**CC\_Assignment#2**

**22i-0846\_22i-1036**

**LL(1) Parsing Table Generator Report**

**Introduction** This report provides an overview of the approach used to generate an LL(1) parsing table for a given context-free grammar (CFG). The process involves transforming the CFG to remove left recursion, performing left factoring, computing FIRST and FOLLOW sets, and constructing the LL(1) parsing table. The implementation is verified through test cases and manual validation.

**Approach** The program follows these key steps:

1. **Reading the CFG:** The grammar rules are stored in a text file and read into data structures that represent production rules in a structured format. Each rule consists of a non-terminal on the left-hand side and a set of possible right-hand side expansions.
2. **Left Factoring**: If multiple productions share a common prefix, they are refactored by introducing a new non-terminal to remove ambiguity. This ensures that the parser can make a single lookahead decision**.**
3. **Left Recursion Elimination**: Direct left recursion occurs when a non-terminal refers to itself at the beginning of a production. Indirect left recursion happens when recursion appears through intermediate non-terminals. Both types are eliminated by restructuring rules into right-recursive form, ensuring they conform to LL(1) constraints.
4. **Computing FIRST Sets:** The FIRST set of a non-terminal consists of the first terminal symbols that can appear in strings derived from that non-terminal. If a production starts with another non-terminal, its FIRST set is recursively included. If epsilon (ε) is in a FIRST set, the next symbols in the production are also considered.
5. **Computing FOLLOW Sets:** The FOLLOW set contains symbols that can appear immediately after a non-terminal in any derivation. The start symbol always includes the end-of-input marker ($). FOLLOW sets are determined using FIRST sets and applying rules to track sequences of symbols across production rules.
6. **Generating LL(1) Parsing Table:** The LL(1) table is created by mapping each non-terminal and terminal combination to a production rule. If a terminal is in the FIRST set of a production, that rule is added to the table. If epsilon (ε) is in the FIRST set, FOLLOW sets help determine valid transitions. The resulting table enables predictive parsing.

**Challenges Faced**

* Managing indirect left recursion efficiently.
* Handling epsilon (ε) transitions in FIRST and FOLLOW sets correctly.
* Ensuring that the parsing table remains conflict-free for LL(1) grammar.

**Verification and Testing**

* The computed FIRST and FOLLOW sets were manually verified against expected values.
* The parsing table was tested with different grammars to check for correctness.
* Sample parsing traces were followed to ensure the table guides the parser correctly.

**Conclusion** The implementation successfully transforms a given CFG into an LL(1) parsing table by systematically computing required sets and resolving conflicts. The approach is useful in compiler design and syntax analysis, providing a structured method for predictive parsing.