CS304: Automata and Formal Languages

Lec 18

PDA = CFG and Deterministic PDA

Rachit Nimavat

September 19, 2025

Outline

Defn: CFG

A CFG is a set of recursive rules used to generate patterns of strings. Think of it as a blueprint for a language.

Definition 1.1

A Context-Free Grammar is a 4-tuple $G = (V, T, P, S)$, where:

V: Finite set of **variables**. think: syntactic categories or placeholders. (e.g., S, E, T in previous examples)

T: Finite set of **terminals** or **tokens**. actual symbols or words of the language. (e.g., $\mathsf{a}, \mathsf{b}, \mathsf{c}, +, \star$ in previous examples)

P: Finite set of **production rules**. Rules have the form $A \to \alpha$, where $A \in V$ and $\alpha \in (V \cup T)^*$, i.e., α is a string consisting of variables and terminals. specifies how to replace variables with strings of variables and terminals. (e.g., $S \to \mathsf{a}S\mathsf{b}$ and $E \to T$) in previous examples

S: The **start symbol** ($S \in V$).

Pushdown Automaton (PDA): Formal Definition

Definition 1.2

PDA is a 7-tuple: $(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$, where:

- Q is a finite set of states
- $q_0 \in Q$ is the start state
- $F \subseteq Q$ is the set of accepting states
- Σ is the alphabet
- Γ is the stack alphabet
- $Z_0 \in \Gamma$ is the initial stack symbol
- δ is the transition function
  - $\delta : Q \times \Sigma \times \Gamma \to \mathcal{P}_{fin}(Q \times \Gamma^*)$ is the transition function
  - $\delta(q, a, x)$ is a finite set of choices for when we are in state q, read a, and the top of the stack is x: tells us which state to move to, and what stack alphabet symbols to replace x with in the stack

Example: Balanced Square Brackets

$$\mathsf{S} \to \varepsilon \,|\, \mathtt{[}\,\mathsf{S}\,\mathtt{]} \,|\, \mathsf{SS}$$

States $Q = \{q_0, q_f\}$ where q_0 is the start state and q_f is the only accepting state
$\Sigma = \{\mathtt{[}, \mathtt{]}\}$
$\Gamma = \{Z_0, \mathtt{[}\}$ where Z_0 is the initial stack symbol
Transition function δ:

Push in the stack when we see a `[`:

$\delta(q_0, \mathtt{[}, Z_0) = \{(q_0, Z_0\mathtt{[})\}$
$\delta(q_0, \mathtt{[}, \mathtt{[}) = \{(q_0, \mathtt{[[})\}$

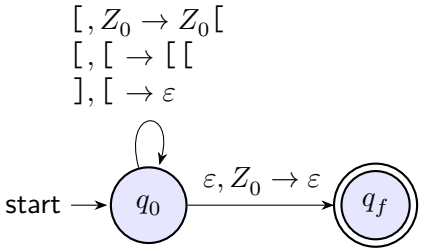
Pop from the stack when we see a `]`, if possible:

$\delta(q_0, \mathtt{]}, \mathtt{[}) = \{(q_0, \varepsilon)\}$

Accept the string if we reach the end of input and the stack has only Z_0:

$\delta(q_0, \varepsilon, Z_0) = \{(q_f, \varepsilon)\}$

Example Diagram: Balanced Square Brackets

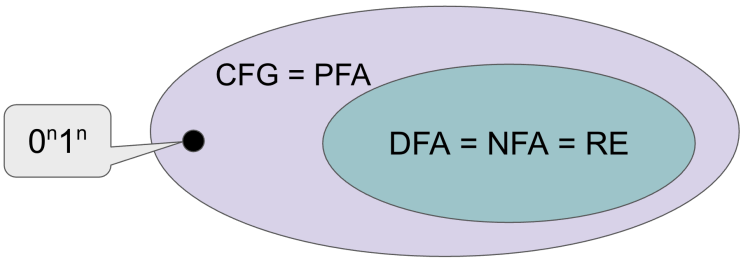


HW: There's another way to define PDA Acceptance: PDA accepts if it ends in empty stack (regardless of state). See Chapter 6.2 of ALC to see its equivalence with our accepting state definition.

Like Automata Trinity, we have another equivalence:

Theorem 2.1

A language is context-free if and only if it is recognized by a pushdown automaton.



Converting CFG to a PDA

Consider a CFG $G = (V, T, P, S)$. Construct a PDA M as follows:

States $Q = \{q_0, q_f\}$ where q_0 is the start state and q_f is the only accepting state

$\Sigma = T$

$\Gamma = V \cup T \cup \{Z_0\}$ where Z_0 is the initial stack symbol

Transition function δ:

For each variable $A \in V$,

$$\delta(q_0, \varepsilon, A) = \{(q_0, \alpha) \mid \text{for each production rule } A \to \alpha \text{ of } P\}$$

For each terminal $a \in T$,

$$\delta(q_0, a, a) = \{(q_0, \varepsilon)\}$$

For when the stack is empty (except for Z_0) and input is fully read,

$$\delta(q_0, \varepsilon, Z_0) = \{(q_f, \varepsilon)\}$$

HW: Prove that the above PDA M accepts the same language as the CFG G.

Converting PDA to a CFG

Idea: Each variable in the CFG represents a **journey** for the PDA.

Before proceeding, normalize the PDA:

- Ensure PDA accepts by empty stack
- Push or Pop only:
  - Pop exactly one symbol from the stack
  - Push exactly one symbol onto the stack (on top of the one it just read)

For each pair of states $p, q \in Q$ in the PDA and each stack-symbol $A \in \Gamma$, we create a variable $[p, A, q]$ in the CFG.

$[p, A, q]$ represents the journey with a contract: "I promise that the string I generate will take the PDA from state p to state q, and the stack will be exactly as it started, except that the symbol A which was on top has been removed."

Converting PDA to a CFG: continued

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

$V = \{[p, A, q] \mid p, q \in Q, A \in \Gamma\}$

$T = \Sigma$

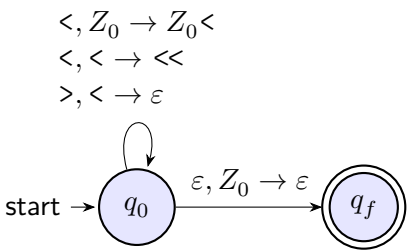
$S = [q_0, Z_0, q_f]$

Production rules P:

1. Pop Transition: For each transition of the form $(q, \varepsilon) \in \delta(p, a, X)$ add a rule: $[p, X, q] \to a$
2. Push Transition: For each transition of the form $(q, XY) \in \delta(p, a, X)$: for each $r_1, r_2 \in Q$, add a rule: $[p, X, r_2] \to a[q, Y, r_1][r_1, X, r_2]$

Bonus: Resulting CFG is in CNF!

PDA to CFG Example: Balanced Angular Brackets



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

$V = \{[q_0, Z_0, q_f], [q_0, <, q_0]\}$, $T = \{<, >\}$, $S = [q_0, Z_0, q_f]$, P:

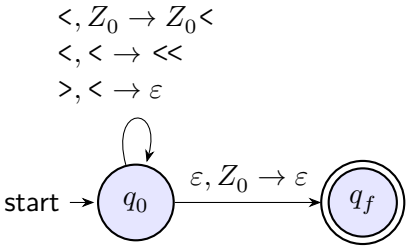
1. Pop Transition: $[q_0, <, q_0] \to >$ $\quad$ $[q_0, Z_0, q_f] \to \varepsilon$
2. Push Transition:

$$[q_0, <, q_0] \to < [q_0, <, q_0][q_0, <, q_0]$$
$$[q_0, Z_0, q_f] \to < [q_0, <, q_0][q_0, Z_0, q_f]$$

Think: $S == [q_0, Z_0, q_f]$ **and** $A == [q_0, <, q_0]$.

PDA to CFG Example: Balanced Angular Brackets

continued



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

$V = \{S, A\}$, $T = \{<, >\}$, P:

1. Pop Transition: A → > S → ε
2. Push Transition: A → <AA S → <AS

Simply, S → ε | <AS **and** A →> | <AA**.**

Deterministic PDA (DPDA)

PDA, but with deterministic transitions: PDA (with state acceptance) that is not allowed to have a choice. [1]

Regular languages are recognized by DPDAs. Every regular language has a DFA, and that is trivially a DPDA that does not use its stack.

Is DPDA with empty-stack acceptance equivalent to DPDA with state acceptance? No! Can't even recognize the language 0^*.

Is DPDA with empty-stack acceptance a strict subset of regular languages? No! Can recognize $0^n 1^n$.

Can DPDA accept languages with ambiguous CFGs? No![2] That's really good! If you design a DPDA, you know the grammar is unambiguous. Great for programming languages!

[1] See, Chapter 6.4 of ALC for defn.
[2] See, Thm 6.21 of ALC for proof.

Venn Diagram of Known (so far) Families of Languages

On Board!