LL(1) Parser

- ## Introduction

In **compiler design**, **syntax analysis** plays a crucial role in processing the structure of programming languages. An LL(1) parser is a predictive, top-down parser that uses a one-symbol lookahead to choose the production rule during parsing. This parser is efficient and commonly used in many compiler implementations.

- ## Purpose

The purpose of the LL(1) parser is to recognize whether the input program can be generated by a given grammar. It aims to parse the input efficiently and determine if it follows the syntax defined by the grammar without having to backtrack during parsing.

* Algorithm

1. Construct the ***LL(1) parsing table*** that contains entries for each combination of non-terminal symbol and lookahead terminal symbol.

2. Use the parsing table to guide the parser's decisions during parsing.

3. Initialize the parsing stack with the start symbol of the grammar and the input token stream.

4. Repeat until the parsing stack is empty:

a. Read the current input token.

b. If the top of the stack is a terminal symbol matching the input token, pop the stack.

c. If the top of the stack is a non-terminal symbol, consult the parsing table to determine the production rule to apply.

d. Replace the non-terminal symbol on top of the stack with the right-hand side of the chosen production rule.

- **Example**

Consider the grammar:

**```**

S → AaB | c

A → d

B → e | ε

**```**

Construct the LL(1) parsing table based on *FIRST* and *FOLLOW* sets, and use it to parse an input token stream.

Parsing table entries:

- (S, a) -> S → AaB

- (S, c) -> S → c

- (A, d) -> A → d

- (B, e) -> B → e

- (B, $) -> B → ε

- ## Conclusion

In conclusion, the LL(1) parser is a fundamental component of syntax analysis in compilers. By utilizing a predictive approach and a one-symbol lookahead, this parser can efficiently process programming language syntax and detect syntax errors in the input code. Implementing the LL(1) parser requires careful construction of the parsing table and adherence to the grammar's rules.

Producer: Elham Jafari

Computer Engineering