THEORY OF COMPUTATION AND COMPILERS

Unit - II

CONTEXT FREE GRAMMARS AND PARSING

- Introduction
- Context-Free Grammars - Derivation, Parse trees, Ambiguity
- Types of Parsers
- LL(K) grammars and LL(1) parsing
- **Bottom-up Parsing - handle pruning** 
- LR Grammar Parsing
- LALR parsing
- Parsing ambiguous grammars
- Error Recovery in Parsing
- YACC programming specification

Dr. R. Madana Mohana
Professor, Artificial Intelligence & Data Science | I/c-Head, Artificial Intelligence & Machine Learning
CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY
Hyderabad - 500 075, Telangana, INDIA
www.cbit.ac.in

Unit-II: Syntax Analysis (or) Parser
Bottom-up Parsing

Outline:

- Introduction
- Reductions
- Handle Pruning
- Shift-Reduce Parsing

Bottom-UP Parsing

Definition:

Construction of the **parse tree** starts from the leaf nodes (tokens or terminals of the grammar) and proceeds towards root (Starting non-terminal). i.e., construction of the parse tree is done from bottom to top of the tree.

- Here the input string is reduced to the starting non-terminal i.e. root.
- Leftmost derivation in reverse is bottom-up parsing.
- The right hand side of the productions can be replaced by the left hand side symbols.

Bottom-UP Parsing

Example:

For the given grammar below, check the string **`id+id*id`** is accepted or not by using bottom-up parsing.

`E → E + E`

`E → E * E`

`E → id`

Bottom-UP Parsing

Example: *Solution*

Given string:

=> id + id * id

=> E + id * id [E → id]

=> E + E * id [E → id]

=> E * id [E → E * E]

=> E * E [E → id]

=> E [E → E * E] // **E** is the staring non-terminal i.e., root

Therefore the given string **id+id*id** is accepted by the bottom-up parser.

Bottom-UP Parsing

Practice Problem:

For the given grammar below, check the string **abbcde** is accepted or not by using bottom-up parsing.

S → aABe

A → Abc | b

B → d

Reductions or Handle

- Bottom-up parsing is the process of " reducing" a string **w** to the start symbol of the grammar.
- At each reduction step, a specific substring matching the body of a production is replaced by the nonterminal at the head of that production.
- The key decisions during bottom-up parsing are about when to reduce and about what production to apply, as the parser proceeds.

Reductions

Definition: Handle or Reduction

If there is a rightmost derivation of the form **S =>* αAw => αβw** then the string **β** in **i^th** right sentential form **αβw** is the handle or reduction if there is production of the form **A → β** such that replacing **β** by **A** in **αβw** results in previous **(i-1)^th** right sentential form **αAw**.

Reductions

Example:

=> **id** + id * id

=> E + **id** * id [E → id]

=> **E + E** * id [E → id]

=> E * id [E → E * E]

=> **E * E** [E → id]

=> E [E → E * E]

This example illustrate a sequence of reductions; the grammar is the expression grammar. The reductions will be discussed in terms of the sequence of strings

id + id * id, E + id * id, E + E * id, E * id, E * E, E

Reductions

Example:

E → E + T | T

T → T * F | F

F → (E) | id

Right sentential form in reverse	Handle or Reduction	Reduction Production
id * id	id	F → id
F * id	F	T → F
T * id	id	F → id
T * F	T * F	T → T * F
T	T	E → T
E		

Handle Pruning

Definition-1:

The process of discovering a handle or reduction in a right-sentential form r_n and reducing it to the appropriate left-hand side to get the previous right-sentential form r_{n-1} is called **handle pruning**.

Example:

Consider the following rightmost derivation:

$$S \Rightarrow r_1 \Rightarrow r_2 \Rightarrow r_3 \Rightarrow \ldots \Rightarrow r_{n-1} \Rightarrow r_n$$

Handle Pruning

Definition-2:

A rightmost derivation in reverse is called **handle pruning**.

Definition-3:

The process of obtaining the starting non-terminal while constructing the bottom-up parse tree by reducing the handle to respective non-terminals is called **handle pruning**.

Handle Pruning

Example:

For the grammar **S → SS+ | SS* | a**

Indicate the handles in the following right sentential form "**SS+a*a+**"

Right sentential form in reverse	Handle	Reduction Production
S S + a * a +	SS+	S → SS+
S a * a +	a	S → a
S S * a +	SS*	S → SS*
S a +	a	S → a
S S +	SS+	S → SS+
S		

Handle Pruning

Limitations in Handle Pruning:

- In which position we have to apply, if there is an error.
- In which position we have to apply rightmost and leftmost derivation.

Bottom-Up Parsing Techniques

1. **Shift-Reduce Parsing (SR Parser):**
   - To overcome handle pruning, this technique is used.
2. **Operator-Precedence Parsing (OR Parser):**
   - An easy way to implementing the Shift-Reduce Parsing (SR Parser).
3. **LR Parsing:**
   - A general method of implementing Shift-Reduce Parsing (SR Parser).

Shift-Reduce Parsing (SR Parser)

- **Shift-Reduce Parser** is an efficient way of implementing bottom-up parser using explicit stack.
- The stack contains grammar symbols and input buffer holds the string to be parsed.
- In this parser, we start from string of terminals and get the start symbol.
- During parsing, always the handle will appear on top of the stack just before it is identified as the handle.

Shift-Reduce Parsing (SR Parser)

Model of SR Parser:

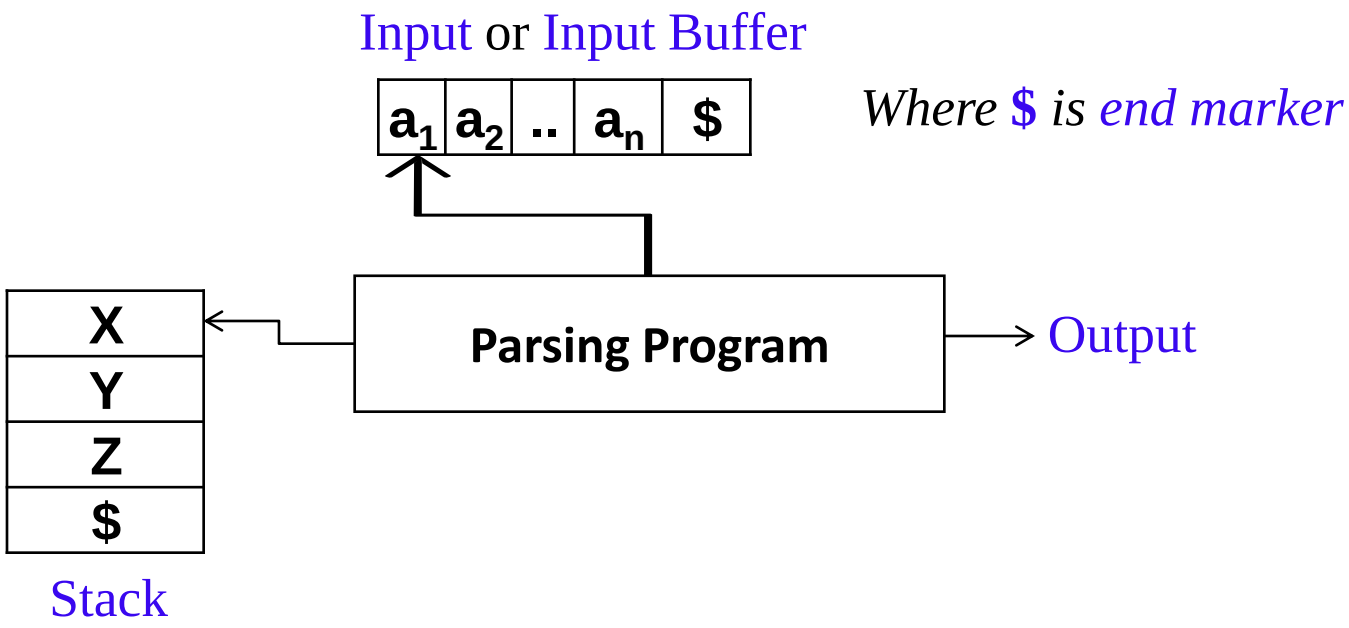

Figure: Block diagram of Shift-Reduce Parser (SR Parser)

Shift-Reduce Parsing (SR Parser)

Working of SR Parser:

The initial configuration of the parser:

Stack	Input
$	w$

The final configuration of the parser:

Stack	Input
$S	$

where **S** is the **Starting Variable**.

Shift-Reduce Parsing (SR Parser)

Working of SR Parser:

There are four actions in SR Parser as given below:

1. **Shift:** The next input symbol is shifted on to the top of the stack.
2. **Reduce:** By knowing the appropriate handle on the stack, the parser reduces this handle to the left hand side of the appropriate production.
3. **Accept:** The parser announces successful completion of parsing.
4. **Error:** The parser discovers an error and appropriate error message is displayed.

Shift-Reduce Parsing (SR Parser)

Example:

Show the sequence of moves made by the SR parser for the string **id*id** during parsing using the given grammar.

Given Grammar:

1.E → E + T | T

2.T → T * F| F

3.F → (E) | id

Shift-Reduce Parsing (SR Parser)

Example: *Solution*

Sequence of moves made by SR Parser:

Stack	Input	Action
$	id*id$	Shift id onto the stack.
$id	*id$	Reduce F → id
$F	*id$	Reduce T → F
$T	*id$	Shift * onto the stack.

Shift-Reduce Parsing (SR Parser)

Example: *Solution*

Sequence of moves made by SR Parser:

Stack	Input	Action
$T*	id$	Shift id onto the stack.
$T*id	$	Reduce F → id
$T*F	$	Reduce T → T * F
$T	$	Reduce E → T
$E	$	ACCEPT

Shift-Reduce Parsing (SR Parser)

Limitations in SR Parser:

- Ambiguity

Summary...

Bottom-Up Parsing:

- Introduction
- Reductions
- Handle Pruning
- Shift-Reduce Parsing

Reading: Aho2, Section 4.5.1 to 4.5.4

Next Lecture: Introduction to LR Parsing: Simple LR