UNIT - II

PARSING

Unit: Outcomes

- To understand the basic concepts of Parser design.
- To understand difference between top down and bottom up parser.
- To understand the implementation details of LL(1) parser.
- To understand the implementation details of LR parsers.
- To understand the implementation details of LALR parsers.
- To understand the role of data-structures in parser design.
- To understand Error Handling in parsers.
- To understand role of parser and selection of parser for language processing.

Introduction

- Parsing is a process that constructs a syntactic structure (i.e. parse tree) from the stream of tokens.
- Context Free Grammars are used to describe syntactic structure of a language.

Type of Parsers

- Majorly Two types:
  - **Top Down Parser**
  - **Bottom Up Parser**
- Other types:
  - **Operator Precedence Parser**
  - **Recursive Descent Parser**

Top–Down Parsing

- A parse tree is created from root to leaves
- The traversal of parse trees is a preorder traversal
- Tracing leftmost derivation
- Two types:
  - Backtracking parser
  - Predictive parser

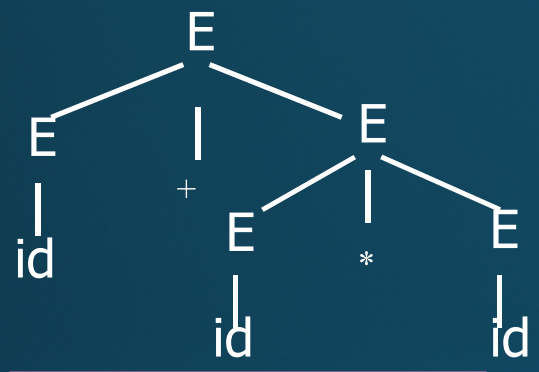
Backtracking: Try different structures and backtrack if it does not matched the input

Bottom–Up Parsing

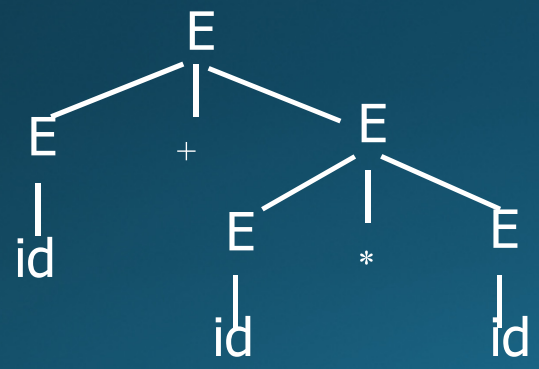
- A parse tree is created from leaves to root
- The traversal of parse trees is a reversal of post-order traversal
- Tracing rightmost derivation
- More powerful than top-down parsing

Predictive: Guess the structure of the parse tree from the next input

Parse Trees and Derivations



Top-down parsing



Bottom-up parsing

$E \Rightarrow E + E$
$\Rightarrow id + E$
$\Rightarrow id + E * E$
$\Rightarrow id + id * E$
$\Rightarrow id + id * id$

$E \Rightarrow E + E$
$\Rightarrow E + E * E$
$\Rightarrow E + E * id$
$\Rightarrow E + id * id$
$\Rightarrow id + id * id$

Top Down Parsing

- **Implementation Requirements:**
- The Grammar should not have left recursion and left factoring.
- The backtracking process helps in determining proper production rule for string generation, if required.
- For example:
  - S → aAb
  - A → c | d
  - String: "acd", then since there are two option/replacements for non-terminal symbol "A", backtracking may be useful.

Parser: LL(1) Parsing

- LL(1)
  - Read input from (**L**) left to right
  - Simulate (**L**) leftmost derivation
  - 1 lookahead symbol
- Use stack to simulate leftmost derivation
  - Part of sentential form produced in the leftmost derivation is stored in the stack.
  - Top of stack is the leftmost nonterminal symbol in the fragment of sentential form.

Concept of LL(1) Parsing

- Simulate leftmost derivation of the input.
- Keep part of sentential form in the stack.
- If the symbol on the top of stack is a terminal, try to match it with the next input token and pop it out of stack.
- If the symbol on the top of stack is a nonterminal X, replace it with Y if we have a production rule $X \rightarrow Y$.
  - Which production will be chosen, if there are both $X \rightarrow Y$ and $X \rightarrow Z$?

First Set

- To be computed for all non-terminals of LHS of production rule.
- Rules:
- A → XYZ, then compute First(A) = First(XYZ)=FIRST(X)
- If FIRST(X) contains terminal symbol in include in FIRST(A)
- If FIRST(X) contains "ε" then FIRST(A) = FIRST(X) – {ε} U FIRST(YZ)
- Now compute FIRST(YZ) = FIRST(Y)
- If FIRST(Y) contains "ε", then FIRST(YZ) = FIRST(Y) – {ε} U FIRST(Z)
- Continue process, till there exist NON-TERMINAL symbol in production rule. [Otherwise "ε" will remain in FIRST SET]

Follow Set

- To be computer for all non-terminals of RHS of production rule.
- While computing FOLLOW information, FIRST information about the NON-TERMINALS of production rules is used.
- Default rule: Follow(S) = {$}
- Rules:
- A → αXβ, then compute Follow(X) = First(β), if first of "β" contains "ε" then continue FIRST Rule if possible.
- After completely operating FIRST Rule, if FOLLOW(X) contains "ε" then FOLLOW(X) = FOLLOW(β) – {ε} U FOLLOW(A)

Construction of LL(1) parsing table

- Structure: Non-Terminals on ROW and Terminals on Columns along with "$" symbol.
- Find First and Follow.
- If A → α is production rule in grammar and symbol "a" is present in First(α) then add A → a in T[A,a]
- If First(α) contains "ε" then add A → α in table T[A,b], where symbol "b" is present in Follow(A)

Example

S → a B D h
B → B b | c
D → E F
E → g | ε
F → f | ε

S → a B D h
B → c B′
B′ → b B′ | ε
D → E F
E → g | ε
F → f | ε

NT	First	Follow
S	{a}	{$}
B	{c}	{g, f, h}
B′	{b, ε}	{g, f, h}
D	{g, f, ε}	{h}
E	{g, ε}	{f, h}
F	{f, ε}	{h}

S→aBDh [First = a]
D → EF
Find FIRST(EF) = {g, f, ε}
Since it contains "ε " find
Follow(D) = {h}

NT/T	a	b	c	g	f	h	$
S	S→aBDh						
B			B→cB′				
B′		B′→bB′		B′→ε	B′→ε	B′→ε	
D				D → EF	D → EF	D→EF	
E				E→g	E→ε	E→ε	
F					F→f	F→ε	

Example

S → ACB | CbB | Ba

A → dg | BC

B → gC | ε

C → ha | ε

No left Recursion

NT	First	Follow
S	{a, b, d, h, g, ε}	{$}
A	d, h, g, ε	{g, h, $}
B	{g,c }	{a,g,,h,$}
C	{h, ε }	{g,h,a,b,$}

NT/T	a	b	D	g	h	$
S						
A						
B						
C						

Example

S → aABb

A → c | ε

B → d | ε

No left Recursion

NT	First	Follow
S		
A		
B		

NT/T	a	b	C	D	$
S					
A					
B					
C					