Automata Theory and Introduction to Compilers

By:
MEGHA L
Assistant Professor
Dept. of CSE, Dr. AIT

MODULE - 3

CONTEXT - FREE GRAMMARS AND LANGUAGES

5. Context-free Grammars and languages:

5.1 → Context-free grammars
5.1.2 → Definition
5.1.1 → Examples
5.1.3 → Derivation
5.2 → Parse trees
5.4 → Ambiguity in grammars and languages.

Text book:

1. John E. Hopcroft, Rajeev Motwani, Jeffrey D. Ullman: Introduction to Automata Theory, Languages and Computation, Publisher: Pearson Education; Third Edition (2011)

Context-free Grammars and Languages:

The regular languages to a larger class of languages, called the "context-free languages". These languages have a natural, recursive notation called "context-free grammars" (CFG).

CFG have played central role in compiler technology since the 1960's, turned the implementation of parsers from a time-consuming, ad-hoc implementation task into a routine job that can be done in an afternoon.

The context-free grammar has been used to describe document formats, via the so-called Document-type definition (DTD) that is used in the XML (extensible markup language) community for information exchange on the Web.

"Parse tree" is the product of a parser for a programming language & is the way that structure of programs is normally captured.

. "Parse tree", a picture of the structure that a grammar places on the strings of its language.

. There is an automaton-like notation called "pushdown automaton", that also describes all and only the CFL.

5.1. Context-Free Grammars

5.1.1 An Informal Example.

Let us consider the language of palindromes - A palindrome is a string that reads the same forward and backward, such as otto or madamimadam ("Madam, I'm Adam,")

. String w is a palindrome if and only if $w = w^R$.

. Consider describing only the palindromes with alphabet $\{0,1\}$.

This language includes strings like 0110, 11011, and ϵ, but not 011 or 0101.

It is easy to verify that the language L_{pal} of palindromes of 0's and 1's is not a Regular language.

Use the pumping lemma.

If L_{pal} is a RL, let n be the associated Constant, and consider the palindrome $w = 0^n 1 0^n$.

If L_{pal} is regular,

then, break w into $w = xyz$,

such that y consists of one or more 0's from the first group.

Thus, xz, would also have to be in L_{pal} if L_{pal} were regular, would have fewer 0's to the left of the done 1 than there are to the right of the 1.

∴, xz cannot be a palindrome.

There is a natural, recursive definition of when a string of 0's and 1's is in L_{pal}.

It starts with a basis saying that a few obvious strings are in L_{pal}, and then exploits the idea that if a string is a palindrome, it must begin and end with the same symbol.

Further, when the first and last symbols are removed, the resulting string must also be a palindrome.

That is:

BASIS: ϵ, 0, and 1 are palindromes.

INDUCTION: If w is a palindrome, so are $0w0$ and $1w1$. No string is a palindrome of 0's and 1's, unless it follows from this basis and induction rule.

CFG is a formal notation for expressing such recursive definitions of languages.

A grammar consists of one or more variables that represent classes of strings i.e, languages.

Example 1:-

The rule that define the palindromes, expressed in the Context-free grammar notation, as shown in figure below.

1. $P \longrightarrow \epsilon$
2. $P \longrightarrow 0$
3. $P \longrightarrow 1$
4. $P \longrightarrow 0P0$
5. $P \longrightarrow 1P1$

figure 1: A context-free grammar for palindrome.

The first 3 rules form the basis, i.e the class of palindromes includes the strings ϵ, 0 and 1. None of the right sides of these rules (the portions following the arrows) contains a variable, they form a basis for the definition.

The last 2 rules form the inductive part of the definition.

For instance, rule 4 says that if we take any string w from the class P, then 0w0 is also in class P.

Rule 5 tells us that 1w1 is also in P.

5.1.2 Definition of Context-Free Grammars

There are four important components in a grammatical description of a language:

1. There is a finite set of symbols that form the strings of the language being defined. This set was {0,1} in the palindrome example we just saw. We call this alphabet the terminals, or terminal symbols.

2. There is a finite set of variables, also called nonterminals or syntactic categories. Each variable represents a language. (i.e set of strings).

In above example, only one variable P, which we used to represent the class of palindromes over alphabet {0,1}.

3. One of the variables represents the language being defined; it is called the start symbol. Other variables represent auxiliary classes of strings that are used to help define the language of the start symbol.

In example P, the only variable, is the start symbol.

4. There is a finite set of productions or rules that represent the recursive definition of a language.

Each production consists of:

(a) A variable that is being (partially) defined by the production. The variable is often called the head of the production.

(b) The production symbol $\rightarrow$.

(c) A string of zero or more terminals and variables,

This string called the body of the production, represents one way to form strings in the language of the variable of the head.

Leave terminals unchanged and substitute for each variable of the body any string that is known to be in the language of that variable.

Example 1 productions in figure.

The 4 components just described form CFG or just grammar, shall represent a CFG G by its 4 components i.e, $G = (V, T, P, S)$.

Where V – is set of variables,
T – Terminals,
P – set of productions,
S – start symbol.

Example 2 :-

The grammar Gpal for the palindromes is represented by $G_{pal} = (\{P\}, \{0, 1\}, A, P)$.

where A – set of five productions saw in example 1 figure.

Example 3 :-

Let us explore a more complex CFG that represents expressions in a typical programming language.

Soln:- limit to the operators + and *, representing addition & multiplication.

- Allow arguments to be identifiers but instead of allowing the full set of typical identifiers (letters followed by zero or more letters and digits), allows only the letters a and b and the digits 0 and 1.

Every identifier must begin with a or b, be followed by any string in {a, b, 0, 1}*.

We need two variables in this grammar. One, call E, represents expressions. It is the start symbol and represents the language of expressions we are defining.

Other variable I, represents identifiers. Its language is actually regular it is the language of the R.E.

$$(a+b).(a+b+0+1)^*$$

However, we shall not use R.E directly in gramm-ars. we a set of productions that say essentially the same thing as this R.E.

1. $E \longrightarrow I$
2. $E \longrightarrow E+E$
3. $E \longrightarrow E*E$
4. $E \longrightarrow (E)$
5. $I \longrightarrow a$
6. $I \longrightarrow b$
7. $I \longrightarrow Ia$
8. $I \longrightarrow Ib$
9. $I \longrightarrow I0$
10. $I \longrightarrow I1$.

Figure2: A context-free grammar for simple expressions

The grammar for expressions is stated formally as $G = (\{E, I\}, T, P, E)$ where

T - set of symbols {+, *, (,), a, b, 0, 1} and

P - set of productions shown in above figure.

Rule (1) is the basis rule for expressions. (expression Can be a single identifier)

Rule (2) through (4) describes the inductive case for expressions.

Rule (2) says that an expression can be two expressions connected by + sign,

Rule (3) says the same with * sign,

Rule (4) take any expression and put matching parentheses around it, the result is also an expression.

Rule (5) through (10) describe identifiers I.

The basis is rules (5) and (6), they say that a and b are identifiers.

The remaining four rules are the inductive case.

5.1.3 Derivations Using a Grammar

Apply the productions of CFG to infer that certain strings are in the language of a certain variable.

There are 2 approaches to this inference.

The more conventional approach is to use the rules from body to head.

We take strings known to be in the language of each of the variables of the body, concatenate them, in the proper order, with any terminals appearing in the body, & infer that the resulting string is in the language of the variable in the head.

This procedure is refered as Recursive inference.

There is another approach to defining the language of a grammar, in which we use the productions from

head to body.

Expand the start symbol using one of its productions (i.e, using a production whose head is the start symbol).

Further productions expand the resulting string by replacing one of the variables by the body of one of its productions, and so on, until we derive a string consisting entirely of terminals.

The language of the grammar is all strings of terminals that we can obtain in this way.

This use of grammars is called derivation.

First approach, Recursive inference - is often more natural to think of grammars are used in derivations, and develop the notation for describing these derivations.

Example 4:-

Figure 3: Inferring strings using the grammar of figure 2.

	String Inferred	For language of	Production used	String(s) used
		I	5	-
(i)	a	I	10	(i)
(ii)	a1	I	9	(ii)
(iii)	a10	I	10	(iii)
(iv)	a101	I	6	-
(v)	b	I	10	(v)
(vi)	b1	I	1	(vi)
(vii)	a1	E	1	(iv)
(viii)	a101	E	1	(v)
(ix)	b	E	1	(vi)
(x)	b1	E	2	(viii), (x)
(xi)	a101 + b1	E	4	(xi)
(xii)	a101 + b1	E	2	(ii), (v)
(xiii)	a1 + b	E	4	(xiii)
(xiv)	a1 + b	E	3	(xii), (xiv)
(xv)	(a101 + b1) * (a1+b)	E	3	(xi), (xiii)

Let us consider some of the inferences we can make using the grammar for expressions in figure 2. Figure 3. summarizes these inferences.

For example, line (i) - infer string a is in the language for I by using production 5.

Line (i) through (iv) - infer that a 1 and a 101 are identifiers by using production 5 once (to get the a) and then applying production 9 once (to attach the 0) and production 10 twice (to attach the two 1's).

Similarly lines (v) through (vi) infer that b and b1 are identifiers.

Lines (vii) through (x) exploit production 1 to infer that, since any identifier is an expression, the strings a1, a101, b and b1, which we inferred in lines (i) through (vi) to be identifiers, are also in the language of variable E.

Lines (xi) & (xiii) use production 2 to infer the sum of these identifiers is an expression.

Lines (xii) and (xiv) use production 4 to infer that the same strings with parentheses around them are also expressions, & line (xv) uses production 3 to multiply them.

The process of deriving strings by applying productions from head to body requires the definition of a new relation symbol $\Rightarrow$.

Suppose $G = (V, T, P, S)$ is a CFGF.

Let $\alpha A \beta$ be a string of terminals and variables, with A a variable.

i.e, α and β are strings in $(V \cup T)^*$, and A is in V.

Let $A \to \gamma$ be a production of G.

Then we say, $\alpha A \beta \underset{G}{\Rightarrow} \alpha \gamma \beta$.

If G is understood,

$\alpha A \beta \Rightarrow \alpha \gamma \beta$.

One derivation step replaces any variable anywhere in the string by the body of one of its productions.

We may extend the $\Rightarrow$ relationship to represent zero, one or many derivation steps, much as the transition function δ of a finite automaton was extended to $\hat{\delta}$.

For derivations, we use a^* to denote "zero or more steps", as follows:

BASIS: For any string α of terminals and variables, we say $\alpha \underset{G}{\overset{*}{\Rightarrow}} \alpha$.

i.e, any string derives itself.

INDUCTION: If $\alpha \underset{G}{\overset{*}{\Rightarrow}} \beta$ and $\beta \underset{G}{\Rightarrow} \gamma$, then

$\alpha \underset{G}{\overset{*}{\Rightarrow}} \gamma$.

i.e, if α can become β by zero or more steps, and one more steps takes β to γ, then α can become γ,

i.e, $\alpha \underset{G}{\overset{*}{\Rightarrow}} \beta$ means that there is a sequence of strings

$\gamma_1, \gamma_2, \ldots\ldots \gamma_n$, for some $n \geq 1$, such that

1. $\alpha = \gamma_1$,
2. $\beta = \gamma_n$, and
3. For $i = 1, 2, \ldots\ldots, n-1$, we have

$$\gamma_i \Rightarrow \gamma_{i+1}.$$

If Grammar G is understood, then we use $\overset{*}{\Rightarrow}$ in place of $\overset{*}{\underset{G}{\Rightarrow}}$.

Example 5 :-

(a101 + b1) * (a1 + b) is in the language of variable E by showing a derivation starting with E as given below :

Soln :- $E \Rightarrow E * E$

$\Rightarrow (E) * E$

$\Rightarrow (E+E) * E$

$\Rightarrow (I+E) * E$

$\Rightarrow (I1+E) * E$

$\Rightarrow (I01+E) * E$

$\Rightarrow (I101+E) * E$

$\Rightarrow (a101+E) * E$

$\Rightarrow (a101+I) * E$

$\Rightarrow (a101+I1) * E$

$\Rightarrow (a101+b1) * E$

$\Rightarrow (a101+b1) * (E)$

$\Rightarrow (a101+b1) * (E+E)$

$\Rightarrow (a101+b1) * (I+E)$

$\Rightarrow (a101+b1) * (I1+E)$

$\Rightarrow (a101+b1) * (a1+E)$

$\Rightarrow (a101+b1) * (a1+I)$

$\Rightarrow (a101+b1) * (a1+b)$

[Refer Figure 2] Simple expressions.

NOTE :-

We can use the $\Rightarrow$ relationship to condense the derivation.

We know $E \Rightarrow E$ by the Basis.

Repeated use of induction yields $E \overset{*}{\Rightarrow} E * E$,

$E \overset{*}{\Rightarrow} (E) * E$,

$E \overset{*}{\Rightarrow} (I + E) * E$ and so on,

until finally $E \overset{*}{\Rightarrow} (a101 + b1) * (a1 + b)$.

The two viewpoints - Recursive inference & derivation - are equivalent. i.e, a string of terminals w is inferred to be in the language of some variable A if and only if $A \overset{*}{\Rightarrow} w$.

~~However, the proof of this fact requires some~~ ~~work~~.

5.1.4. Leftmost and Rightmost Derivations

In order to restrict the number of choices we have in deriving a string, it is often useful to require that at each step we replace the leftmost variable by one of its production bodies.

Such a derivation is called Leftmost derivation (LMD).

Indicate that a derivation is leftmost by using the relations $\underset{lm}{\Rightarrow}$ and $\overset{*}{\underset{lm}{\Rightarrow}}$, for one or many steps, respectively.

Similarly, it is possible to require that at each step the rightmost variable is replaced by one of its bodies. Call the derivation rightmost and use the symbols $\underset{rm}{\Rightarrow}$ and $\overset{*}{\underset{rm}{\Rightarrow}}$ to indicate one or many rightmost derivation steps, respectively.

Example 6 :-

The derivation of example 5 was an solution of Leftmost derivation (LMD). and described as follows:

$$E \underset{lm}{\Rightarrow} E*E \Rightarrow (E)*E \Rightarrow (E+E)*E$$
$$\underset{lm}{\Rightarrow} (I+E)*E \underset{lm}{\Rightarrow} (I1+E)*E$$
$$\underset{lm}{\Rightarrow} (I01+E)*E \underset{lm}{\Rightarrow} (I101+E)*E$$
$$\underset{lm}{\Rightarrow} (a101+E)*E$$
$$\underset{lm}{\Rightarrow} (a101+I)*E \underset{lm}{\Rightarrow} (a101+I1)*E$$
$$\underset{lm}{\Rightarrow} (a101+b1)*E$$
$$\underset{lm}{\Rightarrow} (a101+b1)*(E)$$
$$\underset{lm}{\Rightarrow} (a101+b1)*(I+E)$$
$$\underset{lm}{\Rightarrow} (a101+b1)*(E+E)$$
$$\underset{lm}{\Rightarrow} (a101+b1)*(I1+E)$$
$$\underset{lm}{\Rightarrow} (a101+b1)*(a1+E).$$
$$\underset{lm}{\Rightarrow} (a101+b1)*(a1+I)$$
$$\underset{lm}{\Rightarrow} (a101+b1)*(a1+b)$$

(or) Summarise it as follows:

$$E \overset{*}{\underset{lm}{\Rightarrow}} (a101+b1)*(a1+b).$$

The derivation for the above can be written as Rightmost derivation (RMD).

It is given by -

$E \underset{rm}{\Rightarrow} E*E \underset{rm}{\Rightarrow} E*(E) \underset{rm}{\Rightarrow} E*(E+E)$

$\underset{rm}{\Rightarrow} E*(E+I) \underset{rm}{\Rightarrow} E*(E+b) \underset{rm}{\Rightarrow} E*(I+b)$

$\underset{rm}{\Rightarrow} E*(I1+b)$

$\underset{rm}{\Rightarrow} (E)*(a1+b)$

$\underset{rm}{\Rightarrow} (E+E)*(a1+b)$

$\underset{rm}{\Rightarrow} (E+I)*(a1+b)$

$\underset{rm}{\Rightarrow} (E+I1)*(a1+b)$

$\underset{rm}{\Rightarrow} (E+b1)*(a1+b)$

$\underset{rm}{\Rightarrow} (I+b1)*(a1+b)$

$\underset{rm}{\Rightarrow} (I1+b1)*(a1+b)$

$\underset{rm}{\Rightarrow} (I01+b1)*(a1+b)$

$\underset{rm}{\Rightarrow} (I101+b1)*(a1+b)$

$\underset{rm}{\Rightarrow} (a101+b1)*(a1+b)$

This derivation allows us to consider

$E \underset{rm}{\overset{*}{\Rightarrow}} a*(a+b00)$.

NOTE:- Any derivation has an equivalent LMD and an equivalent RMD. i.e, if w is a terminal string, and A is a variable, then

$A \overset{*}{\Rightarrow} w$. if and only if $A \underset{lm}{\overset{*}{\Rightarrow}} w$, and

$A \overset{*}{\Rightarrow} w$ if and only if $A \underset{lm}{\overset{*}{\Rightarrow}} w$.

5.1.5 The language of a Grammar.

If $G(V, T, P, S)$ is a CFG, the language of G, denoted $L(G)$, is the set of terminal strings that have derivations from the start symbol.

i.e, $L(G) = \{w \text{ in } T^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w\}$.

If a Language L is the language of some CFG, then L is said to be CFL (Context-Free language).

For instance, the grammar of figure 1 defined the language of palindromes over alphabet $\{0,1\}$.

Thus, the set of palindromes is a CFL.

We can prove that statement, as follows:

Theorem:

$L(G_{pal})$, where G_{pal} is the grammar of example 1, is the set of palindrome over $\{0,1\}$.

Proof:- prove that a string w in $\{0,1\}^*$ is in $L(G_{pal})$ if and only if it is a palindrome,

i.e $w = w^R$.

(If) Suppose w is a palindrome.

By induction on $|w|$ that w is in $L(G_{pal})$.

BASIS: We use length 0 and 1 as the basis.

If $|w| = 0$ or $|w| = 1$, then

w is ϵ, 0 or 1.

Since there are productions $P \to \epsilon$, $P \to 0$, and $P \to 1$, we conclude that $P \overset{*}{\Rightarrow} w$ in any of the basis cases.

INDUCTION:

Suppose $|w| \geq 2$.

Since $w = w^R$, w must begin and end with the same symbol.

i.e, $w = 0x0$ (or) $w = 1x1$.

Moreover, x must be a palindrome.

i.e $x = x^R$.

Note that $|w| \geq 2$. to infer that there are two distinct 0's or 1's, at either end of w.

If $w = 0x0$, then, inductive hypothesis to claim that $P \overset{*}{\Rightarrow} x$.

Then there is a derivation of w from P, namely $P \Rightarrow 0P0 \overset{*}{\Rightarrow} 0x0 = w$.

If $w = 1x1$, then, argument is the same; but we use the production $P \to 1P1$ at the first step.

Conclude that w is in $L(G_{pal})$ and complete the proof.

(Only if), Assume that w is in $L(G_{pal})$.

i.e, $P \overset{*}{\Rightarrow} w$. we must conclude that w is a palindrome.

The proof is an induction on the number of steps in a derivation of w from P.

BASIS: If the derivation is one step, then it must use one of the three productions that do not have P in the body.

i.e, the derivation is $P \Rightarrow \epsilon$, $P \Rightarrow 0$, or $P \Rightarrow 1$

Since ϵ, 0, and 1 are all palindromes, the basis is proven.

INDUCTION:

Now, suppose that the derivation takes $n+1$ steps, where $n \geq 1$, and the statement is true for all derivations of n steps.

i.e, if $P \overset{*}{\Longrightarrow} x$ in n steps, then x is a palindrome

Consider an $(n+1)$ step derivation of w, which must be of the form

$$P \Longrightarrow 0P0 \overset{*}{\Longrightarrow} 0x0 = w.$$

(or) $$P \Longrightarrow 1P1 \overset{*}{\Longrightarrow} 1x1 = w,$$

since $n+1$ steps is at least two steps, and the productions $P \longrightarrow 0P0$ and $P \longrightarrow 1P1$. are the only productions whose use allows additional steps of a derivation.

Note that in either case, $P \overset{*}{\Longrightarrow} x$ in n steps.

By the inductive hypothesis, W.K.T x is a palindrome i.e, $x = x^R$. But if so, then $0x0$ and $1x1$ are also palindrome.

For instance, $(0x0)^R = 0x^R0 = 0x0$.

$\therefore$ w is palindrome, which Completes the proof.

5.1.6 Sentential Forms.

Derivations from the start symbol produce strings that have a special role called "Sentential forms".

i.e, if $G = (V, T, P, S)$ is a CFG, then any string α in $(V \cup T)^*$ such that $S \overset{*}{\Longrightarrow} \alpha$ is a sentential form.

If $S \underset{lm}{\overset{*}{\Rightarrow}} \alpha$, then α is a left-sentential form, and

If $S \underset{rm}{\overset{*}{\Rightarrow}} \alpha$, then α is a right-sentential form.

NOTE :- The Language L(G) is those sentential forms that are in T^*, i.e, they consist solely of terminals.

Example 7 :-

Consider the grammar for the expressions from figure 2.

For example, E * (I+E) is a sentential form, since there is a derivation.

$$E \Rightarrow E*E \Rightarrow E*(E) \Rightarrow E*(E+E)$$
$$\Rightarrow E*(I+E).$$

This derivation is neither LMD or RMD, since at the last step, the middle E is replaced.

As an example of a Left-sentential form, Consider a*E, with the LMD.

$$E \underset{lm}{\Rightarrow} E*E$$
$$E \underset{lm}{\Rightarrow} I*E$$
$$E \underset{lm}{\Rightarrow} a*E.$$

III^{ly}, with the RMD

$E \underset{rm}{\Rightarrow} E * E$

$E \underset{rm}{\Rightarrow} E * (E)$

$E \underset{rm}{\Rightarrow} E * (E+E)$.

Shows that $E * (E+E)$ is a right-sentential form.

Problems :-

(1.) The following grammar generates the grammar of the language consisting of all strings of even length :

$S \longrightarrow AS \mid \epsilon$

$A \longrightarrow aa \mid ab \mid ba \mid bb$.

Give leftmost and Rightmost derivations for the following strings :

a) aabbba

b) baabab

c) aaabbb.

Soln:- a) aabbba.

LMD :-

$S \underset{lm}{\longrightarrow} AS$

$S \underset{lm}{\longrightarrow} aaS \quad (A \rightarrow aa)$

$S \underset{lm}{\longrightarrow} aaAS \quad (S \rightarrow AS)$

$S \underset{lm}{\longrightarrow} aabbS \quad (A \rightarrow bb)$

$S \underset{lm}{\longrightarrow} aabbAS \quad (S \rightarrow AS)$

$S \underset{lm}{\longrightarrow} aabbbaS \quad (A \rightarrow ba)$

$S \underset{lm}{\longrightarrow} aabbba\epsilon \quad (S \rightarrow \epsilon)$.

$S \Rightarrow aabbba$ Hence Proved for LMD

RMD :-

$S \xrightarrow[rm]{} AS$

$S \xrightarrow[rm]{} AAS$ ($S \to AS$)

$S \xrightarrow[rm]{} AAAS$ ($S \to AS$)

$S \xrightarrow[rm]{} AAA\epsilon$ ($S \to \epsilon$)

$S \xrightarrow[rm]{} AAba$ ($A \to ba$)

$S \xrightarrow[rm]{} Abbba$ ($A \to bb$)

$S \xrightarrow[rm]{} aabbba$ ($A \to aa$)

Hence proved for RMD.

b) baabab.

Soln:- LMD

$S \xrightarrow[lm]{} AS$

$S \xrightarrow[lm]{} baS$

$S \xrightarrow[lm]{} baAS$

$S \xrightarrow[lm]{} baabS$

$S \xrightarrow[lm]{} baabAS$

$S \xrightarrow[lm]{} baababS$

$S \xrightarrow[lm]{} baabab\epsilon$

$S \xrightarrow[lm]{} baabab.$

Hence proved

RMD

$S \xrightarrow[rm]{} AS$

$S \xrightarrow[rm]{} AAS$

$S \xrightarrow[rm]{} AAAS$

$S \xrightarrow[rm]{} AAA\epsilon$

$S \xrightarrow[rm]{} AAA$

$S \xrightarrow[rm]{} AAab$

$S \xrightarrow[rm]{} Aabab$

$S \xrightarrow[rm]{} baabab.$

Hence proved

c) aaabbb

Soln:- LMD

$S \xrightarrow[lm]{} AS$

$S \xrightarrow[lm]{} AAS$

$S \xrightarrow[lm]{} aaAS$

$S \xrightarrow[lm]{} aaabS$

$S \xrightarrow[lm]{} aaabAS$

$S \xrightarrow[lm]{} aaabbbS$

$S \xrightarrow[lm]{} aaabbb\epsilon$

$S \xrightarrow[lm]{} aaabbb$.

Hence proved.

RMD

$S \xrightarrow[rm]{} AS$

$S \xrightarrow[rm]{} AAS$

$S \xrightarrow[rm]{} AAAS$.

$S \xrightarrow[rm]{} AAA\epsilon$

$S \xrightarrow[rm]{} AAbb$

$S \xrightarrow[rm]{} Aabbb$

$S \xrightarrow[rm]{} aaabbb$

Hence proved.

(2.) Consider the CFG G defined by productions:

$S \longrightarrow aSbS \mid bSaS \mid \epsilon$.

Prove that L(G) is the set of all strings with an equal number of a's and b's.

Soln:- LMD:-

Case 1:- $S \xrightarrow[lm]{} aSbS$

$S \xrightarrow[lm]{} a\epsilon bS$.

$S \xrightarrow[lm]{} abS$.

$S \xrightarrow[lm]{} ab\epsilon$.

$S \xrightarrow[lm]{} ab$.

Hence proved.

Case 2:- $S \xrightarrow[lm]{} bSaS$

$S \xrightarrow[lm]{} baSbSaS$

$S \xrightarrow[lm]{} ba\epsilon bSaS$

$S \xrightarrow[lm]{} babSaS$.

$S \xrightarrow[lm]{} bab\epsilon aS$.

$S \xrightarrow[lm]{} babaS$.

$S \xrightarrow[lm]{} baba\epsilon$

$S \xrightarrow[lm]{} baba$

Hence proved

NOTE:- The condition is equal number of a's & b's to find the L(G).

5.2 Parse Trees

Tree shows how the symbols of a terminal string are grouped into substrings, each of which belongs to the language of one of the variables of the grammar.

The tree known as a "parse tree" when used in compiler, is the data structure of Choice to represent the source program.

In a Compiler, the tree structure of the source program facilitates the translation of the source program into executable code by allowing natural, recursive function to perform this translation process.

5.2.1 Constructing Parse Trees

Let us fix on a grammar $G = (V, T, P, S)$.

The parse trees for G are trees with the following conditions:

1. Each interior node is labeled by a variable in V.
2. Each leaf is labeled by either a variable, a terminal, or ϵ.
   If the leaf is labelled ϵ, then it must be the only child of its labeled.
3. If an interior node is labeled A, and its children are labelled $X_1, X_2, \ldots, X_k$ respectively, from the left, then
   $A \longrightarrow X_1 X_2 \cdots X_k$ is a production in P.

NOTE :- the only time one of the X's can be ϵ is if that is the label of the only child, and $A \rightarrow \epsilon$ is a production of G.

Example 8 :- Figure (a) shows a parse tree which uses the expression grammar given below.

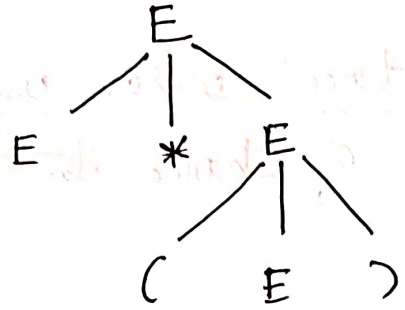


Figure (a): A parse tree showing the derivation of E*(E) from E.

1. $E \rightarrow I$
2. $E \rightarrow E+E$
3. $E \rightarrow E*E$
4. $E \rightarrow (E)$
5. $I \rightarrow a$
6. $I \rightarrow b$
7. $I \rightarrow Ia$
8. $I \rightarrow Ib$
9. $I \rightarrow Io$
10. $I \rightarrow I1$

The root is labelled with the variable E. The production at the root is $E \rightarrow E*E$. Since the three children of the root have labels E, *, and E.

Example 9 :- Figure (b) shows a parse tree for the palindrome grammar ~~figure~~ given below.

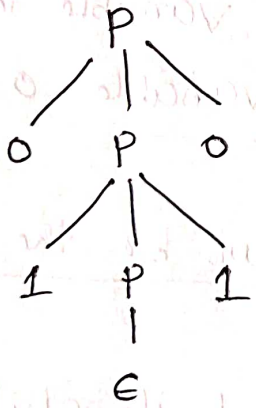


Figure (b): A parse tree showing the derivation $P \overset{*}{\Rightarrow} 0110$

1. $P \rightarrow \epsilon$
2. $P \rightarrow 0$
3. $P \rightarrow 1$
4. $P \rightarrow 0P0$
5. $P \rightarrow 1P1$

The production used at the root is $P \rightarrow 0P0$, and at the middle child of the root it is $P \rightarrow 1P1$.

Note :- At the bottom is a use of the production $P \rightarrow \epsilon$. That use, where the node labeled by the head has one child, labeled ϵ, is the only time that a node labeled ϵ can appear in a parse tree.

5.2.2. The yield of a parse tree

If we look at the leaves of any parse tree and concatenate them from the left, we get a string, called the yield of the tree, which is always a string that is derived from the root variable.

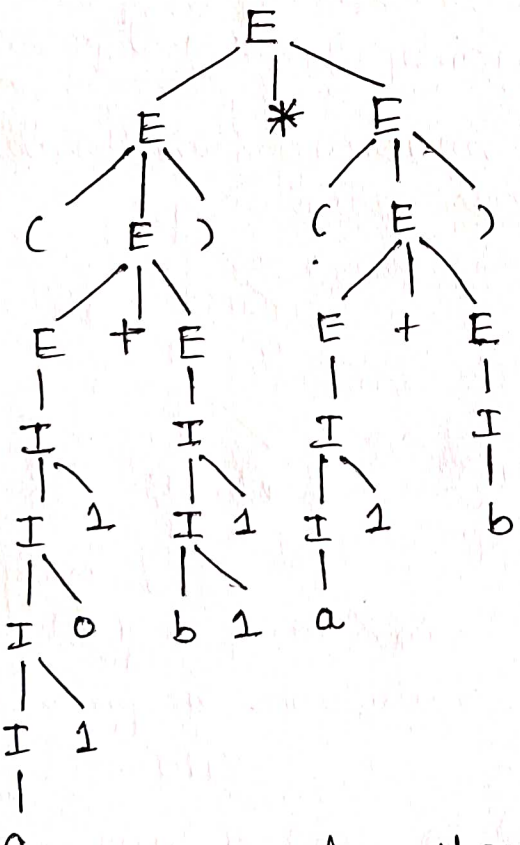
Special importance of are those parse trees such that:

1. The yield is a terminal string. i.e, all leaves are labeled either with a terminal or with ϵ.

2. The root is labeled by the start symbol.

Example 10:- Figure(c) is an example of a tree with a terminal string as yield and the start symbol at the root.

This tree's yield is the string (a101+b1)*(a1+b) that was derived in example 5



Figure(c): Parse tree showing (a101+b1)*(a1+b) is in the language of our expression grammar.