National University
of computer and emerging sciences

Assignment Report

CS-4031 Compiler Construction

Assignment 02

Arshman Khawar (22I-2427)

Rehan Tariq (22I-0965)

CS-6A

Department of Computer Science BS(CS)

FAST-NUCES Islamabad

1. Introduction

This report provides an overview of the approach taken to implement the Context-Free Grammar (CFG) processor as required by the assignment. The program was developed in C and performs left factoring, left recursion removal, FIRST and FOLLOW set computation, and LL(1) parsing table construction.

2. Approach

2.1 Reading the Grammar

The program reads the CFG from an input file (`grammar.txt`), where each production is specified in a simple format (e.g., `A -> A a`). The grammar is stored in a structured format using arrays and string operations.

2.2 Left Factoring

Left factoring was implemented by identifying common prefixes among productions and introducing new non-terminals to factor out the shared parts. For example:

S -> A B C | A C

was transformed into:

S -> A S'

S' -> B C | C

This ensures that the grammar remains suitable for predictive parsing.

2.3 Left Recursion Removal

Left recursion was eliminated to make the grammar suitable for top-down parsing. The recursive productions were rewritten using a new non-terminal to remove direct left recursion. Example transformation:

A -> A a

was rewritten as:

A -> a A'

A' -> a A' | ε

This prevents infinite recursion when parsing.

2.4 FIRST Set Computation

The FIRST set for each non-terminal was computed iteratively by analyzing the starting symbols of its right-hand side (RHS) alternatives. If the RHS began with a terminal, it was directly added to the FIRST set. If it started with a non-terminal, its FIRST set was propagated. Example result:

FIRST(A) = { a }

FIRST(B) = { b }

FIRST(C) = { c }

2.5 FOLLOW Set Computation

The FOLLOW sets were computed using the FIRST sets and considering positions where non-terminals appeared within productions. The FOLLOW set of the start symbol was initialized with `$`. Example result:

FOLLOW(A) = { b , c }

FOLLOW(B) = { c }

2.6 LL(1) Parsing Table Construction

Using the computed FIRST and FOLLOW sets, an LL(1) parsing table was built. Each table entry corresponds to a non-terminal and a terminal, determining which production should be used during parsing.

3. Challenges Faced

1. **Handling Left Recursion Cases:** Special care was needed for cases where all alternatives were left-recursive.
2. **Efficient Parsing Table Construction:** Ensuring correct placement of entries required careful tracking of FIRST and FOLLOW set propagation.
3. **String Manipulation in C:** Managing string-based grammar rules efficiently without memory leaks was a key challenge.

4. Verification of Correctness

1. **Stepwise Output Analysis:** Each transformation stage was printed and compared against expected results.
2. **Manual Checking of FIRST & FOLLOW Sets:** The computed sets were verified against theoretical calculations.
3. **Parsing Table Validation:** Conflicts were checked, and the final parsing table was inspected for correctness.

5. Conclusion

The implemented C program successfully performs left factoring, left recursion removal, FIRST and FOLLOW set computation, and LL(1) parsing table construction. The correctness of results was verified through stepwise outputs and manual validation.