

THEORY OF COMPUTATION

Chapter 5

PUSH DOWN AUTOMATA

Study Notes for Engineering Students

Topics Covered in this Chapter
5.1 Definition of a Pushdown Automaton
5.2 Two Types of Acceptance by PDA
5.3 Correspondence between PDA and Context Free Language: PDA corresponding to a given CFG; CFG corresponding to a given PDA; Only Concept of Deterministic PDA and Deterministic CFL

This Book Includes
Detailed theory in paragraph, list and table form • Neatly drawn diagrams • Fully solved examples with step-by-step traces • Key points for quick revision • 15 MCQs with Answer Key • 10 Broad Questions

CONTENTS

About this Chapter	3
5.1 Definition of a Pushdown Automaton	5
5.1.1 Why do we need a Pushdown Automaton?	5
5.1.2 Model of a PDA	5
5.1.3 Formal Definition of a PDA	6
5.1.4 Instantaneous Description (ID) and Moves	7
5.1.5 Solved Example 1: PDA for $L = \{anbn \mid n \geq 1\}$	7
5.1.6 Solved Example 2: PDA for $L = \{wcwR \mid w \in \{a, b\}^*\}$	8
5.1.7 Nondeterminism in a PDA: the language $\{wwR\}$	9
5.1.8 Finite Automaton versus Pushdown Automaton	10
5.1.9 Applications of PDA	10
5.2 Two Types of Acceptance by PDA	11
5.2.1 Acceptance by Final State	11
5.2.2 Acceptance by Empty Stack	11
5.2.3 Comparison of the Two Modes of Acceptance	12
5.2.4 Equivalence of the Two Modes	12
5.3 Correspondence between PDA and Context Free Language	15
5.3.1 PDA Corresponding to a Given CFG	15
5.3.2 CFG Corresponding to a Given PDA	18
5.3.3 Deterministic PDA and Deterministic CFL (Concept Only)	20
Quick Revision Table	23
Multiple Choice Questions (15)	24
Answer Key for MCQs	26
Broad Questions (10)	27

About this Chapter

Finite automata (DFA and NFA) are the simplest computing machines. They accept exactly the **regular languages**, but they have only a finite amount of memory, so they cannot count without limit. A finite automaton cannot even check whether a string has the same number of a's followed by the same number of b's, i.e. the language $\{a^n b^n \mid n \geq 1\}$. Real programming languages, however, contain nested structures such as balanced brackets, nested blocks and nested expressions. To recognise such languages we need a machine with an **unbounded memory** that is accessed in a disciplined way. The **Pushdown Automaton (PDA)** provides this memory in the form of a **stack**.

In this chapter you will learn what a PDA is, how it computes, in what two ways it can accept a string, and how PDAs are exactly equivalent to context free grammars. The chapter ends with the idea of deterministic PDAs, which are important in compiler design.

Learning Outcomes

- Define a PDA formally and describe its components and moves.
- Describe the configuration (instantaneous description) of a PDA and trace its computation on a string.
- Design simple PDAs for languages such as $a^n b^n$ and $w c w^R$.
- Explain acceptance by final state and by empty stack, and convert one into the other.
- Construct a PDA for a given CFG and a CFG for a given PDA.
- State the meaning of DPDA and DCFL and compare them with NPDA and CFL.

Prerequisites

Topic	What you should already know
Finite Automata	DFA, NFA, ϵ -moves, regular languages
Strings and Languages	Alphabet, string, empty string ϵ , concatenation, reversal w^R , Σ^*
Context Free Grammar	Variables, terminals, productions, derivations, leftmost derivation
Stack (Data Structure)	LIFO order, push and pop operations

Notation Used in this Chapter

Symbol	Meaning
Q	Finite set of states
Σ	Input alphabet (symbols that appear in the input string)
Γ	Stack alphabet (symbols that may be pushed on the stack)
δ	Transition function
q_0	Start (initial) state
Z_0	Initial stack symbol (bottom marker of the stack)
F	Set of final (accepting) states
ε	Empty string (no input symbol read, or nothing pushed)
$\vdash$	One move of the PDA; $\vdash^*$ means zero or more moves
(q, w, α)	Instantaneous description: state q , remaining input w , stack contents α
$L(M), N(M)$	Language accepted by M by final state, and by empty stack

Stack Convention

In every stack string, the **leftmost symbol is the top** of the stack and the rightmost symbol is the bottom. For example, the string AAZ_0 means: top = A, then A, bottom = Z_0 .

5.1 Definition of a Pushdown Automaton

5.1.1 Why do we need a Pushdown Automaton?

A finite automaton remembers the past only through its current state. Because the number of states is finite, it can distinguish only finitely many situations. To accept $a^n b^n$, the machine has to remember how many a's have been read, and n can be arbitrarily large. A finite number of states is simply not enough.

The solution is to give the machine a **stack**, which is a Last-In-First-Out (LIFO) memory of unlimited size. While reading the a's, the machine pushes one symbol for each a. While reading the b's, it pops one symbol for each b. If the stack becomes empty exactly when the input ends, the number of a's and b's was equal. A good real-life picture is a pile of plates: you may add or remove a plate only at the top.

Definition (informal)

A **Pushdown Automaton (PDA)** is a finite automaton (with ϵ -moves and nondeterminism) that is additionally equipped with a stack. In short: **PDA = NFA with ϵ -moves + Stack**.

5.1.2 Model of a PDA

The physical model of a PDA has three parts, shown in Figure 5.1.

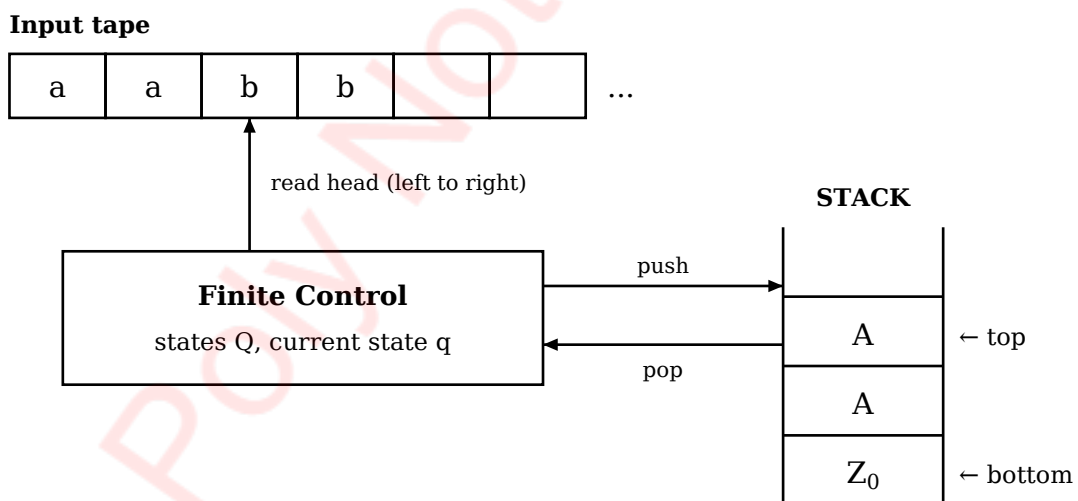


Figure 5.1: Block diagram (model) of a Pushdown Automaton

- **Input tape:** holds the input string, one symbol per cell. The read head moves only from left to right.
- **Finite control:** a finite set of states. It decides the next move using the current state, the current input symbol and the symbol on top of the stack.
- **Stack:** an unbounded LIFO store. Only the top symbol can be examined, and only the top can be changed.

What happens in one move? Depending on (i) the current state, (ii) the next input symbol (or no input at all, in an ε -move) and (iii) the top stack symbol, the PDA does three things together:

1. it changes to a new state (possibly the same state);
2. it replaces the top stack symbol by a string of stack symbols (possibly the empty string, which means a pop);
3. it advances the read head by one cell if an input symbol was read; in an ε -move the head stays where it is.

The basic stack operations can all be written as one kind of transition. Let the current top symbol be X .

Operation	Transition form	Effect on the stack
Pop	$(p, \varepsilon) \in \delta(q, a, X)$	X is removed; the stack becomes shorter by one symbol
Push one symbol	$(p, YX) \in \delta(q, a, X)$	Y is placed above X ; the stack grows by one symbol
Push many symbols	$(p, Y_1 Y_2 X) \in \delta(q, a, X)$	Y_1 and Y_2 are placed above X , with Y_1 on top
Replace	$(p, Y) \in \delta(q, a, X)$	X is replaced by Y ; the height does not change
No change	$(p, X) \in \delta(q, a, X)$	Stack remains the same; only the state changes
ε -move	$(p, \gamma) \in \delta(q, \varepsilon, X)$	Done without reading any input symbol

5.1.3 Formal Definition of a PDA

A pushdown automaton is a 7-tuple

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

Component	Meaning	Remarks
Q	Finite non-empty set of states	Same as in a finite automaton
Σ	Finite input alphabet	Symbols found in the input string
Γ	Finite stack alphabet	May be different from Σ ; usually contains extra marker symbols
δ	Transition function	$\delta : Q \times (\Sigma \cup \{\varepsilon\}) \times \Gamma \rightarrow \text{finite subsets of } Q \times \Gamma^*$
$q_0 \in Q$	Start state	The computation begins here
$Z_0 \in \Gamma$	Initial stack symbol	The stack contains only Z_0 at the start
$F \subseteq Q$	Set of final states	Used only in acceptance by final state

Understanding the transition function. The statement

$$\delta(q, a, X) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_k, \gamma_k)\}$$

means: when the PDA is in state q , the next input symbol is a and the top of the stack is X , then it may *choose any one* pair (p_i, γ_i) ; it moves to state p_i , replaces X by the string γ_i on the stack, and moves its head one cell to the right. If $a = \varepsilon$, the move is an ε -move and the

head does not advance.

Important points about δ

- The result of δ is a **set** of pairs, so a PDA is **nondeterministic** in general.
- The third argument of δ is always a stack symbol: **a PDA cannot move when its stack is empty**.
- If $\delta(q, a, X) = \emptyset$ the PDA has no move and that particular computation path dies (it is stuck).
- The right-hand side string γ is written with its leftmost symbol on top of the stack.
- Since δ returns finite subsets, there are only finitely many choices in any single move.

5.1.4 Instantaneous Description (ID) and Moves

To describe a computation we need a snapshot of the machine at any instant. This snapshot is called the **instantaneous description** (also called a **configuration**) and is the triple

$$(q, w, \gamma)$$

Part	Meaning
q	the current state
w	the remaining (still unread) part of the input string
γ	the current contents of the stack, top symbol written first

The move relation $\vdash$. If $(p, \beta) \in \delta(q, a, X)$, where $a \in \Sigma \cup \{\epsilon\}$, then for every $w \in \Sigma^*$ and $\gamma \in \Gamma^*$,

$$(q, aw, X\gamma) \vdash (p, w, \beta\gamma)$$

That is, the PDA reads a , removes X from the top, pushes β in its place, and goes to state p . The symbol $\vdash^*$ denotes zero or more consecutive moves (the reflexive and transitive closure of $\vdash$). We write $(q, w, \gamma) \vdash^* (p, \epsilon, \gamma')$ to say that the PDA can reach the configuration (p, ϵ, γ') from (q, w, γ) .

Useful property (extending the input and the stack)

If $(q, x, \alpha) \vdash^* (p, y, \beta)$, then for any string $w \in \Sigma^*$ and any $\gamma \in \Gamma^*$, $(q, xw, \alpha\gamma) \vdash^* (p, yw, \beta\gamma)$. In words: extra input at the end and extra symbols at the bottom of the stack do not disturb a computation that never looks at them. This property is used in the proofs of the CFG-PDA equivalence.

5.1.5 Solved Example 1: PDA for $L = \{a^n b^n \mid n \geq 1\}$

Idea. Push an A for every a read. After the last a , each b pops one A . When all b 's are read and only Z_0 is left on the stack, move to the final state.

Let $M = (\{q_0, q_1, q_2\}, \{a, b\}, \{A, Z_0\}, \delta, q_0, Z_0, \{q_2\})$ where δ is given below.

No.	Transition	Action performed
1	$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$	first a: push A above Z_0
2	$\delta(q_0, a, A) = \{(q_0, AA)\}$	further a's: push one more A
3	$\delta(q_0, b, A) = \{(q_1, \epsilon)\}$	first b: pop A and change to q_1
4	$\delta(q_1, b, A) = \{(q_1, \epsilon)\}$	further b's: pop one A each
5	$\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$	only Z_0 left: all a's matched, go to the final state

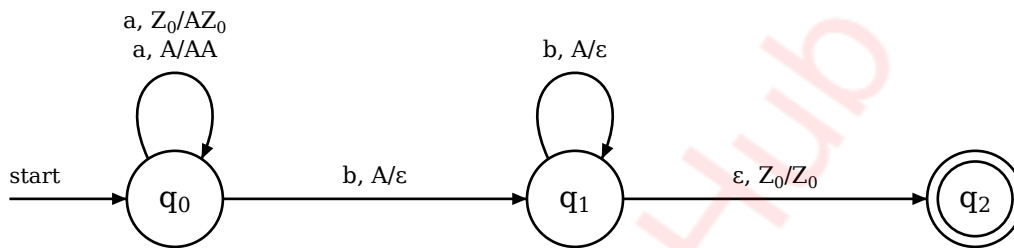


Figure 5.2: Transition diagram of the PDA for $a^n b^n$. An edge label " $a, X/\gamma$ " means: read a , top of stack is X , replace X by γ .

Computation on $w = aabb$ (accepted).

Step	Configuration (q , remaining input, stack)	Move used
1	$(q_0, aabb, Z_0)$	start configuration
2	(q_0, abb, AZ_0)	rule 1
3	(q_0, bb, AAZ_0)	rule 2
4	(q_1, b, AZ_0)	rule 3 (pop A)
5	(q_1, ϵ, Z_0)	rule 4 (pop A)
6	(q_2, ϵ, Z_0)	rule 5; input finished and $q_2 \in F$, so ACCEPT

Rejected strings. For $w = aab$: after reading all symbols the configuration is (q_1, ϵ, AZ_0) . Rule 5 needs Z_0 on top but the top is A, so the PDA is stuck and does not reach q_2 ; the string is rejected. For $w = ba$: there is no move for (q_0, b, Z_0) , so it is rejected immediately. The empty string is also rejected because the language requires $n \geq 1$.

5.1.6 Solved Example 2: PDA for $L = \{wcw^R \mid w \in \{a, b\}^*\}$

Idea. Here w^R is the reverse of w , so strings look like $abcba$ or $bacab$. The marker c tells the machine where the middle is. In state q_0 the PDA pushes every symbol of w . After reading c it goes to q_1 and compares each further input symbol with the top of the stack, popping when they match. Because the stack gives the symbols in reverse order, it matches w^R automatically.

$M = (\{q_0, q_1, q_2\}, \{a, b, c\}, \{a, b, Z_0\}, \delta, q_0, Z_0, \{q_2\})$. The transitions are:

No.	Transition	Action
1	$\delta(q_0, a, Z_0) = \{(q_0, aZ_0)\}$	push a on empty stack
2	$\delta(q_0, b, Z_0) = \{(q_0, bZ_0)\}$	push b on empty stack
3	$\delta(q_0, a, a) = \{(q_0, aa)\}; \delta(q_0, a, b) = \{(q_0, ab)\}$	push a on any symbol
4	$\delta(q_0, b, a) = \{(q_0, ba)\}; \delta(q_0, b, b) = \{(q_0, bb)\}$	push b on any symbol
5	$\delta(q_0, c, Z_0) = \{(q_1, Z_0)\}; \delta(q_0, c, a) = \{(q_1, a)\}; \delta(q_0, c, b) = \{(q_1, b)\}$	read middle marker c, stack unchanged
6	$\delta(q_1, a, a) = \{(q_1, \epsilon)\}; \delta(q_1, b, b) = \{(q_1, \epsilon)\}$	match and pop
7	$\delta(q_1, \epsilon, Z_0) = \{(q_2, Z_0)\}$	everything matched: go to final state

Computation on $w = abcba$ (accepted).

Step	Configuration (q , remaining input, stack)	Move used
1	$(q_0, abcba, Z_0)$	start
2	$(q_0, bcba, aZ_0)$	rule 1
3	(q_0, cba, baZ_0)	rule 4
4	(q_1, ba, baZ_0)	rule 5
5	(q_1, a, aZ_0)	rule 6
6	(q_1, ϵ, Z_0)	rule 6
7	(q_2, ϵ, Z_0)	rule 7, ACCEPT

This PDA is **deterministic**, because the marker c tells it exactly when to switch from pushing to popping.

5.1.7 Nondeterminism in a PDA: the language $\{ww^R\}$

Now consider $L = \{ww^R \mid w \in \{a, b\}^*\}$, i.e. even-length palindromes such as $abba$ or $aabbaa$. There is no middle marker, so the machine does not know when to stop pushing and start popping. A PDA solves this by **guessing** the middle: in addition to the push moves of Example 2, it has the ϵ -moves

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

which let it leave the pushing state at any moment without reading input. On every input at least one guess is correct if the string is of the form ww^R , and the string is accepted if *some* branch accepts. We shall see in Section 5.3.3 that no deterministic PDA can accept this language.

5.1.8 Finite Automaton versus Pushdown Automaton

Basis	Finite Automaton (FA)	Pushdown Automaton (PDA)
Memory	Finite (only the states)	Finite states plus an unbounded stack
Language class	Regular languages	Context free languages
Equivalent grammar	Regular grammar	Context free grammar
Definition	5-tuple $(Q, \Sigma, \delta, q_0, F)$	7-tuple $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
Language $a^n b^n$	Cannot be accepted	Can be accepted
Determinism	DFA and NFA have equal power	DPDA is weaker than NPDA
Typical use	Lexical analysis (tokens)	Syntax analysis (parsing)

5.1.9 Applications of PDA

- **Parsing in compilers:** checking that a program follows the context free syntax of the language.
- **Balanced symbols:** verifying that brackets, parentheses and begin-end blocks are properly nested.
- **Expression evaluation:** converting and evaluating infix, prefix and postfix expressions with a stack.
- **Markup validation:** checking proper nesting of tags in HTML and XML documents.
- **Natural language processing:** modelling nested grammatical structure.

Key Points: Section 5.1

- PDA = ϵ -NFA + stack; it accepts exactly the context free languages.
- $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$; δ maps $Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.
- A move reads at most one input symbol, pops the top symbol and pushes a string in its place.
- $ID = (\text{state}, \text{remaining input}, \text{stack})$; $(q, aw, X\gamma) \vdash (p, w, \beta\gamma)$ when $(p, \beta) \in \delta(q, a, X)$.
- A PDA cannot move on an empty stack. Nondeterminism lets it guess, e.g. the middle of ww^R .

5.2 Two Types of Acceptance by PDA

A finite automaton accepts when it ends in a final state. A PDA has one more component, the stack, so there are two natural ways to decide acceptance after the whole input has been read. Both are in common use and both lead to the same class of languages, namely the context free languages.

5.2.1 Acceptance by Final State

Definition

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. The language accepted by M **by final state** is
$$L(M) = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha) \text{ for some } q \in F \text{ and some } \alpha \in \Gamma^* \}$$

So a string w is accepted if the PDA, starting in q_0 with only Z_0 on the stack, can read the whole of w and enter a final state. **The contents of the stack do not matter:** α can be any string, including ε . The PDA of Example 1 (Section 5.1.5) works in this way: at the end, the stack still contains Z_0 and the state is $q_2 \in F$.

5.2.2 Acceptance by Empty Stack

Definition

The language accepted by M **by empty stack** (also called by null stack) is
$$N(M) = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon) \text{ for some } q \in Q \}$$

Here the string is accepted if, after the entire input has been consumed, the stack has been completely emptied. The state at that moment is irrelevant. Therefore the set F of final states plays no role, and a PDA that accepts by empty stack is written as a 6-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$, or with $F = \emptyset$.

Remark. Since a PDA cannot make a move on an empty stack, an empty-stack computation ends at the instant the last symbol is popped. If the PDA empties the stack before the input is finished, it is stuck and the string is not accepted.

Example. We modify the PDA of Example 1 so that it accepts $a^n b^n$ by empty stack. We drop q_2 and change the last rule so that it pops Z_0 instead of moving to a final state:

No.	Transition	Action
1	$\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$	push A
2	$\delta(q_0, a, A) = \{(q_0, AA)\}$	push A
3	$\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$	pop A, go to q_1
4	$\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$	pop A
5	$\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$	pop Z_0 : the stack becomes empty

Computation on $w = aabb$.

Step	Configuration (q , remaining input, stack)	Move used
1	$(q_0, aabb, Z_0)$	start
2	(q_0, abb, AZ_0)	rule 1
3	(q_0, bb, AAZ_0)	rule 2
4	(q_1, b, AZ_0)	rule 3
5	(q_1, ϵ, Z_0)	rule 4
6	$(q_1, \epsilon, \epsilon)$	rule 5; input finished and stack empty, so ACCEPT

5.2.3 Comparison of the Two Modes of Acceptance

Basis	Acceptance by final state	Acceptance by empty stack
Notation	$L(M)$	$N(M)$
Condition	Input fully read and state $\in F$	Input fully read and stack = ϵ
Role of F	Essential	Not used ($F = \emptyset$)
Stack at the end	Any contents	Must be empty
Role of the state at the end	Must be a final state	Any state
Tuple	7-tuple	6-tuple (no F)
Convenient for	Direct design of acceptors	Proving equivalence with CFG

Caution

For the *same* machine M , the languages $L(M)$ and $N(M)$ are in general **different**. For example, the machine of Example 1 has $N(M) = \emptyset$ (it never empties its stack), but $L(M) = \{a^n b^n \mid n \geq 1\}$. The statement of equivalence below is about languages, not about one fixed machine.

5.2.4 Equivalence of the Two Modes

Theorem

A language L is accepted by some PDA by final state if and only if L is accepted by some (other) PDA by empty stack.

This theorem allows us to use whichever mode is more convenient. The proof is by construction. The key trick is to introduce a **new bottom marker X_0** and new states, so that the original machine can be simulated without interference.

(a) Empty stack to final state: given M with $N(M) = L$, build M' with $L(M') = L$

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, \emptyset)$. Define $M' = (Q \cup \{p_0, pf\}, \Sigma, \Gamma \cup \{X_0\}, \delta', p_0, X_0, \{pf\})$ where X_0 is a new stack symbol, p_0 is the new start state, pf is the new final state, and δ' contains:

1. $\delta'(p_0, \varepsilon, X_0) = \{(q_0, Z_0 X_0)\}$: put the marker X_0 under Z_0 and start M ;
2. every move of M : $\delta'(q, a, Y)$ includes $\delta(q, a, Y)$ for all $q \in Q, a \in \Sigma \cup \{\varepsilon\}, Y \in \Gamma$;
3. $\delta'(q, \varepsilon, X_0) = \{(pf, \varepsilon)\}$ for every $q \in Q$: if M has emptied its own stack, X_0 becomes visible and M' moves to pf .

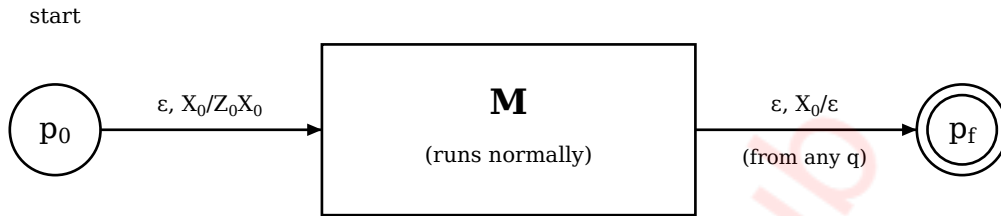


Figure 5.3: Construction of a final-state PDA from an empty-stack PDA

(b) Final state to empty stack: given M with $L(M) = L$, build M'' with $N(M'') = L$

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Define $M'' = (Q \cup \{p_0, p\}, \Sigma, \Gamma \cup \{X_0\}, \delta'', p_0, X_0, \emptyset)$ where X_0, p_0 and p are new, and δ'' contains:

1. $\delta''(p_0, \varepsilon, X_0) = \{(q_0, Z_0 X_0)\}$: start M above the marker X_0 ;
2. every move of M is retained;
3. for each $q \in F$ and each $Y \in \Gamma \cup \{X_0\}$: $(p, \varepsilon) \in \delta''(q, \varepsilon, Y)$: whenever M enters a final state, M'' may jump to p ;
4. $\delta''(p, \varepsilon, Y) = \{(p, \varepsilon)\}$ for all $Y \in \Gamma \cup \{X_0\}$: in state p the machine pops every symbol until the stack is empty.

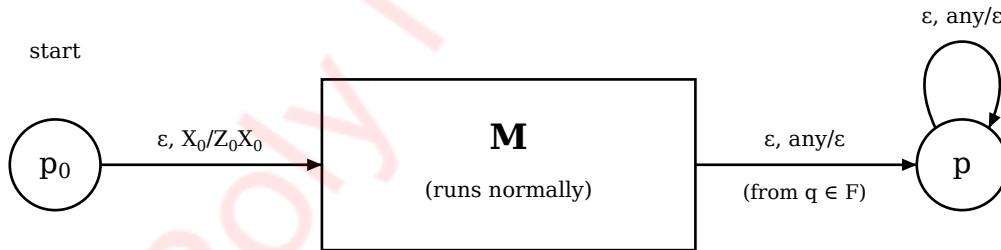


Figure 5.4: Construction of an empty-stack PDA from a final-state PDA

Why is X_0 needed? In (b), M itself might empty its stack without accepting. Without the marker, M'' would then wrongly accept that string. With X_0 at the bottom, the stack of M'' becomes empty only when state p pops everything, which can happen only after M has reached a final state. In (a), X_0 prevents M' from getting stuck with an empty stack, and lets it detect the moment when M empties its stack.

Key Points: Section 5.2

- Final state: $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \alpha)$ with $q \in F$. Empty stack: $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \varepsilon)$.
- In both modes the whole input must be read before acceptance is decided.
- The two modes accept the same class of languages (the CFLs); conversion uses a new marker X_0 and new states p_0 , p_f or p .
- For one fixed machine, $L(M)$ and $N(M)$ can be different languages.

Poly Notes Hub

5.3 Correspondence between PDA and Context Free Language

Context free grammars (CFGs) generate context free languages (CFLs). The central result of this chapter is that pushdown automata and context free grammars are two different ways of describing the *same* class of languages. This is the context free counterpart of the equivalence between finite automata and regular expressions.

Theorem (Equivalence of PDA and CFG)

A language L is context free if and only if there exists a PDA M such that $L = N(M)$. (By Section 5.2.4 the same holds for acceptance by final state.)



Both describe exactly the Context-Free Languages

Figure 5.5: Equivalence of CFG and PDA

The theorem has two halves, and each half is proved by an explicit construction: Section 5.3.1 converts a CFG into a PDA, and Section 5.3.2 converts a PDA into a CFG. Before that, let us recall the components of a CFG.

Component of $G = (V, T, P, S)$	Meaning
V	Finite set of variables (non-terminals)
T	Finite set of terminals (the alphabet of the language)
P	Finite set of productions of the form $A \rightarrow \alpha$, where $A \in V$ and $\alpha \in (V \cup T)^*$
$S \in V$	Start symbol

A **leftmost derivation** always replaces the leftmost variable of the current sentential form. Every string of $L(G)$ has at least one leftmost derivation, and this is what the PDA will imitate.

5.3.1 PDA Corresponding to a Given CFG

Idea. The PDA keeps the part of the derivation that is still to be matched on its stack and simulates a leftmost derivation of the input string. If the top of the stack is a variable A , the PDA nondeterministically replaces it by the right-hand side of one of A 's productions. If the top of the stack is a terminal a , the PDA compares it with the next input symbol; on a match both are discarded. When the stack is empty and the input is finished, the string has been derived.

Construction

Given a CFG $G = (V, T, P, S)$, the PDA is $M = (\{q\}, T, V \cup T, \delta, q, S)$, accepting by empty stack, with:

Rule	Transition	Meaning
1. For every variable A	$\delta(q, \varepsilon, A) = \{(q, \alpha) \mid A \rightarrow \alpha \text{ is a production in } P\}$	Replace A on top of the stack by the right-hand side of a production (no input is read)
2. For every terminal a	$\delta(q, a, a) = \{(q, \varepsilon)\}$	If the input symbol equals the stack top, read it and pop it

- The PDA has only **one state** q ; all the work is done by the stack.
- The stack alphabet is $\Gamma = V \cup T$, that is, variables as well as terminals.
- The initial stack symbol is the start symbol S of the grammar (so $Z_0 = S$).
- The result is $N(M) = L(G)$: w is accepted exactly when $S \Rightarrow^* w$ by a leftmost derivation.

Steps to convert a CFG into a PDA

1. Write down V, T, S and the productions of the given grammar.
2. Take a PDA with a single state q , input alphabet T , stack alphabet $V \cup T$ and initial stack symbol S .
3. For each variable A write $\delta(q, \varepsilon, A)$ containing one pair (q, α) for every production $A \rightarrow \alpha$.
4. For each terminal a write $\delta(q, a, a) = \{(q, \varepsilon)\}$.
5. Check the PDA by simulating a leftmost derivation on a sample string.

Solved Example 3: PDA for the grammar $S \rightarrow aSb \mid ab$

Here $V = \{S\}$, $T = \{a, b\}$. The PDA is $M = (\{q\}, \{a, b\}, \{S, a, b\}, \delta, q, S)$ with

No.	Transition
1	$\delta(q, \varepsilon, S) = \{(q, aSb), (q, ab)\}$
2	$\delta(q, a, a) = \{(q, \varepsilon)\}$
3	$\delta(q, b, b) = \{(q, \varepsilon)\}$

Acceptance of $w = aabb$. The leftmost derivation is $S \Rightarrow aSb \Rightarrow aabb$. The PDA imitates it as follows.

Step	Configuration (q, remaining input, stack)	Move used
1	(q, aabb, S)	start; stack holds S
2	(q, aabb, aSb)	rule 1: use $S \rightarrow aSb$
3	(q, abb, Sb)	rule 2: match a and pop
4	(q, abb, abb)	rule 1: use $S \rightarrow ab$
5	(q, bb, bb)	rule 2: match a
6	(q, b, b)	rule 3: match b
7	(q, ϵ , ϵ)	rule 3: match b; stack empty, ACCEPT

Solved Example 4: PDA for the grammar $S \rightarrow aSa \mid bSb \mid \epsilon$

This grammar generates the even-length palindromes $\{ww^R\}$. The PDA is $M = (\{q\}, \{a, b\}, \{S, a, b\}, \delta, q, S)$ with

No.	Transition
1	$\delta(q, \epsilon, S) = \{(q, aSa), (q, bSb), (q, \epsilon)\}$
2	$\delta(q, a, a) = \{(q, \epsilon)\}$
3	$\delta(q, b, b) = \{(q, \epsilon)\}$

Acceptance of $w = abba$. Derivation: $S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba$.

Step	Configuration (q, remaining input, stack)	Move used
1	(q, abba, S)	start
2	(q, abba, aSa)	rule 1: $S \rightarrow aSa$
3	(q, bba, Sa)	match a
4	(q, bba, bSba)	rule 1: $S \rightarrow bSb$
5	(q, ba, Sba)	match b
6	(q, ba, ba)	rule 1: $S \rightarrow \epsilon$
7	(q, a, a)	match b
8	(q, ϵ , ϵ)	match a; ACCEPT

Note

If the grammar is in Greibach Normal Form (every production is $A \rightarrow a\alpha$ with a terminal a), one can build a PDA that reads one input symbol in every move and uses no ϵ -moves. The construction above works for any CFG and is the one asked in examinations.

5.3.2 CFG Corresponding to a Given PDA

The reverse construction is more involved. We start with a PDA M that accepts by empty stack (if it accepts by final state, first convert it using Section 5.2.4).

Idea. Each variable of the grammar is a triple written $[p X q]$, where p and q are states and X is a stack symbol. The intended meaning is:

Meaning of the variable $[p X q]$

$[p X q]$ generates exactly those strings w for which the PDA, starting in state p with X on top of the stack, can read w , end in state q , and at the same time have **popped** X (and everything pushed above it) without ever touching the symbols below X . In symbols: $(p, w, X) \vdash^* (q, \varepsilon, \varepsilon)$.

With this meaning, the start symbol S must generate all strings w with $(q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon)$ for some state p , which is exactly acceptance by empty stack.

Construction

Given $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ accepting by empty stack, the CFG is $G = (V, \Sigma, P, S)$ where $V = \{S\} \cup \{[p X q] \mid p, q \in Q, X \in \Gamma\}$, and P contains the following productions:

Rule	Condition on δ	Productions generated
1. Start	(always)	$S \rightarrow [q_0 Z_0 p]$ for every state $p \in Q$
2. Pop move	$(r, \varepsilon) \in \delta(q, a, X)$, with $a \in \Sigma \cup \{\varepsilon\}$	$[q X r] \rightarrow a$
3. Replace/push move	$(r, Y_1 Y_2 \dots Y_k) \in \delta(q, a, X)$, with $k \geq 1$	$[q X r_k] \rightarrow a [r Y_1 r_1] [r_1 Y_2 r_2] \dots [r_{k-1} Y_k r_k]$ for all choices of the states $r_1, \dots, r_k \in Q$

Explanation of Rule 3. To pop X , the PDA first replaces X by $Y_1 Y_2 \dots Y_k$ while reading a and moving to r . Afterwards it must eventually pop Y_1 (moving from r to some state r_1), then pop Y_2 (from r_1 to r_2), and so on up to Y_k , reaching the final state r_k . Since we do not know the intermediate states, we include a production for every possible choice. For $k = 1$ the production is simply $[q X r_1] \rightarrow a [r Y_1 r_1]$.

Size of the grammar. There are at most $|Q|^2 \cdot |\Gamma| + 1$ variables. A transition that pushes k symbols gives $|Q|^k$ productions, so the grammar can be large; many variables turn out to be useless and are removed after the construction.

Steps to convert a PDA into a CFG

1. Make sure the PDA accepts by empty stack.
2. Write the start productions $S \rightarrow [q_0 Z_0 p]$ for every state p .
3. For every transition with an empty push (pop move) write the production of Rule 2.
4. For every transition that pushes one or more symbols write all productions of Rule 3.
5. Remove useless variables (those that cannot derive a terminal string or cannot be reached from S) and simplify.

Solved Example 5: CFG for a PDA

Let $M = (\{q_0, q_1\}, \{0, 1\}, \{X, Z_0\}, \delta, q_0, Z_0)$ accept by empty stack, with

No.	Transition	Action
1	$\delta(q_0, 0, Z_0) = \{(q_0, XZ_0)\}$	first 0: push X
2	$\delta(q_0, 0, X) = \{(q_0, XX)\}$	further 0's: push X
3	$\delta(q_0, 1, X) = \{(q_1, \epsilon)\}$	first 1: pop X
4	$\delta(q_1, 1, X) = \{(q_1, \epsilon)\}$	further 1's: pop X
5	$\delta(q_1, \epsilon, Z_0) = \{(q_1, \epsilon)\}$	pop Z_0 : stack empty

Step 1: start productions.

$$S \rightarrow [q_0 Z_0 q_0] \mid [q_0 Z_0 q_1]$$

Step 2: pop moves (Rule 2) from transitions 3, 4 and 5:

$$[q_0 X q_1] \rightarrow 1, [q_1 X q_1] \rightarrow 1, [q_1 Z_0 q_1] \rightarrow \epsilon$$

Step 3: push moves (Rule 3). Transition 1 pushes two symbols, so for all $r_1, r_2 \in \{q_0, q_1\}$: $[q_0 Z_0 r_2] \rightarrow 0 [q_0 X r_1] [r_1 Z_0 r_2]$. Transition 2 gives $[q_0 X r_2] \rightarrow 0 [q_0 X r_1] [r_1 X r_2]$. Written out, these are 4 + 4 productions:

$$\begin{aligned} [q_0 Z_0 q_0] &\rightarrow 0 [q_0 X q_0] [q_0 Z_0 q_0] \mid 0 [q_0 X q_1] [q_1 Z_0 q_0] \\ [q_0 Z_0 q_1] &\rightarrow 0 [q_0 X q_0] [q_0 Z_0 q_1] \mid 0 [q_0 X q_1] [q_1 Z_0 q_1] \\ [q_0 X q_0] &\rightarrow 0 [q_0 X q_0] [q_0 X q_0] \mid 0 [q_0 X q_1] [q_1 X q_0] \\ [q_0 X q_1] &\rightarrow 0 [q_0 X q_0] [q_0 X q_1] \mid 0 [q_0 X q_1] [q_1 X q_1] \end{aligned}$$

Step 4: simplification. A variable that can never end up in terminals is useless. From q_1 the PDA can never return to q_0 , so $[q_1 X q_0]$, $[q_1 Z_0 q_0]$ cannot derive any string; this makes $[q_0 X q_0]$ and $[q_0 Z_0 q_0]$ useless as well, and any production containing them can be deleted. The reduced grammar is

$$\begin{aligned} S &\rightarrow [q_0 Z_0 q_1] \\ [q_0 Z_0 q_1] &\rightarrow 0 [q_0 X q_1] [q_1 Z_0 q_1] \\ [q_0 X q_1] &\rightarrow 0 [q_0 X q_1] [q_1 X q_1] \mid 1 \\ [q_1 X q_1] &\rightarrow 1 \\ [q_1 Z_0 q_1] &\rightarrow \epsilon \end{aligned}$$

Renaming $A = [q_0 X q_1]$, $B = [q_1 Z_0 q_1]$, $C = [q_1 X q_1]$, we get the readable grammar

$$S \rightarrow 0AB, A \rightarrow 0AC \mid 1, B \rightarrow \epsilon, C \rightarrow 1$$

Check with $w = 0011$: $S \Rightarrow 0AB \Rightarrow 00ACB \Rightarrow 001CB \Rightarrow 0011B \Rightarrow 0011$. The grammar generates $\{0^n 1^n \mid n \geq 1\}$, which is the language of the PDA.

Summary: the two constructions side by side

Feature	CFG $\rightarrow$ PDA	PDA $\rightarrow$ CFG
Input	$G = (V, T, P, S)$	$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0)$ with empty-stack acceptance
Output	$M = (\{q\}, T, V \cup T, \delta, q, S)$	$G = (V', \Sigma, P', S')$
Number of states	Exactly one	Any (as in M)
Variables / stack symbols	Stack holds variables and terminals	Variables are S and triples $[p \ X \ q]$
Idea	PDA simulates a leftmost derivation	Variables describe "pop X going from p to q "
Size	Small, one rule per production	Can be large, needs simplification

5.3.3 Deterministic PDA and Deterministic CFL (Concept Only)

We have seen that a PDA is, in general, nondeterministic. A nondeterministic machine is not directly implementable, because a real program cannot "guess". The question therefore arises: can the nondeterminism be removed, as it can for finite automata? For PDAs the answer is **no**. This leads to a smaller, practically important class of languages.

Definition: Deterministic PDA (DPDA)

A PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is **deterministic** if for every $q \in Q$, $a \in \Sigma$ and $X \in \Gamma$:

- (1) $\delta(q, a, X)$ has **at most one** element, and $\delta(q, \varepsilon, X)$ has at most one element; and
- (2) if $\delta(q, \varepsilon, X) \neq \emptyset$, then $\delta(q, a, X) = \emptyset$ for every $a \in \Sigma$ (an ε -move is never possible together with a move on a real input symbol).

In other words, in every configuration the DPDA has **at most one** possible move. The PDAs of Example 1 and Example 2 are deterministic; the PDA for ww^R is not.

Definition: Deterministic Context Free Language (DCFL)

A language L is a **DCFL** if $L = L(M)$ (acceptance by final state) for some deterministic PDA M .

Remark on the mode of acceptance. For nondeterministic PDAs, final-state and empty-stack acceptance are equivalent. For deterministic PDAs this is **not** true: final-state acceptance is strictly more powerful. A DPDA accepting by empty stack can accept only languages with the *prefix property* (no word is a proper prefix of another word of the language). That is why DCFLs are defined using final states.

Comparison of DPDA and NPDA

Basis	Deterministic PDA (DPDA)	Nondeterministic PDA (NPDA)
Moves in a configuration	At most one	Zero, one or many
ϵ -moves	Allowed only if no input move is possible for the same state and stack top	Freely allowed
Languages accepted	Deterministic CFLs (DCFL)	All CFLs
Power	Weaker	Stronger. DPDA $\neq$ NPDA (unlike DFA = NFA)
Example accepted	$a^n b^n$, $w c w^R$, balanced parentheses	All of these and also ww^R
Use	Practical parsers (LR parsing) in compilers	Theoretical model of all CFLs

Facts to remember about DCFLs

- Every regular language is a DCFL, and every DCFL is a CFL. The inclusions are proper.
- $L = \{ww^R \mid w \in \{a, b\}^*\}$ is a CFL but **not** a DCFL: a deterministic machine cannot know where the middle is. Another standard example of a CFL that is not a DCFL is $\{a^i b^j c^k \mid i = j \text{ or } j = k\}$.
- Every DCFL has an unambiguous grammar. The converse is false, but any **inherently ambiguous** CFL is not a DCFL.
- DCFLs correspond to LR(k) grammars, which are used by parser generators such as YACC and Bison.

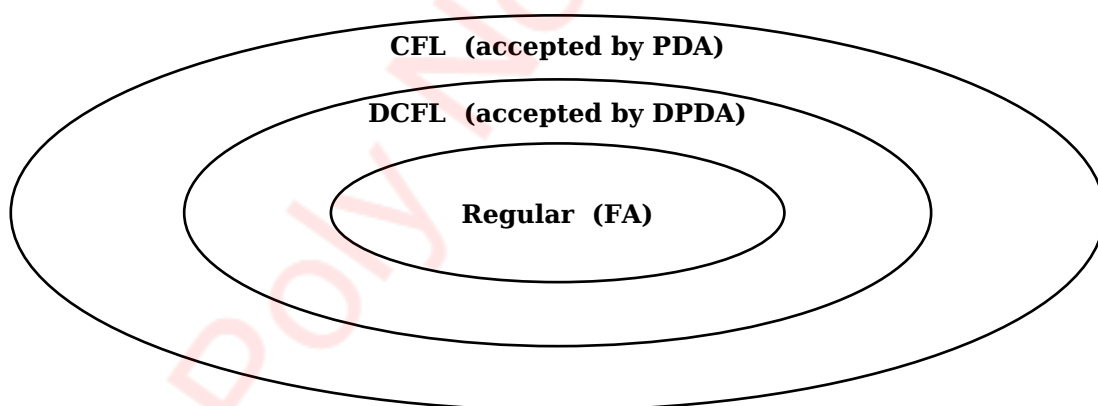


Figure 5.6: Relationship between Regular languages, DCFLs and CFLs

Closure properties (for revision)

Operation	Regular	DCFL	CFL
Union	Yes	No	Yes
Intersection	Yes	No	No
Complement	Yes	Yes	No
Concatenation	Yes	No	Yes
Kleene star	Yes	No	Yes
Intersection with a regular language	Yes	Yes	Yes

The hierarchy of languages and machines

Language class	Machine	Grammar	Example
Regular	DFA / NFA	Regular grammar	a^*b^*
DCFL	DPDA	LR(k) grammar	$a^n b^n, w c w^R$
CFL	NPDA	Context free grammar	ww^R
Context sensitive	Linear bounded automaton	Context sensitive grammar	$a^n b^n c^n$

Key Points: Section 5.3

- CFLs are exactly the languages accepted by PDAs.
- CFG $\rightarrow$ PDA: one state q , $\Gamma = V \cup T$, start stack symbol S , $\delta(q, \varepsilon, A) = \{(q, \alpha) \mid A \rightarrow \alpha\}$, $\delta(q, a, a) = \{(q, \varepsilon)\}$.
- PDA $\rightarrow$ CFG: $S \rightarrow [q_0 Z_0 p]; [q X r] \rightarrow a$ for pops; $[q X r_k] \rightarrow a [r Y_1 r_1] \dots [r_{k-1} Y_k r_k]$ for pushes.
- A variable $[p X q]$ derives the strings that take the PDA from p to q while popping X .
- DPDA has at most one move in every configuration; DCFL = language of a DPDA; Regular $\subset$ DCFL $\subset$ CFL.
- ww^R is a CFL but not a DCFL.

Quick Revision Table

Concept	Definition / Formula
PDA	$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$
Transition function	$\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \text{finite subsets of } Q \times \Gamma^*$
Instantaneous description	(q, w, γ) : state, remaining input, stack (top at left)
Move	$(q, aw, X\gamma) \vdash (p, w, \beta\gamma)$ if $(p, \beta) \in \delta(q, a, X)$
Acceptance by final state	$L(M) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \alpha), q \in F\}$
Acceptance by empty stack	$N(M) = \{w \mid (q_0, w, Z_0) \vdash^* (q, \epsilon, \epsilon)\}$
Conversion needs	New bottom marker X_0 and new states (p_0 , p_f or p)
CFG $\rightarrow$ PDA	$M = (\{q\}, T, V \cup T, \delta, q, S)$; $\delta(q, \epsilon, A) = \{(q, \alpha) \mid A \rightarrow \alpha\}$; $\delta(q, a, a) = \{(q, \epsilon)\}$
PDA $\rightarrow$ CFG	Variables $[p X q]$; $S \rightarrow [q_0 Z_0 p]$; $[q X r] \rightarrow a$ for $(r, \epsilon) \in \delta(q, a, X)$
DPDA	At most one move in every configuration
DCFL	Language accepted by a DPDA (final state); $\text{Regular} \subset \text{DCFL} \subset \text{CFL}$

Tips for Examinations

- Always write the 7-tuple completely and list all transitions in a table.
- Show the computation as a sequence of IDs; mention the rule used at each step.
- In CFG $\rightarrow$ PDA problems, always show the simulation of a leftmost derivation of one string.
- In PDA $\rightarrow$ CFG problems, remove useless variables at the end and state the simplified grammar.
- Draw the transition diagram with edge labels of the form (input, top / push string).

Multiple Choice Questions (15)

Choose the correct option. The answer key is given after the questions.

Q1. Which extra memory structure does a PDA have in comparison with a finite automaton?

- (a) Queue
- (b) Tape
- (c) Counter
- (d) Stack

Q2. A pushdown automaton is formally defined as a tuple of how many components?

- (a) 8
- (b) 5
- (c) 6
- (d) 7

Q3. Which of the following languages can be accepted by a PDA but **not** by any finite automaton?

- (a) $\{a^n b^n \mid n \geq 1\}$
- (b) $(a + b)^*$
- (c) a^*b^*
- (d) $a(a + b)^*b$

Q4. The transition $\delta(q, a, X) = \{(p, YZ)\}$ means that the PDA

- (a) moves to p, pushes X and pops YZ
- (b) stays in q and pushes YZ above X
- (c) moves to p and replaces X by YZ, with Y becoming the top of the stack
- (d) moves to p and replaces X by YZ, with Z becoming the top of the stack

Q5. The instantaneous description of a PDA is written as

- (a) (state, remaining input, stack contents)
- (b) (state, stack contents, remaining input)
- (c) (input read so far, state, stack contents)
- (d) (stack contents, input, final state)

Q6. In acceptance by empty stack, a string w is accepted if $(q_0, w, Z_0) \vdash^* (q, \varepsilon, \gamma)$ where γ equals

- (a) Z_0
- (b) any string over Γ
- (c) a final state
- (d) ε

Q7. For a PDA that accepts by empty stack, the set of final states F is

- (a) equal to Q
- (b) irrelevant (taken as $\emptyset$)
- (c) equal to $\{q_0\}$
- (d) required to contain at least one state

Q8. When a final-state PDA is converted into an empty-stack PDA, a new bottom marker X_0 is introduced mainly to

- (a) prevent the original PDA from emptying the stack without having accepted
- (b) make the PDA deterministic
- (c) store the input string
- (d) increase the number of final states

Q9. The PDA constructed from a context free grammar (by the standard method) has how many states?

- (a) $|V|$
- (b) $|T|$
- (c) 1
- (d) $|P|$

Q10. In the PDA constructed from a CFG, for every terminal a the transition is

- (a) $\delta(q, a, S) = \{(q, a)\}$
- (b) $\delta(q, \epsilon, a) = \{(q, a)\}$
- (c) $\delta(q, a, a) = \{(q, aa)\}$
- (d) $\delta(q, a, a) = \{(q, \epsilon)\}$

Q11. In the CFG constructed from a PDA, the variables other than S are of the form

- (a) single stack symbols
- (b) pairs $[p a]$
- (c) $[p X q]$ where p, q are states and X is a stack symbol
- (d) input symbols only

Q12. If $(r, \epsilon) \in \delta(q, a, X)$, the CFG construction gives the production

- (a) $[q X r] \rightarrow a$
- (b) $[r X q] \rightarrow a$
- (c) $S \rightarrow a$
- (d) $[q a r] \rightarrow X$

Q13. The class of languages accepted by nondeterministic PDAs is exactly the class of

- (a) regular languages
- (b) context free languages
- (c) context sensitive languages
- (d) recursively enumerable languages

Q14. Which of the following is a context free language but **not** a deterministic context free language?

- (a) $\{a^n b^n \mid n \geq 1\}$
- (b) $\{ww^R \mid w \in \{a, b\}^*\}$
- (c) $\{wcw^R \mid w \in \{a, b\}^*\}$
- (d) balanced parentheses

Q15. Which statement about DPDA and NPDA is correct?

- (a) Both have equal power
- (b) DPDA is less powerful than NPDA
- (c) DPDA is more powerful than NPDA
- (d) A DPDA can accept every CFL

Answer Key for MCQs

Q.	Answer	Reason
1	(d)	The stack is the extra memory; a PDA is an ϵ -NFA plus a stack.
2	(d)	$M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ has seven components.
3	(a)	$a^n b^n$ needs unbounded counting, which a stack provides; the others are regular.
4	(c)	The first symbol Y of the pushed string YZ becomes the top.
5	(a)	ID = (q, w, γ): state, unread input, stack with the top at the left.
6	(d)	Empty stack means the stack contains the empty string ϵ .
7	(b)	Final states are not used in empty-stack acceptance, so $F = \emptyset$.
8	(a)	X_0 stops M from emptying the stack on its own and giving a false accept.
9	(c)	The construction uses a single state q.
10	(d)	Matching a terminal: read a and pop a, i.e. $\delta(q, a, a) = \{(q, \epsilon)\}$.
11	(c)	Variables are triples [p X q].
12	(a)	A pop move gives $[q X r] \rightarrow a$ (Rule 2).
13	(b)	PDA $\equiv$ CFG: both define the context free languages.
14	(b)	ww^R needs a guess of the middle; no DPDA can do this.
15	(b)	DPDA $\subset$ NPDA properly, unlike DFA and NFA.

Broad Questions (10)

- Q1.** Define a pushdown automaton formally. Explain each component of the 7-tuple and draw the block diagram (model) of a PDA.
- Q2.** What is an instantaneous description of a PDA? Define the move relation $\vdash$ and $\vdash^*$. Explain with an example how a PDA processes the string aabb.
- Q3.** Design a PDA that accepts $L = \{a^n b^n \mid n \geq 1\}$ by final state. Give the transition table, draw the transition diagram and show the computation on aaabbb.
- Q4.** Construct a PDA for $L = \{wcw^R \mid w \in \{a, b\}^*\}$. Show the sequence of IDs for the string abbcbbba. Is this PDA deterministic? Justify.
- Q5.** Explain the two types of acceptance by a PDA. Define $L(M)$ and $N(M)$ with suitable examples and tabulate the differences.
- Q6.** Prove that a language accepted by a PDA by empty stack is also accepted by some PDA by final state, and conversely. Give the constructions with diagrams.
- Q7.** Describe the procedure for constructing a PDA from a given context free grammar. Construct a PDA for the grammar $S \rightarrow aSb \mid ab$ and trace the acceptance of aaabbb.
- Q8.** Convert the grammar $S \rightarrow aSa \mid bSb \mid c$ into an equivalent PDA and show that the string abcba is accepted.
- Q9.** Explain how a context free grammar is obtained from a given PDA. Construct the CFG for the PDA $M = (\{q_0, q_1\}, \{a, b\}, \{A, Z_0\}, \delta, q_0, Z_0)$ with $\delta(q_0, a, Z_0) = \{(q_0, AZ_0)\}$, $\delta(q_0, a, A) = \{(q_0, AA)\}$, $\delta(q_0, b, A) = \{(q_1, \varepsilon)\}$, $\delta(q_1, b, A) = \{(q_1, \varepsilon)\}$, $\delta(q_1, \varepsilon, Z_0) = \{(q_1, \varepsilon)\}$, and simplify it.
- Q10.** Define a deterministic PDA and a deterministic CFL. Differentiate between DPDA and NPDA, and explain with an example why $\{ww^R\}$ is a CFL but not a DCFL. Write the relationship among regular languages, DCFLs and CFLs.

— End of Chapter 5 —