9.12 Noam Chomsky's Classification of Grammars

Grammars are categorised by the *productions* that define them. The *four types* of grammars, due to *Noam Chomsky* (who developed the theory of formal languages) , are referred to as *'the chomsky hierarchy of grammars'*.

Let $G = (V, T, P, S)$ be a grammar. Let $A, B \in V$ and $\alpha, \alpha', \beta \in (V \cup T)^*$. Here, α, α' and β' could be the null word.

9.12.1 *Phrase-structured grammar*

- Any *phrase-structured grammar* (or unrestricted grammar) is of **type 0**. G is said to be of type 0, if all the productions are of the form $\alpha \rightarrow \beta$, where $\alpha \in (V \cup T)^+$ and $\beta \in (V \cup T)^*$.
- Clearly, it is seen that α cannot be $\in$, which means that no $\in$ can be on the left side of any production. However, $\in$ can appear on the right-hand side of any production.
- A language $L(G)$ is *recursively enumerable (or type 0)*, if the grammar G is of *type 0*.
- The language recogniser in this case is *Turing Machine.*

EXAMPLE 9.12.1: (Type 0 grammar)

$$S \rightarrow aBb \mid \in$$
$$ab \rightarrow bBB$$
$$bB \rightarrow ab$$

9.12.2 *Context-sensitive grammar*

- G is *context-sensitive* (or **type 1**), if every production is of the form $\alpha\beta$, where $\alpha, \beta \in (V \cup T)^+$.
- Clearly, it is seen that α and β cannot be $\in$, which means that no $\in$ appears on the left- and right-hand sides of any production. Thus, this grammar is $\in$-free.
- A language $L(G)$ is *context-sensitive* (or *type 1*), if the grammar G is context sensitive.
- The language recogniser in this case is *linear-bounded automata.*

EXAMPLE 9.12.2: (Type-1 grammar)

$$S \rightarrow aBb$$
$$aB \rightarrow bBB$$
$$bB \rightarrow ab$$

9.12.3 *Context-free grammar*

- G is *context-free* (or **type 2**), if every production is of the form $A \rightarrow \alpha$, where $\alpha \in (V \cup T)^*$ and A is non-terminal.
- In a context-free grammar, the LHS of every production is a single non-terminal symbol A, which α can replace.
- Clearly it is seen that, α can have $\in$, only on the right-hand side of any production.
- A language $L(G)$ is *context-free*, if the grammar G is context-free (or type 2).
- The language recogniser, in this case, is *Pushdown Automata.*

EXAMPLE 9.12.3: (Typo-2 grammar)

$$S \rightarrow asa$$
$$S \rightarrow bsb$$
$$S \rightarrow a|b|\in$$

9.12.4 *Regular grammar*

- G is a *regular grammar* (or **type 3**) iff, the grammar is *right or left-linear*. In other words, every production is of the from $A \rightarrow t$ or $A \rightarrow tB$, where $t \in T$. That is, the L.H.S. of every production consists of a single non-terminal symbol A and the R.H.S. consists of a terminal symbol t or a terminal symbol t followed by a non-terminal symbol B.
- A language $L(G)$ is *regular*, if the grammar G is regular (or *type 3*).
- The language recogniser in this case is *finite automaton*.

EXAMPLE 9.12.4: (Type-3 grammar)

a. $S \rightarrow abS$
$S \rightarrow a$

b. $S \rightarrow Aab$
$A \rightarrow Aab|B$
$B \rightarrow a$

A regular grammar is aslo context-free and a context-free grammar is also *context-sensitive*. The Venn diagram in figure 9.28 clearly shows the *Chomsky hierarchy* of the various grammars.

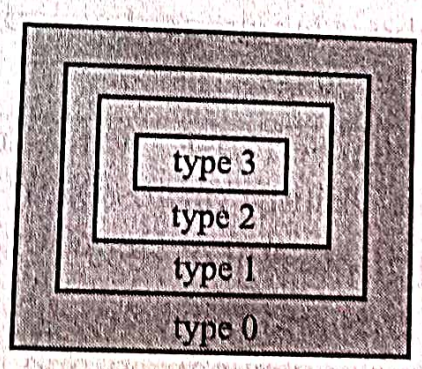


Figure 9.28. *Chomsky hierarchy of Grammars*

The following examples clarify the above definitions:

EXAMPLE 9.12.5: Let $G = (V, T, P, S)$, where $V = \{A, B, S\}$, $T = \{a, b\}$ and $P = \{S \to aA, A \to bA, A \to a\}$.

Here, every production is $A \to t$ or $A \to tB$.

Hence, G is a regular grammar and the production A is recursive. Consequently, $L(G) = \{ab^n a | n \geq 0\}$ is a regular language.

EXAMPLE 9.12.6: Let $G = (V, T, P, S)$, where $V = \{A, S\}$, $T = \{a, b\}$, and $P = \{S \to aS, S \to Aa, A \to b\}$.

Here, the R.H.S. of the production, $S \to Aa$, contains the terminal symbol a on the right of the non-terminal symbol A. Hence, G is not regular.

However, every production is of the form $w \to \alpha$, where $w \in V$ and $\alpha \in (V \cup T)^*$. So, G is context-free and $L(G)$ is a context-free language.

We further discuss the chomsky hierarchy of grammar, in chapter 15, in detail.

Avram Noam Chomsky (1928), a linguist, writer, and a political activist, was born in Philadelphia, as the son of a Hebrew schlor. At 10, he proofread the manuscrift of his father's edition of a thirteenth century Hebrew grammar.

On graduating from Central High School in Philadelphia in 1945, Chomsky entered the University of Philadelphia and received his B.A in 1949 and M.A two years later.

Chomsky received his P.hD in linguistics from the University of Pennsylvania in 1955 and joined the faculty at the Massachusetts Institute of Technology.

His first book, *Syntactic Structure* (1957), developed from his notes for an introductory course in linguistics, triggered the Chomskyan revolution in linguistics by disputing traditional ideas about language development. Chomsky is considered the *father of the theory of formal languages*.

In 1966, Chomsky became the Ferrari P. Ward Professor of Modern Language and Linguistics. He had been a visiting professor at Columbia, Princeton and the University of California at Los Angels and at Berkeley.

A recipient of numerous awards and honorary degrees, includng the Kyoto prize in Basic Sciences in 1988, Chomsky was named one of the thousand 'makers of the twentieth century' by the *London Times*.

9 Context-Free Grammars and Context-Free Languages

'Grammar is the mathematics of a language and mathematics is the grammar of creation.'

Introduction

In any language such as English, Hindi or Sanskrit, words can be combined in several ways. Naturally, some combinations form valid sentences, while others do not. The validity of a sentence is determined by the *grammar* of a language, which comprises a set of *rules*. For instance, 'The boy prepares tea quickly', although meaningless, is a perfectly legal sentence. In other words, the sentences in a language may be nonsensical, but they must obey the rules of grammar. The discussion in this chapter deals with only the *syntax* of sentence (the way the words are combined) and not with the *semantics* of sentences (meaning).

In the previous chapters, two different (though equivalent) methods: *finite automata* and *regular expressions* were introduced for describing languages. These methods have their own limitations in the sense that some simple languages, such as $\{0^n1^n | n \geq 0\}$, cannot be described by these methods. Formal languages and grammars are widely used in connection with programming languages. During programming, we proceed with an intuitive knowledge of the languages, which leads to errors. Therefore, a precise description of the language is needed, at almost every step, which helps to understand the syntax diagrams found in programming texts. Among the ways in which programming languages can be defined precisely, *grammars* or *context-free grammars* are most widely used. This method happens to be a very powerful method and such grammars can describe certain features which have a recursive structure. *Context-free-grammars* (CFG) were first used in the study of human languages. Actually, the understanding of the relationship of terms such as noun, verb, preposition and their respective phrases, leads to a natural recursive process (because of the possibility of appearance of noun phrases inside verb phrases and vice versa). CFG are capable of capturing important aspects of these relationships.

Now, an important application of CFG could be found in the specification and compilation of programming languages. A grammar for a programming language

a. facilitates the learning process of the language syntax, and
b. provides reference for the designers of compilers and interpreters.

Most compilers and interpreters contain a component, called a *parser*, that extracts the meaning of a program, prior to its execution. Construction of such a parser is possible only through CFG. Further, the collection of languages (associated with CFG) is called context-free language (CFL), which includes all the regular languages as well as some additional languages. In this chapter, a formal definition and properties of CFG and CFL are discussed.

9.1 Some Important Illustrations

9.1.1 *CFG in programming languages*

The grammar that describes a typical language like PASCAL is very extensive. Hence, a smaller language (i.e. a part of it) is considered.

For the set of all legal identifiers in PASCAL (which is a language), the grammar is as follows:

$$\begin{aligned}
\text{<id>} &\longrightarrow \text{<letter>}\ \text{<rest>} \\
\text{<rest>} &\longrightarrow \text{<letter>}\ \text{<rest>} \mid \text{<digit>}\ \text{<rest>} \mid \in \\
\text{<letter>} &\longrightarrow a|b|\cdots|z \\
\text{<digit>} &\longrightarrow 0|1|\cdots|9
\end{aligned}$$

In this grammar, the variables are $< id >$, $< letter >$, $< digit >$ and $< rest >$, with $a, b, \ldots, z, 0, 1, \ldots, 9$ as the terminals.

The derivation of the identifier a_0 is

$$\begin{aligned}
\text{<id>} &\Rightarrow \text{<letter>}\ \text{<rest>} \\
&\Rightarrow a\ \text{<rest>} \\
&\Rightarrow a\ \text{<digit>}\ \text{<rest>} \\
&\Rightarrow a_0\ \text{<rest>} \\
&\Rightarrow a_0.
\end{aligned}$$

The transition diagram for the above derivation is:

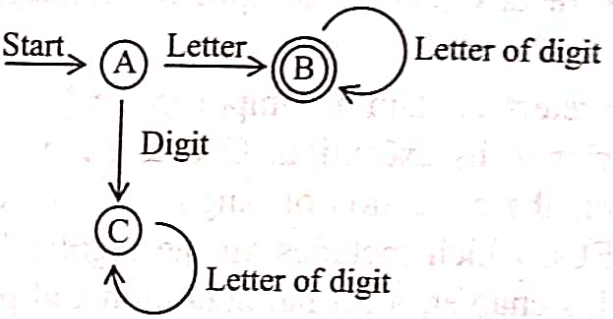


Figure 9.1. *Transition diagram.*

9.1.2 *CFG in English language*

A language is meaningful, only if a grammar is used to derive the language.

For example, English grammar has rules for constructing sentences as enlisted below:

- A *sentence* can be a *subject* followed by a *predicate*.
- A *subject* can be a *noun-phrase*.
- A *noun-phrase* can be an *adjective*, followed by a *noun-phrase*.
- A *noun-phrase* can be an *article*, followed by a *noun-phrase*.
- A *noun-phrase* can be a *noun*.
- A *predicate* can be a *verb* or a verb followed by a *noun-phrase*.
- A *noun* can be: person, fish, stapler, book, bird, dog.
- A *verb* can be: reads, touches, grabs, eats, sings, bark.
- An *adjective* can be: big, small, beautiful, wonderful.
- An *article* can be: the, a, an.

The symbolic representations of these rules are:

$$\begin{aligned}
\langle sentence \rangle &\longrightarrow \langle subject \rangle \langle predicate \rangle \\
\langle subject \rangle &\longrightarrow \langle noun\text{-}phrase \rangle \\
\langle noun\text{-}phrase \rangle &\longrightarrow \langle adjective \rangle \langle noun\text{-}phrase \rangle \\
\langle noun\text{-}phrase \rangle &\longrightarrow \langle article \rangle \langle noun\text{-}phrase \rangle \\
\langle noun\text{-}phrase \rangle &\longrightarrow \langle noun \rangle \\
\langle predicate \rangle &\longrightarrow \langle verb \rangle \langle noun\text{-}phrase \rangle \\
\langle predicate \rangle &\longrightarrow \langle verb \rangle \ .
\end{aligned}$$

Now, let us construct or derive the following sentence using the above rules.

a. 'The small person eats the big fish.'

<sentence> ⇒ <subject> <predicate>
⇒ <noun-phrase> <predicate>
⇒ <noun-phrase> <verb> <noun-phrase>
⇒ <article> <noun-phrase> <verb> <noun-phrase>
⇒ <article> <adjective> <noun-phrase> <verb> <noun-phrase>
⇒ <article> <adjective> <noun> <verb> <noun-phrase>
⇒ <article> <adjective> <noun> <verb> <article> <noun-phrase>
⇒ <article> <adjective> <noun> <verb> <article> <adjective> <noun-phrase>
⇒ <article> <adjective> <noun> <verb> <article> <adjective> <noun>
⇒ the <adjective> <noun> <verb> <article> <adjective> <noun>
⇒ the small <noun> <verb> <article> <adjective> <noun>
⇒ the small person <verb> <article> <adjective> <noun>
⇒ the small person eats <article> <adjective> <noun>
⇒ the small person eats the <adjective> <noun>
⇒ the small person eats the big <noun>
⇒ the small person eats the big fish.

b. 'The bird sings.'

<sentence> ⇒ <noun-phrase> <predicate>
⇒ <article> <noun> <predicate>
⇒ <article> <noun> <verb>
⇒ the <noun> <verb>
⇒ the bird <verb>
⇒ the bird sings .

c. 'A dog barks.'

<sentence> ⇒ <noun-phrase> <predicate>
⇒ <noun-phrase> <verb>
⇒ <article> <noun> <verb>

$\Rightarrow$ a <noun> <verb>

$\Rightarrow$ a dog <verb>

$\Rightarrow$ a dog barks.

d. 'Ram reads.'

<sentence> $\Rightarrow$ <noun-phrase> <predicate>

$\Rightarrow$ <noun> <predicate>

$\Rightarrow$ <noun> <verb>

$\Rightarrow$ Ram <verb>

$\Rightarrow$ Ram reads.

Thus, the language of this grammar is

$L = \{$

'a bird sings',

'a dog barks',

'Ram reads',

'small person eats the big fish',

.....

$\}$

9.1.3 *CFG in construction of tokens*

The smallest individual unit of a program is known as a token, such as identifier, constant, string, operator etc. For constructing identifiers in C language, the rules are listed below:

- Identifier (id) is a combination of letters, digits, under score.
- First character in identifier is either a letter or underscore., followed by any number of letters or digits.
- Identifiers are case-sensitive.

The grammar for the above rules is:

<id> $\longrightarrow$ <letter> <rest>

<rest> $\longrightarrow$ <letter> <rest> | <digit> <rest> | $\in$,

<letter> $\longrightarrow -|a|b|\ldots|z$

<digit> $\longrightarrow 0|1|2|\ldots|9.$

Now, let us construct or derive the following identifiers from the above rules.

a. P_1

$$\begin{aligned}
\langle id\rangle &\Rightarrow \langle letter\rangle\ \langle rest\rangle \\
&\Rightarrow \langle letter\rangle\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow P\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow P_1\ \langle rest\rangle \\
&\Rightarrow P_1
\end{aligned}$$

b. emp_12

$$\begin{aligned}
\langle id\rangle &\Rightarrow \langle letter\rangle\ \langle rest\rangle \\
&\Rightarrow \langle letter\rangle\ \langle letter\rangle\ \langle rest\rangle \\
&\Rightarrow \langle letter\rangle\ \langle letter\rangle\ \langle letter\rangle\ \langle rest\rangle \\
&\Rightarrow \langle letter\rangle\ \langle letter\rangle\ \langle letter\rangle\ \langle letter\rangle\ \langle rest\rangle \\
&\Rightarrow \langle letter\rangle\ \langle letter\rangle\ \langle letter\rangle\ \langle letter\rangle\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow \langle letter\rangle\ \langle letter\rangle\ \langle letter\rangle\ \langle letter\rangle\ \langle digit\rangle\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow e\ \langle letter\rangle\ \langle letter\rangle\ \langle letter\rangle\ \langle digit\rangle\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow em\ \langle letter\rangle\ \langle letter\rangle\ \langle digit\rangle\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow emp\ \langle letter\rangle\ \langle digit\rangle\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow emp_\ \langle digit\rangle\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow emp_1\ \langle digit\rangle\ \langle rest\rangle \\
&\Rightarrow emp_12\ \langle rest\rangle \\
&\Rightarrow emp_12.
\end{aligned}$$

The language of the above grammar is $L = \{P_1, emp\_12, \_count, i, \ldots.\}$.

Thus the topic of context-free languages (language defined by context-free grammar) is perhaps the most important aspect of formal language. It is applied in defining programming languages, in formalising the notion of parsing, for simplifying translation of programming languages, in various string-processing applications and in the construction of efficient compilers.

9.2 Ways to Use a Grammar

There are two ways to use a grammar.

- To generate a string of the language. This is easy to do. Start with *Start* symbol and apply derivation steps, until a string (composed entirely of terminals) is obtained.

  EXAMPLE 9.2.1: Constructions of identifiers in Pascal.

- To recognise strings, i.e., to test whether they belong to the language.

  EXAMPLE 9.2.2: An automaton to recognise whether the given string is a palindrome, over $\Sigma = \{a, b\}$. The grammar used is:

$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow \in .$$

Regular Language	Context-free Language
• It is the language that is described by regular expression. • For regular languages, the corresponding acceptor is finite automaton. • Regular languages are closed under union, product, kleene star, intersection and complement. • Regular languages are represented using regular expressions, in FA. • They are used in text editors, sequential circuits etc.	• It is the language that is defined by context-free grammar. • For context-free languages, the corresponding acceptor is push-down automaton. • Context-free languages are closed under union, product and kleene star. • All non-regular languages are represented using context-free grammars. • They are used in programming languages, statements and compilers.

Table 9.1 *Comparison of regular and context-free languages*

9.3 Structure of Grammar

Grammars not only produce natural languages, but also formal ones. If L is a language over an alphabet A, then a grammar for L consists of a set of rules of the form:

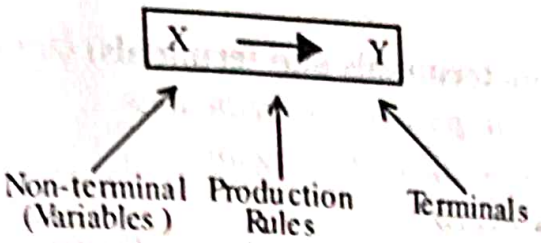


Figure 9.2. *Structure of grammar.*

where, x and y denote strings of symbols taken from A and from a set of grammar symbols, disjoint from A.

9.3.1 Production rule

The grammar rule $x \rightarrow y$ is called a production rule.

EXAMPLE 9.3.1: <*id*>→<*letter*> <*rest*> is a production rule.

9.3.2 Start symbol

Every grammar has a special grammar symbol called the start symbol.

EXAMPLE 9.3.2:

- <*sentence*>→<*noun-phrase*><*predicate*>
- <*id*>→<*letters*> <*rest*>

  Here <*sentence*>, <*id*> are called start symbols.

For a grammar, there must be at least one production with left side, consisting of only the start symbol.

Note–1: If S is the start symbol for a grammar, then there must be at least one production of the form $S \rightarrow Y$.

9.3.3 Non-terminals and Terminals

a. **Non-terminals** – The symbols that may be replaced by other symbols are called non-terminals or variables.

b. **Terminals** – The symbols that cannot be replaced by other symbols are called terminals.

EXAMPLE 9.3.3: (Non–terminals and terminals)

- <*sentence*> → <*noun-phrase*> <*predicate*>
  <*noun-phrase*> → <*article*> <*noun*>
  <*predicate*> → <*verb*> .
  Non-terminals – <*noun-phrase*> and <*predicate*>,
  Terminals – <*verb*>, <*noun*> and <*article*>
- <*id*> ⟶ <*letter*> <*rest*>
  <*rest*> ⟶ <*letter*> <*rest*> | <*digit*> <*rest*>.
  <*letter*> ⟶ $a|b|\ldots.|z.$, <*digit*> ⟶ $0|1|\ldots.|9$
  Non-terminals – <*rest*> <*digit*> <*letter*>
  Terminals – $a|b|\ldots.|z., 0|1|\ldots.|9.$
- If $A = \{a, b, c\}$, then grammar for the language A is

$$S \to \in$$
$$S \to aS$$
$$S \to bS$$

  Non-terminals – S
  Terminals – $a, b.$

c. Grammar rule can be of the form:

One non-terminal ⟶ string of non terminal, or,
One non-terminal ⟶ choice of terminals.

EXAMPLE 9.3.4:

- <*rest*> ⟶ <*letter*> <*rest*> | <*letter*> <*digit*> | ∈
  id ⟶ <*letter*> <*rest*>
- $S \to aS$
  $S \to a|b|c$

d. Convention

- Terminals will typically be smaller case letters.
- Non-terminals will typically be upper case letters.

e. ∈ is neither a non-terminal (since it cannot be replaced with something else) nor a terminal (since it disappears from the string).

9.3.4 *Productions*

Productions refer to the set of rules, used to construct the valid sentences from the given grammar.

EXAMPLE 9.3.3: (Non-terminals and terminals)

- <sentence> → <noun-phrase> <predicate>
  <noun-phrase> → <article> <noun>
  <predicate> → <verb> .
  Non-terminals – <noun-phrase> and <predicate>,
  Terminals – <verb>, <noun> and <article>

- <id> ⟶ <letter> <rest>
  <rest> ⟶ <letter> <rest> | <digit><rest>.
  <letter> ⟶ $a|b|\ldots|z$., <digit> ⟶ $0|1|\ldots|9$
  Non-terminals – <rest> <digit> <letter>
  Terminals – $a|b|\ldots|z, 0|1|\ldots|9$.

- If $A = \{a, b, c\}$, then grammar for the language A is

$$S \to \in$$
$$S \to aS$$
$$S \to bS$$

Non-terminals – S
Terminals – a, b.

c. Grammar rule can be of the form:

One non-terminal ⟶ string of non terminal, or,
One non-terminal ⟶ choice of terminals.

EXAMPLE 9.3.4:

- <rest> ⟶ <letter> <rest> | <letter> <digit> | ∈
  id ⟶ <letter> <rest>
- $S \to aS$
  $S \to a|b|c$

d. Convention

- Terminals will typically be smaller case letters.
- Non-terminals will typically be upper case letters.

e. ∈ is neither a non-terminal (since it cannot be replaced with something else) nor a terminal (since it disappears from the string).

9.3.4 *Productions*

Productions refer to the set of rules, used to construct the valid sentences from the given grammar.

EXAMPLE 9.3.5: (Productions)

- $id \longrightarrow$ <letter> <rest>
- <sentence> $\longrightarrow$ <noun-phrase> <predicate>
- $S \rightarrow aSa$
- $S \rightarrow bSb$
- $S \rightarrow a$.

9.3.5 Forms of production

Production can be

a. **Unit-production** – Any production of the form $A \rightarrow B$ is a unit production.

EXAMPLE 9.3.6:

(i) S → AalB
(ii) B → Albb

Here, $S \rightarrow B$ and $B \rightarrow A$ are unit production. $S \rightarrow Aa$, $B \rightarrow bb$, are non-unit productions.

b. **∈-production** – Any production of the form $A \rightarrow \in$ is called ∈-production.

EXAMPLE 9.3.7:

(i) <rest> $\longrightarrow$ <letter> <rest> | <letter> <digit> | ∈
(ii) $A \rightarrow aa| \in$

c. **Recursive production** – A production is called recursive if its left side occurs on its right side, or if non-terminals are present on both sides of '→'.

EXAMPLE 9.3.8:

(i) $S \rightarrow aS$
(ii) <rest> $\longrightarrow$ <letter> <rest>

d. **Indirectly recursive production** – A production, which is not directly recursive, is an indirectly recursive production.

EXAMPLE 9.3.9: Consider the production rules of the form

$$S \rightarrow b|aA$$

$$A \rightarrow c|bS.$$

EXAMPLE 9.3.5: (Productions)

- *id* ⟶ <*letter*> <*rest*>
- <*sentence*> ⟶ <*noun-phrase*> <*predicate*>
- $S \rightarrow aSa$
  $S \rightarrow bSb$
  $S \rightarrow a.$

9.3.5 *Forms of production*

Production can be

a. **Unit-production** – Any production of the form $A \rightarrow B$ is a unit production.

EXAMPLE 9.3.6:

(i) S → AaIB
(ii) B → AIbb

Here, $S \rightarrow B$ and $B \rightarrow A$ are unit production. $S \rightarrow Aa$, $B \rightarrow bb$, are non-unit productions.

b. **∈-production** – Any production of the form $A \rightarrow \in$ is called ∈-production.

EXAMPLE 9.3.7:

(i) <*rest*> ⟶ <*letter*> <*rest*> | <*letter*> <*digit*> | ∈
(ii) $A \rightarrow aa| \in$

c. **Recursive production** – A production is called recursive if its left side occurs on its right side, or if non-terminals are present on both sides of '→'.

EXAMPLE 9.3.8:

(i) $S \rightarrow aS$
(ii) <*rest*> ⟶ <*letter*> <*rest*>

d. **Indirectly recursive production** – A production, which is not directly recursive, is an indirectly recursive production.

EXAMPLE 9.3.9: Consider the production rules of the form

$$S \rightarrow b|aA$$

$$A \rightarrow c|bS.$$

Here,

- $S \Rightarrow aA$
  $\Rightarrow abS$
- $A \Rightarrow bS$
  $\Rightarrow baA.$

Hence, $S \rightarrow aA$ and $A \rightarrow bs$ are indirectly recursive.

9.3.6 *Backus Normal Form (BNF)*

The compact notation, used to represent the production rule, is called BNF.

EXAMPLE 9.3.10: (BNF)

(i) <rest> ⟶ <letter> <rest>
<rest> ⟶ <digits> <rest>
<rest> ⟶ ∈

These can be written using BNF as

<rest> ⟶ <letter> <rest> | <digits> <rest> | ∈

(ii) $S \rightarrow SS$
$S \rightarrow a$
$S \rightarrow \in.$

BNF: $S \rightarrow SS|a| \in.$

We discuss the extended BNF in the next chapter, in section 10.9.2.

9.3.7 *Derivation*

The sequence of substitutions, used to obtain a string, is called a derivation. Derivation refers to replacing an instance of a given string's non-terminal, by the right-hand side of the production rule, whose left-hand side contains the non-terminal to be replaced.

If x and y are sentential forms (see section 9.9.4) and $\alpha \rightarrow \beta$ is a production, then the replacement of α by β in $x\alpha y$ is called a derivation and denoted by:

$$S \rightarrow x\alpha y \quad \text{(Production rule).}$$
$$\alpha \rightarrow \beta$$
$$x\alpha y \Rightarrow x\beta y. \quad \text{(Derivation)}$$

Derivation produces a new string from a given string. Therefore derivation can be used repeatedly to obtain a new string from a given string. If the string obtained, as a result of the derivation, contains only terminal symbols, then no further derivations are possible.

Note–2: Two types of arrows: $\rightarrow$ Used in the Statement of Productions.
$\Rightarrow$ Used in the derivation of word.

EXAMPLE 9.3.11: (Derivation)

- Consider the production rule:
  $S \rightarrow aSb$
  $S \rightarrow \in$
  Derivation for ab:

$$S \Rightarrow aSb$$
$$S \Rightarrow a \in b$$
$$S \Rightarrow ab$$

- Consider the productions:
  $S \rightarrow Ab$
  $A \rightarrow aAb$
  $A \rightarrow \in$
  Derivation for $aabbb$:

$$S \Rightarrow Ab$$
$$\Rightarrow aAbb$$
$$\Rightarrow aaAbbb$$
$$\Rightarrow aabbb$$

- $id \longrightarrow <letter>\ <rest>$
  $<rest> \longrightarrow <letter>\ <rest> \mid <digit>\ <rest> \mid \in$
  $<letter> \longrightarrow a|b| \ldots .|z$
  $<digit> \longrightarrow 0| \ldots |9$
  Derivation for $a0$:

$$id \Rightarrow <letter><rest>$$
$$\Rightarrow <letter><digit><rest>$$
$$\Rightarrow a0 \in$$
$$\Rightarrow a0$$

9.3.8 *Forms of derivation*

Derivation can be *Leftmost* or *Rightmost* derivation.

- Leftmost derivation – A derivation is said to be leftmost if in each step, the leftmost variable in the sentential form is replaced.
- Rightmost derivation – A derivation is said to be rightmost, if in each step, the rightmost variable in the sentential form is replaced.

EXAMPLE 9.3.12: (Leftmost and rightmost derivation)

(i) Consider the production rules:

$$S \to AB$$
$$A \to aaA$$
$$A \to \in$$
$$B \to Bb$$
$$B \to \in$$

Leftmost derivation for *aab*

$$S \Rightarrow AB$$
$$\Rightarrow aaAB$$
$$\Rightarrow aaB$$
$$\Rightarrow aaBb$$
$$\Rightarrow aab$$

Rightmost derivation for *aab*

$$S \Rightarrow AB$$
$$\Rightarrow ABb$$
$$\Rightarrow Ab$$
$$\Rightarrow aaAb$$
$$\Rightarrow aab$$

(ii) Consider the production rules:

$$S \to aAB$$
$$A \to bBb$$
$$B \to A \mid \in$$

9.3.8 *Forms of derivation*

Derivation can be *Leftmost* or *Rightmost* derivation.

- Leftmost derivation – A derivation is said to be leftmost if in each step, the leftmost variable in the sentential form is replaced.
- Rightmost derivation – A derivation is said to be rightmost, if in each step, the rightmost variable in the sentential form is replaced.

EXAMPLE 9.3.12: (Leftmost and rightmost derivation)

(i) Consider the production rules:

$$S \to AB$$
$$A \to aaA$$
$$A \to \in$$
$$B \to Bb$$
$$B \to \in$$

Leftmost derivation for *aab*

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow aaAB \\ &\Rightarrow aaB \\ &\Rightarrow aaBb \\ &\Rightarrow aab \end{aligned}$$

Rightmost derivation for *aab*

$$\begin{aligned} S &\Rightarrow AB \\ &\Rightarrow ABb \\ &\Rightarrow Ab \\ &\Rightarrow aaAb \\ &\Rightarrow aab \end{aligned}$$

(ii) Consider the production rules:

$$S \to aAB$$
$$A \to bBb$$
$$B \to A|\in$$

Leftmost derivation for *abbbb*

$$\begin{aligned} S &\Rightarrow aAB \\ &\Rightarrow abBbB \\ &\Rightarrow abAbB \\ &\Rightarrow abbBbbB \\ &\Rightarrow abbbbB \\ &\Rightarrow abbbb \end{aligned}$$

Rightmost derivation for *abbbb*

$$\begin{aligned} S &\Rightarrow aAB \\ &\Rightarrow aA \\ &\Rightarrow abBb \\ &\Rightarrow abAb \\ &\Rightarrow abbBbb \\ &\Rightarrow abbbb \end{aligned}$$

9.3.9 *The $\overset{*}{\Longrightarrow}$ notion in derivation*

Any derivation involves the application of production rules. If the production rule is applied once, then we write:

$$w_1 \Rightarrow w_2.$$

If the production rule is applied more than once, then we write:

$$w_1 \overset{*}{\Longrightarrow} w_n,$$

which means $w_1 \Rightarrow w_2 \Rightarrow w_3 \Rightarrow \ldots \Rightarrow w_n$.

EXAMPLE 9.3.13: ($\overset{*}{\Longrightarrow}$ notion in derivation)

(i) Consider the productions:

$$S \rightarrow aSb$$
$$S \rightarrow \in$$

Derivation for *aaabbb*:

We have

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

Instead, we write this as $S \overset{*}{\Longrightarrow} aaabbb$.

(ii) For the production rules:

$S \rightarrow aSb$

$S \rightarrow \in$

We say

$$S \overset{*}{\Longrightarrow} \in \quad (S \Rightarrow \in)$$

$$S \overset{*}{\Longrightarrow} ab \quad (S \Rightarrow aSb \Rightarrow a \in b \Rightarrow ab)$$

$$S \overset{*}{\Longrightarrow} aabb \quad (S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb)$$

$$S \overset{*}{\Longrightarrow} aaSbb \quad (S \Rightarrow aSb \Rightarrow aaSbb).$$

(iii) Consider

$$S \rightarrow Ab$$

$$A \rightarrow aAb$$

$$A \rightarrow \in$$

Derivation for *aaaaabbbbbb*:

$$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb$$

$$\Rightarrow aaaAbbbb$$

$$\Rightarrow aaaaAbbbbb$$

$$\Rightarrow aaaaaAbbbbbb$$

$$\Rightarrow aaaaabbbbbb.$$

Instead, we write this as,

$$S \overset{*}{\Longrightarrow} aaaaabbbbbb$$

or $S \overset{*}{\Longrightarrow} a^n b^n b.$

We further discuss derivation tree in detail in section 9.9.

We have

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb$$

Instead, we write this as $S \overset{*}{\Longrightarrow} aaabbb$.

(ii) For the production rules:

$S \rightarrow aSb$

$S \rightarrow \in$

We say

$$S \overset{*}{\Longrightarrow} \in \quad (S \Rightarrow \in)$$

$$S \overset{*}{\Longrightarrow} ab \quad (S \Rightarrow aSb \Rightarrow a \in b \Rightarrow ab)$$

$$S \overset{*}{\Longrightarrow} aabb \quad (S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aabb)$$

$$S \overset{*}{\Longrightarrow} aaSbb \quad (S \Rightarrow aSb \Rightarrow aaSbb).$$

(iii) Consider

$$S \rightarrow Ab$$

$$A \rightarrow aAb$$

$$A \rightarrow \in$$

Derivation for $aaaaabbbbbb$:

$$S \Rightarrow Ab \Rightarrow aAbb \Rightarrow aaAbbb$$

$$\Rightarrow aaaAbbbb$$

$$\Rightarrow aaaaAbbbbb$$

$$\Rightarrow aaaaaAbbbbbb$$

$$\Rightarrow aaaaabbbbbb.$$

Instead, we write this as,

$$S \overset{*}{\Longrightarrow} aaaaabbbbbb$$

or $S \overset{*}{\Longrightarrow} a^n b^n b.$

We further discuss derivation tree in detail in section 9.9.

9.4 Formal Definition of Context-Free Grammars

A context free grammar is a method for (recursively) describing the grammar of a given language. A CFG is a set of variables, each of which represents a language. The language, represented by the variables, is described in terms of primitive symbols called *terminals*. The rules relating to the variables are called *productions*.

A CFG, G, is formally defined as a 4-tuple

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

where
V—A finite set of Variables or Non-terminals.
T—A finite set of terminals.
P—A finite set of production rules.
S—Start symbol, $S \in V$.

Each production is of the form $A \rightarrow \alpha$, where A is a variable, α may be either terminal or non-terminal, i.e., $A \in V$ and $\alpha \in (V \cup T)^*$.

EXAMPLE 9.4.1: (Context-free grammar)

a. $G = (\{s\}, \{a, b\}, P, S)$, where P contains

$$P = \{\ S \rightarrow aSa,$$
$$S \rightarrow bsb,$$
$$S \rightarrow \in$$
$$\}.$$

b. $G = (\{id, Letter, rest, digit\}, \{a..z, 0..9\}, P, id), P$ is

$$P = \{<id> \rightarrow <letter>\ <rest>,$$
$$<rest> \rightarrow <letter>\ <rest> \mid <digit>\ <rest> \mid \in,$$
$$<letter> \rightarrow a|b|\ldots|z,$$
$$<digit> \rightarrow 0|1|\ldots|9$$
$$\}.$$

c. $G = ((S, A), \{a, b\}, P, S)$ where P is

$$S \rightarrow aAS|a$$
$$A \rightarrow SbA|SS|ba.$$

9.5 Types of Grammar

In this section, we discuss some of the commonly used grammars, and give the complete classification of grammar in chapter 15.

9.5.1 *Linear and Nonlinear Grammars*

A grammar, with at most one variable (non-terminal) at the right side of a production, is a linear grammar, otherwise it is nonlinear.

EXAMPLE 9.5.1: (Linear grammar)

(i) $S \rightarrow aSb$
$S \rightarrow \in$

(ii) $S \rightarrow Ab$
$A \rightarrow aAb$
$A \rightarrow \in$

(iii) $'S \rightarrow A$
$A \rightarrow aB| \in$
$B \rightarrow Ab.$

EXAMPLE 9.5.2: (Nonlinear grammar)

(i) $S \rightarrow SS$
$S \rightarrow \in$
$S \rightarrow aSb$
$S \rightarrow bSa.$

(ii) $S \rightarrow aA$
$A \rightarrow \in$
$S \rightarrow aSAS.$

9.5.2 *Right and Left-Linear Grammars*

Linear grammars are further classified into:

a. Right-linear grammar

A grammar $G = \{V, T, S, P\}$ is right-linear, if all productions are of the form

$$A \to xB$$

or, $A \to x$

where $A, B \in V$ and $x \in T^*$.

EXAMPLE 9.5.3: (Right-linear grammar)

(i) $S \to abS$

$S \to a.$

(ii) $S \to A$

$A \to aA$

$A \to \in.$

b. Left-linear grammar

A grammar $G = \{V, T, S, P\}$ is left-linear, if all productions are of form

$$A \to Bx$$

or, $A \to x$

where $A, B, \in V$ and $x \in T^*$.

EXAMPLE 9.5.4: (Left-linear grammar)

(i) $S \to Aab$

$A \to Aab|B$

$B \to a$

(ii) $S \to Ab$

$S \to Sb$

$A \to \in.$

9.5.3 *Regular Grammar*

A grammar $G = \{V, T, S, P\}$ is said to be regular, if it is either right-linear or left-linear.

EXAMPLE 9.5.5: (Regular grammar)

(i) $S \to abS$

$S \to a$

(ii) $S \to Aab$

$A \to Aab|B$

$B \to a$

9.5.4 *Non-regular Grammar*

A grammar $G = (V, T, S, P)$ is non-regular, if it is neither right-Linear nor left-Linear.

EXAMPLE 9.5.6: (Non-regular grammar)

(i) $S \to aSb$
$S \to \in$

(ii) $S \to Ab$
$A \to aAb$
$A \to \in$

(iii) $S \to \in$
$S \to aSb$
$S \to bsa$

9.5.5 *Simple or S-grammar*

A grammar $G = (V, T, S, P)$ is said to be a simple grammar or S-grammar, if all productions are of the form

$$A \to ax,$$

where $A \in V$, $a \in T$ and $x \in V^*$ and any pair (A, a) occurs at most once in P.

EXAMPLE 9.5.7: S-grammar

(i) $S \to aS|bSS|c$ is a simple grammar, since pairs (S, a), (S, b) occur only once in production.

(ii) $S \to aAS|a$
$A \to SbA$.
Pair (S, a) occurs only once in production and is a S-grammar.

(iii) $S \to aS|bSS|aSS|c$ is not a simple grammar, since pair (a, S) occurs twice in production as aS and aSS.

9.5.6 *Recursive grammar*

A grammar $G = (V, T, S, P)$ is recursive, if it contains either a recursive production or an indirectly recursive production.

EXAMPLE 9.5.8: Recursive grammar

(i) $S \rightarrow aS$

(ii) $S \rightarrow SS$

$S \rightarrow \in$

$S \rightarrow aSb$

(iii) $S \rightarrow b|aA$

$A \rightarrow c|bS.$

9.6 Language of a Context-Free Grammar

Definition: The language generated (defined, derived, produced) by a CFG, is the set of all strings of terminals, that can be produced from the start symbol S using the productions as substitutions. A language generated by a CFG, G is called a *context-free language* (CFL) and is denoted by L(G)..

Thus, if G is a CFG, with S as a start symbol and set of terminals T, then the language of G is the set defined as

$$L(G) = \{w | w \in T^* \text{ and } S \overset{*}{\Longrightarrow} w\}.$$

We discuss the context free language and its properties in detail in chapter 15.

EXAMPLE 9.6.1: For the given string sets,

a. what is the language accepted by CFG
b. what grammar is required to derive these strings
c. what is the regular expression?

(i) $\{\in, a, aa, \ldots., a^n, \ldots.\}$.

- $L(CFG) = \{a^n | n \geq 0\}$.
- The CFG to derive these strings is

$$P = \{$$
$$S \rightarrow \in$$
$$S \rightarrow aS.$$
$$\}$$

where S is the start symbol, $V = \{S\}$ and T = {a}.

- $RE = a^*$
  Derivation for the string aaa :

$$S \Rightarrow aS \Rightarrow aaS \Rightarrow aaaS \Rightarrow aaa.$$

$$\text{i.e., } S \overset{*}{\Longrightarrow} aaa.$$

(ii) $\{\in, ab, aabb, \ldots, a^n b^n, \ldots\}$

- $L(CFG) = \{a^n b^n | n \geq 0\}$.
- CFG to derive the above strings:
  Here, any string in this language is either $\in$ or of the form axb for some string x in the language. The following grammar will derive any of the strings.

$$P = \{$$
$$S \rightarrow \in$$
$$S \rightarrow aSb.$$
$$\}$$

where $V = \{S\}$, $T = \{a, b\}$ and $S = \{S\}$.

- $RE = (ab)^*$
  Derivation for the string $aaabbb$:

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow aaabbb.$$

$$\text{i.e. } S \overset{*}{\Longrightarrow} aaabbb.$$

(iii) $\{\in, ab, abab, \ldots.(ab)^n, \ldots.\}$

- $L(CFG) = \{(ab)^n | n \geq 0\}$.
- Only string in this language is either $\in$ or of the form abx, for some string x in the language. The grammar is:

$$P = \{$$
$$S \rightarrow \in$$
$$S \rightarrow abS.$$
$$\}$$

where $V = \{S\}$, $T = \{a, b\}$ and $S = \{S\}$.

- $RE = (ab + ab)^*$
  Derivation for the string $ababab$:

$$S \Rightarrow abS \Rightarrow ababS \Rightarrow abababS \Rightarrow abababab.$$

(iv) $\{a, aab, aabab, \ldots.\}$

- $L(CFG) = \{(ab)^n \cdot a | n \geq 0\}$.
- Grammar:

$$P = \{$$
$$S \rightarrow abS$$
$$S \rightarrow a$$
$$\}$$

where $V = \{S\}$, $T = \{a, b\}$ and $S = \{S\}$.

- $RE = (ab)^* \cdot a$

(v) $\{\in, aa, bb, abba, \ldots\ldots\}$.

- $L(CFG) = \{(ab)^n | n \geq 0$ such that $w \notin aba\}$.
- Grammar:

$$P = \{$$
$$S \rightarrow aSa,$$
$$S \rightarrow bSb,$$
$$S \rightarrow \in$$
$$\}$$

where, $V = \{S\}$, $T = \{a, b\}$ and $S = \{S\}$.

- $RE = (a + b)^* \cdot (a + b)^*$

Derivation for the string *abba*:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba.$$
$$\text{i.e. } S \overset{*}{\Rightarrow} abba.$$

(vi) $\{a, b, aa, bb, \ldots\}$

- $L(CFG) = \{(ab)^n | n \geq |\}$.
- Grammar:

$$P = \{S \rightarrow aS|bS|a|b\} \text{ where } V = \{S\}, T = \{a, b\} \text{and} S = \{S\}.$$

- $RE = (a + b)^+$

Derivation for the string *abbab*:

$$S \Rightarrow aS \Rightarrow abS \Rightarrow abbS \Rightarrow abbaS \Rightarrow abbab.$$

(iv) $\{a, aab, aabab, \ldots\}$

- $L(CFG) = \{(ab)^n \cdot a | n \geq 0\}$,
- Grammar:

$$P = \{$$
$$S \rightarrow abS$$
$$S \rightarrow a$$
$$\}$$

where $V = \{S\}$, $T = \{a, b\}$ and $S = \{S\}$,

- $RE = (ab)^* \cdot a$

(v) $\{\epsilon, aa, bb, abba, \ldots\ldots\}$,

- $L(CFG) = \{(ab)^n | n \geq 0$ such that $w \neq aba\}$,
- Grammar:

$$P = \{$$
$$S \rightarrow aSa,$$
$$S \rightarrow bSb,$$
$$S \rightarrow \epsilon$$
$$\}$$

where, $V = \{S\}$, $T = \{a, b\}$ and $S = \{S\}$.

- $RE = (a + b)^* \cdot (a + b)^*$

Derivation for the string *abba*:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abba.$$
$$\text{i.e. } S \overset{*}{\Rightarrow} abba.$$

(vi) $\{a, b, aa, bb, \ldots\}$

- $L(CFG) = \{(ab)^n | n \geq 1\}$.
- Grammar:

$$P = \{S \rightarrow aS|bS|a|b\} \text{ where } V = \{S\}, T = \{a, b\} \text{and} S = \{S\}.$$

- $RE = (a + b)^+$

Derivation for the string *abbab*:

$$S \Rightarrow aS \Rightarrow abS \Rightarrow abbS \Rightarrow abbaS \Rightarrow abbab.$$

(vii) $L = \{\in, aaa, aaaaaa, \ldots\}$.

- $L(CFG) = \{w : |w| \mod 3 = 0\}$
- Grammar:

$$P = \{$$
$$S \to \in$$
$$S \to aaaS$$
$$\}$$

where $V = \{S\}$, $T = \{a\}$and $S = \{S\}$.

- $RE = \big((a+b)(a+b)(a+b)\big)^*$

9.7 Operations on Production Rules

Suppose M and N are languages, whose grammars have disjoint sets of non-terminals. Also, assume that start symbols, for the grammars of M and N, are A and B respectively. Then, following are the rules to find new grammars generated from M and N:

- Union Rule: The language $M \cup N$ starts with two productions,

$$S \to A|B.$$

- Product Rule: The language MN starts with the productions

$$S \to AB.$$

- Closure Rule: The language M^* starts with the productions

$$S \to AS|\in .$$

EXAMPLE 9.7.1: (Operations on production rules)

a. **Union Rule**

If $L=\{\in, a, b, aa, bb, \ldots, a^n, b^n, \ldots\}$ with
$RE = (a+b)^*$,
L can be written as the union of M and N

$$\text{i.e} \quad L = M \cup N$$

where, $M = \{a^n | n \geq 0\}$ and $N = \{b^n | n \geq 0\}$.

The grammar for L is.

$$S \to A|B \quad \text{(union rule)}.$$
$$A \to \in |aA \quad \text{(grammar for } M\text{)},$$
$$B \to \in |bB \quad \text{(grammar for } N\text{)}.$$

b. **Product Rule**

If $L = \{\in, ab, aabb, aaaabbbb, aabbb \ldots\}$ with $RE = (ab)^*$ i.e. $L = \{a^m b^n | m, n \geq 0\}$, L can be written as a product

$$L = MN.$$

where $M = \{a^m | m \geq 0\}$ and $N = \{b^n | n \geq 0\}$.
Thus grammar for L is:

$$S \to AB \quad \text{(product rule)}.$$
$$A \to \in |aA \quad \text{(grammar for } M\text{)}.$$
$$B \to \in |bB \quad \text{(grammar for } N\text{)}.$$

c. **Closure Rule**

If $L = \{\in, aa, bb, aabb, aaaabb, bbbb, aaaabbbb, \ldots\}$., or, $L = \{aa, bb\}^*$ with $RE = (aa)^*.(bb)^*$ and if some $M = \{aa, bb\}$, then $L = M^*$.
The grammar for L is:

$$S \to AS| \in \quad \text{(closure rule)}.$$
$$A \to aa|bb \quad \text{(grammar for } M\text{)}.$$

This grammar can also be written as (by substituting A).

$$S \to aaS|bbS| \in .$$

9.8 Design of a CFG

The basic design strategy for CFG is as follows:

a. Understand the language specification by listing the examples of the strings in the language.
b. Determine the 'base case' productions of CFG by listing the shortest strings in the language.

The grammar for L is.

$$S \rightarrow A|B \quad \text{(union rule)}.$$
$$A \rightarrow \in |aA \quad \text{(grammar for } M\text{)}.$$
$$B \rightarrow \in |bB \quad \text{(grammar for } N\text{)}.$$

b. **Product Rule**

If $L = \{\in, ab, aabb, aaaabbbb, aabbb \ldots\}$ with $RE = (ab)^*$ i.e. $L = \{a^m b^n | m, n \geq 0\}$.

L can be written as a product

$$L = MN.$$

where $M = \{a^m | m \geq 0\}$ and $N = \{b^n | n \geq 0\}$.

Thus grammar for L is:

$$S \rightarrow AB \quad \text{(product rule)}.$$
$$A \rightarrow \in |aA \quad \text{(grammar for } M\text{)}.$$
$$B \rightarrow \in |bB \quad \text{(grammar for } N\text{)}.$$

c. **Closure Rule**

If $L = \{\in, aa, bb, aabb, aaaabb, bbbb, aaaabbbb, \ldots\}$., or, $L = \{aa, bb\}^*$ with $RE = (aa)^*.(bb)^*$ and if some $M = \{aa, bb\}$, then $L = M^*$.

The grammar for L is:

$$S \rightarrow AS| \in \quad \text{(closure rule)}.$$
$$A \rightarrow aa|bb \quad \text{(grammar for } M\text{)}.$$

This grammar can also be written as (by substituting A).

$$S \rightarrow aaS|bbS| \in .$$

9.8 Design of a CFG

The basic design strategy for CFG is as follows:

a. Understand the language specification by listing the examples of the strings in the language.
b. Determine the 'base case' productions of CFG by listing the shortest strings in the language.

c. Identify the rules for combining smaller sentences into larger ones.
d. Test the CFG obtained on a number of carefully chosen examples. All of the base cases should be tested, along with all of the alternative productions. Also, whether the grammar is consistent with the strings listed in step a or not is checked.

EXAMPLE 9.8.1: Obtain a CFG for the language of palindrome over the alphabet $\Sigma = \{a, b, c\}$.

Solution: The recursive definition of a palindrome is,

a. $\in$ is a palindrome.
b. a, b and c are palindromes.
c. If w is a palindrome then the strings awa, bwb and cwc are palindromes.

Let the CFG, $G = (V, T, P, S)$, where:

$$V = \{S\}$$

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

$$P = \{$$

$$S \to aSa$$
$$S \to bSb$$
$$S \to cSC$$
$$S \to a|b|c| \in \qquad \text{[By definition 1 and 2]}$$

$$\}$$

S is the start symbol.

- Derivation for the string $abcba$, which is a palindrome, is shown below:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow abcba.$$

- Derivation for the string $cbaabc$, which is a palindrome, is shown below:

$$S \Rightarrow cSc \Rightarrow cbSbc \Rightarrow cbaSabc \Rightarrow cbaabc.$$

The language of CFG is $L(G) = \{w^R = w | w \in L\}$.

EXAMPLE 9.8.2: Obtain a CFG for the language of even palindrome, over the alphabet $\Sigma = \{a, b\}$.

Solution: Let the CFG, $G = (V, T, P, S)$, where:

$$V = \{S\}$$
$$T = \{a, b\}$$
$$P = \{$$
$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow \in$$
$$\}$$

S is the start symbol.

- Derivation for the string *abba*, which is an even palindrome, is shown below:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow ab \in ba \Rightarrow abba.$$

EXAMPLE 9.8.3: Obtain a CFG for the language of odd palindrome over the alphabet $\Sigma = \{a, b\}$.

Solution: Let the CFG, $G = (V, T, P, S)$, where:

$$V = \{S\}$$
$$T = \{a, b\}$$
$$P = \{$$
$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow a|b$$
$$\}$$

S is the start symbol.

Derivation for the string *aaa* which is an odd palindrome is

$$S \Rightarrow aSa \Rightarrow aaa.$$

EXAMPLE 9.8.4: Construct a CFG to generate the set of all balanced parenthesis over the alphabet $\Sigma = \{(,)\}$.

Solution: Set of all balanced parenthesis over $\{(,)\}$ is recursively defined as

a. $\in$ and ( ) are balanced.
b. If w is balanced, so is (w).
c. If w and x are balanced, so is wx.
d. Nothing else is balanced.

$$V = \{S\}$$
$$T = \{a, b\}$$
$$P = \{$$
$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow \in$$
$$\}$$

S is the start symbol.

- Derivation for the string *abba*, which is an even palindrome, is shown below:

$$S \Rightarrow aSa \Rightarrow abSba \Rightarrow ab \in ba \Rightarrow abba.$$

EXAMPLE 9.8.3: Obtain a CFG for the language of odd palindrome over the alphabet $\Sigma = \{a, b\}$.

Solution: Let the CFG, $G = (V, T, P, S)$, where:

$$V = \{S\}$$
$$T = \{a, b\}$$
$$P = \{$$
$$S \rightarrow aSa$$
$$S \rightarrow bSb$$
$$S \rightarrow a|b$$
$$\}$$

S is the start symbol.

Derivation for the string *aaa* which is an odd palindrome is

$$S \Rightarrow aSa \Rightarrow aaa.$$

EXAMPLE 9.8.4: Construct a CFG to generate the set of all balanced parenthesis over the alphabet $\Sigma = \{(,)\}$.

Solution: Set of all balanced parenthesis over $\{(,)\}$ is recursively defined as

a. $\in$ and ( ) are balanced.
b. If w is balanced, so is (w).
c. If w and x are balanced, so is wx.
d. Nothing else is balanced.

Let the CFG, $G = (V, T, P, S)$, where:

$$V = \{S\}$$

$$T = \{(,)\}$$

$$P = \{$$

$$S \rightarrow \in \mid () \qquad \text{[Definition-a]}$$
$$S \rightarrow (S) \qquad \text{[Definition-b]}$$
$$S \rightarrow SS \qquad \text{[Definition-c]}$$

$$\}$$

S is the start symbol.

Derivation for ((( ))), which is a balanced parenthesis is shown below:

$$S \Rightarrow (S) \Rightarrow ((S)) \Rightarrow (((S))) \Rightarrow ((( ))).$$

EXAMPLE 9.8.5: Obtain a CFG to generate a language of all non-palindrome over the alphabet $\Sigma = \{a, b\}$.

Solution: The following cases are considered:

a. Generate palindromes on both left and right side.
b. Generate a non-palindrome.
c. Make one non-terminal to generate any number of a's and b's.
d. Generate any combination of a's and b's.

Let the CFG, $G = (V, T, P, S)$, where

$$V = \{S, A, B\}$$

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

$$P = \{$$

$$S \rightarrow aSa \mid bSb \qquad \text{[By a]}$$
$$S \rightarrow A \qquad \text{[By b]}$$
$$A \rightarrow aBb \mid bBa \qquad \text{[By c]}$$
$$B \rightarrow aB \mid bB \mid \in \qquad \text{[By d]}$$

$$\}$$

S is the start symbol.

Derivation for the string *abbaab*, which is not a palindrome, is shown below:

$$S \Rightarrow A \Rightarrow aBb \Rightarrow abBb \Rightarrow abbBb \Rightarrow abbaBb \Rightarrow abbaaBb \Rightarrow abbaab.$$

EXAMPLE 9.8.6: Construct a CFG to generate a restricted class of arithmetic expressions on integers.

Solution: An arithmetic expression AE can be recursively defined as follows:

a. An expression AE can be an identifier.
b. If AE is any arithmetic expression, then
   - $AE + AE$
   - $AE - AE$
   - $AE * AE$
   - $AE|AE$
   - $AE \wedge AE$
   - (AE)

   are all arithmetic expressions.

Consider a set of operators $\{+, -, *, /, \wedge\}$ and an identifier (I), which can start with any of the letters from $\{a, b, c\}$. Then

$$I \rightarrow Ia|Ib|Ic|a|b|c.$$

Let the CFG, $G = (V, T, P, S)$,

where $V = \{AE, I\}$

$$T = \{+, -, *, /, \wedge, a, b, c\}$$

$$P = \{$$

$$AE \rightarrow I$$
$$AE \rightarrow AE + AE$$
$$AE \rightarrow AE - AE$$
$$AE \rightarrow AE * AE$$
$$AE \rightarrow AE|AE$$
$$AE \rightarrow AE \wedge AE$$
$$AE \rightarrow (AE)$$
$$I \rightarrow Ia|Ib|Ic|a|b|c.$$

$$\}$$

AE is the start symbol.

Derivation to generate the arithmetic expression $a + (b - c)|a$:

$$\begin{aligned} AE &\Rightarrow (AE) \\ &\Rightarrow (AE + AE) \\ &\Rightarrow (AE + AE|AE) \\ &\Rightarrow (AE + (AE)|AE) \\ &\Rightarrow (AE + (AE - AE)|AE) \\ &\Rightarrow (I + (I - I)|I) \\ &\Rightarrow (a + (b - c)|a) \end{aligned}$$

EXAMPLE 9.8.7: Find the grammar for the language of decimal numerals by observing that a decimal numeral is either a digit or a digit followed by decimal numeral.

Solution: A number N can be recursively defined as follows:

a. A number is a digit (Digit).
b. A number followed by a digit and vice-versa is also a number.

$$N \rightarrow Digit$$
$$N \rightarrow N\,Digit|Digit\,N.$$

Let $S = \{+, -, \in\}$ denote the sign of a number, Digit $= \{0, 1, \ldots 9\}$ denote digits forming the number and an integer I, which can be a number(N) or the sign of a number(S) followed by the number and so on, i.e.,

$$I \rightarrow N|SN.$$

Let the CFG, $G = (V, T, P, S)$,

where $V = \{\text{Digit}, S, N, I\}$

$T = \{+, -, 0, 1, \ldots 9\}$

$P = \{$

$I \rightarrow N|SN$
$N \rightarrow Digit|NDigit|DigitN$
$S \rightarrow +| - | \in$
$Digit \rightarrow 0|1|\ldots.|9.$

$\}$

I is the start symbol.

Derivation to generate the number 9540 is shown below.

$$
\begin{aligned}
I &\Rightarrow N \\
&\Rightarrow NDigit \\
&\Rightarrow N0 \\
&\Rightarrow NDigit0 \\
&\Rightarrow N40 \\
&\Rightarrow NDigit40 \\
&\Rightarrow N540 \\
&\Rightarrow Digit540 \\
&\Rightarrow 9540
\end{aligned}
$$

EXAMPLE 9.8.8: Obtain the CFG to generate the regular expression $(011 + 1)^*(01)^*$.

Solution: The given RE $(011 + 1)^*(01)^*$ is of the form A^*B^* where A is 011 or 1 and B is 01.

Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A, B\}$

$T = \{0, 1\}$

$P = \{$

$S \rightarrow AB$

$A \rightarrow 011A|1A| \in$

$B \rightarrow 01B| \in .$

$\}$

S is the start symbol.

EXAMPLE 9.8.9: Obtain the CFG for the regular expression $(a + b)^*aa(a + b)^*$.

Solution: The given $RE(a + b)^*aa(a + b)^*$ is of the form X^*aaX^*, where X is $(a + b)$.

Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, X\}$

$T = \{a, b\}$

$P = \{$

$$S \rightarrow XaaX$$
$$X \rightarrow aX|bX|\in$$

}

S is the start symbol.

Derivation for the string *abbaaba* is shown below:

$$S \Rightarrow XaaX \Rightarrow aXaaX \Rightarrow abXaaX \Rightarrow abbaaX \Rightarrow abbaabX \Rightarrow abbaabaX \Rightarrow abbaaba.$$

EXAMPLE 9.8.10: Find the CFG for the regular expression $(a+b)^*aa(a+b)^*$ given X, Y and S are non-terminals. S is the start symbol with:

a. X productions, producing words ending with a.

b. Y productions, producing words starting with a.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, X, Y\}$

$T = \{a, b\}$

$P = \{$

$S \rightarrow XY$
$X \rightarrow aX|bX|a$
$Y \rightarrow Ya|Yb|a$

$\}$

S is the start symbol.

EXAMPLE 9.8.11: Obtain the CFG to generate the language $L = \{w|n_a(w) = n_b(w)\}$.

Solution: The grammar to be generated should have number of a's in the string w equal to the number of b's in the string w. The following cases are considered:

a. $\in$ (denotes) zero a s and zero b s

b. 'a' followed by the symbol 'b'

c. b's followed by the symbol 'a'

d. string which starts and ends with the same symbol.

Let the CFG, $G = (V, T, P, S)$,

where $V = \{S\}$

$T = \{a, b\}$

$P = \{$

$S \to \in$ [By 1]

$S \to aSb$ [By 2]

$S \to bSa$ [By 3]

$S \to SS$ [By 4]

$\}$

S is the start symbol.

Derivation for the string $abba$ for $n_a(w) = n_b(w)$ is shown below:

$$S \Rightarrow SS \Rightarrow aSbS \Rightarrow abS \Rightarrow abbSa \Rightarrow abba.$$

EXAMPLE 9.8.12: Obtain the CFG to generate the language $L = \{w | n_a(w) > n_b(w)\}$.

Solution: The following cases are considered:

a. $\in$ (denotes) zero a s and zero b s
b. a followed by the symbol 'b'
c. b followed by the symbol a
d. string that starts and ends with same symbol
e. to produce one extra 'a' than 'b'
f. to insert as many 'a's' as possible.

Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, X, Y\}$

$T = \{a, b\}$

$P = \{$

$S \to XY|YX|XYX$ [By 6].

$X \to aXb$

$X \to bXa$

$X \to XX$

$X \to \in$

$Y \to aY|a$ [By 5].

$\}$

S is the start symbol.

EXAMPLE 9.8.13: Obtain the CFG to generate the language $L = \{w | n_a(w) = 2n_b(w)\}$.

Solution: The following cases are considered:

a. $\in$ (denotes) zero a s and zero b s
b. 'aa' followed by the symbol 'b'
c. 'b' followed by the symbols 'aa'
d. string that starts and ends with same symbol.

Let the CFG, $G = (V, T, P, S)$,

where $V = \{S\}$

$T = \{a, b\}$

$P = \{$

$S \rightarrow \in$

$S \rightarrow aaSb$

$S \rightarrow bSaa$

$S \rightarrow SS$

$\}$

S is the start symbol.

EXAMPLE 9.8.14: Obtain a CFG to generate the language $L = \{w | n_a(w) \neq n_b(w)\}$.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A, B\}$

$T = \{a, b\}$

$P = \{$

$S \rightarrow A|B$

$A \rightarrow a|Aa|aA|bAA|AbA|AAb$

$B \rightarrow b|Bb|bB|aBB|BaB|BBa$

$\}$

S is the start symbol.

EXAMPLE 9.8.15: Obtain a grammar to generate the language $L = \{ww^R | w \in \{a, b\}^*\}$, where w^R is the reverse of w.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S\}$

$T = \{a, b\}$

$P = \{$

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow \in$

$\}$

S is the start symbol.

EXAMPLE 9.8.16: Obtain a grammar to generate the language $L = \{wcw^R | w \in \{a, b\}^*\}$.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S\}$

$T = \{a, b\}$

$P = \{$

$S \rightarrow aSa$

$S \rightarrow bSb$

$S \rightarrow C$

$\}$

S is the start symbol.

EXAMPLE 9.8.17: Obtain a CFG to generate the language $L = \{0^n 1^n | n \geq 0\}$.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S\}$

$T = \{0, 1\}$

$P = \{$

$S \rightarrow 0S1$

$S \rightarrow \in$

$\}$

S is the start symbol.

To derive 0^n1^n, we apply the first production $n-1$ times, followed by an application of second production. This gives

$$\begin{aligned} S &\Rightarrow 0S1 \\ &\Rightarrow 00S11 \\ &\Rightarrow 0^2S1^2 \\ &\Rightarrow \cdots \\ &\Rightarrow 0^{n-1}S1^{n-1} \\ &\Rightarrow 0^n1^n \end{aligned}$$

EXAMPLE 9.8.18: Construct a CFG to generate the language $L\{a^nb^{2n}|n \geq 1\}$.

Solution: Since $n \geq 1$, $\in$ production are not considered. Let the CFG, $G = (V, T, P, S)$,

where $V = \{S\}$

$T = \{a, b\}$

$P = \{$

$S \rightarrow aSbb$

$S \rightarrow abb$

$\}$

S is the start symbol.

To derive a^nb^{2n}, we apply the first production $n-1$ times, followed by an application of second production. This gives

$$\begin{aligned} S &\Rightarrow aSbb \\ &\Rightarrow aaSbbbb \\ &\Rightarrow a^2Sb^4 \\ &\Rightarrow \ldots \\ &\Rightarrow a^{n-1}Sb^{2(n-1)} \\ &\Rightarrow a^nb^{2n}. \end{aligned}$$

EXAMPLE 9.8.19: Obtain a grammar to generate the language $L = \{0^n1^{n+1}|n \geq 0\}$

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A\}$

$T = \{0, 1\}$

$P = \{$

$S \to A1$

$A \to 0A1$

$A \to \in .$

$\}$

S is the start symbol.

To derive $0^n 1^{n+1}$, we apply the second production $n - 1$ times, followed by an application of third production. This gives

$$S \Rightarrow A1$$
$$S \Rightarrow 0A1 \cdot 1$$
$$S \Rightarrow 00A11 \cdot 1$$
$$S \Rightarrow 0^2A1^2 \cdot 1$$
$$\Rightarrow \cdots$$
$$\Rightarrow 0^{n-1}A1^{n-1} \cdot 1$$
$$\Rightarrow 0^n1^n \cdot 1$$
$$\Rightarrow 0^n1^{n+1}.$$

EXAMPLE 9.8.20: Obtain the grammar to generate the language $L = \{a^m b^m c^n | m \geq 1 \text{ and } n \geq 0\}$.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A\}$

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

$P = \{$

$S \to A|Sc$

$A \to ab|aAb$

$\}$

S is the start symbol.

The derivation for $a^m b^m c^n$ is as follows:

$$
\begin{aligned}
S &\Rightarrow Sc \\
&\Rightarrow Scc \\
&\Rightarrow Acc \\
&\Rightarrow aAbcc \\
&\Rightarrow aaAbbcc \\
&\Rightarrow a^2Ab^2c^2 \\
&\Rightarrow \ldots \\
&\Rightarrow a^{m-1}Ab^{m-1}c^n \\
&\Rightarrow a^m b^m c^n.
\end{aligned}
$$

EXAMPLE 9.8.21: Obtain a CFG to generate $L = \{ab(bbaa)^n bba(ba)^n | n \geq 0\}$.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A\}$

$T = \{a, b\}$

$P = \{$

$S \rightarrow abA$

$A \rightarrow bbaaAba|bba$

$\}$

S is the start symbol.

EXAMPLE 9.8.22: Obtain a grammar to generate the language $L = \{a^m b^n | m \neq n, m \geq 0, n \geq 0\}$.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A, B, C\}$

$T = \{a, b\}$

$$P = \{$$
$$S \to aSb$$
$$S \to A$$
$$S \to B$$
$$A \to aA|Aa|a$$
$$B \to bB|Bb|b$$
$$\}$$

S is the start symbol.

EXAMPLE 9.8.23: Obtain a grammar to generate the language $L = \{w : |w| \mod 3 = 0\}$ over $\Sigma = \{a\}$.

Solution: We need to generate the grammar for the language

$$L = \{\in, aaa, aaaaaa, \ldots\}.$$

In other words, any string w generated should have the length, equal to a multiple of 3.

Let the CFG, $G = (V, T, P, S)$,

where $V = \{S\}$

$T = \{a\}$

$P = \{$

$S \to \in$

$S \to aaaS$

$\}$

S is the start symbol.

EXAMPLE 9.8.24: Find a grammar for language $L = \{a^m b^n | m, n \in N \text{ and } n > m\}$.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A\}$

$T = \{a, b\}$

$P = \{$

$S \to aSb|aAb$

$A \to bA|b.$

$\}$

S is the start symbol.

EXAMPLE 9.8.25: Find a grammar of the language $L = \{a^n bc^n | n \in N\}$

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A\}$

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

$P = \{$

$S \to \in$

$S \to abA$

$A \to cS|c$

$\}$

S is the start symbol.

EXAMPLE 9.8.26: Find the grammar for language over $\Sigma = \{a, b\}$, in which all words are of the form $a^X b^Y a^Z$. Here, $X, Y, Z = 1, 2, 3, \ldots$ and $Y = 5X + 7Z$.

Solution: Let the CFG, $G = (V, T, P, S)$,

where $V = \{S, A, B\}$

$T = \{a, b\}$

$P = \{$

$S \to AB$

$A \to aAb^5 | \in$

$B \to b^7 Ba | \in$

$\}$

S is the start symbol.

EXAMPLE 9.8.27: Obtain a grammar to generate the following over $\Sigma = \{a, b\}$.

a. Set of all strings with exactly one a.
b. Set of all strings with atleast one a.

Solution:

a. $G = (V, T, P, S)$ where $V = \{S, A\}$, $T = \{a, b\}$, $S = \{S\}$ and

$$P = \{$$
$$S \rightarrow bS|aA$$
$$A \rightarrow bA| \in$$
$$\}$$

b. $G = (V, T, P, S)$ where $V = \{S, A\}, T = \{a, b\}, S = \{S\}$ and

$$P = \{$$
$$S \rightarrow bS|aA$$
$$A \rightarrow aA|bA| \in$$
$$\}.$$

9.9 Derivation Tree

When deriving a string w from S, if every derivation is considered to be a step in the tree construction, then the graphical display of the derivation of string w results in a tree structure. This is called a *derivation tree* or *parse tree* or *generation tree* or *production tree*.

Thus a derivation tree is the display of derivations, as a tree. A tree is said to be a derivation tree if it satisfies the following requirements:

a. All leaf nodes of the tree are labelled by terminals of the grammar.
b. The root of the tree is labelled with start symbol of the grammar.
c. The interior nodes are labelled using non-terminals.
d. If an interior node has a label A, and it has n descendents with labels $x_1, x_2, \ldots, x_n$ from left to right, then the production rule $A \rightarrow x_1x_2x_3 \ldots, x_n$ must exist in the grammar.

The derivation tree is useful to display the derivations as trees. A structure of trees for the words of a language is useful in applications such as the compilation of programming languages.

9.9.1 *Formal Definition*

Let $G = (V, T, P, S)$ be a CFG. A tree is a derivation tree for G, if and only if,

a. every vertex has a label, which is a symbol of terminal (T), non-terminal (V) or the null string $\in$,

b. the root of the tree has start symbol of the grammar as its label (S),
c. if a vertex is interior and has a label A, then $A \in V$,
d. if A is a label and $A \to x_1, x_2, \ldots, x_n$ is the production, then the production rule $A \to x_1 x_2 x_3 \ldots, x_n$ must exist in the grammar.

EXAMPLE 9.9.1: (Derivation tree)

a. Consider $G = (\{S, A\}, \{a, b\}, P, S)$, where P is,

$$S \to aAS|a$$
$$A \to SbA|SS|ba.$$

A derivation tree of G, to obtain the string $w = aabbaa$, is given by:

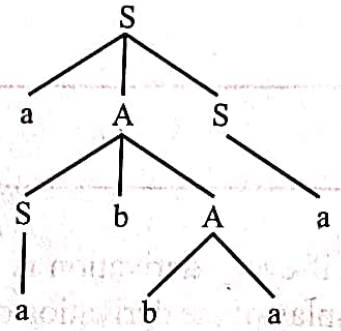


Figure 9.3. *Parse tree for the string $w = aabbaa$.*

9.9.2 *Yield of a derivation tree*

The yield of a tree is the string of symbols obtained by only reading the leaves of the tree from left to right, without considering the $\in$-symbols. The yield is always derived from the root and is always a terminal string.

EXAMPLE: The yield of the tree in figure 9.3 is *aabbaa*.

9.9.3 *Subtree of a derivation tree*

A subtree of a derivation tree is a particular vertex of the tree, together with all its descendants, the edges connecting them and their labels. It looks just like a derivation tree, except that the label of the root may not be the start symbol of the grammar. If the variable A labels the root, then we call the subtree an A-tree. Thus 'S-tree' is a synonym for 'derivation tree', if S is the start symbol.

EXAMPLE 9.9.2: (Subtree)
For the derivation tree of G in figure 9.3 the subtree is

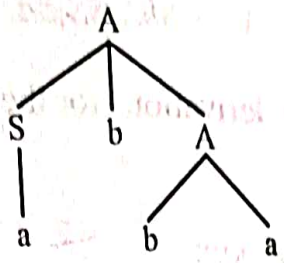


Figure 9.4. *Subtree of Figure 9.8*

The label of the root of this subtree is A and hence it's an A-tree.

EXAMPLE 9.9.3: For the grammar G with production rules

$$E \to E + E$$
$$E \to E * E$$
$$E \to id.$$

where $V = \{E\}, T = \{id\}$, obtain the derivation and the derivation tree for the string $w = id + id * id$.

Solution: Derivation for w:

$$\begin{aligned} E &\Rightarrow E + E. \\ &\Rightarrow E + E * E. \\ &\Rightarrow id + E * E. \\ &\Rightarrow id + id * E. \\ &\Rightarrow id + id * id. \end{aligned}$$

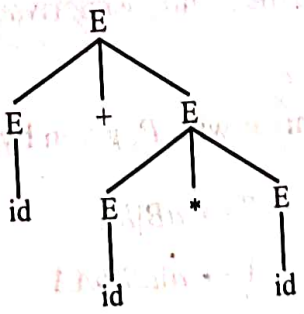


Figure 9.5. *Derivation tree for* $id + id * id$.

EXAMPLE 9.9.4: For $G = (\{S, A\}, \{a, b\}, S, P)$, where P is,

$$S \rightarrow aAS|a$$
$$A \rightarrow SbA|SS|ba.$$

Find the left most and the rightmost derivations for the string *aabbaa*. Also, draw the parse tree.

Solution: Leftmost derivation of *aabbaa*:

$$\begin{aligned} S &\Rightarrow aAS \\ &\Rightarrow aSbAS \\ &\Rightarrow aabAS \\ &\Rightarrow aabbaS \\ &\Rightarrow aabbaa \end{aligned}$$

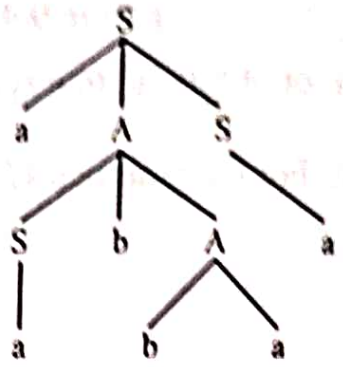


Figure 9.6 *Parse tree for leftmost derivation of the string aabbaa*

Rightmost derivation of *aabbaa*:

$$\begin{aligned} S &\Rightarrow aAS \\ &\Rightarrow aAa \\ &\Rightarrow aSbAa \\ &\Rightarrow aSbbaa \\ &\Rightarrow aabbaa \end{aligned}$$

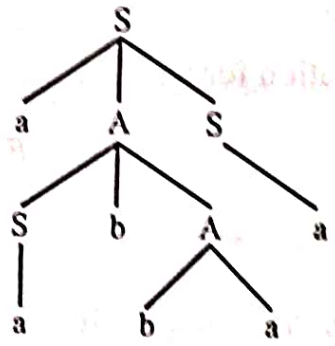


Figure 9.7 *Parse tree for rightmost derivation of the string aabbaa.*

Note-3: Parse tree for the left and the rightmost derivations is same in the above example.

EXAMPLE 9.9.5: Let G be a grammar with P, given by:

$$S \rightarrow aB|bA$$
$$A \rightarrow a|aS|bAA$$
$$B \rightarrow b|bS|aBB.$$

For the string $w = aaabbabbba$, find the left and the rightmost derivations and also draw the parse tree.

Solution: Leftmost derivation

$$
\begin{aligned}
S &\Rightarrow aB \\
&\Rightarrow aaBB \\
&\Rightarrow aaaBBB \\
&\Rightarrow aaabBB \\
&\Rightarrow aaabbSB \\
&\Rightarrow aaabbaBB \\
&\Rightarrow aaabbabB \\
&\Rightarrow aaabbabbS \\
&\Rightarrow aaabbabbbA \\
&\Rightarrow aaabbabbba
\end{aligned}
$$

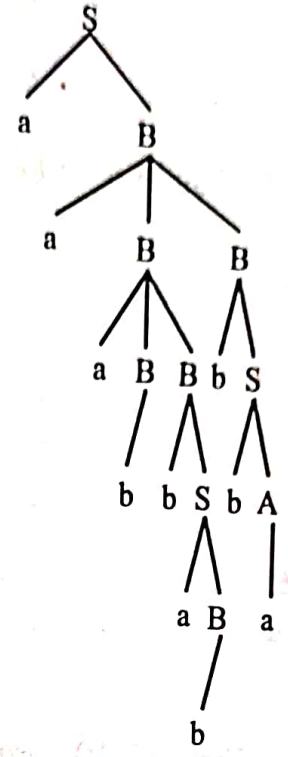


Figure 9.8 *Parse tree for leftmost derivation of the string w*

ightmost derivation:

$$
\begin{aligned}
S &\Rightarrow aB \\
&\Rightarrow aaBB \\
&\Rightarrow aaBbS \\
&\Rightarrow aaBbbA \\
&\Rightarrow aaBbba \\
&\Rightarrow aaaBBbba \\
&\Rightarrow aaaBbbba \\
&\Rightarrow aaabSbbba \\
&\Rightarrow aaabbAbbba \\
&\Rightarrow aaabbabbba
\end{aligned}
$$

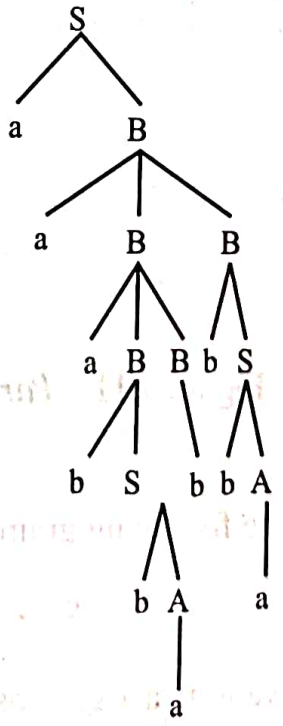


Figure 9.9 *Parse tree for Rightmost derivation of the string w.*

EXAMPLE 9.9.6: Obtain the left most and the rightmost derivations and parse tree for the grammar, whose production rules are:

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow id.$$

given the string for derivation as $w = id + id * id$.

Solution: Leftmost derivation

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow id + E \\ &\Rightarrow id + E * E \\ &\Rightarrow id + id * E \\ &\Rightarrow id + id * id \end{aligned}$$

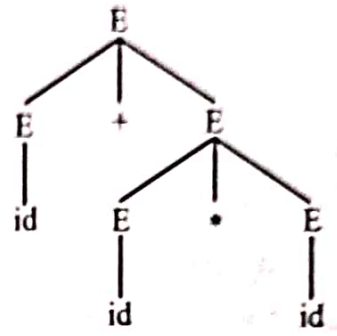


Figure 9.10 *Parse tree for leftmost derivation of the string w*

Rightmost derivation

$$\begin{aligned} E &\Rightarrow E * E \\ &\Rightarrow E * id \\ &\Rightarrow E + E * id \\ &\Rightarrow E + id * id \\ &\Rightarrow id + id * id \end{aligned}$$

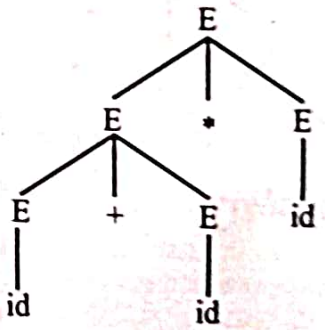


Figure 9.11 *Parse tree for rightmost derivation of the string w*

EXAMPLE 9.9.7: Given the following grammar:

$$S \rightarrow S[S] | \in,$$

construct a leftmost derivation, a rightmost derivation and a parse tree for each of the following strings:

a. [ ]

Solution: Leftmost derivation:

$$\begin{aligned} S &\Rightarrow S[S] \\ &\Rightarrow [S] \\ &\Rightarrow [\,] \end{aligned}$$

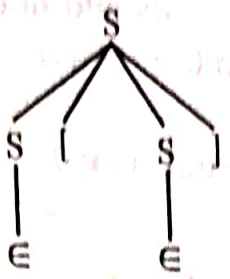


Figure 9.12 *Parse tree for leftmost derivation of [ ]*

Rightmost derivation:

$$\begin{aligned} S &\Rightarrow S[S] \\ S &\Rightarrow S[\,] \\ S &\Rightarrow [\,] \end{aligned}$$

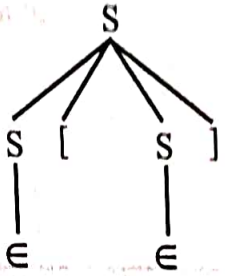


Figure 9.13 *Parse tree for rightmost dervation of [ ]*

b. [[]]

Solution: Leftmost derivation:

$$\begin{aligned} S &\Rightarrow S[S] \\ S &\Rightarrow [S] \\ &\Rightarrow [S[S]] \\ &\Rightarrow [[S]] \\ &\Rightarrow [[\,]] \end{aligned}$$

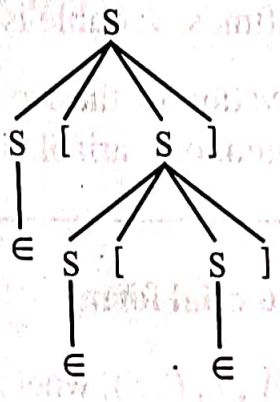


Figure 9.14 *Parse tree for leftmost derivation of [[ ]]*

Rightmost derivation:

$$\begin{aligned} S &\Rightarrow S[S] \\ S &\Rightarrow S[S[S]] \\ &\Rightarrow S[S[\,]] \\ &\Rightarrow S[[\,]] \\ &\Rightarrow [[\,]] \end{aligned}$$

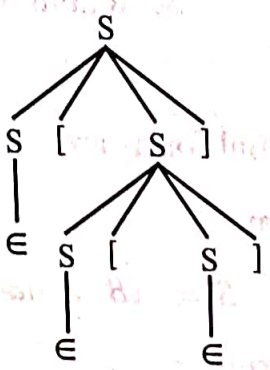
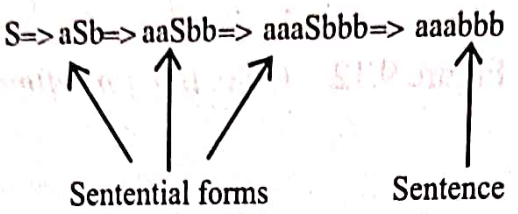


Figure 9.15 *Parse tree for rightmost derivation of [[ ]]*

9.9.4 *Sentential form*

A string, made up of terminals and non-terminals, is called a sentential form (a sentence that contains variables and terminals).

EXAMPLE 9.9.8: (Sentential form)



Formal definition:

Let $G = (V, T, P, S)$ be a CFG. Any sting $w \in (V \cup T)^*$, which is derivable from the start symbol S (denoted by $S \overset{*}{\Rightarrow} \alpha$), is called a sentence or sentential from of G.

- If there is a derivation of the form $S \overset{*}{\Longrightarrow} \alpha$, where at each step in the derivation process only a leftmost variable is replaced, then α is called *left-sentential form.*
- If there is a derivation of the from $S \overset{*}{\Longrightarrow} \alpha$, where at each step in the derivation process only a rightmost variable is replaced, then α is called *right-sentential form*

EXAMPLE 9.9.9: (Sentential form)

For the grammar $G = (V, T, P, S)$, where P is

Sentential form

$$S \to AB$$
$$A \to aaA \mid \in$$
$$B \to Bb \mid \in$$

the left and right sentential forms are:

a. Left-sentential form:

$$S \Rightarrow AB \Rightarrow aaAB \Rightarrow aaB \Rightarrow aaBb \Rightarrow aab.$$

b. Right-sentential form:

$$S \Rightarrow AB \Rightarrow ABb \Rightarrow Ab \Rightarrow aaAb \Rightarrow aab$$

9.10 Ambiguous and Unambiguous Grammars

- A CFG is ambiguous, if for atleast one word in its CFL, there are two possible derivations of the word that correspond to two different syntax or parse trees.
- A grammar is said to be ambiguous, if its language contains some string that has two different parse trees.
- A grammar is said to be unambiguous, if its language has exactly one parse tree.
- If grammar G is ambiguous, then for some w in $L(G)$, there exists more than one parse tree. Hence, there is more than one leftmost order of derivation and equivalently, there is more than one rightmost order of derivation.
- If grammar G is unambiguous, then for some w in $L(G)$, there exists exactly one parse tree. Hence, there exists exactly one leftmost order of derivation and equivalently one rightmost order of derivation.

Note 4: A grammar is ambiguous if there are more than one leftmost (or rightmost) derivation for given w.

EXAMPLE 9.10.1: (Ambiguous and unambiguous grammar)

a. Consider a grammar for arithmetic expressions:

$$E \rightarrow a|b$$

$$E \rightarrow E - E.$$

This grammar is ambiguous, because, to derive the string $w = a - b - a$, there exist two distinct parse trees, as shown in figure 9.21.

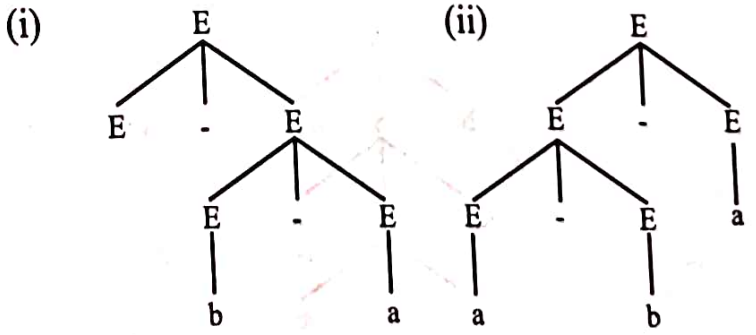


Figure 9.16. *Two distinct parse trees for the derivations of string $w = a - b - a$.*

There are two distinct parse trees, which means that there are two distinct left or rightmost derivations.

Leftmost derivations:

(i) $E \Rightarrow E - E \Rightarrow a - E \Rightarrow a - E - E \Rightarrow a - b - E \Rightarrow a - b - a$

(ii) $E \Rightarrow E - E \Rightarrow E - E - E \Rightarrow a - E - E \Rightarrow a - b - E \Rightarrow a - b - a$

The same is the case with rightmost derivation.

b. Consider the grammar

$$S \rightarrow aS|a \quad \text{where,} \quad V = \{S\} \text{ and } T = \{a\}.$$

This grammar is unambiguous, because to derive the string $w = aa$, there exists exactly one parse tree, as shown in figure 9.22.

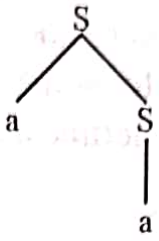


Figure 9.17. *Parse tree to derive $w = aa$.*

There is, exactly one parse tree and hence, exactly one leftmost or rightmost derivation.

Leftmost derivation: $S \Rightarrow aS \Rightarrow aa$.

Rightmost derivation: $S \Rightarrow aS \Rightarrow aa$.

EXAMPLE 9.10.2: Given below is the grammar for palindrome over alphabets $\{a, b\}$. Verify whether it is ambiguous or unambiguous.

$$S \rightarrow aSa|bSb|a|b|\in .$$

Solution: Consider $w = babbab$.

Derivation: $S \Rightarrow bSb \Rightarrow baSab \Rightarrow babSbab \Rightarrow babbab$.

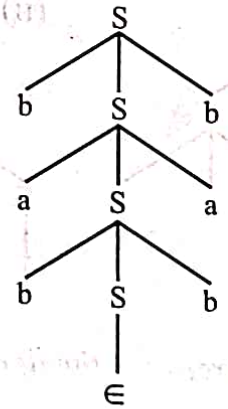


Figure 9.18. *Parse tree of the string w.*

There is exactly one parse tree and hence the grammar is unambiguous.

EXAMPLE 9.10.3: Show that the grammar

$$S \rightarrow aS|aSb|x$$
$$X \rightarrow Xa|a$$

where $V = \{S, X\}$ and $T = \{a, b\}$ is ambiguous.

Solution: The word $w = aa$ has two different leftmost derivations, that correspond to different parse trees.

a. $S \Rightarrow aS \Rightarrow aX \Rightarrow aa$.

Figure 9.19. *First Parse tree to derive $w = aa$.*

b. $S \Rightarrow X \Rightarrow Xa \Rightarrow aa$.

Figure 9.20. *Second Parse tree to derive $w = aa$.*

The same is the case with rightmost derivation. Hence, the grammar is ambiguous.

EXAMPLE 9.10.4: Prove that $S \rightarrow aSbS|bSaS| \in$ is ambiguous.

Solution: The word $w = abab$ has two different leftmost derivations, that correspond to different parse trees.

a. $S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab$.

EXAMPLE 9.10.3: Show that the grammar

$$S \to aS|aSb|x$$
$$X \to Xa|a$$

where $V = \{S, X\}$ and $T = \{a, b\}$ is ambiguous.

Solution: The word $w = aa$ has two different leftmost derivations, that correspond to different parse trees.

a. $S \Rightarrow aS \Rightarrow aX \Rightarrow aa$.

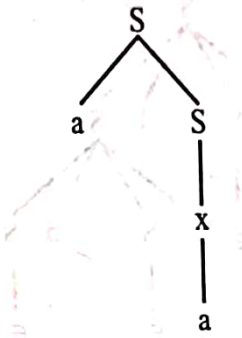


Figure 9.19. *First Parse tree to derive* $w = aa$.

b. $S \Rightarrow X \Rightarrow Xa \Rightarrow aa$.

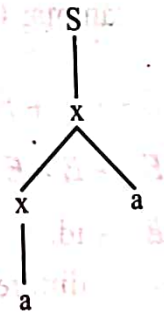


Figure 9.20. *Second Parse tree to derive* $w = aa$.

The same is the case with rightmost derivation. Hence, the grammar is ambiguous.

EXAMPLE 9.10.4: Prove that $S \to aSbS|bSaS| \in$ is ambiguous.

Solution: The word $w = abab$ has two different leftmost derivations, that correspond to different parse trees.

a. $S \Rightarrow aSbS \Rightarrow abSaSbS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab$.

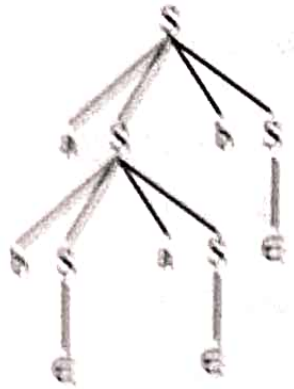


Figure 9.21. *First Parse tree to derive* $w = abab$.

b. $S \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab$.

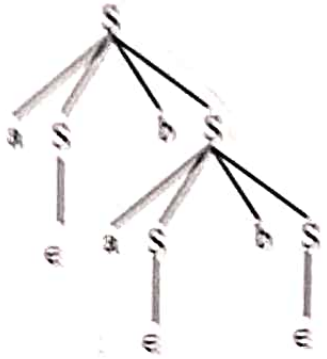


Figure 9.22. *Second Parse tree to derive* $w = abab$.

The same is the case with rightmost derivation. Hence, the grammar is ambiguous.

EXAMPLE 9.10.5: Show that the given grammar G is ambiguous.

$$E \rightarrow E + E$$
$$E \rightarrow E * E$$
$$E \rightarrow \text{id}.$$

Solution: The word $w = id + id * id$ has two different leftmost derivations, that correspond to different parse trees.

a. $E \Rightarrow E + E \Rightarrow id + E \Rightarrow id + E * E \Rightarrow id + id * E \Rightarrow id + id * id$.

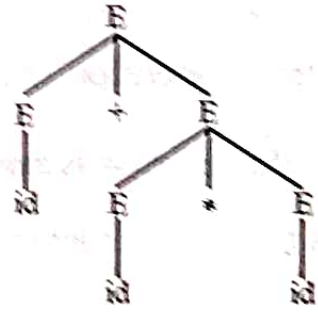


Figure 9.23. *First Parse tree to derive* $w = id + id * id$.

b. $E \Rightarrow E * E \Rightarrow E + E * E \Rightarrow \text{id} + E * E \Rightarrow id + id * E \Rightarrow id + id * id.$

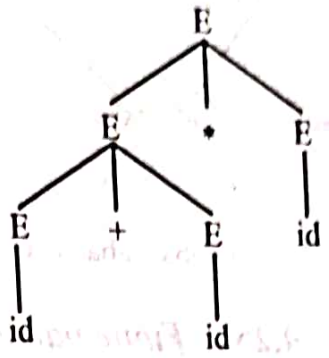


Figure 9.24. *Second Parse tree to derive $w = id + id * id$.*

The same is the case with rightmost derivation. Hence, the grammar is ambiguous.

9.11 Total Language Tree

Definition: For a given CFG, the total language tree is the tree

- with root S,
- whose children are all the productions of S,
- whose second descendants are all the working strings, that can be constructed by applying one production to the leftmost non-terminal in each of the children and so on.

EXAMPLE 9.11.1: For the grammar $G = \{V, T, P, S\}, V = \{S, X\}, T = \{a, b\}$ with production rules

$$S \rightarrow aX|Xa|aXbXa$$
$$X \rightarrow ba|ab,$$

the CFG has total language tree as shown in figure 9.25.

The language of CFG i.e. CFL is finite.

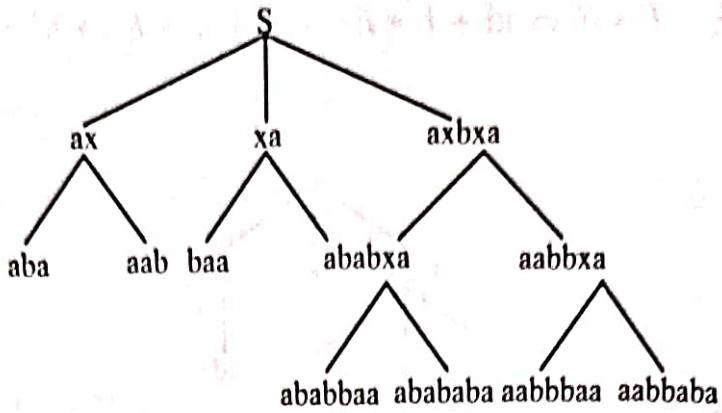


Figure 9.25. *Finite parse tree.*

EXAMPLE 9.11.2: For the grammar $G = \{V, T, P, S\}$ where, $V = \{S, X\}, T = \{a, b\}$ and P is

$$S \rightarrow aSb|aX$$

$$X \rightarrow bX|a,$$

We can construct the total language tree as:

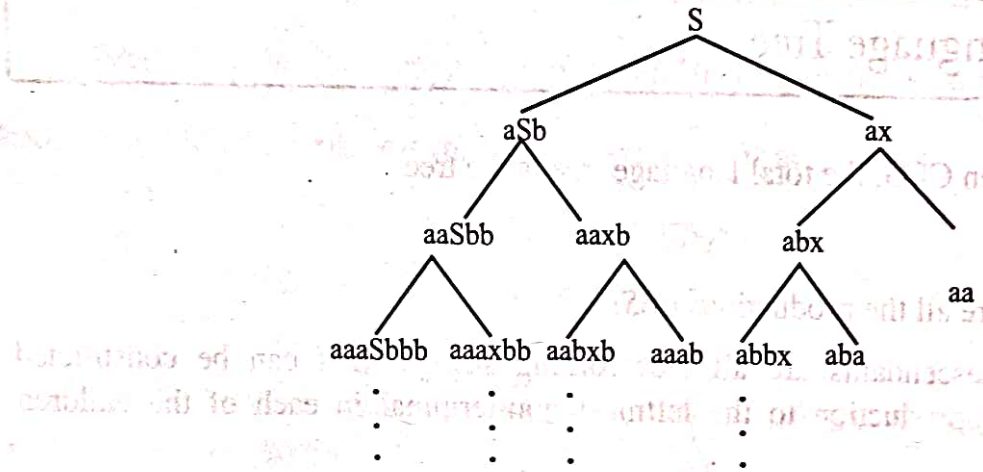


Figure 9.26. *Infinite parse tree.*

The CFL of this CFG is infinite.

EXAMPLE 9.11.3: For grammar $G = \{V, T, P, S\}, V = \{S, X\}, T = \{a\}$, with P as

$$S \rightarrow X|a$$

$$X \rightarrow aX,$$

the total language tree is:

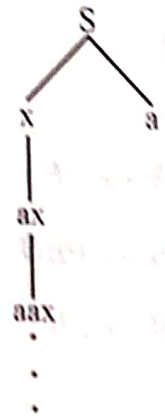


Figure 9.27. *Infinite parse tree.*

The tree is infinite, but $CFL = \{a\}$.

9.13 Relation between Regular Grammar and Finite Automata

Since the language accepted by finite automata is a regular language, hence the following relation exists between regular grammar and finite automaton.

a. **Finite automaton from regular grammar**
For a given right-linear grammar G, there exists a language $L(G)$, which is accepted by a finite automaton.

b. **Regular grammar from finite automaton**
For a given finite automaton M, there exists a right-linear grammar G such that $L(M) = L(G)$.

c. **Regular expression from regular grammar**
For a given regular G, there exists a regular expression that specifies $L(G)$.

9.13.1 *Finite automaton from regular grammar*

Let $G = (V, T, P, S)$ be a right-linear grammar. Let $V = \{V_0, V_1, \ldots\}$ be the variables and productions P (of G) be:

$$P = \{\ V_i \rightarrow a_1 a_2, \ldots a_m V_j$$

or

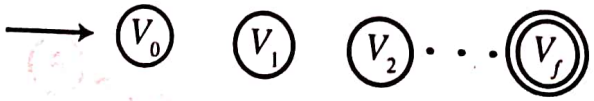
$$V_i \rightarrow a_1, a_2, \ldots a_m$$

$\}$.

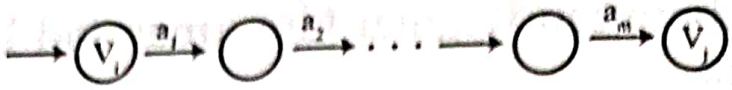
We construct a finite automaton $M = (Q, \Sigma, \delta, q_0, F)$, from these productions, by using the following steps:

Step-1: Start symbol of grammar is the start symbol of $M (q_0 = V_0)$.

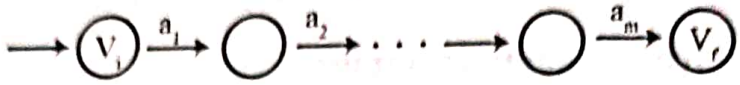
Step-2: Each variable V_i, of G, corresponds to a state in M.



Step-3: For each production of the form $V_i \rightarrow a_1 a_2 \ldots a_m V_j$, if the transitions of M are of the from $S(V_i, a_1 a_2 \ldots a_m) = V_j$, add the transitions and intermediate states.



Step-4: For each production of the form $V_i \rightarrow a_1 a_2 \ldots a_m$ if the transitions of M are of the form $(V_i, a_1 a_2 \ldots a_m) = V_f$ (where V_f is the final state of M), add the transitions and intermediate states.



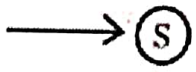
Note-5: For any production of the form $V_i \rightarrow \in$, then $S(V_i, \in) = V_i$ and V_i is also the final state of M.

EXAMPLE 9.13.1: Construct finite automaton to accept the language generated by the following grammar:

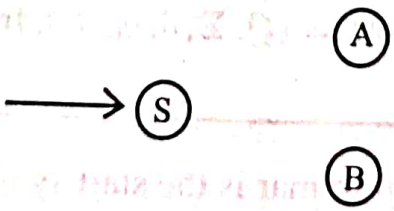
$$S \rightarrow aA|B$$
$$A \rightarrow aaB$$
$$B \rightarrow bB|a$$

Solution: Let G be the right-linear grammar and M the finite automata.

Step-1: The start symbol of G is the start symbol for M, i.e., $q_0 = \{S\}$.

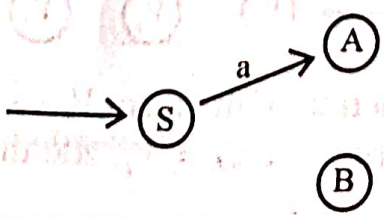


Step-2: Every variable $V = \{S, A, B\}$, of G, Corresponds to a state in M.

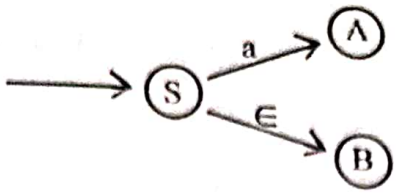


Step-3: Compute the transitions of M for all production of the form $V_i \rightarrow a_1 a_2 \ldots a_m V_j$.

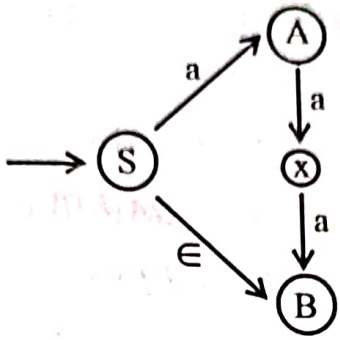
a. $S \rightarrow aA$



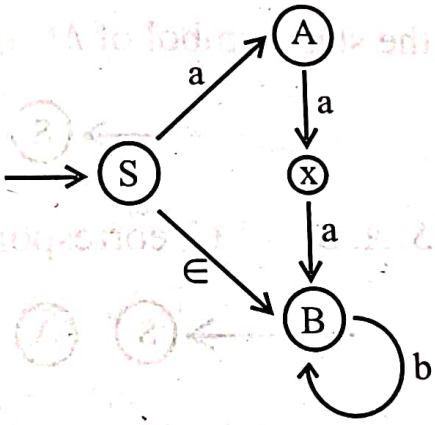
b. $S \rightarrow B$



c. $A \rightarrow aaB$
Add transitions and intermediate states (X).



d. $B \rightarrow bB$



p-4: Compute for all productions of the form $V_i \rightarrow a_1 a_2 \ldots a_m$
$B \rightarrow a$

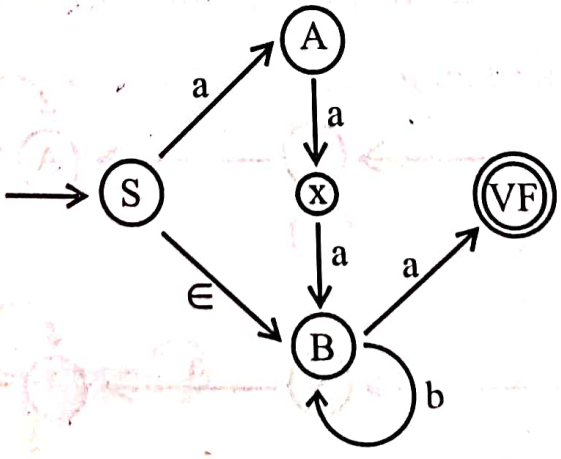


Figure 9.29. *Final M with* $L(M) = L(G) = aaab^*a + b^*a$.

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

where

$$Q = \{S, A, X, B, V_f\}$$
$$\Sigma = \{a, b\}$$
$$q_0 = S$$
$$F = V_f.$$

EXAMPLE 9.13.2: Construct a FA to accept the language generated by the following grammar:

$$S \to aA| \in$$
$$A \to aA|bB| \in$$
$$B \to bB| \in$$

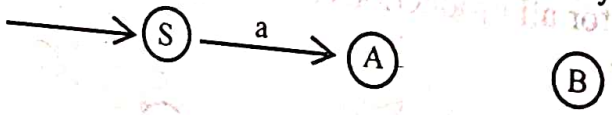
Solution: Let G be the right-linear grammar and M the finite automaton.

Step-1: Start symbol of G is the start symbol of M, $q_0 = \{S\}$.

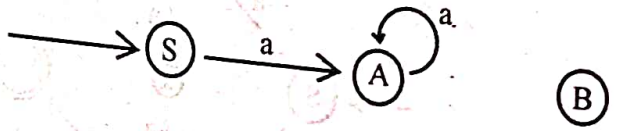
Step-2: Every variable $V = \{S, A, B\}$, of G, corresponds to a state in M.

Step-3: Compute for all productions of the form $V_i \to a_1 a_2 \ldots a_m V_j$.

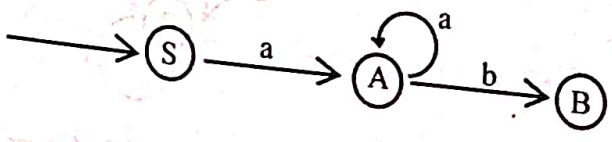
a. $S \to aA$



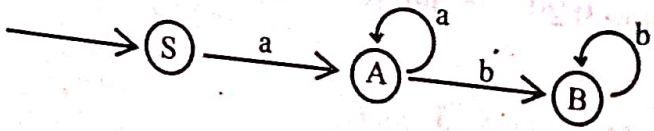
b. $A \to aA$



c. $A \to bB$.



d. $B \to bB$



There are productions of the form $V_f \to \in$, i.e., $S \to \in$, $A \to \in$ and $B \to \in$. Therefore $\{S, A, B\}$ are V_f. Thus, final M is:

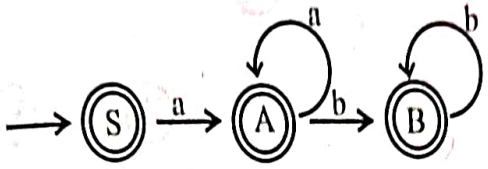


Figure 9.30. *Final M with* $L(M) = L(G)$.

Thus, $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{S, A, B\}$, $\Sigma = \{a, b\}$, $q_0 = \{S\}$ and F = {A, S, B}.

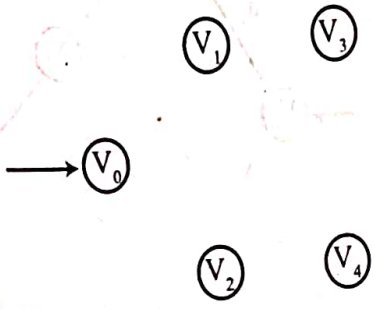
EXAMPLE 9.13.3: Construct the automaton for the grammar

$$V_0 \to a_1 V_1 | a_3 V_2$$
$$V_1 \to a_2 a_4 V_3 | a_3 a_4 a_8 V_4$$
$$V_2 \to a_5 V_4$$
$$V_3 \to a_9 V_1 | a_5$$
$$V_4 \to a_0$$

Solution: Let G be the right-linear grammar and M the finite automaton.

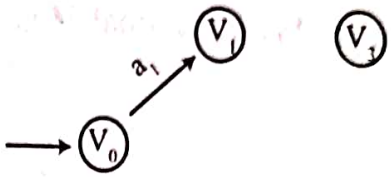
Step-1: Start symbol of G is the start symbol of M, $q_0 = \{V_0\}$.

Step-2: Every variable $V = \{S, A, B\}$, of G, corresponds to a state in M.

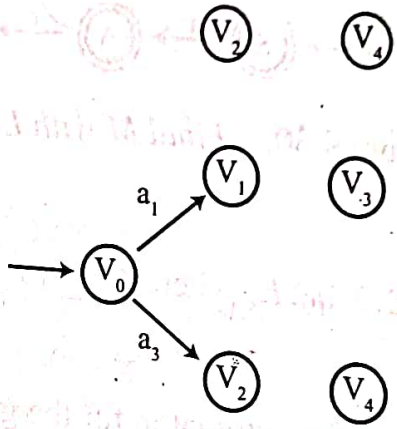


Step-3: Compute for all productions of the form $V_i \to a_1 a_2 \ldots a_m V_j$.

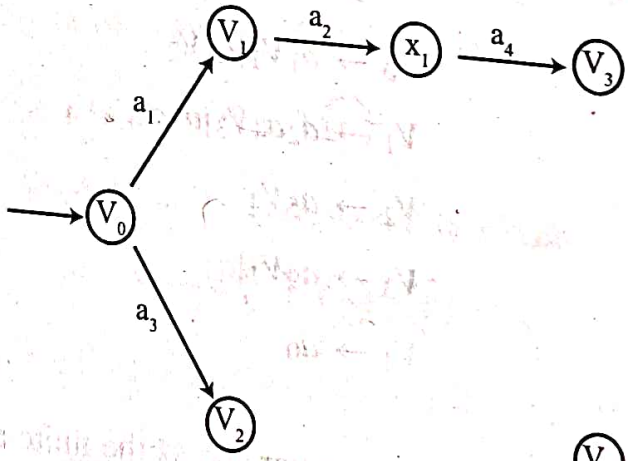
a. $V_0 \rightarrow a_1 V_1$



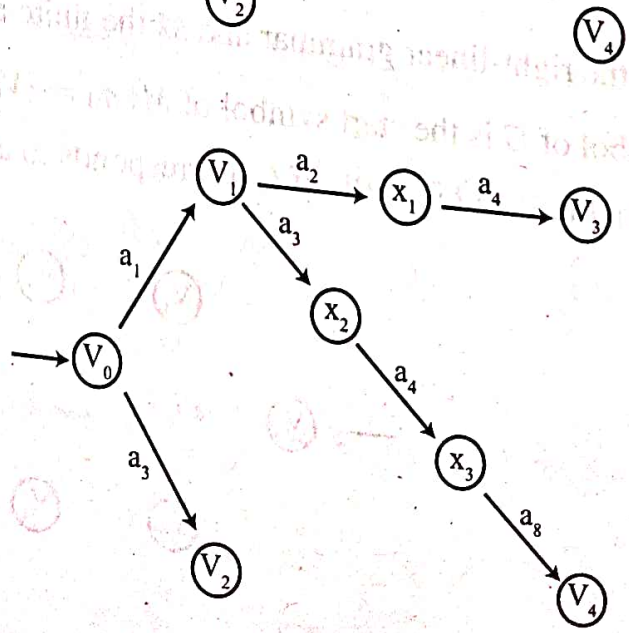
b. $V_0 \rightarrow a_3 V_2$



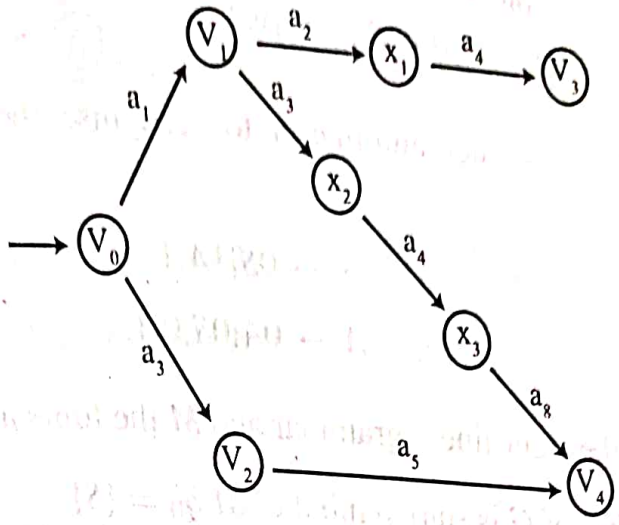
c. $V_1 \rightarrow a_2 a_4 V_3$



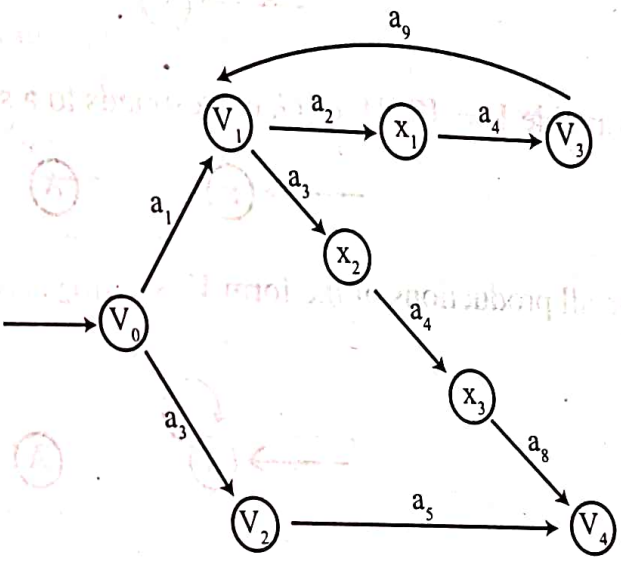
d. $V_1 \rightarrow a_3 a_4 a_8 V_4$



e. $V_2 \rightarrow a_5 V_4$



f. $V_3 \rightarrow a_9 V_1$



Step-4: Compute for all productions of the form $V_i \rightarrow a_1 a_2 \ldots a_m$ as computing $V_3 \rightarrow a_5$ and $V_4 \rightarrow a_0$ gives us the final automata shown in figure 9.31.

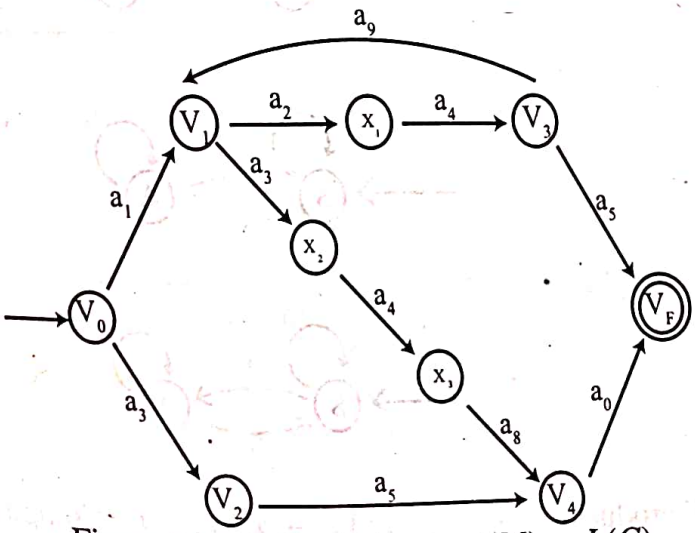


Figure 9.31. *Final M with* $L(M) = L(G)$.

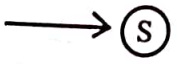
Thus, $M = (Q, \Sigma, \delta, q_0, F)$ where $Q = \{v_0, v_1, v_2, v_3, v_4, x_1, x_2, x_3\}$, $[\Sigma = \{a_0, a_1, a_2, a_3, a_4, a_5, a_8, a_9\}$, $q_0 = \{v_0\}$ and $F = \{v_f\}$.

EXAMPLE 9.13.4: Construct automaton to recognise the language generated by the grammar:

$$S \to 0S|1A|1$$
$$A \to 0A|0S|0|1.$$

Solution: Let G be the right-linear grammar and M the finite automaton.

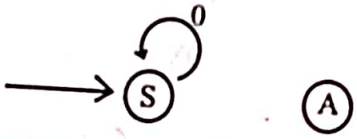
Step-1: Start symbol of G is start symbol of M $q_0 = \{S\}$.



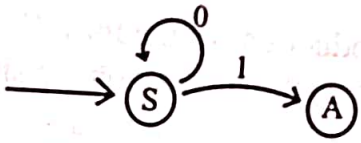
Step-2: Every variable $V = \{S, A\}$, of G, corresponds to a state in M.

Step-3: Compute all productions of the form $V_i \to a_1 a_2 \ldots a_m V_j$.

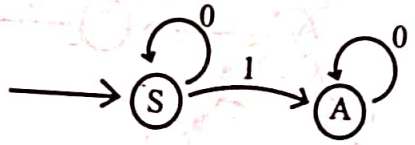
a. $S \to 0S$



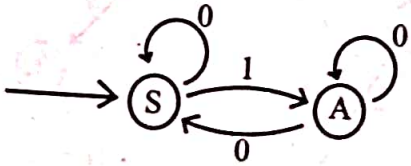
b. $S \to 1A$



c. $A \to 0A$



d. $A \to 0S$



Step-4: Compute all productions of the form $V_i \to a_1 a_2 \ldots a_m$. Computing $S \to 1, A \to 0$ and $A \to 1$ gives the final automaton as:

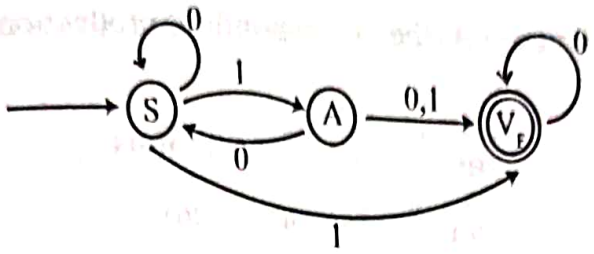


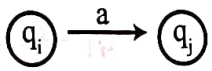
Figure 9.32. *Final M with L(M) = L(G).*

Thus, $M = (Q, \Sigma, \delta, q_0, F)$, where $Q = \{S, A, V_f\}$, $\Sigma = \{0, 1\}$ and $q_0 = \{s\}$, $F = \{V_f\}$.

9.13.2 *Regular grammar from Finite automata.*

Let $M = (Q, \Sigma, S, q_0, F)$ be a finite automata where $Q = \{q_0, q_1, \ldots q_n\}$, is the finite number of states and $\Sigma = \{a_1, a_2, \ldots a_m\}$ is the set of input symbols. Construct a regular grammar $G = (V, T, P, S)$, where $V = \{q_0, \ldots q_n\}$, $T = \Sigma$ and $S = q_0$, by using the following steps:

Step-1: For any transition of the form,



i.e., $\delta(q_i, a) = q_j$, the corresponding production is $q_i \rightarrow aq_j$.

Step-2: For any final state of M,

the production added is $q_f \rightarrow \in$.

EXAMPLE 9.13.5: Construct a regular grammar from the following finite automaton.

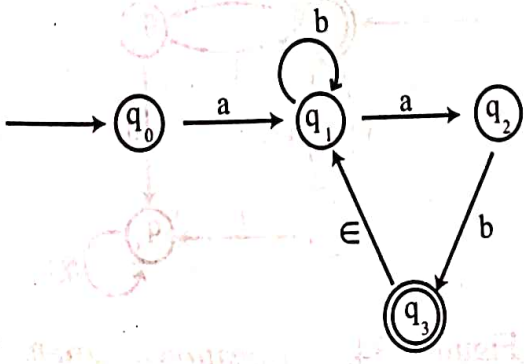


Figure 9.33. *Transition diagram*

Solution: For each transition of FA, the corresponding productions are shown below:

$q_i \xrightarrow{a} q_j$	**Productions**
$q_0 \xrightarrow{a} q_1$	$q_0 \to aq_1$
$q_1 \xrightarrow{b} q_1$	$q_1 \to bq_1$
$q_1 \xrightarrow{a} q_2$	$q_1 \to aq_2$
$q_2 \xrightarrow{b} q_3$	$q_2 \to bq_3$
$q_3 \xrightarrow{\in} q_1$	$q_3 \to q_1$
$q_3 = q_f$	$q_3 \to \in$

Thus, $G = (V, T, P, S)$ is the required grammar, where $V = \{q_0, q_1, q_2, q_3\}$, $T = \{a, b\}$, $S = \{q_0\}$, $F = \{q_3\}$ and

$$P = \{$$
$$q_0 \to aq_1$$
$$q_1 \to bq_1$$
$$q_1 \to aq_2$$
$$q_2 \to bq_3$$
$$q_3 \to q_1$$
$$q_3 \to \in$$
$$\}.$$

EXAMPLE 9.13.6: Obtain the regular grammar for the FA shown in figure 9.34.

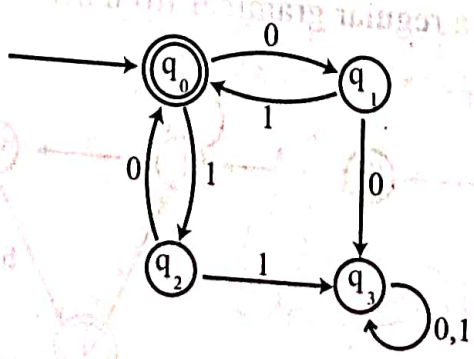


Figure 9.34. *Transition diagram.*

Solution: For each transition of FA, the corresponding productions are shown below:

$q_i \xrightarrow{a} q_j$	Productions
$q_0 \xrightarrow{0} q_1$	$q_0 \to 0q_1$
$q_1 \xrightarrow{1} q_0$	$q_1 \to 1q_0$
$q_0 \xrightarrow{1} q_2$	$q_0 \to 1q_2$
$q_2 \xrightarrow{0} q_0$	$q_2 \to 0q_0$
$q_2 \xrightarrow{1} q_3$	$q_2 \to 1q_3$
$q_1 \xrightarrow{0} q_3$	$q_1 \to 0q_3$
$q_3 \xrightarrow{0} q_3$	$q_3 \to 0q_3$
$q_3 \xrightarrow{1} q_3$	$q_3 \to 1q_3$
$q_0 = q_f$	$q_0 \to \in$

Thus, $G = (V, T, P, S)$, where $V = \{q_0, q_1, q_2, q_3\}$, $T = \{0, 1\}$, $S = \{q_0\}$, $F = \{q_0\}$ and P is given above.

EXAMPLE 9.13.7: Obtain the regular grammar from the *FA* given in figure 9.35.

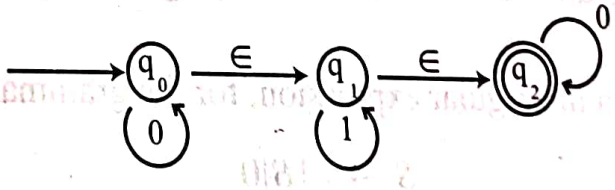


Figure 9.35. *Transition diagram.*

Solution: For each transition of *FA*, the corresponding productions are shown below:

$q_i \xrightarrow{a} q_j$	Productions
$q_0 \xrightarrow{0} q_0$	$q_0 \to 0q_0$
$q_0 \xrightarrow{\in} q_1$	$q_0 \to q_1$
$q_1 \xrightarrow{1} q_1$	$q_1 \to 1q_1$
$q_1 \xrightarrow{\in} q_2$	$q_1 \to q_2$
$q_2 \xrightarrow{0} q_2$	$q_2 \to 0q_2$
$q_f = q_2$	$q_2 \to \in$

Thus, $G = (V, T, P, S)$ is the required grammar, where, $V = \{q_0, q_1, q_2\}$, $T = \{0, 1\}$, $S = \{q_0\}$, $F = \{q_2\}$ and

$$P = \{$$
$$q_0 \rightarrow 0q_0$$
$$q_0 \rightarrow q_1$$
$$q_1 \rightarrow 1q_2$$
$$q_1 \rightarrow q_2$$
$$q_2 \rightarrow 0q_2$$
$$q_2 \rightarrow \in .$$
$$\}.$$

9.13.3 *Regular Expression from Regular Grammar*

Let $G = (V, T, P, S)$ be a right-linear grammar. A regular expression R, that specifies $L(G)$, can be directly obtained as follows.

a. Replace the '$\rightarrow$' symbols in the grammar's productions with '$=$' symbols to get a set of equations.
b. Convert the equations of the form $A = aA|b$, as $A = a^*b$.
c. Solve the set of equations obtained, to obtain the value of the variable S (where S is the start symbol of the grammar). The result is the regular expression, specifying $L(G)$.

EXAMPLE 9.13.8: Obtain the regular expression, for the grammar given below:

$$S \rightarrow 01B|0$$
$$B \rightarrow 1B|11.$$

Solution:

Step-1: Replace $\rightarrow$ by $=$ in the above productions, so that,

$$S = 01B|0 \quad \ldots\ldots (1)$$
$$B = |B|11. \quad \ldots\ldots (2)$$

Step-2: $B = 1B|11 \Rightarrow B = 1^*11$ $[\because A = aA|b$ is $A = a^*b]$

Step-3: Substituting for B in eq. (1), we get

$$S = 01B|0$$
$$\Rightarrow \quad S = 011^*11|0$$
$$\text{or} \quad S = (011^*11 + 0).$$

EXAMPLE 9.13.9: Obtain the regular expression for the grammar given below.

$$S \rightarrow baS|aA$$
$$A \rightarrow bbA|bb.$$

Solution:

Step-1: Replace $\rightarrow$ by $=$ in the above productions, so that,

$$S = baS|aA \quad \ldots\ldots (1)$$
$$A = bbA|bb. \quad \ldots\ldots (2)$$

Step-2: $A = bbA|bb \Rightarrow A = (bb)^*bb$. $(\because A = aA|b$ is $A = a^*b)$.

Step-3: Substituting for A in eqn (1), yields

$$S = baS|aA$$
$$S = baS|(a(bb)^*bb)$$
$$\Rightarrow S = (ba)^*(a(bb)^*bb).$$

EXAMPLE 9.13.10: Obtain the regular expression for the grammar given below.

$$S \rightarrow 0A|0|1B$$
$$A \rightarrow 1A|1$$
$$B \rightarrow 0B|1S$$

Solution:

Step-1: Replace $\rightarrow$ by $=$ in the above production, so that,

$$S = 0A|0|1B \quad \ldots\ldots (1)$$
$$A = 1A|1 \quad \ldots\ldots (2)$$
$$B = 0B|1S \quad \ldots\ldots (3)$$

Step-2:

$$A = 1A|1 \Rightarrow A = 1^*1$$
$$B = 0B|1S \Rightarrow B = 0^*1S$$

Step-3: Substituting for A and B in eq. (1), we get

$$S = 0A|0|1B$$
$$S = 01^*1|0|10^*1S$$
$$S = (10^*1)S|(01^*1|0)$$
$$S = (10^*1)^*(01^*1|0) \quad (\because A = aA|b \text{ is } A = a^*b)$$

or $S = (10^*1)^* \cdot (01^*1 + 0)$.

9.14 Exercises

1. Compare regular languages and context-free languages.
2. Define grammar. Explain the structure for writing a grammar.
3. Define production rule for grammar. Explain its different forms.
4. What is a context-free grammar? Explain with examples.
5. Explain different types of grammar with examples.
6. Draw a CFG on $\{a, b\}$, to generate a language $L = \{a^n ww^R b^n | w \in \Sigma^*, n \geq 1\}$.
7. Obtain a CFG to generate $L = \{w | w \in \{a, b\}^*, n_a(w) \geq n_b(w)\}$.
8. Obtain a grammar to generate the language $L = \{w : |w| \mod 3 > 0\}$ on $\Sigma = \{a\}$.
9. Obtain a grammar to generate the set of all strings, with no more than three a's on $\Sigma = \{a, b\}$.
10. Obtain a grammar to generate the language $L = \{a^i b^j c^k | i + 2j = k, i \geq 0, j \geq 0\}$.
11. Present the formal definition of derivation tree for G.
12. Explain with examples, the leftmost and rightmost derivation.
13. For the grammar $G = (V, T, P, S)$, where $V = \{s\}$, $T = \{a, b\}$ and

$$P = \{$$
$$S \rightarrow aSa|bSb| \in$$
$$\}$$

construct a leftmost, rightmost derivation and parse tree.

a. *aaaaaa*
b. *abbbba*
c. *bababbabab*

14. What is meant by ambiguous and unambiguous grammars?
15. Show that the following grammars are ambiguous:
    a. $S \rightarrow aS|X$
       $X \rightarrow aX|a$
    b. $S \rightarrow aSbS$
       $S \rightarrow bSaS| \in$
16. What is a regular grammar? Explain the relation between regular grammar and finite automaton.
17. Construct finite automaton to accept the language, generated by the following grammars:
    a. $S \rightarrow aS|X$
       $X \rightarrow aX|a$
    b. $S \rightarrow AB|aaB$
       $A \rightarrow a|Aa$
       $B \rightarrow b$
    c. $S \rightarrow SS|dS|Sd|c$
18. Construct a regular grammar, from the following finite automaton:
    a.

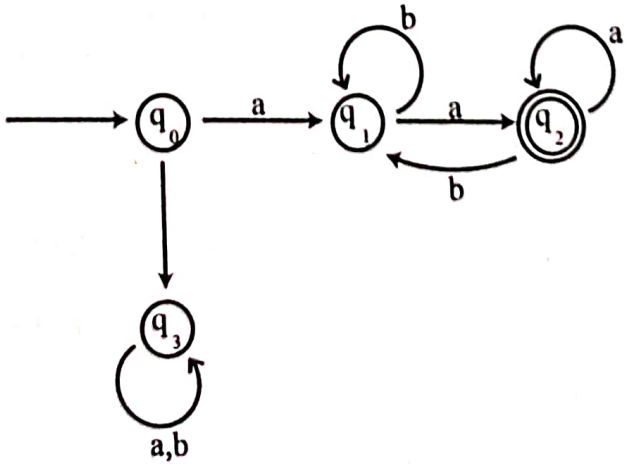


Figure 9.36. *Transition diagram*

b.

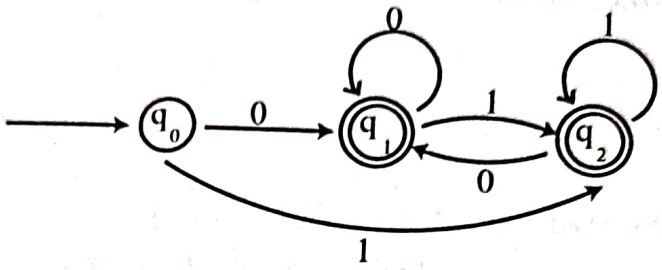


Figure 9.37. *Transition diagram*

19. Obtain regular expressions from the following grammar.

    a. $S \rightarrow AB|aaB$

    $A \rightarrow aA|a$

    $B \rightarrow b$

    b. $S \rightarrow XaaX$

    $X \rightarrow aX|bX|\in$

20. Write a procedure to transform a left-linear grammar to right-linear grammar.

10 Simplification of Context-Free Grammar

'The supreme excellence is simplicity.'

Introduction

In this chapter, the important and unimportant symbols in a CFG are discussed. Methods to eliminate the unwanted productions are also illustrated. This procedure leads to the reduction of a CFG into a more compact form. The salient features of the three normal forms, due to *Chomsky*, *Greibach* and *Kuroda*, are discussed and well-examined problems are included, to make the concepts clear.

10.1 Simplification of Context-Free Grammars

In a CFG, it is not necessary to use all the symbols in V or all the productions in P, for deriving sentences. So, we can reduce the complexity of the grammar, without reducing the generating power of CFG. This can be done by the following procedures:

- By eliminating the useless symbols.
- By eliminating the unit productions.
- By eliminating the null productions, if $\in$ is not included in the sentence.

10.1.1 *Useful and useless symbols in CFG*

A grammar symbol X in CFG in *useful*, if and only if:

a. it derives a string of terminals, and
b. it is used in derivation of atleast one w in $L(CFG)$.

Formally, in a CFG, $G = (V, T, P, S)$, X is *useful* if and only if:

a. $X \xrightarrow{*} w$, where w is in T^*.
b. $S \xrightarrow{*} \alpha XB \xrightarrow{*} w$, in $L(G)$.

A grammar symbol X in CFG is *useless* if:

a. it does not derive a string of terminals, and
b. it does not occur in a derivation sequance of any w in $L(CFG)$.

Formally, in a CFG, $G = (V, T, P, S)$, X is *useless*, if it does not satisfy either of the following conditions:

a. $X \xrightarrow{*} w$, where w is in T^* or
b. $S \xrightarrow{*} \alpha XB \xrightarrow{*} w$, in $L(G)$.

Thus to find the useless and useful symbols in a grammar, we need to:

a. identify the non-terminals which are derived from terminal strings, and
b. identify variables, which are reachable from start state.

EXAMPLE 10.1.1: Identify the useful and useless symbols in $G = (V, T, P, S)$, given: $V = \{S, A, B\}$, $T = \{a\}$ and, P is

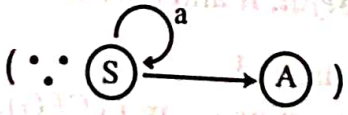
$$S \to a|AB \text{ and } A \to a.$$

Solution:

- To find the variables which are derived from the terminal string: S and A are derived from the terminal strings,

$$\text{i.e., } S \to a \quad \text{and} \quad A \to a.$$

  B is not derived from the terminal string.
- To find the variables, which are reachable in derivation, from the start state: A is not reachable in derivation from start state.



Thus, S is useful and A, B are useless symbols in G.

$$\Rightarrow G = (\{s\}, \{a\}, S \to a, S)$$

EXAMPLE 10.1.2: Identify the useful and useless symbols in $G = (V, T, P, S)$, given: $V = \{S, A, B, C\}$, $T = \{a, b\}$ and P is

$$S \to aS|A|C$$

$$A \to a$$

$$B \to aa$$

$$C \to aCb.$$

Solution:

- To find the variables which are derived from the terminal string: S, A and B can be derived from the terminal string,

$$\text{i.e.}, S \to aS|A,\ A \to a,\ B \to aa.$$

- To find the variables, which are reachable in derivation from the start-state: B is not reachable in derivation from the start state.
  Thus, S, A are useful and B, C are useless symbols in G

$$\Rightarrow G = (\{S, A\},\ \{a\},\ \{S \to as|A,\ A \to a\}, S).$$

EXAMPLE 10.1.3: Identify the useful and useless symbols in $G = (V, T, P, S)$ given: $V = \{S, A, B, C\}$, $T = \{a, b\}$ and P is

$$S \to AB|CA$$

$$B \to BC|AB$$

$$A \to a$$

$$C \to aB|b.$$

Solution:

- To find the variables which can be derived from the terminal string: S, A and C can be derived from the terminal strings,

$$\text{i.e.}, S \to CA,\ A \to a,\ C \to b.$$

- To find the variables which are reachable from the start state: S, C, A – all are reachable from the start state.

$$(\because \text{S} \longrightarrow \text{C} \longrightarrow \text{A})$$

Thus, S, C, A are useful and B is a *useless symbols* in G.

$$\Rightarrow G = (\{S, C, A\}, \{a, b\}, \{S \rightarrow CA, A \rightarrow a, C \rightarrow b\}, S)$$

10.2 Reduction of a Grammar

Reduction of a grammar refers to the identification of (i) those grammar symbols, which are useless and (ii) those productions, that do not play any role in the derivation of any w in $L(G)$.

Thus, the reduction of a given grammar G, involves:

a. identification of those grammar symbols, that are not capable of deriving a w in T^*
b. identification of those grammar symbols that are not used in any derivation
c. elimination of the above identified symbols.

10.2.1 *Reduction Algorithm*

Given a grammar $G = \{V, T, P, S\}$, obtaining the reduced form of G i.e., $G' = (V', T, P', S)$ involves the following steps:

Step-1: Set $V' = \{\phi\}$.
Step-2: Find the variables, which derive the terminal string, and add them to V'.
Step-3: Repeat the following procedure, until no more variables are added to V'. For every $A \in V$, for which P has a production of the form $A \rightarrow x_1, x_2, \ldots x_n$ with all x_i in $(V' \cup T)$, add A to V' (i.e., find the variables reachable from the start state).
Step-4: Take P' as all productions in P, all of whose symbols are in $(V' \cup T)$.

EXAMPLE 10.2.1: Find the reduced grammar equivalent to the CFG, where $V = \{A, B, C, S\}$, $T = \{a, b\}$, P is

$$S \rightarrow AB|CA$$
$$B \rightarrow BC|AB$$
$$A \rightarrow a$$
$$C \rightarrow aB|b.$$

Solution:

- Set $V' = \{\phi\}$.
- Find the variables which derive the terminal strings: S, A and C derive the terminal strings $\Rightarrow V' = \{S, A, C\}$.
- Find the variables which are reachable from the start state. S, C, A – all are reachable from the start state.

$$\Rightarrow V' = \{S, A, C\}$$

- $P' = \{S \rightarrow CA, A \rightarrow a, C \rightarrow b\}$

Thus, $G' = (\{S, A, C\}, T, \{S \rightarrow CA, A \rightarrow a, C \rightarrow b\}, S)$.

EXAMPLE 10.2.2: Find the reduced grammar, equivalent to the CFG,

$$G = (\{S, A, B, C\}, \{a, b, d\}, S, P)$$

where P is

$$S \rightarrow AC|SB$$
$$A \rightarrow bASC|a$$
$$B \rightarrow aSB|bbC$$
$$C \rightarrow BC|ad.$$

Solution:

- Set $V' = \{\phi\}$
- Find the variables which derive terminal strings: S, A, C derive terminal strings

$$\Rightarrow V' = \{S, A, C\}.$$

- Find the variables, which are reachable from the start state. S, A, C – all are reachable from the start state.

$$\Rightarrow V' = \{S, A, C\}.$$

- $P' = \{S \rightarrow AC, A \rightarrow bASC|a, C \rightarrow ad\}$.

Thus, $G' = (\{S, A, C\}, T, \{S \rightarrow AC, A \rightarrow bASC|a, C \rightarrow ad\}, S)$.

EXAMPLE 10.2.3: Find the reduced grammar equivalent to CFG. $V = \{S, B, A, X\}$, $T = \{a, b, q, d\}$ and P contains

$$S \rightarrow aB|bX$$
$$A \rightarrow BAd|bSX|q$$
$$B \rightarrow aSB|bBX$$
$$X \rightarrow SBD|aBX|ad.$$

Solution:

- Set $V' = \{\phi\}$.
- Find the variables which derive terminal strings: $A \to a, X \to ad, S \to bx \Rightarrow V' = \{S, A, X\}$.
- Find the variables, which are reachable from the start state. S and X are reachable from the start state $\Rightarrow V' = \{S, X\}$.
- $P' = \{S \to bX, X \to ad\}$.

Thus, $G' = (\{S, X\}, T, \{S \to bX, X \to ad\}, S)$.

EXAMPLE 10.2.4: Find the reduced grammar for the given grammar G, where P is

$$S \to aAa$$
$$A \to bBB$$
$$B \to ab$$
$$C \to ab.$$

Solution:

- Set $V' = \{\phi\}$.
- Find variables which derive the terminal string:

$$B \to ab, C \to ab \Rightarrow V' = \{B, C\}.$$

- Find the variables, which are reachable from start state.

$$S \to aAa \to abBBa \to abababa$$

i.e. S, A, B are reachable from start state.

$(\because$ S → A → B $)$

$$\Rightarrow V' = \{S, A, B\}.$$

- $P' = \{S \to aAa, A \to bBB, B \to ab\}$.

Thus, $G' = (\{S, A, B\}, T, \{S \to aAa, A \to bBB, B \to ab\})$.

EXAMPLE 10.2.5: Find the reduced grammar equivalent to the given grammar, whose P is

$$A \to xyz|Xyzz$$
$$X \to Xz|xYx$$
$$Y \to yYy|XZ$$
$$Z \to Zy|z.$$

Solution:

- Set $V' = \{\phi\}$.
- Find the variable which derive terminal strings:

$$A \to xyz,\ Z \to z \Rightarrow V' = \{A, Z\}.$$

- Find the variables, reachable from the start state. Z is not reachable from the start state $\Rightarrow V' = \{A\}$.
- $P' = \{A \to xyz\}$.

Thus, $G' = (\{A\}, T, A \to xyz, A)$.

EXAMPLE 10.2.6: Find the useless symbols in the following grammar and modify the grammar, so that it has no useless symbols:

$$S \to 0|A$$
$$A \to AB$$
$$B \to 1$$

Solution:

- Set $V' = \{\phi\}$.
- Find the variables which derive terminal strings:

$$S \to 0,\ B \to 1 \Rightarrow V' = \{S, B\}.$$

- Find the variables, which are reachable from the start state: B is not reachable from the start state

$$\Rightarrow V' = \{S\}.$$

- $P' = \{S \to 0\}$.

Thus, $G = (\{S\}, T, \{S \to 0\}, S)$.

10.3 Elimination of ϵ-Productions

A production of the form $A \to \epsilon$ is called an ϵ production. If A is a non-terminal, and $A \xrightarrow{*} \epsilon$ (i.e if A leads to an empty string in zero, one or more derivations), then A is called a *'Nullable non-terminal'*.

EXAMPLE 10.3.1: For the grammar:

$$S \to aS_1b$$
$$S_1 \to aS_1b| \in,$$

the $\in$-production is: $S_1 \to \in$.

EXAMPLE 10.3.2: For the grammar:

$$S \to ACB|cbB|Ba$$
$$A \to da|BC$$
$$B \to gC| \in$$
$$C \to ha| \in$$

It is clear that $B \to \in$ and $C \to \in$ are $\in$-productions.
Consider,

$$A \Rightarrow BC$$
$$\Rightarrow B \in$$
$$\Rightarrow \in$$

i.e. $A \overset{*}{\Longrightarrow} \in$, hence $A \to \in$. Thus, A is an $\in$-production.
Again,

$$S \Rightarrow ACB$$
$$\Rightarrow AC$$
$$\Rightarrow A$$
$$\Rightarrow \in .$$

i.e. $S \overset{*}{\Longrightarrow} \in$, hence $S \to \in$ is also an $\in$-production.
$\therefore$, B, C, A, S are all nullable.

Theorem I: If $G = (V, T, P, S)$ is a CFG, then there exists a CFG G', having no null productions such that $L(G') = L(G) - \{\in\}$.
Proof: This theorem can be proved (in a constructive way) as follows:

Step-1: *Construction of a set of nullable variables.*

The nullable variables can be computed, recursively, as follows:

a. $W_1 = \{A \in V' | A \rightarrow \in \text{ is in } P\}$.

b. $W_{i+1} = W_1 \cup \{A | A \rightarrow \alpha \text{ with } \alpha \in W_i^*\}$.

By the definition of W_i,

$$W_i \sqsubseteq W_{i+1} \text{ for all i.}$$

As V' is finite,

$$W_{k+1} = W_k \text{ for some } k \leqslant |V'|.$$

So $W_{k+j} = W_k$, for all j. Let $W = W_k$.

Therefore, W is the set of all nullable variables.

Step-2: *Construction of P'*

a. Any production, whose R.H.S. has any nullable variables should be included in P.

b. If $A \rightarrow X_1, X_2, \ldots X_k$ is in P, then the productions of the form $A \rightarrow \alpha_1\alpha_2\alpha_3 \ldots \alpha_n$ are included in P', where $\alpha = X_i$, if $X_i \notin W$ (i.e., if X_i is not nullable) and $\alpha_i = X_i$ or $\in$ (if $X \in W$ and $\alpha_1, \alpha_2, \ldots \alpha_k \neq \in$).

Since the nullable variables are eliminated, it is clear that G' does not contain any null productions. Hence proved.

10.3.1 Method to eliminate ∈-production

Step-1: *Identify the null productions and nullable variables.* Compute the nullable variable by using recursive definition as:

a. $W_1 = \{A \in V' | A \rightarrow \in \text{ is in } P\}$.

b. $W_{i+1} = W_1 \cup \{A | A \rightarrow \alpha, \text{ with } \alpha \in W_i^*\}$.

Step-2: *Construction of P'*

a. Any production whose R.H.S. does not have any nullable variables is included in P'.

b. If there is a production say $A \to \in$, then to eliminate $\in$-production, look for all those productions in the grammar whose right-sides contain A and replace each occurrence of A in these productions. Thus, obtain the non-$\in$-productions to be added to the grammar, so that language's generation remains the same.

EXAMPLE 10.3.3: Construct a grammar G', without null-productions for the given grammar G, whose productions are:

$$S \to aS|AB$$
$$A \to \in$$
$$B \to \in$$
$$D \to b.$$

Solution:

Step-1: *Construction of nullable set*

$$W_1 = \{A|A \to \in \text{ is in } P\}$$
$$W_1 = \{A, B\} \quad (\because A \to \in, B \to \in)$$
$$W_{i+1} = W_1 \cup \{A|A \to \alpha \quad \text{with} \quad \alpha \in W_i^*\}$$
$$W_2 = \{A, B\} \cup \{S\} \quad \{\because S \Rightarrow AB \Rightarrow \in\}$$
$$W_2 = \{S, A, B\}$$
$$W_3 = \{S, A, B\} \cup \phi \Rightarrow W_3 = \{S, A, B\}.$$

Thus, $W = \{S, A, B\}$, is the set of all nullable variables.

Step-2: *Construction of P'*

a. R.H.S with no nullable variables, is $D \to b$.

b. $S \to aS$ generates $S \to aS$ and $S \to a \quad (\because S \to \in)$.
$S \to AB$ generates $S \to AB$, $S \to A$ and $S \to B \quad (\because A \to \in, B \to \in)$.

Thus, $P' = \{S \to aS|a|AB|A|B, D \to b\}$. Hence, G' is without $\in$-production.

EXAMPLE 10.3.4: Find a CFG, without $\in$-productions, equivalent to the following grammar defined by:

$$S \to ABaC,\ A \to BC,\ B \to b|\in,\ C \to D|\in \text{ and } D \to d.$$

Solution:

Step-1: *Construction of nullable set*

$$W_1 = \{A | A \to \in \text{ is in } P\}$$

$$W_1 = \{B, C\} \quad (\because B \to \in, C \to \in).$$

$$W_{i+1} = W_1 \cup \{A | A \to \alpha \text{ with } \alpha \in W_i^*\}.$$

$$W_2 = \{B, C\} \cup \{A\} \qquad (\because A \Rightarrow BC \Rightarrow \in)$$

$$W_2 = \{A, B, C\}.$$

Similarly, $W_3 = \{A, B, C\} \cup \phi \Rightarrow W_3 = \{A, B, C\}$.

Thus, $W = \{A, B, C\}$.

Step-2: *Construction of* P'.

a. R.H.S. with no nullable variables are $B \to b, C \to D, D \to d$.

b. $S \to ABaC$ generates

$$S \to ABaC, \quad S \to ABa, S \to AaC, S \to BaC$$

$$S \to aC, S \to Ba, S \to Aa.$$

$A \to BC$ generates $A \to BC, A \to B, A \to C$

Thus,

$$P' = \{S \to ABaC | ABa | AaC | BaC | aC | Ba | Aa,$$

$$A \to BC | B | C$$

$$B \to b$$

$$C \to D,$$

$$D \to d\}.$$

Thus, G' is without $\in$-productions.

EXAMPLE 10.3.5: Consider the following grammar and eliminate all $\in$-productions, without changing the language generated by the grammar.

$$S \to AaA$$

$$A \to Sb | bCC | \in$$

$$C \to CC | abb$$

Solution:

Step-1: *Construction of nullable set.*

$$W_1 = \{A | A \to \in \text{ is in } P\}.$$
$$W_1 = \{A\} \quad (\because A \to \in).$$
$$W_{i+1} = W_i \cup \{A | A \to \alpha \text{ with } \alpha \in W_i^*\}.$$
$$W_2 = \{A\} \cup \phi \Rightarrow W_2 = \{A\}.$$

Thus, $W = \{A\}$.

Step-2: *Construction of P'.*

a. R.H.S. with no nullable variables is, $C \to CC|abb$.

b. $S \to AaA$ generates

$$S \to AaA, S \to Aa, S \to aA, S \to a.$$

Thus,

$$P' = \{S \to AaA|Aa|aA|a$$
$$A \to Sb|bCC$$
$$C \to CC|abb.\}.$$

Thus, G' is without $\in$-productions.

10.4 Elimination of Unit Productions

The following are the equivalent definitions for the concept of unit production:

- Any production of the form $A \to B$, whose R.H.S. consists of a single variable, is called a *unit production*. All other productions including $A \to a$, $\in$ are non-unit productions.
- A production of the form $A \to B$, where A and B are both non-terminals, is called a unit production.

Note–1:
Presence of unit production in a grammar increases the cost of derivations.

EXAMPLE 10.4.1: Consider a grammar

$$S \to AB$$
$$A \to a$$
$$B \to C|b$$
$$C \to D$$
$$D \to E$$
$$E \to a.$$

Here,

- unit productions $-B \to C$
  $C \to D$
  $D \to E$,
- non-unit productions $-S \to AB$
  $A \to a$
  $E \to a$.

10.4.1 *Method to eliminate unit productions*

Step-1: *Identify the unit productions in the grammar.*

Step-2: *Construction of P'*

a. Identify all non-unit productions and add to P'.

b. If there exists a unit production $A \to B$ in the grammar, then apply the following steps until there are no unit productions left.

- Select a unit production $A \to B$, such that, there exists atleast one non-unit production
  $$B \to \alpha$$
- Now, for every non-unit production $B \to \alpha$, add production $A \to \alpha$ to the grammar and eliminate $A \to B$ from the grammar.

EXAMPLE 10.4.2: Eliminate all unit productions from the grammar given below:

$$S \to Aa|B$$
$$B \to A|bb$$
$$A \to a|bc|B$$

Solution:

Step-1: *Identify all unit productions*

a. $S \rightarrow B$.

b. $B \rightarrow A$.

c. $A \rightarrow B$.

Step-2: *Construction of P'*

a. To eliminate $S \rightarrow B$
Substitute the alternative of B to $S \rightarrow B$, i.e.,

$$S \rightarrow Aa|B \Rightarrow S \rightarrow Aa|A|bb.$$

But $S \rightarrow A$ is a unit production.
Substitute alternative of A to $S \rightarrow Aa|A|bb$

$$S \rightarrow Aa|A|bb \Rightarrow S \rightarrow Aa|a|bc|bb$$

b. To eliminate $B \rightarrow A$
Substitute the alternative of A to $B \rightarrow A$, so that

$$B \rightarrow A|bb \Rightarrow B \rightarrow a|bc|bb.$$

c. To eliminate $A \rightarrow B$
Substitute alternative of B to $A \rightarrow B$, so that

$$A \rightarrow a|bc|B \Rightarrow A \rightarrow a|bc|bb.$$

Thus, P' is

$$S \rightarrow Aa|a|bc|bb$$

$$B \rightarrow a|bc|bb$$

$$A \rightarrow a|bc|bb.$$

EXAMPLE 10.4.3: Given the grammar below, eliminate all the unit productions.

$$S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow C|b$$

$$C \rightarrow D$$

$$D \rightarrow E$$

$$E \rightarrow a$$

Solution:

Step-1: *Identify all unit productions*

a. $B \rightarrow C$

b. $C \rightarrow D$

c. $D \rightarrow E$

Step-2: *Construction of P'*

$P' = \{S \rightarrow AB, A \rightarrow a, E \rightarrow a\}$

a. To eliminate $B \rightarrow C$

Substitute alternative of C to $B \rightarrow C$, so that

$$B \rightarrow C|b \Rightarrow B \rightarrow D|b.$$

Since D is a unit production, substitute alternative of D to $B \rightarrow D|b$.

$$B \rightarrow D|b \Rightarrow B \rightarrow E|b.$$

Since $\in$ is unit production, substitute alternative of E to $B \rightarrow E|b$.

$$B \rightarrow E|b \Rightarrow \boxed{B \rightarrow a|b}$$

b. To eliminate $C \rightarrow D$

Substitute alternative of D to $C \rightarrow D$, so that

$$C \rightarrow D \Rightarrow C \rightarrow E$$

Since E is a unit production, substitute alternative of E to $C \rightarrow E$.

$$C \rightarrow E \Rightarrow \boxed{C \rightarrow a}$$

c. To eliminate $D \rightarrow E$

Substitute alternative of E to $D \rightarrow E$, so that

$$D \rightarrow E \Rightarrow \boxed{D \rightarrow a}.$$

Thus, the modified P:

$$S \rightarrow AB$$

$$A \rightarrow a$$

$$B \rightarrow a|b$$

$$C \rightarrow a, D \rightarrow a \text{ and } E \rightarrow a.$$

EXAMPLE 10.4.4: Eliminate all unit productions from the following grammar:

$$S \to a|aA|B|C$$
$$A \to aB| \in$$
$$B \to Aa$$
$$C \to cCD$$
$$D \to ddd.$$

Solution: The given grammar is with $\in$-productions. After the elimination of $\in$ from the above grammar, P':

$$S \to a|aA|B|c|a$$
$$A \to aB$$
$$B \to Aa|a$$
$$C \to cCD$$
$$D \to ddd.$$

Step-1: *Identify all unit productions*

a. $S \to B$
b. $S \to C$.

Step-2: *Construct* P''

$$P'' = \{A \to aB, B \to Aa|a, C \to cCD, D \to ddd\}$$

a. To eliminate $S \to B$
Substitute alternative of B to $S \to B$, so that

$$S \to a|aA|B|C|a \Rightarrow S \to a|aA|Aa|a|C|a.$$

b. To eliminate $S \to C$
Substitute alternative of C to $S \to C$, so that $S \to a|aA|Aa|a|C|a$
$\Rightarrow S \to a|aA|Aa|a|cCD|a.$

Finally, P'' is: $S \to a|aA|Aa|a|cCD|a$
$A \to aB$
$B \to Aa|a$
$C \to cCD$
$D \to ddd.$

10.5 Normal Forms for Context-Free Grammars

When the productions in CFG are made to satisfy certain restrictions, then CFG is said to be in *normal form*.

The following are some types of normal forms.

a. *Chomsky normal form*
b. *Greibach normal form*
c. *Kuroda normal Form*

10.6 Chomsky Normal Form

Chomsky normal form (CNF) puts restrictions on *number of symbols* on the right of a production. In other words the strings, on the right of a production, consist of *not more than two symbols*.

Formally, a CFG is said to be in chomsky normal form, if all productions are of the form

$$A \to BC$$

$$\text{or}, \quad A \to a,$$

where A, B, C are non-terminals and a is a terminal.

EXAMPLE 10.6.1: The grammar

$$S \to BS|a$$

$$B \to SB|b$$

is in CNF.

EXAMPLE 10.6.2: The grammar

$$S \to BS|BSB$$

$$B \to SB|aa$$

is not in CNF, because both the productions $S \to BSB$ and $B \to aa$ violate the conditions of CNF.

EXAMPLE 10.6.3: The grammar

$$S \to AB|CA$$
$$B \to BC|AB$$
$$A \to a$$
$$C \to aB|b$$

is not in CNF, because $C \to aB$ violates the conditions.

10.6.1 *Reductions of CFG to CNF*

Theorem II: For every CFL, without $\in$ generated by a grammar G_1, there is an equivalent grammar G_2 in CNF.

Proof: Let $G_1 = (V, T, P, S)$ be the CFG generating a language not containing $\in$. We construct $G_2 = (V'', T, P'', S)$ as follows:

Step-1: *Eliminate $\in$-productions and unit productions*

Eliminate all $\in$ and unit productions from G_1, by using an appropriate method. If a production contains a single terminal symbol at right, it is in acceptable form. Add all such productions to P' and all variables to V'.

Step-2: *Eliminate terminals on R.H.S.*

All productions in P of the form $A \to BC$ or $A \to a$ are included in P' and all variables of such productions are included in V'. Consider a production of the form $A \to x_1x_2 \ldots x_n$, $n \geq 2$. If x_i is a terminal 'a', introduce a new variable 'T_a' and replace the terminal x_i by T_a. These new productions are added to P' and new variables are added to V'.

Step-3: *Restrict the number of variables on RHS*

All productions, that are already in acceptable form of P', are added to P'' and all respective variables are included in V''. Consider the production $A \to A_1A_2 \ldots A_n$, $n \geq 3$. Now, introduce new productions $A \to A_1K_1$. $K_1 \to A_2K_2 \ldots K_{n-2} \to A_{m-1}A_m$ to P'' and respective new variables $K_1, K_2, \ldots K_{n-2}$ are added to V''.

Thus, we get G_2 in CNF.

Hence proved.

EXAMPLE 10.6.4: Reduce the grammar into CNF.

$$S \to aAD$$
$$A \to aB|bAB$$
$$B \to b$$
$$D \to d.$$

Solution:

Step-1: *Eliminate null and unit productions.* There are no such productions.

Step-2: $P' = \{B \to b,\ D \to d\}$ is in CNF.
Reduce the number of terminals in

$$S \to aAD$$
$$A \to aB|bAB.$$

Thus,

$$S \to aAD \Rightarrow S \to T_aAD, \quad \text{where} \quad T_a \to a$$
$$A \to aB \Rightarrow A \to T_aB, \quad \text{where} \quad T_a \to a$$
$$A \to bAB \Rightarrow A \to T_bAB, \quad \text{where} \quad T_b \to b.$$

Step-3: *Restrict number of variables on the R.H.S. (for those productions which are not in CNF).*
Thus,

$$S \to T_aAD \Rightarrow S \to T_aK_1, \quad \text{where} \quad K_1 \to AD$$
$$A \to T_aB \Rightarrow A \to T_aB, \quad \text{already in CNF}$$
$$A \to T_bAB \Rightarrow A \to T_bK_2, \quad \text{where} \quad K_2 \to AB.$$

Thus,

$$P'' = \{S \to T_aK_1$$
$$A \to T_aB$$
$$A \to T_bK_2, \quad K_1 \to AD, \quad K_2 \to AB$$
$$B \to b$$
$$D \to d, \quad T_a \to a, \quad T_b \to b\}.$$

$$V'' = \{S, A, B, D, K_1, K_2, T_a, T_b\}.$$

EXAMPLE 10.6.5: For the grammar given below, find an equivalent grammar in CNF.

$$S \rightarrow bA|aB$$
$$A \rightarrow bAA|aS|a$$
$$B \rightarrow aBB|bS|b$$

Solution:

Step-1: *There are no null and unit productions.*

Step-2: $P' = \{A \rightarrow a, B \rightarrow b\}$ is in CNF.
Reduce the number of terminals in

$$S \rightarrow bA|aB$$
$$A \rightarrow bAA|aS$$
$$B \rightarrow aBB|bS.$$

Thus,

$$S \rightarrow bA \Rightarrow S \rightarrow T_bA, \quad \text{where } T_b \rightarrow b.$$
$$S \rightarrow aB \Rightarrow S \rightarrow T_aB, \quad \text{where } T_a \rightarrow a$$
$$A \rightarrow bAA \Rightarrow A \rightarrow T_bAA, \quad \text{where } T_b \rightarrow b$$
$$A \rightarrow aS \Rightarrow A \rightarrow T_aS, \quad \text{where } T_a \rightarrow a.$$
$$B \rightarrow aBB \Rightarrow B \rightarrow T_aBB, \quad \text{where } T_a \rightarrow a$$
$$B \rightarrow bS \Rightarrow B \rightarrow T_bS, \quad \text{where } T_b \rightarrow b.$$

Step-3: $P' = \{A \rightarrow a, B \rightarrow b, S \rightarrow T_bA, S \rightarrow T_aB, A \rightarrow T_aS, B \rightarrow T_bS\}$ are now in CNF.
Restrict the number of variables for production:

$$A \rightarrow T_bAA \qquad B \rightarrow T_aBB.$$

So that,

$$A \rightarrow T_bAA \Rightarrow A \rightarrow T_bK_1, \text{where} K_1 \rightarrow AA.$$
$$B \rightarrow T_aBB \Rightarrow B \rightarrow T_aK_2, \text{where} K_2 \rightarrow BB.$$

Thus,

$$V'' = \{S, A, B, T_a, T_b, K_1, K_2\}.$$

$$\begin{aligned} P'' = & S \to T_bA|T_aB \\ & A \to T_bK_1|T_aS|a \\ & B \to T_aK_2|T_bS|b \\ & K_1 \to AA \\ & K_2 \to BB, T_a \to a \text{ and } T_b \to b. \end{aligned}$$

EXAMPLE 10.6.6: Convert the given grammar to CNF.

$$\begin{aligned} S &\to AB|CA \\ B &\to BC|AB \\ A &\to a \\ C &\to aB|b \end{aligned}$$

Solution:

Step-1: *Eliminate* $\in$ *and unit productions.* There are no such productions.

Step-2: $P' = \{S \to AB|CA, B \to BC|AB, C \to b, A \to a\}$ are in CNF. *Reduce the number of terminals in* $C \to aB$.

$$C \to aB \Rightarrow C \to T_aB \text{ where } T_a \to a.$$

Thus,

$$\begin{aligned} P'' = & S \to AB|CA \\ & B \to BC|AB \\ & C \to T_aB|b \\ & A \to a \\ & T_a \to b \end{aligned}$$

$$V'' = \{S, B, C, A, T_a\}.$$

EXAMPLE 10.6.7: Write the equivalent CNF, for grammar:

$$\begin{aligned} S &\to AB \\ A &\to aab \\ B &\to aAC. \end{aligned}$$

Solution:

Step-1: *There are no unit and null productions.*

Step-2: $P' = \{S \to AB\}$ is in CNF.
Reduce the number of terminals for $A \to aab, B \to aAC$.

$$A \to aab \Rightarrow A \to T_aT_aT_b \text{ where } \begin{matrix} T_a \to a \\ T_b \to b \end{matrix}.$$

$$B \to aAC \Rightarrow B \to T_aAC.$$

Step-3: *Restrict number of variables on the R.H.S. for* $\begin{matrix} A \to T_aT_aT_b \\ B \to T_aA_c \end{matrix}$ and

$$A \to T_aT_aT_b \Rightarrow A \to T_aK_1 \quad \text{where} \quad k_1 \to T_aT_b$$
$$B \to T_aAC \Rightarrow B \to T_aK_2 \quad \text{where} \quad k_2 \to AC.$$

Thus,

$$\begin{aligned} P'' = \{&S \to AB \\ &A \to T_aK_1 \\ &B \to T_aK_2 \\ &K_1 \to T_aT_b \\ &k_2 \to AC \\ &T_a \to a \\ &T_b \to b\} \end{aligned}$$
$$V'' = \{S, A, B, K_1, K_2, T_a, T_b\}.$$

EXAMPLE 10.6.8: Reduce the grammar, into CNF.

$$S \to A0B$$
$$A \to AA|0S|0$$
$$B \to 0BB|1S|1$$

Solution:

Step-1: *There are no unit and null productions.*

Step-2: $P' = \{A \to AA, A \to 0, B \to 1\}$. is in CNF.

Reduce the number of terminals for $S \to A0B$

$$A \to 0S$$

$$B \to 0BB|1S$$

$$S \to A0B \Rightarrow S \to AT_0B \quad \text{where} \quad T_0 \to 0$$

$$A \to 0S \Rightarrow A \to T_0S \quad \text{where} \quad T_0 \to 0$$

$$B \to 0BB \Rightarrow B \to T_0BB \quad \text{where} \quad T_0 \to 0$$

$$B \to 1S \Rightarrow B \to T_1S \quad \text{where} \quad T_1 \to 1.$$

Step-3: $P' = \{A \to AA, A \to 0, A \to T_0S, B \to T_1S, B \to 1\}$ is in CNF

Restrict the number of variables on the R.H.S. for $S \to AT_0B$ and $B \to T_0BB$.

$$S \to AT_0B \Rightarrow S \to AK_1 \quad \text{where} \quad k_1 \to T_0B$$

$$B \to T_0BB \Rightarrow B \to T_0K_2 \quad \text{where} \quad k_2 \to BB.$$

Thus,

$$P'' = \{S \to AK_1$$

$$A \to AA|T_0S|0$$

$$B \to T_0K_2|T_1S|1$$

$$K_1 \to T_0B$$

$$k_2 \to BB$$

$$T_0 \to 0$$

$$T_1 \to 1\}$$

$$V'' = \{S, A, B, K_1, K_2, T_0, T_1\}.$$

10.7 Greibach Normal Form

Greibach normal form (GNF) puts restrictions not on the length of R.H.S. of a production, but on the positions in which terminals and variables appear. Arguments justifying GNF are little complicated and not very transparent.

Formally, a CFG is said to be in Greibach Normal Form, if all productions are of the form

$$A \rightarrow ax,$$

where, a is a terminal i.e, $a \in T$ and x is a string of variables (possibly empty), $x \in V^*$.

EXAMPLE 10.7.1: The grammar

$$S \rightarrow AB$$
$$A \rightarrow aA|bB|b$$
$$B \rightarrow b$$

is not in GNF because $S \rightarrow AB$ is not in the form $A \rightarrow ax$.

EXAMPLE 10.7.2: The grammar

$$S \rightarrow aAB|bBB|bB$$
$$A \rightarrow aA|bB|b$$
$$B \rightarrow b$$

is in GNF.

EXAMPLE 10.7.3: The grammar

$$S \rightarrow aSB|aB$$
$$B \rightarrow b$$

is in GNF.

Sheila Greibach was born in New York City in 1939. She attended Radcliffe College, where she received her BA in 1960. In 1963, she received her Ph.D. from Harvard University.

Greibach is well known for her work on formal languages. One of her contributions is Greibach normal form, a normal form for grammars in which every production is of the form $A \rightarrow aB$ or $A \rightarrow a$.

Theorem III: For every CFG, G with $\in \notin L(G)$, there exists an equivalent grammar G' in Greibach Normal Form.

Proof and algorithm:

Outline of the proof:

Firstly, we convert the given grammar into Chomsky NF:

- Start with a grammar G = (V, T, P, S).
- Eliminate useless variables that cannot become terminals.
- Eliminate useless variables that cannot be reached.
- Eliminate $\in$-productions.
- Eliminate unit productions.
- Convert grammar to Chomsky Normal Form.

Then, we convert this grammar into an equivalent grammar in Greibach NF. The core of this procedure to construct a grammar in GNF is the following:
We sort our productions, so that we get a sequential line of dependencies for those variables, which come first on the R.H.S. of the productions. Every derivation will use variables in accordance with this sequential ordering. The last variable in the sequence of productions or variables has only terminal rules. Thus, every derivation will end (latest) with this variable. We use this variable to start a step-by-step substitution of the first R.H.S variable in the other rules, so that we successively get all rules, starting with a terminal.
This way, we transform the grammar, starting with a CNF, into GNF.

Steps to Convert a CNF grammar into Greibach Normal Form:

a. Relabel all variables such that the names are $A_1, A_2, \ldots, A_m$.
b. We want to order the productions, which are not terminal but contain variables. For this purpose, the indexing of the variables is used, so that

$$A_i \to A_j\alpha \qquad \text{with } i < j, \quad \text{for all } i = 1, \ldots, m \text{ and } j = 2, \ldots, m.$$

We perform the ordering process by the substitution of the first variable on the R.H.S., if the production violates the condition above (see c and d). During this process we also eliminate left-recursive rules, i.e., rules of the form

$$A_i \to A_i\alpha$$

as soon as we encounter them in the ordering process (see *e*).

The outcome of this phase is that all rules are sorted and have either a higher-numbered variable as first symbol on the R.H.S., or a single terminal.

c. We start the ordering with A_1.

A_1 - productions can have only a higher-numbered variable as first variable on the R.H.S., or a single terminal. (For left-recursive rules $A_i \rightarrow A_i \ldots$ see e).

d. We, now, assume that all rules are okay up to A_{k-1}. The next rule we encounter, with A_k on the LHS, is the first one, which is not okay:

$$A_k \rightarrow A_l\alpha \qquad \text{with } k > l.$$

We resolve this problem by substituting A_l. Since $l < k$, the A_l-rules have already gone through the sorting process and are in the proper format:

$$A_l \rightarrow A_j\alpha \qquad \text{with } l < j.$$

Now, we substitute the RHSs of A_l in the A_k - rule, and come up with:

$$A_k \rightarrow A_l\alpha \qquad \text{or} \quad A_k \rightarrow a \qquad \text{for some a.}$$

If l is still less than k, we substitute again. This process is repeated until we get *atleast* A_k on the R.H.S. All rules up to A_{k-1} are already sorted and in proper form, and the first variable on the R.H.S. of the A_{k-1} - production must, therefore, be atleast A_k.

e. When we encounter a left-recursive rule during this process, we resolve this left-recursion immediately.
Let's assume that we have a set of left-recursive rules for A_k

$$1.\ A_k \rightarrow A_k\alpha_1 | A_k\alpha_2 | \ldots | A_k\alpha_r$$

and a set of not left-recursive rules:

$$2.\ A_k \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n.$$

We convert the left-recursion over A_k into a right-recursion, using a new variable B_k and new productions:

$$3.\ B_k \rightarrow \alpha_1 | \alpha_1 B_k | \alpha_2 | \alpha_2 B_k | \ldots | \alpha_r | \alpha_r B_k.$$

We integrate this with the non left-recursive rules for A_k:

$$4.\ A_k \rightarrow \beta_1|\beta_1 B_k|\beta_2|\beta_2 B_k|\ldots|\beta_n|\beta n B_k.$$

The rules 3 and 4 replace the original rules 1 and 2.

When you look at rule 4, you can see that it generates the original R.H.Ss β_i from the non left-recursive rules for A_k. It also includes the recursion through the variable B_k but starting 'at the end', i.e. using the string β_i first, and then the recursive variable B_k. B_k can now be substituted with just one of the α_i, or the recursion can generate more of those α_i. Thus, the new productions 3 and 4 are generating the same strings as the old productions 1 and 2.

f. When we are done with this, we find that all the rules are in the proper format (according to b). They either start with a higher numbered variable, or a terminal (from the original rules, directly or through substitution).

In particular, the rules for the highest-numbered variable have to start with a terminal (since no variable has a higher index than this one).

g. Now, we use this ordering and again perform a set of substitutions, to ensure that all productions start with a single terminal.

We start these substitutions from 'the end'. We know that rules for A_m must start with a single terminal. All other rules might start with a terminal, possibly followed by a higher-numbered variable, and then the other variables.

We start the substitutions with the rules for the second highest variable A_{m-1}. If the R.H.S. of an A_{m-1}-production starts with a variable, it must be A_m. We substitute this A_m with its R.H.S. and get a new rule for A_{m-1}, which starts with a single terminal. We proceed backwards doing this, through all A_i down to A_1. Now all A_i rules start with a single terminal symbol, followed by nothing or variables only.

h. In the last step, we bring the B-rules (introduced for the removal of left-recursion) into proper format. If the R.H.S. of a B-rule starts with a variable, we just need to replace this variable with its R.H.S. (which starts with a terminal). Then, B_i rules also conform with GNF.

i. The conversion of CNF grammar into GNF is complete.

EXAMPLE 10.7.4: Bring the grammar G with $V = \{S, A, B\}, T = \{a, b\}$ and productions p

$$S \to A$$
$$A \to aBa|a$$
$$B \to bAb|b$$

into GNF.

Solution:

Step-1: *Simplify G*

There are no useless variables or productions and no $\in$-productions. Remove unit-production $S \to A$. Replace A with RHS of A (after calculating transitive closure of unit-productions - but there is only one unit-dependency here, i.e., $A \Rightarrow B$).

$$S \to aBa|a \text{ (new rule)}$$

Step-2: *Transform G into an equivalent grammer G′ in Chomsky NF*

a. Substitute terminals on RHS with variables:

$$S \to C_1BC_1|a \quad C_1 \to a$$
$$A \to C_1BC_1|a \quad C_2 \to b$$
$$B \to C_2AC_2|b.$$

b. Break down the rules:

$$S \to C_1D_1|a \quad D_1 \to BC_1 \quad C_1 \to a$$
$$A \to C_1D_1|a$$
$$B \to C_2D_2|b \quad D_2 \to AC_2 \quad C_2 \to b.$$

Step-3: *Transform G′ into equivalent grammar G′ in Greibach NF*

Rename the variables to $V_1, V_2, \ldots$ in the productions p'.

$$S = V_1; A = V_2; B = V_3; C_1 = V_4; C_2 = V_5; D_1 = V_6; D_2 = V_7$$
$$V_1 \to V_4V_6|a \quad V_6 \to V_3V_4 \quad V_4 \to a$$
$$V_2 \to V_4V_6|a$$
$$V_3 \to V_5V_7|b \quad V_7 \to V_2V_5 \quad V_5 \to b$$

Step-4: *Order the productions and remove recursion where necessary*

$$V_1 \rightarrow V_4V_6|a$$
$$V_2 \rightarrow V_4V_6|a$$
$$V_3 \rightarrow V_5V_7|b$$
$$V_4 \rightarrow a$$
$$V_5 \rightarrow b$$
$$V_6 \rightarrow V_3V_4$$
$$V_7 \rightarrow V_2V_5$$

Rules for V_1, V_2, V_3, V_4, V_5 are fine, with respect to ordering.

a. Modify V_6 – rule$V_6 \rightarrow V_3V_4$
   - Substitute R.H.Ss of V_3 in V_6-rule:

$$V_6 \rightarrow V_5V_7V_4|bV_4$$

   - Substitute R.H.Ss of V_5 in the modified V_6-rule:

$$\mathbf{V_6 \rightarrow b\ V_7V_4|b\ V_4} \quad \textit{final } V_6 - \textit{rule}$$

b. Modify V_7-rule $V_7 \rightarrow V_2V_5$
   - Substitute R.H.Ss of V_2 in V_7-rule:

$$V_7 \rightarrow V_4V_6V_5|aV_5$$

   - Substitute R.H.Ss of V_4 in the modified V_7-rule:

$$\mathbf{V_7 \rightarrow a\ V_6V_5|a\ V_5} \quad \textit{final } V_7 - \textit{rule}$$

Now all the rules are sorted properly, according to the ordering constraint: if $V_1 \rightarrow V_j \ldots$ then $i < j$.

The new productions are:

$$V_1 \rightarrow V_4V_6|a$$
$$V_2 \rightarrow V_4V_6|a$$
$$V_3 \rightarrow V_5V_7|b$$
$$V_4 \rightarrow a$$
$$V_5 \rightarrow b$$
$$V_6 \rightarrow bV_7V_4|bV_4$$
$$V_7 \rightarrow aV_6V_5|aV_5$$

Step-5: *Substitute to achieve Greibach Normal Form*
We have to substitute backwards, all leading variables on the R.H.S., (i.e. in all those rules, which still start with a variable).
These are the V_3-, V_2-, and V_1-rules:

$$V_3 \to V_5V_7|b \qquad \text{out!}$$
$$\mathbf{V_3 \to b\,V_7|b} \qquad \textit{new rule}$$
$$V_2 \to V_4V_6|a \qquad \text{out!}$$
$$\mathbf{V_2 \to a\,V_6|a} \qquad \textit{new rule}$$
$$V_1 \to V_4V_6|a \qquad \text{out!}$$
$$\mathbf{V_1 \to a\,V_6|a} \qquad \textit{new rule}$$
$$V_4 \to a$$
$$V_5 \to b$$
$$V_6 \to b\,V_7V_4|b\,V_4$$
$$V_7 \to a\,V_6V_5|a\,V_5$$

The grammar is now in GNF:

$$V_1 \to aV_6|a$$
$$V_2 \to aV_6|a$$
$$V_3 \to bV_7|b$$
$$V_4 \to a$$
$$V_5 \to b$$
$$V_6 \to bV_7V_4|bV_4$$
$$V_7 \to aV_6V_5|aV_5.$$

EXAMPLE 10.7.5: Bring the grammar G with $V = \{S, A, B\}, T = \{a, b\}$ and productions P

$$S \to AB$$
$$A \to BSB$$
$$A \to a$$
$$B \to b$$

into Greibach NF.

Solution:

Step-1: *Simplify G*

There are no useless variables or productions, no $\in$-productions, no unit-production. Nothing to do here.

Step-2: *Transform G into an equivalent grammar G′ in Chomsky NF*

$$S \to AB$$

$$A \to BSB$$

$$A \to a$$

$$B \to b$$

a. There is no terminal on R.H.S., together with other variables or terminals. Nothing to do here.

b. Break down the rules.
Modify the A-rule as:

$A \to BSB$ (introduce variable D_1 and "break down" the rule)

$\mathbf{A \to B\ D_1}$ *2 new rules*

$\mathbf{D_1 \to S\ B}$

c. Grammar G' is in Chomsky NF:

$$S \to A\,B$$

$$A \to B\,D_1$$

$$D_1 \to S\,B$$

$$A \to a$$

$$B \to b$$

Step-3: *Transform G′ into an equivalent grammar G′ in Greibach NF*
Rename the variables in the productions P' to $V_1, V_2 \ldots$:

$$S = V_1; A = V_2; B = V_3; D_1 = V_4$$

$$V_1 \to V_2V_3$$

$$V_2 \to V_3V_4 | a$$

$$V_3 \to b$$

$$V_4 \to V_1V_3$$

Step-4: *Order the productions and remove recursion where necessary*

$V_1 \rightarrow V_2V_3$ okay.

$V_2 \rightarrow V_3V_4|a$ okay.

$V_3 \rightarrow b$ okay.

$V_4 \rightarrow V_1V_3$ needs to be modified.

a. Modify V_4-rule

- Substitute R.H.S. of V_1 in V_4-rule:

$$V_4 \rightarrow V_2V_3V_3$$

- Substitute R.H.S.s of V_2 in modified V_4-rule:

$$V_4 \rightarrow V_3V_4V_3V_3|aV_3V_3$$

- Substitute R.H.S. of V_3 in modified V_4-rule above:

$$V_4 \rightarrow bV_4V_3V_3|aV_3V_3 \quad \text{new } V_4 - \text{rule}$$

All the rules are now ordered properly:

$$V_1 \rightarrow V_2V_3$$
$$V_2 \rightarrow V_3V_4|a$$
$$V_3 \rightarrow b$$
$$V_4 \rightarrow bV_4V_3V_3|aV_3V_3$$

Step-5: *Substitute backwards to achieve Greibach Normal Form*

$V_4 \rightarrow bV_4V_3V_3|aV_3V_3$ okay.

$V_3 \rightarrow b$ okay.

$V_2 \rightarrow V_3V_4|a$ needs to be modified, substitute V_3

$V_2 \rightarrow bV_4|a$ *new V_2-rule*

$V_1 \rightarrow V_2V_3$ needs to be modified, substitute V_2

$V_1 \rightarrow bV_4V_3|aV_3$ *new V_1-rule*

All the rules are fine now. Grammar is in Greibach NF.

$$V_1 \rightarrow bV_4V_3|aV_3$$
$$V_2 \rightarrow bV_4|a$$
$$V_3 \rightarrow b$$
$$V_4 \rightarrow bV_4V_3V_3|aV_3V_3$$

EXAMPLE 10.7.6: Find a GNF grammar, equivalent to the following CFG.

$$S \to AA|0$$
$$A \to SS|1$$

Solution:

Step-1: *Simplify G*
There are no useless productions, no $\in$-productions, no unit-productions. Nothing to do here.

Step-2: *Transform G into equivalent CNF.*
CFG is already in CNF.

Step-3: *Transform G into equivalent grammar G′ in GNF* Rename the variables in the Productions P' to $A_1, A_2, \ldots$.

$$S = A_1; A = A_2$$
$$A_1 \to A_2A_2|0$$
$$A_2 \to A_1A_1|1$$

Step-4: Order the productions and remove recursion, wherever necessary.

$A_1 \to A_2A_2|0$ No Modification needed
$A_2 \to A_1A_1|1$ Needs to be modified.

a. Modify A_2-rule
Substitute RHS of A_1 in A_2-rule.

$$A_2 \to A_2A_2A_1|0A_1|1$$

All the rules are in proper order.

$$A_1 \to A_2A_2|0$$
$$A_2 \to A_2A_2A_1|0A_1|1$$

b. Resolving Left recursion
$A_2 \to A_2A_2A_1|0A_1|1$ needs to be resolved.
Introduce a new variable B and a new production $B \to \alpha_1|\alpha_1B \ldots$.

Now,

$A_2 \rightarrow A_2A_2A_1|0A_1|1$ is nothing but,

$A_2 \rightarrow A_2A_2A_1|1$

$A_2 \rightarrow 0A_1|1$

Consider, $\alpha_1 = A_2A_1 \Rightarrow B \rightarrow A_2A_1$ and $B \rightarrow A_2A_1B$

$(\because B \rightarrow \alpha_1$ and $B \rightarrow \alpha_1 B)$.

Substituting B into the RHS of A_2 rules which is recursive:

$$A_2 \rightarrow A_2B|1B. \qquad (\because \alpha = 1 \Rightarrow B \rightarrow 1B)$$

Again, substitute for A_2 in A_2-rule.

$$A_2 \rightarrow 0A_1B|1B$$

Now, rules without recursion are:

$$A_1 \rightarrow A_2A_2|0$$
$$A_2 \rightarrow 0A_1B|1B$$
$$A_2 \rightarrow 0A_1|1$$
$$B \rightarrow A_2A_1$$
$$B \rightarrow A_2A_1B,$$

or the production rule in a compact form is:

$$A_1 \rightarrow A_2A_2|0$$
$$A_2 \rightarrow 0A_1B|1B|0A_1|1$$
$$B \rightarrow A_2A_1|A_2A_1B$$

Step-5: *Substitute to achieve GNF*

$A_1 \rightarrow A_2A_2|0$ needs to be modified.

$A_2 \rightarrow 0A_1B|1B|0A_1|1$

$B \rightarrow A_2A_1|A_2A_1B$ needs to be modified.

a. Modify A_1-rule: Substitute the R.H.S of A_2 in A_1-rule (only for starting variable in A_1-rule): $A_1 \rightarrow 0A_1BA_2|1BA_2|0A_1A_2|1A_2|0$.

b. Modify B-rule: Substitute the R.H.S of A_2 in B-rule:

$$B \rightarrow 0A_1BA_1|1BA_1|0A_1A_1|1A_1|0A_1BA_1B|1BA_1B|0A_1A_1B|1A_1B.$$

All the rules are in perfect order and the grammar is in GNF:

$$A_1 \rightarrow 0A_1BA_2|1BA_2|0A_1A_2|1A_2|0$$

$$A_2 \rightarrow 0A_1B|1B|0A_1|1$$

$$B \rightarrow 0A_1BA_1|1BA_1|0A_1A_1|1A_1|0A_1BA_1B|1BA_1B|0A_1A_1B|1A_1B$$

EXAMPLE 10.7.7: Transform the CFG into GNF, given $G = (\{A_1, A_2, A_3\}, \{a, b\}, P, A_1)$ and productions P as,

$$A_1 \rightarrow A_2A_3$$

$$A_2 \rightarrow A_3A_1|b$$

$$A_3 \rightarrow A_1A_2|a.$$

Solution:

Step-1: *Simplify G*
There are no useless productions, no $\in$-productions, no unit-productions. Nothing to do here.

Step-2: *Transform G into an equivalent CNF.*
G is already in CNF. Nothing to do here.

Step-3: *Transform G into an equivalent GNF.*

Order the productions and remove recursion, wherever necessary.

$A_1 \rightarrow A_2A_3$ okay

$A_2 \rightarrow A_3A_1|b$ okay

$A_3 \rightarrow A_1A_2|a$ Needs to be modified.

a. Modify A_3-rule:

- Substitute R.H.S. of A_1 *in* A_3-rule:
  $A_3 \rightarrow A_2A_3A_2|a$. Needs to be modified.
- Substitute R.H.S. of A_2 *in* A_3-rule:
  $A_3 \rightarrow A_3A_1A_3A_2|bA_3A_2|a$.

Now all the rules are ordered properly.

$$A_1 \rightarrow A_2A_3$$

$$A_2 \rightarrow A_3A_1|b$$

$$A_3 \rightarrow A_3A_1A_3|bA_3A_2|a.$$

b. Resolve left recursion.

Hence, $A_3 \rightarrow A_3A_1A_3A_2|bA_3A_2|a$ needs to be resolved. Introduce a new variable B and a new production $B \rightarrow \alpha_1|\alpha_1 B \ldots\ldots$
Consider, $\alpha = A_1A_3A_2$

$$B \rightarrow \alpha \Rightarrow B \rightarrow A_1A_3A_2$$
$$B \rightarrow \alpha B \Rightarrow B \rightarrow A_1A_3A_2B$$
$$\text{or} \quad B \rightarrow A_1A_3A_2|A_1A_3A_2B.$$

Substituting B into the RHS of A_3-rules, which is recursive:

$$A_3 \rightarrow A_3B|bA_3A_2|aB \qquad (\because \alpha = a \Rightarrow B \rightarrow aB.)$$

Again, substituting for A_3 in A_3-rule:

$$A_3 \rightarrow bA_3A_2B|aB|bA_3A_2|a.$$

Now rules, without recursion, are:

$$A_1 \rightarrow A_2A_3$$
$$A_2 \rightarrow A_3A_1|b$$
$$A_3 \rightarrow bA_3A_2B|aB|bA_3A_2|a$$
$$B \rightarrow A_1A_3A_2|A_1A_3A_2B.$$

Step-4: *Substitute backward to obtain GNF*

$A_1 \rightarrow A_2A_3$ needs to be modified.
$A_2 \rightarrow A_3A_1|b$ needs to be modified.
$A_3 \rightarrow bA_3A_2B|aB|bA_3A_2|a$
$B \rightarrow A_1A_3A_2|A_1A_3A_2B$ needs to be modified.

a. Modify A_2-rule:
Substitute the RHS of A_3 in A_2-rule (only for starting variable of A_2-rule).

$$A_2 \rightarrow bA_3A_2BA_1|aBA_1|bA_3A_2A_1|aA_1|b.$$

b. Modify A_1-rule:
Substitute the R.H.S. of A_2 in the A_1-rule (only for starting variable of A_1-rule)

$$A_1 \rightarrow bA_3A_2BA_1A_3|aBA_1A_3|bA_3A_2A_1A_3|aA_1A_3|bA_3.$$

c. Modify B-rule:
Substitute R.H.S. of A_1 in B-rule (only for the starting variable of B-rule).

$$B \rightarrow bA_3A_2BA_1A_3A_3A_2|bA_3A_2BA_1A_3A_3A_2B.|$$
$$aBA_1A_3A_3A_2|aBA_1A_3A_3A_2B|$$
$$bA_3A_2A_1A_3A_3A_2|bA_3A_2A_1A_3A_3A_2B.|$$
$$aA_1A_3A_3A_2|aA_1A_3A_3A_2B|$$
$$bA_3A_3A_2|bA_3A_3A_2B.$$

All the rules are properly ordered. Hence, the grammar is in GNF.

$$A_1 \rightarrow bA_3A_2BA_1A_3|aBA_1A_3|bA_3A_2A_1A_3|aA_1A_3|bA_3.$$
$$A_2 \rightarrow bA_3A_2BA_1|aBA_1|bA_3A_2A_1|aA_1|b.$$
$$A_3 \rightarrow bA_3A_2B|aB|bA_3A_2|a.$$
$$B \rightarrow bA_3A_2BA_1A_3A_3A_2|bA_3A_2BA_1A_3A_3A_2B|$$
$$aBA_1A_3A_3A_2|aBA_1A_3A_3A_2B|$$
$$bA_3A_2A_1A_3A_3A_2|bA_3A_2A_1A_3A_3A_2B.|$$
$$aA_1A_3A_3A_2|aA_1A_3A_3A_2B|$$
$$bA_3A_3A_2|bA_3A_3A_2B.$$

10.8 Chomsky vs. Greibach Normal Form

Chomsky Normal Form.

- CNF is named after Noam Chomsky.
- CNF puts restrictions on number of symbols, on the right side of production.
- CFG is in CNF, only if all productions are of form $A \rightarrow BC$ or $A \rightarrow a$.
- CNF is important because it yeilds efficient algorithms.

Greibach Normal Form

- GNF is named after Sheila Greibach.
- GNF puts restrictions not on the length of R.H.S., of production, but on the positions, in which terminals and variables can appear.
- CFG is in GNF, only if all the productions are of the form $A \rightarrow ax$.
- GNF is used
  a. to prove that every CFL can be accepted by non-deterministic PDA.

b. to construct a PDA equivalent of CFG.
c. and because it is convinent to use in parsing.

Note–2:
Kuroda Normal Form (KNF)
A CFG is in KNF, if and only if, all the productions are of the form

$$AB \rightarrow CD \text{ or } A \rightarrow BC \text{ or } A \rightarrow B \text{ or } A \rightarrow a$$

where, A, B, C and D are nonterminals and a is a terminal.

10.9 Application of Context-Free Grammars

10.9.1 *Parsing*

Parsing is nothing but finding a sequence of productions by which the string W in $L(G)$ is derived. Thus, parsing a string W is finding a derivation for that string i.e., a sequence of applications of production rules, in which starts with S and ends in W.

Parsing a string is like recognising a string. An algorithm to recognise a string will give only a yes/no as answer. However, an algorithm to parse a string will give additional information about how the string can be formed from the grammar. Generally, the only realistic way to recognise a string of a context-free grammar is to parse it.

10.9.2 *Design of programming languages*

CFG plays an important role in the design of programming languages. The use of balanced parentheses or the arithmetic expressions or the conditional expressions along with various operators, can be easily expressed using CFG. To describe the programming languages, the CFG uses **BNF notations**.

As discussed earlier, *Backus-Naur Form* or BNF is a compact notation to represent the production rule. CFG uses BNF for defining (the syntax of) context-free parts of programming languages. The advantages of using BNF is that it uses a small number of symbols (distinct from those) used in programming languages to define programming languages. The extended BNF notation mainly consists of:

a. Angular brackets - <>
This is to denote variables.

Example: - < *identifier* > denotes the class of identifiers.

b. ::=
It is used to denote 'is defined as' (alternate to → in production rule).

Example: < *identifier* >::=< *letter* > (< *letter* > | < *digit* >)
This states that an identifier is defined by writing a letter, followed by zero or more number of letters and digits.

c. '|', '*'
It means 'or', 'zero or more number of times'.

d. [ ]
It denotes zero or one.

Example: [< *digit* >] denotes zero or one of < *digit* >.
Thus, with extended BNF all non-terminals are written within <> and terminals are written without <>. The '→' in the production is replaced by '::='.

John Warner Backus (born in December 3, 1924) is an American computer scientist, notable as leader of the team that invented the first high-level programming language (FORTRAN), inventor of, the Backus-Naur form (BNF, the almost universally used notation to define formal language syntax) and also the concept of function-level programming. He received his B.S. and M.S. in Mathematics from Columbia University.

He received the W.W McDowell Award From the IEEE in 1967, the National Medal of Science in 1975, the A.M Turing Award from the Association for Computing Machinery in 1977 and an honorary doctorate from York University, England in 1985.

Peter Naur (born in October 25, 1928) is a Danish pioneer in Computer Science and winner of Turing Award. His last name is the *N* in the BNF notation (Backus-Naur form), used in the description of the syntax for most programming languages. He contributed to the creation of the ALGOL 60 programming language.

He received his M.A in astronomy from Copenhagen University in 1949 and Ph.D., in astronomy from the same University in 1957. In 1963, he was given the Hagemanns Gold Medal and three years later the Rosenhjaer Prize.

His main areas of inquiry are design, structure and performance of computer programs and algorithms. He has done pioneering works in areas such as software engineering and software architecture also.

< linking > ::= < a href = "path" > < linked-item >

< linked-item > ::= < string > | < image >

10.10 Exercises

1. Define useful and useless symbols in CFG.
2. Identify useful and useless symbols for the grammar shown below:
   a. $S \rightarrow aSa|bSb|A$
      $A \rightarrow aBb|bBa$
      $B \rightarrow aB|bB| \in$
   b. $S \rightarrow aA|bB$
      $A \rightarrow aA|a$
      $B \rightarrow bB$
      $D \rightarrow ab|Ea$
      $E \rightarrow aC|d$
3. What is reduction of grammar? Write a general procedure to reduce a given grammar.
4. Obtain a reduced grammar for the grammar shown below:

$$S \rightarrow aAa$$

$$A \rightarrow Sb|bCC|aDA$$

$$C \rightarrow ab|aD$$

$$E \rightarrow aC$$

$$D \rightarrow aAD.$$

5. What is an $\in$-production? Explain nullable variables with examples.

6. Identify the nullable variables from the grammar shown below:

$$S \rightarrow BAAB$$
$$A \rightarrow 0A_2|2A_0| \in$$
$$B \rightarrow AB|1B| \in .$$

7. For the grammars shown below, eliminate the $\in$-productions.

   a. $S \rightarrow ABAC$
   $A \rightarrow aA| \in$
   $B \rightarrow bB| \in$
   $C \rightarrow c.$

   b. $S \rightarrow aSa|bSb|A$
   $A \rightarrow aBb|bBa$
   $B \rightarrow aB|bB| \in .$

   c. $S \rightarrow aA|a|B|C$
   $A \rightarrow aB| \in$
   $B \rightarrow aA$
   $C \rightarrow cCD$
   $D \rightarrow abd.$

8. What is a unit production? Give a general method of eliminating unit productions, from the grammar.

9. Eliminate the unit productions from the following grammars:

   a. $S \rightarrow AB$
   $A \rightarrow a$
   $B \rightarrow C|b$
   $C \rightarrow D$
   $D \rightarrow E|bC$
   $E \rightarrow d|Ab.$

   b. $S \rightarrow A0|B$
   $B \rightarrow A|11$
   $A \rightarrow 0|12|B.$

   c. $S \rightarrow Aa|B|Ca$
   $B \rightarrow aB|b$
   $C \rightarrow Db|D$
   $D \rightarrow E|d$
   $E \rightarrow ab.$

10. What is Chomsky Normal form? Give the general procedure to transform a grammar to Chomsky Normal Form.

11. Simplify the following CFG and convert it into CNF.

a. $S \rightarrow AaB|aaB$
$A \rightarrow \in$
$B \rightarrow bbA| \in .$

b. $AE \rightarrow AE + T|T$
$T \rightarrow T * E|E$
$E \rightarrow (AE)|I$
$I \rightarrow a|b|c|Ia|Ib|Ic.$

12. What is Greibach Normal form? Give the general procedure to transform a grammar to Greibach Normal Form.

13. Convert the following grammars into GNF.

a. $S \rightarrow AB1|0$
$A \rightarrow 00A|B$
$B \rightarrow |A|.$

b. $A \rightarrow BC$
$B \rightarrow CA|b$
$C \rightarrow AB|a.$

14. Bring out the differences between CNF and GNF.