Assignment 01 Student 6 Answers

(Theory**): What are terminals and non-terminals in a grammar?**

In the context of formal grammars, particularly context-free grammars (CFGs) used in compiler design for syntax analysis, terminals and non-terminals are fundamental components that define the structure and elements of the language being described.

- **Non-terminals**: These are abstract symbols or variables that represent syntactic categories or phrases in the language. They can be expanded or replaced by applying production rules, which allow them to derive sequences of other symbols (including terminals and other non-terminals). Non-terminals are typically denoted by uppercase letters (e.g., S, E, T, A) or descriptive names (e.g., Expr, Stmt). They serve as placeholders for parts of the sentence structure and are essential for recursion and hierarchical organization in the grammar. The start symbol (usually S) is always a non-terminal, from which all valid strings in the language are derived. Non-terminals do not appear in the final derived string; they are fully replaced during the derivation process.
- **Terminals**: These are the atomic, indivisible symbols that form the actual content of the strings in the language. They cannot be further expanded or replaced by production rules—they are the "leaves" or endpoints of any derivation. Terminals represent the lexicon of the language, such as keywords, operators, literals, or punctuation. They are often denoted by lowercase letters (e.g., a, b, c), symbols (e.g., +, *, ()), or tokens like "id" for identifiers or "num" for numbers. In a parse tree, terminals appear at the leaf nodes and are what the lexical analyzer (scanner) produces from the source code.

To illustrate the difference:

- Consider a simple CFG for balanced parentheses: S → (S) | ε Here, S is the non-terminal (it can be replaced by "(S)" or the empty string ε). The terminals are "(" and ")" (they are the basic symbols that appear in the final strings, like "()", "(())"). Derivation example: S → (S) → (ε) = "()" Non-terminal S is replaced until only terminals remain.
- Another example from arithmetic expressions: E → E + T | T T → id Non-terminals: E (expression), T (term). Terminals: +, id (where "id" represents identifiers like variable names). This allows deriving strings like "x + y", where "x" and "y" are instances of the terminal "id".

The distinction is crucial in syntax analysis: Non-terminals enable recursive definitions for complex structures (e.g., nested expressions), while terminals ground the grammar in concrete input symbols. In parsing, the goal is to derive the input string (composed solely of terminals) from the start non-terminal, constructing a parse tree that shows the hierarchical application of rules. This aligns with compiler phases, where lexical analysis identifies terminals, and syntax analysis uses non-terminals to validate structure. Issues like ambiguity

or recursion often involve how non-terminals are defined, and transformations (e.g., removing left recursion) focus on non-terminals to make the grammar suitable for parsers like LL or LR.