Simulating Derivations of Context-Free Grammar

KULDEEP VAYADANDE (kuldeep.vayadande@gmail.com)
VIT, PUNE

Research Article

Keywords: Context-Free Grammar, Leftmost Derivation, Parse Tree, Python, Rightmost Derivation

Posted Date: February 28th, 2022

DOI: https://doi.org/10.21203/rs.3.rs-1395103/v1

Abstract

As opposed to automata, grammars are used to generate strings instead of identifying them. The use of regular languages and finite automata is simple and restrictive. Context-Free Grammar or CFG is a formal grammar used to produce all possible combinations of strings in a given formal language. Context-Free Grammar consists of a set of grammar rules which are finite and predetermined. In computer science, a CFG is said to be ambiguous if a given string could be generated by the grammar in more than one way. Syntax of high-level programming languages, Parser programs and compiler design can be described using Context-Free Grammar. This paper presents a method to implement the derivations of Context-Free Grammar using Python. By applying an appropriate production rule to the leftmost non-terminal in each step, a leftmost derivation is obtained. On the contrary, by applying the appropriate production rule to the rightmost non-terminal in each step, a rightmost derivation is obtained.

I. Introduction

According to Chomsky Hierarchy, Context-Free Grammar belongs to grammar type-2. A CFG is quadruple, G = (V, T, P, S). G is the grammar used to generate the string of a language by using a finite collection of production rules. T is the final collection of terminal symbols which are denoted by lowercase letters. V is the final collection of non-terminal symbols which are denoted by uppercase letters. P is a finite collection of production rules. These rules are used for interchanging the non-terminals symbols in a string present on the left side of the production with other terminal or nonterminal symbols present on the right side of the production. S is considered to be the start symbol starting from which a string is derived. A Non-terminal is substituted repeatedly on the right-hand of the production to derive a string until all the non-terminals have been terminal symbols.

Programming languages in which a string can be generated in more than one way are said to have ambiguous grammar. In such a case, to select the intended parse tree of an ambiguous construct, semantic information is required. A derivation or a parse tree is used to represent the semantic information of a string which is generated from a Context-Free Grammar.

A derivation tree has a node that consists of a variable present on the left of the production rule while the child nodes consist of variables present on the right of the same production rule. A derivation tree illustrates how each variable is substituted in the derivation from the root identified with the start symbol, to the leaves or the terminals [1].

At any stage during a parse, on deriving a form that is not yet a sentence (sentential form), we have two choices to make:

a. Determine which non-terminal in the sentential form to which a production rule must be applied to.

b. Or which production rule for that non-terminal to apply

The first decision here is relatively easy. The input string is scanned from left to right in order to efficiently derive the leftmost terminal of the output sentence. Thus, the process of choosing the leftmost non-terminal occurring in the sentential form and applying a production rule is performed. This type of parsing technique is called the top-down parse and is also known as the leftmost derivation [2]. The top-down parsing technique starts the evaluation at the root node of the tree and proceeds systematically down the tree until it reaches the leaves and the given string is generated.

The situation is reversed in the bottom-up parsing technique, and the production rules are applied in the reverse order to the symbols present on the left. The Bottom-up approach starts the evaluation directly from the leaves i.e., the lowest level of the tree and proceeds upwards systematically for parsing the node.

Ii. Literature Review

String moment properties and the lengths of derivation of stochastic CFGs are investigated in this paper. The issue with calculating the moments of the string length is demonstrated to be identical to the problem of calculating the moments of the derivation length. The paper shows that moments exist and are finite in nature, and it demonstrates that the finiteness of the subsequent moments if the existing initial moment is finite[3][8].

The derivation complexity function on G for each grammar G in d(F) is defined. It demonstrates that any constant factor n can always speed up derivations, in the sense that an corresponding grammar G' in (F) may be found for each positive integer n [4][7].

A CFG is used with Z notation in this paper to aid in the corroboration of a compiler component. The authors have defined the grammar, and then presented the technique for language derivation applying the leftmost derivations. As part of the parsing analysis, the ambiguity of a language is examined [5][9].

Component-wise derivations are introduced, as well as the first implementation of Chomsky-Schützenberger parsing. It makes use of a variety of context-free grammar and involves numerous enhancements in order to attain feasibility. Finally, the results are compared to those of current grammar-based parsers [6].

Iii. Methodology

The project proposes a method to simulate the derivations of Context-Free Grammars. The project is programmed in Python and the WebApp is made with the help of Streamlit which is a framework popularly used to create quick, simple apps using Python.

A. Proposed System

In the system proposed in the paper, a Production Rule is taken from the user as a string input. The user is then provided can enter another Production by increasing the ‘Number of Productions’ field by clicking the ‘+’ button. The production rules entered by the user are then saved to a text file – ‘ProductionRules.txt’ using the concepts of file handling in Python. Next, the user is prompted to enter a string to check whether it forms a part of the language generated by the productions of the Context-Free Grammar. If the language can be generated using the production rules provided by the user, the derivation is displayed on the WebApp. Else, the program prints ‘The string does not exist’ and asks the user to ‘Try Again’.

B. Flowchart

Figure 1. describes a flowchart that provides a precise overview of the working of the proposed method to derive the leftmost(LMD) and rightmost(RMD) derivations of a string using the given productions.

C. Algorithm

Step 1:- Inputting Productions

Productions are rules that have the form, Variable -> String of Variables and Terminals

Consider S to be the start symbol, then

S -> aSb

S -> a

can be said to be two productions for the Context-Free Grammar. The input entered by the user in the program follows the same format and is accepted as a string. This value is stored in the variable *‘rule’*. This ‘*rule*’ is then stored into a text file of the name ‘ProductionRules.txt’

Adding another production can be done by pressing ‘+’ button next to the ‘Number of Productions’ field.

Step 2:- Saving the productions to a dictionary of list of tuples

After all the productions have been entered, the program reads the productions from the file ‘ProductionRules.txt’ line by line and splits it into two parts: Variable and string of variables and terminals.

For the production rules S -> aSb and S-> a, a dictionary of the form {'S': [('a', 'S', 'b'), ('a',)]} is created.

Step 3:- Inputting a string from the user

A string is inputted from the user to check whether the string forms a part of the language or not. The string inputted by the user is stored in the variable ‘w’. After which each character of the string is separated by a single white space.

Step 4:- Checking if the string is part of Grammar

To obtain the Leftmost derivation, we call the function ‘gen_random_left()’ and pass ‘S’ as a parameter. This function is responsible for generating a random sentence starting with the given symbol. Since, we have assumed ‘S’ to be our starting symbol, we have passed ‘S’ as a parameter to the function. The variable ‘rand_prod’ will store a random production starting with the given symbol from the dictionary ‘prod’. The function ‘gen_random_left()’ calls another function ‘replace_left()’ which will replace the non-terminals by the productions one at a time. The function ‘gen_random_left()’ calls itself recursively till the string is generated. The string which is stored in the variable ‘sentence’ is then returned to the main function.

To obtain the Rightmost derivation, we call the function ‘gen_random_right()’ and pass ‘S’ as a parameter. Here, the number of capital alphabets is calculated ‘cap’ and depending on the value, the ‘rule’ is passed to ‘replace_right()’. If ‘cap’>1, then rule is reversed and passed to ‘replace_right()’, else it is passed as it is. ‘replace_right()’ which will replace the non-terminals by the productions till the given string one at a time. The string which is stored in the variable ‘sentence’ is then returned to the main function. In the main() function, this process is repeated multiple times so as to cover all possible combinations. The returned string is compared to the string inputted by the user. If the string belongs to the language, the derivation is printed. Else, the program prints ‘Try Again’ as the output.

Iv. Results And Discussion

The developed system thus is able to accept productions from the user by adding the number of productions by clicking the ‘+’ button. This is demonstrated in the Fig. 2.

Next, the user is prompted to enter the test string to obtain its leftmost or rightmost derivations if the string exists in the grammar.

It also able to correctly identify whether the input string is part of the language or not. generates the correct derivation if the string is generated by the Context-Free Grammar. If the user enters the string ‘aabb’ which exists in the grammar inputted, the following Leftmost and Rightmost Derivation is obtained.

The result obtained after pressing the ‘Submit’ button is as follows:

Leftmost derivation:

S = > AB = > aaB = > aabb = > aabb

Rightmost derivation:

S = > AB = > Abb = > aabb = > aabb

If the input string is not accepted by the Context-Free Grammar, the program prints the message 'The string doesn't exist in the grammar' and asks the user to 'Try Again'.

V. Limitations

a. If too many production rules are entered, the derivation may or may not be accurately derived.
b. The project only includes the checking of strings containing English Alphabets and ^, use of operators in expressions do not yield the correct output.

Vi. Conclusion

Thus, in this paper we propose a system that is able to generate derivations for an input string using productions given by the user and identify whether the string is part of a language. It successfully simulates the derivations of CFGs.

Vii. Future Scope

CFGs are an essential part of Natural Language Processing. This study can be useful to explore the applications of CFG in Grammar Checkers, Information Extraction and Translations. The system could be made to accept more complex strings or expressions in the future.

References

1. P. Linz, *"An introduction to Formal Languages and Automata"*, Jones and Bartlett Learning, 5th edition.
2. J. Power, "*Notes on Formal Language Theory and Parsing*", (2002).
3. S E. Hutchins, "Moments of string and derivation lengths of stochastic context-free grammars." Information Sciences 4.2 (1972): 179–191.
4. Ginsburg, Seymour, and N. Lynch. "Derivation complexity in context-free grammar forms." SIAM Journal on Computing 6.1 (1977): 123–138.
5. K. A. Buragga, and N.A. Zafar. "Formal Parsing Analysis of Context-Free Grammar Using Left Most Derivations." *International Conference on Software Engineering Advances*. 2011.
6. T. Ruprecht, and T. Denkinger. "Implementation of a Chomsky-Schützenberger n-best parser for weighted multiple context-free grammars." *Proceedings of the* 2019 *Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. 2019.
7. Raza, Mir Adil, Kuldeep Baban Vayadande, and H. D. Preetham. *"DJANGO MANAGEMENT OF MEDICAL STORE."*, International Research Journal of Modernization in Engineering Technology and Science, Volume:02/Issue:11/November – 2020
8. K.B. Vayadande, Nikhil D. Karande," *Automatic Detection and Correction of Software Faults: A Review Paper*", International Journal for Research in Applied Science & Engineering Technology (IJRASET)

ISSN: 2321–9653, Volume 8 Issue IV Apr 2020.

9. Kuldeep Vayadande, Ritesh Pokarne, Mahalaxmi Phaldesai, Tanushri Bhuruk, Tanmai Patil, Prachi Kumar, "*SIMULATION OF CONWAY'S GAME OF LIFE USING CELLULAR AUTOMATA"* International Research Journal of Engineering and Technology (IRJET), Volume: 09 Issue: 01 | Jan 2022, e-ISSN: 2395-0056, p-ISSN: 2395-0072
10. Varad Ingale, Kuldeep Vayadande, Vivek Verma, Abhishek Yeole, Sahil Zawar, Zoya Jamadar. <bi>Lexical analyzer using DFA</bi>, International Journal of Advance Research, Ideas and Innovations in Technology, www.IJARIIT.com.

Figures

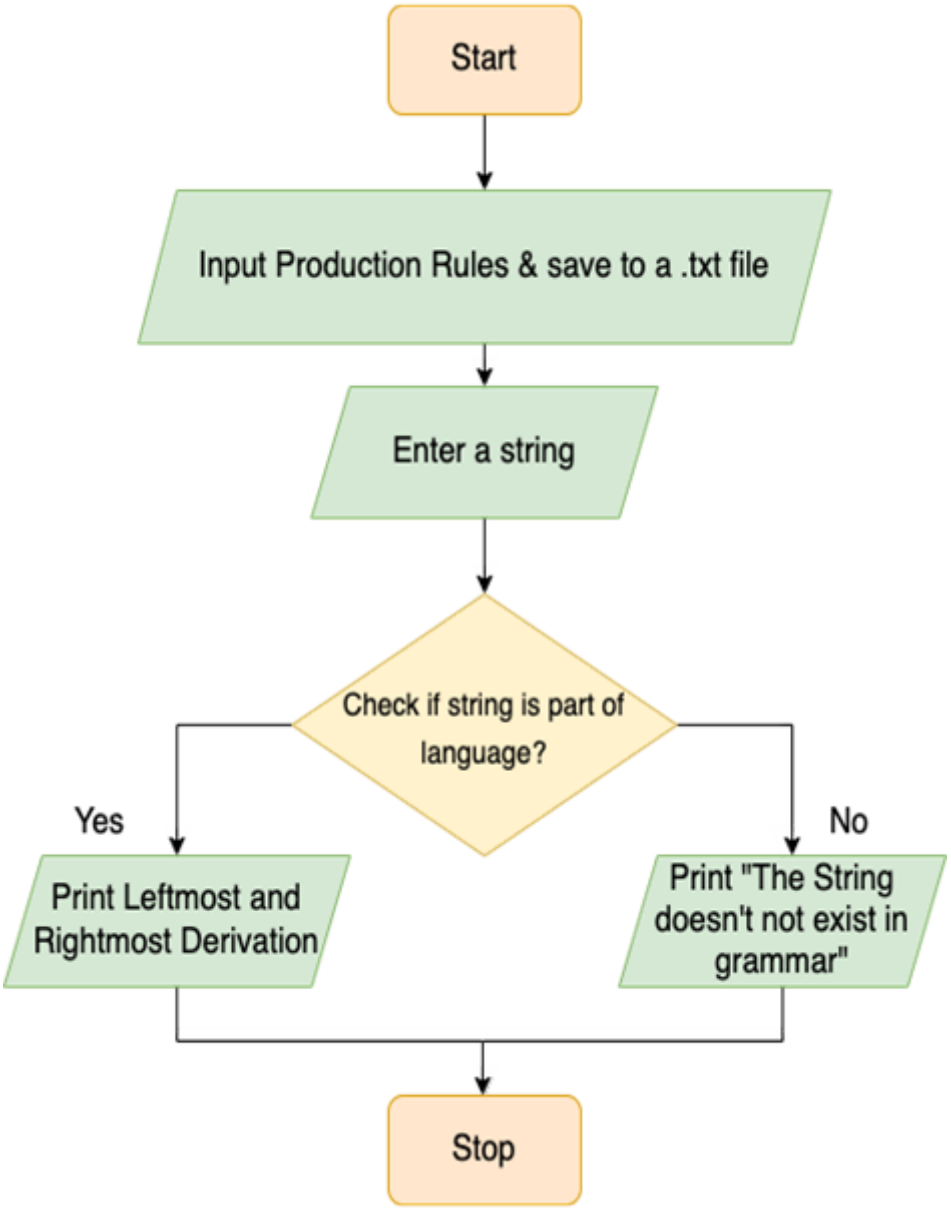

Figure 1

Flowchart of the model

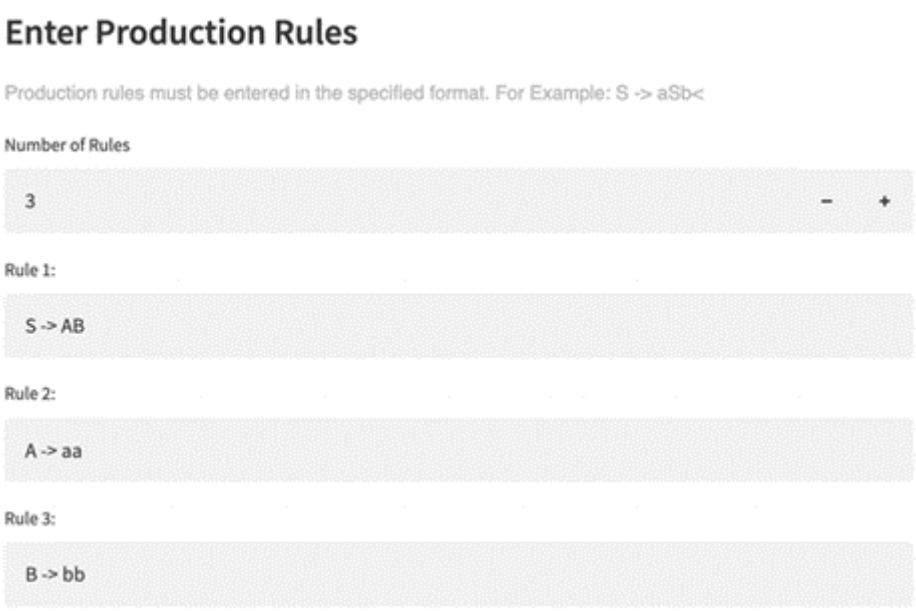

Figure 2

Inputting the Productions