LL(1) Parser Implementation Report

Approach

In this assignment, we extended the LL(1) parser from Assignment 2 to include a parsing stack to parse input strings. The key components of my solution include:

1. **Stack Implementation**: We implemented a stack data structure to store symbols during parsing. The stack is initialized with the start symbol and $ (end marker).

2. **Input Processing**: The parser reads strings from an input file, where each line contains space-separated terminals to be parsed.

3. **Parsing Algorithm**: For each input string:

    - Initialize stack with $ and the grammar's start symbol
    - For each step in the parsing process:
        - If top of stack is a terminal, match it with current input
        - If top of stack is a non-terminal, use the parsing table to expand it
        - Handle errors when no production rule is applicable
4. **Error Handling**: The parser detects syntax errors such as:

    - Missing entries in the parsing table
    - Mismatched terminals
    - Invalid symbols
5. When an error is encountered, the parser records it, attempts recovery (by popping or skipping symbols), and continues parsing.

6. **Output Generation**: The parser produces a detailed report showing:

    - Step-by-step parsing process with stack contents
    - Current input and action taken at each step
    - Error messages when applicable
    - Success/failure status after parsing each line
    - Summary of total errors encountered

Challenges Faced

1. **Error Recovery**: Implementing error recovery strategies that would allow parsing to continue after encountering an error was challenging, we opted for a simple approach of

popping the expected terminal or skipping the current token.

2. **Stack Management**: Ensuring that the stack operations were correctly implemented, especially when dealing with epsilon productions.

3. **Reporting**: Creating a clear, informative report that shows exactly what's happening at each step of the parsing process.

Verification of Correctness

To verify the correctness of my implementation, we tested it with several input strings:

1. **Valid strings** according to the grammar:
    - Strings that should be accepted (e.g., "a a a a", "a a c")
2. **Invalid strings** to test error handling:
    - Strings with invalid symbols (e.g., "invalid input")
    - Strings with syntax errors (e.g., wrong sequence of terminals)

The parser correctly identifies valid strings and reports errors for invalid ones. The parsing steps clearly show how the stack evolves and how productions are applied.

For manual verification:

1. Trace through the parsing steps for a simple valid string.
2. Verify that the stack operations match what would be expected for an LL(1) parser.
3. Check that error messages are accurate and helpful.

Conclusion

The LL(1) parser with stack implementation successfully parses input strings according to the grammar defined in Assignment 2. It provides detailed output showing each step of the parsing process and properly handles syntax errors. The implementation maintains the correct behavior of an LL(1) parser while adding user-friendly features like detailed step-by-step reporting.