Definition 5.1 A Pushdown Automaton

A *pushdown automaton* (PDA) is a 7-tuple $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$, where

Q is a finite set of states.

Σ and Γ are finite sets, the *input* and *stack* alphabets.

q_0, the initial state, is an element of Q.

Z_0, the initial stack symbol, is an element of Γ.

A, the set of accepting states, is a subset of Q.

δ, the transition function, is a function from $Q \times (\Sigma \cup \{\Lambda\}) \times \Gamma$ to the set of finite subsets of $Q \times \Gamma^*$.

Push Down Automata (PDA): (FA + Stack)

PDA is defined by $(Q, \Sigma, \delta, q_0, Z_0, F, \Gamma)$

Q : finite set of states.

Σ : input symbol.

δ : transition function.

q_0 : Initial state.

Z_0 : Bottom of the stack.

F : set of final states. ⇒ without final states also we can design PDA.

Γ : Stack alphabet.

PDA

- Deterministic PDA
  $\delta : Q \times \{\Sigma \cup \epsilon\} \times \Gamma \to Q \times \Gamma^*$
- Non-deterministic PDA
  $\delta : Q \times \{\Sigma \cup \epsilon\} \times \Gamma \to 2^{(Q \times \Gamma^*)}$

❖ **Definition:** A pushdown automata (PDA) is a seven tuple

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

where

Q - is set of finite states.

Σ - Set of input alphabets.

Γ - Set of stack alphabets.

δ - transition from $Q \times (\Sigma \cup \varepsilon) \times \Gamma$ to finite sub set of $Q \times \Gamma^*$

δ is called the transition function of M.

$q_0 \in Q$ is the start state of machine.

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

$F \subseteq Q$ is set of final states.

5.3. Instantaneous Description

The current configuration of PDA at any given instant can be described by an *instantaneous description* (in short we can call ID). An ID gives the current state of the PDA, the remaining string to be processed and the entire contents of the stack. Thus, an instantaneous description ID can be defined as shown below.

❖ **Definition:** Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. An ID (instaneous description) is defined as 3-tuple or a triple

$$(q, w, \alpha)$$

where q is the current state, *w* is the string to be processed and α is the current contents of stack. The leftmost symbol in the string α is on top of the stack and rightmost symbol in α is at the bottom of the stack.

Let the current configuration of PDA be

$$(q, aw, Z\alpha)$$

It means

- q - is the current state;
- aw - is the string to be processed;
- $Z\alpha$ - is the current contents of the stack with Z as the topmost symbol on the stack.

If the transition defined is $\delta(q, a, Z) = (p, \beta)$ then the new configuration obtained will be

$$(p, w, \beta\alpha)$$

The move from the current configuration to next configuration is given by

$$(q, aw, Z\alpha) \vdash (p, w, \beta\alpha)$$

This can be read as "the configuration $(q, aw, Z\alpha)$ derives $(p, w, \beta\alpha)$ in one move". If an arbitrary number of moves are used to move from one configuration to another configuration then the moves are denoted by the symbol $\vdash^*$ and $\vdash^+$.

Instantaneous Description (ID):

During execution, PDA goes through a sequence of configurations. Each configuration is called instantaneous description (ID) consists of (i). state (ii). input yet to be scanned and (iii) stack content & is represented by a triplet (q, x, γ) where q is the current state, x is the input yet to be read (leftmost symbol of x is the current i/p) and γ is the stack content (leftmost symbol of γ is the current top-of-stack symbol).

Move:

Each move involves a change from one ID to another.

The symbol $\vdash$ is used to represent a move.

$(p, ax, A\alpha) \vdash (q, x, \beta\alpha)$ if $\delta(p, a, A)$ includes (q, β). [The symbol a is consumed. PDA changes to state q and A is replaced by β). $\vdash^*$ represents a sequence of moves.

5.6. Deterministic and Non-deterministic PDA

In Example 5.10, we have seen that there can be multiple transitions defined from a state. The PDA defined in Example 5.10, is clearly a non-deterministic PDA. Now, let us see the difference between *deterministic* PDA and *non-deterministic* PDA.

❖ **Definition**: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The PDA is deterministic if

1. $\delta(q, a, Z)$ has only one element;
2. If $\delta(q, \varepsilon, Z)$ is not empty, then $\delta(q, a, Z)$ should be empty.

Both the conditions should be satisfied for the PDA to be deterministic. If one of the conditions fails, the PDA is non-deterministic.

In the PDAs discussed in Examples 5.3 to 5.10, let us see what are deterministic PDAs and what are non-deterministic PDAs.

The PDA should satisfy the two conditions shown in the definition to be deterministic:

1. $\delta(q, a, Z)$ has only one element: Note that in the transitions, for each $q \in Q$, $a \in \Sigma$ and $Z \in \Gamma$, there is only one component defined and the first condition is satisfied.
2. The second condition states that if $\delta(q, \varepsilon, Z)$ is not empty, then $\delta(q, a, Z)$ should be empty, i.e., If there is an ε-transition, (in this case it is $\delta(q_1, \varepsilon, Z_0)$), then there should not be any transition from the state q_1 when top of the stack is Z_0 which is true.

Since, the PDA satisfies both the conditions, the PDA is deterministic.

Definition 5.10 A Deterministic Pushdown Automaton

A pushdown automaton $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ is *deterministic* if it satisfies both of the following conditions.

1. For every $q \in Q$, every $\sigma \in \Sigma \cup \{\Lambda\}$, and every $X \in \Gamma$, the set $\delta(q, \sigma, X)$ has at most one element.
2. For every $q \in Q$, every $\sigma \in \Sigma$, and every $X \in \Gamma$, the two sets $\delta(q, \sigma, X)$ and $\delta(q, \Lambda, X)$ cannot both be nonempty.

A language L is a *deterministic context-free language* (DCFL) if there is a deterministic PDA (DPDA) accepting L.

5.4. Acceptance of a Language by PDA

There are two cases wherein a string w is accepted by a PDA:

- Get the final state from the start state.
- Get an empty stack from the start state.

In the first case, we say that the language is accepted by a **final state** and in the second case we say that the language is accepted by an **empty stack** or **null stack**. The formal definitions to accept the string by a final state and by an empty stack are defined as follows.

❖ **Definition**: Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a PDA. The language L(M) accepted by a final state is defined as

$$L(M)= \{w \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \alpha)\}$$

for some $\alpha \in \Gamma^*$, $p \in F$ and $w \in \Sigma^*$. It means that the PDA, currently in state q_0, after scanning the string *w* enters into a final state p. Once the machine is in state p, the input symbol should be ε and the contents of the stack are irrelevant. Any thing can be there on the stack. The only point to remember here is that when all the symbols in string *w* have been read and when the machine is in the final state the final contents of the stack are irrelevant.

We can also define a language N(M) accepted by an empty stack (Null stack) as

$$N(M) = \{w \mid (q_0, w, Z_0) \vdash^* (p, \varepsilon, \varepsilon)\}$$

for $w \in \Sigma^*$, q_0, $p \in Q$. It means that when the string *w* is accepted by an empty stack, the final state is irrelevant, the input should be completely read and the stack should be empty. Here the state p is not the final state, only thing is that the string *w* should be completely read and stack should be empty.

Definition 5.2 Acceptance by a PDA

If $M = (Q, \Sigma, \Gamma, q_0, Z_0, A, \delta)$ and $x \in \Sigma^*$, the string x is *accepted* by M if

$$(q_0, x, Z_0) \vdash_M^* (q, \Lambda, \alpha)$$

for some $\alpha \in \Gamma^*$ and some $q \in A$. A language $L \subseteq \Sigma^*$ is said to be accepted by M if L is precisely the set of strings accepted by M; in this case, we write $L = L(M)$. Sometimes a string accepted by M, or a language accepted by M, is said to be accepted *by final state*.

Language Accepted by a PDA:

PDA can accept a string in 2 ways.

② Acceptance by Final State:

An input w is said to be accepted by final state iff $(q_0, w, z_0) \vdash^* (p, \epsilon, \beta)$ for some p in F.

Language accepted by a PDA M by final state $L(M)$ is defined as:

$$L(M) = \{w \mid (q_0, w, z_0) \vdash^* (p, \epsilon, \beta) \text{ for some } p \text{ in } F\}$$

⑥ Acceptance by Empty Stack:

An input w is said to be accepted by a PDA by empty stack iff $(q_0, w, z_0) \vdash^* (p, \epsilon, \epsilon)$

Language accepted by a PDA M by empty stack $N(M)$ is defined as:

$$N(M) = \{w \mid (q_0, w, z_0) \vdash^* (q, \epsilon, \epsilon)\}.$$

The Pumping Lemma for Context-Free Languages (CFLs)

Theorem 6.1: Pumping Lemma for CFLs

If L is a context-free language, then there exists an integer n such that for any string $u \in L$ with $|u| \geq n$, we can write u as:

$$u = vwxyz$$

where the substrings v, w, x, y, and z satisfy:

1. $|wy| > 0$ (i.e., at least one of w or y is non-empty).
2. $|wxy| \leq n$ (i.e., the length of wxy is at most n).
3. For every $m \geq 0$, $vw^m xy^m z \in L$ (i.e., repeating w and y any number of times keeps the string in L).

Proof (Concise for 6 Marks)

1. **Context-Free Grammar & Derivation Tree:**
   - Since L is context-free, there exists a **context-free grammar** G in **Chomsky Normal Form (CNF)**.
   - A CNF derivation tree is a **binary tree**, and a tree of height h has at most 2^h leaf nodes.
   - If $|u| \geq n = 2^{p+1}$ (where p is the number of variables in G), the tree must have height $> p$.
2. **Repeating Variable:**
   - Since there are only p variables but at least $p+1$ interior nodes along some path, **a variable A must repeat** along this path.
   - Define substrings:
     - x is derived from the **lower occurrence** of A.
     - wxy is derived from the **higher occurrence** of A.
     - v and z complete the string.
3. **Derivation Steps & Pumping Condition:**
   - The derivation follows:

$$S \Rightarrow^* vAz \Rightarrow^* vwAyz \Rightarrow^* vwxyz$$

   - **Condition (1):** $|wy| > 0$ since A repeats.
   - **Condition (2):** $|wxy| \leq n$ due to the height bound.
   - **Condition (3):** Pumping works since A produces itself, meaning $vw^m xy^m z \in L$ for all $m \geq 0$.

Thus, the Pumping Lemma for CFLs is proved.

Decision Properties of CFLs:

1. emptiness prob } decidable
2. finiteness prob }
3. membership ~ } un-decidable
4. equivalance ~ }

(1) emptiness prob: checking if the given CFG generates empty lang or not.

Algorithm: 1. eliminate useless variables from the given CFG.

2. if starting symbol also includes useless variables then the grammar generates empty lang otherwise, non-empty lang.

Check whether the following grammar generates empty lang or not:

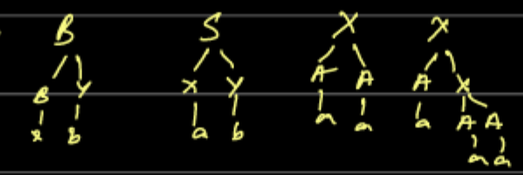
Ex: $S \to XY$ ✓

$X \to AA \mid AX$ ✓

$A \to a$ ✓

$B \to BY$ ✗ useless

$Y \to b$ ✓



B → useless variable

ans ⇒ $S \to XY$

$X \to AA \mid AX$

$A \to a$

$Y \to b$

⇒ Starting sym does not include any useless variable i.e(B).

⇒ generates non-empty lang. (If at all B may present then empty lang).

Ex: $S \to aSb \mid Sb \mid Sa$

⇒ all prods are useless as no terminations present

⇒ generates empty lang.

② finiteness prob: checking if the CFG gen ∞ or finite lang.

Algo ⇒ ① Simplify the gramma.

② Convert it to CNF form.

③ construct CNF graph.

④ if graph has loops / cycles ⇒ gramer generate ∞ lang.

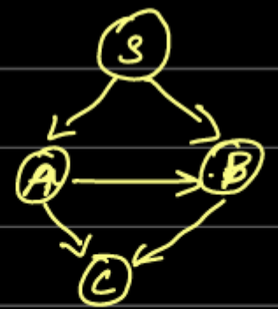
Ex: $S \to AB$

$A \to BC \mid a$

$B \to CC \mid b$

$C \to a$

* ⇒ no null production
* ⇒ no unit production
* ⇒ no useless variables
* ⇒ CNF form: $A \to BC$ or $A \to a$ (All are in CNF form already)
* ⇒ graph CNF:



(draw in order)
(only variables are considered)

* no loop ⇒ finite lang generated //