**LL(1) Parser:-** A predictive Parser, in which the parsing table has no multiple defined entries is said to be LL(1) Parser.

In LL(1) Parser, the First 'L' stands for 'Scanning the input from left to Right' and Second 'L' stands for 'Left Most Derivation' and '1' stands for using one input symbol at each step to make parsing decision.

**Bottom-UP Parser:-** Bottom-up Parser build parse tree from the bottom (leaves) to the TOP (Root).

The bottom-up parsing can be implemented by Shift Reduce Parser (SR Parser)
(SR Parser)

**Shift Reduce Parser:-** A shift reduce parser tries to reduce the given input string to the start symbol of the grammar by using rightmost derivation in reverse

input string $\xrightarrow{\text{Reduced to}}$ start symbol of Grammar.

At each step, a particular substring matching the right side of a production is replaced by the symbol on the left side of that production.

ex:- $S \to aABb$
$A \to aA|a$
$B \to bB|b$

$\Rightarrow a\underline{a}abb$
$\Rightarrow$
$\Rightarrow a\underline{aA}bb$
$aA\underline{bb}$
$\Rightarrow aA\underline{B}b$

Substring "aaabb"

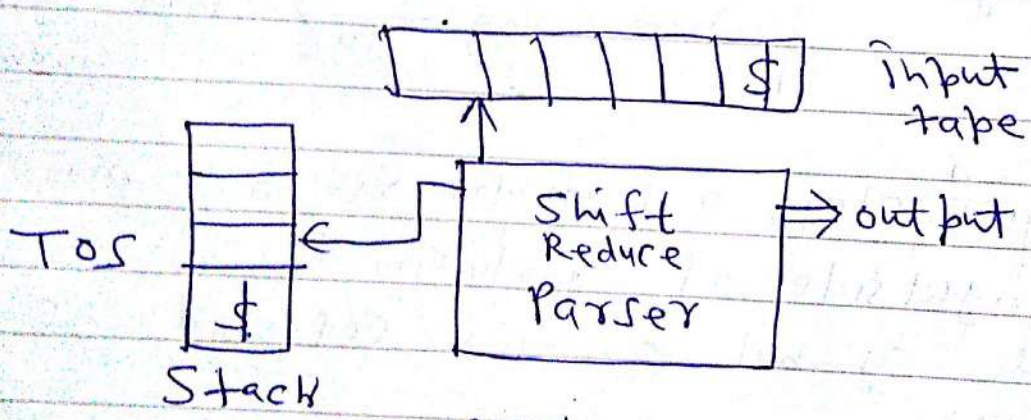
~~aaabb~~
$\Rightarrow$ ~~aAbbb~~
$\Rightarrow$ ~~aAbBb~~
$\Rightarrow aABb$
$\Rightarrow S$

**Handles:** A "handle" of a string is a substring that matches the rightside of a production and whose reduction to the nonterminal on the left side of the production represents one step along the reverse of a rightmost derivation.

**Handle Pruning:-** Handle Pruning in bottom up parser (shift reduce parser) is essentially a process of detecting handles & using them in reduction.

| Sentential form | Handle | Production |
|---|---|---|
| id + id * id | id | E → id |
| E + id * id | id | E → id |
| E + E * id | id | E → id |
| E + E * E | E * E | E → E * E |
| E + E | E + E | E → E + E |
| E | | |

## Parsing using Shift Reduce Parser "OR" Stack implementation of shift Reduce Parser




Model of a shift Reducer Parser

A Shift Reduce (SR) Parser Consists of

(i) A stack to hold grammar Symbols.
(ii) An input tape consists of input string.
(iii) A $ symbol at the end of stack and end of input tape.

Date

The parser performs following basic operations:-

(1) Shift:- The next input symbol is shifted onto the top of the stack.

(2) Reduce:- If the handle appears on the top of stack, then pop-off it and replace it by LHS ~~top pop~~ Push-in.

(3) Accept:- If the stack contains start variable only and input buffer is empty at the same time, then action is called "Accept".

(4) Error:- If ~~Stack Cannot~~ Parser can not either perform 'shift' or 'Reduce' or can not even perform the 'accept' action is called an "Error".

---

e.g.:- CFG $E \to E+E$

$E \to E*E$

$E \to id$

Perform shift Reduce Parsing of string " $id_1+id_2*id_3$ "

| Stack | Input string | Action |
|---|---|---|
| $ | id+id*id$ | shift |
| $id | +id*id$ | Reduce E→id |
| $E | +id*id$ | shift |
| $E+ | id*id$ | shift |
| $E+id | *id$ | Reduce E→id |
| $E+E | *id$ | shift |
| $E+E* | id$ | shift |
| $E+E*id | $ | Reduce E→id |
| $E+E*E | $ | E→E*E |
| $E+E | $ | E→E+E |
| $E | $ | Accept |

## Operator Precedence Grammar:-

A grammar G is said to be operator precedence grammar if it posses the following properties:-

1. No production on the right side is ∈.
2. has no two adjacent non-terminal on the right side.

A grammar with only property (2) is called Operator grammar.

Consider the following grammar:-

$E \rightarrow EAE \mid (E) \mid -E \mid id$ — (1)

$A \rightarrow + \mid - \mid * \mid / \mid \uparrow$ — (2)

This grammar is not operator precedence grammar because the right side has two Consecutive non terminal.

If we put the value of A in first Grammar, then $E \rightarrow E+E \mid (E) \mid -E \mid -E \mid id \mid E-E \mid E*E \mid E/E \mid E\uparrow E$

Now it is operator precedence grammar.

### Adv. of operator precedence grammar:-

1. This type of parsing is simple to implement.

### Disadv. of operator precedence grammar:-

1. It is hard to handle tokens like minus sign, which has two different precedence (it may be unary or binary).

(b) $a \lessdot b$

## Computation of Operator-Precedence Relation (alternate method)

### Algorithm for Computation of Operator-Precedence Relation

① Compute Leading (A) & Trailing (A) for each non-terminal.

② For each production $A \to X_1 X_2 \cdots X_i X_{i+1} X_{i+2} \cdots X_n$

(a) If $X_i$ and $X_{i+1}$ are both terminals, then $X_i = X_{i+1}$ & If $X_i$ & $X_{i+2}$ = terminals, $X_{i+1}$ = Nonterminal, set $X_i = X_{i+2}$

(b) If $X_i$ = terminal & $X_{i+1}$ = Non terminal then for all 'a' in Leading ($X_{i+1}$), set $X_i \lessdot a$.
$X_i <$ Leading ($X_{i+1}$)

(c) If $X_i$ = Non-Terminal & $X_{i+1}$ = Terminal, then for all (a) in Trailing ($X_i$), set $a \gtrdot X_{i+1}$.
Trailing ($X_i$) $> X_{i+1}$

③ Set $\$ \lessdot a$ for all a in Leading (S) and set $b \gtrdot \$$ for all b in trailing (S), where S → start symbol.
$\$ <$ Leading (S) & trailing (S) $> \$$

### Computation of Leading :-

① If $A \to \gamma a \delta$, where $\gamma$ is $\epsilon$ or single Nonterminal then Leading (A) = {a}

② If $A \to B \alpha$, where $B \to$ terminable, $\alpha \to$ anything, then each terminal in Leading (B) will be in Leading (A), Lead(A) ⊇ Lead (B)

### Computation of Trailing :-

① If $A \to \gamma a \delta$, where $\delta$ is $\epsilon$ or single Nonterminal then Trailing (A) = {a}

② If $A \to \alpha B$, then each terminal in Trailing (B) will be in trailing (A). trailing (A) ⊇ trailing (B)
$\alpha$ = anything, B = terminable

**Example:-** $E \to E+T \mid T$, $T \to T*F \mid F$, $F \to (E) \mid id$ } is a Grammar(G)

Determine the operator precedence Relation:-

**Ans** First Calculate Leading & Trailing for each Non-terminal

$E \to E+T$ (A → Aα)

Lead(E) = {+} — rule-①

Since $E \to T$ (A → B), Lead(E) ⊇ Lead(T)

$T \to T*F$ (A → Aα)

Lead(T) = {*} — rule-①

$F \to (E) \mid id$ (A → (A), A → a)

Lead(F) = {(, id} — rule-①

Using rule-②

$T \to F$ (A → B)

Lead(T) ⊇ Lead(F) = {(, id}

So Lead(T) ⊇ {*, (, id}

Using rule-②

$E \to T$

Lead(E) ⊇ Lead(T) ⊇ {*, (, id}

So Lead(E) = {+, *, (, id}

$T \to F$ (A → B)

Trail(T) ⊇ Trail(F) ⊇ {), id}

So Trail(T) ⊇ {*, ), id}

$E \to T$

Trail(E) ⊇ Trail(T) = {*, ), id}

So Trail(E) = {+, *, ), id}

$E \to E+T$

Trail(E) = {+} — rule-①

$T \to T*F$

Trail(T) = {*} — rule-①

$F \to (E) \mid id$

Trail(F) = {), id} — rule-①

$E \to E+T$

$E \to T$

$T \to T*F$

$T \to F$

$F \to (E)$

$F \to id$

| $o_1$ \ $o_2$ | + | * | ( | ) | id | $ |
|---|---|---|---|---|---|---|
| + | > | > | < | > | < | > |
| * | > | > | < | > | < | > |
| ( | < | < | < | = | < | |
| ) | > | > | | > | | > |
| id | > | > | | > | | > |
| $ | < | < | < | | < | |

Operator Precedence Relations

for $E \to E+T$ — (1)

Since E is Nonterminal & + is terminal

& Trail(E) = {+, *, ), id} using rule 2(c)

So [{+, *, ), id} > {+}]

Trail(E) > {+}

from the same production rule $E \to E+T$

Since + is terminal & T is a Nonterminal

& Lead(T) = {*, (, id}

[{+} < {*, (, id}]

{+} < Lead(T)

for $T \to T * F$ — (2)

Since T is non-terminal & * is terminal

& Trail(T) = {*, ), id}

So [{*, ), id} > {*}]

Trail(T) > {*}

$T \to T * F$

Since * is terminal & F is Non-terminal

Lead(F) = { (, id}

[{*} < { (, id}]

{*} < Lead(F)

for $F \to (E)$ — (3)

Since ( is terminal & E is non-terminal

Lead(E) = {*, +, (, id}

[{(} < {*, +, (, id}]

{(} < Lead(E)

&

Trail(E) > {)}

for F = (E)

since E is non-terminal & ) is terminal

Trail (E) = { *, +, ), id }

[{*, +, ), id} > { ) }]

for F = (E)

terminal Nonterminal terminal

Now

Using (1) So {(} = {)} (Using rule 2(a).)

{$} < Leading (E)

{$} < { *, +, (, id }

and trailing (E) > {$}

{*, +, ), id} > {$}

## Operator Precedence Parsing Algorithm

The operator Precedence parsing algorithm is as given:-

(1) If only $ is on the stack and $ is on the input then accept the string.

(2) If a is the top symbol on the stack & b is the current input symbol

(i) If $a < b$ or $a = b$, then shift b onto the stack.

(ii) If $a > b$ then pop the stack until the top stack terminal is related by $<$ to the terminal most recently popped. After that the handle b/w $< \ldots >$ is reduced by corresponding LHS

Parse Using following Parsing relations:-

| | id | + | * | $ |
|---|---|---|---|---|
| id | | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ |
| + | $\lessdot$ | $\gtrdot$ | $\lessdot$ | $\gtrdot$ |
| * | $\lessdot$ | $\gtrdot$ | $\gtrdot$ | $\gtrdot$ |
| $ | $\lessdot$ | $\lessdot$ | $\lessdot$ | |

| Stack | Input | Remark |
|---|---|---|
| $ | id+id$ | $ <· id |
| $id | +id$ | id ·> + |
| $ | +id$ | $ <· + |
| $+ | id$ | + <· id |
| $+id | $ | id ·> $ |
| $+ | $ | + ·> $ |
| $ | $ | Accept |

**Precedence Relations:-** The operator precedence parser depends upon the following three precedence relations..

| Relation | Meaning |
|---|---|
| a <· b | a yields precedence to b. |
| a =· b | a has same precedence as b. |
| a ·> b | a takes precedence over b. |

## Precedence Functions for operator Precedence Parser:-

* Operator precedence parser does not store the table of precedence relations.
* The precedence relations table can be encoded by two precedence functions f and g which maps terminal symbols to integers.

### Algorithm for finding precedence Functions:-

(1) Create symbol $f_a$ & $g_a$ for each terminal either a or $.

(2) Partition the created symbols into as many groups as possible. If $a = b$, then $f_a$ & $g_b$ are in the same group.

(3) Create a directed graph whose Nodes are the groups found in (2). If $a \lessdot b$, place an edge of $g_b$ to $f_a$. If $a \gtrdot b$, place an edge from $f_a$ to $g_b$.

(4) If the graph has a cycle, then no precedence function exist. If there are no cycles, then $f(a)$ = length of longest path from $f_a$
$g(a)$ = length " " " " $g_a$

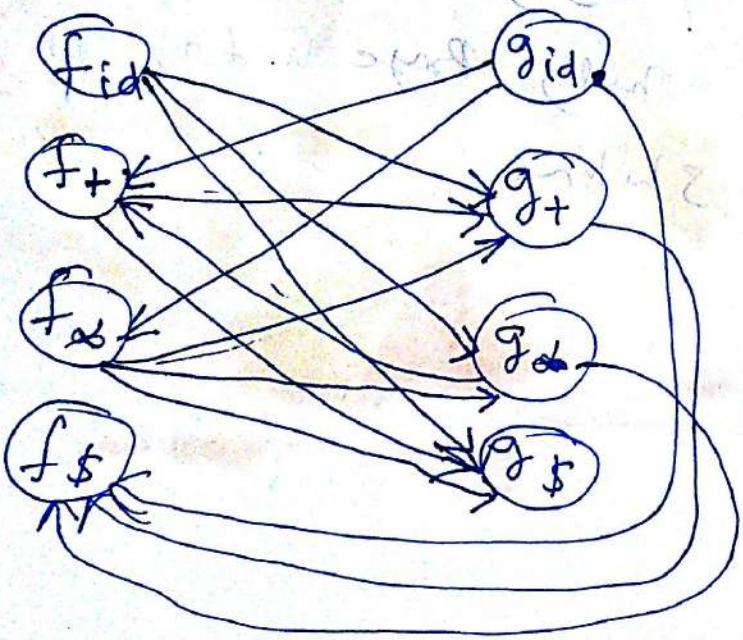
If $a \gtrdot b$, then $f(a) > g(b)$
If $a \doteq b$, then $f(a) = g(b)$
If $a \lessdot b$, then $f(a) < g(b)$

for the following operator precedence Relation matrix, Determine the operator precedence functions

Q8

| | id | + | * | $ |
|---|---|---|---|---|
| id | | ·> | ·> | ·> |
| + | <· | ·> | <· | ·> |
| * | <· | ·> | ·> | ·> |
| $ | <· | <· | <· | |

Q9




| | id | + | * | $ |
|---|---|---|---|---|
| f | 4 | 2 | 4 | 0 |
| g | 5 | 1 | 3 | 0 |

Operator Precedence Functions

Lalit Saraswat CS-3rd year

## LR Parser:-

This is the most effective method of bottom-up parsing which can be used to parse the large class of CFGs.

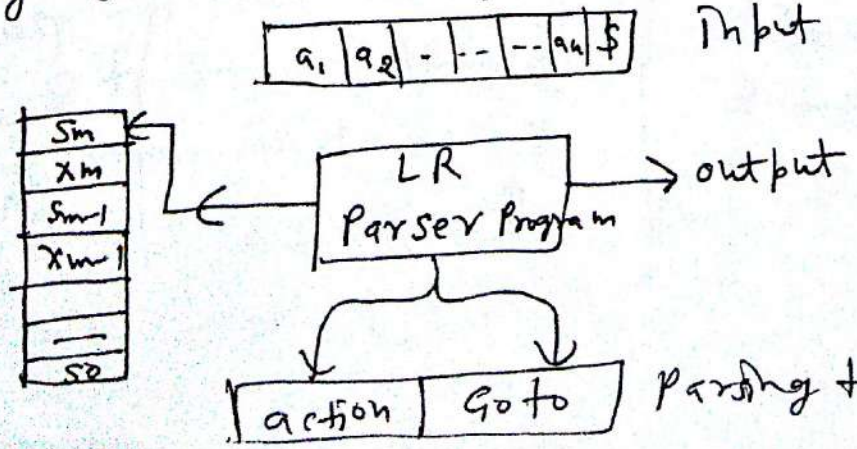
- This method is also called "LR(k)" parsing, where
  - L stands for Left to Right Scanning.
  - R stands for Right most derivation in reverse
  - k is the no. of input symbol used to make parsing decision. When k is omitted, k is assumed to be 1.

## Properties of LR Parser:-

1. LR parser detect syntactical errors very efficiently.
2. LR parser works using non backtracking shift reduce technique.
3. LR parser can be constructed to recognize most of programming language for which CFG can be written.

## Model (Structure) of an LR Parser:-

The model of LR parser consists of an input, an output, a stack, LR parser program and a parsing table having two parts (action & goto)



The working of the Parsing program for an input string is as follows:-

① Initially push 0 as initially state onto the stack & place the input string with $ (end marker) on the input tape.

② If S is onto the top of the stack and a is the symbol from input buffer.

(a) If action[S, a] = shift J, then push 'a' and then push J onto the stack. Advance the input lookahead pointer.

(b) If action[S, a] = reduce $A \rightarrow \beta$ then Pop off $2*|\beta|$ symbols from stack. If i is on the top of stack, then push A, then push goto[i, A] on the top of the stack.

③ If action[S, a] = accept, then (halt) the parsing process. It indicates the successful parsing.

routine.

**LR Grammar:-** A grammar for which we can construct a parsing table in which every entry is uniquely defined is said to be an LR grammar. A grammar that can be parsed by an LR parser examining upto k input symbol on each move is called an LR(k) grammar.

## Types of LR parser:-

The various types of LR parser is as follows:-

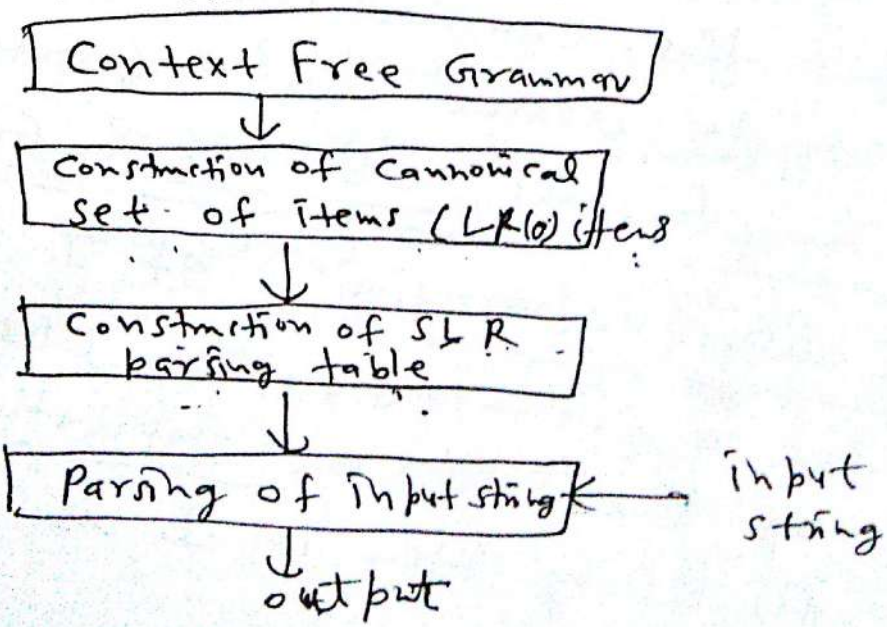
(1) **SLR Parser:-** also called simple LR parser. It is easy to implement but covers only a certain class of grammars.

(2) **CLR Parser:-** also called canonical LR parser. It is very expensive (difficult) to implement. But it works on a very large class of grammar.

(3) **LALR Parser:-** ~~LOOKAHEAD~~ Lookahead LR parser is intermediate in power between SLR and CLR parser. LALR parser will work on most of the grammar & with some effort can be implemented efficiently.

---

**SLR Parser:-** In SLR parsing, the parsing can be done as follows:-




→

The basic Definition of

~~Canonical Collection~~ of LR(0) items & other terms

(27)

① LR(0) item :- is a production rule in which • is inserted at some position in RHS of the rule.

Production $A \rightarrow XYZ$ generates four items

$A \rightarrow \cdot XYZ$

$A \rightarrow X \cdot YZ$

$A \rightarrow XY \cdot Z$

$A \rightarrow XYZ \cdot$

$G' \begin{cases} S' \rightarrow S \\ S \rightarrow aA \\ A \rightarrow b \end{cases}$

Production $A \rightarrow \epsilon$ generates only one item, $A \rightarrow \cdot$

② Augmented Grammar If G is a grammar with start symbol S, then G' (Augmented Grammar) for G, is G with a new start symbol S' & having production $S' \rightarrow S$.

The purpose of this grammar is to indicate the acceptance of input. i.e. when parser is about to reduce $S' \rightarrow S$, it reaches to acceptance state.

③ Canonical LR(0) Collections :- To construct the Canonical LR(0) Collection of items, we need to define an augmented grammar & two functions, CLOSURE & GOTO.

Closure operation :- If I is the set of items for a grammar G, then Closure (I) can be calculated by the rules :- Computation of closure

① Every item in I is in Closure (I).

(2) If $A \rightarrow \alpha . B \beta$ is in closure(I) and $B \rightarrow \gamma$ is a production, then add the item $B \rightarrow . \gamma$ to I, if not already there.

Goto operation:- Goto Computation

The Goto(I, X) : Can be calculated by first identify all the items in I in which the dot(.) has X on the right side. Then, move the dot(.) in all the selected items one position to the right (i.e. over X), and then take a closure of the set of these items.

$A \rightarrow B.X$

Algorithm for the construction of Canonical Collection of LR(0) items

(1) for the grammar G, add $(S' \rightarrow . S)$ in the set of items C. where C is the Cannonical Collection of set of LR(0) items.

(2) for each set of items I in C and for each grammar symbol X (may be terminal / Non-terminal) such that Goto(I, X) is not empty and is not in C, then add Goto(I, X) to C. This process will be repeated until no more set of items can be added to C.

(1) Find the Canonical Collection of LR(0)/SLR(0) Items for the following grammar-- (30)

Production No.

I =
(1) $E \to E+T$
(2) $E \to T$
(3) $T \to T*F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to id$

Ans First we will add augmented Grammar $E' \to E$ to it.

So $I = E' \to E.$
$E \to E+T$
$E \to T$
$T \to T*F$
$T \to F.$
$F \to (E)$
$F \to id$

Taking Closure (I) =

$I_0 = E' \to .E$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

Now apply goto to various Nonterminals & Terminals having . at L.H.S.

$I_1 = goto(I_0, E)$
$= E' \to E.$
$E \to E.+T$

$I_2 = goto(I_0, T)$
$= E \to T.$
$T \to T. * F$

$I_3 = goto(I_0, F)$
$= T \to F.$

$I_4 = goto(I_0, ()$
$= F \to (.E)$
$E \to .E+T$
$E \to .T$
$T \to .T * F$
$T \to .F$
$F \to .(E)$
$F \to .id$

$I_5 = goto(I_0, id)$
$= F \to id.$

Now for the terminal & Nonterminal in $I_1, I_2, I_3, I_4$ & $I_5$.

$I_6 = goto(I_1, +)$
$= E \to E+.T$
$T \to .T * F$
$T \to .F$
$F \to .(E)$
$F \to .id$

$goto(I_6, F)$
$\Rightarrow T \to F.$
$\Rightarrow I_3$

$goto(I_2, *) = I_7$
$= T \to T * .F$
$F \to .(E)$
$F \to .id$

$I_3$ does not have any Nonterminal & terminal

Now for $I_4$

$goto(I_4, E) = I_8$
$= F \to (E.)$
$E \to E.+T$

$goto(I_4, T) = I_2$
$= T \to T. * F$ ||
$E \to T.$
$= I_2$ (already calculated)

$goto(I_4, F)$
$= T \to F.$
$= I_3$ (already calculated)

$goto(I_4, ( ) = I_4$
$= F \to (.E)$
$E \to .E+T$
$E \to .T$
$T \to .T * F$
$T \to .F = I_4$
$F \to .(E)$ (already
$F \to .id$ Calculated.

$goto(I_4, id)$ already
$= F \to id.$ Calculated
$= I_5$

$I_5$ does not have any terminal or non-terminal

for $I_6$,

$I_9 = goto(I_6, T)$
$\Rightarrow E \to E+T.$
$T \to T.*F$

for $I_7$,

$I_{10} = goto(I_7, F)$
$= T \to T*F.$ ✓

for $I_8$,

$I_{11} = goto(I_8, ))$
$= F \to (E).$ ✓

$goto(I_8, +)$ ✓
$I_6 \equiv E \to E+.T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

$I_4 = goto(I_7, ($
$= F \to (.E)$
$E \to .E+T$ ✓
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

$goto(I_7, id)$
$= F \to id.$ ✓
$= I_5$

$goto(I_6, ($
$= F \to (.E)$
$E \to .E+T$
$E \to .T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$
$= I_4$

$goto(I_6, id)$
$\Rightarrow F \to id.$
$= I_5$

$goto(I_9, *)$
$\Rightarrow T \to T*.F$
$F \to .(E)$
$F \to .id = I_7$

So Now there is no item that can be added in the set of items. The collection of items is from $I_0$ to $I_{11}$.

## Algorithm for the Construction of SLR parsing table :-

An SLR Parsing Table consists of a parsing action function "ACTION" and a goto function "GOTO"

(1) Initially construct $C = \{I_0, I_1, I_2 \dots I_n\}$, where C is a Collection of LR(0) items.

(2) The parsing action for each item $I_i$ are as given :-

(a) If $A \to \alpha . a\beta$ is in $I_i$ & goto $(I_i, a) = I_j$ then set action $[I_i, a] \to$ shift j. In this 'a' is a terminal.

(b) If there is a rule $A \to \alpha .$ is in $I_i$ then set action $[I_i, a]$ to reduce $A \to \alpha$ for all symbol a, where $a \in$ follow(A). But A must not be an augmented grammar S'.

(c) If $S' \to S .$ is in $I_i$, ~~then action $[I_i, \$]$~~ then action $[I_i, \$]$ = "accept".

(3) The goto Transitions for state i is Considered for non-terminals only. If goto $[I_i, A] = I_j$, then goto $[I_i, A] = j$.

(4) All the entries not defined by rule 2 and rule 3 are Considered to be "error".

---

Now follow(E) = {+, ), $}

follow(T) = {+, *, ), $}

follow(F) = {+, *, ), $}

As we have already calculated :-

(34) (34)

Since $I_0 \Rightarrow$ $E' \to \cdot E$

$E \to \cdot E+T$

$E \to \cdot T$

$T \to \cdot T * F$

$T \to \cdot F$

$F \to \cdot (E)$

$F \to \cdot id$

$F \to \cdot (E)$ is in $I_0$

$A \to \alpha \cdot a \beta$

$\alpha = \epsilon, a = (, \beta = E)$

goto $(I_0, ()) = I_4$, So action $[0, (] = S_4$

Now for $F \to \epsilon \cdot id$ | goto $(I_0, id) = I_5$,
$A \to \alpha \cdot a \beta$ | action $[0, id] = S_5$

---

$I_1 = E' \to E \cdot$

$E \to E \cdot +T$

for $E \to E \cdot +T$
$A \to \alpha \cdot a \beta$
$a = +$
goto $(I_1, +) = I_6$, action $[1, +] = S_6$

for $E' \to E \cdot$
So action $[1, \$] =$ accept

---

$I_2 =$ $E \to T \cdot$
$T \to T \cdot * F$

for $E \to T \cdot$
$A \to \alpha \cdot \beta$
So follow $(E) = \{+, ), \$\}$
action $(2, +) = r_2$
action $(2, )) = r_2$
action $(2, \$) = r_2$
| since it is production No. 2

$I_2 \to T \to T \cdot * F$
$A \to \alpha \cdot a \beta$
$a = *$
$\to$ goto $(I_2, *) = I_7$
So action $(2, *) = S_7$

---

$I_3 \Rightarrow T \to F \cdot$
$A \to \alpha \cdot$
follow $(T) = \{+, *, ), \$\}$
since it is production No. 4
$(3, +) = r_4, (3, *) = r_4, (3, )) = r_4,$
$(3, \$) = r_4$

---

$I_4 \Rightarrow F \to (\cdot E)$
$E \to \cdot E+T$
$E \to \cdot T$
$T \to \cdot T * F$
$T \to \cdot F$
$F \to \cdot (E)$
$F \to \cdot id$

for $F \to \epsilon \cdot (E)$
$A \to \alpha \cdot a \beta$
$a = ($
goto $(I_4, () = I_4$
action $(4, () = S_4$

for $F \to \cdot id$, $a = id$
$A \to \alpha \cdot a \beta$
goto $(I_4, id) = I_5$, action $(4, id) = S_5$

---

$I_5 \Rightarrow T \to F \cdot$
$A \to \alpha \cdot$
follow $(T) = \{+, *, ), \$\}$

$I_2 \to T \to T \cdot * F$
$A \to \alpha \cdot a \beta$
$a = *$
$\to$

Now for $I_5$ = $F \to id.$
$A \to \alpha.$
follow(A) = (+, *, ), $)
action(5,+) = $r_6$
action(5,*) = $r_6$
action(5,)) = $r_6$
action(5,$) = $r_6$
Production No. 6

for $I_7$,
$F \to .id$
$A \to \alpha . a \beta$
$a = id$
goto($I_7$, id) = $I_5$
So action(7, id) = $S_5$

Now for $I_6$
$E \to E+.T$
$T \to .T*F$
$T \to .F$
$F \to .(E)$
$F \to .id$

$F \to .(E)$
$A \to \alpha . a \beta$
$\alpha = \in$, $a = ($, $\beta = E)$
goto($I_6$, () = $I_4$
So
action(6, () = $S_4$

$F \to .id$
$A \to \alpha . a \beta$
$a = id$
goto($I_6$, id) = $I_5$
So action(6, id) = $S_5$

Now for $I_8$
$F \to (E.)$
$E \to E.+T$
So for
$E \to E.+T$
$A \to \alpha . a \beta$
$a = +$
goto($I_8$, +) = $I_6$
So action(8, +) = $S_6$

for $I_8$
$F \to (E.)$
$A \to \alpha . a \beta$
$a = )$
goto($I_8$, )) = $I_{11}$
So action(8, )) = $S_{11}$

Now for $I_7$
$\Rightarrow T \to T*.F$
$F \to .(E)$
$F \to .id$

from + since
$F \to .(E)$
$A \to \alpha . a \beta$
$a = ($
goto($I_7$, () = $I_4$
So, action(7, () = $S_4$

for $I_9$, $E \to E+T.$
$T \to T.*F$

for $T \to T.*F$
$A \to \alpha . a \beta$
$a = *$
goto($I_9$, *) = $I_7$ So
action(9, *) = $S_7$

for $E \to E+T.$
$A \to \alpha .$
So, follow(E) = {+, ), $}
(9, +) = $r_1$
(9, )) = $r_1$
(9, $) = $r_1$

for $I_{10}$,

$T \rightarrow \underline{T * F}$.

$A \rightarrow \alpha$.

follow(T) = (+, *, ), $)

(10, +) = $r_3$

(10, *) = $r_3$

(10, )) = $r_3$

(10, $) = $r_3$

for $I_{11}$, $F \rightarrow \underline{(E)}$.

$A \rightarrow \alpha$.

follow(F) = {+, *, ), $}

(11, +) = $r_5$

(11, *) = $r_5$

(11, )) = $r_5$

(11, $) = $r_5$

The goto transition of SLR parsing Table can be obtained using Non-terminals

as goto($I_0$, E) = $I_1$

goto($I_0$, T) = $I_2$

goto($I_0$, F) = $I_3$

goto($I_4$, E) = $I_8$

goto($I_4$, T) = $I_2$

goto($I_4$, F) = $I_3$

goto($I_6$, T) = $I_9$

goto($I_6$, F) = $I_3$

goto($I_7$, F) = $I_{10}$

The SLR parsing Table is as given:-

| State | Action | | | | | | Goto | | |
|---|---|---|---|---|---|---|---|---|---|
| | id | + | * | ( | ) | $ | E | T | F |
| | | | | | | | 1 | 2 | 3 |
| $I_0$ | $S_5$ | | | $S_4$ | | | | | |
| $I_1$ | | $S_6$ | | | | accept | | | |
| $I_2$ | | $r_2$ | $S_7$ | | $r_2$ | $r_2$ | | | |
| $I_3$ | | $r_4$ | $r_4$ | | $r_4$ | $r_4$ | | | |
| $I_4$ | $S_5$ | | | $S_4$ | | | 8 | 2 | 3 |
| $I_5$ | | $r_6$ | $r_6$ | | $r_6$ | $r_6$ | | | |
| $I_6$ | $S_5$ | | | $S_4$ | | | | 9 | 3 |
| $I_7$ | $S_5$ | | | $S_4$ | | | | | 10 |
| $I_8$ | | $S_6$ | | | $S_{11}$ | | | | |
| $I_9$ | | $r_1$ | $S_7$ | | $r_1$ | $r_1$ | | | |
| $I_{10}$ | | $r_3$ | $r_3$ | | $r_3$ | $r_3$ | | | |
| $I_{11}$ | | $r_5$ | $r_5$ | | $r_5$ | $r_5$ | | | |

SLR Parsing Table

→ →

Next page

)E→T
)T→T*F
)T→F
)F→(E)
)F→id

# Moves of LR parser

The parsing moves of "id * id + id" string is as given:-

| | Stack | Input Buffer | Parsing Action |
|---|---|---|---|
| (1) | $0 | id*id+id$ | shift, (0,id) → $S_5$ |
| (2) | $0id5 | *id+id$ | (5,*) = $r_6$ goto(0,F) = 3<br>Reduce F→id |
| (3) | $0F3 | *id+id$ | (3,*) = $r_4$ , goto(0,T) = 2<br>Reduce T→F, |
| (4) | $0T2 | *id+id$ | (2,*) = $S_7$, Shift |
| (5) | $0T2*7 | id+id$ | (7,id) = $S_5$, Shift |
| (6) | $0T2*7id5 | +id$ | (5,+) = $r_6$, goto(7,F) = 10<br>F→id, |
| (7) | $0T2*7F10 | +id$ | (10,+) = $r_3$, goto(0,T) = 2<br>T→T*F |
| (8) | $0T2 | +id$ | (2,+) = $r_2$ goto(0,E) = 1<br>E→T |
| (9) | $0E1 | +id$ | (1,+) = $S_6$, |
| (10) | $0E1+6 | id$ | (6,id) = $S_5$ |
| (11) | $0E1+6id5 | $ | (5,$) = $r_6$, goto(6,F) = 3<br>F→id |
| (12) | $0E1+6F3 | $ | (3,$) = $r_4$, goto(6,T) = 9<br>T→F, |
| (13) | $0E1+6T9 | $ | (9,$) = $r_1$, goto(0,E) = 1<br>E→E+T' |
| (14) | $0E1 | $ | (1,$) = accept |

**LALR Parser:-** In this types of parser, the lookahead Symbol is generated for each set of item. The states of SLR & LALR are always same.

In LALR Parser, the construction of LR(1) items are same, but the only difference is that In LALR parser, we will merge the two states having same First Component but different second Component.

ex:-

$S \to CC$

$C \to aC$

$C \to d$

Construct the parsing table for LALR parser:-

Ans $I_0 =$ $S' \to .S, \$$

$S \to .CC, \$$

$C \to .aC, a/d$

$C \to .d, a/d$

$I_1$ : goto $(I_0, S)$

$S' \to S., \$$

$I_2$ : goto $(I_0, C)$

$S \to C.C, \$$

$C \to .aC, \$$

$C \to .d, \$$

$I_3$ = goto $(I_0, a)$

$= C \to a.C, a/d$

$C \to .aC, a/d$

$C \to .d, a/d$

$I_4$ = goto $(I_0, d)$

$C \to d., a/d$

$I_5$ = goto $(I_2, C)$

$S \to CC., \$$

$I_6$ = goto $(I_2, a)$

$C \to a.C, \$$

$C \to .aC, \$$

$C \to .d, \$$

$I_7$ = goto $(I_2, d)$

$C \to d., \$$

$I_8$ : goto $(I_3, C)$

$C \to aC., a/d$

$I_9$ : goto $(I_6, C)$

$C \to aC., \$$

Since $I_3$ & $I_6$ have the same first Component and different second Component. So merging them

$I_{36}$ : $C \to a \cdot C, a|d|\$$
$C \to \cdot aC, a|d|\$$
$C \to \cdot d, a|d|\$$

$I_4$ & $I_7$ Combining them,

$I_{47}$ : $C \to d\cdot, a|d|\$$

$I_8$ & $I_9$ Combining them,

$I_{89}$ : $C \to aC\cdot, a|d|\$$

Parsing Table

| | Action | | | Goto | |
|---|---|---|---|---|---|
| | a | d | $ | S | C |
| 0 | $S_{36}$ | $S_{47}$ | | 1 | 2 |
| 1 | | | Accept | | |
| 2 | $S_{36}$ | $S_{47}$ | | | 5 |
| 36 | $S_{36}$ | $S_{47}$ | | | 89 |
| 47 | $r_3$ | $r_3$ | $r_3$ | | |
| 5 | | | $r_1$ | | |
| 89 | $r_2$ | $r_2$ | $r_2$ | | |

Parsing the Input String using LALR Parser

Parsing the input string "aadd" using CLR Parsing:

| Stack | Input Buffer | Action Table | Goto Table | Parsing action |
|---|---|---|---|---|
| $0 | aadd$ | action(0,a)=$S_3$ | | Shift |
| $0a3 | add$ | action(3,a)=$S_3$ | | Shift |
| $0a3a3 | dd$ | action(3,d)=$S_4$ | | Shift |
| $0a3a3d4 | d$ | action(4,d)=$r_3$ | (3,C)=8 | Reduce by C→d |
| $0a3a3C8 | d$ | action(8,d)=$r_2$ | (3,C)=8 | Reduce by C→aC |
| $0a3C8 | d$ | action(8,d)=$r_2$ | (0,C)=2 | Reduce by C→aC |
| $0C2 | d$ | action(2,d)=$S_7$ | | Shift |
| $0C2d7 | $ | action(7,$)=$r_3$ | (8,C)=5 | Reduce by C→d |
| $0C2C5 | $ | action(5,$)=$r_1$ | (0,S)=1 | Reduce by S→CC |
| $051 | $ | Accept | | |

**Example 6.9.** Consider the following given grammar and design a SLR parser table.

$$G \begin{cases} S \rightarrow AA & 1 \\ A \rightarrow aA & 2 \\ A \rightarrow b & 3 \end{cases}$$

**Solution.** The augmented grammar is

$G'$

$$\begin{aligned} S' &\to S \\ S &\to AA \\ A &\to aA \\ A &\to b \end{aligned}$$

The canonical collection of sets of LR(0) items are as follows :

$I_0$ = closure $(S' \to \cdot S)$

$= \{S' \to \cdot S$
$S \to \cdot AA$
$A \to \cdot aA$
$A \to \cdot b$
$\}$

go to $(I_0, S)$ = Closure $(S' \to S\cdot) = \{S' \to S\cdot\} = I_1$

go to $(I_0, A)$ = Closure $(S \to A\cdot A) = \{S \to A\cdot A$
$A \to \cdot aA$
$A \to \cdot b\} = I_2$

go to $(I_0, a)$ = Closure $(A \to a\cdot A) = \{A \to a\cdot A$
$A \to \cdot aA$
$A \to \cdot b\} = I_3$

go to $(I_0, b)$ = Closure $(A \to b\cdot)$
$= \{A \to b\cdot\} = I_4$

go to $(I_2, A)$ = Closure $(S \to AA\cdot) = \{S \to AA\cdot\} = I_5$

go to $(I_2, a)$ = Closure $(A \to a\cdot A) = \{A \to a\cdot A$
$A \to \cdot aA$
$A \to \cdot b$
$\}$ (same as $I_3$)

go to $(I_2, b)$ = Closure $(A \to b\cdot) = \{A \to b\cdot\}$
= (same as $I_4$)

go to $(I_3, A)$ = Closure $(A \to aA\cdot) = \{A \to aA\cdot\} = I_6$

go to $(I_3, a)$ = Closure $(A \to a\cdot A) = \{A \to a\cdot A$
$A \to \cdot aA$
$A \to \cdot b$
$\}$ = (same as $I_3$)

go to $(I_3, b)$ = Closure $(C \to b\cdot) = \{A \to b\cdot\}$
= (same as $I_4$)

$C = \{I_0, I_1, I_2, I_3, I_4, I_5, I_6\}$

We can easily find out the transition diagram as follows :

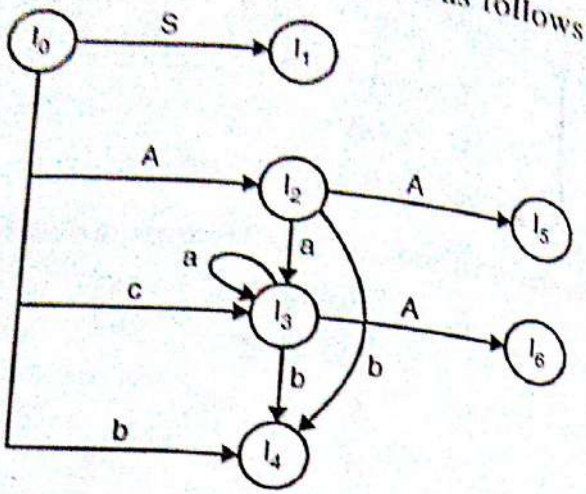



Fig. 6.26.

Production of grammar can be numbered as follows :

$$S \rightarrow AA$$
$$A \rightarrow aA$$
$$A \rightarrow b$$

Let us calculate the First and follow to construct the parsing table :

$$\text{First}(S') = \text{First}(S) = \{a, b\}$$
$$\text{First}(S) = \text{First}(AA) = \{a, b\}$$
$$\text{First}(A) = \text{First}(aA) \cup \text{First}(b)$$
$$= \{a, b\}$$
$$\text{Follow}(S') = \{\$\}$$
$$\text{Follow}(S) = \text{Follow}(S') = \{\$\}$$
$$\text{Follow}(A) = \text{First}(A) \cup \text{Follow}(S)$$
$$= \{a, b\} \cup \{\$\}$$
$$= \{a, b, \$\}$$

Now we have all the required data to construct the parsing table as follows:

| | Action table | | | Goto table | |
|---|---|---|---|---|---|
| | $a$ | $b$ | $\$$ | $S$ | $A$ |
| $I_0$ | $S_3$ | $S_4$ | | 1 | 2 |
| $I_1$ | | | accept | | |
| $I_2$ | $S_3$ | $S_4$ | | | 5 |
| $I_3$ | $S_3$ | $S_4$ | | | 6 |
| $I_4$ | $R_3$ | $R_3$ | $R_3$ | | |
| $I_5$ | | | $R_1$ | | |
| $I_6$ | $R_2$ | $R_2$ | $R_2$ | | |

**Example 6.10. Construct an SLR parsing table for the following grammar :**

$$A \rightarrow aAa/bAb/ba.$$

**Solution.** Very first let us write the augmented grammar as follows :

$$S \rightarrow A$$
$$A \rightarrow aAa$$
$$A \rightarrow bAb$$
$$A \rightarrow ba.$$

Now let us obtain the canonical collection of sets of LR(0) items, as follows :

Closure $(S \rightarrow \cdot A)$ = { $S \rightarrow \cdot A$
$A \rightarrow \cdot aAa$
$A \rightarrow \cdot bAb$
$A \rightarrow \cdot ba$
} $= I_0$

go to $(I_0, A)$ = Closure $(S \rightarrow A\cdot) = \{S \rightarrow A\cdot\} = I_1$

go to $(I_0, a)$ = Closure $(A \rightarrow a\cdot Aa) = \{A \rightarrow a\cdot Aa$
$A \rightarrow \cdot aAa$
$A \rightarrow \cdot bAb$
$A \rightarrow \cdot ba$
$\} = I_2$

go to $(I_0, b)$ = Closure $(A \rightarrow b\cdot Ab$ = $\{A \rightarrow b\cdot Ab$
$A \rightarrow b\cdot a)$ $A \rightarrow b\cdot a$
$A \rightarrow \cdot aAa$
$A \rightarrow \cdot bAb$
$A \rightarrow \cdot ba$
$\} = I_3$

go to $(I_2, A)$ = Closure $(A \rightarrow aA\cdot a) = \{A \rightarrow aA\cdot a\} = I_4$

go to $(I_2, a)$ = Closure $(A \rightarrow a\cdot Aa)$
= { $A \rightarrow a\cdot Aa$
$A \rightarrow \cdot aAa$
$A \rightarrow \cdot bAb$
$A \rightarrow \cdot ba$
} = (same as $I_2$)

go to $(I_2, b)$ = Closure $(A \rightarrow b\cdot Ab$
$A \rightarrow b\cdot a)$
= { $A \rightarrow b\cdot Ab$
$A \rightarrow b\cdot a$
$A \rightarrow \cdot aAa$

$A \rightarrow \cdot bAb$
$A \rightarrow \cdot ba$
$\}$ = (same as $I_3$)

go to $(I_3, A)$ = Closure $(A \rightarrow bA \cdot b)$ = (same as $I_5$)

go to $(I_3, a)$ = Closure $(A \rightarrow ba \cdot$
$A \rightarrow a \cdot Aa)$
= $\{ A \rightarrow ba \cdot$
$A \rightarrow a \cdot Aa$
$A \rightarrow \cdot aAa$
$A \rightarrow \cdot bAb$
$A \rightarrow \cdot ba$
$\} = I_6$

go to $(I_3, b)$ = Closure $(A \rightarrow aS \cdot a$
$A \rightarrow b \cdot a)$ = (same as $I_3$)

goto $(I_4, a)$ = Closure $(A \rightarrow aAa \cdot)$
= $\{A \rightarrow aAa \cdot\} = I_7$.

go to $(I_5, b)$ = Closure $(A \rightarrow bAb \cdot) = \{A \rightarrow bAb \cdot\} = I$

go to $(I_6, A)$ = Closure $(A \rightarrow aA \cdot a) = \{$same as $I_4)$

go to $(I_6, a)$ = Closure $(A \rightarrow a \cdot Aa)$ = (same as $I_2$)

go to $(I_6, b)$ = Closure $(A \rightarrow b \cdot Ab$
$A \rightarrow b \cdot a)$
= (same as $I_3$).

The transition diagram of DFA is given below :

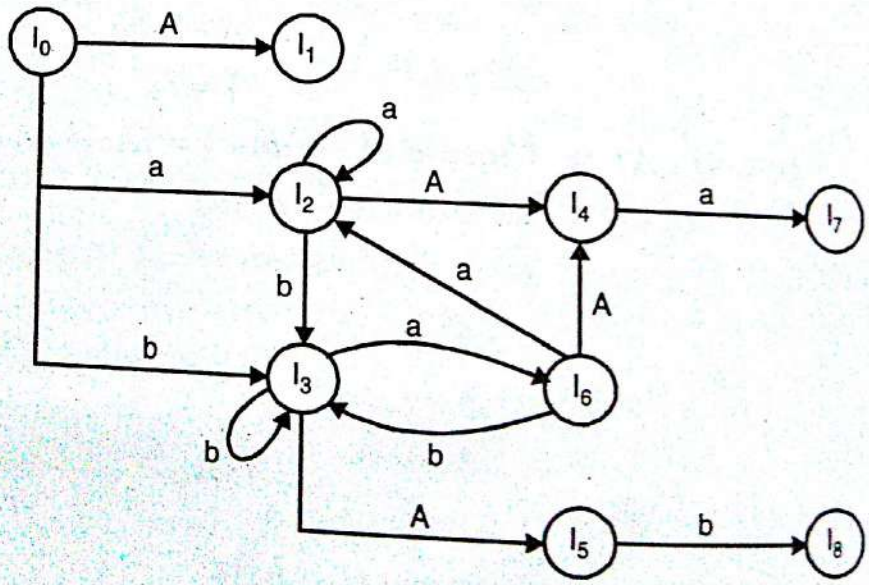



**Fig. 6.27.**

Again consider the given grammar

$A \rightarrow aAa$
$A \rightarrow bAb$
$A \rightarrow ba$

Follow $(S)$ $= \{\$\}$

Follow $(A)$ $= \{a, b, \$\}$ {from $A \to aAa, A \to bAb, S \to A$ respectively)

SLR parsing table can be designed as follows :

| | Action table | | | Goto table |
|---|---|---|---|---|
| | $a$ | $b$ | $\$$ | $A$ |
| $I_0$ | $S_2$ | $S_3$ | | 1 |
| $I_1$ | | | accept | |
| $I_2$ | $S_2$ | $S_3$ | | 4 |
| $I_3$ | $S_6$ | $S_3$ | | 5 |
| $I_4$ | $S_7$ | | | |
| $I_5$ | | $S_8$ | | |
| $I_6$ | $S_2/R_3$ | $S_3/R_3$ | $R_3$ | 4 |
| $I_7$ | $R_1$ | $R_1$ | $R_1$ | |
| $I_8$ | $R_2$ | $R_2$ | $R_2$ | |

**Example 6.11. Construct an SLR parsing table for the following grammar :**

$$S \to xAy/xBy/xAz$$
$$A \to aS/b$$
$$B \to b$$

**Solution.** Given grammar is :

$$S \to xAy/xBy/xAz$$
$$A \to aS/b$$
$$B \to b$$

Augmented grammar is as follows :

$$S' \to S$$
$$S \to xAy/xBy/xAz$$
$$A \to aS/b$$
$$B \to b$$

Now let us obtain the canonical collection of sets of LR(0) items as follows :

$I_0$ = Closure $(S' \to \cdot S)$ $= \{ S' \to \cdot S$
$S \to \cdot xAy$
$S \to \cdot xBy$
$S \to \cdot xAz$

$I_1$ = go to $(I_0, S)$ = Closure $(S' \to S\cdot\} = \{S' \to S\cdot\}$

$I_2$ = go to $(I_0, x)$ = Closure $(S \to x \cdot Ay$
$S \to x \cdot By$
$S \to x \cdot Az)$

$= \{S \to x \cdot Ay, S \to x \cdot By, S \to x \cdot Az, A \to \cdot bS, A \to \cdot b, B \to \cdot b$
$\}$

$I_3$ = go to $(I_2, A)$ = Closure $(S \to xA \cdot y$
$S \to xA \cdot z$

$= \{S \to xA \cdot y$
$S \to xA \cdot z$
$\}.$

$I_4 = \text{go to } (I_2, B) = \text{Closure } (S \to xB \cdot y)$
$= \{S \to xB \cdot y\}$

$I_5 = \text{go to } (I_2, q) = \text{Closure } (A \to b \cdot S$
$A \to b\cdot$
$B \to b\cdot)$
$= \{A \to b \cdot S$
$A \to b\cdot$
$B \to b\cdot$
$S \to \cdot xAy$
$S \to \cdot xBy$
$S \to \cdot xAz$
$\}$

$I_6 = \text{go to } (I_3, y) = \text{Closure } (S \to xAy\cdot)$
$= \{S \to xAy\cdot\}$

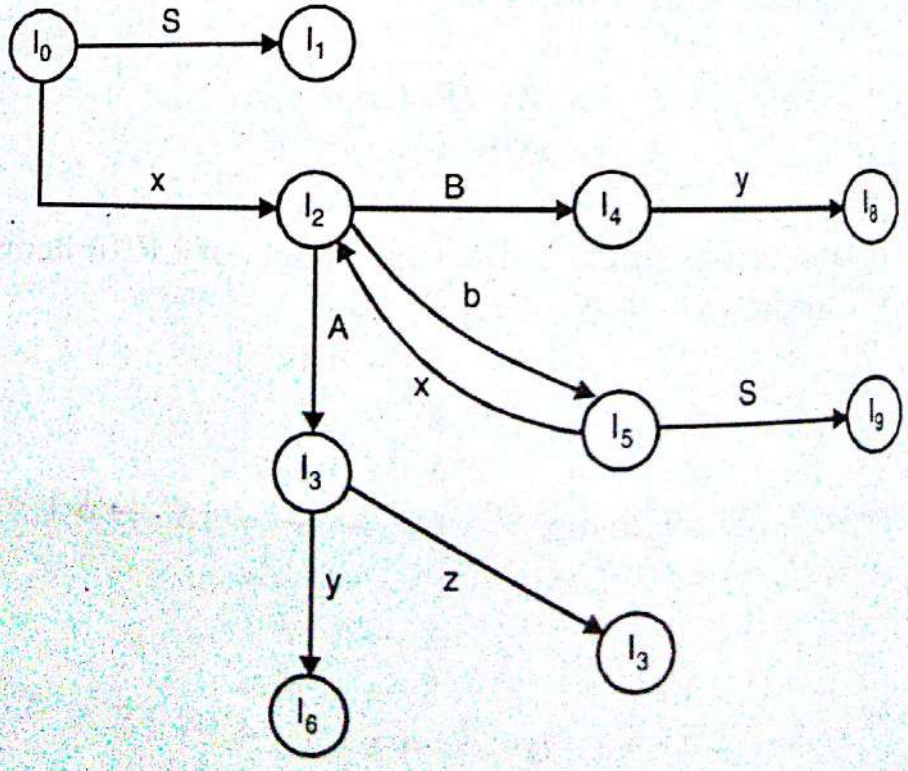
$I_7 = \text{go to } (I_3, z) = \text{Closure } (S \to xAz\cdot\}$
$= \{S \to xAz\cdot\}$

$I_8 = \text{go to } (I_4, y) = \text{Closure } (S \to xBy\cdot\}$
$= \{S \to xBy\cdot\}$

$I_9 = \text{go to } (I_5, S) = \text{Closure } (A \to bS\cdot\}$
$= \{A \to bS\cdot\}$

$\text{go to } (I_5, x) = \text{Closure } (S \to x\cdot Ay\}$
$S \to x \cdot By$
$S \to x \cdot Az$
$\} = (\text{same as } I_2)$

The transition diagram will be as follows :



$S \rightarrow \cdot bBc, \$$
$S \rightarrow \cdot dc, \$$
$S \rightarrow \cdot bda, \$$
$B \rightarrow \cdot d, a$
}

$I_1 = \text{goto}(I_0, S) = \{S' \rightarrow S\cdot, \$\}$
$I_2 = \text{goto}(I_0, A) = \{S \rightarrow B\cdot a, \$\}$
$I_3 = \text{goto}(I_0, b) = \{S \rightarrow b\cdot Bc, \$$
$S \rightarrow b\cdot da, \$$
$B \rightarrow \cdot d, c$
}

$I_4 = \text{goto}(I_0, d) = \{S \rightarrow d\cdot c, \$$
$B \rightarrow d\cdot, a$
}

$I_5 = \text{goto}(I_2, a) = \{S \rightarrow Ba\cdot, \$\}$
$I_6 = \text{goto}(I_3, A) = \{S \rightarrow bB\cdot c, \$\}$
$I_7 = \text{goto}(I_3, d) = \{S \rightarrow bd\cdot a, \$$
$B \rightarrow d\cdot, c$
}

$I_8 = \text{goto}(I_4, c) = \{S \rightarrow dc, \$\}$
$I_9 = \text{goto}(I_6, c) = \{S \rightarrow bBc\cdot, \$\}$
$I_{10} = \text{goto}(I_7, a) = \{S \rightarrow bda\cdot\$\}$

There is no set of LR(1) items in the canonical collection that have identical LR(0)–part items and that differ only in their lookaheads. So the LALR(1) parsing table for the above grammar is given below :

| | Action table | | | | | Goto table | |
|---|---|---|---|---|---|---|---|
| | *a* | *b* | *c* | *d* | *$* | *S* | *B* |
| $I_0$ | | $S_3$ | | $S_4$ | | 1 | 2 |
| $I_1$ | | | | | Accept | | |
| $I_2$ | $S_5$ | | | | | | |
| $I_3$ | | | | $S_7$ | | | 2 |
| $I_4$ | $R_5$ | | $S_8$ | | | | |
| $I_5$ | | | | | $R_1$ | | |
| $I_6$ | $S_{10}$ | | $S_9$ | | | | |
| $I_7$ | | | $R_5$ | | | | |
| $I_8$ | | | | | $R_3$ | | |
| $I_9$ | | | | | $R_2$ | | |
| $I_{10}$ | | | | | $R_4$ | | |

## EXERCISE

1. Obtain the canonical collection of sets of LR(1) items for the following grammar

$$S \rightarrow SB / Ca$$
$$B \rightarrow Bb / \in$$
$$C \rightarrow aB / c$$

2. Find the LR(1) parsing table for the following grammar :

$$T \rightarrow \text{int}$$
$$L \rightarrow L, id / id$$

3. Construct an LALR(1) parsing table for the following grammar :

$$D \rightarrow L : T$$
$$L \rightarrow L, id / id$$
$$T \rightarrow \text{integer}$$

4. Construct an SLR(1) parsing table for the following grammar :

$$S \rightarrow A )$$
$$A \rightarrow A, P / ( P, P$$
$$P \rightarrow \{\text{num, num}\}$$

5. Consider the following grammar :

$$S \rightarrow AS / b$$
$$A \rightarrow SA / a$$

(*i*) List all the LR(0) items for the above grammar.
(*ii*) Construct an NFA whose states are the LR(0) items from (*i*).
(*iii*) Is the grammar SLR ? If so construct the SLR parsing table.

6. Consider the following grammar :

$$E \rightarrow E + T / T$$
$$T \rightarrow TF / F$$
$$F \rightarrow F * / a / b$$

(*i*) Construct the SLR parsing table for this grammar.
(*ii*) Construct the LALR parsing table.

7. Show that following grammar :

$$S \rightarrow AaAb / BbBa$$
$$A \rightarrow \in$$
$$B \rightarrow \in$$

is LL(1) but not SLR(1).

8. Show that following grammar

$$S \rightarrow Aa / bAc / Bc / bBa$$
$$A \rightarrow d$$
$$B \rightarrow d$$

is LR(1) also design the LR(1) parsing table.

9. Construct an SLR parsing table for the grammar :

$$E \rightarrow E \underline{\text{sub}}\ R / E \text{ sub } E / \{E\} / c$$
$$R \rightarrow E \underline{\text{sub}}\ E / E$$

resolve the parsing action conflict.

10. Construct the LALR parsing table for the following grammar :

$$S \rightarrow AA$$
$$A \rightarrow aA / b$$

11. Discuss algorithm for computation of the sets of LR(1) items. Also show that the following grammar is LR(1) but not LALR(1)

$$S \rightarrow Aa / bAc / Bc / bBa$$
$$A \rightarrow d$$
$$B \rightarrow d$$

□□□