THEORY OF COMPUTATION LECTURE NOTES

UNIT 3 **(10 Lectures)**

Pushdown Automata: Definition Formal Definition of Pushdown Automata, A Graphical Notation for PDA's, Instantaneous Descriptions of a PDA,

Languages of PDA: Acceptance by Final State, Acceptance by Empty Stack, From Empty Stack to Final State, From Final State to Empty Stack

Equivalence of PDA's and CFG's: From Grammars to Pushdown Automata, From PDA's to Grammars

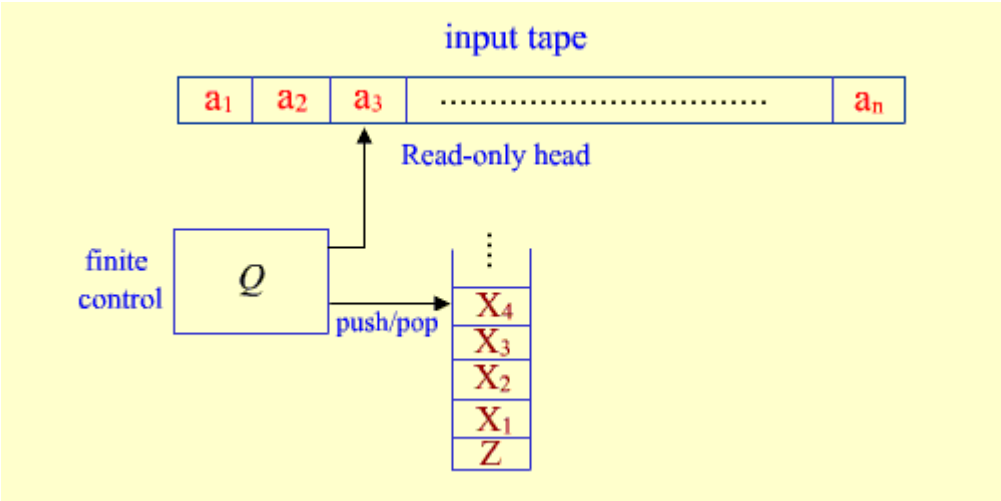
Deterministic Pushdown Automata: Definition of a Deterministic PDA, Regular Languages and Deterministic PDA's, DPDA's and Context-Free Languages, DPDA's and Ambiguous Grammars

Properties of Context-Free Languages: Normal Forms for Context-Free Grammars, The Pumping Lemma for Context-Free Languages, Closure Properties of Context-Free Languages, Decision Properties of CFL's

Push down automata:

Regular language can be charaterized as the language accepted by finite automata. Similarly, we can characterize the context-free language as the langauge accepted by a class of machines called "Pushdown Automata" (PDA). A pushdown automation is an extension of the NFA.

It is observed that FA have limited capability. (in the sense that the class of languages accepted or characterized by them is small). This is due to the "finite memory" (number of states) and "no external memory" involved with them. A PDA is simply an NFA augmented with an "external stack memory". The addition of a stack provides the PDA with a last-in, first-out memory management cpapability. This "Stack" or "pushdown store" can be used to record a potentially unbounded information. It is due to this memory management capability with the help of the stack that a PDA can overcome the memory limitations that prevents a FA to accept many interesting languages like $\{a^n b^n \mid n \geq 0\}$. Although, a PDA can store an unbounded amount of information on the stack, its access to the information on the stack is limited. It can push an element onto the top of the stack and pop off an element from the top of the stack. To read down into the stack the top elements must be popped off and are lost. Due to this limited access to the information on the stack, a PDA still has some limitations and cannot accept some other interesting languages.



As shown in figure, a PDA has three components: an input tape with read only head, a finite control and a pushdown store.

The input head is read-only and may only move from left to right, one symbol (or cell) at a time. In each step, the PDA pops the top symbol off the stack; based on this symbol, the input symbol it is currently reading, and

its present state, it can push a sequence of symbols onto the stack, move its read-only head one cell (or symbol) to the right, and enter a new state, as defined by the transition rules of the PDA.

PDA are nondeterministic, by default. That is, $\in$- transitions are also allowed in which the PDA can pop and push, and change state without reading the next input symbol or moving its read-only head. Besides this, there may be multiple options for possible next moves.

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

- Q is a finite set of states,
- Σ is a finite set of input symbols (input alphabets),
- Γ is a finite set of stack symbols (stack alphabets),
- δ is a transition function from $Q \times (\Sigma \cup \{\in\}) \times \Gamma$ to subset of $Q \times \Gamma^*$
- $q_0 \in Q$ is the start state
- $z_0 \in \Gamma$, is the initial stack symbol, and
- $F^* \subseteq Q$, is the final or accept states.

Explanation of the transition function, δ:

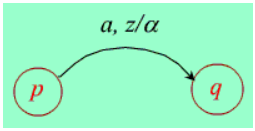
If, for any $a \in \Sigma$, $\delta(q, a, z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots, (p_k, \beta_k)\}$. This means intitutively that whenever the PDA is in state q reading input symbol a and z on top of the stack, it can nondeterministically for any i, $1 \le i \le k$

- go to state p_i
- pop z off the stack
- push β_i onto the stack (where $\beta_i \in \Gamma^*$) (The usual convention is that if $\beta_i = X_1 X_2 \cdots X_n$, then X_1 will be at the top and X_n at the bottom.)
- move read head right one cell past the current symbol a.

If $a = \in$, then $\delta(q, \in, z) = \{(p_1, \beta_1), (p_2, \beta_2), \cdots (p_k, \beta_k)\}$ means intitutively that whenver the PDA is in state q with z on the top of the stack regardless of the current input symbol, it can nondeterministically for any i, $1 \le i \le k$,

- go to state p_i
- pop z off the stack
- push β_i onto the stack, and
- leave its read-only head where it is.

State transition diagram : A PDA can also be depicted by a state transition diagram. The labels on the arcs indicate both the input and the stack operation. The transition $\delta(p,a,z)=\{(q,\alpha)\}$ for $a\in\Sigma\cup\{\in\}$, $p,q\in Q, z\in\Gamma$ and $\alpha\in\Gamma^*$ is depicted by



Final states are indicated by double circles and the start state is indicated by an arrow to it from nowhere.

Configuration or Instantaneous Description (ID) :

A configuration or an instantaneous description (ID) of PDA at any moment during its computation is an element of $Q\times\Sigma^*\times\Gamma^*$ describing the current state, the portion of the input remaining to be read (i.e. under and to the right of the read head), and the current stack contents. Only these three elements can affect the computation from that point on and, hence, are parts of the ID.

The start or inital configuartion (or ID) on input w is (q_0,w,z_0). That is, the PDA always starts in its start state, q_0, with its read head pointing to the leftmost input symbol and the stack containing only the start/initial stack symbol, z_0.

The "next move relation" one figure describes how the PDA can move from one configuration to another in one step.

Formally,

$(q,aw,z\alpha)\vdash_M(p,w,\beta\alpha)$
iff $(p,\beta)\in\delta(q,a,z)$
'a' may be $\in$ or an input symbol.

Let I, J, K be IDs of a PDA. We define we write $I\vdash_M^i K$, if ID I can become K after exactly i moves. The relations $\vdash_M^N$ and $\vdash_M^*$ define as follows

$I\vdash_N^0 K$

$I\vdash_N^{n+1} J$ if $\exists K$ such that $I\vdash_N^n K$ and $K\vdash_N^1 J$

$I\vdash_N^* J$ if $\exists\ n\geq 0$ such that $I\vdash_N^n J$.

That is, $\vdash_M^*$ is the reflexive, transitive closure of $\vdash_M$. We say that $I \vdash_N^* J$ if the ID J follows from the ID I in zero or more moves.

(Note **:** subscript M can be dropped when the particular PDA M is understood.)

Language accepted by a PDA M

There are two alternative definiton of acceptance as given below.

1. Acceptance by final state **:**

Consider the PDA $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Informally, the PDA M is said to accept its input ω by final state if it enters any final state in zero or more moves after reading its entire input, starting in the start configuration on input ω.

Formally, we define $L(M)$, the language accepted by final state to be

$$\{ \omega \in \Sigma^* \mid (q_0, \omega, Z_0) \vdash_N^* (p, \in, \beta) \text{ for some } p \in F \text{ and } \beta \in \Gamma^* \}$$

2. Acceptance by empty stack (or Null stack) **:** The PDA M accepts its input ω by empty stack if starting in the start configuration on input ω, it ever empties the stack w/o pushing anything back on after reading the entire input. Formally, we define $N(M)$, the language accepted by empty stack, to be

$$\{ \omega \in \Sigma^* \mid (q_0, \omega, z_0) \vdash_N^* (p, \in, \in) \text{ for some } p \in Q \}$$

Note that the set of final states, F is irrelevant in this case and we usually let the F to be the empty set i.e. $F = Q$.

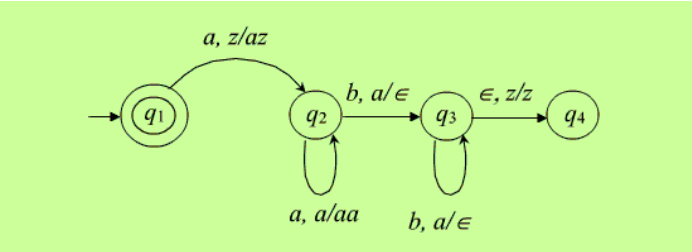
Example 1 : Here is a PDA that accepts the language $\{a^n b^n \mid n \geq 0\}$.

$M = (Q, \Sigma, \Gamma, \delta, q_1, Z, F)$
$Q = \{q_1, q_2, q_3, q_4\}$
$\Sigma = \{a, b\}$
$\Gamma = \{a, b, z\}$
$F = \{q_1, q_4\}$, and δ consists of the following transitions

1. $\delta(q_1, a, z) = \{(q_2, az)\}$
2. $\delta(q_2, a, a) = \{(q_2, aa)\}$
3. $\delta(q_2, b, a) = \{(q_3, \in)\}$

4. $\delta(q_3, b, a) = \{(q_3, \in)\}$
5. $\delta(q_3, \in, z) = \{(q_4, z)\}$

The PDA can also be described by the adjacent transition diagram.



Informally, whenever the PDA M sees an input a in the start state q_1 with the start symbol z on the top of the stack it pushes a onto the stack and changes state to q_0. (to remember that it has seen the first 'a'). On state q_0 if it sees anymore a, it simply pushes it onto the stack. Note that when M is on state q_0, the symbol on the top of the stack can only be a. On state q_0 if it sees the first b with a on the top of the stack, then it needs to start comparison of numbers of a's and b's, since all the a's at the begining of the input have already been pushed onto the stack. It start this process by popping off the a from the top of the stack and enters in state q3 (to remember that the comparison process has begun). On state q_0, it expects only b's in the input (if it sees any more a in the input thus the input will not be in the proper form of anbn). Hence there is no more on input a when it is in state q_0. On state q_0 it pops off an a from the top of the stack for every b in the input. When it sees the last b on state q3 (i.e. when the input is exaushted), then the last a from the stack will be popped off and the start symbol z is exposed. This is the only possible case when the input (i.e. on $\in$-input) the PDA M will move to state q_0 which is an accept state.
we can show the computation of the PDA on a given input using the IDs and next move relations. For example, following are the computation on two input strings.

Let the input be aabb. we start with the start configuration and proceed to the subsequent IDs using the transition function defined

$(q_1, aabb, z) \vdash (q_2, abb, az)$ (using transition 1)

$\vdash (q_2, bb, aaz)$ (using transition 2)

$\vdash (q_3, b, az)$ (using transition 3)

$\vdash (q_3, \in, z)$ (using transition 4), $\vdash (q_4, \in, z)$ (using transition 5) , q_0 is final state. Hence , accept. So the string $aabb$ is rightly accepted by M

we can show the computation of the PDA on a given input using the IDs and next move relations. For example, following are the computation on two input strings.

i) Let the input be $aabab$.

$$(q_1, aabab, z) \vdash (q_2, abab, az)$$

$$\vdash (q_2, bab, aaz)$$

$$\vdash (q_3, ab, az)$$

No further move is defined at this point.

Hence the PDA gets stuck and the string $aabab$ is not accepted.

Example 2 : We give an example of a PDA M that accepts the set of balanced strings of parentheses [] by empty stack.
The PDA M is given below.

$M = (\{q\}, \{[,]\}, \{z, [\}, \delta, q, z, \phi)$ where δ is defined as

$$\delta(q, [, z) = \{(q, [z)\}$$
$$\delta(q, [, [) = \{(q, [[)\}$$
$$\delta(q,], [) = \{(q, \in)\}$$
$$\delta(q, \in, z) = \{(q, \in)\}$$

Informally, whenever it sees a [, it will push the] onto the stack. (first two transitions), and whenever it sees a] and the top of the stack symbol is [, it will pop the symbol [off the stack. (The third transition). The fourth transition is used when the input is exhausted in order to pop z off the stack (to empty the stack) and accept. Note that there is only one state and no final state. The following is a sequence of configurations leading to the acceptance of the string [[ ] [ ]] [ ].

$$(q, [[\,][\,]][\,], z) \vdash (q, [\,][\,]][\,], [z) \vdash (q,][\,]][\,], [[z) \vdash (q, [\,]][\,], [z) \vdash (q,]][\,], [[z)$$

$$\vdash (q,]][\,], [[z) \vdash (q,][\,], [z) \vdash (q, [\,], z) \vdash (q,], [z) \vdash (q, \in, z) \vdash (q, \in, \in)$$

Equivalence of acceptance by final state and empty stack.

It turns out that the two definitions of acceptance of a language by a PDA - accpetance by final state and empty stack- are equivalent in the sense that if a language can be accepted by empty stack by some PDA, it can also be accepted by final state by some other PDA and vice versa. Hence it doesn't matter which one we use, since

each kind of machine can simulate the other.Given any arbitrary PDA M that accpets the language L by final state or empty stack, we can always construct an equivalent PDA M with a single final state that accpets exactly the same language L. The construction process of M' from M and the proof of equivalence of M & M' are given below.

There are two cases to be considered.

CASE I : PDA M accepts by final state, Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ Let *q*f be a new state not in Q. Consider the PDA $M' = \left(Q \cup \{q_f\}, \Sigma, \Gamma, \delta', q_0, z_0, \{q_f\}\right)$ where δ' as well as the following transition.

$\delta'(q, \in, X)$ contains $(q_f, X) \quad \forall\ q \in F$ and $X \in \Gamma$. It is easy to show that M and M' are equivalent i.e. $L(M) = L(M')$

Let $\varpi \in L(M)$. Then $(q_0, \varpi, z_0) \vdash_M^* (q, \in, \gamma)$ for some $q \in F$ and $\gamma \in \Gamma^*$
Then $(q_0, \varpi, z_0) \vdash_{M'}^* (q, \in, \gamma) \vdash_{M'}^* (q_f, \in, \gamma)$

Thus M' accepts ϖ

Conversely, let M' accepts ϖ i.e. $\varpi \in L(M')$, then $(q_0, \varpi, z_0) \vdash_{M'}^* (q, \in, \gamma) \vdash_{M'}^1 (q_f, \in, \gamma)$ for $q \in F$ M' inherits all other moves except the last one from M. Hence $(q_0, \varpi, z_0) \vdash_M^* (q, \in, \gamma)$ for some $q \in F$.

Thus M accepts ϖ . Informally, on any input M' simulate all the moves of M and enters in its own final state q_f whenever M enters in any one of its final status in F. Thus M' accepts a string ϖ iff M accepts it.

CASE II : PDA M accepts by empty stack.

We will construct M' from M in such a way that M' simulates M and detects when M empties its stack. M' enters its final state q_f when and only when M empties its stack.Thus M' will accept a string ϖ iff M accepts.

Let $M' = \left(Q \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{X\}, \delta', q_0', X, \{q_f\}\right)$ where $q_0',\ q_f \notin Q$ and $X \in \Gamma$ and δ' contains all the transition of δ, as well as the following two transitions.

1. $\delta'(q_0, \in, X) = \{(q_0, z_0 X)\}$ and

2. $\delta'(q, \in, X) = \{(q_f, \in)\}, \quad \forall q \in Q$

Transitions 1 causes M' to enter the initial configuration of M except that M' will have its own bottom-of-stack marker X which is below the symbols of M's stack. From this point onward M' will simulate every move of M since all the transitions of M are also in M'

If M ever empties its stack, then M' when simulating M will empty its stack except the symbol X at the bottom. At this point, M' will enter its final state q_f by using transition rule 2, thereby (correctly) accepting the input. We will prove that M and M' are equivalent.

Let M accepts ω. Then

$(q_0, \omega, z_0) \vdash_M^* (q, \in, \in)$ for some $q \in Q$. But then

$(q_0, \omega, X) \vdash_{M'}^1 (q_0, \omega, z_0 X)$ (by transition rule 1)

$\vdash_{M'}^* (q, \in, X)$ (Since M' includes all the moves of M)

$\vdash_{M'}^1 (q_f, \in, \in)$ (by transition rule 2)

Hence, M' also accepts ω. Conversely, let M' accepts ω.

Then $(q_0', \omega, X) \vdash_{M'}^1 (q_0, \omega, z_0 X) \vdash_{M'}^* (q, \in, X) \vdash_{M'}^1 (q_f, \in, \in)$ for some $q \in Q$

Every move in the sequence, $(q_0, \omega, z_0 X) \vdash_{M'}^* (q, \in, X)$ were taken from M.

Hence, M starting with its initial configuration will eventually empty its stack and accept the input i.e.

$(q_0, \omega, z_0) \vdash_M^* (q, \in, \in)$

Equivalence of PDA's and CFG's:
We will now show that pushdown automata and context-free grammars are equivalent in expressive power, that is, the language accepted by PDAs are exactly the context-free languages. To show this, we have to prove each of the following:

i) Given any arbitrary CFG G there exists some PDA M that accepts exactly the same language generated by G.
ii) Given any arbitrary PDA M there exists a CFG G that generates exactly the same language accpeted by M.

(i) CFA to PDA

We will first prove that the first part i.e. we want to show to convert a given CFG to an equivalent PDA.

Let the given CFG is $G = (N, \Sigma, P, S)$. Without loss of generality we can assume that G is in Greibach Normal Form i.e. all productions of G are of the form .

$A \to cB_1B_2 \cdots B_k$ where $c \in \Sigma \cup \{\varepsilon\}$ and $k \geq 0$.

From the given CFG G we now construct an equivalent PDA M that accepts by empty stack. Note that there is only one state in M. Let

$M = (\{q\}, \Sigma, N, \delta, q, S, \phi)$, where

- q is the only state
- Σ is the input alphabet,
- N is the stack alphabet ,
- q is the start state.
- S is the start/initial stack symbol, and δ, the transition relation is defined as follows

  For each production $A \to cB_1B_2 \cdots B_k \in P$, $(q, B_1B_2 \ldots B_k) \in \delta(q, c, A)$. We now want to show that M and G are equivalent i.e. $L(G)=N(M)$. i.e. for any $w \in \Sigma^*$. $w \in L(G)$ iff $w \in N(M)$.

If $w \in L(G)$, then by definition of $L(G)$, there must be a leftmost derivation starting with S and deriving w.

i.e. $S \overset{*}{\underset{G}{\Rightarrow}} w$

Again if $w \in N(M)$, then one sysmbol. Therefore we need to show that for any $w \in \Sigma^*$.

$S \overset{*}{\underset{G}{\Rightarrow}} w$ iff $(q, w, s) \vdash (q, \in, \in)$.

But we will prove a more general result as given in the following lemma. Replacing A by S (the startsymbol) and γ by $\in$ gives the required proof.

Lemma For any $x, y \in \Sigma^*$, $\gamma \in N^*$ and $A \in N$, $A \overset{n}{\underset{G}{\Rightarrow}} x\gamma$ via a leftmost derivative iff $(q, xy, A) \vdash_M^n (q, y, \gamma)$.

Proof : The proof is by induction on n.

Basis : $n = 0$

$A \overset{0}{\underset{G}{\Rightarrow}} x\gamma$ iff $A = x\gamma$ i.e. $x = \in$ and $\gamma = A$

iff $(q, xy, A) = (q, y, \gamma)$

iff $(q, xy, A) \vdash_M^0 (q, y, \gamma)$

Induction Step :

First, assume that $A \overset{n+1}{\underset{G}{\Rightarrow}} x\gamma$ via a leftmost derivation. Let the last production applied in their derivation is $B \to c\beta$ for some $c \in \Sigma \cup \{\in\}$ and $\beta \in N^*$.

Then, for some $\omega \in \Sigma^*$, $\alpha \in N^*$

$A \overset{n}{\underset{G}{\Rightarrow}} \omega B\alpha \overset{1}{\underset{G}{\Rightarrow}} \omega c\,\beta\alpha = x\gamma$

where $x = \omega c$ and $\gamma = \beta\alpha$

Now by the indirection hypothesis, we get,

$(q, \omega cy, A) \vdash_M^n (q, cy, B\alpha)$ .....(1)

Again by the construction of M, we get

$(q, \beta) \in \delta(q, c, B)$

so, from (1), we get

$(q, \omega cy, A) \vdash_M^n (q, cy, B\alpha) \vdash_M^1 (q, y, \beta\alpha)$

since $x = \omega c$ and $\gamma = \beta\alpha$, we get $(q, xy, A) \vdash_M^{n+1} (q, y, \gamma)$

That is, if $A \overset{n+1}{\underset{G}{\Rightarrow}} x\gamma$, then $(q, xy, A) \vdash_M^{n+1} (q, y, \gamma)$. Conversely, assume that $(q, xy, A) \vdash_M^{n+1} (q, y, \gamma)$ and let

$\delta(q,c,B)=(q,\beta)$ be the transition used in the last move. Then for some $\omega\in\Sigma^*$, $c\in\Sigma\cup\{\in\}$ and $\alpha \in \Gamma^*$

$(q,\omega cy,A)\vdash_M^n(q,cy,B\alpha)\vdash_M^1(q,y,\beta\alpha)$ where $x=\omega c$ and $\gamma=\beta\alpha$.

Now, by the induction hypothesis, we get

$A\overset{n}{\underset{G}{\Rightarrow}}\omega B\alpha$ via a leftmost derivation.

Again, by the construction of M, $B\to c\beta$ must be a production of G. [Since $(q,\beta)\in\delta(q,c,B)$]. Applying the production to the sentential form $\omega B\alpha$ we get

$A\overset{n}{\underset{G}{\Rightarrow}}\omega B\alpha\overset{1}{\underset{G}{\Rightarrow}}\omega c\beta\alpha=x\gamma$

i.e. $A\overset{n+1}{\underset{G}{\Rightarrow}}x\gamma$

via a leftmost derivation.

Hence the proof.

Example : Consider the CFG G in GNF

S $\to$ aAB
A $\to$ a / aA
B $\to$ a / bB

The one state PDA M equivalent to G is shown below. For convenience, a production of G and the corresponding transition in M are marked by the same encircled number.

(1) S $\to$ aAB
(2) A $\to$ a
(3) A $\to$ aA
(4) B $\to$ a
(5) B $\to$ bB

$M=(\{q\},\{a,b\},\{S,A,B\},\delta,q,S,\Sigma)$. We have used the same construction discussed earlier

Some Useful Explanations :
Consider the moves of M on input *aaaba* leading to acceptance of the string.
Steps

1. $(q, aaaba, s) \overset{(1)}{\underset{M}{\vdash}} (q, aaba, AB)$

2. $\overset{(2)}{\underset{M}{\vdash}} (q, aba, AB)$

3. $\overset{(3)}{\underset{M}{\vdash}} (q, ba, B)$

4. $\overset{(4)}{\underset{M}{\vdash}} (q, a, B)$

5. $\overset{(5)}{\underset{M}{\vdash}} (q, \in, \in)$ Accept by empty stack.

Note : encircled numbers here shows the transitions rule applied at every step.
Now consider the derivation of the same string under grammar G. Once again, the production used at every step is shown with encircled number.

$$S \overset{(1)}{\underset{G}{\Rightarrow}} aAB \overset{(3)}{\underset{G}{\Rightarrow}} aaAB \overset{(2)}{\underset{G}{\Rightarrow}} aaaB \overset{(5)}{\underset{G}{\Rightarrow}} aaabB \overset{(4)}{\underset{G}{\Rightarrow}} aaaba$$

Steps $\rightarrow$ 1 $\rightarrow$ 2 $\rightarrow$ 3 $\rightarrow$ 4 $\rightarrow$ 5

Observations:

- There is an one-to-one correspondence of the sequence of moves of the PDA M and the derivation sequence under the CFG G for the same input string in the sense that - number of steps in both the cases are same and transition rule corresponding to the same production is used at every step (as shown by encircled number).
- considering the moves of the PDA and derivation under G together, it is also observed that at every step the input read so far and the stack content together is exactly identical to the corresponding sentential form i.e.
  <what is Read><stack> = <sentential form>

Say, at step 2, Read so far = a
stack = AB

Sentential form = aAB From this property we claim that $(q, x, S) \overset{*}{\underset{M}{\vdash}} (q, \in, \alpha)$ iff $S \overset{*}{\underset{G}{\Rightarrow}} x\alpha$. If the claim is true, then apply with $\alpha = \in$ and we get $(q, x, S) \overset{*}{\underset{M}{\vdash}} (q, \in, \in)$ iff $S \overset{*}{\underset{G}{\Rightarrow}} x$ or $x \in N(M)$ iff $x \in L(G)$ (by definition)

Thus $N(M) = L(G)$ as desired. Note that we have already proved a more general version of the claim

PDA and CFG:

We now want to show that for every PDA M that accpets by empty stack, there is a CFG G such that $L(G) = N(M)$

we first see whether the "reverse of the construction" that was used in part (i) can be used here to construct an equivalent CFG from any PDA M.

It can be show that this reverse construction works only for single state PDAs.

- That is, for every one-state PDA M there is CFG G such that $L(G) = N(M)$. For every move of the PDA M $(q,\ B_1B_2\cdots B_K)\in\delta(q,\ c,\ A)$ we introduce a production $A\ \rightarrow cB_1B_2\cdots B_K$ in the grammar $G=(N,\ \Sigma,\ P,\ S)$ where $N = T$ and $S = z_0$.

we can now apply the proof in part (i) in the reverse direction to show that $L(G) = N(M)$.

But the reverse construction does not work for PDAs with more than one state. For example, consider the PDA M produced here to accept the langauge $\left\{a^nba^n\ |\ n\geq 1\right\}$

$$M=\left(\{p,\ q\},\{a,\ b\},\{z_0,\ A\},\delta,p,z_0,\phi\right)$$

Now let us construct CFG $G=(N,\ \Sigma,\ P,\ S)$ using the "reverse" construction.

(Note $N=\{z_0,\ A\},\ S=z_0$).

Transitions in M	Corresponding Production in G
$a, z_0/A$	$z_0\rightarrow aA$
$a,\ A/AA$	$A\rightarrow aAA$
$b,\ A/A$	$A\rightarrow bA$
$a,\ A/\in$	$A\rightarrow a$

We can drive strings like aabaa which is in the language.

$$s=z_0\underset{G}{\Rightarrow}aA\underset{G}{\Rightarrow}aaAA\underset{G}{\Rightarrow}aabAA\underset{G}{\Rightarrow}aabaA\underset{G}{\Rightarrow}aabaa$$

But under this grammar we can also derive some strings which are not in the language. e.g

$$s=z_0\Rightarrow aA\Rightarrow abA\Rightarrow abaAA\Rightarrow abaaA\Rightarrow abaaa$$

and $s=z_0\Rightarrow aA\Rightarrow aa$. But $aa, abaa\notin L(M)$

Therefore, to complete the proof of part (ii) we need to prove the following claim also.

Claim: For every PDA M there is some one-state PDA M' such that $N(M)=N(M')$.

It is quite possible to prove the above claim. But here we will adopt a different approach. We start with any arbitrary PDA M that accepts by empty stack and directly construct an equivalent CFG G.

PDA to CFG

We want to construct a CFG G to simulate any arbitrary PDA M with one or more states. Without loss of generality we can assume that the PDA M accepts by empty stack.

The idea is to use nonterminal of the form <PAq> whenever PDA M in state P with A on top of the stack goes to state q_0. That is, for example, for a given transition of the PDA corresponding production in the grammar as shown below,

And, we would like to show, in general, that $\langle pAq \rangle \overset{*}{\underset{G}{\Rightarrow}} w$ iff the PDA M, when started from state P with A on the top of the stack will finish processing w, arrive at state q and remove A from the stack.

we are now ready to give the construction of an equivalent CFG G from a given PDA M. we need to introduce two kinds of producitons in the grammar as given below. The reason for introduction of the first kind of production will be justified at a later point. Introduction of the second type of production has been justified in the above discussion.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, z_0, \phi)$ be a PDA. We construct from M a equivalent CFG $G = (N, \Sigma, P, S)$

Where

- N is the set of nonterminals of the form <PAq> for $p, q \in Q$ and $A \in \Gamma$ and P contains the follwoing two kind of production
  1. $S \rightarrow \langle q_0 z_0 q \rangle \quad \forall\, q \in Q$
  2. If $(q_1, B_1 B_2 \cdots B_n) \in \delta(q, a, A)$, then for every choice of the sequence $q_2, q_3, \cdots, q_{n+1}$, $q_i \in Q$, $2 \le i \le n+1$.

Include the follwoing production

$$\langle q_A\, q_{n+1} \rangle \rightarrow a \langle q_1 B_1 q_2 \rangle \langle q_2 B_2 q_3 \rangle \cdots \langle q_n B_n q_{n+1} \rangle$$

If $n = 0$, then the production is $\langle q_A\, q_1 \rangle \rightarrow a$ .For the whole exercise to be meaningful we want $\langle q_A\, q_{n+1} \rangle \overset{*}{\underset{G}{\Rightarrow}} w$ means there is a sequence of transitions (for PDA M), starting in state q, ending in q_{n+1}, during which the PDA M consumes the input string w and removes A from the stack (and, of course, all other symbols pushed onto stack in A's place, and so on.)

That is we want to claim that

$$\langle pAq \rangle \overset{*}{\underset{G}{\Rightarrow}} w \text{ iff } (p, w, A) \vdash (q, \in, \in)$$

If this claim is true, then let $p = q_0, A = z_0$ to get $\langle q_0\ z_0\ q \rangle \overset{*}{\underset{G}{\Rightarrow}} w$ iff $(q_0, w, z_0) \vdash (q, \in, \in)$ for some $q \in Q$. But for all $q \in Q$ we have $S \rightarrow \langle q_0 z_0 q \rangle$ as production in G. Therefore,

$S \overset{1}{\underset{G}{\Rightarrow}} \langle q_0 z_0 q \rangle \overset{*}{\underset{G}{\Rightarrow}}$ iff $(q_0, w, z_0) \vdash (q, \in, \in)$ i.e. $S \overset{*}{\underset{G}{\Rightarrow}} w$ iff PDA M accepts w by empty stack or $L(G) = N(M)$

Now, to show that the above construction of CFG G from any PDA M works, we need to prove the proposed claim.

Note: At this point, the justification for introduction of the first type of production (of the form $S \to \langle q_0 z_0 q \rangle$) in the CFG G, is quite clear. This helps use deriving a string from the start symbol of the grammar.

Proof : Of the claim $\langle PAq \rangle \overset{*}{\underset{G}{\Rightarrow}} w$ iff $(P, w, A) \vdash (q, \in, \in)$ for some $w \in \Sigma^*$, $A \in \Gamma$ and $p, q \in Q$
The proof is by induction on the number of steps in a derivation of G (which of course is equal to the number of moves taken by M). Let the number of steps taken is n.

The proof consists of two parts: ' *if* ' part and ' *only if* ' part. First, consider the ' *if* ' part

If $(P, w, A) \vdash (q, \in, \in)$ then $\langle PAq \rangle \overset{*}{\underset{G}{\Rightarrow}} w$.

Basis is $n = 1$
Then $(P, w, A)\ (q, \in, \in)$. In this case, it is clear that $w \in \Sigma \cup \{\in'\}$. Hence, by construction $\langle PAq \rangle \to w$ is a production of G.

Then

Inductive Hypothesis :

$$\forall i < n\ (P, w, A) \vdash (q, \in, \in) \Rightarrow \langle PAq \rangle \overset{*}{\underset{G}{\Rightarrow}} w$$

Inductive Step : $(P, w, A) \vdash (q, \in, \in)$

For $n > 1$, let $w = ax$ for some $a \in \Sigma \cup \{\in\}$ and $x \in \Sigma^*$ consider the first move of the PDA M which uses the general transition $(q_1, B_1 B_2 \cdots B_n) \in \delta(p, a, A)(p, w, A)$ =
$(p, ax, A) \vdash (q_1, x, B_1 B_2 \cdots B_n) \vdash (q, \in, \in)$. Now M must remove $B_1 B_2 \cdots B_n$ from stack while consuming x in the remaining n-1 moves.

Let $x = x_1 x_2 \cdots x_n$, where $x_1 x_2 \cdots x_n$ is the prefix of x that M has consumed when B_{i+1} first appears at top of the stack. Then there must exist a sequence of states in M (as per construction) $q_2, q_3, \cdots q_n, q_{n+1}$ (with $q_{n+1} = p$), such that

$(p, ax, A) \vdash (q_1,\ x,\ B_1B_2\cdots B_n) = (q_1,\ x_1x_2\cdots x_n,\ B_1B_2\cdots B_n)$

$(q_2,\ x_2x_3\cdots x_n,\ B_2B_3\cdots B_n)$ [This step implies $(q_1,\ x_1,\ B_1) \vdash (q_2, \in, \in)$]

$(q_3,\ x_3x_4\cdots x_n,\ B_3B_4\cdots B_n)$ [This step implies $(q_2,\ x_2,\ B_2) \vdash (q_3, \in, \in)$]

...

$\vdash (q_n,\ x_n, B_n)$

$\vdash (q_{n+1}, \in, \in) = (q, \in, \in)$

[Note: Each step takes less than or equal to n -1 moves because the total number of moves required assumed to be n-1.]

That is, in general

$(q_i,\ x_i,\ B_i) \vdash (q_{i+1}, \in, \in)$, $1 \le i \le n+1$.

So, applying inductive hypothesis we get

$\langle q_iB_iq_{i+1}\rangle \overset{*}{\underset{G}{\Rightarrow}} x_i$, $1 \le i \le n+1$. But corresponding to the original move

$(p, w, A) = (p, ax, A) \vdash (q_1,\ x,\ B_1B_2\cdots B_n)$ in M we have added the following production in G.

We can show the computation of the PDA on a given input using the IDs and next move relations. For example, following are the computation on two input strings.

i) Let the input be *aabb*. we start with the start configuration and proceed to the subsequent IDs using the transition function defined

$(q_1, aabb, z) \vdash (q_2, abb, az)$ (using transition 1) , $\vdash (q_2, bb, aaz)$ (using transition 2)

$\vdash (q_3, b, az)$ (using transition 3), $(q_3, \in, z)$ (using transition 4)

$\vdash (q_4, \in, z)$ (using transition 5) , q_0 is final state. Hence, accept.

So the string *aabb* is rightly accepted by M.

we can show the computation of the PDA on a given input using the IDs and next move relations. For example, following are the computation on two input strings.

i) Let the input be *aabab*.

$(q_1, aabab, z) \vdash (q_2, abab, az)$

$\vdash (q_2, bab, aaz)$

$\vdash (q_3, ab, az)$

No further move is defined at this point.

Hence the PDA gets stuck and the string *aabab* is not accepted.

The following is a sequence of configurations leading to the acceptance of the string [[ ] [ ]] [ ].

$(q,[[\,][\,]][\,],z) \vdash (q,[\,][\,]][\,],[z) \vdash (q,\,][\,]][\,],[[z)$

$\vdash (q,[\,]][\,],[z) \vdash (q,\,]][\,],[[z) \vdash (q,\,][\,],[\,z)$

$\vdash (q,[\,],z) \vdash (q,\,],[z) \vdash (q,\in,z) \vdash (q,\in,\in)$

Equivalence of acceptance by final state and empty stack.

It turns out that the two definitions of acceptance of a language by a PDA - accpetance by final state and empty stack- are equivalent in the sense that if a language can be accepted by empty stack by some PDA, it can also be accepted by final state by some other PDA and vice versa. Hence it doesn't matter which one we use, since each kind of machine can simulate the other.Given any arbitrary PDA M that accpets the language L by final state or empty stack, we can always construct an equivalent PDA M with a single final state that accpets exactly the same language L. The construction process of M' from M and the proof of equivalence of M & M' are given below

There are two cases to be considered.

CASE 1 : PDA M accepts by final state, Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$. Let q_f be a new state not in Q. Consider the PDA $M' = \left(Q \cup \{q_f\}, \Sigma, \Gamma, \delta', q_0, z_0, \{q_f\}\right)$ where δ' as well as the following transition.

$\delta'(q,\in,X)$ contains $(q_f, X) \quad \forall\ q \in F$ and $X \in \Gamma$. It is easy to show that M and M' are equivalent i.e.

$L(M) = L(M')$.

Let $w \in L(M)$. Then $(q_0, w, z_0) \vdash_M^* (q,\in,\gamma)$ for some $q \in F$ and $\Gamma \in \Sigma^*$

Then $(q_0,\ w,\ z_0) \vdash_{M'}^* (q,\in,\gamma) \vdash_{M'}^* (q_f,\in,\gamma)$.

Thus M' accepts w.

Conversely, let M' accepts ω i.e. $\omega \in L(M')$, then $(q_0, \omega, z_0) \vdash^*_{M'} (q, \in, \gamma) \vdash^1_{M'} (q_f, \in, \gamma)$ for some $q \in F$. M' inherits all other moves except the last one from M. Hence $(q_0, \omega, z_0) \vdash^*_{M} (q, \in, \gamma)$ for some $q \in F$.

Thus M accepts ω. Informally, on any input M' simulate all the moves of M and enters in its own final state q_f whenever M enters in any one of its final status in F. Thus M' accepts a string ω iff M accepts it.

CASE 2 : PDA M accepts by empty stack.

we will construct M' from M in such a way that M' simulates M and detects when M empties its stack. M' enters its final state q_f when and only when M empties its stack.Thus M' will accept a string ω iff M accepts.

Let $M' = (Q \cup \{q_0', q_f\}, \Sigma, \Gamma \cup \{X\}, \delta', q_0', X, \{q_f\})$ where $q_0',\ q_f \notin Q$ and $X \in \Gamma$ and δ' contains all the transition of δ, as well as the following two transitions.

1. $\delta'(q_0, \in, X) = \{(q_0, z_0 X)\}$ and

2. $\delta'(q, \in, X) = \{(q_f, \in)\}, \quad \forall q \in Q$

Transitions 1 causes M' to enter the initial configuration of M except that M' will have its own bottom-of-stack marker X which is below the symbols of M's stack. From this point onward M' will simulate every move of M since all the transitions of M are also in M'.

If M ever empties its stack, then M' when simulating M will empty its stack except the symbol X at the bottom. At this point M', will enter its final state q_f by using transition rule 2, thereby (correctly) accepting the input. we will prove that M and M' are equivalent.

Let M accepts ω.

Then

$(q_0, \omega, z_0) \vdash (q, \in, \in)$ for some $q \in Q$. But then,

$(q_0, \omega, X) \vdash^1_{M'} (q_0, \omega, z_0 X)$ (by transition rule 1)

$\vdash^*_{M'} (q, \in, X)$ (since M' include all the moves of M)

$\vdash_{M'}^{1} (q_f, \in, \in)$ (by transition rule 2)

Hence, M' also accepts ϖ .Conversely, let M' accepts ϖ .

Then $(q_0', \varpi, X) \vdash_{M'}^{1} (q_0, \varpi, z_0 X) \vdash_{M'}^{*} (q, \in, X) \vdash_{M'}^{1} (q_f, \in, \in)$ for some Q .

Every move in the sequence

$(q_0, \varpi, z_0 X) \vdash_{M'}^{*} (q, \in, X)$ were taken from M.

Hence, M starting with its initial configuration will eventually empty its stack and accept the input i.e.

$(q_0, \varpi, z_0) \vdash_{M}^{*} (q, \in, \in)$.

Deterministic PDA:

we define a PDA $P = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ to be *deterministic* (a deterministic PDA or DPDA), if and only if the following conditions are met:

1. $\delta(q, a, X)$ has at most one member for any q in Q, a in Σ or $a = \epsilon$, and X in Γ.

2. If $\delta(q, a, X)$ is nonempty, for some a in Σ, then $\delta(q, \epsilon, X)$ must be empty.

Regular Languages and DPDA's The DPDA's accepts a class of languages that is in between the regular languages and CFL's.

Theorem 6.17: If L is a regular language, then $L = L(P)$ for some DPDA P.

PROOF: Essentially, a DPDA can simulate a deterministic finite automaton. The PDA keeps some stack symbol Z_0 on its stack, because a PDA has to have a stack, but really the PDA ignores its state. Formally, let $A = (Q, \Sigma, \delta_A, q_0, F)$ be a DFA. Construct DPDA

$$P = (Q, \Sigma, \{Z_0\}, \delta_P, q_0, Z_0, F)$$

by defining $\delta_P(q, a, Z_0) = \{(p, Z_0)\}$ for all states p and q in Q, such that $\delta_A(q, a) = p$.

We claim that $(q_0, w, Z_0) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, Z_0)$ if and only if $\hat{\delta}_A(q_0, w) = p$. That is, P simulates A using its state. The proofs in both directions are easy inductions on $|w|$, and we leave them for the reader to complete. Since both A and P accept by entering one of the states of F, we conclude that their languages are the same. □

Deterministic Pushdown Automata (DPDA) and Deterministic Context-free Languages (DCFLs)

Pushdown automata that we have already defined and discussed are nondeterministic by default, that is , there may be two or more moves involving the same combinations of state, input symbol, and top of the stock, and again, for some state and top of the stock the machine may either read and input symbol or make an $\in$- transition (without consuming any input).

In deterministic *PDA* , there is never a choice of move in any situation. This is handled by preventing the above mentioned two cases as described in the definition below.

Defnition : Let $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ be a *PDA* . Then *M* is deterministic if and only if both the following conditions are satisfied.

1. $\delta(q, a, X)$ has at most one element for any $q \in Q, a \in \Sigma \cup \{\in\}$, and $X \in \Gamma$ (this condition prevents multiple choice f any combination of q, a and X)
2. If $\delta(q, \in, X) \neq \phi$ and $\delta(q, a, X) = \phi$ for every $a \in \Sigma$

(This condition prevents the possibility of a choice between a move with or without an input symbol).

Empty Production Removal

The productions of context-free grammars can be coerced into a variety of forms without affecting the expressive power of the grammars. If the empty string does not belong to a language, then there is a way to eliminate the productions of the form $A \rightarrow \lambda$ from the grammar.

If the empty string belongs to a language, then we can eliminate λ from all productions save for the single production $S \rightarrow \lambda$. In this case we can also eliminate any occurrences of S from the right-hand side of productions.

Procedure to find CFG with out empty Productions

Step (i): For all productions $A \rightarrow \lambda$, put A into V_N.

Step (ii): Repeat the following steps until no further variables are added to V_N.
For all productions

$$B \rightarrow A_1 A_2 \ \ldots\ldots \ A_n.$$

Step (i): For all productions $A \rightarrow \lambda$, put A into V_N.

Step (ii): Repeat the following steps until no further variables are added to V_N.
For all productions

$$B \rightarrow A_1 A_2 \ \ldots\ldots \ A_n.$$

where $A_1, A_2, A_3, \ldots\ldots, A_n$ are in V_N, put B into V_N.

To find $\hat{P}$, let us consider all productions in P of the form

$$A \to x_1 x_2 \ \ldots\ldots\ x_m, m \geq 1$$

for each $x_i \in V \cup T$.

Unit production removal

Any production of a CFG of the form

$$A \to B$$

where $A, B \in V$ is called a "Unit-production". Having variable one on either side of a production is sometimes undesirable.

"Substitution Rule" is made use of in removing the unit-productions.

Given $G = (V, T, S, P)$, a CFG with no λ-productions, there exists a CFG $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ that does not have any unit-productions and that is equivalent to G.

Let us illustrate the procedure to remove unit-production through example 2.4.6.

Procedure to remove the unit productions:

Find all variables B, for each A such that

$$A \overset{*}{\Rightarrow} B$$

This is done by sketching a "depending graph" with an edge (C, D) whenever the grammar has unit-production $C \to D$, then $A \overset{*}{\Rightarrow} B$ holds whenever there is a walk between A and B.

The new grammar $\hat{G}$, equivalent to G is obtained by letting into $\hat{P}$ all non-unit productions of P.

Then for all A and B satisfying $A \overset{*}{\Rightarrow} B$, we add to $\hat{P}$

$$A \to y_1 \mid y_2 \mid \ldots\ldots \mid y_n$$

where $B \to y_1 \mid y_2 \mid \ldots\ldots \mid y_n$ is the set of all rules in $\hat{P}$ with B on the left.

Left Recursion Removal

A variable A is left-recursive if it occurs in a production of the form

$$A \to Ax$$

for any $x \in (V \cup T)^*$.

A grammar is left-recursive if it contains at least one left-recursive variable.

Every content-free language can be represented by a grammar that is not left-recursive.

NORMAL FORMS

Two kinds of normal forms viz., Chomsky Normal Form and Greibach Normal Form (GNF) are considered here.

Chomsky Normal Form (CNF)

Any context-free language L without any λ-production is generated by a grammar is which productions are of the form $A \to BC$ or $A \to a$, where A, B $\in V_N$, and a $\in V_T$.

Procedure to find Equivalent Grammar in CNF

(i) Eliminate the unit productions, and λ-productions if any,

(ii) Eliminate the terminals on the right hand side of length two or more.

(iii) Restrict the number of variables on the right hand side of productions to two.

Proof:

For Step (i): Apply the following theorem: "Every context free language can be generated by a grammar with no useless symbols and no unit productions".

At the end of this step the RHS of any production has a single terminal or two or more symbols.

Let us assume the equivalent resulting grammar as $G = (V_N, V_T, P, S)$.

For Step (ii): Consider any production of the form

$$A \to y_1 y_2 \ \ldots\ldots\ y_m, \quad m \geq 2.$$

If y_1 is a terminal, say 'a', then introduce a new variable B_a and a production

$$B_a \to a$$

Repeat this for every terminal on RHS.

Let P' be the set of productions in P together with the new productions

$B_a \to a$. Let V_N' be the set of variables in V_N together with $B_a's$ introduced for every terminal on RHS.

The resulting grammar $G_1 = (V_N', V_T, P', S)$ is equivalent to G and every production in P' has either a single terminal or two or more variables.

For step (iii): Consider $A \to B_1 B_2 \ldots\ldots B_m$

where B_i's are variables and $m \geq 3$.

If $m = 2$, then $A \to B_1, B_2$ is in proper form.

The production $A \to B_1 B_2 \ldots\ldots B_m$ is replaced by new productions

$$\begin{aligned} A &\to B_1 D_1, \\ D_1 &\to B_2 D_2, \\ &\ldots\ldots \ \ldots\ldots \\ &\ldots\ldots \ \ldots\ldots \\ D_{m-2} &\to B_{m-1} B_m \end{aligned}$$

where $D_i'S$ are new variables.

The grammar thus obtained is G_2, which is in CNF.

Example

Obtain a grammar in Chomsky Normal Form (CNF) equivalent to the grammar *G* with productions *P* given

$S \to aAbB$

$A \to aA \mid a$

$B \to bB \mid b.$

Solution

(i) There are no unit productions in the given set of P.

(ii) Amongst the given productions, we have

$$A \to a,$$
$$B \to b$$

which are in proper form.

For $S \to aAbB$, we have

$$S \to B_a AB_b B,$$
$$B_a \to a$$
$$B_b \to b.$$

For $A \to aA$, we have

$$A \to B_a A$$

For $B \to bB$, we have

$$B \to B_b B.$$

(iii) In P' above, we have only

$$S \to B_a AB_b B$$

not in proper form.

Hence we assume new variables D_1 and D_2 and the productions

$$S \to B_a D_1$$
$$D_1 \to AD_2$$
$$D_2 \to B_b B$$

Therefore the grammar in Chomsky Normal Form (CNF) is G_2 with the productions given by

$$S \to B_a D_1,$$
$$D_1 \to AD_2,$$
$$D_2 \to B_b B,$$
$$A \to B_a A,$$
$$B \to B_b B,$$
$$B_a \to a,$$
$$B_b \to b,$$
$$A \to a,$$

and

$$B \to b.$$

Pumping Lemma for CFG

A "Pumping Lemma" is a theorem used to show that, if certain strings belong to a language, then certain other strings must also belong to the language. Let us discuss a Pumping Lemma for CFL. We will show that , if *L* is a context-free language, then strings of *L* that are at least '*m*' symbols long can be "pumped" to produce additional strings in *L*. The value of '*m*' depends on the particular language. Let *L* be an infinite context-free language. Then there is some positive integer '*m*' such that, if *S* is a string of *L* of Length at least '*m*', then

(i) S = uvwxy (for some u, v, w, x, y)

(ii) $|vwx| \leq m$

(iii) $|vx| \geq 1$

(iv) $uv^i wx^i y \in L$.

for all non-negative values of *i*.

It should be understood that

(i) If *S* is sufficiently long string, then there are two substrings, *v* and *x*, somewhere in *S*. There is stuff (*u*) before *v*, stuff (*w*) between *v* and *x*, and stuff (*y*), after *x*.

(ii) The stuff between *v* and *x* won't be too long, because | *vwx* | can't be larger than *m*.

(iii) Substrings *v* and *x* won't both be empty, though either one could be.

(iv) If we duplicate substring *v*, some number (i) of times, and duplicate *x* the same number of times, the resultant string will also be in *L*.

Definitions

A variable is useful if it occurs in the derivation of some string. This requires that

(a) the variable occurs in some sentential form (you can get to the variable if you start from *S*), and

(b) a string of terminals can be derived from the sentential form (the variable is not a "dead end").

A variable is "recursive" if it can generate a string containing itself. For example, variable *A* is recursive if

$$S \overset{*}{\Rightarrow} uAy$$

for some values of u and y.

A recursive variable A can be either

(i) "Directly Recursive", i.e., there is a production

$$A \rightarrow x_1 A x_2$$

for some strings $x_1, x_2 \in (T \cup V)^*$, or

(ii) "Indirectly Recursive", i.e., there are variables x_i and productions

$$A \rightarrow X_1 \ldots$$
$$X_1 \rightarrow \ldots X_2 \ldots$$
$$X_2 \rightarrow \ldots X_3 \ldots$$
$$X_N \rightarrow \ldots A \ldots$$

Proof of Pumping Lemma

(a) Suppose we have a CFL given by *L*. Then there is some context-free Grammar *G* that generates *L*. Suppose

(i) *L* is infinite, hence there is no proper upper bound on the length of strings belonging to *L*.

(ii) *L* does not contain l.

(iii) *G* has no productions or l-productions.

There are only a finite number of variables in a grammar and the productions for each variable have finite lengths. The only way that a grammar can generate arbitrarily long strings is if one or more variables is both useful and recursive. Suppose no variable is recursive. Since the start symbol is non recursive, it must be defined only in terms of terminals and other variables. Then since those variables are non recursive, they have to be defined in terms of terminals and still other variables and so on.

After a while we run out of "other variables" while the generated string is still finite. Therefore there is an upper bond on the length of the string which can be generated from the start symbol. This contradicts our statement that the language is finite.

Hence, our assumption that no variable is recursive must be incorrect.

(b) Let us consider a string X belonging to L. If X is sufficiently long, then the derivation of X must have involved recursive use of some variable A. Since A was used in the derivation, the derivation should have started as

$$S \overset{*}{\Rightarrow} uAy$$

for some values of u and y. Since A was used recursively the derivation must have continued as

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy$$

Finally the derivation must have eliminated all variables to reach a string X in the language.

$$S \overset{*}{\Rightarrow} uAy \overset{*}{\Rightarrow} uvAxy \overset{*}{\Rightarrow} uvwxy = x$$

This shows that derivation steps

$$A \overset{*}{\Rightarrow} vAx$$

and

$$A \overset{*}{\Rightarrow} w$$

are possible. Hence the derivation

$$A \overset{*}{\Rightarrow} vwx$$

must also be possible.

It should be noted here that the above does not imply that a was used recursively only once. The * of $\overset{*}{\Rightarrow}$ could cover many uses of A, as well as other recursive variables.

There has to be some "last" recursive step. Consider the longest strings that can be derived for v, w and x without the use of recursion. Then there is a number 'm' such that $|vwx| < m$.

Since the grammar does not contain any λ-productions or unit productions, every derivation step either introduces a terminal or increases the length of the sentential form. Since $A \overset{*}{\Rightarrow} vAx$, it follows that $|vx| > 0$.

Finally, since $uvAxy$ occurs in the derivation, and $A \overset{*}{\Rightarrow} vAx$ and $A \overset{*}{\Rightarrow} w$ are both possible, it follows that $uv^i wx^i y$ also belongs to L.

This completes the proof of all parts of Lemma.

Usage of Pumping Lemma

The Pumping Lemma can be used to show that certain languages are not context free.

Let us show that the language

$$L = \{a^i b^i c^i \mid i > 0\}$$

is not context-free.

Proof: Suppose L is a context-free language.

If string $X \in L$, where $|X| > m$, it follows that $X = uvwxy$, where $|vwx| \leq m$.

Choose a value i that is greater than m. Then, wherever vwx occurs in the string $a^i b^i c^i$, it cannot contain more than two distinct letters it can be all a's, all b's, all c's, or it can be a's and b's, or it can be b's and c's.

Therefore the string vx cannot contain more than two distinct letters; but by the "Pumping Lemma" it cannot be empty, either, so it must contain at least one letter.

Now we are ready to "pump".

Since $uvwxy$ is in L, uv^2wx^2y must also be in L. Since v and x can't both be empty,

$$|uv^2wx^2y| > |uvwxy|,$$

so we have added letters.

Both since vx does not contain all three distinct letters, we cannot have added the same number of each letter.

Therefore, uv^2wx^2y cannot be in L.

Thus we have arrived at a "contradiction".

Hence our original assumption, that L is context free should be false. Hence the language L is not con text-free.

Example

Check whether the language given by $L = \{a_m b_m c_n : m \leq n \leq 2m\}$ is a CFL or not.

Solution

Let $s = a^n b^n c^{2n}$, n being obtained from Pumping Lemma.

Then $s = uvwxy$, where $1 \leq |vx| \leq n$.

Therefore, vx cannot have all the three symbols a, b, c.

If you assume that vx has only a's and b's then we can shoose i such that uv^iwx^iy has more than $2n$ occurrence of a or b and exactly $2n$ occurences of c.

Hence $uv^iwx^iy \notin L$, which is a contradiction. Hence L is not a CFL.

Closure properties of CFL – Substitution

Let Σ be an alphabet, and suppose that for every symbol a in Σ, we choose a language L_a. These chosen languages can be over any alphabets, not necessarily Σ and not necessarily the same. This choice of languages defines a function s (a *substitution*) on Σ, and we shall refer to L_a as $s(a)$ for each symbol a.

If $w = a_1 a_2 \cdots a_n$ is a string in Σ^*, then $s(w)$ is the language of all strings $x_1 x_2 \cdots x_n$ such that string x_i is in the language $s(a_i)$, for $i = 1, 2, \ldots, n$. Put another way, $s(w)$ is the concatenation of the languages $s(a_1)s(a_2)\cdots s(a_n)$. We can further extend the definition of s to apply to languages: $s(L)$ is the union of $s(w)$ for all strings w in L.

Theorem 7.23: If L is a context-free language over alphabet Σ, and s is a substitution on Σ such that $s(a)$ is a CFL for each a in Σ, then $s(L)$ is a CFL.

PROOF: The essential idea is that we may take a CFG for L and replace each terminal a by the start symbol of a CFG for language $s(a)$. The result is a single CFG that generates $s(L)$. However, there are a few details that must be gotten right to make this idea work.

More formally, start with grammars for each of the relevant languages, say $G = (V, \Sigma, P, S)$ for L and $G_a = (V_a, T_a, P_a, S_a)$ for each a in Σ. Since we can choose any names we wish for variables, let us make sure that the sets of variables are disjoint; that is, there is no symbol A that is in two or more of V and any of the V_a's. The purpose of this choice of names is to make sure that when we combine the productions of the various grammars into one set of productions, we cannot get accidental mixing of the productions from two grammars and thus have derivations that do not resemble the derivations in any of the given grammars.

We construct a new grammar $G' = (V', T', P', S)$ for $s(L)$, as follows:

- V' is the union of V and all the V_a's for a in Σ.
- T' is the union of all the T_a's for a in Σ.
- P' consists of:
    1. All productions in any P_a, for a in Σ.
    2. The productions of P, but with each terminal a in their bodies replaced by S_a everywhere a occurs.

Thus, all parse trees in grammar G' start out like parse trees in G, but instead of generating a yield in Σ^*, there is a frontier in the tree where all nodes have labels that are S_a for some a in Σ. Then, dangling from each such node is a parse tree of G_a, whose yield is a terminal string that is in the language $s(a)$.

Applications of substitution theorem

Theorem 7.24: The context-free languages are closed under the following operations:

1. Union.
2. Concatenation.
3. Closure (*), and positive closure ($^+$).
4. Homomorphism.

PROOF: Each requires only that we set up the proper substitution. The proofs below each involve substitution of context-free languages into other context-free languages, and therefore produce CFL's by Theorem 7.23.

1. *Union*: Let L_1 and L_2 be CFL's. Then $L_1 \cup L_2$ is the language $s(L)$, where L is the language $\{1, 2\}$, and s is the substitution defined by $s(1) = L_1$ and $s(2) = L_2$.
2. *Concatenation*: Again let L_1 and L_2 be CFL's. Then $L_1 L_2$ is the language $s(L)$, where L is the language $\{12\}$, and s is the same substitution as in case (1).
3. *Closure and positive closure*: If L_1 is a CFL, L is the language $\{1\}^*$, and s is the substitution $s(1) = L_1$, then $L_1^* = s(L)$. Similarly, if L is instead the language $\{1\}^+$, then $L_1^+ = s(L)$.
4. Suppose L is a CFL over alphabet Σ, and h is a homomorphism on Σ. Let s be the substitution that replaces each symbol a in Σ by the language consisting of the one string that is $h(a)$. That is, $s(a) = \{h(a)\}$, for all a in Σ. Then $h(L) = s(L)$.

Reversal

Theorem 7.25: If L is a CFL, then so is L^R.

PROOF: Let $L = L(G)$ for some CFL $G = (V, T, P, S)$. Construct $G^R = (V, T, P^R, S)$, where P^R is the "reverse" of each production in P. That is, if $A \to \alpha$ is a production of G, then $A \to \alpha^R$ is a production of G^R. It is an easy induction on the lengths of derivations in G and G^R to show that $L(G^R) = L^R$. Essentially, all the sentential forms of G^R are reverses of sentential forms of G, and vice-versa. We leave the formal proof as an exercise. □

Inverse Homomorphism:

Theorem 7.30: Let L be a CFL and h a homomorphism. Then $h^{-1}(L)$ is a CFL.

PROOF: Suppose h applies to symbols of alphabet Σ and produces strings in T^*. We also assume that L is a language over alphabet T. As suggested above, we start with a PDA $P = (Q, T, \Gamma, \delta, q_0, Z_0, F)$ that accepts L by final state. We construct a new PDA

$$P' = (Q', \Sigma, \delta', (q_0, \epsilon), Z_0, F \times \{\epsilon\}) \tag{7.1}$$

where:

1. Q' is the set of pairs (q, x) such that:

   (a) q is a state in Q, and

   (b) x is a suffix (not necessarily proper) of some string $h(a)$ for some input symbol a in Σ.

   That is, the first component of the state of P' is the state of P, and the second component is the buffer. We assume that the buffer will periodically be loaded with a string $h(a)$, and then allowed to shrink from the front, as we use its symbols to feed the simulated PDA P. Note that since Σ is finite, and $h(a)$ is finite for all a, there are only a finite number of states for P'.

2. δ' is defined by the following rules:

   (a) $\delta'\big((q, \epsilon), a, X\big) = \{\big((q, h(a)\big), X\big)\}$ for all symbols a in Σ, all states q in Q, and stack symbols X in Γ. Note that a cannot be ϵ here. When the buffer is empty, P' can consume its next input symbol a and place $h(a)$ in the buffer.

   (b) If $\delta(q, b, X)$ contains (p, γ), where b is in T or $b = \epsilon$, then

   $$\delta'\big((q, bx), \epsilon, X\big)$$

   contains $\big((p, x), \gamma\big)$. That is, P' always has the option of simulating a move of P, using the front of its buffer. If b is a symbol in T, then the buffer must not be empty, but if $b = \epsilon$, then the buffer can be empty.

3. Note that, as defined in (7.1), the start state of P' is (q_0, ϵ); i.e., P' starts in the start state of P with an empty buffer.

4. Likewise, the accepting states of P', as per (7.1), are those states (q, ϵ) such that q is an accepting state of P.

The following statement characterizes the relationship between P' and P:

- $(q_0, h(w), Z_0) \overset{*}{\underset{P}{\vdash}} (p, \epsilon, \gamma)$ if and only if $\big((q_0, \epsilon), w, Z_0\big) \overset{*}{\underset{P'}{\vdash}} \big((p, \epsilon), \epsilon, \gamma\big)$.