SLR(1) Parser

- ## Introduction

Within compiler design, syntax analysis plays a vital role in processing programming languages. An SLR(1) parser, which stands for Simple LR(1) parser, is a type of bottom-up parser that uses one token lookahead to shift or reduce symbols efficiently based on a parsing table. This parser is capable of handling a broad class of context-free grammars while maintaining simplicity in the parsing process.

- ## Purpose

The main goal of an SLR(1) parser is to recognize the structure of the input program and determine its adherence to the specified grammar rules. By employing a parsing table that incorporates lookahead information, the parser can make informed decisions on shifting or reducing symbols to construct a parse tree for the input program.

* Algorithm

1. **Augment the grammar rules with special start and end symbols to facilitate parsing.**

2. **Construct *LR(1)* items to represent the augmented grammar rules with a dot to mark the current position in the production and the lookahead token.**

3. **Build the canonical collection of *LR(1)* items by applying closure and goto operations to determine parser states based on *LR(1)* items.**

4. **Generate the *SLR(1) parsing table* by populating entries for shift, reduce, and goto actions using the parser states and transitions.**

5. **Initialize the parsing stack with the start state, process the input token stream while considering lookahead tokens, and apply appropriate actions based on the parsing table entries.**

- **Example**

Consider the augmented grammar:

S' → S

S → AA

A → a | b

Construct the *LR(1) items*, build the canonical collection of *LR(1)* items, and create the *SLR(1) parsing table* for the provided grammar with lookahead tokens.

Parsing table entries:

- (0, a) -> Shift 2

- (0, b) -> Shift 3

- (1, $) -> Accept

- (2, a) -> Reduce A → a

- (2, b) -> Reduce A → b

- (3, a) -> Shift 4

- (3, b) -> Shift 5

- (4, $) -> Reduce A → a

- (5, $) -> Reduce A → b

- ## Conclusion

The *SLR(1) parser* serves as a valuable tool for *syntax analysis* in compilers, providing a balance between parsing power and simplicity. By incorporating lookahead information into the parsing process and generating a parsing table from *LR(1) items* and states, the parser can effectively handle a diverse set of context-free grammars and produce parse trees for input programs.

Producer: Elham Jafari

Computer Engineering