Table of Contents

Ambiguous grammar:
There are two ways to draw a parse tree:
1. Top-down Approach:
2. Bottom-up Approach:
Leftmost and Rightmost Derivation of a String:
1.Leftmost derivation:
2. Rightmost derivation:
Related posts:

A parse tree is a diagrammatic representation of the parsed structure of a sentence or string.

In the syntax analysis phase, a compiler verifies whether or not the tokens generated by the lexical analyzer are grouped according to the syntactic rules of the language. This is done by parse tree.

Than from parse tree intermediate code can be generated.

A parse tree for a grammar G is a tree where

- The root is the start symbol for G
- The interior nodes are the nonterminals of G
- The leaf nodes are the terminal symbols of G.
- The children of a node T (from left to right) correspond to the symbols on the right hand side of some production for T in G.

Ambiguous grammar:

A grammar that produces more than one parse tree for some sentence is said to be ambiguous.

There are two ways to draw a parse tree:

1. Top-down Approach:

- Starts with the starting symbol S
- Goes down to tree leaves

2. Bottom-up Approach:

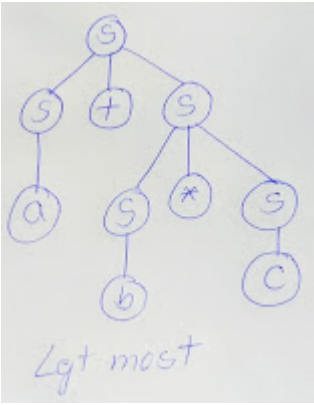
- Starts from tree leaves
- Proceeds upward to the starting symbol S

Leftmost and Rightmost Derivation of a String:

1.Leftmost derivation:

A leftmost derivation is obtained by applying production to the leftmost variable in each step.

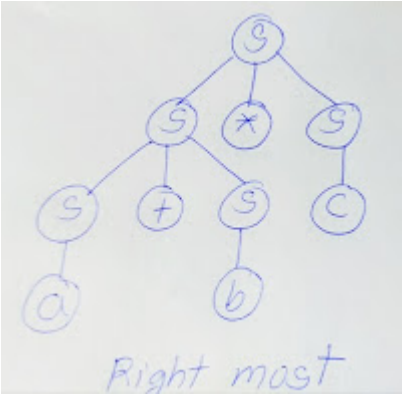
For example, S –> a+b*c



2. Rightmost derivation:

A rightmost derivation is obtained by applying production to the rightmost variable in each step.

For example, S –> a+b*c



Related posts:

1. Sequence Control & Expression | PPL
2. PPL:Named Constants
3. Parse Tree | PPL | Prof. Jayesh Umre
4. Basic elements of Prolog
5. Loops | PPL | Prof. Jayesh Umre
6. Subprograms Parameter passing methods | PPL | Prof. Jayesh Umre
7. Programming Paradigms | PPL | Prof. Jayesh Umre
8. Subprograms Introduction | PPL | Prof. Jayesh Umre
9. Phases of Compiler | PPL | Prof. Jayesh Umre
10. Influences on Language design | PPL | Prof. Jayesh Umre
11. Fundamentals of Subprograms | PPL | Prof. Jayesh Umre
12. Programming Paradigm

13. Influences on Language Design
14. Language Evaluation Criteria
15. OOP in C++ | PPL
16. OOP in C# | PPL
17. OOP in Java | PPL
18. PPL: Abstraction & Encapsulation
19. PPL: Semaphores
20. PPL: Introduction to 4GL
21. PPL: Variable Initialization
22. PPL: Conditional Statements
23. PPL: Array
24. PPL: Strong Typing
25. PPL: Coroutines
26. PPL: Exception Handler in C++
27. PPL: OOP in PHP
28. PPL: Character Data Type
29. PPL: Exceptions
30. PPL: Heap based storage management
31. PPL: Primitive Data Type
32. PPL: Data types
33. Programming Environments | PPL
34. Virtual Machine | PPL
35. PPL: Local referencing environments
36. Generic Subprograms
37. Local referencing environments | PPL | Prof. Jayesh Umre
38. Generic Subprograms | PPL | Prof. Jayesh Umre
39. PPL: Java Threads

40. PPL: Loops
41. PPL: Exception Handling
42. PPL: C# Threads
43. Pointer & Reference Type | PPL
44. Scope and lifetime of variable
45. Design issues for functions
46. Parameter passing methods
47. Fundamentals of sub-programs
48. Subprograms
49. Design issues of subprogram
50. Garbage Collection
51. Issues in Language Translation
52. PPL Previous years solved papers
53. Type Checking | PPL | Prof. Jayesh Umre
54. PPL RGPV May 2018 solved paper discussion| Prof. Jayesh Umre
55. PPL Viva Voce
56. PPL RGPV June 2017 Solved paper | Prof. Jayesh Umre
57. Concurrency
58. Basic elements of Prolog
59. Introduction and overview of Logic programming
60. Application of Logic programming
61. PPL: Influences on Language Design
62. Language Evaluation Criteria PPL
63. PPL: Sequence Control & Expression
64. PPL: Programming Environments
65. PPL: Virtual Machine
66. PPL: Programming Paradigm

67. PPL: Pointer & Reference Type
68. try-catch block in C++