).** Regular Grammars – right linear and left linear grammars, equivalence between regular grammar and FA.

MODULE III: **CFG and PDA** [14 Periods]

A: **Context Free Grammar** - Derivation trees, sentential forms, right most and left most derivations of strings. Ambiguity in Context frees Grammars. Minimization of Context free grammars, CNF, GNF, Pumping Lemma for Context Free Languages.

B: **Push Down Automata** - Definition, model, acceptance of CFL, Acceptance by final state, acceptance by empty state and its equivalence, Equivalence of CFL and PDA (proofs not required), Introduction to DCFL and DPDA.

MODULE IV: **Computable Functions** [12 Periods]

Turing Machine - Definition, model, Design of TM, computable functions.

Recursive Enumerable Languages and Theorems - Recursively enumerable languages, Church's hypothesis, counter machine, types of Turing Machines (proofs not required)

MODULE V: **Computability Theory** [12 Periods]

Linear Bounded Automata - Linear Bounded Automata and context sensitive languages, LR (0) grammar, decidability of problems, Universal TM.

P and NP Problems - Undecidable problems about Turing Machine – Post's Correspondence Problem, The classes P and NP.

TEXT BOOKS:

1. H.E. Hopcroft, R. Motwani and J.D Ullman, "Introduction to Automata Theory, Languages and Computations", Second Edition, Pearson Education, 2007.
2. KVN SUNITHA N Kalyani, "Formal languages and Automata Theory", Pearson Education

REFERENCES:

1. H.R.Lewis and C.H.Papadimitriou, "Elements of The theory of Computation", Second Edition, Pearson Education/PHI, 2003
2. J.Martin, "Introduction to Languages and the Theory of Computation", Third Edition, TMH, 2003.
3. Micheal Sipser, "Introduction of the Theory and Computation", Thomson Brokecole, 1997.

E-RESOURCES:

1. https://www.iitg.ernet.in/dgoswami/Flat-Notes.pdf
2. https://www.pdfdrive.com/introduction-to-automata-theory-formal-language-and-computability-theory-e37220113.html
3. https://freevideolectures.com/course/3379/formal-languages-and-automata-theory
4. http://nptel.ac.in/courses/111103016/

Course Outcomes:

At the end of the course, students will be able to

1. **Define** the theory of automata types of automata and FA with outputs.
2. **Differentiate** regular languages and applying pumping lemma.
3. **Classify** grammars checking ambiguity able to apply pumping lemma for CFLvarious types of PDA.
4. **Illustrate** Turing machine concept and in turn the technique applied in computers.
5. **Analyze** P vs NP- Class problems and NP-Hard vs NP-complete problems, LBA, LR Grammar, Counter machines, Decidability of Problems.

CO- PO, PSO Mapping
(3/2/1 indicates strength of correlation) 3-Strong, 2-Medium, 1-Weak

COs	**Programme Outcomes (POs)**												**PSOs**		
	PO1	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**	**PSO3**
CO1	3	2	2									2	2	2	
CO2		2	2	2	2							2	2	2	
CO3		2	2	2	2							2	2	2	
CO4		2	2	2	2							2	2	2	
CO5		2	2	2	2							2	2	2	

MODULE III: CFG and PDA

B: **Push Down Automata** - Definition, model, acceptance of CFL, Acceptance by final state, acceptance by empty state and its equivalence, Equivalence of CFL and PDA (proofs not required), Introduction to DCFL and DPDA.

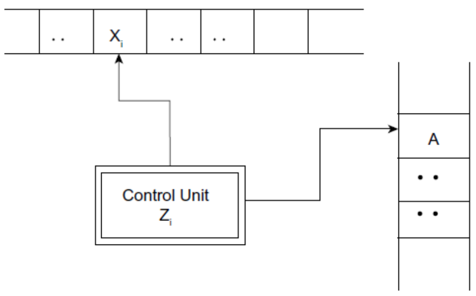
Module – III Subjective Question Bank with Notes

S NO	Questions	Marks	BT Level	CO
	Module-III			
1	i) Design a PDA which accepts $L = \{WCW^r \mid W \in (a+b)^*\}$. ii) Construct a PDA equivalent to the following grammar: $S \rightarrow aAA,\ A \rightarrow aS \mid bS \mid a$	**3** **2**	**L4** **L3**	**3**
	OR			
2	i) Design a PDA which accepts the language $L = \{a^n b^n \mid n \geq 1\}$. ii) Design a PDA that accepts $L = \{a^3 b^n c^n \mid n \geq 0\}$.	**3** **2**	**L4** **L2**	**3**
3	i. Design a PDA which accepts equal number of a's and b's over $\Sigma = \{a, b\}$. ii. Design a PDA that accepts $L = \{0^n 1^{2n} \mid n \geq 1\}$.	**2** **3**	**L2**	**3**
	OR			
4	Construct the equivalent CFG for the following PDA $M = \{\{q0, q1\}, \{a, b\}, \{Z, Z0\}, \delta, q0, Z0, \emptyset\}$ where δ is defined by $\delta(q0, b, Z0) = (q0, ZZ0)$ $\delta(q0, \varepsilon, Z0) = (q0, \varepsilon)$ $\delta(q0, b, Z) = (q0, ZZ)$ $\delta(q0, a, Z) = (q1, Z)$ $\delta(q1, b, Z) = (q1, \varepsilon)$ $\delta(q1, a, Z0) = (q0, Z0)$	**5**	**L3**	**3**

The automata has very limited memory capabilities. It cannot recognize all CFLs. We understand that this is because Finite Automata have strictly finite memories; whereas recognition of a CFL may require storing an unbounded amount of information. For example, scanning a string from the language $L = \{a^n b^n\}$. We must not only check that all a's precede the first b, but also match the number of a's with the number of b's. Since n is unbounded, this counting cannot be done with a finite memory.

The difficulty can be avoided by adding an auxiliary memory in the form of a stack. All the a's are read and added to a stack. This is how they are remembered. When b's are read, match the number of a's with the number of b's by taking out an a from the stack for each b. This is how we can strengthen the power of automaton, by adding a stack structure.

Pushdown Automata:



Conceptual Model of a Pushdown Automation

As Fig indicates, a PDA consists of three components: 1) An input tape, 2) a finite control and 3) a stack structure. The input tape consists of a linear configuration of cells each of which contains a character from the input alphabet. This tape can be moved one cell at a time to the right. The stack is also a sequential structure that has a first element and grows in either direction from the other end. The control unit has some pointer (head) which points the current symbol that is to be read. The head positioned over the current stack element can read and write special stack characters from that position. The current stack element is always the top element of the stack; hence, the name stack. The control unit contains both tape head and stack head and finds itself at any moment in a particular state.

2. Define pushdown automaton

Def: A finite state pushdown automation is a 7-tuple M = $(Q, \Sigma, \delta, \tau, q_0, Z_0, F)$

where Q: a finite set of states

Σ: a finite set of input alphabet

Γ: a finite set of stack alphabet

q_0: the start state. $q_0 \in Q$

$F \subseteq Q$: a set of final state, and

δ: a transition function from

$Q \times (\Sigma \cup \{\varepsilon\}) \times (\Gamma) \rightarrow Q \times \Gamma^*$

Z_0 is the initial stack symbol. $Z_0 \in \Gamma$.

PDA has 2 alphabets:

a) An input alphabet Σ (for input strings)

b) A stack alphabet Γ (stored on the stack)

A move on PDA may indicate:

i) An element may be added to the stack

ii) An element may be deleted from the stack:
$(q, a, Z_0) = (q, \varepsilon)$ and

iii) There may or may not be change of state

Ex:

i) $\delta(q, a, Z_0) = (q, aZ_0)$ indicates that in the state q on seeing a, a is pushed onto stack. There is no change of state.

ii) $\delta(q, a, Z_0) = (q, \varepsilon)$ indicates that in the state q on seeing a, the current top symbol Z_0 is deleted from the stack.

iii) $\delta(q, a, Z_0) = (q1, aZ_0)$ indicates that a is pushed onto the stack and the state is changed to q1.

Graphical Representation of PDA

Let M = $(Q, \Sigma, \tau, \delta, q_0, Z_0, F)$ be a PDA where $Q = \{p, q\}$, $\Sigma = \{a, b, c\}$, $\tau = \{a, b\}$, $q_0 = q$, $F = \{p\}$, and δ is given by the following equations:

$\delta(q, a, \varepsilon) = \{(q, a)\}$

$\delta(q, b, \varepsilon) = \{(q, b)\}$

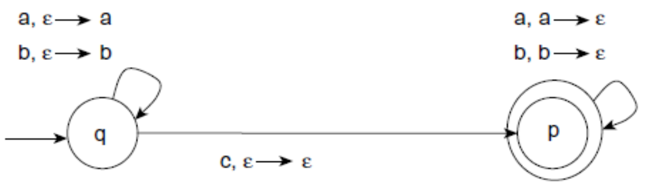
$\delta(q, c, \varepsilon) = \{(p, \varepsilon)\}$

$\delta(p, a, a) = \{(p, \varepsilon)\}$

$\delta(p, b, b) = \{(p, \varepsilon)\}$

It is very hard to visualize what M does. The transition diagram helps us to understand. We use the notation a, b → c on a transition between states q_i and q_j to mean

$\delta(q_i, a, b) = \{(q_j, c)\}$.



In words, M works as follows: It stays in state q pushing input symbols ‘a’ and ‘b’ onto the stack (irrespective of what is at the top of the stack) until it encounters a ‘c’. Then it moves to state p, in which it repeatedly pops a and b symbols off the stack provided the input symbol is identical to that on top of the stack. If at the end of the string, the stack is empty, then the machine accepts. However, notice that failure in processing a string is not explicitly represented. For example, what happens if the input string has no c in it? In that case, M will never leave state q. Once the input is finished, we find that we are not in a final state and so cannot accept the string.

Instantaneous Description of PDA

During processing, the PDA moves from one configuration to another configuration. At any given instance, the configuration of PDA is expressed by the state of finite control, the content of stack and the input. The configuration is expressed as a triple (q, x, γ), where

1. q is the state.
2. x is the input string to be processed.
3. γ is the content of the stack where the leftmost symbol corresponds to top of stack, and the rightmost is the bottom element.

When string ababcbcb is processed, the instantaneous description is as shown below.

$$
\begin{aligned}
& (q, ababcbab, e) \\
\Rightarrow\ & (q, babcbab, a) \\
\Rightarrow\ & (q, abcbab, ba) \\
\Rightarrow\ & (q, bcbab, aba) \\
\Rightarrow\ & (q, cbab, baba) \\
\Rightarrow\ & (p, bab, baba) \\
\Rightarrow\ & (p, ab, aba) \\
\Rightarrow\ & (p, b, ba) \\
\Rightarrow\ & (p, \varepsilon, a)
\end{aligned}
$$

At this point, the input string is exhausted and the computation stops. We cannot accept the original string although we are in an ‘accept’ state because the stack is not empty.

Thus we see that L(M) = $\{wcw^R \mid w \in \{a, b\}^*\}$. This example used **c as a marker** telling the machine when to change states. It turns out that such an expedient is not needed because we have non-determinism at our disposal.

Language Acceptance by PDA

A language can be accepted by a PDA using two approaches:

1. **Acceptance by final state:** The PDA accepts its input by consuming it and finally it enters the final state.
2. **Acceptance by empty stack:** On reading the input string from the initial configuration for some PDA, the stack of PDA becomes empty.

1) **Design a PDA which accepts L = $\{WCW^r \mid W \in (a + b)^*\}$.**

Read the string w and push it onto the stack until it encounters c. After that, read each symbol, if it matches with the top of the stack, pop off the symbol. When input is read completely, if stack becomes empty, then string is accepted.

The PDA can be given as follows:
Let q_0 be initial state, q_f be final state and Z_0 be the bottom of the stack as shown in Figure.

$$\delta(q_0, a, Z_0) = (q_0, aZ_0)$$
$$\delta(q_0, a, a) = (q_0, aa)$$
$$\delta(q_0, a, b) = (q_0, ab)$$
$$\delta(q_0, b, Z_0) = (q_0, bZ_0)$$
$$\delta(q_0, b, a) = (q_0, ba)$$
$$\delta(q_0, b, b) = (q_0, bb)$$
$$\delta(q_0, c, Z_0) = (q_1, Z_0)$$
$$\delta(q_0, c, a) = (q_1, a)$$
$$\delta(q_0, c, b) = (q_1, b)$$
$$\delta(q_1, a, a) = (q_1, \varepsilon)$$
$$\delta(q_1, b, b) = (q_1, \varepsilon)$$
$$\delta(q_1, \varepsilon, Z_0) = (q_f, Z_0)$$

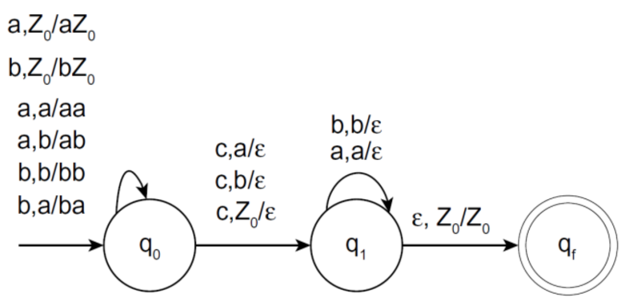


Fig. *PDA*

The final PDA is given by M = {(q_0, q_1, q_f), (a, b, c), (a, b, Z_0), δ,q_0, Z_0, {q_f}}.

2. Convert CFG to PDA.
I → a | b, S → aA, A → aABC | bB | a, B → b, C → c

Suppose L is a context free language.
Then there is a PDA M such that L = N(M).

It is required to construct M = ({q}, T,V, δ, q_1, S,Φ) where δ(q, a, A) contains (q, γ) whenever A→aγ is in P.

PDA accepted by empty stack is given by M = ({q}, {a, b, c}, {a, b, c, S, A, B,C, I}, {δ, q, S, φ}) where transition function δ is given by

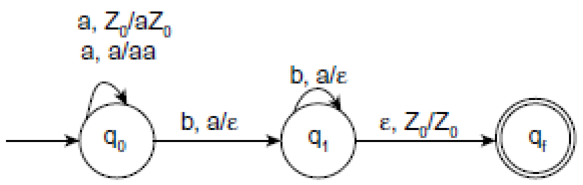
$$\delta(q, a, I) = \{(q, \varepsilon)\}$$
$$\delta(q, b, I) = \{(q, \varepsilon)\}$$
$$\delta(q, a, S) = \{(q, A)\}$$
$$\delta(q, a, A) = \{(q, ABC), (q, \varepsilon)\}$$
$$\delta(q, b, A) = \{(q, B)\}$$
$$\delta(q, b, B) = \{(q, \varepsilon)\}$$
$$\delta(q, c, C) = \{(q, \varepsilon)\}$$

3. Design a PDA which accepts the language L = {$a^n b^n$ | n ≥ 1}.

PDA can be constructed as follows:
Let q_0 be initial state, q_f final state and Z_0 bottom of the stack. Read each ‘a’ and push it onto the stack. Then read each b and pop out from stack for matching with a’s. When all b’s are read, if the stack is empty, then string is valid. If any a’s are left over on stack or b’s on input type then the string is rejected.

Suppose the input is aaabbb. The steps involved are shown in Figure.

The PDA moves are the following:
$\delta(q_0, a, Z_0) = (q_0, aZ_0)$ push a
$\delta(q_0, a, a) = (q_0, aa)$ push a
$\delta(q_0, b, a) = (q_1, \varepsilon)$ pop a and change the state
$\delta(q_1, b, a) = (q_1, \varepsilon)$ pop a
$\delta(q_1, \varepsilon, Z_0) = (q_f , Z_0)$ change to final state q_f and halt



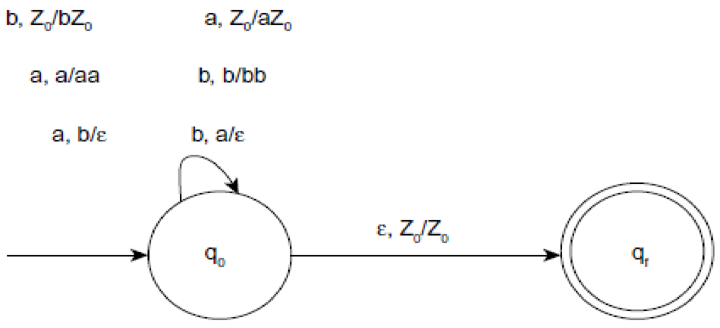
4. Design a PDA which accepts equal number of a's and b's over Σ = {a, b}.

The input can start with either a or b, and they can occur in any order, as bbaaba or bababa or babbaa and so on are all valid in the language.

To design such a PDA, read the first symbol either a or b push it on the stack. Then read the next symbol. If the next symbol and top of stack are same, push it onto the stack. Whenever top of stack and tape symbol are different, pop off the stack. When entire tape is read, if the stack becomes empty, then string is accepted.

The PDA can be given as follows:

Let q_0 be initial state, q_f be final state and Z_0 be the bottom of the stack and is shown in the Figure



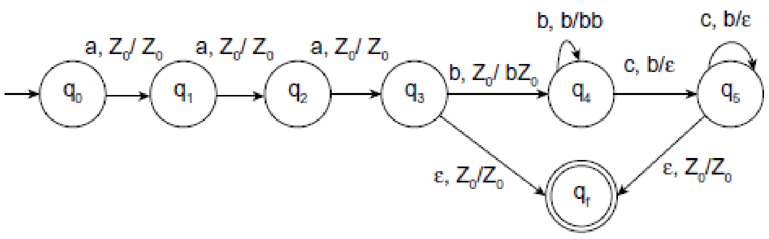
$\delta(q_0, a, Z_0) = (q_0, aZ_0)$
$\delta(q_0, b, Z_0) = (q_0, bZ_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, a, b) = (q_0, \varepsilon)$
$\delta(q_0, b, a) = (q_0, \varepsilon)$
$\delta(q_0, \varepsilon, Z_0) = (q_f, Z_0)$

5. Design a PDA that accepts $L = \{a^3b^nc^n \mid n \geq 0\}$.

Here, ensure that exactly three a's are read initially. Then read all b's onto the stack and match them with c's by poping off. To ensure three a's, read each 'a' and change the state and is shown in Figure.

$\delta(q_0, a, Z_0) = (q_1, Z_0)$
$\delta(q_1, a, Z_0) = (q_2, Z_0)$
$\delta(q_2, a, Z_0) = (q_3, Z_0)$
$\delta(q_3, \varepsilon, Z_0) = (q_f, Z_0)$
$\delta(q_3, b, Z_0) = (q_4, bZ_0)$

$\delta(q_4, b, b) = (q_4, bb)$
$\delta(q_4, c, b) = (q_5, \varepsilon)$
$\delta(q_5, c, b) = (q_5, \varepsilon)$
$\delta(q_5, \varepsilon, Z_0) = (q_f, Z_0)$



6. List and illustrate the rules to convert the PDA to CFG

Suppose M is a PDA. Then there is a grammar G such that L(G) = L(M), i.e., L(M) is context free.

The basic idea is to consider any two states p, q of PDA M and think about what strings could be consumed in executing M from p to q. Those strings will be represented by a variable [p, a, q] in G, the grammar we are building. Thus S, the start variable, will stand for all strings consumed in going from q_0 to an accept state. The construction goes as follows: given PDA M = {Q, Σ, τ, δ, q_0, Z_0, Ø}, we will construct a grammar G such that L(G) = L(M).

To convert the PDA to CFG, we use the following three rules:
R1: The productions for start symbol S are given by
$S \rightarrow [q_0, Z_0, q]$ for each state q in Q.

R2: Each move that pops a symbol from stack with transition as $\delta(\boldsymbol{q}, a, Z_i) = (\boldsymbol{q1}, \varepsilon)$
induces a production as $[\boldsymbol{q}, Z_i, \boldsymbol{q_1}] \rightarrow a$ for $\boldsymbol{q_1}$ in Q.

R3: Each move that does not pop symbol from stack with transition as $\delta(\boldsymbol{q}, a, Z_0) = (\boldsymbol{q_1}, Z_1Z_2 Z_3Z_4\ldots..)$

induces a production as
$[\boldsymbol{q}, Z_0, \boldsymbol{q}m] \rightarrow a[\boldsymbol{q_1}, Z_1\ q_2]\ [q_2, Z_2\ q_3]\ [q_3, Z_3\ q_4]\ [q_4, Z_4\ q_5]\ldots[q_{m-1}, Z_m\ \boldsymbol{q}_m]$ for each $\boldsymbol{q}_i$ in Q, where $l \le i \le m$.

After defining all the rules, apply simplification of grammar to get reduced grammar.

7. Examine Construct the equivalent CFG for the following PDA M = {{q_0, q_1}, {a, b}, {Z, Z_0}, δ, q_0, Z_0, Ø} where δ is defined by

$\delta(q_0, b, Z_0) = (q_0, ZZ_0)$
$\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$
$\delta(q_0, b, Z) = (q_0, ZZ)$
$\delta(q_0, a, Z) = (q_1, Z)$
$\delta(q_1, b, Z) = (q_1, \varepsilon)$
$\delta(q_1, a, Z_0) = (q_0, Z_0)$

The states are q_0 and q_1, and the stack symbols are Z and Z_0. The states are {S, $[q_0, Z_0, q_0]$, $[q_0, Z_0, q_1]$, $[q_1, Z_0, q_0]$, $[q_1, Z_0, q_1]$, $[q_0, Z, q_0]$, $[q_0, Z, q_1]$, $[q_1, Z, q_0]$, $[q_1, Z, q_1]$}.

S- Productions are given by Rule 1 $S \rightarrow [q_0, Z_0, q_0] \mid [q_0, Z_0, q_1]$	$[q_0, Z_0, q_0] = A$ $[q_0, Z, q_0] = C$ $[q_0, Z_0, q_1] = B$ $[q_0, Z, q_1] = D$ $S \rightarrow A \mid B$

(1) The CFG for $\delta(q_0, b, Z_0) = (q_0, ZZ_0)$ is obtained by rule 3	$[q_1, Z_0, q_0] = E$ $[q_1, Z_0, q_1] = F$
$[q_0, Z_0, q_0] \rightarrow b\,[q_0, Z, q_0]\,[q_0, Z_0, q_0]$	$A \rightarrow bCA$
$[q_0, Z_0, q_0] \rightarrow b\,[q_0, Z, q_1]\,[q_1, Z_0, q_0]$	$A \rightarrow bDE$
$[q_0, Z_0, q_1] \rightarrow b\,[q_0, Z, q_0]\,[q_0, Z_0, q_1]$	$B \rightarrow bCB$
$[q_0, Z_0, q_1] \rightarrow b\,[q_0, Z, q_1]\,[q_1, Z_0, q_1]$	$B \rightarrow BDF$
(2) The CFG for $\delta(q_0, \varepsilon, Z_0) = (q_0, \varepsilon)$ is obtained by rule 2	
$[q_0, Z_0, q_0] \rightarrow \varepsilon$	$A \rightarrow \varepsilon$
(3) The CFG for $\delta(q0, b, Z) = (q0, ZZ)$ is obtained by rule 3	$[q_1, Z, q_0] = G$ $[q_1, Z, q_1] = H$
$[q_0, Z, q_0] \rightarrow b\,[q_0, Z, q_0]\,[q_0, Z, q_0]$	$C \rightarrow bCC$
$[q_0, Z, q_0] \rightarrow b\,[q_0, Z, q_1]\,[q_1, Z, q_0]$	$C \rightarrow bDG$
$[q_0, Z, q_1] \rightarrow b\,[q_0, Z, q_0]\,[q_0, Z, q_1]$	$D \rightarrow bCD$
$[q_0, Z, q_1] \rightarrow b\,[q_0, Z, q_1]\,[q_1, Z, q_1]$	$D \rightarrow bDH$
(4) The CFG for $\delta(q_0, a, Z) = (q_1, Z)$ is obtained by rule 3	
$[q_0, Z, q_0] \rightarrow a\,[q_1, Z, q_0]$	$C \rightarrow aG$
$[q_0, Z, q_1] \rightarrow a[q_1, Z, q_1]$	$D \rightarrow aH$
(5) The CFG for $\delta(q_1, b, Z) = (q_1, \varepsilon)$ is obtained by rule 2	$H \rightarrow b$
$[q_1, Z, q_1] \rightarrow b$	
(6) The CFG for $\delta(q_1, a, Z_0) = (q_0, Z_0)$ is obtained by rule 2	
$[q_1, Z_0, q_0] \rightarrow a\,[q_0, Z_0, q_0]$	$E \rightarrow aA$
$[q_1, Z_0, q_1] \rightarrow a[q_0, Z_0, q_1]$	$F \rightarrow aB$

Simplifying grammar: In the above grammar, first identify the non-terminals that are not defined and eliminate the productions that refer to these productions. Similarly, use the procedure of eliminating the useless symbols and useless productions. Hence the complete grammar is as follows:

$S \rightarrow [q_0, Z_0, q_0]$	$S \rightarrow A$
$[q_0, Z_0, q_0] \rightarrow b\,[q_0, Z, q_1]\,[q_1, Z_0, q_0]$	$A \rightarrow bDE$
$[q_0, Z_0, q_0] \rightarrow \varepsilon$	$A \rightarrow \varepsilon$
$[q_0, Z, q_1] \rightarrow b\,[q_0, Z, q_1]\,[q_1, Z, q_1]$	$D \rightarrow bDH$
$[q_0, Z, q_1] \rightarrow a[q_1, Z, q_1]$	$D \rightarrow aH$
$[q_1, Z, q_1] \rightarrow b$	$H \rightarrow b$
$[q_1, Z_0, q_0] \rightarrow a\,[q_0, Z_0, q_0]$	$E \rightarrow aA$

3. Construct PDA to accept if–else of a C program and convert it to CFG. (This does not accept nested if statements).

Let the PDA P = ({q}, {i, e}, {X, Z}, δ, q, Z, Ø), where δ is given by
$\delta(q, i, Z) = \{(q, XZ)\}$,
$\delta(q, e, X) = \{(q, \varepsilon)\}$ and
$\delta(q, \varepsilon, Z) = \{(q, \varepsilon)\}$

Non-terminals are {S, [qXq], [qZq]},
Equivalent productions are
$S \rightarrow [qZq]$
$[qZq] \rightarrow i[qXq][qZq]$
$[qXq] \rightarrow e$
$[qZq] \rightarrow \varepsilon$

If [qZq] is renamed to A and [qXq] is renamed to B, then the CFG can be defined by
G = ({S, A, B}, {i, e}, P, S) where P is
$S \rightarrow A$
$A \rightarrow iBA \mid \varepsilon$
$B \rightarrow e$

4. Construct PDA to accept the language L = {$a^n b^{2n}$ | a, b ∈ Σ, n ≥ 1}, by final state.

Acceptance by final state: The PDA accepts its input by consuming it and finally it enters the final state.

Here, we have to match each ‘a’ with two b’s. So, Read the a’s and push it on the stack. After that, read one ‘b’ and change to different state. Then read second b. Now pop off from the stack and reset the state to initial state to continue the same process with each ‘b’. When input is read completely, if the stack becomes empty, then it is successful.

PDA M = ({q_0, q_1, q_2, q_f}, {a, b}, {a, Z_0}, δ, q_0, Z_0, {q_f}), where δ is defined by the following rules:
$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$,
$\delta(q_0, a, a) = \{(q_0, aa)\}$
$\delta(q_0, b, a) = \{(q_1, a)\}$,
$\delta(q_1, b, a) = \{(q_2, \varepsilon)\}$
$\delta(q_2, b, a) = (q_1, a)$
$\delta(q_2, \varepsilon, Z_0) = \{(q_f, Z_0)\}$

Construct a PDA equivalent to the following grammar: S → aAA, A → aS | bS | a

The grammar is in GNF. Hence we can apply the rule as follows:

$S \rightarrow aAA$	corresponds $\delta(q, a, S) = (q, AA)$
$A \rightarrow aS$	corresponds $\delta(q, a, A) = (q, S)$
$A \rightarrow bS$	corresponds $\delta(q, b, A) = (q, S)$
$A \rightarrow a$	corresponds $\delta(q, a, A) = (q, \varepsilon)$

5. Explain the model of Deterministic PDA and Deterministic Context Free Language.

A PDA that has at most one choice of move in any state is called a deterministic PDA. Nondeterministic PDA (NPDA) provides non-determinism in the moves defined. Deterministic PDAs (DPDAs) are very useful in programming languages. For example, parsers used in **Yet Another Compiler Compiler (YACC)** are deterministic PDA’s (DPDA).

Definition 2: A PDA P = (Q, Σ, τ, δ, q_0, Z_0, F) is deterministic if and only if
1. δ(q, a, X) has at most one move for q ∈ Q, a ∈ Σ ∪ {ε}, and X ∈ τ.
2. If δ(q, a, X) is not empty for some a ∈ Σ, then δ(q, e, X) must be empty.

DPDA is less powerful than NPDA. The context free languages could be recognized by NPDAs.

Deterministic Context Free Language (DCFL)

The class of languages accepted by DPDA is in between that of regular languages and CFLs. Such a language is called a deterministic context free language (DCFL) and is a subset of the languages accepted by NPDA. NPDA can be constructed for accepting language of palindromes, but not by DPDA.

The syntax of programming languages can be described by DCFLs. Compiler writing system requires a restricted form of context free grammar, also known as DCFG. LR grammars are restricted grammars that generate DCFL.

It is useful for the compiler designer to find out whether the given grammar is suitable for defining the syntax of the language. To find whether the language is a CFL, we can use this pumping lemma; but this pumping lemma cannot be used to find whether the grammar is DCFL. To identify whether the language is DCFL or not, we can use the closure properties of DCFL as they are not closed on all properties as CFLs.

6. Compare NFA and PDA.

NFA	PDA
The language accepted by NFA is the regular language	The language accepted by PDA is a context free language
NFA has no memory	PDA is essentially a NFA with a stack (memory)
It can store only a limited amount of information	Amount of information it can store is unlimited
A language/string is accepted only by reaching the final state	It accepts a language either by an empty stack or by reaching a final state

7. Compare NPDA and DPDA.

DPDA	NPDA
The PDA that has at most one choice of move in any state is called a deterministic PDA	NPDA provides non-determinism to PDA
Deterministic PDA's (DPDAs) are very useful in programming languages. For example, parsers are deterministic.	DCFLs are recognized by NPDA. Syntax of most of the programming languages is described by DCFLs.
Strings such as $\{wcw^R \mid w \text{ is in } (a+b)^*\}$can be recognized by DPDA	Strings such as $\{ww^R \mid w \text{ is in } (a+b)^*\}$ cannot be recognized by DPDAs.

P. V Ramana Murthy
LEE; B.E(Comp); M.Tech(CS); (Ph.D(CSE));
Malla Reddy Engineering College (Autonomous)