Introduction to Formal Languages, Automata and Computability

K. Krithivasan and R. Rama

Language

Strings are defined over an alphabet which is finite. Alphabet may vary depending upon the application. Elements of an alphabet are called symbols. Usually we denote the basic alphabet set either as Σ or T. For example, the following are a few examples of an alphabet set.

$$\begin{aligned} \Sigma_1 &= \{a, b\} \\ \Sigma_2 &= \{0, 1, 2\} \\ \Sigma_3 &= \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}. \end{aligned}$$

contd.

A string or word is a finite sequence of symbols from the alphabet, usually written as concatenated symbols and not separated by gaps or commas. For example if $\Sigma = \{a, b\}$, a string $abbab$ is a string or word over Σ. If w is a string over an alphabet Σ, then the length of w written as $len(w)$ or $|w|$ is the number of symbols it contains. If $|w| = 0$, then w is called as empty string denoted either as λ or ϵ.

contd.

For any word w, $w\epsilon = \epsilon w = w$. For any string $w = a_1 \ldots a_n$ of length n, the reverse of w is written as w^R which is the string $a_n a_{n-1} \ldots a_1$, where each symbol a_i belongs to the basic alphabet Σ. A string z that is appearing consecutively within another string w is called a substring or subword of w. For example aab is a substring of $baabb$.

contd.

The set of all strings over an alphabet Σ is denoted by Σ^* which includes the empty string ϵ. For example for $\Sigma = \{0, 1\}$, $\Sigma^* = \{\epsilon, 0, 1, 00, 01, 10, 11, \dots\}$. Note that Σ^* is a countably infinite set. Also Σ^n denotes the set of all strings over Σ whose length is n. Hence $\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots$ and $\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \Sigma^3 \dots$. Subsets of Σ^* are called languages. For example if $\Sigma = \{a, b\}$

$$
\begin{aligned}
L_1 &= \{\epsilon, a, b\} \\
L_2 &= \{ab, aabb, aaabbb, \dots\} \\
L_3 &= \{w \in \Sigma^* / |w|_a = |w|_b\}
\end{aligned}
$$

In the above example, L_1 is finite, L_2 and L_3 are infinite languages. ϕ denotes an empty language.

contd.

Definition A set is an unordered collection of objects.

contd.

Definition A set is an unordered collection of objects.
Example Let W denote the set of well formed parentheses. It can be defined inductively as follows:
Basis clause : [ ] $\in W$
Inductive clause : if $x, y \in W$, $xy \in W$ and $[x] \in W$
Extremal clause : No object is a member of W unless its being so follows from a finite number of applications of the basis and the inductive clauses.

Language

Definition Let Σ be any alphabet set. Σ^+ is a set of nonempty strings over Σ defined as follows:

1. **Basis :** If $a \in \Sigma$, then $a \in \Sigma^+$.
2. **Induction :** If $\alpha \in \Sigma^+$ and $a \in \Sigma$, $\alpha a, a\alpha$ are in Σ^+.
3. No other element belong to Σ^+.

Clearly the set Σ^+ contains all strings of length n, $n \geq 1$.

Language

Definition Let Σ be any alphabet set. Σ^+ is a set of nonempty strings over Σ defined as follows:

1. **Basis :** If $a \in \Sigma$, then $a \in \Sigma^+$.
2. **Induction :** If $\alpha \in \Sigma^+$ and $a \in \Sigma$, $\alpha a, a\alpha$ are in Σ^+.
3. No other element belong to Σ^+.

Clearly the set Σ^+ contains all strings of length n, $n \geq 1$.

Definition Let Σ be any alphabet set. Σ^* is defined as follows:

1. **Basis :** $\epsilon \in \Sigma^*$.
2. **Induction :** If $\alpha \in \Sigma^*$, $a \in \Sigma$, then $a\alpha, \alpha a \in \Sigma^*$.
3. No other element is in Σ^*.

contd.

Since languages are sets, one can define the set-theoretic operations of union, intersection, difference, complement in the usual fashion. The following operations are also defined for languages. If $x = a_1 \ldots a_n$, $y = b_1 \ldots b_m$, the concatenation of x and y is defined as $xy = a_1 \ldots a_n b_1 \ldots b_m$. The catenation (or concatenation) of two languages L_1 and L_2 is defined by, $L_1 L_2 = \{w_1 w_2 / w_1 \in L_1 \text{ and } w_2 \in L_2\}$. Note that concatenation of languages is associative because concatenation of strings is associative. Also $L^0 = \{\epsilon\}$ and $L\phi = \phi L = \phi$, $L\epsilon = \epsilon L = L$.

contd.

The concatenation closure (Kleene closure) of a language L, in symbols L^* is defined to be the union of all powers of L:

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

$$\text{Also } L^+ = \bigcup_{i=1}^{\infty} L^i.$$

contd.

The right quotient and right derivative are the following sets respectively.

$$L_1 \backslash L_2 = \{y | yz \in L_1 \text{ for some } z \in L_2\}$$

$$\partial_z^r L = L/\{z\} = \{y/yz \in L\}$$

Similarly left quotient of a language L_1 by a language L_2 is defined by

$$L_2/L_1 = \{z | yz \in L_1 \text{ for some } y \in L_2\}.$$

The left derivative of a language L with respect to a word y is denoted as $\partial_y L$ which is equal to $\{z | yz \in L\}$.

contd.

The mirror image (or reversal) of a language is the collection of the mirror images of its words and $mir(L) = \{mir(w)/w \in L\}$ or $L^R = \{w^R/w \in L\}$.
The operations substitution and homomorphism are defined as follows.
For each symbol a of an alphabet Σ, let $\sigma(a)$ be a language over Σ_a. Also $\sigma(\epsilon) = \epsilon$, $\sigma(\alpha\beta) = \sigma(\alpha).\sigma(\beta)$ for $\alpha, \beta \in \Sigma^+$. σ is a mapping from Σ^* to 2^{V^*} where V is the union of the alphabets Σ_a, is called a substitution. For a language L over Σ, we define

$$\sigma(L) = \{\alpha/\alpha \in \sigma(\beta) \text{ for some } \beta \in L\}.$$

contd.

A substitution is ϵ-free if and only if none of the language $\sigma(a)$ contains ϵ. A family of languages is closed under substitution if and only if whenever L is in the family and σ is a substitution such that $\sigma(a)$ is in the family, then $\sigma(L)$ is also in the family.
A substitution σ such that $\sigma(a)$ consists of a single word w_a is called a homomorphism. It is called ϵ-free homomorphism if none of $\sigma(a)$ is ϵ.

Algebraically, one can see that Σ^* is a free semigroup with ϵ as its identity.

contd.

The homomorphism which is defined above agrees with the customary definition of homomorphism of one semigroup into another.
Inverse homomorphism can be defined as follows:

$$h^{-1}(w) = \{x | h(x) = w\}$$

$$h^{-1}(L) = \{x | h(x) \text{ is in } L\}.$$

It should be noted that $h(h^{-1}(L))$ need not be equal to L. Generally $h(h^{-1}(L)) \subseteq L$ and $h^{-1}(h(L)) \supseteq L$.

Grammar

Definition A phrase-structure grammar or a type 0 grammar is a 4-tuple $G = (N, T, P, S)$, where N is a finite set of nonterminal symbols called the nonterminal alphabet, T is a finite set of terminal symbols called the terminal alphabet, $S \in N$ is the start symbol and P is a set of productions (also called production rules or simply rules) of the form $u \rightarrow v$, where $u \in (N \cup T)^* N (N \cup T)^*$ and $v \in (N \cup T)^*$. Derivations are defined as follows:

If $\alpha u \beta$ is a string in $(N \cup T)^*$ and $u \rightarrow v$ is a rule in P, from $\alpha u \beta$ we get $\alpha v \beta$ by replacing u by v. This is denoted as $\alpha u \beta \Rightarrow \alpha v \beta$. $\Rightarrow$ is read as 'directly derives'.

contd.

If $\alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \ldots, \alpha_{n-1} \Rightarrow \alpha_n$, the derivation is denoted as $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$ or $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$. $\overset{*}{\Rightarrow}$ is the reflexive, transitive closure of $\Rightarrow$.

Definition The language generated by a grammar $G = (N, T, P, S)$ is the set of terminal strings derivable in the grammar from the start symbol.

$$L(G) = \{w / w \in T^*, S \overset{*}{\Rightarrow} w\}$$

Example

Consider the grammar $G = (N, T, P, S)$ where $N = \{S, A\}$, $T = \{a, b, c\}$, production rules in P are

$S \rightarrow aSc, \quad S \rightarrow aAc, \quad A \rightarrow b$

A typical derivation in the grammar is

$$\begin{aligned} S &\Rightarrow aSc \\ &\Rightarrow aaScc \\ &\Rightarrow aaaAccc \\ &\Rightarrow aaabccc \end{aligned}$$

The language generated is $L(G) = \{a^n bc^n / n \geq 1\}$.

Lengths

Definition If the rules are of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$, $\alpha, \beta \in (N \cup T)^*$, $A \in N$, $\gamma \in (N \cup T)^+$, the grammar is called context-sensitive grammar.

Lengths

Definition If the rules are of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$, $\alpha, \beta \in (N \cup T)^*$, $A \in N$, $\gamma \in (N \cup T)^+$, the grammar is called context-sensitive grammar.

Definition If in the rule $u \rightarrow v$, $|u| \leqslant |v|$, the grammar is called length increasing grammar.

Lengths

Definition If the rules are of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$, $\alpha, \beta \in (N \cup T)^*$, $A \in N$, $\gamma \in (N \cup T)^+$, the grammar is called context-sensitive grammar.

Definition If in the rule $u \rightarrow v$, $|u| \leqslant |v|$, the grammar is called length increasing grammar.

Example Let $G = (N, T, P, S)$ where $N = \{S, B\}$, $T = \{a, b, c\}$,
P has the following rules:

1. $S \rightarrow aSBc$
2. $S \rightarrow abc$
3. $cB \rightarrow Bc$
4. $bB \rightarrow bb$

contd..

Let us consider the language generated. The number appearing above $\Rightarrow$ denotes the rule being used.

$$
\begin{aligned}
S &\overset{2}{\Rightarrow} abc; \text{ here } abc \in L(G) \\
S &\overset{1}{\Rightarrow} aSBc \\
&\overset{2}{\Rightarrow} aabcBc \\
&\overset{3}{\Rightarrow} aabBcc \\
&\overset{4}{\Rightarrow} aabbcc, \quad a^2b^2c^2 \in L(G)
\end{aligned}
$$

contd.

Similarly

$$
\begin{aligned}
S &\overset{1}{\Rightarrow} aSBc \\
&\overset{1}{\Rightarrow} aaSBcBc \\
&\overset{2}{\Rightarrow} aaabcBcBc \\
&\overset{3}{\Rightarrow} aaabBccBc \\
&\overset{3}{\Rightarrow} aaabBcBcc \\
&\overset{3}{\Rightarrow} aaabBBccc \\
&\overset{4}{\Rightarrow} aaabbBccc \\
&\overset{4}{\Rightarrow} aaabbbccc, \quad a^3b^3c^3 \in L(G)
\end{aligned}
$$

contd.

In general any string of the form $a^n b^n c^n$ will be generated.

$$
\begin{aligned}
S &\overset{*}{\Rightarrow} a^{n-1}S(Bc)^{n-1} \text{ (by applying rule 1 (n-1) times)} \\
&\Rightarrow a^n bc(Bc)^{n-1} \text{ (rule 2 once)} \\
&\overset{*}{\Rightarrow} a^n bB^{n-1}c^n \text{ (by applying rule 3 } \frac{n(n-1)}{2} \text{ times)} \\
&\overset{*}{\Rightarrow} a^n b^n c^n \text{ (by applying rule 4,}(n-1) \text{ times)}
\end{aligned}
$$

Hence $L(G) = \{a^n b^n c^n / n \geq 1\}$. This is a type 1 language.

Type2 language

Definition If in a grammar, the production rules are of the form, $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$, the grammar is called a type 2 grammar or context-free grammar. The language generated is called a type 2 language or context-free language.

Type2 language

Definition If in a grammar, the production rules are of the form, $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$, the grammar is called a type 2 grammar or context-free grammar. The language generated is called a type 2 language or context-free language.

Definition If the rules are of the form $A \rightarrow \alpha B, A \rightarrow \beta, A, B \in N, \alpha, \beta \in T^*$, the grammar is called a right linear grammar or type 3 grammar and the language generated is called a type 3 language or regular set. We can even put the restriction that the rules can be of the form $A \rightarrow aB, A \rightarrow b$, where $A, B \in N, a \in T, b \in T \cup \epsilon$. This is possible because a rule $A \rightarrow a_1 \ldots a_k B$ can be split into $A \rightarrow a_1 B_1$, $B_1 \rightarrow a_2 B_2, \ldots, B_{k-1} \rightarrow a_k B$ by introducing new nonterminals $B_1, \ldots, B_k$.

Example

Let $G = (N, T, P, S)$ where $N = \{S\}, T = \{a, b\}$
P consists of the following rules.

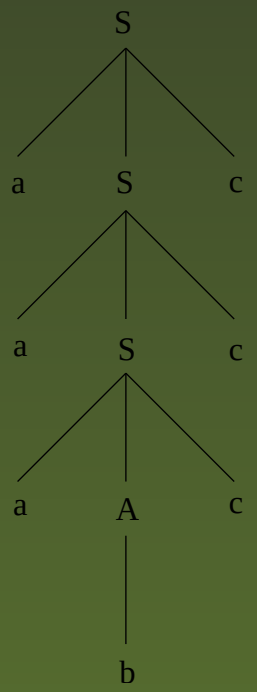
1. $S \rightarrow aS$
2. $S \rightarrow bS$
3. $S \rightarrow \epsilon$

This grammar generates all strings in T^*. For example, the string $abbaab$ is generated as follows:

$$
\begin{aligned}
S &\Rightarrow aS \text{ (rule 1)} \\
&\Rightarrow abS \text{ (rule 2)} \\
&\Rightarrow abbS \text{ (rule 2)} \\
&\Rightarrow abbaS \text{ (rule 1)} \\
&\Rightarrow abbaaS \text{ (rule 1)} \\
&\Rightarrow abbaabS \text{ (rule 2)} \\
&\Rightarrow abbaab \text{ (rule 3)}
\end{aligned}
$$

Derivation tree

We have considered the definition of a grammar and derivation. Each derivation can be represented by a tree called a derivation tree (sometimes called parse tree). A derivation tree for the derivation considered in previous example with grammar $S \rightarrow aSc, S \rightarrow aAc, A \rightarrow b$ is



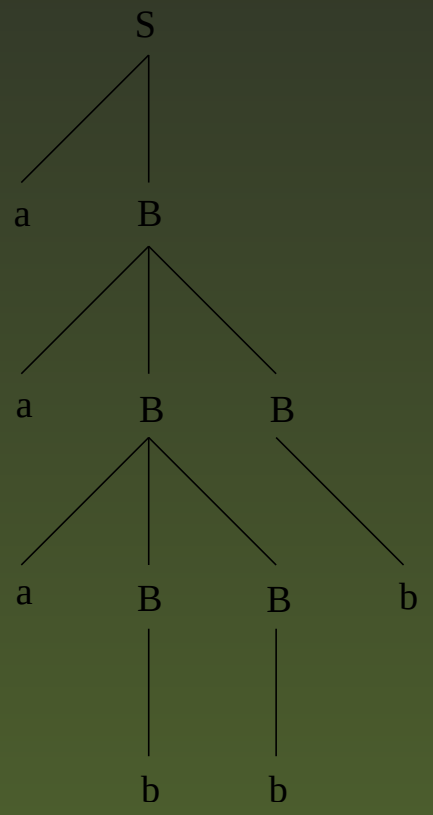
Example

Consider the following CFG, $G = (N, T, P, S)$, $N = \{S, A, B\}$, $T = \{a, b\}$. P consists of the following productions

1. $S \rightarrow aB$
2. $B \rightarrow b$
3. $B \rightarrow bS$
4. $B \rightarrow aBB$
5. $S \rightarrow bA$
6. $A \rightarrow a$
7. $A \rightarrow aS$
8. $A \rightarrow bAA$

contd..

The derivation tree for *aaabbb* is as follows,

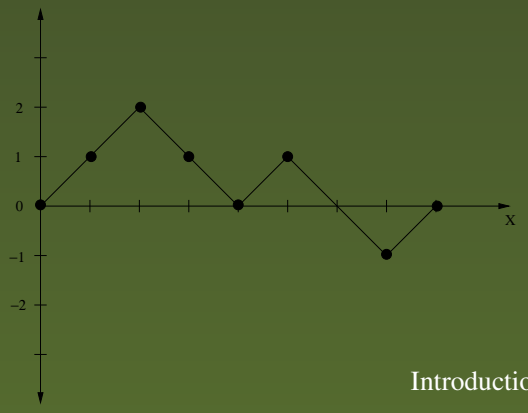


Example

Consider the grammar $G = (\{S\}, \{a, b\}, \{S \rightarrow SaSbS, S \rightarrow SbSaS, S \rightarrow \epsilon\}, S)$. The language generated by this grammar is the same as the language generated by the grammar $G = (N, T, P, S)$, $N = \{S, A, B\}$, $T = \{a, b\}$, P consists of the following productions $S \rightarrow aB, B \rightarrow b, B \rightarrow bS, B \rightarrow aBB, S \rightarrow bA, A \rightarrow a, A \rightarrow aS, A \rightarrow bAA$, except that ϵ, the empty string is also generated here.

Example

Consider the grammar $G = (\{S\}, \{a, b\}, \{S \rightarrow SaSbS, S \rightarrow SbSaS, S \rightarrow \epsilon\}, S)$. The language generated by this grammar is the same as the language generated by the grammar $G = (N, T, P, S)$, $N = \{S, A, B\}$, $T = \{a, b\}$, P consists of the following productions $S \rightarrow aB, B \rightarrow b, B \rightarrow bS, B \rightarrow aBB, S \rightarrow bA, A \rightarrow a, A \rightarrow aS, A \rightarrow bAA$, except that ϵ, the empty string is also generated here.



contd..

Consider a string w having equal number of a's and b's. We use induction.

Basis

$|w| = 0$, $S \Rightarrow \epsilon$

$|w| = 2$, it is either ab or ba, then

$S \Rightarrow SaSbS \stackrel{*}{\Rightarrow} ab$

$S \Rightarrow SbSaS \stackrel{*}{\Rightarrow} ba.$

contd..

Induction Assume that the result holds up to strings of length $k-1$. Prove that the result holds for strings of length k. Draw a graph where the x axis represents the length of the prefixes of the given string. y axis represents the number of a's - number of b's. For the string $aabbabba$, the graph will look as given in the previous figure. For a given string w with equal number of a's and b's there are 3 possibilities.

1. The string begins with 'a' and ends with 'b'.
2. The string begins with 'b' and ends with 'a'.
3. Other two cases (begins with a and ends with a, begins with b and ends with b)

contd..

In the first case $w = aw_1b$ and w_1 has equal numbers of a's and b's. So we have $S \Rightarrow SaSbS \Rightarrow aSbS \Rightarrow aSb \stackrel{*}{\Rightarrow} aw_1b$ as $S \stackrel{*}{\Rightarrow} w_1$ by inductive hypotheses. A similar argument holds for case 2. In case 3, the graph mentioned above will cross the x axis.

Consider $w = w_1w_2$ where w_1, w_2 have equal number of a's and b's. Let us say w_1 begins with a, and w_1 corresponds to the portion where the graph touches the x axis for the first time. In the above example $w_1 = aabb$ and $w_2 = abba$.

contd..

In this case we can have a derivation as follows:

$$\begin{aligned} S &\overset{1}{\Rightarrow} SaSbS \\ &\overset{3}{\Rightarrow} aSbS \\ &\overset{*}{\Rightarrow} aw_1'bS(w_1 = aw_1'b) \\ &\overset{*}{\Rightarrow} aw_1'bw_2 \\ &\overset{*}{\Rightarrow} w_1w_2. \end{aligned}$$

$S \overset{*}{\Rightarrow} w$ follows from inductive hypothesis.

CS

Theorem Every context-sensitive language is length increasing and conversely.

That every context-sensitive language is length increasing can be seen from definitions.

Every length increasing language is context-sensitive can be seen from the following construction.

Let L be a length increasing language generated by $G = (N, T, P, S)$. Without loss of generality, one can assume that the productions in P are of the form $X \to a$, $X \to X_1 \dots X_m$, $X_1 \dots X_m \to Y_1 \dots Y_n$, $2 \leq m \leq n$, $X, X_1, \dots, X_m, Y_1, \dots, Y_n \in N$, $a \in T$. Productions in P which are already context-sensitive productions are not modified. Hence consider, a production of the form

$$X_1 \dots X_m \to Y_1 \dots Y_n, \;\; 2 \leq m \leq n$$

contd..

It is replaced by the following set of context-sensitive productions:

$$
\begin{aligned}
X_1 \dots X_m &\rightarrow Z_1 X_2 \dots X_m \\
Z_1 X_2 \dots X_m &\rightarrow Z_1 Z_2 X_3 \dots X_m \\
&\vdots \\
Z_1 Z_2 \dots Z_{m-1} X_m &\rightarrow Z_1 Z_2 \dots Z_m Y_{m+1} \dots Y_n \\
Z_1 Z_2 \dots Z_m Y_{m+1} \dots Y_n &\rightarrow Y_1 Z_2 \dots Z_m Y_{m+1} \dots Y_n \\
&\vdots \\
Y_1 Y_2 \dots Y_{m-1} Z_m Y_{m+1} \dots Y_n &\rightarrow Y_1 Y_2 \dots Y_m Y_{m+1} \dots Y_n
\end{aligned}
$$

where $Z_k, 1 \leq k \leq m$ are new nonterminals.

contd..

Each production that is not context-sensitive is to be replaced by a set of context-sensitive productions as mentioned above. Application of this set of rules has the same effect as applying $X_1 \ldots X_m \rightarrow Y_1 \ldots Y_n$. Hence a new grammar G' thus obtained is context-sensitive that is equivalent to G.

Example Let $L = \{a^n b^m c^n d^m / n, m \geq 1\}$.

The type 1 grammar generating this CSL is given by $G = (N, T, P, S)$ with $N = \{S, A, B, X, Y\}$,

$T = \{a, b, c, d\}$

contd..

and P =

$$\begin{aligned} S &\rightarrow aAB|aB \\ A &\rightarrow aAX|aX \\ B &\rightarrow bBd|bYd \\ Xb &\rightarrow bX \\ XY &\rightarrow Yc \\ Y &\rightarrow c. \end{aligned}$$

contd..

Sample Derivations

$$
\begin{aligned}
S \Rightarrow aB \Rightarrow abYd &\Rightarrow abcd. \\
S \Rightarrow aB \Rightarrow abBd &\Rightarrow abbYdd \Rightarrow ab^2cd^2. \\
S \Rightarrow aAB \Rightarrow aaXB &\Rightarrow aaXbYd \\
&\Rightarrow aabXYd \\
&\Rightarrow aabYcd \\
&\Rightarrow a^2bc^2d.
\end{aligned}
$$

contd..

$$
\begin{aligned}
S \Rightarrow aAB &\Rightarrow aaAXB \\
&\Rightarrow aaaXXB \\
&\Rightarrow aaaXXbYd \\
&\Rightarrow aaaXbXYd \\
&\Rightarrow aaabXXYd \\
&\Rightarrow aaabXYcd \\
&\Rightarrow aaabYccd \\
&\Rightarrow aaabcccd.
\end{aligned}
$$

Exercise

Consider the length increasing grammar with productions P = $S \rightarrow aSBc, S \rightarrow abc, cB \rightarrow Bc, bB \rightarrow bb$. All rules except the rule $cB \rightarrow Bc$ are context-sensitive. The following grammar is a context-sensitive grammar equivalent to the above grammar.

$$\begin{aligned} S &\rightarrow aSBC \\ S &\rightarrow abc \\ C &\rightarrow c \\ CB &\rightarrow DB \\ DB &\rightarrow DC \\ DC &\rightarrow BC. \end{aligned}$$

Ambiguity

Definition Let $G = (N, T, P, S)$ be a CFG. A word w in $L(G)$ is said to be ambiguously derivable in G, if it has two or more different derivation trees in G.
Since the correspondence between derivation trees and leftmost derivations is a bijection, an equivalent definition in terms of leftmost derivations can be given.

Ambiguity

Definition Let $G = (N, T, P, S)$ be a CFG. A word w in $L(G)$ is said to be ambiguously derivable in G, if it has two or more different derivation trees in G.
Since the correspondence between derivation trees and leftmost derivations is a bijection, an equivalent definition in terms of leftmost derivations can be given.
DefinitionLet $G = (N, T, P, S)$ be a CFG. A word w in $L(G)$ is said to be ambiguously derivable in G, if it has two or more different leftmost derivations in G.

Ambiguity

Definition Let $G = (N, T, P, S)$ be a CFG. A word w in $L(G)$ is said to be ambiguously derivable in G, if it has two or more different derivation trees in G.
Since the correspondence between derivation trees and leftmost derivations is a bijection, an equivalent definition in terms of leftmost derivations can be given.
DefinitionLet $G = (N, T, P, S)$ be a CFG. A word w in $L(G)$ is said to be ambiguously derivable in G, if it has two or more different leftmost derivations in G.

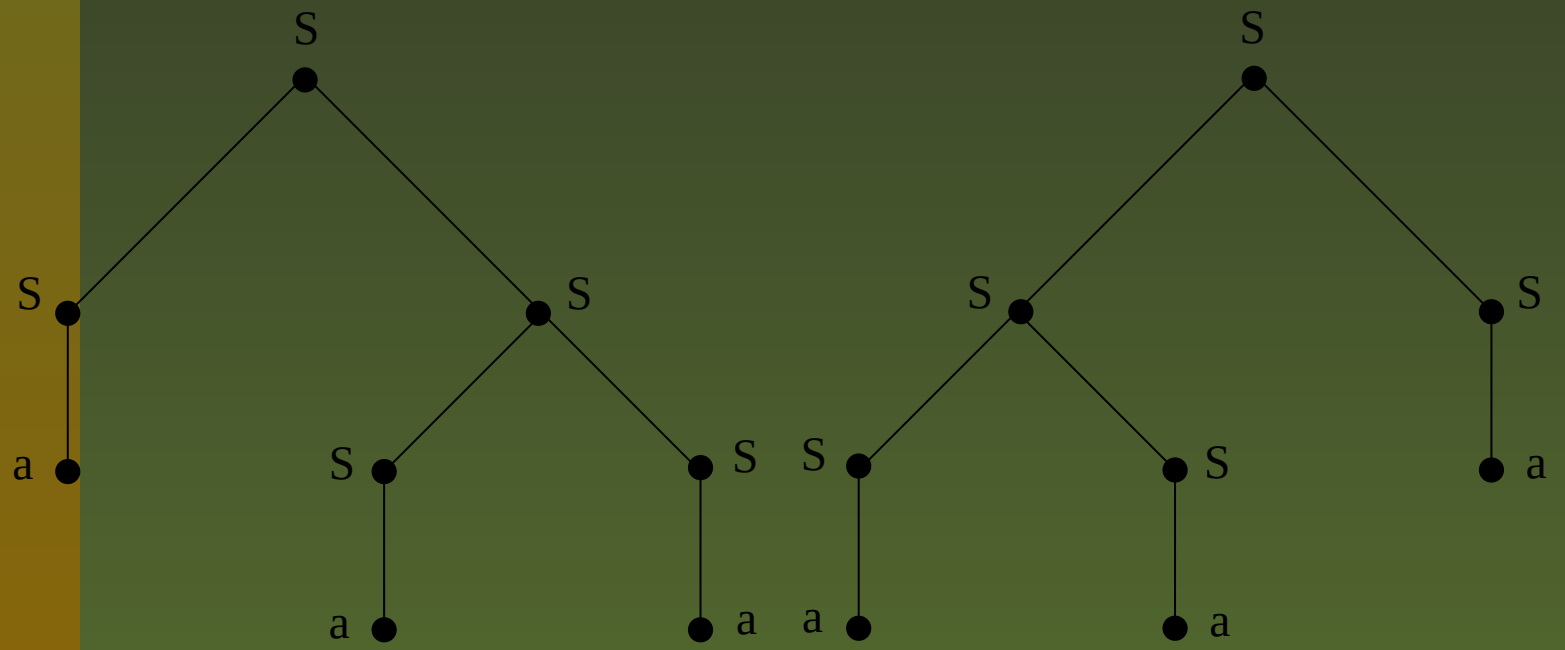
DefinitionA CFG is said to be ambiguous if there is a word w in $L(G)$ which is ambiguously derivable. Otherwise it is unambiguous.

Example

Consider the grammar G with rules $S \rightarrow SS,\ S \rightarrow a$ where S is the nonterminal and a is the terminal symbol. $L(G) = \{a^n / n \geq 1\}$.This grammar is ambiguous as a^3 has two different derivation trees as follows

Example

Consider the grammar G with rules $S \rightarrow SS,\ S \rightarrow a$ where S is the nonterminal and a is the terminal symbol. $L(G) = \{a^n / n \geq 1\}$.This grammar is ambiguous as a^3 has two different derivation trees as follows



Ambiguity contd..

Definition A CFL L is said to be inherently ambiguous if all the grammars generating it are ambiguous or in other words, there is no unambiguous grammar generating it.

Example $L = \{a^n b^n c^p / n, m, p \geq 1, n = m \text{ or } m = p\}$.

This can be looked at as $L = L_1 \cup L_2$

$$\begin{aligned} L_1 &= \{a^n b^n c^p / n, p \geq 1\} \\ L_2 &= \{a^n b^m c^m / n, m \geq 1\}. \end{aligned}$$

L_1 and L_2 can be generated individually by unambiguous grammars, but any grammar generating $L_1 \cup L_2$ will be ambiguous. Since strings of the form $a^n b^n c^n$ will have two different derivation trees, one corresponding to L_1 and another corresponding to L_2. Hence L is inherently ambiguous.

Ambiguity contd..

Definition A CFL L is said to be inherently ambiguous if all the grammars generating it are ambiguous or in other words, there is no unambiguous grammar generating it.

Example $L = \{a^n b^n c^p / n, m, p \geq 1, n = m \text{ or } m = p\}$.

This can be looked at as $L = L_1 \cup L_2$

$$\begin{aligned} L_1 &= \{a^n b^n c^p / n, p \geq 1\} \\ L_2 &= \{a^n b^m c^m / n, m \geq 1\}. \end{aligned}$$

L_1 and L_2 can be generated individually by unambiguous grammars, but any grammar generating $L_1 \cup L_2$ will be ambiguous. Since strings of the form $a^n b^n c^n$ will have two different derivation trees, one corresponding to L_1 and another corresponding to L_2. Hence L is inherently ambiguous.

contd..

Theorem It is undecidable to determine whether a given CFG G is ambiguous or not.

contd..

Theorem It is undecidable to determine whether a given CFG G is ambiguous or not.

Theorem It is undecidable to determine whether a given CFL L is ambiguous or not.

contd..

Theorem It is undecidable to determine whether a given CFG G is ambiguous or not.

Theorem It is undecidable to determine whether a given CFL L is ambiguous or not.

DefinitionA CFL L is bounded if there exists strings $w_1, \ldots, w_k$ such that $L \subseteq w_1^* w_2^* \ldots w_k^*$.

contd..

Theorem It is undecidable to determine whether a given CFG G is ambiguous or not.

Theorem It is undecidable to determine whether a given CFL L is ambiguous or not.

DefinitionA CFL L is bounded if there exists strings $w_1, \ldots, w_k$ such that $L \subseteq w_1^* w_2^* \ldots w_k^*$.

TheoremThere exists an algorithm to find out whether a given bounded CFL is inherently ambiguous or not.

contd..

Theorem It is undecidable to determine whether a given CFG G is ambiguous or not.

Theorem It is undecidable to determine whether a given CFL L is ambiguous or not.

Definition A CFL L is bounded if there exists strings $w_1, \ldots, w_k$ such that $L \subseteq w_1^* w_2^* \ldots w_k^*$.

Theorem There exists an algorithm to find out whether a given bounded CFL is inherently ambiguous or not.

Definition Let $G = (N, T, P, S)$ be a CFG then the degree of ambiguity of G is the maximum number of derivation trees a string $w \in L(G)$ can have in G.

contd.

We can also use the idea of power series and find out the number of different derivation trees a string can have. Consider the grammar with rules
$S \rightarrow SS, S \rightarrow a$ write an equation $S = SS + a$
Initial solution is $S = a$, $S_1 = a$
Use this in the equation for S on the right-hand side

$$\begin{aligned} S_2 &= S_1 S_1 + a \\ &= aa + a. \\ S_3 &= S_2 S_2 + a \\ &= (aa + a)(aa + a) + a \\ &= a^4 + 2a^3 + a^2 + a. \end{aligned}$$

contd.

$$\begin{aligned} S_4 &= S_3S_3 + a \\ &= (a^4 + 2a^3 + a^2 + a)^2 + a \\ &= a^8 + 4a^7 + 6a^6 + 6a^5 + 5a^4 + 2a^3 + a^2 + a. \end{aligned}$$

We can proceed like this using $S_i = S_{i-1}S_{i-1} + a$

In S_i, upto strings of length i, the coefficient of the string will give the number of different derivation trees it can have in G. For example, in S_4 coefficient of a^4 is 5 and a^4 has 5 different derivation trees in G. The coefficient of a^3 is 2 - the number of different derivation trees for a^3 is 2 in G.

Simplification

Definition Let $G = (N, T, P, S)$ be a CFG. A variable X in N is said to be useful if and only if there is at least a string $\alpha \in L(G)$ such that

$$S \stackrel{*}{\Rightarrow} \alpha_1 X \alpha_2 \stackrel{*}{\Rightarrow} \alpha,$$

where $\alpha_1, \alpha_2 \in (N \cup T)^*$ i.e., X is useful because it appears in at least one derivation from S to a word α in $L(G)$. Otherwise X is useless. Consequently the production involving X is useful.
One can understand the 'useful symbol' concept in two steps.

Step 1 For a symbol $X \in N$ to be useful it should occur in some derivation starting from S, i.e., $S \stackrel{*}{\Rightarrow} \alpha_1 X \alpha_2$.

contd.

Step 2 Also X has to derive a string $\alpha \in T^*$ i.e., $X \stackrel{*}{\Rightarrow} \alpha$

These two conditions are necessary. But they are not sufficient. These two conditions may be satisfied still α_1 or α_2 may contain a nonterminal from which a terminal string cannot be derived. So the usefulness of a symbol has to be tested in two steps as above.

Lemma Let $G = (N, T, P, S)$ be a CFG such that $L(G) \neq \phi$. Then there exists an equivalent context-free grammar $G' = (N', T', P', S)$ that does not contain any useless symbol or productions.

contd.

The context-free grammar G' is obtained by the following elimination procedures
I. First eliminate all those symbols X such that X does not derive any string over T^*. Let $G_2 = (N_2, T, P_2, S)$ be the grammar thus modified. As $L(G) \neq \phi$, S will not be eliminated. The following algorithm identifies symbols not to be eliminated.
Algorithm GENERATING
Step 1 Let $GEN = T$;
Step 2 If $A \rightarrow \alpha$ and every symbol of α belongs to GEN, then add A to GEN.

Remove from N all those symbols that are not in the set GEN and all the rules using them.

contd.

Let the resultant grammar be $G_2 = (N_2, T, P_2, S)$.

II. Now eliminate all symbols in the grammar G_2 that are not occurring in any derivation from 'S'. i.e., $S \stackrel{*}{\nRightarrow} \alpha_1 X \alpha_2$.

Algorithm REACHABLE

Let $REACH = \{S\}$.

If $A \in REACH$ and $A \rightarrow \alpha \in P$ then every symbol of α is in $REACH$.

The above algorithm terminates as any grammar has only finite set of rules. It collects all those symbols which are reachable from S through derivations from S.

Now in G_2 remove all those symbols that are not in $REACH$ and also productions involving them. Hence one gets the modified grammar $G' = (N', T', P', S)$, a new CFG.

contd.

without useless symbols and productions using them. The equivalence of G with G' can be easily seen since only symbols and productions leading to derivation of terminal strings from S are present in G'. Hence $L(G) = L(G')$.

Theorem Given a CFG $G = (N, T, P, S)$. Procedure I of the previous lemma is executed to get $G_2 = (N_2, T, P_2, S)$ and procedure II of previous lemma is executed to get $G' = (N', T', P', S)$. Then G' contains no useless symbols.

Suppose G' contains a symbol X (say) which is useless. It is easily seen that $N' \subseteq N_2$, $T' \subseteq T$, $P' \subseteq P_2$.

contd.

Since X is obtained after execution of II, $S \stackrel{*}{\Rightarrow} \alpha_1 X \alpha_2, \alpha_1, \alpha_2 \in (N' \cup T')^*$. Every symbol of N' is also in N_2. Since G_2 is obtained by execution of I, it is possible to get a terminal string from every symbol of N_2 and hence from N' : $\alpha_1 \stackrel{*}{\Rightarrow} w_1$ and $\alpha_2 \stackrel{*}{\Rightarrow} w_2, w_1, w_2 \in T^*$. Thus $S \stackrel{*}{\Rightarrow} \alpha_1 X \alpha_2 \stackrel{*}{\Rightarrow} w_1 X w_2 \stackrel{*}{\Rightarrow} w_1 w w_2$.
Clearly S is not useless as supposed. Hence G' contains only useful symbols.

Example

Let $G = (N, T, P, S)$ be a CFG with
$N = \{S, A, B, C\}$
$T = \{a, b\}$ and
$P = \{S \rightarrow Sa|A|C, A \rightarrow a, B \rightarrow bb, C \rightarrow aC|B\}$
First 'GEN' set will be $\{S, A, B, a, b\}$. Then
$G_2 = (N_2, T, P_2, S)$ where $N_2 = \{S, A, B\}$
$P = \{S \rightarrow Sa|A, A \rightarrow a, B \rightarrow bb\}$
In G_2, $REACH$ set will be $\{S, A, a\}$. Hence
$G' = (N', T', P', S)$ where
$N' = \{S, A\}$
$T' = \{a\}$
$P' = \{S \rightarrow Sa|A, A \rightarrow a\}$
$L(G) = L(G') = \{a^n | n \geq 1\}$.

ϵ-rule Elimination

Definition Any production of the form $A \rightarrow \epsilon$ is called a ϵ-rule. If $A \stackrel{*}{\Rightarrow} \epsilon$, then we call A a nullable symbol.

Theorem Let $G = (N, T, P, S)$ be a CFG such that $\epsilon \notin L(G)$. Then there exists a CFG without ϵ-rules generating $L(G)$.

Before modifying the grammar one has to identify the set of null-able symbols of G. This is done by the following procedure.

Algorithm NULL

1. Let $NULL := \phi$,
2. If $A \rightarrow \epsilon \in P$, then $A \in NULL$,
3. If $A \rightarrow B_1 \ldots B_t \in P$, and each B_i is in $NULL$, then $A \in NULL$

contd.

Run algorithm $NULL$ for G and get the $NULL$ set. The modification of G to get $G' = (N, T, P', S)$ with respect to $NULL$ is given below.
If $A \rightarrow A_1 \ldots A_t \in P, t \geq 1$ and if n $(n < t)$ of these A_is are in $NULL$. Then P will contain 2^n rules where the variables in $NULL$ are either present or absent in all possible combinations. If $n = t$ then remove $A \rightarrow \epsilon$ from P. The grammar $G' = (N, T, P', S)$ thus obtained is ϵ-free. To prove that a word $w \in L(G)$ if and only if $w \in L(G')$. As G and G' do not differ in N and T, one can equivalently show that,

$$A \stackrel{*}{\Rightarrow}_G w$$

contd.

if and only if

$$A \stackrel{*}{\Rightarrow}_{G'} w$$

for $A \in N$. Clearly $w \neq \epsilon$. Let $A \stackrel{n}{\Rightarrow}_G w$. $n = 1$, $A \stackrel{*}{\Rightarrow}_G w$ then $A \rightarrow w$ is in P and hence in P'. Then $A \stackrel{*}{\Rightarrow}_{G'} w$. Assume the result to be true for all derivations from A of length $n-1$. Let $A \stackrel{n}{\Rightarrow}_G w$ i.e., $A \underset{G}{\Rightarrow} Y_1, Y_2, \ldots, Y_k \stackrel{n-1}{\Longrightarrow}_G w$. Let $w = w_1 \ldots w_k$ and let $X_1, X_2, \ldots, X_m$ be those Y_j's in order such that $Y_j \stackrel{*}{\Rightarrow} w_j, w_j \neq \epsilon$. Clearly $k \geq 1$ as $w \neq \epsilon$. Hence $A \rightarrow X_1 \ldots X_m$ is a rule in G'.

contd.

We can see that $X_1 \ldots X_m \stackrel{*}{\Rightarrow}_G w$ as some Y_j derive only ϵ. Since each $Y_j \stackrel{*}{\Rightarrow} w_j$, w_j takes fewer than n steps, by induction, $Y_j \stackrel{*}{\Rightarrow} w_j$, for $w_j \neq \epsilon$. Hence $A \Rightarrow_G X_1 \ldots X_m \stackrel{*}{\Rightarrow} w$.

Conversely if $A \stackrel{*}{\Rightarrow}_{G'} w$. Then to show that $A \stackrel{*}{\Rightarrow}_G w$ also. Again the proof is by induction on the number of derivation steps.

Basis If $A \underset{G'}{\Rightarrow} w$, then $A \rightarrow w \in G'$. By the construction of G', one can see that there exists a rule $A \rightarrow \alpha$ in G such that α and w differ only in nullable symbols. Hence $A \underset{G}{\Rightarrow} \alpha \stackrel{*}{\Rightarrow}_G w$ where for $\alpha \stackrel{*}{\Rightarrow} w$ only ϵ-rules are used.

contd.

Induction

Assume $A \stackrel{n}{\Rightarrow}_{G'} w$ $n > 1$. Then let $A \underset{G'}{\Rightarrow} Y_1 \ldots Y_k \stackrel{*}{\Rightarrow}_G w$. For $A \underset{G'}{\Rightarrow} Y_1 \ldots Y_k$ the corresponding equivalent derivation by G will be $A \underset{G'}{\Rightarrow} X_1 \ldots X_m \stackrel{*}{\Rightarrow}_G Y_1 \ldots Y_k$ as some of the X_i' are nullable. Hence $A \stackrel{*}{\Rightarrow}_G Y_1 \ldots Y_k \stackrel{*}{\Rightarrow}_G w_1 \ldots w_k = w$ by induction hypothesis. Hence $A \stackrel{*}{\Rightarrow}_G w$. Hence $L(G) = L(G')$.

Example

Consider $G = (N, T, P, S)$ where
$N = \{S, A, B, C, D\}$, $T = \{0, 1\}$ and
$P = \{S \rightarrow AB0C, A \rightarrow BC, B \rightarrow 1|\epsilon, C \rightarrow D|\epsilon,$
$D \rightarrow \epsilon\}$
The set $NULL = \{A, B, C, D\}$.
Then G' will be with
$P' = \{S \rightarrow AB0C|AB0|A0C|B0C|A0|B0|0C|0,$
$A \rightarrow BC|B|C, B \rightarrow 1, C \rightarrow D\}$.

Procedure to Eliminate Unit Rules

Definition Any rule of the form $X \rightarrow Y, X, Y \in N$ is called a unit rule. Note that $A \rightarrow a$ with $a \in T$ is not a unit rule.

In simplification of context-free grammars another important step is to make the given context-free grammar unit rule free i.e., to eliminate unit rules. This is essential for the application in compilers. As the compiler spends time on each rule used in parsing by generating semantic routines, having unnecessary unit rules will increase compiler time.

Lemma Let G be a CFG, then there exists a CFG G' without unit rules such that $L(G) = L(G')$.

contd.

Let $G = (N, T, P, S)$ be a CFG. First find the sets of pairs of nonterminals (A, B) in G such that $A \stackrel{*}{\Rightarrow} B$ by unit rules. Such a pair is called a unit-pair.

Algorithm UNIT-PAIR

1. $(A, A) \in UNIT - PAIR$, for every variable $A \in N$.
2. If$(A, B) \in UNIT - PAIR$ and $B \rightarrow C \in P$ then $(A, C) \in UNIT - PAIR$

Now construct $G' = (N, T, P', S)$ as follows. Remove all unit productions. For every unit-pair (X, Y), if $Y \rightarrow \alpha \in P$ is a nonunit rule, add to P', $X \rightarrow \alpha$. Thus G' has no unit rules.

contd.

Let us consider leftmost derivations in G and G'. If $w \in L(G')$, then $S \Rightarrow^*_{G'} w$. Clearly there exists a derivation of w from S by G where zero or more applications of unit rules are used. Hence $S \Rightarrow^*_G w$ whose length may be different from $S \Rightarrow^*_{G'} w$.

If $w \in L(G)$, then one can consider $S \Rightarrow^*_{G'} w$ by G. A sequence of unit-productions applied in this derivation is to be written by a nonunit production. Since this is a leftmost derivation, for such sequence there exists one rule in G' doing the same job. Hence one can see the simulation of a derivation of G with G'.

contd.

Hence $L(G) = L(G')$.

Example Let $G = (N, T, P, S)$ be a CFG where $N = \{X, Y\}, T = \{a, b\}$ and $P = \{X \to aX|Y|b, Y \to bK|K|b, K \to a\}$

$UNIT - PAIR =$

$\{(X, X), (Y, Y), (K, K), (X, Y), (Y, K), (X, K)\}$

Then $G' = (N, T, P', S)$ where

$P' = \{X \to aX|bK|b|a, Y \to bK|b|a, K \to a\}$.

Remark Since removal of ϵ-rule can introduce unit productions, to get a simplified CFG to generate $L(G) - \{\epsilon\}$, the following steps have to be used in the order given.

contd.

1. Remove ϵ-rules
2. Remove unit-rules
3. Remove useless symbols.
   (i) Remove symbols not deriving terminal strings.
   (ii) Remove symbols not reachable from S.

Example It is essential that steps 3(i) and 3(ii) have to be executed in that order. If step 3(ii) is executed first and then step 3(i), we may not get the required reduced grammar. Consider the CFG
$G = (N, T, P, S)$ where $N = \{S, A, B, C\}$,
$T = \{a, b\}$ and $P = \{S \rightarrow ABC|a, A \rightarrow a, C \rightarrow b\}$
$L(G) = \{a\}$

contd.

Applying step 3(ii) first removes nothing. Then apply step 3(i) which removes B and $S \rightarrow ABC$ leaving $S \rightarrow a$, $A \rightarrow a$, $C \rightarrow b$. Though A and C do not contribute to the set $L(G)$, they are not removed.

On the other hand applying step 3(i) first, removes B, $S \rightarrow ABC$. Afterwards apply step 3(ii), removes $A, C, A \rightarrow a$, $C \rightarrow b$. Hence $S \rightarrow a$ is the only rule left which is the required result.

Normal form

The most popular normal forms are Weak Chomsky Normal Form, Chomsky Normal Form, Strong Chomsky Normal Form, Greibach Normal Form.

Definition Let $G = (N, T, P, S)$ be a CFG. If each rule in P is of the form $A \rightarrow \Delta$, $A \rightarrow a$ or $A \rightarrow \epsilon$, where $A \in N$, $\Delta \in N^+$, $a \in T$, then G is said to be in Weak Chomsky Normal Form (WCNF).

Example Let $G = (N, T, P, S)$ be a CFG where $N = \{S, A, B\}$, $T = \{a, b\}$ and $P = \{S \rightarrow ASB|AB, A \rightarrow a, B \rightarrow b\}$. G is in WCNF.

contd.

Theorem For any CFG $G = (N, T, P, S)$ there exists a CFG G' in WCNF such that $L(G) = L(G')$.
Let $G = (N, T, P, S)$ be a CFG. One can construct an equivalent CFG in WCNF as below. Let $G' = (N', T', P', S')$ be an equivalent CFG where $N' = N \cup \{A_a / a \in T\}$, none of A_a's belong to N. $P' = \{A \to \Delta | A \to \alpha \in P$ and every occurrence of a symbol from T present in α is replaced by A_a, giving $\Delta\} \cup \{A_a \to a | a \in T\}$. Clearly $\Delta \in N'^{+}$ and P' gets the required form. G' is in WCNF. That G and G' equivalent can be seen easily.

Chomsky Normal Form

Definition Let $\epsilon \notin L(G)$ and $G = (N, T, P, S)$ be a CFG. G is said to be in Chomsky Normal Form (CNF) if all its productions are of the form $A \to BC$ or $A \to a$, $A, B, C \in N, a \in T$.

Example The following CFGs are in CNF.

1. $G_1 = (N, T, P, S)$ where $N = \{S, A, B, C\}$, $T = \{0, 1\}$ and $P = \{S \to AB|AC|SS, C \to SA, A \to 0, B \to 1\}$.
2. $G_2 = (N, T, P, S)$ where $N = \{S, A, B, C\}$, $T = \{a, b\}$ and $P = \{S \to AS|SB, A \to AB|a, B \to b\}$.

No CFG in CNF can generate ϵ. If ϵ is to be added to $L(G)$, then a new start symbol S' is to be taken and $S' \to \epsilon$ should be added. For every rule $S \to \alpha$, $S' \to \alpha$ should be added which make sure that the new start symbol does not appear on the right-hand side of any production.

contd.

Theorem Given CFG G, there exists an equivalent CFG G'' in CNF.

Let $G = (N, T, P, S)$ be a CFG without ϵ rules, unit-rules and useless symbols and also $\epsilon \notin L(G)$. Modify G to G' such that $G' = (N', T, P', S)$ is in WCNF. Let $A \rightarrow \Delta \in P$. If $|\Delta| = 2$, such rules need not be modified. If $|\Delta| \geq 3$, the modification is as below: If $A \rightarrow \Delta = A_1 A_2 A_3$, the new set of equivalent rules will be:

$$A \rightarrow A_1 B_1$$

$$B_1 \rightarrow A_2 A_3.$$

Similarly if $A \rightarrow A_1 A_2 \ldots A_n \in P$, it is replaced by

contd.

$$A \rightarrow A_1 B_1$$

$$B_1 \rightarrow A_2 B_2$$

$$\vdots$$

$$B_{n-2} \rightarrow A_{n-1} A_n.$$

Let P'' be the collection of modified rules and $G'' = (N'', T, P'', S)$ be the modified grammar which is clearly in CNF. Also $L(G) = L(G'')$.

contd.

Example Let $G = (N, T, P, S)$ be a CFG where
$N = \{S, A, B\}, T = \{a, b\}$
$P = \{S \rightarrow SAB|AB|SBC, A \rightarrow AB|a,$
$B \rightarrow BAB|b, C \rightarrow b\}.$
Clearly G is not in CNF but in WCNF. Hence the modification of rules in P are as below:
For $S \rightarrow SAB$, the equivalent rules are
$S \rightarrow SB_1, B_1 \rightarrow AB.$
For $S \rightarrow SBC$, the equivalent rules are
$S \rightarrow SB_2, B_2 \rightarrow BC.$
For $B \rightarrow BAB$, the equivalent rules are
$B \rightarrow BB_3, B_3 \rightarrow AB.$

contd.

Hence $G'' = (N'', T, P'', S)$ will be with
$N'' = \{S, A, B, C, B_1, B_2, B_3\}, T = \{a, b\}$

$$P'' = \{S \to SB_1|SB_2|SA, B_1 \to AB, B_2 \to BC, B_3 \to AB,$$
$$A \to AB|a, B \to BB_3|b, C \to b\}.$$

Clearly G'' is in CNF.

Strong Chomsky Normal Form

Definition A CFG $G = (N, T, P, S)$ is said to be in Strong Chomsky Normal Form (SCNF) when rules in P are only of the forms $A \rightarrow a$, $A \rightarrow BC$ where $A, B, C \in N$, $a \in T$ subject to the following conditions:
(i) if $A \rightarrow BC \in P$, then $B \neq C$.
(ii) if $A \rightarrow BC \in P$, then for each rule $X \rightarrow DE \in P$, we have $E \neq B$ and $D \neq C$.

contd.

Theorem For every CFG $G = (N, T, P, S)$ there exists an equivalent CFG in SCNF.

Let $G = (N, T, P, S)$ be a CFG in CNF. One can construct an equivalent CFG.

$G' = (N', T, P', S')$ in SCNF as below.

$N' = \{S'\} \cup \{A_L, A_R | A \in N\}$

$T = T$

contd.

$$\begin{aligned} P' = &\{A_L \to B_L C_R, A_R \to B_L C_R | A \to BC \in P\} \\ &\cup \{S' \to X_L Y_R | S \to XY \in P\} \\ &\cup \{S' \to a | S \to a \in P\} \\ &\cup \{A_L \to a, A_R \to a | A \to a \in P, a \in T\}. \end{aligned}$$

Clearly $L(G) = L(G')$ and G' is in SCNF.

contd.

Example Let $G = (N, T, P, S)$ be a CFG where $N = \{S, A, B\}, T = \{0, 1\}$ and $P = \{S \to AB|0, B \to BA|1, A \to AB|0\}$. Then $G' = (N', T, P', S')$ in SCNF will be with

$$\begin{aligned} N' &= \{S', S_L, S_R, A_L, A_R, B_L, B_R\} \\ T &= \{0, 1\} \\ P' &= \{S' \to A_L B_R|0, S_L \to A_L B_R|0, \\ &\quad S_R \to A_L B_R|0, A_L \to A_L B_R|0, \\ &\quad A_R \to A_L B_R|0, B_L \to B_L A_R|1, \\ &\quad B_R \to B_L A_R|1\}. \end{aligned}$$

Greibach Normal Form

Definition Let $\epsilon \notin L(G)$ and $G = (N, T, P, S)$ be a CFG. G is said to be in Greibach Normal Form (GNF), if each rule in P rewrites a variable into a word in TN^* i.e., each rule will be of the form $A \rightarrow a\alpha$, $a \in T$, $\alpha \in N^*$.