Context Free Grammar(CFG)

Jay Mehta
Department of Information & Technology
Shah & Anchor Kutchhi Engineering College
University of Mumbai , India
jay.mehta@sakec.ac.in

***Abstract*—Setting free syntaxes comprise of terminals (w1, w2, … , wV), nonterminals (N1, N 2,… , Nn), a beginning image (N1), and rules. Terminal images speak to setting that show up in the strings produced by the syntax. Nonterminal images are placeholders for examples of terminal images that can be created by nonterminals. A beginning image must be utilized as an exceptional nonterminal to show up during the underlying string age. Rules are utilized to supplant nonterminals in a string with different terminals/nonterminals.**

I. INTRODUCTION

In formal language theory, a context-free grammar (CFG) is a formal grammar in which every production rule is of the form

$$A \to \alpha$$

Where A is a single nonterminal symbol, and α is a string of terminals and/or nonterminals (α can be empty). A formal grammar is considered "context free" when its production rules can be applied regardless of the context of a nonterminal. No matter which symbols surround it, the single nonterminal on the left hand side can always be replaced by the right hand side. This is what distinguishes it from a context-sensitive grammar.A formal grammar is essentially a set of production rules that describe all possible strings in a given formal language. Production rules are simple replacements.

A context-free grammar *G* is defined by the 4-tuple:

$$G = (V, \Sigma, R, S)$$

- *V* is a finite set; each element $v \in V$ is called *a nonterminal character* or a *variable*. Each variable represents a different type of phrase or clause in the sentence. Variables are also sometimes called syntactic categories. Each variable defines a sub-language of the language defined by *G*.
- Σ is a limited arrangement of terminals, disjoint from V, which make up the genuine substance of the sentence. The arrangement of terminals is the letters in order of the language characterized by the punctuation G.
- is a finite relation from V to $(V \cup \Sigma)^*$ where the asterisk represents the Kleene star operation. The members of R are called the (rewrite) rules or productions of the grammar. (also commonly symbolized by a P)
- S is the start variable (or start symbol), used to represent the whole sentence (or program). It must be an element of V.

II. DESIGN AND DERIVATIONS

A. Basic Strategy for CFG Design :-

1. Understand the specification of *L*. Produce examples of strings in and not in the language..
2. Rundown the briefest strings in the language so as to decide the "base case" creations.
3. Comprehend the key auxiliary properties of the language; that is, distinguish the guidelines for joining littler sentences into bigger ones. These standards will decide the valuable creations that fabricate the long sentences in the language.
4. Test your model CFG on various painstakingly picked models. The entirety of the base cases ought to be tried, alongside the entirety of the elective creations (for a given left-hand nonterminal). Check if the syntax is predictable with your model strings from stage 1.
5. Demonstrate that your punctuation is right. This requires two contentions. One is that each string logical in G is in L; that is, L(G) ⊆ L. The other is that each x in L has a determination in G; that is, L ⊆ L(G). Either of these might be self-evident. Know about the way that there are regularly unobtrusive imperfections in a sentence structure that can permit you to infer strings not in the expected language

B. Derivation & Syntax Trees

A *derivation* of a string for a grammar is a sequence of grammar rule applications that transform the start symbol into the string. A derivation proves that the string belongs to the grammar's language.

A derivation is fully determined by giving, for each step:

- the rule applied in that step
- the occurrence of its left-hand side to which it is applied

For clarity, the intermediate string is usually given as well.

1. $S \to S + S$
2. $S \to 1$
3. $S \to a$

the string

$1 + 1 + a$

can be derived from the start symbol *S* with the following derivation:

$\rightarrow S + S$ (by rule 1. on S)

$\rightarrow S + S + S$ (by rule 1. on the second S)

$\rightarrow 1 + S + S$ (by rule 2. on the first S)

$\rightarrow 1 + S + S$ (by rule 2. on the first S)

$\rightarrow 1 + 1 + S$ (by rule 2. on the second S)

$\rightarrow 1 + 1 + a$ (by rule 3. on the third S)

Often, a strategy is followed that deterministically chooses the next nonterminal to rewrite:

- in a *leftmost derivation*, it is always the leftmost nonterminal;
- in a *rightmost derivation*, it is always the rightmost nonterminal.

Given such a strategy, a derivation is completely determined by the sequence of rules applied. For instance, one leftmost derivation of the same string is

S

$\rightarrow S + S$ (by rule 1 on the leftmost S)

$\rightarrow 1 + S$ (by rule 2 on the leftmost S)

$\rightarrow 1 + S + S$ (by rule 1 on the leftmost S)

$\rightarrow 1 + 1 + S$ (by rule 2 on the leftmost S)

$\rightarrow 1 + 1 + a$ (by rule 3 on the leftmost S),

which can be summarized as

rule 1

rule 2

rule 1

rule 2

rule 3.

One rightmost derivation is:

S

$\rightarrow S + S$ (by rule 1 on the rightmost S)

$\rightarrow S + S + S$ (by rule 1 on the rightmost S)

$\rightarrow S + S + a$ (by rule 3 on the rightmost S)

$\rightarrow S + 1 + a$ (by rule 2 on the rightmost S)

$\rightarrow 1 + 1 + a$ (by rule 2 on the rightmost S),

which can be summarized as :

rule 1

rule 1

rule 3

rule 2

rule 2.

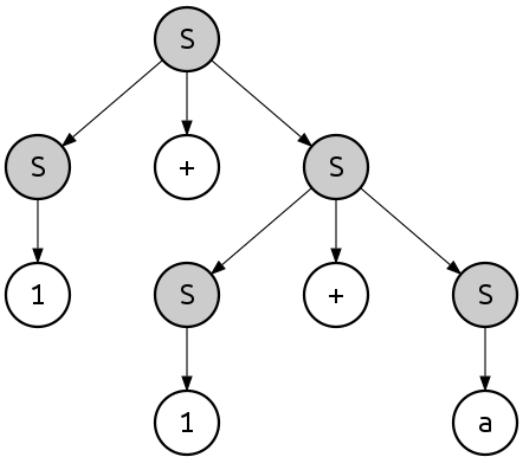
The distinction between leftmost derivation and rightmost derivation is important because in most parsers the transformation of the input is defined by giving a piece of code for every grammar rule that is executed whenever the rule is applied. Therefore, it is important to know whether the parser determines a leftmost or a rightmost derivation because this determines the order in which the pieces of code will be executed.

For instance LL and LR Parses are given as,

A derivation also imposes in some sense a hierarchical structure on the string that is derived. For example, if the string "1 + 1 + a" is derived according to the leftmost derivation outlined above, the structure of the string would be:

$\{\{1\}_S + \{\{1\}_S + \{a\}_S\}_S\}_S$

where $\{...\}_S$ indicates a substring recognized as belonging to S. This hierarchy can also be seen as a tree:



This tree is called a *parse tree* or "concrete syntax tree" of the string, by contrast with the abstract syntax tree. In this case the presented leftmost and the rightmost derivations define the same parse tree; however, there is another rightmost derivation of the same string

S

$\rightarrow S + S$ (by rule 1 on the rightmost S)

$\rightarrow S + a$ (by rule 3 on the rightmost S)

$\rightarrow S + S + a$ (by rule 1 on the rightmost S)

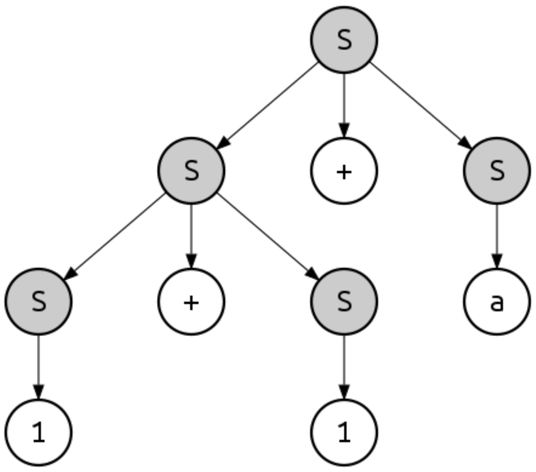
$\rightarrow S + 1 + a$ (by rule 2 on the rightmost S)

$\rightarrow 1 + 1 + a$ (by rule 2 on the rightmost S),

which defines a string with a different structure

$\{\{\{1\}_S + \{a\}_S\}_S + \{a\}_S\}_S$

and a different parse parse tree:



Note however that both parse trees can be obtained by both leftmost and rightmost derivations. For example, the last tree can be obtained with the leftmost derivation as follows:

S

$\rightarrow S + S$ (by rule 1 on the leftmost S)

$\rightarrow S + S + S$ (by rule 1 on the leftmost S)

$\rightarrow 1 + S + S$ (by rule 2 on the leftmost S)

$\rightarrow 1 + 1 + S$ (by rule 2 on the leftmost S)

$\rightarrow 1 + 1 + a$ (by rule 3 on the leftmost S),

If there is more than one parsing tree in a string in the grammar language then grammar is said to be an ambiguous grammar. These grammars are typically difficult to decipher, since the parser is not always able to determine which grammar rule to apply. Ambiguity is typically a function of grammar, not language, and an unambiguous grammar that produces the same context-free language can be found. There are, however, other languages that can only be produced by ambiguous grammars; inherently ambiguous languages are called these.

III. Examples and Related Work

These certain examples will give us an idea about how CFGs work out in real time problems as well theoretical situations.They will also tell us how it is differentiated from other types of languages ,i.e what are its constraints that define a CFG and separate it from like Regular Expressions(RegEx) which is comprised by Regular Grammar .

Also certain Real life projects with vast scope for CFG usage shall be discussed in this section.

A. *Examples*

Let us consider a basic example firstly,

1) Let any set of production rules in a CFG be

S → X+X | X*X |X| a

over an alphabet {a}.

The **leftmost** derivation for the string "a+a*a" maybe,

S → X+X

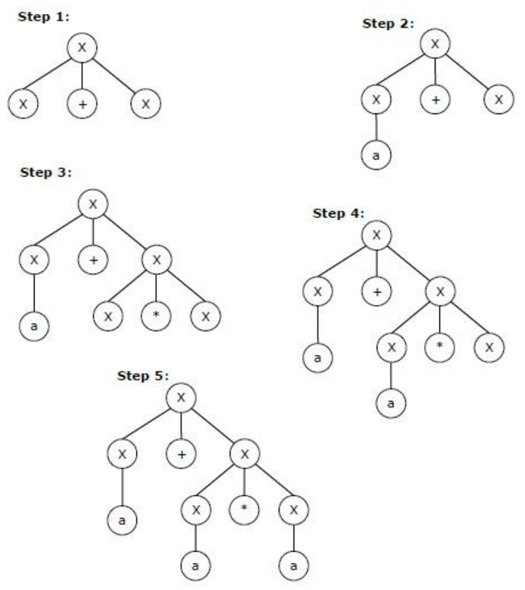
→ a+X

→ a + X*X

→ a+a*X

→ a+a*a

The stepwise derivation of the above string is shown as below in the figure,



The **rightmost** derivation for the above string "a+a*a" maybe,

S → X*X

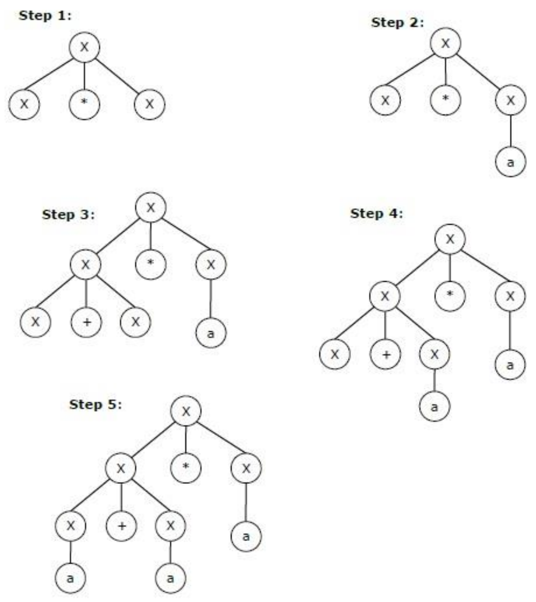
→ X*a

→ X+X*a

→ X+a*a

→ a+a*a

The stepwise derivation of the above string is shown as below in the figure,



2) Write a CFG for the language,

L = {wcw^r | w ϵ (a, b)*}

Therefore,

Let G be a CFG for the language L. G = (Vn, Vt, P, S)

Here, Vn = {S}

Vt = {a, b, c}

And P is given by

S → aSa

S → bSb

S → c

Let us check that abbcbba can be derived from the given CFG.

S =>aSa (use the production S → aSa)

=> abSba (use S → bSb)

=> abbSbba (use S →bSb)

=> abbcbba (use S→c)

So string abbcbba can be derived from given CFG.

B. Related Work & Other Aspects

➢ Probabilistic Context Free Grammars (PCFG)

A Probabilistic Context Free Grammar is comprised of

• a CFG G

• a weighting parameter, α p ⟶ β,∀α ∈ N,β ∈ (N ∪ Σ)*

Three major properties of PCFG, suggested by probability theory, are as follows:

1) for a particular nonterminal the sum of the weight of each input must be as large as 1. That is to say,

$$\sum_{\alpha\to\beta\epsilon R,\alpha\in N,\beta\in(N\cup\Sigma)^*} (\alpha \xrightarrow{p} \beta) = 1$$

2) The defined probability for any output must be between 0 and 1,that is $0 \leq \alpha \xrightarrow{p} \beta \leq 1.$

3) given the production rules $\alpha_1 \xrightarrow{p_1} \beta_1,\ \alpha_2 \xrightarrow{p_2} \beta_2,$ the probability of deriving a particularstring w from S is given as:

$$P(w) = \prod_{k=1} \alpha_k \xrightarrow{p} \beta_k$$

PCFGs have defined applications within a variety of domains.

Here are some fields which have benefited from this concept and applications of the same**:-**

Random String Generator Various developers have developed and are available on the internet a variety of random name generators; such as: Behind the Name6, Random Name Generator7, and Fantasy Name Generator8. Such systems are based solely on random numbers and do not produce complete or practical social profiles and none of these systems are powered by systematic / algorithmic synthesis of any kind.

Formal Grammar Applications Is a CFG-based tool that takes raw, instant financial messages and attempts to understand and create semantic summaries from them. For the development of compilers, CFGs often find huge applications. PCFGs are used in the simulation of natural language and RNA (Ribonucleic Acid).

Pii Detection in Electronic Documents Aura Ea provided a method for the automated identification of PII in documents without the use of metadata, demonstrating that the document authouring mechanism consists of other processes that embed PII in documents through other artifacts. Raghavan Ea proposed a novel approach based on PCFGs

➢ Latent Context Free Grammar

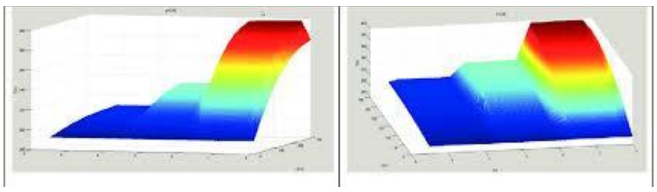
The setting free property permits the language structure to catch long haul recursions of settled expressions in an image succession. CFG is an augmentation to customary language structure, which is identical to HMMs. Stochastic without context punctuation (SCFG) acquaints likelihood appropriations with every creation rules, and it has been effectively utilized in normal language preparing and biomedical investigation. Right now, will utilize setting free principles to embody the games space information and rationales. Besides, so as to process realvalued video perceptions, we propose inert setting free sentence structure (LCFG) as the augmentation of the general SCFG with constant discriminative terminal images.

$$\sum_{\substack{(\alpha\to\beta)\in\mathbf{R}\\ \alpha=X}} p(\alpha\to\beta) = 1$$

• Iteration. For i = 1 to L−1, j = i+1 to L,and v ∈ N

α (i,j,v) = max (y,z) (j−1) max (k=i)

α(i,k,y) + α(k + 1,j,z) + logp(v ➝ y)



REFERENCES

[1] Abejide Ade-Ibijola, “Synthesis of Social Media Profiles Using a Probabilistic Context-Free Grammar”, Formal Structures, Algorithms and Industrial Applications Research Cluster, University of Johannesburg.

[2] Xingzhong Xu, Hong Man, “Interpreting Sports Tactic Based on Latent-Context Free Grammar”, Stevens Institute of Technology, Castle Point on Hudson, Hoboken, NJ,.

[3] John C. Kieffer, En-hui Yangi K. Elissa, “Design of Context-Free Grammars for Lossless Data Compression” , Dept. of Electrical &c

Computer Engineering, University of hfinnesota, Minneapolis, Dept. of Electrical & Computer Engineering, University of Waterloo, Waterloo, Ontario.

[4] Anna Corazza and Giorgio Satta, “Probabilistic Context-Free Grammars Estimated from Infinite Distribution”, Member, IEEE Computer Society

[5]

[6] M. Young, The Technical Writer’s Handbook. Mill Valley, CA: University Science, 1989.

“