A parse tree is like a family tree for a sentence in a programming language or any other context-free language. It shows how the sentence can be broken down into smaller parts according to the rules of a grammar.

Here's a simpler breakdown of the conditions for constructing a parse tree from a context-free grammar (CFG):

1. Every part of the tree needs a label: Just like every person in a family tree has a name, every part of a parse tree needs to be labeled. The label can be either a word (like "noun" or "verb") or a symbol from the grammar (like "S" for the start symbol).
2. The start symbol goes at the top: Just as a family tree usually starts with the oldest generation at the top, a parse tree starts with the start symbol of the grammar at the top.
3. Inside parts of the tree are non-terminals: If you look inside a family tree, you see people's names, not just "mom" or "dad". Similarly, the inside parts of a parse tree are labeled with non-terminal symbols, which represent groups of words or phrases according to the grammar.
4. Each production rule forms a branch: Think of a production rule in the grammar as a recipe for making a certain part of a sentence. When constructing the parse tree, each production rule corresponds to a branch where the parent node is labeled with the non-terminal symbol being replaced, and the children nodes are labeled with the symbols or words that it's replaced with.
5. Terminals or null symbols are at the ends: Just like leaves on a family tree are the individual people, the leaves of a parse tree are the individual words or symbols from the input sentence, or sometimes they can be empty (represented by ε), depending on the grammar.

Define parse tree. What are the conditions for constructing a parse tree from a CFG ?

Related posts:

1. What are the types of passes in compiler ?
2. Discuss the role of compiler writing tools. Describe various compiler writing tools.
3. What do you mean by regular expression ? Write the formal recursive definition of a regular expression.
4. How does finite automata useful for lexical analysis ?
5. Explain the implementation of lexical analyzer.
6. Write short notes on lexical analyzer generator.
7. Explain the automatic generation of lexical analyzer.
8. Explain the term token, lexeme and pattern.
9. What are the various LEX actions that are used in LEX programming ?
10. Describe grammar.
11. Explain formal grammar and its application to syntax analyzer.
12. Describe the capabilities of CFG.
13. What is parser ? Write the role of parser. What are the most popular parsing techniques ? OR Explain about basic parsing techniques. What is top-down parsing ? Explain in detail.
14. What are the common conflicts that can be encountered in shift-reduce parser ?
15. Differentiate between top-down and bottom-up parser.Under which conditions predictive parsing can be constructed for a grammar ?
16. Differentiate between recursive descent parsing and predictive parsing.
17. What is the difference between S-attributed and L-attributed definitions ?
18. What is intermediate code generation and discuss benefits of intermediate code ?
19. Define parse tree. Why parse tree construction is only possible for CFG ?
20. Discuss symbol table with its capabilities ?
21. What are the symbol table requirements ? What are the demerits in the uniform

Define parse tree. What are the conditions for constructing a parse tree from a CFG ?

structure of symbol table ?