Practical - 9

Aim: Implement bottom up parser.

Theory:

Bottom up parser: It is a type of parser that builds the parse that builds the parse tree starting from the input symbols (leaves) and works its way up to the start symbol (root). It attempts to reduce a string of terminals to the start symbol by repeatedly applying grammar productions in reverse, a process known as reduction. The most common botton-up parsing technique is Shift - Reduce Parsing, where the parser shifts input symbols onto a stack and reduces them to non-terminals using grammar rules.

Bottom up parsers can handle a wider class of grammars than top-down parsers, including left-recursive grammars. One popular type is the LR parser, which includes SLR, LALR and Canonical LR parsers. These parsers use parsing tables (ACTION and GOTO) to determine whether to shift, reduce, accept or report an error. Bottom up parsing is powerful and commonly used in compilers due to its ability to parse complex programming language grammars.

Action Table: The Action table is a key component of an LR parser that tells the parser what action to taken when it is in a certain state and sees a specific input symbol. The action can be Shift (move the next input symbol onto the stack and go to a new state), Reduce (replace a string on the stack with a non-terminal using a production rule). Accept (if the input string has been successfully parsed), or Error (if the input is invalid). This table ensures that the parser behaves correctly at every step during bottom-up parsing.

Goto Table: The Goto table is used after a Reduce operation in an LR parser. Once a production is reduced and a non-terminal is placed on top of the stack, the GOTO table tells the parser which state it should transition to next based on the current state and the new non-terminal. Together Action table, GOTO table helps the parser maintain control and direction through its finite set of states.

Closure Function: The closure operation is used during the constructio of the LR parsing tables. Given a set of LR items (representing states), the Closure function adds new items to account for what could possibly come next in the input. It expands any item with a dot before a non-terminal by including all the productions of that non-terminal, placing a dot at the beginning. This operation heps build a complete and correct set of parser states.

SLR (Simple LR) : Simple parsing is the simplest form of LR parsing. It uses Follow sets to determine when a reduction should be applied. Although it is relatively easy to implement and creates small parsing tables, SLR can struggle with more complex or ambiguous grammars because it uses limited information for making parsing decision. As a result, it is less powerful and may lead to conflicts in some grammars.

CLR (Canonical LR) : Canonical LR parsing, also known as LR(1) parsing, is the most powerful and accurate form of LR parsing. It uses LR(1) items, which include both the position of the dot and a lookahead symbol. This provides precise context for every reduction, eliminating ambiguity. However, because it stored detailed information for each state, CLR generates very large parsing tables, making it more complex and memory intensive.

LALR (Look-Ahead LR) : LALR parsing is a practical compromise between SLR and CLR. It merges similar states from the CLR parser to reduce table size, while still using lookahead symbol for better parsing decisions than SLR. LALR parsers are widely used in compiler construction tools like YACC because they offers a good balance between power and efficiency, handling most programming languages effectively without the complexity of full CLR parsing.

Conclusion : We have studied about bottom up parser and have implemented SLR parser to print to ~~Action~~. Goto table its states and the SLR parsing table.