Parul® University
Vadodara, Gujarat
NAAC GRADE A++

Information and Communication Technology

Grammars

Study Guide

Prof. Riddhi Atulkumar Mehta
CSE, PIT
Parul University

3.1 Context-free grammars (CFG) and languages (CFL), Chomsky normal forms

Definition of CFG

A CFG is defined as a 4-tuple:

G = (V, Σ, R, S)

- V → Set of variables (non-terminals)
- Σ → Set of terminals
- R → Set of production rules (A → α)
- S → Start symbol (S ∈ V)

Example of CFG:

Grammar G = ({S}, {a, b}, R, S) with:

- S → aSb | ε

Generates strings:

- E
- Ab
- Aabb
- Aaabbb
- ✅ Language: L = { $a^n b^n$ | n ≥ 0 }

Derivations in CFGs:

Leftmost derivation: Expand the **leftmost** non-terminal
Rightmost derivation: Expand the **rightmost** non-terminal

Parse tree visualizes the derivation

Example for S → aSb | ε
Derivation for aaabbb:
S ⇒ aSb ⇒ aaSbb ⇒ aaaSbbb ⇒ aaabbb

Ambiguity in CFGs:

- A grammar is **ambiguous** if a string has **more than one parse tree**
- Ambiguity is undesirable in programming languages

Example:

Grammar:
S → S + S | S * S | a
String: a + a * a
Two parse trees possible ⇒ ambiguous

Normal Forms in CFGs

- Normal forms are restricted forms of CFGs used in algorithms like parsing and simplification.
- Two main types:

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

Chomsky Normal Form (CNF):

Every production is of the form:

1. A → BC (non-terminals only)
2. A → a (terminal only)
3. S → ε (if ε ∈ L)

Restrictions:

- B, C ∈ V (non-terminals), B ≠ start symbol
- a ∈ Σ (terminal)

Steps for CNF Conversion

1. Eliminate ϵ\epsilonϵ-productions (except possibly S→ϵS \rightarrow \epsilonS→ϵ).
2. Eliminate unit productions: A→BA \rightarrow BA→B.
3. Eliminate useless symbols:
4. Non-generating symbols.
5. Non-reachable symbols.
6. Convert terminals in long rules to single-terminal productions.
7. Break long right-hand sides into binary productions using new variables.

3.2 Non-deterministic Pushdown Automata (NPDA) and Equivalence with CFG

Introduction

- **What is a Pushdown Automaton (PDA)?**

  A PDA is a finite automaton equipped with a stack.

Used to recognize context-free languages (CFLs).

- **Two types:**

  Deterministic PDA (DPDA)

  Non-deterministic PDA (NPDA)

Formal Definition of NPDA:

NPDA is a 7-tuple:
M=(Q,Σ,Γ,δ,q0,Z0,F)

where:

- Q: set of states
- Σ: input alphabet
- Γ: stack alphabet
- δ: transition function
- q0: start state
- Z0: initial stack symbol
- F: set of accepting states

NPDA Transition Function

$\boldsymbol{\delta : Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow P(Q \times \Gamma^*)}$

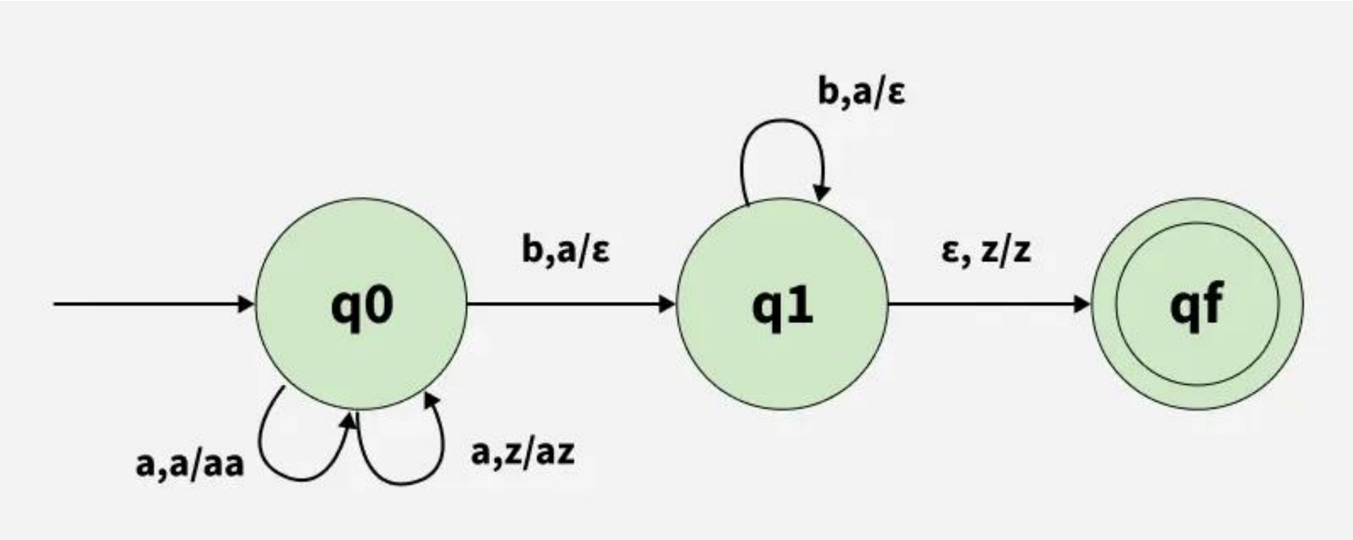
Meaning:

- Based on current state, input symbol (or ε), and top of the stack.
- Can move to new states and update the stack.

Example of NPDA

Design a non deterministic PDA for accepting the language L = {an bn | n>=1}, i.e.,

L = {ab, aabb, aaabbb, aaaabbbb, ……}



Required PDA

Stack transition functions

δ (q0, a, z) ⊢ (q0, az) [push a on empty stack]

δ (q0, a, a) ⊢ (q0, aa) [push a's]

δ (q0, b, a) ⊢ (q1, ϵ) [pop a when b comes(state change)]

δ (q1, b, a) ⊢ (q1, ϵ) [pop a for each b]

δ (q1, ϵ , z) ⊢ (qf, z) [final state]

Equivalence to CFG

- **CFG ⇒ NPDA**: For any CFG, construct an NPDA that simulates leftmost derivations.
- **NPDA ⇒ CFG**: For any NPDA, construct a CFG by simulating transitions with grammar rules.

Thus, **CFL = Language accepted by NPDAs = Language generated by CFGs.**

3.3 Parse Trees, Ambiguity in CFG, Pumping Lemma for CFLs

Parse Tree

- A parse tree visually represents how a string is derived from a CFG.
- Structure:
- Root: Start symbol

- Internal nodes: Non-terminals (variables)
- Leaves: Terminals or ε (empty string)
- Reading the leaves from left to right gives the derived string.

Definition:

Grammar:

$S \rightarrow aSb$ | ε S→aSb | ε

String: aabb

Derivation:

$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aab \Rightarrow aabb$

S

**/ | **

a S b

/|

a S b

|

ε

Each step replaces a non-terminal with its production.

Ambiguity in CFG:

- A CFG is **ambiguous** if there exists at least one string that has:
    - More than one **parse tree**, or
    - More than one **leftmost or rightmost derivation**
- Ambiguity leads to confusion in interpretation, especially in compilers.

Pumping Lemma for CFLs

- Used to **prove that a language is not context-free**
- For every context-free language L, there exists a constant p (pumping length) such that:

Then there is a pumping length n such that any string w εL of length>=n can be written as follows −

|w|>=n

We can break w into 5 strings, w=uvxyz, such as the ones given below

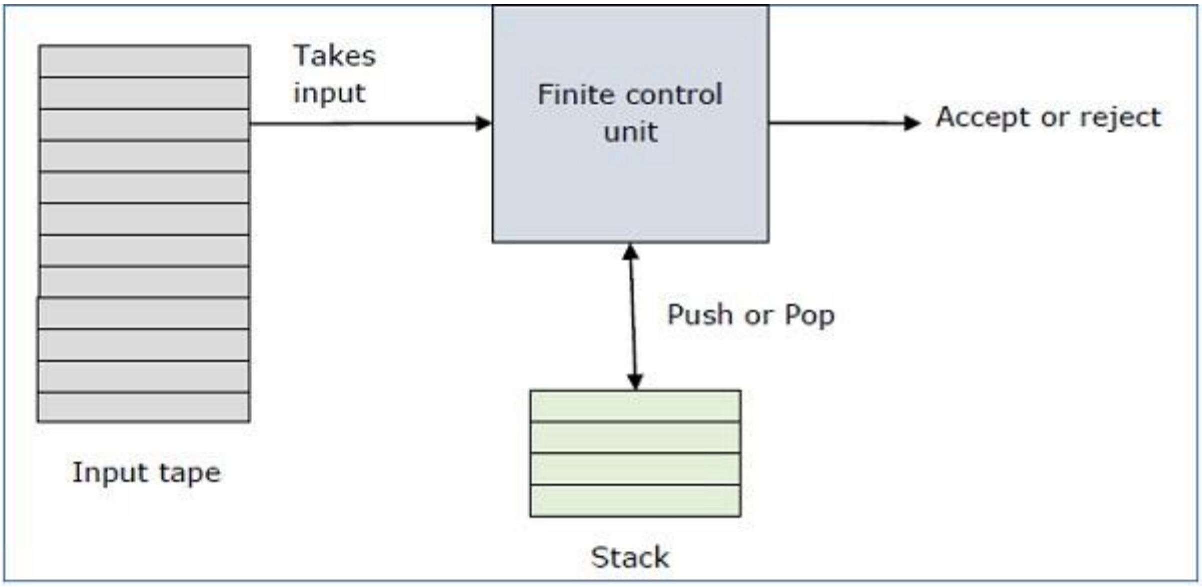
- |vxy| >=n
- |vy| # ε

- For all k>=0, the string $uv^kxy^yz \in L$

3.4 Deterministic Pushdown Automata (DPDA), Closure Properties of CFLs

What is a DPDA?

- A pushdown automaton is a way to implement a context-free grammar in a similar way we design DFA for a regular grammar.
- A DFA can remember a finite amount of information, but a PDA can remember an infinite amount of information.
- Basically a pushdown automaton is −
- **"Finite state machine" + "a stack"**
- A pushdown automaton has three components −
- an input tape,
- a control unit, and
- a stack with infinite size.
- The stack head scans the top symbol of the stack.
- A stack does two operations −
- Push − a new symbol is added at the top.
- Pop − the top symbol is read and removed



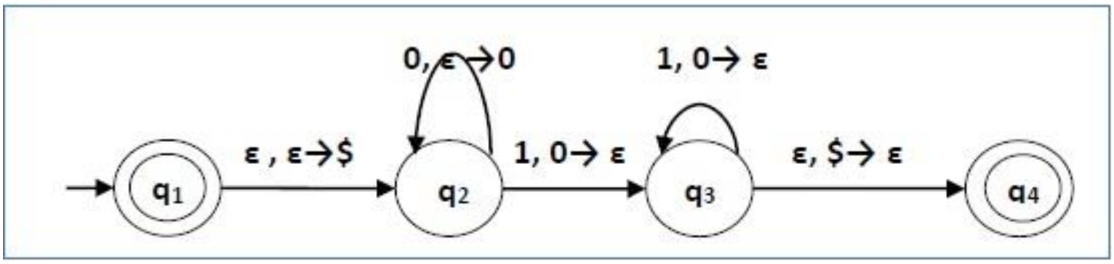
Formal Definition:

A PDA can be formally described as a 7-tuple (Q, ∑, S, δ, q_0, I, F) −

- **Q** is the finite number of states
- **∑** is input alphabet
- **S** is stack symbols
- is the transition function: Q × (∑ ∪ {ε}) × S × Q × S*
- **q_0** is the initial state (q_0 ∈ Q)
- **I** is the initial stack top symbol (I ∈ S)
- **F** is a set of accepting states (F ∈ Q)

Example:

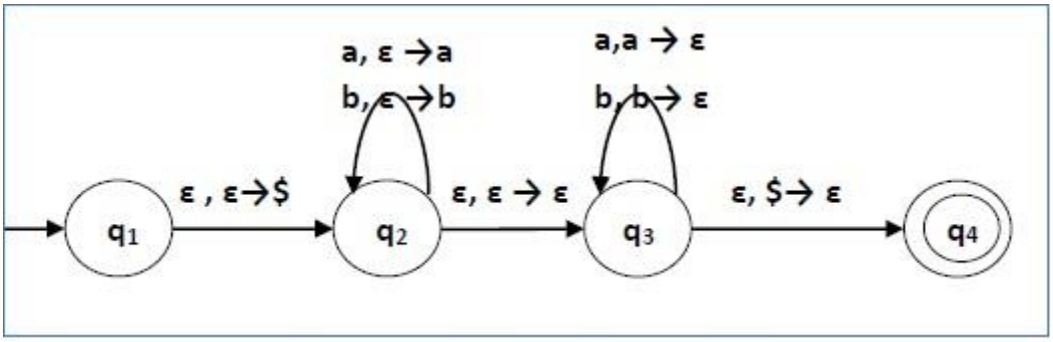
Construct a PDA that accepts L = {0n 1n | n ≥ 0}



PDA for L= $\{0^n 1^n \mid n \geq 0\}$

- Initially we put a special symbol $ into the empty stack.
- Then at state q2, if we encounter input 0 and top is Null, we push 0 into stack. This may iterate. And if we encounter input 1 and top is 0, we pop this 0.
- Then at state q3, if we encounter input 1 and top is 0, we pop this 0. This may also iterate. And if we encounter input 1 and top is 0, we pop the top element.
- If the special symbol $ is encountered at top of the stack, it is popped out and it finally goes to the accepting state q4.

Construct a PDA that accepts L = { wwR | w = (a+b)* }



PDA for L= $\{ww^R \mid w = (a+b)^*\}$

- Initially we put a special symbol $ into the empty stack.
- At state q2, the w is being read.
- In state q3, each 0 or 1 is popped when it matches the input. If any other input is given, the PDA will go to a dead state. When we reach that special symbol $, we go to the accepting state q4.

Difference Between NPDA and DPDA

Attribute	Deterministic PDA	Non-Deterministic PDA
Transition Function	Single next state for each input symbol	Multiple possible next states for each input symbol
Acceptance	Accepts input if it reaches an accepting state	Accepts input if any computation path reaches an accepting state
Complexity	Less expressive but easier to analyze	More expressive but harder to analyze
Determinism	Determined by input and current state	Non-deterministic choices made during computation

Closure Properties

Operation	CFLs (NPDA)	DCFLs (DPDA)
Union	✓	✗
Concatenation	✓	✗
Kleene Star	✓	✗
Intersection	✗	✗
Complement	✗	✓
Homomorphism	✓	✓
Inverse Homomorphism	✓	✓

3.5 Context-Sensitive Grammars (CSG) and Languages (CSL)

Context-Sensitive Languages (CSLs)

Before we enter context-sensitive languages, it's essential to understand the context-free languages (CFLs), as they form the foundation for our discussion.

What is Context-Free Grammar?

A context-free language is generated by a **context-free grammar** (CFG). In a CFG, production rules have the form: A → X, Where −

- **A** is a variable (non-terminal)
- **X** is any string of terminals or variables

- CSLs are languages generated by Context-Sensitive Grammars
- Can be recognized by Linear Bounded Automata (LBA)
- Proper superset of CFLs: CFL ⊂ CSL
- Used to describe more complex syntax rules, like type agreement in natural language
- The context-sensitive languages extend the concept of CFLs by allowing production rules to depend on the context in which variables appear.
- This seemingly small change leads to a significant increase in expressive power.
- A context-sensitive grammar has production rules of the form: $\alpha A\beta \rightarrow \alpha X\beta$, where −
- α, β are strings of terminals and/or variables (can be empty)
- A is a variable
- X is a non-empty string of terminals or variables

Context Sensitive Grammar

Proper superset of CFL.

Closed under: union, intersection, concatenation, Kleene star, complement.

Differences with Other Grammars:

Grammar Type	**Example Language**	**Automaton**
Regular	**a***	**Finite Automaton**
Context-Free (CFG)	**a^nb^n**	**Pushdown Automaton**
Context-Sensitive	**A^nb^nc^n**	**Linear Bounded Automaton**
Recursively Enumerable	**All computable languages**	**Turing Machine**

References:

1. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). Introduction to Automata Theory, Languages, and Computation (3rd ed.). Pearson Education. — Chapter 5: Context-Free Grammars and Languages.
2. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). — Chapter 6: Pushdown Automata.
3. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). — Chapter 7: Properties of Context-Free Languages.
4. Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2006). — Chapter 9: Introduction to Turing Machines (Sections on Context-Sensitive Languages and Linear Bounded Automata).
5. GeeksforGeeks. (n.d.). Chomsky Normal Form (CNF) in Theory of Computation. Retrieved from https://www.geeksforgeeks.org/chomsky-normal-form/
6. GeeksforGeeks. (n.d.). Pumping Lemma for Context-Free Languages. Retrieved from https://www.geeksforgeeks.org/pumping-lemma-for-context-free-languages/
7. TutorialsPoint. (n.d.). Pushdown Automata. Retrieved from https://www.tutorialspoint.com/automata_theory/pushdown_automata.htm

Parul® University
Vadodara, Gujarat

NAAC GRADE A++

https://paruluniversity.ac.in/