1 Pushdown automaton

The pushdown automaton is essentially an ϵ-NFA with the addition of a stack (first in-last out) i.e finite automaton with control of both an input tape and a stack. The stack can be read, pushed (insert), and popped (delete) only at the top just like the stack data structure. That is, symbols may be pushed or popped only at the top of the stack. When a symbol is pushed at the top, the symbol previously at the top becomes second from the top, the symbol previously second from the top becomes third, and so on. Similarly, when a symbol is popped from the top of the list, the symbol previously second from the top becomes the top symbol, the symbol previously third from the top becomes second, and so on.

Preliminaries:

– Push and Pop operation in stack

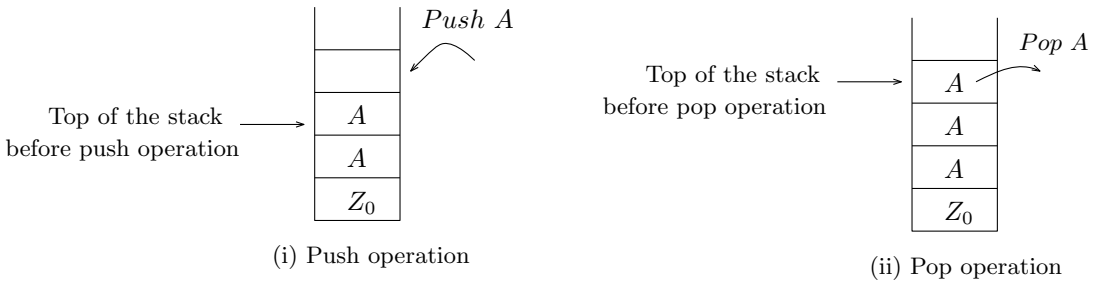


Fig. 1. *Push and Pop operation in stack*

– Pushdown automaton (PDA) involves seven components, $P = (Q, \sum, \Gamma, \delta, q_0, Z_0, F)$.
Q : A finite set of states.
$\sum$: A finite set of input symbols or alphabets.
Γ : A finite stack alphabet.
δ : A mapping from (transition function) $Q \times (\sum \cup \{\epsilon\}) \times \Gamma$ to finite subsets of $Q \times \Gamma^*$.
q_0: The start state or initial state.
Z_0: A particular stack symbol called the start symbol.
F: The set of final states.

– The interpretation of
(i) $\delta(q_0, a, Z_0) = (q_1, AZ_0)$
q_0 and q_1 are the states.
a is an input symbol
Z_0 is a stack symbol in particular start symbol.
A is a stack symbol.

PDA is in state q_0, with input symbol a and Z_0 is the top symbol on the stack, PDA moves to the state q_1, and pushes A onto the stack. Now the top of the stack becomes A and Z will be the second symbol in the stack. Advance the input head by one symbol.

(ii) $\delta(q_0, a, A) = (q_1, \epsilon)$
PDA is in state q_0, with input symbol a and A is the top symbol on the stack, PDA moves to the state q_1, and pop A from the stack. The symbol previously second from the top becomes the top symbol. Advance the input head by one symbol.

(iii) $\delta(q_0, \epsilon, A) = (q_f, Z_0)$. Here, ϵ denotes, input tape is empty. The PDA reaches the final state q_f.

(iv) In transition diagram the notation aZ_0/AZ_0 denotes, if input symbol is a and TOP is Z_0, then we push A onto the stack. We always write top two symbols in the stack.

An interesting question is that, can PDA accept non-regular languages? In this section we shall answer this with some case studies.

1. $L = \{a^n b^n \mid n \geq 1\}$.

   Idea behind the PDA construction:
   When we read a, we push A onto the stack and when we read b, pop A from the stack. Once the input tape is empty and the top of the stack (TOP) is Z_0, we say that the string is accepted by the PDA. In case if the input tape is empty and there are any symbols left (TOP is not Z_0), then we say that the string is rejected.

   Stack configuration:
   Consider a valid input string *aaabbb*. We shall trace the above approach for the string *aaabbb*.

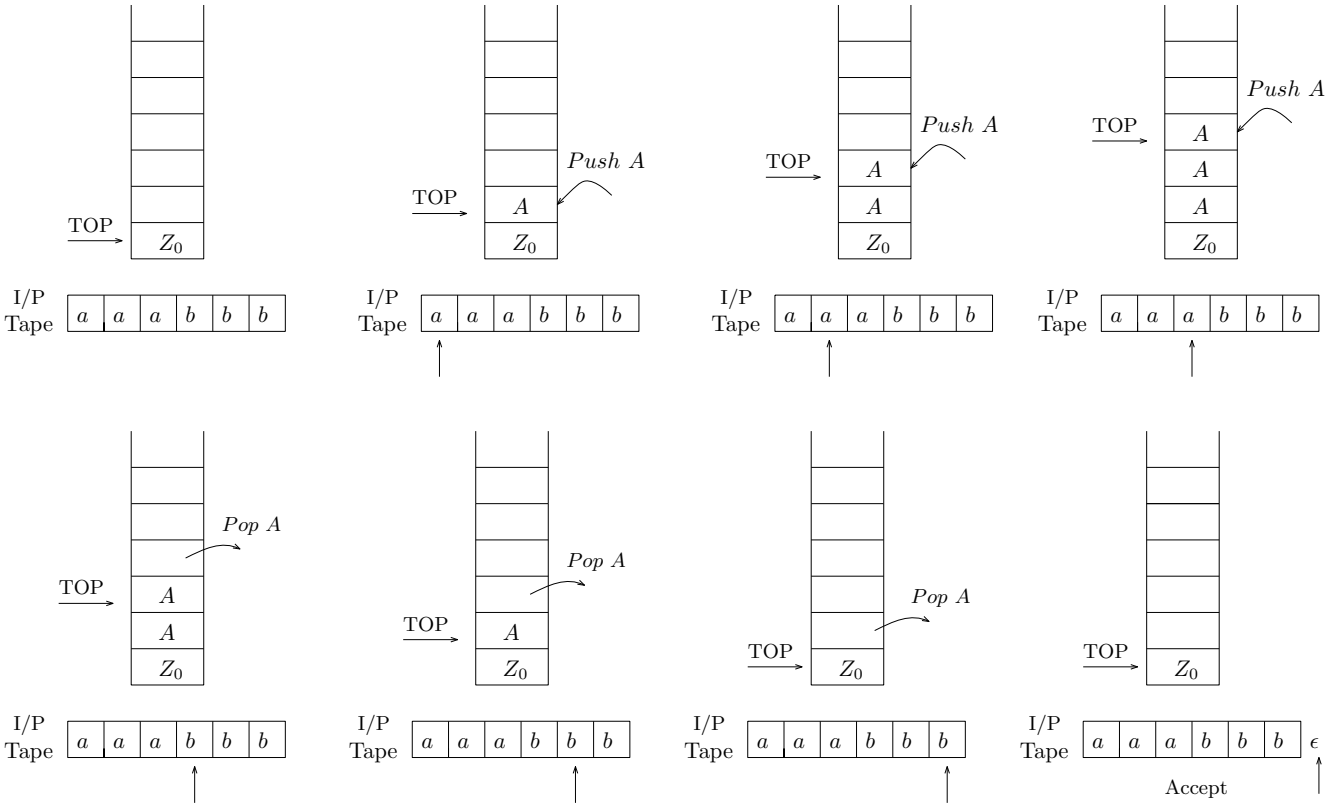


Fig. 2. *Stack configuration for the input string aaabbb.*

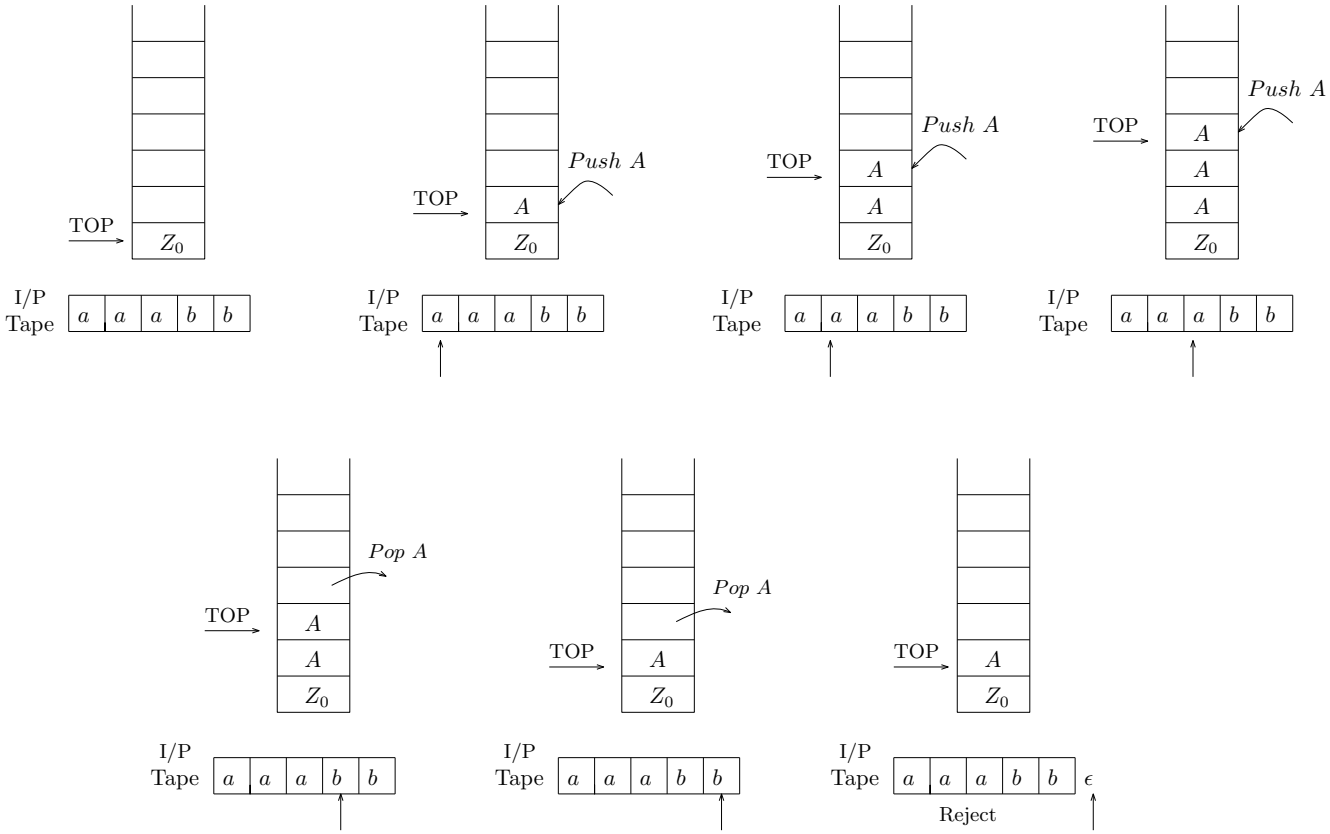


Fig. 3. *Stack configuration for the input string aaabb.*

Transition function:

- $\delta(q_0, a, Z_0) = (q_1, AZ_0)$
  When we are in state q_0 and we see the start symbol Z_0 at the top of the stack and input symbol is a. We push A onto the stack, leaving Z_0 below to mark the empty stack. TOP is A.
- $\delta(q_1, a, A) = (q_1, AA)$
  When we are in state q_1 and if the input symbol is a and TOP is A. We stay in the state q_1 and push A onto the stack. TOP is A.
- $\delta(q_1, b, A) = (q_2, \epsilon)$
  When we are in state q_1 and if the input symbol is b and TOP is A. We move to the state q_2 from q_1 and pop A from the stack.
- $\delta(q_2, b, A) = (q_2, \epsilon)$
  When we are in state q_2 and if the input symbol is b and TOP is A. We stay in the state q_2 itself and pop A from the stack.
- $\delta(q_2, \epsilon, Z_0) = (q_f, Z_0)$ or (q_f, ϵ)
  When we are in state q_2 and if the input tape is empty and TOP is Z_0. We move to the final state q_f.

The remaining possible transitions leads to reject state
$\delta(q_0, b, Z_0) = (q_r, Z_0)$
$\delta(q_2, a, A) = (q_r, A)$
$\delta(q_2, \epsilon, A) = (q_r, A)$
$\delta(q_2, a, Z_0) = (q_r, Z_0)$, where q_r is the dead state.

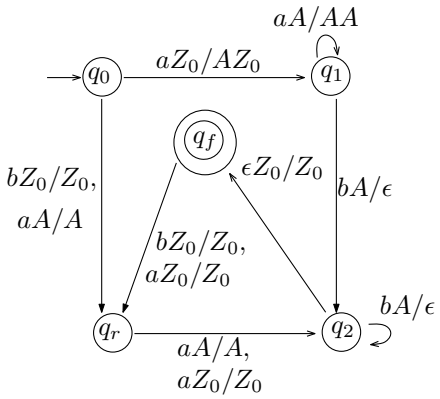


Fig. 4. *Transition diagram for the language* $L = \{a^n b^n \mid n \geq 1\}$

This shows that the language $L = \{a^n b^n \mid n \geq 1\}$ is accepted by PDA. Therefore, PDA accepts non-regular languages.

2. $L = \{x \mid x \in \{a, b\}^*, x$ that accepts equal number of $a's$ and $b's\}$.
   Approach:
   When we read a and TOP is Z_0 or A, we push A onto the stack. Similarly, when we read b and TOP is Z_0 or B, we push B onto the stack. If the input symbol is a and TOP is B, then we pop B. Similarly, If the input symbol is b and TOP is A, then we pop A. Once the input tape is empty and the top of the stack (TOP) is Z_0, we say that the string is accepted by the PDA. In case if the input tape is empty and there are any symbols left (TOP is not Z_0), then we say that the string is rejected.

   Transition function:
   $(q_0, a, Z_0) = (q_0, AZ_0)$
   $(q_0, b, Z_0) = (q_0, BZ_0)$
   $(q_0, a, A) = (q_0, AA)$
   $(q_0, b, B) = (q_0, BB)$
   $(q_0, a, B) = (q_0, \epsilon)$
   $(q_0, b, A) = (q_0, \epsilon)$
   $(q_0, \epsilon, Z_0) = (q_f, Z_0)$, Accepted.
3. $L = \{a^n b^{2n} \mid n \geq 1\}$.
   Approach:
   When we read a, we push two $A's$ onto the stack. When we read b, we pop A from the stack. Once the input tape is empty and the top of the stack (TOP) is Z_0, we say that the string is accepted by the PDA.

   Transition function:
   $(q_0, a, Z_0) = (q_0, AA\ Z_0)$
   $(q_1, a, A) = (q_0, AA\ A)$
   $(q_1, a, A) = (q_2, \epsilon)$
   $(q_2, b, A) = (q_2, \epsilon)$
   $(q_2, \epsilon, Z_0) = (q_f, Z_0)$ or (q_f, ϵ) or (q_2, ϵ)
   $(q_0, b, Z_0) = (q_r, Z_0)$
   $(q_2, a, A) = (q_r, A)$
   $(q_2, a, Z_0) = (q_r, Z_0)$

4. $L = \{a^n b^m \mid n < m \text{ and } n, m \geq 1\}$.
   We can rewrite L as $L' = \{a^n b^n b^k \mid n, k \geq 1\}$.
   Approach:
   When we read a, we push A onto the stack. When we read b and TOP is A, we pop A from the stack. Suppose, the input symbol is b and TOP is A, then we do nothing . i,e ignore the $b's$.

   Transition function:
   $(q_0, a, Z_0) = (q_1, AZ_0)$
   $(q_1, a, A) = (q_1, AA)$
   $(q_1, b, A) = (q_2, \epsilon)$
   $(q_2, b, A) = (q_2, \epsilon)$
   $(q_2, b, Z_0) = (q_2, Z_0)$
   $(q_2, \epsilon, Z_0) = (q_f, Z_0)$ or (q_f, ϵ)

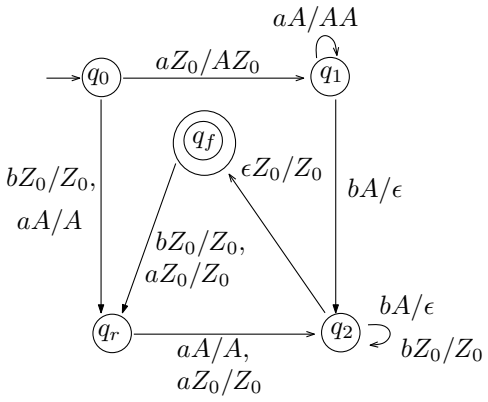


Fig. 5. *Transition diagram for the language* $L = \{a^n b^m \mid n < m \text{ and } n, m \geq 1\}$

5. $L = \{a^n b^m c^{n+m} \mid n < m \text{ and } n, m \geq 1\}$.
   Approach:
   When we read a, we push A onto the stack. Similarly for b, we push B onto the stack. When we read c, we pop the symbol pointed by TOP from the stack.

   Transition function:
   $(q_0, a, Z_0) = (q_1, AZ_0)$
   $(q_1, a, A) = (q_1, AA)$
   $(q_1, b, A) = (q_2, BA)$
   $(q_2, b, B) = (q_2, BB)$
   $(q_2, c, B) = (q_2, \epsilon)$
   $(q_2, c, Z_0) = (q_f, Z_0)$
6. $L = \{wcw^R \mid w \in a, b^*\}$.
   w^R : Reverse of w, c : Delimiter. Approach:
   When we read a, we push A onto the stack. Similarly for b, we push B onto the stack. When we read c, we do nothing. When we read a and TOP is A, we pop A. If we see b and TOP is B, we pop B.

   Transition function:
   $(q_0, a, Z_0) = (q_1, AZ_0)$
   $(q_0, b, Z_0) = (q_1, BZ_0)$
   $(q_1, a, A) = (q_1, AA)$

$(q_0, a, B) = (q_1, AB)$
$(q_1, b, A) = (q_1, BA)$
$(q_1, b, B) = (q_1, BB)$
$(q_1, c, A) = (q_2, A)$
$(q_1, c, B) = (q_2, B)$
$(q_0, c, z_0) = (q_2, z_0)$
$(q_2, a, A) = (q_2, \epsilon)$
$(q_2, b, B) = (q_2, \epsilon)$
$(q_2, \epsilon, Z_0) = (q_f, z_0)$ or (q_2, ϵ)

So far, what we have seen is a deterministic pushdown automaton (DPDA) because for an arbitrary $q \in Q$, $a \in \sum$ and $X \in \Gamma$, transition $\delta(q, a, X)$, is mapped to a unique entry in $Q \times \Gamma^*$. If $\delta(q, a, X)$ is defined, then $\delta(q, \epsilon, X)$ is undefined. On the contrary, assume that both are defined. Let a be the input symbol. We can see it as $a = \epsilon a = a\epsilon = \epsilon\epsilon \ldots \epsilon a = a\epsilon\epsilon\epsilon \ldots \epsilon$. Since both ae defined, we can take any one branch. This brings non-determinism, which is a contradiction.
Note: We know that in FA, $\delta(q, a)$ must be uniquely defined and $\delta(q, \epsilon)$ is undefined.
Consider the language $L = \{ww^R \mid w \in a, b^*\}$.The PDA has to decide deterministically where the first half of the w ends and w^R starts. We know that the PDA can only read a symbol and move one cell right. Also, machine does not know the length of w and middle point. A natural design is that at each stage the machine has to decide whether the input symbol is part of w or w^R. Clearly, we see that the machine is a non-deterministic machine. We call this machine as non-deterministic pushdown automaton (NDPDA). We say that a language is accepted by NPDA, if there exist at least one sequence of moves from q_0 to q_f. Let us design a NPDA for the language $L = \{ww^R \mid w \in a, b^*\}$

Transition function:
$(q_0, a, Z_0) = (q_0, AZ_0)$
$(q_0, a, A) = (q_0, AA),\ (q_1, \epsilon)$
The transition $(q_0, a, A) = (q_0, AA),\ (q_1, \epsilon)$ tells that there are two branches. The machine guesses that the symbol a is part of w and takes one branch to push A. If the machine decides it is part of w^R then it pop A by taking another branch.
$(q_0, b, Z_0) = (q_0, BZ_0)$
$(q_0, b, B) = (q_0, BB),\ (q_1, \epsilon)$
$(q_0, a, B) = (q_0, AB)$
$(q_0, b, A) = (q_0, BA)$
$(q_1, b, B) = (q_1, \epsilon)$
$(q_1, a, A) = (q_1, \epsilon)$
$(q_0, \epsilon, Z_0) = (q_1, Z_0)$
$(q_1, \epsilon, Z_0) = (q_f, Z_0)$ or (q_f, ϵ)

2 Grammar

A grammar is a 4-tuple $G = (N, T, P, S)$, where N is a finite set of non-terminal alphabet (we denote it using uppercase), T is a finite set of terminal alphabet (we denote it using lowercase), $S \in N$ is the start symbol and P is a set of productions of the form $u \to v$, where $u \in (N \cup T)^* N (N \cup T)^*$ and $v \in (N \cup T)^*$. Each language has a grammar. For instance consider a statement in English language "India is a country". Any English statement can be written as noun phrase and verb phrase. The production is:

$$S \to \text{Noun phrase Verb phrase}$$

$$\text{Verb phrase} \to \text{verb noun phase}$$

$$\text{Noun phrase} \to \text{Adjective Noun}$$

For the statement: India is a country, the derivation is

$$S \rightarrow \text{Noun phrase Verb phrase}$$
$$\rightarrow \text{India Verb phrase}$$
$$\rightarrow \text{India Verb noun phrase}$$
$$\rightarrow \text{India is Adjective Noun}$$
$$\rightarrow \text{India is a country}$$

Similarly, we can generate all statements in English using the productions.

3 Regular grammars

The class of languages accepted by finite automaton can be represented as a regular expression, transition diagram, transition function or regular grammar. If $G = (N, T, P, S)$, where N, T, S is same as defined before. The production P for finite automaton is:

$$u \rightarrow vw$$

where $u \in N$, $v \in T$, $w \in N^*$. We can generate regular grammar from a finite automaton or directly from a language.

Consider the $L = \{a^n b^m \mid n, m \geq 1\}$. The strings present in the language contains at least one a and at least one b. To ensure this we can write production as

$$q_0 \rightarrow aq_1$$
$$q_1 \rightarrow aq_1$$
$$q_1 \rightarrow bq$$
$$q_2 \rightarrow bq_2$$
$$q_2 \rightarrow \epsilon$$

Note that the productions $q_0 \rightarrow aq_1$, $q_1 \rightarrow aq_1$ generates a^n. Similarly the productions $q_1 \rightarrow bq$, $q_2 \rightarrow bq_2$ generates b^m. We shall look into the derivation of the string $aaab$, $q_0 \rightarrow aq_1 \rightarrow aaq_1 \rightarrow aaaq_1 \rightarrow aaabq_2 \rightarrow aaab\epsilon = aaab$.

Consider the L which is set of all strings over $\{0, 1\}^*$ and ending with 011. For this language, we know that regular expression is $(0 + 1)^*011$ and the corresponding NFA is in Figure 6. By using NFA in

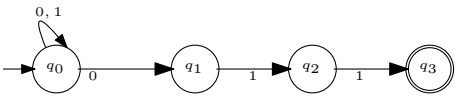


Fig. 6. NFA.

Figure 6, we can production as:

$$q_0 \rightarrow 0q_0$$
$$q_0 \rightarrow 1q_0$$
$$q_0 \rightarrow 0q_1$$
$$q_1 \rightarrow 1q_2$$
$$q_2 \rightarrow 1q_3$$
$$q_3 \rightarrow \epsilon$$

4 Context-Free grammars

We know that, a language is regular if it can be built from individual strings by concatenation, union, and closure. We shall consider a wider class of context-free languages, which are languages that can be built from individual strings by concatenation, union, and recursion. Further, the class of languages accepted by finite automaton is subset of class of languages accepted by pushdown automaton. The class of languages accepted by pushdown automaton is precisely context-free grammar. Thus, regular grammar is special case of context-free grammar (or context-free grammar is generalization of regular grammar).
Let us formally define a context-free grammar (CFG):
A context-free grammar is a 4-tuple $G = (N, T, P, S)$ where N is set of non-terminals, T is the set of terminals Σ, P are elements of production which is of the form $A \to \alpha$ ($A \in N$, $\alpha \in (N \cup T)^*$, α is a string over $\{N, T\}^*$), S is the start symbol.
Consider the language $L = \{a^n b^n \mid n \geq 0\}$. We know that L is not a regular language. We shall generate the set of productions to show that L is context-free grammar. Consider the production

$$S \to aSb \mid \epsilon.$$

We have not mentioned about N, T which are known from P. We shall write only productions for a given language. To derive the string $a^n B^n$, by starting with $S \to aSb$, again applying the production $S \to aSb$, we get $aaSbb$. By applying the production $S \to aSb$ for n times, we get $a^n Sb^n$. Finally, by applying the production $S \to \epsilon$, we get $a^n b^n$. From this it is clear that regular grammar is special case of context-free grammar.
We say that the string is part of language if starting from S such that $S \to \alpha_1 \to \alpha_2 \to \alpha_3 \to \ldots \to w$, where $\alpha_i \in (N \cup T)^*$, $w \in T^*$, $w \in L(G)$. We use $L(G)$ to represent the set of strings generated by G. We say that the language is generated by CFG:
If $G = (N, T, P, S)$ is a CFG, the language generated by G is

$$L(G) = \{x \in \Sigma^* \mid S \to^*_G x\}$$

A language L is a context-free language (CFL) if there is a CFG G with $L = L(G)$.
We shall generate CFG for some more languages that are accepted by PDA:

1. **Consider the language** $L_2 = \{a^n b^{2n} \mid n \geq 1\}$. The string $a^n b^{2n}$ can be seen as $a^n b^n b^n$. For each a, two b's should be generated. Hence the production P_2 can be written as

   $$S \to aSbb$$

   $$S \to abb$$

   Note that the production $S \to abb$ generates strings when $n = 1$ and when $n \geq 2$, the production $S \to aSbb$ is used $n - 1$ times for n^{th} time $S \to abb$ production is used. Hence any string of the form $a^n b^{2n}$ can be generated by using P_2.
2. **Consider the language** $L_2 = \{a^n b^m \mid n < m,\ n, m \geq 1\}$. The string $a^n b^m$ can be seen as $a^n b^n b^k$, $m = n + k$. Whenever a is generated corresponding b should be generated and at least one more b should be generated. The production is

   $$S \to AB$$

   $$A \to aAb$$

   $$A \to ab$$

   $$B \to bB$$

   $$B \to b$$

   Note that the production $A \to aAb$ $A \to ab$ generates strings of the form $a^n b^n$ and the production $B \to bB$ $B \to b$ generates at least one b to any string generated.

3. **Consider the language** $L_3 = \{a^n b^m c^{n+m} \mid n, m \geq 1\}$. The string $a^n b^m c^{n+m}$ can be written as $a^n b^m c^n c^m$. The production should generate corresponding c whenever a is generated. Similarly, whenever b is generated corresponding c should also be generated. Hence consider the production P_3:

$$S \to aSc$$

$$S \to bSc$$

The production $S \to bSc$ says that strings can start with b which is incorrect. Hence the production P_3 is modified as:

$$S \to aSc$$

$$S \to aAc$$

$$A \to bAc$$

$$A \to bc$$

We shall derive the string $aaabbccccc$ which is part of L_2 by using G:

$$S \to aSc \to aaScc \to aaaAccc \to aaabAcccc \to aaabbccccc$$

Consider the production P_2':

$$S \to aSc$$

$$S \to aAc$$

$$A \to bAc$$

$$A \to \epsilon$$

Let us analyze the equivalence of P_2 and P_2':
Consider the string $ac \notin L$, by using P_2', $S \to aAc \to a\epsilon c = ac$ which is not in L, hence p_2' is not a valid production for L.

4. **Consider the language** L_4 **which accepts set of all strings of the form** ww^R**, where** w **is the set of all strings over** $\{a, b\}^*$**.**
Let us consider some strings that are in L_4, ϵ, aa, bb, $abba$, $abbba$, $aabbaa$. We can observe that the alphabets at $1, n$ are same, $2, n-1$ are same, ..., $i, n-i+1$ are same. With this observation, we can write production as:

$$S \to aSa$$

$$S \to bSb$$

$$S \to \epsilon$$

Note the string ϵ is in L, hence $S \to \epsilon$ is in the production. When we use ϵ in the production we should be careful about the strings that are generated.

5. **Consider the language** L_5 **which accepts set of all strings of the form** wcw^R**, where** w **is the set of all strings over** $\{a, b\}^*$**.**
Let us consider some strings that are in L_5, ϵ, aa, bb, $abba$, $abbba$, $aabbaa$. We can observe that the alphabets at $1, n$ are same, $2, n-1$ are same, ..., $i, n-i+1$ are same. With this observation, we can write production as:

$$S \to aSa$$

$$S \to bSb$$

$$S \to c$$

Note that the length of any string in L is odd, hence $S \to c$ is in the production generates the middle symbol c.

6. **Consider the language $L_6 = \{w \mid w$ is a palindrome$\}$.** We know that a palindrome is a string w having the property that $w = w^R$, i.e., reading w from left to right gives the same result as reading w from right to left. Form this, We can observe that the strings in L_6 will have the property that: the alphabets at $1, n$ are same, $2, n-1$ are same, ..., $i, n-i+1$ are same. If string is even, then whenever 0 is generated corresponding 0 will be generated, similarly for 1. When the string is of odd length, whenever 0 is generated corresponding 0 will be generated, similarly for 1 and middle symbol can either be 0 or 1. With this observation, we can write production as:

$$S \to 0S0$$

$$S \to 1S1$$

$$S \to \epsilon$$

$$S \to 0$$

$$S \to 1$$

7. **Consider the language $L_7 = \{x \mid x \in \{0,1\}^*,\ n_0(x) = n_1(x)\}$, where $n_0(x)$ represents the number of 0's in x and $n_1(x)$ represents the number of 1's in x.** Since $n_0(x) = n_1(x)$ in w, whenever 0 is generated 1 should be generated and vice versa. For this the production is

$$S \to 0S1$$

$$S \to 1S0$$

This production rule can be applied in any order. Hence the production

$$S \to SS$$

Therefore the production for G corresponding to L_7 is

$$S \to SS$$

$$S \to 0S1$$

$$S \to 1S0$$

$$S \to \epsilon$$

8. **Consider the language $L_8 = \{x \mid x \in \{0,1\}^*,\ n_0(x) \neq n_1(x)\}$, where $n_0(x)$ represents the number of 0's in x and $n_1(x)$ represents the number of 1's in x.**
Consider the languages $L_9 = \{x \mid x \in \{0,1\}^*,\ n_0(x) > n_1(x)\}$, and $L_{10} = \{x \mid x \in \{0,1\}^*,\ n_0(x) < n_1(x)\}$. Clearly, $L_8 = L_9 \cup L_{10}$.
Let us consider L_9. The strings that contains only 0's should be in $L(G)$. Hence the production $A \to 0 \mid 0A$
Consider the strings of length one: the possible strings are 0; the strings of length two: the possible strings are 00; the strings of length three: the possible strings are 001, 100, 010; the strings of length four are precisely 0's added to the strings of length three; the strings of length five are the strings either 0 or 1 concatenated to the strings of length four, and so on. Since $n_0(x) >_1 (x)$ in w, the number of 0's should be at least one more than the number of 1's. Consider the production:

$$A \to 1AA \mid A1A \mid AA1$$

$$A \to 0 \mid 0A$$

Note that in the production at any iteration the number of A's is more than the number of 1's. Hence the production generates strings in which $n_0(x) > n_1(x)$
Similarly, for L_{10}, we can write the production as:

$$B \to 0BB \mid B0B \mid BB0$$

$$B \to 1 \mid 1B$$

Therefore, we can write the production for L_8 in which a string is from either L_9 or L_{10} as

$$S \to A \mid B$$

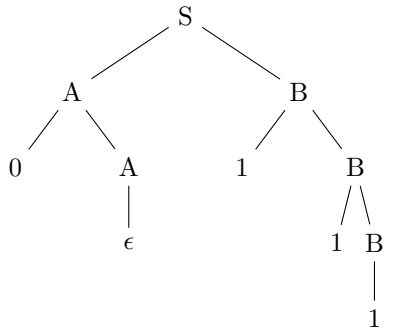
$$A \to 1AA \mid A1A \mid AA1 \mid 0 \mid 0A$$

$$B \to 0BB \mid B0B \mid BB0 \mid 1 \mid 1B$$

From this we can observe that: If L_1 is a CFG and L_2 is a CFG, then $L_1 \cup L_2$ is a CFG.

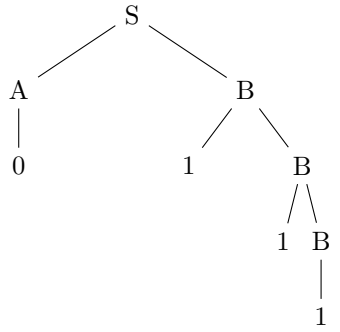
Derivations in CFG A derivation of a string w in a grammar G is a *leftmost derivation* if at every step the leftmost remaining variable is the one replaced. Similarly, a derivation of a string w in a grammar G is a *rightmost derivation* if at every step the rightmost remaining variable is the one replaced. A derivation of a string w in a grammar G is a *mixed derivation* if at every step the remaining variable is replaced in any order.
Consider this production $S \to AB$, $A \to 0A$, $B \to 1B$, $A \to 0 \mid \epsilon$, $B \to 1 \mid \epsilon$. We shall construct all the derivations and its corresponding parse trees for the string 0111. Consider this derivation:

$$\begin{aligned} S &\to AB \\ &\to A1B \\ &\to A11B \\ &\to A111 \\ &\to 0111 \end{aligned}$$



Note that at each step leftmost non-terminal is replaced, hence it is leftmost derivation. Consider this derivation:

$$\begin{aligned} S &\to AB \\ &\to 0AB \\ &\to 0\epsilon B \\ &\to 0\epsilon 1B \\ &\to 0\epsilon 11B \\ &\to 0\epsilon 111 = 0111 \end{aligned}$$



Note that at each step rightmost non-terminal is replaced, hence it is rightmost derivation.
Consider this derivation:

$$\begin{aligned} S &\to AB \\ &\to A1B \\ &\to A11B \\ &\to 011B \\ &\to 0111 \end{aligned}$$

Note that at each step rightmost non-terminal is replaced, hence it is rightmost derivation.
A string that is accepted by a grammar, one come up with a leftmost grammar or rightmost grammar or mixed grammar.

Ambiguity: If a grammar generates the same string in several different ways by using a particular derivation, we say that the string is derived ambiguously in that grammar. If a grammar generates some string ambiguously, we say that the grammar is ambiguous (i.e., there exists a string $w \in L(G)$ such that w has at least two leftmost derivations).
Consider the production

$$E \to E + E|(E)|a$$

we shall derive for the string $a + a + a$:

$$\begin{aligned} E &\to E + E \\ &\to E + a \\ &\to E + E + a \\ &\to E + a + a \\ &\to a + a + a \end{aligned} \qquad \begin{aligned} E &\to E + E \\ &\to E + E + E \\ &\to E + E + a \\ &\to E + a + a \\ &\to a + a + a \end{aligned}$$

Note that both the derivations are rightmost derivations for the same string. Hence the grammar is ambiguous grammar.

Concatenation:
Construct a CFG for th following language $L = a^i b^j c^k$, $j > i + k$.
We split the given CFG into a known CFG. Since $j > i + k$, we rewrite this as follows
$a^i b^{i+k} b^m c^k$, $m \geq 1$.
$a^i b^i b^k b^m c^k$
$L : a^i b^i b^m b^k c^k$
$L : L_1 \cdot L_2 \cdot L_3$
$L_1 : a^i b^i$, $i \geq 0$
$L_2 : b^m$, $m \geq 1$
$L_3 : b^k c^k$, $k \geq 0$

$L_1 : A \to aAb \mid \epsilon$
$L_2 : B \to bB \mid b$
$L_3 : C \to bCc \mid \epsilon$
We write this as follows
$S \to ABC$
$A \to aAb \mid \epsilon$
$B \to bB \mid b$
$C \to bCc \mid \epsilon$
Remark: If L_1, L_2, L_3 have CFG then $L : L_1 \cdot L_2 \cdot L_3$ has CFG. This shows that concatenation is closed (I.e if L_1, L_2 are context free then $L : L_1 \cdot L_2$ is context free).

KleeneStar:
Consider the regular expression $L : (01)^* \mid (11 + 0)^*$
We know that finite automata exists for the regular expression which means we can construct a regular grammar.
$S \to A \mid B$
$A \to 01A \mid \epsilon$
$B \to (11 + 0)B \mid \epsilon$
We further simplify this and we get
$S \to A \mid B$
$A \to 01A \mid \epsilon$
$B \to CB \mid \epsilon$
$C \to 11 \mid 0$
Kleene star is close with respect to CFG. If L is having CFG then L^* is also having CFG.
An interesting question is that if L_1, L_2 both are having CFG, can we construct CFG for $L_1 \cap L_2$?

5 Simplification of Context-Free Grammars (CFG)

In a CFG, it may happen that all the production rules and symbols are not needed for the derivation of strings. Such CFG can be simplified without reducing its generative power by eliminating those productions and symbols. This elimination process is called as simplification of CFGs. Simplifications of CFG make it easier to prove facts about Context-free languages (CFL). For example, one can claim that if a language is a CFL, then it has grammar in some special form. Simplifications of CFG involve the following rules:

- **Rule1: Remove ϵ-rules.** The productions of type $A \to \epsilon$ are called ϵ productions. Suppose there is an epsilon production, then it should be removed. Exception: $S \to \epsilon$, S is a start symbol. i.e., If ϵ is part of the language, then it should be preserved.
- **Rule 2: Remove unit productions** . Any production rule in the form $A \to B$ where A and B are Non-terminal is called unit production.
- **Rule 3: Remove useless symbols / Not reachable symbols.** A symbol is useless if it does not appear on the right-hand side of the production rule and does not take part in the derivation of any string.

1. Simplify the following CFG
   $S \to AB0C$
   $A \to BC$
   $B \to 1 \mid \epsilon$
   $C \to D \mid \epsilon$
   $D \to \epsilon$
   $\Sigma = \{0, 1\}$, Non-terminals $= \{S, A, B, C, D\}$.
   We first remove ϵ-rules from the production. Here we have three ϵ-rules, $B \to \epsilon$, $C \to \epsilon$ and $D \to \epsilon$. By substituting the ϵ in place of B and C in $A \to BC$, we get $A \to \epsilon$. Now substitute ϵ in place of A, B and C in $S \to AB0C$.
   $S \to AB0C \mid B0C \mid A0C \mid AB0 \mid 0C \mid B0 \mid A0 \mid 0$.
   $A \to BC \mid C \mid B$
   $B \to 1$
   $C \to D$
   Now we remove unit productions. $A \to B$ and $C \to D$ are unit productions. From $A \to B$, $B \to 1$, we get $A \to 1$. No strings belongs to Σ^* is generated by D. So we remove $C \to D$ and $A \to C$.
   $S \to AB0C \mid B0C \mid A0C \mid AB0 \mid 0C \mid B0 \mid A0 \mid 0$.
   $A \to BC \mid 1$
   $B \to 1$
   Now we apply rule 3 i.e remove useless symbols. Here the non-terminal symbol C is useless and we remove it.
   The simplified CFG is :
   $S \to \mid AB0 \mid B0 \mid A0 \mid 0$.
   $A \to \ 1$
   $B \to 1$

2. Simplify the following CFG
   $S \to eSe$
   $S \to GH$
   $G \to cGb$
   $G \to \epsilon$
   $H \to JHd$
   $H \to \epsilon$
   $J \to bJ$
   $J \to f$
   $\Sigma = \{b, c, d, e, f\}$, Non-terminals $= \{S, G, H, J\}$. We first remove ϵ-rules from the production. Now substitute ϵ in place of G, and H in the productions $S \to GH$, $G \to cGb$, $H \to JHd$.

$S \to eSe \mid GH \mid G \mid H$
$G \to cGb \mid cb$
$H \to JHd \mid Jd$
$J \to bJ \mid f$

Now we remove unit productions. $S \to G \mid H$ are unit productions.
$S \to eSe \mid GH \mid cGb \mid cb \mid JHd \mid Jd$
$G \to cGb \mid cb$
$H \to JHd \mid Jd$
$J \to bJ \mid f$

No useless symbols in these productions. If we start from S, we will always generate a string over Σ^*. The simplified CFG is :
$S \to eSe \mid GH \mid cGb \mid cb \mid JHd \mid Jd$
$G \to cGb \mid cb$
$H \to JHd \mid Jd$
$J \to bJ \mid f$

3. Analyze the language accepted by the above simplified CFG.
Consider the production $S \to eSe$. We observe that we get even number of $e's$ by substituting eSe for S repeatedly. Similarly, on substituting cGb and cb for G repeatedly we get equal number of $c's$ and $b's$.
From G we get $G \to c^m b^m$, $m \geq 1$
$H \to J^k d^k$, $k \geq 1$
Note that J is an non-terminal. We expand J as follows. $J\ J\ J \implies bJ\ J\ J \implies bbJ\ J\ J \implies bbJ\ J\ bJ \implies bbJ\ J\ bbJ \implies bbJ\ J\ bbbJ \implies bbJ\ J\ bbbf \implies bbf\ f\ bbbf \implies b^2 f f b^3 f \implies (b^* f)^k d^k$
Now we shall what can be generated starting from S.
$S \to c^m b^m$, $m \geq 1$ from $S \to cGb \mid cb$
$\cup$
$c^m b^m (b^* f)^k d^k$, $m, k \geq 1$ from $S \to GH$
$\cup$
$e^n S^n e^n \implies (e^n c^m b^m e^n \cup e^n c^m b^m (b^* f)^k d^k e^n) \cup (b^* f)^k d^k$

$L_1 = c^m b^m$, $m \geq 1$
$L_2 = c^m b^m (b^* f)^k d^k$, $m, k \geq 1$
$L_3 = e^n c^m b^m e^n$, $m, n \geq 1$
$L_4 = e^n c^m b^m (b^* f)^k d^k e^n)$, $m, n, k \geq 1$
$L_5 = (b^* f)^k d^k$, $k \geq 1$

Normalization in automata: Normalization is performed to standardize the grammar. By simplifying the grammar, the grammar gets simplified but does not gets standardized. This is because the RHS of productions has no specific format. Normal form brings the RHS of productions into a specific format. This helps us understand the CFGs and the properties of non-context-free languages. There are two normal forms, namely Chomsky normal form and Greibach normal form. We shall now see how to reduce an arbitrary CFG to a Chomsky normal form and Greibach normal form.

5.1 Chomsky Normal Form (CNF)

A CFG is said to be in Chomsky Normal Form if the following are true.

– Every rule is of the form
(i) $A \to BC$, or
(ii) $A \to a$,
where A, B, C are non-terminals and a is a terminal.

Note: Suppose the given CFG is not in CNF then we first simplify the given CFG followed by CNF conversion (CFG $\rightarrow$ Simplification $\rightarrow$ CNF). Note that ϵ should not be part of the given G ($\epsilon \notin L(G)$.)

1. Consider the following CFG:
   $S \rightarrow a$
   $S \rightarrow AS$
   $A \rightarrow a$
   Every production satisfies the properties of CNF. Therefore, the above CFG is in CNF.
2. Consider the following CFG:
   $S \rightarrow ASA$
   $S \rightarrow a$
   $S \rightarrow AB$
   $A \rightarrow aA$
   $A \rightarrow bB$
   $A \rightarrow \epsilon$
   $B \rightarrow \epsilon$

   We apply simplification Rule-1 and remove ϵ-rules. On substituting ϵ in the place of A and B in the above productions we get,
   $S \rightarrow ASA \mid SA \mid AS \mid S \mid A \mid B \mid \epsilon$
   $S \rightarrow a$
   $A \rightarrow aA \mid a$
   $A \rightarrow bB \mid b$

   Now apply Rule-2 and eliminate unit productions. Unit productions are $S \rightarrow S \mid A \mid B$
   $S \rightarrow ASA \mid SA \mid AS \mid \epsilon$
   $S \rightarrow aA \mid a \mid bB \mid b$
   $A \rightarrow aA \mid a$
   $A \rightarrow bB \mid b$

   Here the non-terminal symbol B is useless and we remove it. Apply Rule-3.
   The simplified CFG is :
   $S \rightarrow ASA \mid SA \mid AS \mid \epsilon$
   $S \rightarrow aA \mid a \mid b$
   $A \rightarrow aA \mid a$
   $A \rightarrow b$

   Note that the language itself contains ϵ. Therefore no CNF exist for the above CFG.
3. Consider the grammar
   $S \rightarrow aSa$
   $S \rightarrow bSb$
   $S \rightarrow a \mid b$
   Note that the given grammar itself is a simplified grammar. The first two productions are not in CNF. We now apply our CNF conversion procedure to convert them into a CNF.
   Consider $S \rightarrow aSa$
   $S \rightarrow A_1A_2$
   $A_1 \rightarrow a$
   $A_2 \rightarrow SA_3$
   $A_3 \rightarrow a$

   Consider $S \rightarrow bSb$
   $S \rightarrow B_1B_2$
   $B_1 \rightarrow b$
   $B_2 \rightarrow SB_3$

$B_3 \to b$

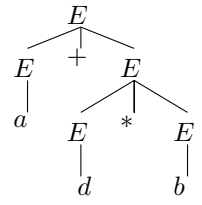
The resultant CNF is the following
$S \to A_1A_2 \mid B_1B_2 \mid a \mid b$
$A_1 \to a$
$A_2 \to SA_3$
$A_3 \to a$
$B_1 \to b$
$B_2 \to SB_3$
$B_3 \to b$
Note: Let G be the given CFG and G' be the modified CFG (CNF). The language accepted by G is same as the language accepted by G' i.e $L(G) = L(G')$.

4. Convert the following simplified CFG into CNF
$E \to E + E$
$E \to E * E$
$E \to (E)$
$E \to a \mid b \mid c \mid d$
We shall now apply CNF conversion procedure
$E \to E + E \implies E \to EE_1$
$E_1 \to E_2E$
$E_2 \to +$

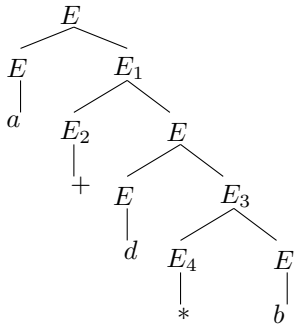
$E \to E * E \implies E \to EE_3$
$E_3 \to E_4E$
$E_4 \to *$

$E \to (E) \implies E_5 \to E_5E_6$
$E_5 \to ($
$E_6 \to EE_7$
$E_7 \to)$
$E \to a \mid b \mid c \mid d$

Observation: Let G be the given grammar, and G' be the CNF form. Consider the derivation tree for the string $a + d * b$ using G and G' shown in Figure 7. We observe that the derivation tree generated out of G is an arbitrary tree, whereas the derivation tree generated out of G' is always a binary tree.



Derivation using G



Derivation using G'

Fig. 7. *derivation tree for the string* $a + d * b$

5.2 Greibach Normal Form (GNF)

A CFG is said to be in Greibach Normal Form if every production is of the form
(i) $A \to a\alpha$, or
(ii) $A \to a$
where a is a terminal and α is a string of zero or more non-terminals. **Note:** $\epsilon \notin L(G)$.

Let us see some example of CNF and GNF
(i) $S \to a$
it is in both CNF and GNF
(ii) $S \to a$
$S \to aSAB$
GNF but not in CNF
(iii) $S \to BC$
$S \to c$
CNF but not in GNF
Note: Now we shall see how to convert CFG to GNF. Having seen CFG to CNF conversion, we will use that to convert an arbitrary CFG to GNF. I.e CFG → Simplification → CNF → GNF. Note that we can directly convert CFG to GNF as well.

1. Consider the following CFG
   $S \to BC$
   $S \to d$
   It is easy to see that the above CFG is in CNF but not in GNF. We substitute d in place of B in the production $S \to BC$.
   $S \to dC$
   $S \to d$
   Now we see that it is in GNF.
2. Consider the following CFG
   $S \to AS$
   $S \to SS$
   $A \to BC$
   $B \to d$
   $C \to c$

   We substitute d in place of B in the production $S \to BC$.
   $A \to dC$
   Now substitute A in $S \to AS$
   $S \to dCS$
   $S \to SS$
   We have two productions that start with S, $S \to dCS$ and $S \to SS$
   Here, $S \to SS$ can not be reduced to GNF. This shows that a simple substitution will not always give GNF. We shall see a technique that will convert CFG to GNF.

   Rule 1
   If $A_k \to A_j\alpha,\ j < k$, then for all $A_j \to \beta$
   Add $A_k \to \beta\alpha$ and remove $A_k \to A_j\alpha$

   Rule 2
   If $A_k \to A_k\alpha$, then add $B_k \to \alpha$ and $B_k \to \alpha B_k$
   remove $A_k \to A_k\alpha$

   Rule 3
   If $A_k \to \beta$, then add $A_k \to \beta B_k$
   β: does not begin with A_k.

3. Convert the following CFG to GNF
   $A_1 \rightarrow A_2A_3$
   $A_2 \rightarrow A_3A_1 \mid b$
   $A_3 \rightarrow A_1A_2 \mid a$

   Apply Rule-1 for $A_3 \rightarrow A_1A_2$, we get
   $A_1 \rightarrow A_2A_3$
   $A_2 \rightarrow A_3A_1 \mid b$
   $A_3 \rightarrow A_2A_3A_2 \mid a$

   Apply Rule-1 for $A_3 \rightarrow A_2A_3A_2$
   $A_1 \rightarrow A_2A_3$
   $A_2 \rightarrow A_3A_1 \mid b$
   $A_3 \rightarrow A_3A_1A_3A_2 \mid bA_3A_2 \mid a$

   Apply Rule-2 for $A_3 \rightarrow A_3A_1A_3A_2$
   Add $B_3 \rightarrow A_1A_3A_2$
   $B_3 \rightarrow A_1A_3A_2B_3$
   Remove $A_3 \rightarrow A_3A_1A_3A_2$

   After applying Rule-1 and 2 we get
   $A_1 \rightarrow A_2A_3$
   $A_2 \rightarrow A_3A_1 \mid b$
   $A_3 \rightarrow bA_3A_2 \mid a$
   $B_3 \rightarrow A_1A_3A_2$
   $B_3 \rightarrow A_1A_3A_2B_3$

   Apply Rule-3 for $A_3 \rightarrow bA_3A_2$
   $A_3 \rightarrow bA_3A_2B_3$
   Apply Rule-3 for $A_3 \rightarrow a$
   $A_3 \rightarrow aB_3$

   Resultant CFG after applying Rule-1,2 and3.
   $A_1 \rightarrow A_2A_3$
   $A_2 \rightarrow A_3A_1 \mid b$
   $A_3 \rightarrow bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$
   $B_3 \rightarrow A_1A_3A_2 \mid A_1A_3A_2B_3$

   The production $A_3 \rightarrow bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$ is in GNF. We use A_3 to convert other productions.
   Substitute A_3 in $A_2 \rightarrow A_3A_1 \implies A_2$ is in GNF
   Substitute A_2 in $A_1 \rightarrow A_2A_3 \implies A_1$ is in GNF
   Substitute A_1 in $B_3 \rightarrow A_1A_3A_2 \mid A_1A_3A_2B_3 \implies B_3$ is in GNF
   The resultant GNF is as follows;
   $A_3 \rightarrow bA_3A_2B_3 \mid aB_3 \mid bA_3A_2 \mid a$
   $A_2 \rightarrow bA_3A_2B_3A_1 \mid aB_3A_1 \mid bA_3A_2A_1 \mid aA_1$
   $A_1 \rightarrow bA_3A_2B_3A_1A_3 \mid aB_3A_1A_3 \mid bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3$
   $B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2 \mid aB_3A_1A_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_3A_2$
   $B_3 \rightarrow bA_3A_2B_3A_1A_3A_3A_2B_3 \mid aB_3A_1A_3A_3A_2B_3 \mid bA_3A_2A_1A_3A_3A_2B_3 \mid aA_1A_3A_3A_2B_3 \mid bA_3A_3A_2B_3$
   Note: The introduction of additional variables is to avoid cyclic productions which is given as part of the grammar.

6 Pumping lemma

Not all useful languages can be generated by context-free grammars, and a pushdown automaton is limited significantly by having to follow the rules of a stack in accessing its memory. The pumping lemma for context-free languages, like the pumping lemma for regular languages, describes a property that every context-free language must have, and as a result it allows us to show that certain languages are not context-free.

The principle behind pumping lemma for Regular Language (RL) is that if an input string is long enough, a finite automaton processing it will have to enter some state a second time.
$\forall i \geq 0,\ uv^iw \in L$. Figure 8, depicts the pumping lemma for regular language. The principle behind pumping

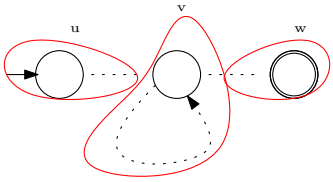


Fig. 8. Pumping lemma for regular language

lemma for Context-Free Language (CFL) is that a sufficiently long derivation in a grammar G will have to contain a self-embedded variable A; that is, the derivation (or at least one with the same derivation tree) looks like

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

$\Rightarrow^*$ means that using one or more productions we obtain the string on right. The string x can be derived from any one of these occurrences of A, so that each of the strings $vxz,\ vwxyz,\ vw^2xy^2z, \ldots$ can be derived from S.
Note that in RL, one component v^i was pumped any number of times, and in CFL, two component w^iy^i was pumped any number of times equally.
Pumping lemma for Context-free Language (CFL):
Let L be a context-free language. Then there exists an integer n, for all $u \in L$, $|u| \geq n$, u can be decomposed into five components, there exists v, w, x, y, z, u can be written as $u = vwxyz$ such that:

1. $|wy| \geq 1$ (At least one of w or y is non-empty),
2. $|wxy| \leq n$,
3. $vw^ixy^iz \in L,$ for all$i \geq 0$.

FOL for Pumping lemma for CFL: CFL $\Rightarrow \exists n \forall u(|u| \geq n \rightarrow \exists v \exists w \exists x \exists y \exists z(u = vwxyz,\ |wy| \geq 1,\ |wxy| \leq n \wedge \forall i(i \geq 0 \rightarrow vw^ixy^iz \in L)))$.
We can use negation of FOL to show that some languages are not CFL. We shall look into an example $L = \{a^nb^nc^n \mid n \geq 1\}$.
Let $u = a^nb^nc^n$, $|u| \geq n$.
u can be written as $u = vwxyz$, $|wxy| \leq n$, $|wy| \geq 1$.
Case 1: wxy is part of a^n
Consider the string $aaaa\ a^{n-5}a\ b^nc^n$ in which $v = aa,\ w = aa,\ x = a^{n-5},\ y = a,\ z = b^nc^n$. Then, $aa\ (a^{n-2})^i\ b^nc^n$. For $i = 2$, $a^2(aa)^2a^2b^nc^n \notin a^nb^nc^n$.

Case 2: wxy is part of b^n
Consider the string $a^nbbb\ b^{n-5}bc^n$ in which $v = a^n,\ w = bbbb,\ x = b^{n-5},\ y = b,\ z = c^n$. Then $a^nbbbb(b^{n-5})^ibc^n$. For $i = 2$, $a^nbbbb(b^{n-5})^2bc^n \notin L$.

Case 3: wxy is part of c^n
Consider the string $a^n b^n ccc\ c^{n-5}c$ in which $v = a^n b^n,\ w = cccc,\ x = c^{n-5},\ y = c$. Then $a^n b^n cccc(c^{n-5})^i c$. For $i = 2$, $a^n b^n cccc(c^{n-5})^2 c \notin L$.

Case 4: wxy is part of $a^n b^n$
Let $wxy = a^4 b^4$. Suppose that $w = a^2,\ x = a^2 b^2,\ y = b^2$, then for $i = 2$, $a^{n-4}(a^2)^2 a^2 b^2 (b^2)^2 a^{n-4}$ has more a's and more b's compared to c's. Hence, not in L.
Similarly, if $w = a^4,\ xy = b^4$, the string generated will have more a's compared to number of b's and c's.
Similarly, if $w = a^4,\ y = b^4$, the string generated will have more b's compared to number of a's and c's.

Case 5: wxy is part of $b^n c^n$
Similar to Case 4, count will not match with number of a's.
Note that wxy cannot span $a^n b^n c^n$. Since $|wxy| \leq n$, wxy can span at most two distinct symbols from $\Sigma = \{a, b, c\}$.

7 Equivalence of PDA and CFG

Now, we shall demonstrate that the languages defined by PDA's are exactly the context-free languages.

7.1 A PDA from given CFG

Given a CFG G, we construct a PDA that simulates the leftmost derivations of G. Any left-sentential form that is not a terminal string can be written as $xA\alpha$, where A is the leftmost variable, x is whatever terminals appear to its left, and α is the string of terminals and variables that appear to the right of A.
Approach: If $A \to \alpha$ is a rule in production, then $\delta(q, \epsilon, A)$ contains (q, α). For each α in T, $\delta(q, a, a)$ contains (q, ϵ).
Let us understand the conversion for the language $L = \{a^n b^n \mid n \geq 1\}$:
Let the given CFG be

$$S \to aSb \mid ab$$

The corresponding PDA should push 'a',and when 'b' is read pop operation is carried out. We have seen that PDA transition function is

$$\delta(q_0, a, z_0) = (q_1, Az_0)$$

$$\delta(q_1, a, A) = (q_1, AA)$$

. Here CFG and PDA are independent but they are equivalent.
For the CFG, we can construct PDA as follows;
Let us understand PDA from the given CFG;
When a non-terminal is on top, go for push. When terminal is on top, go for pop. For the string $aabb$, the push and pop operation is simulated in the stack as in Figure 9;

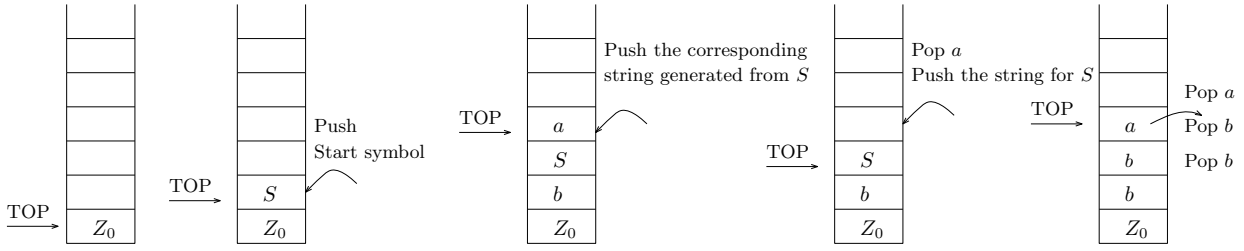


Fig. 9. PDA construction using stack

$$\delta(q_0, \epsilon, z_0) = (q_1, Sz_0)$$

For push operation:

$$\delta(q_1, \epsilon, S) = (q_1, abz_0)$$

$$\delta(q_1, \epsilon, S) = (q_1, aSbz_0)$$

Note that the above two productions are non-deterministic.

$$\delta(q_1, a, a) = (q_1, \epsilon)$$

$$\delta(q_1, b, b) = (q_1, \epsilon)$$

Note that the above two productions are non-deterministic productions corresponding to pop operation. For a string w, there may be many productions, one has to non-deterministically choose right production.

We shall look into the conversion for one more language $L = \{a^n b^m c^m d^n \mid n, m \geq 1\}$:
Let the given CFG be

$$S \rightarrow aSd \mid aAd$$

$$A \rightarrow bAc \mid bc$$

For the string *aabbbcccdd*, the push and pop operation is simulated in the stack as in Figure 9;

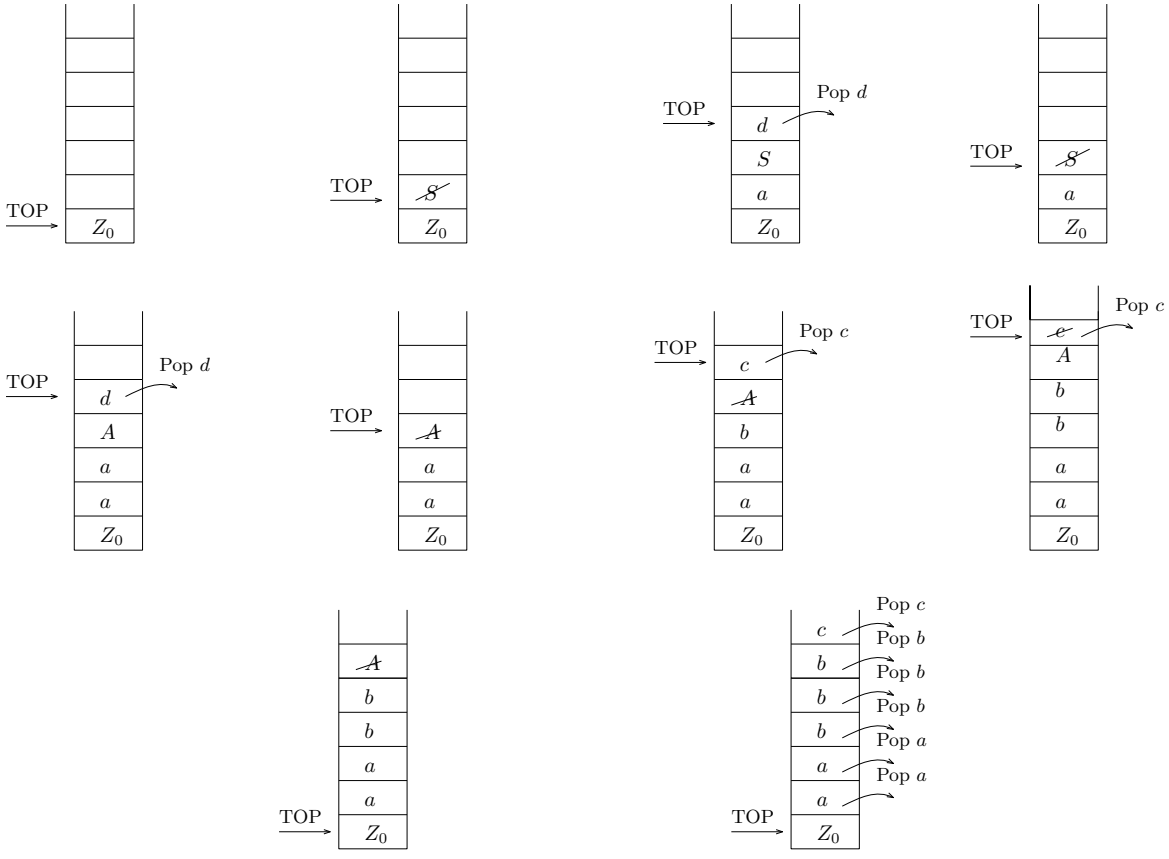


Fig. 10. PDA construction using stack

The corresponding PDA for the CFG is;

$$\delta(q_0, \epsilon, z_0) = (q_1, Sz_0)$$

The first transition is to push the start symbol.

$$\delta(q_1, \epsilon, S) = (q_1, aSdz_0)$$

$$\delta(q_1, \epsilon, S) = (q_1, aAdz_0)$$

$$\delta(q_1, \epsilon, A) = (q_1, bAc)$$

$$\delta(q_1, \epsilon, A) = (q_1, bc)$$

$$\delta(q_1, a, a) = (q_1, \epsilon)$$

$$\delta(q_1, b, b) = (q_1, \epsilon)$$

$$\delta(q_1, c, c) = (q_1, \epsilon)$$

$$\delta(q_1, d, d) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, z_0) = (q_f, z_0)$$

7.2 A CFG from given PDA

Given a PDA acceptance by empty store, we can define production for CFG as follows;
Rule 1: The start symbol is

$$[q_0, z_0, P]$$

Rule 2: If $\delta(q, a, z_0) = (p, B_1, B_2, \ldots, B_m)$, then production includes rules of the form;

$$[q, z_0, q_m] \rightarrow a[p, B_1, q_1][q_1, B_2, q_2] \ldots [q_{m-1}, B_m, q_m],\ q_i \in Q$$

Rule 3: If $\delta(q, a, z_0) = (p, \epsilon)$, then production includes

$$[q, z_0, p] \rightarrow a$$

Let us consider the following transition function of a PDA corresponding to language $L = \{a^n \mid n \geq 1\}$.

$$\delta(q_0, a, z_0) = (q_1, z_0)$$

$$\delta(q_1, a, z_0) = (q_1, z_0)$$

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$

By using the rules mentioned, we shall construct productions as follows; By using rule 1, start symbols are

$$[q_0, z_0, q_0]$$

$$[q_0, z_0, q_1]$$

By using rule 2 in $\delta(q_0, a, z_0) = (q_1, z_0)$, we obtain production as follows;

$$[q_0, z_0, q_0] \rightarrow a[q_1, z_0, q_0]$$

$$[q_0, z_0, q_1] \rightarrow a[q_1, z_0, q_1]$$

By using rule 2 in $\delta(q_1, a, z_0) = (q_1, z_0)$, we obtain production as follows;

$$[q_1, z_0, q_0] \rightarrow a[q_1, z_0, q_0]$$

$$[q_1, z_0, q_1] \rightarrow a[q_1, z_0, q_1]$$

By using rule 3 in $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$, we obtain production as follows;

$$[q_1, z_0, q_1] \to \epsilon$$

Let us derive for the string aaa; $[q_0, z_0, q_1] \to a[q_1, z_0, q_1] \to aa[q_1, z_0, q_1] \to aaa[q_1, z_0, q_1] \to aaa\epsilon = aaa$
Suppose that for deriving the string aaa; $[q_0, z_0, q_1] \to a[q_1, z_0, q_1] \to aa[q_1, z_0, q_1] \ldots$ We cannot generate any string. Hence $[q_0, z_0, q_1]$ cannot be used as start symbol.
By using simplification, we obtain CFG as;

$$[q_0, z_0, q_1]$$

$$[q_0, z_0, q_1] \to a[q_1, z_0, q_1]$$

$$[q_1, z_0, q_1] \to a[q_1, z_0, q_1]$$

$$[q_1, z_0, q_1] \to \epsilon$$

By using change of variable; $[q_0, z_0, q_1]$ as S, and $[q_1, z_0, q_1]$, the obtained CFG is

$$S$$

$$S \to aA$$

$$A \to aA$$

$$A \to \epsilon$$

We shall look into another example of converting PDA to CFG for the language $L = \{a^n b^n \mid n \geq 1\}$
The transition function of PDA is

$$\delta(q_0, a, z_0) = (q_0, Az_0)$$

$$\delta(q_0, a, A) = (q_0, AA)$$

$$\delta(q_0, b, A) = (q_1, \epsilon)$$

$$\delta(q_1, b, A) = (q_1, \epsilon)$$

$$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$$

The start symbol is

$$[q_0, z_0, q_0]$$

$$[q_0, z_0, q_1]$$

For the transition $\delta(q_0, a, z_0) = (q_0, Az_0)$, the corresponding productions are;

$$[q_0, z_0, q_0] \to a[q_0, A, q_0][q_0, z_0, q_0]$$

$$[q_0, z_0, q_1] \to a[q_0, A, q_0][q_0, z_0, q_1]$$

$$[q_0, z_0, q_0] \to a[q_0, A, q_1][q_1, z_0, q_0]$$

$$[q_0, z_0, q_1] \to a[q_0, A, q_1][q_1, z_0, q_1]$$

For the transition $\delta(q_0, a, A) = (q_0, AA)$, the corresponding productions are;

$$[q_0, A, q_0] \to a[q_0, A, q_0][q_0, A, q_0]$$

$$[q_0, A, q_1] \to a[q_0, A, q_0][q_0, A, q_1]$$

$$[q_0, A, q_0] \to a[q_0, A, q_1][q_1, A, q_0]$$

$$[q_0, A, q_1] \to a[q_0, A, q_1][q_1, A, q_1]$$

For the transition $\delta(q_0, b, A) = (q_1, \epsilon)$, the corresponding productions are;

$$[q_0, A, q_1] \to b$$

For the transition $\delta(q_1, b, A) = (q_1, \epsilon)$, the corresponding productions are;

$$[q_1, A, q_1] \to b$$

For the transition $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$, the corresponding productions are;

$$[q_1, z_0, q_1] \to \epsilon$$

By using simplification, we obtain CFG as;

$$[q_0, z_0, q_1]$$
$$[q_0, z_0, q_1] \to a[q_0, A, q_1][q_1, z_0, q_1]$$
$$[q_0, A, q_1] \to a[q_0, A, q_1][q_1, A, q_1]$$

By using change of variable; the CFG obtained as;

$$S$$
$$S \to aA$$
$$A \to aAB$$
$$A \to b$$
$$B \to b$$