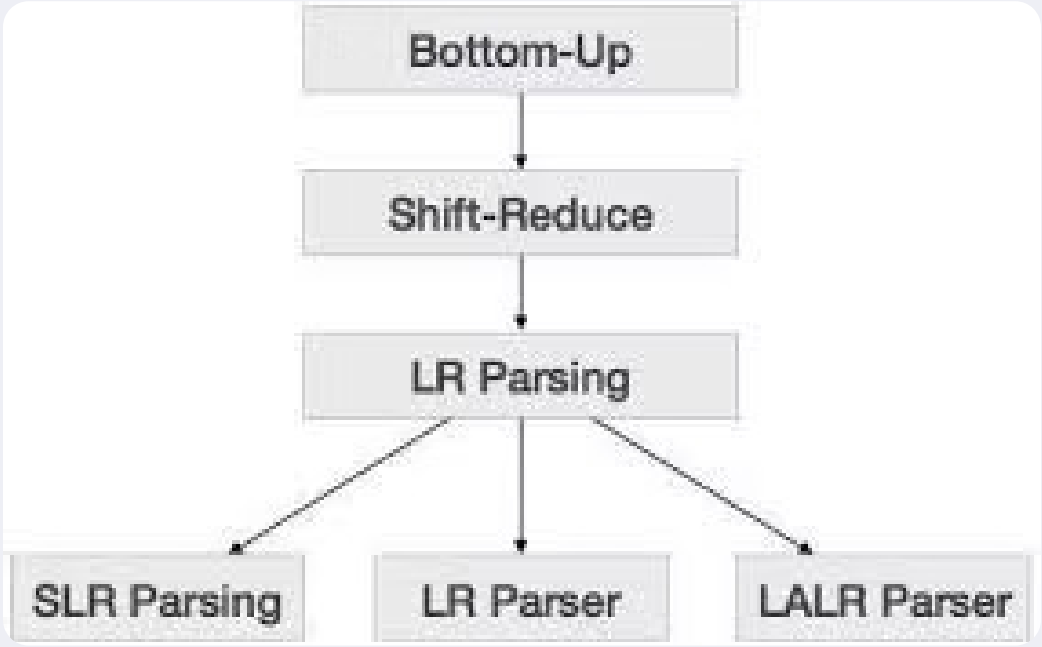
Bottom-Up Parser: Implementing Deterministic Finite Automata

PRESENTED BY: FAJAR AAMIR SHEIKH(SP22-BCS-031)

Introduction to Bottom-Up Parsers



- Builds parse tree from tokens to root
- Core operations: "Shift" tokens, "Reduce" sequences
- Handles many context-free grammars (CFGs)
- Key advantage: Early error detection

The Role of DFAs in LR Parsing

Parser's State Machine

DFA acts as the core control unit.

Representing Configurations

Each state signifies a unique parser configuration.

Collections of LR Items

States are sets of productions with a "dot" marker.

Grammar Symbol Transitions

DFA edges are labeled by terminals or non-terminals.

LR Parsing Overview: Shift-Reduce

Stack	Input	Action
$	id*id+id $	Shift id
$ id	*id+id $	Reduce E -> id
$ E	*id+id $	Shift *
$ E *	id+id $	Shift id
$ E * id	+id $	Reduce E -> id
$ E * E	+id $	Reduce E-> E * E
$ E	+id $	Shift +
$ E +	id $	Shift id
$ E + id	$	Reduce E -> id
$ E + E	$	Reduce E -> E + E
$ E	$	Accept

- Izhan

Stack Utilization

Stores parser states and grammar symbols

Input Buffer

Holds remaining tokens for processing

Table-Driven Decisions

Action and GOTO tables guide Shift, Reduce, Accept, Error

DFA-Derived Tables

Parser tables generated directly from the DFA structure

Constructing the DFA: Canonical LR(0) Items

Augmented Grammar

Start with S' -> . S, forms initial state I0

Closure Operation

Expand items with new productions based on dot position

GOTO Operation

GOTO(I, X) finds next state for symbol X

State Formation

Each unique item set becomes a DFA state

Example

A -> α . B β implies B -> . γ δ is added

Parser Table Generation from DFA

Action Table

Specifies Shift (Sj), Reduce (Ri), or Accept (acc) actions for (state, terminal) pairs. Derived from items with dot before terminal or at the end.

GOTO Table

Indicates the next state for (state, non-terminal) pairs. Directly determined by DFA transitions on non-terminals.

> Conflicts (Shift-Reduce, Reduce-Reduce) require lookaheads (SLR, LALR, LR(1)) to resolve ambiguity.

Bottom-Up Parser Algorithm (Shift-Reduce)

Initialize Stack

Push initial state (0)

Loop Execution

Until Accept or Error

Consult Action Table

Based on current state and input token

Shift Operation

Push token, then new state to stack

Reduce Operation

Pop symbols, push non-terminal, GOTO new state

Accept/Error

Parsing successful or syntax error detected

Conclusion: Power of DFA in Parsing

- DFAs enable efficient LR parser construction.
- Foundation for robust compiler front-ends.
- Facilitates precise syntax error detection.
- Enhances compiler reliability.

Deterministic Finite Automata are indispensable for syntax analysis, providing the structural backbone for powerful and efficient bottom-up parsers.