## 4.5 Bottom-Up parsing

General style of bottom-up parsing is shift-reduce parsing.
The process of bottom-up parsing is to "reduce" a string w to the start symbol S if a grammar.

Consider the grammar : $S \to CC$
$C \to aC \mid b$

I/P string aabab can be reduced to S by scanning for a substring that matches to the right side of some production. Scanning from left to right we get the following reduction steps:

a a b a b
a a C a b
a C a b
C a b
C a C
C C
S

The above reduction is reverse of rightmost derivation:: $S \overset{rm}{\Rightarrow} CC$
$\overset{rm}{\Rightarrow} CaC$
$\overset{rm}{\Rightarrow} Cab$
$\overset{rm}{\Rightarrow} aCab$
$\overset{rm}{\Rightarrow} aaCab$
$\overset{rm}{\Rightarrow} aabab$

## Handle:

Informally, a "handle" of a string w is a substring that matches the right side of a production, and where reduction to the nonterminal on the left side of the production represent one step along the reverse of a rightmost derivation.

Formally, a handle of a right-sentential form $\gamma$ is a production $A \to \beta$ and a position of $\gamma$ where the string $\beta$ may be found and replaced by A to produce the previous right-sentential form in a rightmost derivation of $\gamma$.

i.e. if $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha \beta w$,

then $A \to \beta$ in the position following $\alpha$ is a handle of $\alpha \beta w$.

Reducing $\beta$ to A in $\alpha \beta w$ can be thought of as "pruning the handle".

Stack implementation of Shift Reduce Parsing:

The 2 data structures used are.

1) Stack: It holds grammar symbols. Initially it contains $ only.

2) Input: It contains string to be parsed. (ended with $). Initially input buffer has w$ where w is the string to be parsed.

When stack has "$S" and input has "$", parser announces successful parsing of string.

Till the input is either accepted or error is reported, the parser keeps taking the following 2 actions.

a) Shift : The next input symbol is shifted onto the TOS.

b) Reduce : The parser (on locating a handle on top of stack) replaces the handle by the non terminal.

Parsing of "aabab" for grammar S → CC
C → aC/b

| Stack | Input | Action |
|---|---|---|
| $ | aabab$ | shift |
| $a | abab$ | shift |
| $aa | bab$ | shift |
| $aab | ab$ | Reduce by C→b |
| $aaC | ab$ | Reduce by C→aC |
| $aC | ab$ | Reduce by C→aC |
| $C | ab$ | Shift |
| $Ca | b$ | shift. |
| $Cab | $ | Reduce by C→b |
| $CaC | $ | Reduce by C→aC |
| $CC | $ | Reduce by S→CC |
| $S | $ | Accept. |

How are the choices of action (shift or Reduce) are made so that the parser works correctly ?

Ans: There are 2 techniques.

A) Operator Precedence & B) LR Parsers

## 4.6 Operator - Precedence Parsing:

The requirement for a grammar to be an operator grammar are

i) It must be of form Type 2 of Chomsky Hierarchy (ie CFG)

ii) It cannot have $\epsilon$-productions (of type $A \to \epsilon$)

iii) No production has 2 adjacent Variables/Non-terminals.

Hence $\left.\begin{array}{l} S \to CC \\ C \to aC \mid b \end{array}\right\}$ is not an operator grammar

$E \to E+E \mid E*E \mid (E) \mid id$ } is an operator grammar

~~It takes as~~

Operator precedence parser takes action based on manually created precedence relations.

| Relation Symbol | Meaning |
|---|---|
| $a \lessdot b$ | a "yields precedence to" b [Action - Shift] |
| $a \doteq b$ | a "has same precedence as" b [Action - Shift] |
| $a \gtrdot b$ | a "takes precedence over" b [Action - Reduce] |

The intention of precedence relations is to delimit the handle of a right-sentential form with $\lessdot$ marking left-end, $\doteq$ in the interior and $\gtrdot$ marking right-end.

The 2 ends of the string are marked by '$' with precedence

'$' $\lessdot$ every terminal of grammar

and

every terminal of grammar $\gtrdot$ '$'.

Example: Consider grammar

$$E \rightarrow E+E \mid E*E \mid id$$

Assuming

i) '*' has higher precedence than '+'

ii) Both '*' and '+' are left associative.

the precedence relation will be

| Top of stack terminal \ Incoming terminal | id | + | * | $ |
|---|---|---|---|---|
| id | | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ |
| + | $\lessdot$ | $\gtrdot$ | $\lessdot$ | $\gtrdot$ |
| * | $\lessdot$ | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ |
| $ | $\lessdot$ | $\lessdot$ | $\lessdot$ | |

Placing ~~relation~~ precedence relation for string id + id * id & determining handle.

$ $\lessdot$ id $\gtrdot$ + $\lessdot$ id $\gtrdot$ * $\lessdot$ id $\gtrdot$ $

Reducing by $E \to id$ given $E+E*E$ placing precedence relation (by ignoring the variables / Non-terminals)

$\$ \lessdot + \lessdot * \gtrdot \$$

Hence reduce by $E \to E*E$. we get $E+E$

Next,

$\$ \lessdot + \gtrdot \$$

Hence finally reduce by $E \to E+E$ to get $E$.

With "$E" in stack and "$" in input the parser accepts the string as valid.

Working of operator precedence parser.

1) Initially top of stack contains '$' and input pointer points to the first symbol of w$ ; where w is the input string followed by $.
2) Let 'a' be the topmost terminal of stack and 'b' be the input terminal.
   - If $a \lessdot b$ or $a \doteq b$ then
     a) Push 'b' on top of stack and
     b) Advance to next input terminal.
   - If $a \gtrdot b$ then
     a) Repeatedly pop all terminals and non-terminals from top of stack until the terminal is $\lessdot$

until the tos is a terminal having $<\cdot$ precedence than the last popped symbol.
[ ~~tos~~ Optionally, consider the left variable of the production found can be pushed on the stack (Hierarchically) ].

3) Repeat step 2 till tos and input contain '$'.

## Tasks

1) Solve Q5 of Assignment 3

2) Refer to the Operator precedence relation for grammar
$E \to E+E \mid E*E \mid E-E \mid E/E \mid E \uparrow E \mid (E) \mid id$
from book.
and hence parse i/p string
$id * (id + id) \uparrow id \uparrow id$.

3) Try to describe precedence relation for grammar.

$$C \to C \,\&\&\, C \mid C \,||\, C \mid (C) \mid b$$

where && has higher precedence over ||
and both operators are left associative