Module 3

Parsing

Predictive Parser (LL(1) Parser)

- Let ‘x’ be a stack top symbol and let ‘a’ be the input symbol
- If x=a=$ then accept and break
- Else if x=a!=$ then pop ‘x’ and remove ‘a’
- Else if M[x,a] x-> y1 y2 …..yk then pop ‘x’ and push yk yk-1…..y1
- Else ERROR()

Shift Reduce Parser

- Definition of Handle
- Definition of Handle Pruning
- Actions in shift reduce parser
- Algorithm

Operator Precedence Parser

- If precedence of b is higher than precedence of a, then we define a < b
- If precedence of b is same as precedence of a, then we define a = b
- If precedence of b is lower than precedence of a, then we define a > b

Precedence relations

Relation	Meaning
$a \lessdot b$	*a* yields precedence to *b*
$a \doteq b$	*a* has the same precedence as *b*
$a \gtrdot b$	*a* takes precedence over *b*

Precedence of operator

- An identifier is always given the higher precedence than any other symbol.
- $ symbol is always given the lowest precedence.
- If two operators have the same precedence, then we go by checking their associativity.

Operator	Precedence	Associativity
id	Highest	-
^		Right
* /		Left
+ -		Left
$	Lowest	

Top Down VS Bottom up Parser

Top Down parser	Bottom up parser
In this technique, the construction of a parse tree begins at the root and works down towards the leaves.	In this technique, the construction of a parse tree begins at the leaves and works up towards the root.
Left recursion can make the top down parser go into an infinite loop.	Left recursion cause, no such problem.
Left-factoring is required.	Left-factoring is not required.
There is no concept of handle selection.	A selection of handles is required.
In this technique, the actions are pop and remove.	In this technique, the actions are shift and reduce.
Derive the string using LMD	Derive the string using RMD in reverse which is called canonical derivation.
It is simple to produce parser.	It is difficult to produce parser.
It uses LL(1) grammar to perform parsing.	It uses SLR, CLR and LALR to perform parsing.
It is not accepting ambiguous grammar.	It is accepting ambiguous grammar.
Example: Recursive Descent Parser, Predictive parser	Example: Shift reduce parser, OPP, SLR, CLR. LALR

Top Down VS Bottom up Parser

Compiler	Interpreter
For Converting the code written in a high-level language into machine-level language so that computers can easily understand.	An Interpreter works line by line on a code. It also converts high-level language to machine language.
It is faster.	It is slower.
It is more efficient.	It is less efficient.
Execution of the program takes place only after the whole program is compiled.	Execution of the program happens after every line is checked or evaluated.
The compiler generates an output in the form of (.exe).	The interpreter does not generate any output.
Object code is permanently saved for future use.	No object code is saved for future use.
CPU utilization is more in the case of a Compile.	CPU utilization is less in the case of a Interpreter.

SLR Parser

Closure operation: If a dot is present before any other terminal, then add all of its productions to the state.

Closure item: An item created by the closure operation on state.

Complete item: An item where dot is at the end of RHS.

LR(0) item: it is a set of production of given grammar G with dot at some position on the right side of the production.