Bottom-Up Parsing

1 Bottom-Up Parsing

Bottom-up parsing constructs the parse tree starting from the input symbols and working up to the start symbol of the grammar. This approach is also called **shift-reduce parsing** because it involves shifting input symbols onto a stack and reducing them using grammar rules.

1.1 Types

1. **LR(0) Parser**: Basic LR parser with no lookahead. It is mainly used for basic syntax analysis in programming languages and serves as the foundation for more advanced parsers like SLR(1), LALR(1), and CLR(1).
2. **SLR(1) Parser**
3. **CLR(1) Parser**
4. **LALR(1) Parser**

2 Four Canonical Actions in Bottom-Up Parsing

1. **Shift**: Move the next input symbol onto the stack.
2. **Reduce**: Replace a set of symbols on the stack with a non-terminal using a grammar rule.
3. **Accept**: Successfully recognize the input as a valid sentence.
4. **Error**: Report a syntax error when no valid action is possible.

3 Closure and GOTO Rules

3.1 Closure Function

The **closure** of an item set adds all productions that can be derived from the current dot position.

Algorithm:

1. Start with an initial item set.
2. For each item $[A \to \alpha \bullet B\beta]$, add $[B \to \bullet\gamma]$ for all possible γ.
3. Repeat until no new items can be added.

Example:

```
Given: S' → •S
Grammar:
S → CC
C → cC | d

Closure({S' → •S}) expands to:
S' → •S
S → •CC
C → •cC
C → •d
```

3.2 GOTO Function

The **GOTO** function moves the dot (•) over a given symbol and returns a new state.

Algorithm:

1. For each item $[A \to \alpha \bullet X\beta]$, move • to the right of X.
2. Compute closure of the new set.

Example:

```
GOTO(I, C):
S → C•C  → S → C C•
```

4 Augmented Grammar

To ensure a clear start state, a new start symbol S' is added.
Example:

```
Original Grammar:
S → CC
C → cC | d

Augmented Grammar:
S' → S
S → CC
C → cC | d
```

This helps in defining **acceptance** and ensures that parsing stops correctly.

5 LR Parsing Process

The parsing process involves:

1. Constructing an **augmented grammar**.
2. Building the **canonical collection of LR items**.
3. Constructing the **LR parsing table**.
4. Parsing the input string using a **shift-reduce mechanism**.

6 LR Parsing Table Construction

The LR parsing table consists of two parts:

- **Action Table**: Defines shift, reduce, accept, or error actions.
- **Goto Table**: Defines transitions for non-terminals.