## LL(1) Parsing Algorithm:

Let 'a' be a symbol from the string w and x be the symbol from top of the stack while (stack is not empty)

If x is a terminal

(1) If $x = a = \$$ then accept.

(2) If $x = a \neq \$$ then

pop x from the stack

collect the next i/p symbol

(3) If $x \neq a$ then Error();

If x is a Non terminal

(1) If $M[x, a]$ is empty then Error()

(2) If $M[x, a] = x \rightarrow Y_1 Y_2 \ldots Y_k$ then

pop x from the stack

push $Y_k, Y_{k-1}, \ldots, Y_1$ into the stack

Let x be the next symbol from top of the stack.

Ex.

Parse $w = (a + a)$

| Stack | Input Buffer | action taken by the parser |
|---|---|---|
| $ S | (a+a) $ | M[S, (] = S → (L) |
| $ ) L ( | (a+a) $ ↑ | Pop off & advance the i/p ptr |
| $ ) L | a+a) $ | M[L, a] = L → SL' |
| $ ) L' S | a+a ) $ | M[S, a] = S → a |
| $ ) L' a | a+a ) $ | Pop off & advance the i/p ptr |
| $ ) L' | +a) $ | M[L', +] = L' → +SL' |
| $ ) L' S + | +a) $ | Pop off & advance the i/p ptr |
| $ ) L' S | a) $ | M(S, a) = S → a |
| $ ) L' a | a ) $ | Pop off & advance the i/p ptr |
| $ ) L' | ) $ | M[L', )] = L' → ∈ |
| $ ) | ) $ | Pop off & advance the i/p ptr |
| $ | $ | Accept |

6n) Construct LL(1) parsing tree for the given

CFG : $S \to CC$

$C \to aC / b$

Also parse the string $w = aab$

Soln: nxt

6n) Compute FIRST & FOLLOW for given CFG

$S \to ABC / Cbb / Ba$

$A \to da / BC$

$B \to g / \epsilon$

$C \to h / \epsilon$

FIRST:

FIRST (C) = $\{h, \epsilon\}$

FIRST (B) = $\{g, \epsilon\}$

FIRST (A) = $\{d\} \cup$ FIRST (B) $\cup$ FIRST (C) (because FIRST(B) contains $\epsilon$)

= $\{d, g, \cancel{h}\} = \{d, g\} \cup \{h, \cancel{\epsilon}\}$ =

so consider next symbol

so consider next symbol, but no next symbol available so take $\epsilon$

FIRST (A) = $\{d, g, h, \epsilon\}$

FIRST (S) = FIRST (ABC) $\cup$ FIRST (Cbb) $\cup$ FIRST (Ba)

FIRST (ABC) :

FIRST (A) = $\{d, g, h, \epsilon\} - \epsilon$

$\cup$ FIRST (B) = $\{g, \epsilon\} - \epsilon$

$\cup$ FIRST (C) = $\{h, \epsilon\} - \epsilon$

all contains $\epsilon$ so,

FIRST (ABC) = FIRST (A) $\cup$ FIRST (B) $\cup$ FIRST (C)

= $\{d, g, h, \epsilon\}$

FIRST (Cbb) :

FIRST (C) = {h, ϵ} - ϵ

∪ FIRST (b) = {b}

FIRST (Cbb) = FIRST (C) - ϵ ∪ FIRST (b)

= {h, b}

FIRST (Ba)

FIRST (B) = {g, ϵ} - ϵ

∪ FIRST (a) = {a}

FIRST (Ba) = FIRST (B) - ϵ ∪ FIRST (a)

= {g, a}

FIRST (S) = FIRST (ABC) ∪ FIRST (Cbb) ∪ FIRST (Ba)

= {d, g, h, ϵ} ∪ {h, b} ∪ {g, a}

**FIRST (S) = {a, b, d, g, h, ϵ}**

FOLLOW :

**FOLLOW (S) = {$}** ('S' has no occurrences in the rules as a body)

FOLLOW (A) :

$S \rightarrow \underset{\alpha}{-} \underset{B}{A} \underset{\beta}{B\,C}$

FOLLOW (B) = FIRST (β) - ϵ

FOLLOW (A) = FIRST (BC) - ϵ

= FIRST (B) - ϵ

= {g, ϵ} - ϵ

contains ϵ go to next

∪ FIRST (C)

= {h, ϵ} - ϵ

⇒ [FOLLOW (B) = FOLLOW (A)]

FOLLOW (A) = FOLLOW [S] = $

FOLLOW (A) : {g, h, $}

FOLLOW (B) :

S → ABC , C → Ba , A → BC

S → A B C
    α β β

FOLLOW (B) = FIRST (β)

FOLLOW (B) = FIRST (C) - ϵ

= {h, ϵ} - ϵ

[FOLLOW (B) = FOLLOW (A)]

= FOLLOW (B) = FOLLOW (S) = $

FOLLOW (B) = {h, $}

S → Ba
    α β β

FOLLOW (B) = FIRST (a)

= {a}

A → Bc
    α β β

FOLLOW (B) = FIRST (C)

= {h, ϵ} - ϵ

[FOLLOW (B) = FOLLOW (A)]

= FOLLOW (B) = FOLLOW (A) = {g, h, $}

FOLLOW (B) = {a, h, g, $}

FOLLOW (C) :

S → ABC  S → Cbb  A → Bc

S → A B C
    α β

[FOLLOW (B) = FOLLOW (A)]

FOLLOW (C) = FOLLOW (S) = $

$S \rightarrow \underset{\alpha}{C} \underset{B}{b} \underset{\beta}{b}$

FOLLOW(C) = FIRST(bb) = FIRST(b)

= {b}

$A \rightarrow \underset{\alpha}{B} \underset{B}{C}$

FOLLOW(C) = FOLLOW(A) = {g, h, $}

**FOLLOW(C) = {$, b, h, g}**

Soln : $S \rightarrow CC \quad C \rightarrow aC/b$

FIRST :

FIRST(S) = FIRST(CC) = FIRST(C)

FIRST(C) = FIRST(aC) ∪ FIRST(b)

**FIRST(C) = {a, b}**

**FIRST(S) = {a, b}**

FOLLOW :

FOLLOW(S) = {$}

FOLLOW(C) :

$S \rightarrow CC \qquad C \rightarrow aC/b$

$S \rightarrow \underset{\alpha}{C} \underset{B}{C} \underset{\beta}{C}$ FOLLOW(C) = FIRST(C)

= {a, b}

$S \rightarrow \underset{\alpha}{C} \underset{B}{C}$ FOLLOW(C) = FOLLOW(S)

= {$}

$C \rightarrow \underset{\alpha}{a} \underset{B}{C}$ FOLLOW(C) = FOLLOW(C)

Follow(S) = {$}  Follow(C) = {$, a, b}

Parsing table.

| M | a | b |
|---|---|---|
| S | S→CC | S→CC |
| C | C→aC | C→b |

S → CC
FIRST(CC) = {a, b}

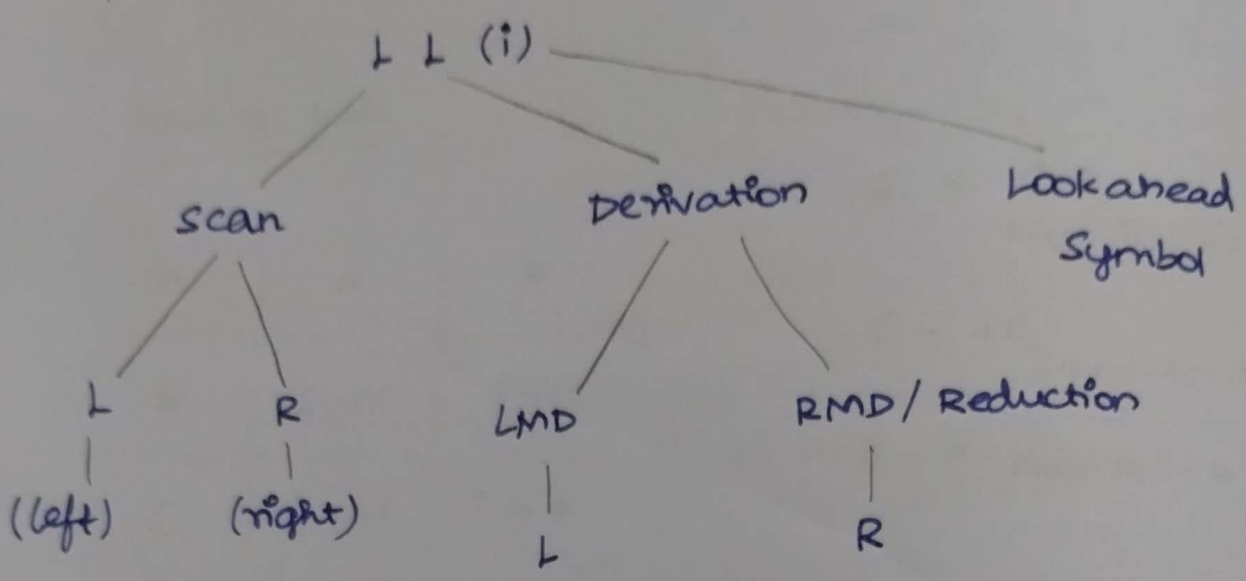
C → aC
FIRST(aC) = FIRST(a) = a

C → b  FIRST(b) = b

(aab)

| Stack | I/P buffer | Decision by parser |
|---|---|---|
| $ S | aab $ | M[S,a] = S→CC |
| $ CC | aab $ | M[C,a] = C→aC |
| $ CaC | aab $ | |
| $ CC | ab $ | M[C,a] = C→aC |
| $ CaC | ab $ | |
| $ CC | b $ | M[C,b] = C→b |
| $ Cb | b $ | |
| $ C | $ | Error. |

## LL(1) Grammar




A grammar is said to be LL(1) iff whenever $A \to \alpha / \beta$ are two distinct production rules in G the following conditions are hold :

(1) For No terminal a both $\alpha$ and $\beta$ derive the string beginning with a.

(2) Atmost one of $\alpha$ or $\beta$ can derive an empty string $\epsilon$

(3) If $\beta^* \Rightarrow \epsilon$ then $\alpha$ does not derive any strip beginning with a terminal in FOLLOW(A)

Q) Show that the given grammar is LL(1) or not

$S \rightarrow iEtS \mid iEtSeS \mid a$

$E \rightarrow b$

Soln :

Eliminating left factor,

$iEtS \rightarrow \alpha$

$iEtS\_ \;/\; iEtSeS$

$\beta_1 \qquad \beta_2$

$S \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid a$

$E \rightarrow b$

$$\begin{bmatrix} S \rightarrow iEtSS' \mid a \\ S' \rightarrow \epsilon \mid eS \\ E \rightarrow b \end{bmatrix}$$

Computation of FIRST.

FIRST (S) = FIRST ($iEtSS'$) $\cup$ FIRST (a)

= FIRST (i) $\cup$ {a}

= {i, a}

FIRST (S') = FIRST ($\epsilon$) $\cup$ FIRST (eS)

= {$\epsilon$} $\cup$ FIRST (e)

= {$\epsilon$} $\cup$ {e}

= {$\epsilon$, e}

FIRST (E) = FIRST (b)

= {b}

## Computation of FOLLOW.

FOLLOW(S) = {$} ∪ ...

S in revised production

$S \rightarrow \underset{\alpha}{iEt}\ \underset{B}{S}\ \underset{\beta}{S'}$ FOLLOW(B) = FIRST(β) - ε

FOLLOW(S) = FIRST(S') - ε

= {e}

as ε is present so taking

$S \rightarrow \underset{\alpha}{iEt}\ \underset{B}{S}\ \underset{\beta}{S'}$ FOLLOW(S) = FOLLOW(S)

**FOLLOW(S) = {$, e}**

FOLLOW(S') =

S' in rev. prod.

$S \rightarrow \underset{\alpha}{iEt}\ \underset{B}{S}\ S'$ FOLLOW(S') = FOLLOW(S)

**FOLLOW(S') = {$, e}**

FOLLOW(E) =

E in rev. prod.

$S \rightarrow \underset{\alpha}{iE}\ \underset{B}{tSS'}\ \underset{\beta}{}$ FOLLOW(B) = FIRST(β) - ε

FOLLOW(E) = FIRST(tSS') - ε

= FIRST(t) - ε

**FOLLOW(E) = {t}**

construction of parsing tree

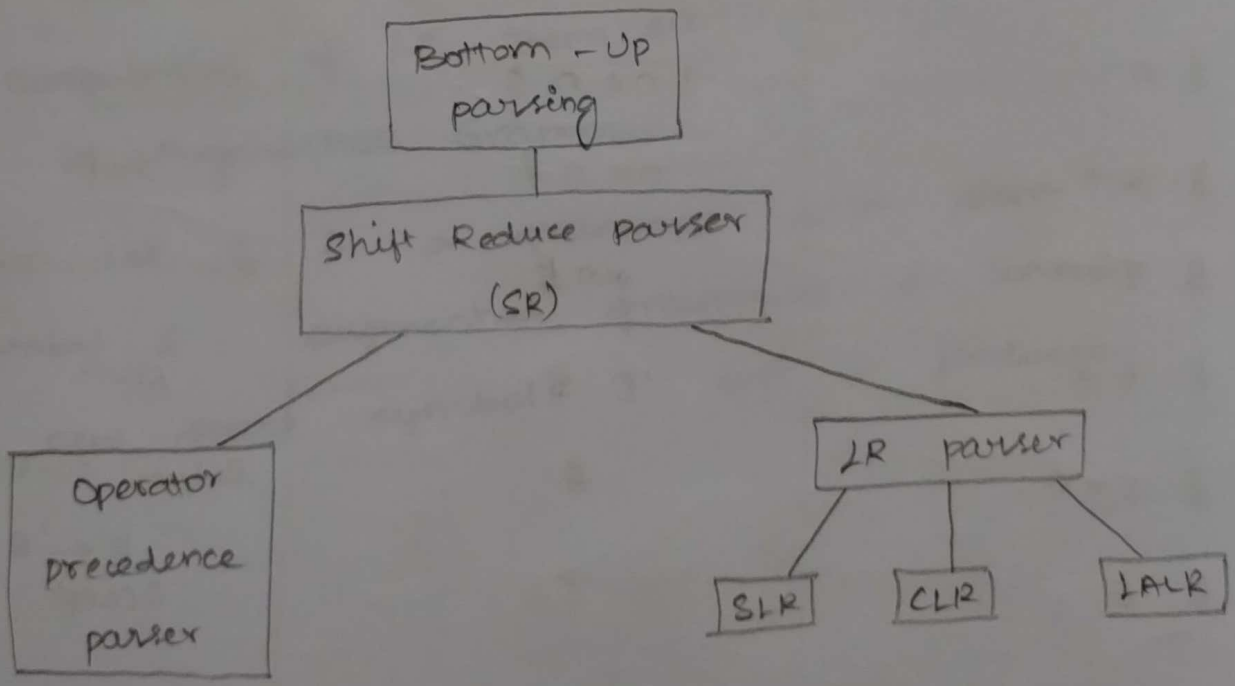
| M | i | t | e | a | b | $ |
|---|---|---|---|---|---|---|
| S | $S \to iEtSS'$ | | | $S \to a$ | | |
| S' | | | $S' \to eS$<br>$S' \to \epsilon$ | | | $S' \to \epsilon$ |
| E | | | | | $E \to b$ | |

There are multiple entries in M [S', e], so the given grammar is not LL(1) grammar

## Bottom - Up parsing :

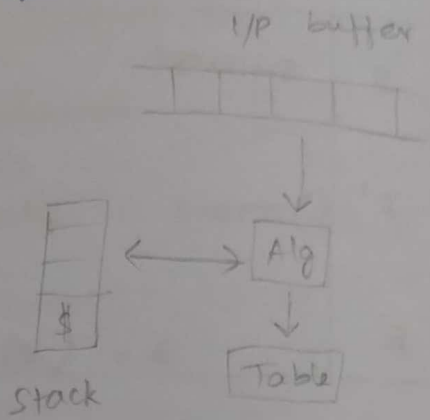
Building the parse tree from leaves to the Root. (Can parse all types of CFG)

Follows Reduction principle



## SR (Shift - Reduce) parser

Basic bottom-up parser




Actions

1) Shift
2) Reduce
3) Accept
4) Error

Qn) Show the state implementation of bottom-up parse to validate the string w.

$E \rightarrow E+n \mid E*n \mid n$

$w = n+n*n$

| Stack | I/p buffer | Action |
|---|---|---|
| $ | n+n*n $ | Shift |
| $ n | +n*n $ | Reduce E→n |
| $ E | +n*n $ | Shift |
| $ E+ | n*n $ | Shift |
| $ E+n | *n $ | Reduce E→E+n |
| $ E* | n $ | Shift |
| $ E*n | $ | Reduce E→E*n |
| $ E | $ | Accept |

## LR (K) parser

L - Left to Right scanning

R - Constructing a RMD in reverse

k - no. of input symbols used to make parsing decision

LR parser

- Simple LR (SLR)
- Canonical LR (CLR)
- Lookahead LR (LLR)

## LR Item Set

$A \rightarrow XYZ$ in G,

$A \rightarrow .XYZ$

$A \rightarrow X.YZ$

$A \rightarrow XY.Z$

$A \rightarrow XYZ.$

## Computation of LR Item set.

(1) Augmented Grammar

Let G is a grammar with start Symbol S, augmented grammar G' contains a new start symbol S' and a production

$S' \rightarrow S$

Two function

(1) Closure

(2) Goto

The closure function:

Let I be the set of items of G. the closure (I) in the set of items constructed from I by:

$R_1$: Initially add every item in I to CLOSURE (I).

$R_2$: If $A \to \alpha . B\beta$ is in CLOSURE (I) and $B \to \gamma$ is a production,

- add $B \to .\gamma$ to CLOSURE (I) if it is not already exist
- apply $R_2$ until no more new items can be added to CLOSURE (I).

Q1) Compute the collection of LR items for the following grammar:

$S \to CC$ - 1

$C \to aC$ - 2

$C \to b$ - 3

Soln,

Computation of LR Items

(1) Augmented Grammar,

$G'$ :- $S' \to S$

$S \to CC$

$C \to aC$

$C \to b$

(2) CLOSURE :

CLOSURE $(S' \to S)$ Add

$S' \to .S$ ($\alpha . B \beta$)

$S \to .CC$ (B γ) Add

$C \to .aC$ (A B γ) Add

$C \to .b$

$I_0$ taken from prod. & add

$I_0 = \{ S' \to .S,$
$S \to .CC,$
$C \to .aC,$
$C \to .b \}$

The goto function :

Let I be the collection of LR Item and x in any grammar symbol in G, GOTO (I, x) is defined to be :

if $[A \to \alpha . X \beta]$ is in I, then include CLOSURE C) of set of all items CLOSURE $(A \to \alpha X . \beta)$.

- GOTO $(I_0, S)$

$S' \to S.$ - $I_1$

- GOTO $(I_0, C)$

$S \to C.C$
$C \to .aC$
$C \to .b$
} - $I_2$

- GOTO $(I_0, a)$

$C \to a.C$
$C \to .aC$
$C \to .b$
} - $I_3$

- GOTO $(I_0, b)$

$C \to b.$ - $I_4$

- GOTO $(I_1, \text{ALL})$ = NULL

- GOTO $(I_2, S)$ = NULL

- GOTO $(I_2, C)$

$S \to CC.$ - $I_5$

- GOTO $(I_2, a)$

$C \to a.C$
$C \to .aC$
$C \to .b$
} - $I_3$

- GOTO $(I_2, b)$

$C \to b.$ - $I_4$

- GOTO $(I_3, C)$

$C \to aC.$ - $I_6$

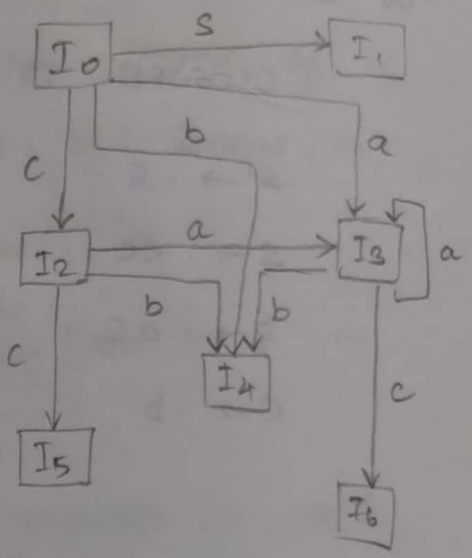
- GOTO $(I_3, a)$

$C \to a.C$
$C \to .aC$
$C \to .b$
} - $I_3$

- GOTO $(I_3, b)$

$C \to b.$ - $I_4$

LR Automata



Q) Construct LR(0) automata for the following grammar

$S \rightarrow L = R \mid R$

$L \rightarrow * R \mid id$

$R \rightarrow L$

Soln,

(1) Augmented Grammar :

G' : $S' \rightarrow S$

$S \rightarrow L = R$

$S \rightarrow R$

$L \rightarrow * R$

$L \rightarrow id$

$R \rightarrow L$

(2) CLOSURE :

CLOSURE $(S' \rightarrow S)$

$S' \rightarrow . S$

$S \rightarrow . L = R$

$S \rightarrow . R$

$L \rightarrow . * R$

$L \rightarrow . id$

$R \rightarrow . L$

$I_0 = \{ S' \rightarrow . S$

$S \rightarrow . L = R$

$S \rightarrow . R$

$L \rightarrow . * R$

$L \rightarrow . id$

$R \rightarrow . L$ $\}$

(3) GOTO

$(I_0, S)$

$S' \rightarrow S .$ $(I_1)$

$(I_0, L)$

$S \rightarrow L . = R$ $(I_2)$

$R \rightarrow L .$

$(I_0, R)$

$S \rightarrow R .$ $(I_3)$

$(I_0, *)$

$L \rightarrow * . R$ $(I_4)$

$R \rightarrow . L$ (expand)

$(I_0, id)$

$L \rightarrow id .$ $(I_5)$

$(I_2, =)$

$S \to L = \cdot R$ - $I_6$

$R \to \cdot L$ (expand)

$L \to$

$(I_4, L)$

$R \to L \cdot$ - $I_8$

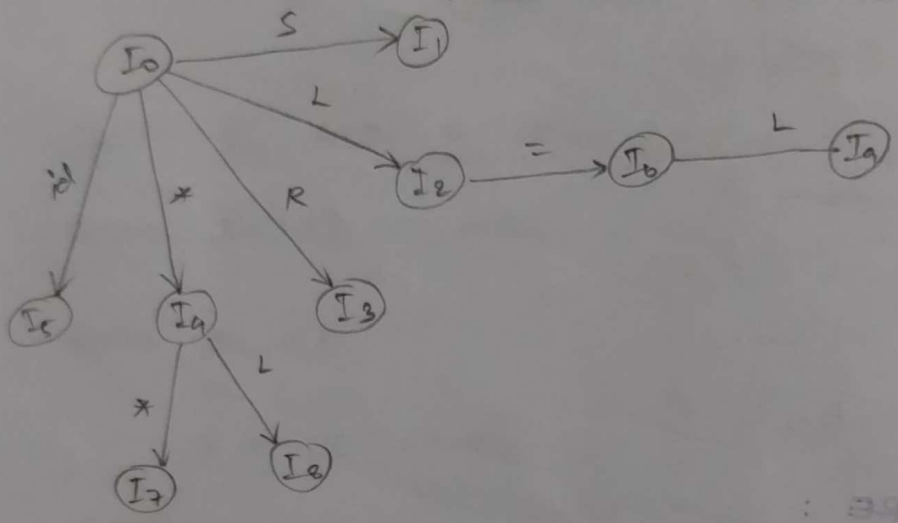
$(I_4, *)$

$L \to * R \cdot$ - $I_7$

$(I_6, L)$

$S \to L = R \cdot$ - $I_9$

LR Automata




## SLR Parser

- Requires LR (0) Items

Steps

1) Computa FIRST & FOLLOW
2) Computation of LR (0) Items set
3) SLR parsing table
4) Parse the i/p string

# SLR Parsing Table - Algorithm

Let $I = \{I_0, I_1, I_2, \ldots I_n\}$ be the collection of LR(0) Items,

ACTION PART : → (all terminal includes $)

Shift ~~Part~~ Action : If GOTO $[I_j, a] = I_k$ where 'a' is a terminal in G

| Set ACTION $[j, a] = S_k$ \| $SHIFT_k$ |
|---|

Reduce Action : If $[A \to \alpha.]$ is in $I_j$

For all 'b' in FOLLOW (A)

| Set ACTION $[j, b]$ = Reduce by $A \to \alpha$ |
|---|

Accept : If $[S' \to S.]$ is in $I_j$ (as subset or even whole)

| Set ACTION $[j, \$]$ = Accept |
|---|

Error: All the other undefined entries in the ACTION part represents Error.

GOTO PART : → (all variables)

If GOTO $[I_j, A] = I_k$ where A is a non-terminal | SET GOTO $[j, A] = k$ |
|---|

Q) Construct SLR parsing for the following grammar

$S \to CC$ ...> $\gamma_1$ - reduce using production 1

$C \to aC$ ...> $\gamma_2$

$C \to b$ ...> $\gamma_3$ also validate the string

~~ABAAB~~ $w = abaab$

Soln,

FIRST & Follow

FIRST:

$S = \{a, b\}$

$C = \{a, b\}$

Follow

$S = \{\$\}$

$C = \{\$, a, b\}$

Computation of LR(0) Item

$I_0$ : $S' \to .S$
$S \to .CC$
$C \to .ac$
$C \to .b$

$I_1$ : $S' \to S.$

$I_2$ : $S \to C.C$
$C \to .aC$
$C \to .b$

$I_3$ : $C \to a.C$
$C \to .aC$
$C \to .b$

$I_4$ : $C \to b.$

$I_5$ : $S \to CC.$

$I_6$ : $C \to aC.$

## SLR Parsing Table

| States | ACTION | | | GOTO | |
|---|---|---|---|---|---|
| | a | b | $ | S | C |
| 0 | $S_3$ | $S_4$ | | 1 | 2 |
| 1 | | | Acc | | |
| 2 | $S_3$ | $S_4$ | | | 5 |
| 3 | $S_3$ | $S_4$ | | | 6 |
| 4 | $R_3$ | $R_3$ | $R_3$ | | |
| 5 | | | | | |
| 6 | $R_2$ | $R_2$ | $R_2$ | | |

## Parsing the i/p string "abaab"

| Stack | I/P Buffer | Action |
|---|---|---|
| initially added → $ 0 | abaab $ | T[0, a] = $S_3$ |
| $ 0 a 3 | baab $ | T[3, b] = $S_4$ |
| $ 0 a 3 b 4 | aab $ | T[4, a] = $R_3$ |
| | | reduce by production 3 |

prev state
$ 0 a 3 b 4
Handle
→ replace by prod 2
C → b

$ 0 a 3 C — now push value of T[3, C] i.e. 6

| | | |
|---|---|---|
| $ 0 a 3 C 6 | aab $ | T[6, a] = $R_2$ |

| | | |
|---|---|---|
| $ 0 a 3 C 6 | C → aC | |
| $ 0 a b $ | push T[0, c] = 2 | |
| $ 0 C 2 | | |
| $ 0 C 2 | aab $ | T[2, a] = S3 |
| $ 0 C 2 a 3 | ab $ | T[3, a] = S3 |
| $ 0 C 2 a 3 a 3 | b $ | T[3, b] = S4 |
| $ 0 C 2 a 3 a 3 b 4 | $ | T[4, $] = R3 |
| $ 0 C 2 a 3 a 3 C 6 | $ | T[6, $] = R2 |
| $ 0 C 2 a 3 C 6 | $ | T[6, $] = R2 |
| $ 0 C 2 C 5 | $ | T[5, $] = R1 |
| $ 0 S 1 | $ | T[1, $] = Accept |

Q) Construct SLR parsing table for following grammar,

E → E+T | T

T → T*F | F

F → (E) | id

Soln,

Computation of LR(0) Items

(1) Augmented Grammar

G': $E' \to E$

$E \to E+T$

$E \to T$

$T \to T*F$

$T \to F$

$F \to (E)$

$F \to id$

(2) CLOSURE

Initially CLOSURE (E').

$E' \to .E$
now expand this

$E \to .E+T$
already expanding

$E \to .T$
now expand

$T \to .T*F$
already expanding

$T \to .F$
now expand

$F \to .(E)$
can expand

$F \to .id$
can expand

$I_0$

GOTO ($I_0$, E) → choose this based on what element we have after "."

go for all E placed after "." & move "." to next symbol.

$I_1$ {
- $E' \to E.$ — no expansion
- $E \to E.\underline{+}T$ — can't expand

GOTO ($I_0$, T)

$I_2$ {
- $E \to T.$ — no expansion
- $T \to T.\underline{*}F$ — can't expand

GOTO ($I_0$, F)

$I_3$ { $T \to F.$

GOTO ($I_0$, C)

$I_4$ {
- $F \to (.\underline{E})$ — now expand (from G')
- $E \to .\underline{E}+T$ — expanding
- $E \to .\underline{T}$ — expand
- $T \to .\underline{T}*F$ — expanding
- $T \to .\underline{F}$ — expand
- $F \to .(E)$ — can't expand
- $F \to .\underline{id}$ — can't expand

GOTO ($I_0$, id)

$I_5$ { $F \to id.$ — no expansion

GOTO ($I_1$, +)

$I_6$ {
- $E \to E+.\underline{T}$ — expand from $I_0$
- $T \to .T*F$
- $T \to .F$
- $F \to .(E)$
- $F \to .id$

GOTO ($I_2$, *)

$I_7$ {
- $T \to T*.\underline{F}$ — expand
- $F \to .(E)$
- $F \to .id$

GOTO ($I_4$, E)

$I_8$ {
- $F \to (E.\underline{)}$ — can't expand
- $E \to E.\underline{+}T$ — can't expand

GOTO ($I_4$, T)

$I_2$ {
- $E \to T.$
- $T \to T.*F$

GOTO ($I_4$, F)

$I_3$ { $T \to F.$

GOTO ($I_4$, C)

$I_4$ {
- $F \to (.\underline{E})$ — expand
- $E \to .E+T$
- $E \to .T$

$I_2$ { $T \to .T * F$, $T \to .F$, $F \to .(E)$, $F \to .id$ }

GOTO ($I_4$, id)

$I_5$ { $F \to id.$ }

GOTO ($I_6$, T)

$I_9$ { $E \to E + T.$, $T \to T. * F$ }

GOTO ($I_6$, F)

$I_3$ { $T \to F.$ }

GOTO ($I_6$, ()

$I_4$ { $F \to (.E)$, $E \to .E + T$, $E \to .T$, $T \to .T * F$, $T \to .F$, $F \to .(E)$, $F \to .id$ }

GOTO ($I_6$, id)

$I_5$ { $F \to id.$ }

GOTO ($I_7$, F)

$I_{10}$ { $T \to T * F.$ }

GOTO ($I_7$, ()

$I_4$ { $F \to (.E)$, $E \to .E + T$, $E \to .T$, $T \to .T * F$, $T \to .F$, $F \to .(E)$, $F \to .id$ }

GOTO ($I_7$, id)

$I_5$ { $F \to id.$ }

GOTO ($I_8$, ))

$I_{11}$ { $F \to (E).$ }

GOTO ($I_8$, +)

$I_6$ { $E \to E + .T$, $T \to .T * F$, $T \to .F$, $F \to .(E)$, $F \to .id$ }

GOTO ($I_9$, *)

$I_7$ { $T \to T * .F$, $F \to .(E)$, $F \to .id$ }

SLR parsing Table

| States | ACTION | | | | | | GOTO | | |
|---|---|---|---|---|---|---|---|---|---|
| | + | * | id | ( | ) | $ | E | T | F |
| 0 | | | $S_5$ | $S_4$ | | | 1 | 2 | 3 |
| 1 | $S_6$ | | | | | Acc | | | |
| 2 | | $S_7$ | | | $R_2$ | $R_2$ | | | |
| 3 | $R_4$ | $R_4$ | | | $R_4$ | $R_4$ | | | |
| 4 | | | $S_5$ | $S_4$ | | | 8 | 2 | 3 |
| 5 | $R_6$ | $R_6$ | | | | | | | |
| 6 | | | $S_5$ | $S_4$ | | | | 9 | 3 |
| 7 | | | $S_5$ | $S_4$ | | | | | 10 |
| 8 | $S_6$ | | | | $S_{11}$ | | | | |
| 9 | | | | | $R_1$ | $R_1$ | | | |
| 10 | $R_3$ | | | | $R_3$ | $R_3$ | | | |
| 11 | $R_5$ | $R_5$ | | | $R_5$ | $R_5$ | | | |

FIRST,

$E = T = F = \{(, id\}$

FOLLOW,

$E = \{\$, )\}$ $T = \{+, *, )\}$ $F = \{*, +, ), \$\}$

6) Check whether the grammar is SLR(1) grammar or not

$S \rightarrow L = R \mid R$

$L \rightarrow *R \mid id$

$R \rightarrow L$

$S = \{*, id\}$
$L = \{*, id\}$
$R = \{*, id\}$

$S = \{\$\}$
$L = \{=, \$\}$
$R = \{\$, =\}$

Soln,

(1) Augmented Grammar

$S' \rightarrow S$

$S \rightarrow L = R$ - 1

$S \rightarrow R$ - 2

$L \rightarrow *R$ - 3

$L \rightarrow id$ - 4

$R \rightarrow L$ - 5

(2) CLOSURE :

CLOSURE $(S' \rightarrow S)$

$S' \rightarrow . S$

$S \rightarrow . L = R$

$S \rightarrow . R$

$L \rightarrow . *R$

$L \rightarrow . id$

$R \rightarrow . L$

$I_0$

GOTO $(I_0, S)$

$I_1$ { $S' \to S.$

GOTO $(I_0, L)$

$I_2$ { $S \to L. = R$ ; $R \to L.$

GOTO $(I_0, R)$

$I_3$ { $S \to R.$

GOTO $(I_0, *)$

$I_4$ { $L \to *. \underset{ex}{R}$ ; $R \to . \underset{ex}{L}$ ; $L \to . * R$ ; $L \to . id$

GOTO $(I_0, id)$

$I_5$ { $L \to id.$

GOTO $(I_1, ALL) = NULL$

GOTO $(I_2, =)$

$I_6$ { $S \to L = . \underset{ex}{R}$ ; $R \to . \underset{ex}{L}$ ; $L \to . * R$ ; $L \to . id$

GOTO $(I_3, ALL) = NULL$

GOTO $(I_4, R)$

$I_7$ { $L \to * R.$

GOTO $(I_4, L)$

$I_8$ { $R \to L.$

GOTO $(I_5, ALL) = NULL$

GOTO $(I_6, R)$

$I_9$ { $S \to L = R.$

GOTO $(I_6, L)$

$I_8$ { $R \to L.$

GOTO $(I_4, *)$

$I_4$ { $L \to * . R$ ; $R \to . L$ ; $L \to . * R$ ; $L \to . id$

GOTO $(I_4, id)$

~~$I_5$~~ { $L \to id.$ → $I_5$

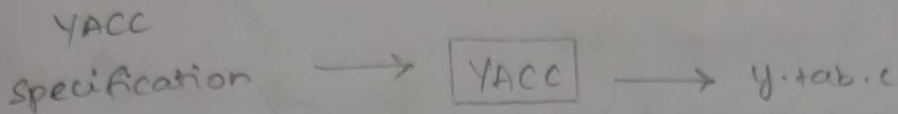
| State | ACTION | | | | GOTO | | |
|---|---|---|---|---|---|---|---|
| | = | id | * | $ | S | L | R |
| 0 | | $S_5$ | $S_4$ | | 1 | 2 | 3 |
| 1 | | | | ACC | | | |
| 2 | $S_6/R_5$ | | | $R_5$ | | | |
| 3 | $R_2$ | | | $R_2$ | | | |
| 4 | | $S_5$ | $S_4$ | | | 8 | 7 |
| 5 | $R_4$ | | | $R_4$ | | | |
| 6 | | | | | | 8 | 9 |
| 7 | $R_3$ | | | $R_3$ | | | |
| 8 | $R_5$ | | | $R_5$ | | | |
| 9 | $R_1$ | | | $R_1$ | | | |

Parser Generator

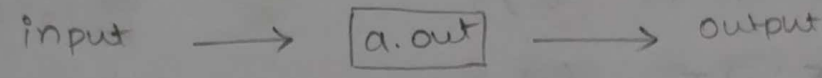
YACC - Yet Another Compiler Compiler

Tool used to design a parser

Based on LALR Parser








YACC specification - structure

definition section

%%

translation rules and actions

%%

Other routines

YACC impl. for arithmetic expression

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | id | num

head → body1 | body2 | body3 | ... bodyn

typical c program

```
head : body1    { }
     | body2    { }
     | body3    { }
       :
     | bodyn    { }
     ;
```

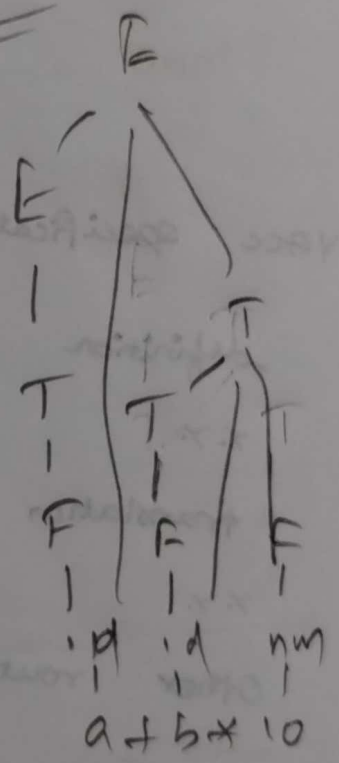
$S \rightarrow IF\ '('\ E_1\ ')'\ S_1\ ';'$
$\quad | \ IF\ '('\ E_1\ ')'\ S_1\ ELSE$
$\quad | \ \vdots$

$E_1 \rightarrow ID\ '<'\ ID$
$\quad | \ ID\ '<'\ NUM$
$\quad | \ \vdots$

Ex, for arith exp.

```
E : E '+' T    { }
  | E '-' T    { }
  | T          { }
```

ash




Lex code (.l)

```
%{
# include "y.tab.h"
%}
%%
[0-9]*      { return NUM;}
[a-zA-Z]*   { return ID;}
%%
```

if (a>b) a=10;

YACC code (.y)

```
%{
#include <stdio.h>
%}
%token ID NUM
%left + -
%left * /
%%
E : E '+' T { $$ }
  | E '-' T
  | T
  ;
T : T '*' F
  | T '/' F
  | F
  ;
F : '(' E ')'
  | ID
  | Num
  ;
%%
int main() {
  printf("Enter your expression :");
  yyparse();
  printf("valid");
  return 0;
}
```

```
int yywrap
int yyerror (char *s) {
  printf ("Syntax Error");
}
int yywrap() {
  return 1;
}
%union {
  int num;
  char ide[10];
}
```

```
> lex par.l
> yacc -d par.y
> gcc lex.yy.c y.tab.c
> ./a.out
```

Enter

```
%token <num> NUM
%token <ide> ID
```

y.s is a ID, yylval.ide;

yylval.num

a+b+10

a — 
b — 
10 — 
+ — 
+ —

## Advanced LR parsers

CLR (Canonical LR) &

LALR (Look Ahead LR)

more powerful than SLR

LR(0) Items

↓

[A → . XYZ]

Compute LR(1) Items

↓

[A → . XYZ, a]

↑ production component

↑ Look Ahead component

## Compute LR(1)

1) Augmented Grammar
2) CLOSURE ()
3) GOTO ()

## CLOSURE function

Each item of the form [A → α.β, a]

1) Add every item in I to CLOSURE (I)
2) For each item

[A → α. Bβ, a] & [B → γ]

B → . γ . b

where b = FIRST (βa)

GOTO function

For each item $[A \to \alpha . X\beta, a]$ in $I_j$

GOTO $(I_j, X)$ will add an item,

CLOSURE $([A \to \alpha X . \beta, a])$

8) construct CLR and LALR parsing table for following grammar

$S \to CC$ -- 1

$C \to aC$ -- 2

$C \to b$ -- 3 and parse string $w = abaab$

Soln,

Computation of LR(1) items

(1) Augmented Grammar

$S' \to S$

$S \to CC$

$C \to aC$

$C \to b$

CLOSURE $(S' \to S)$

$I_0$

$S' \to . S, \$$ → initially $

$S \to . CC, \$$ = FIRST($\beta a$) FIRST($)

$C \to . aC, a/b$ FIRST(CC) FIRST(C) = {a, b}

$C \to . b, a/b$

same as above

GOTO $(I_0, S)$

$I_1$ $S' \to S.$

GOTO $(I_0, C)$

$I_2$

$S \to C . C, \$$

$C \to . aC, \$$

$C \to . b, \$$

GOTO $(I_0, a)$

$I_3$

$C \to a . C, a/b$

$C \to . aC, a/b$

$C \to . b, a/b$

GOTO $(I_0, b)$

$I_4$ { $C \to b.$ , a/b

GOTO $(I_2, C)$

$I_5$ { $S \to CC.$ , \$

GOTO $(I_2, a)$

$I_6$ { $C \to a \cdot C$ , \$ (α = a, B = C, β = ε, a = \$)
$C \to .aC$ , \$ FIRST(Ba)
$C \to .b$ , \$

GOTO $(I_2, b)$

$I_7$ { $C \to b.$ , \$

GOTO $(I_3, C)$

$I_8$ { $C \to aC.$ , a/b

GOTO $(I_3, a)$

$I_3$ { $C \to a \cdot C$ , a/b (α = a, B = C, β = ε) FIRST(βa)
$C \to .aC$ , a/b
$C \to .b$ , a/b

GOTO $(I_3, b)$

$I_4$ { $C \to b.$ , a/b

GOTO $(I_6, C)$

$I_9$ { $C \to aC.$ , \$

GOTO $(I_6, a)$

$I_6$ { $C \to a \cdot C$ , \$ (α = a, B = C, β = ε, a = \$)
$C \to .aC$ , \$ FIRST(
$C \to .b$ , \$

GOTO $(I_6, b)$

$I_7$ { $C \to b.$ , \$

For constructing parsing table, ACTION - Reduction differs from SLR is

In SLR

GOTO $(I_j, A)$

$I_i$ {
↳ row  Follow(A) = column
original eq - val

In CLR,

GOTO $(I_j, A)$

$I_i$ { ⊔ , a
↳ row  ↓ column
original eq - val

## CLR parsing table

| State | ACTION | | | GOTO | |
|---|---|---|---|---|---|
| | a | b | $ | S | C |
| 0 | $S_3$ | $S_4$ | | 1 | 2 |
| 1 | | | Acc | | |
| 2 | $S_6$ | $S_7$ | | | 5 |
| 3 | $S_3$ | $S_4$ | | | 8 |
| 4 | $R_3$ | $R_3$ | | | |
| 5 | | | $R_1$ | | |
| 6 | $S_6$ | $S_7$ | | | 9 |
| 7 | | | $R_3$ | | |
| 8 | $R_2$ | $R_2$ | | | |
| 9 | | | $R_2$ | | |

## LALR parsing table

From the collection of LR(1) Items,

The set $I_3$ & $I_6$

$I_3$: $C \to a.b, a/b$

$C \to .aC, a/b$

$C \to .b, a/b$

$I_6$: $C \to a.b, \$$

$C \to .aC, \$$

$C \to .b, \$$

$\therefore$ $I_3$, $I_6$ are merged to form a new set

$I_{36}$ as,

$I_{36}$ :

$C \to a \cdot C$ , $a|b|\$$

$C \to \cdot aC$ , $a|b|\$$

$C \to \cdot b$ , $a|b|\$$

The set $I_4$ , $I_7$

$I_4$ : $C \to b \cdot$ , $a|b$

$I_7$ : $C \to b \cdot$ , $\$$

are merged as $I_{47}$

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

The set $I_8$ , $I_9$

$I_8$ : $C \to aC \cdot$ , $a|b$

$I_9$ : $C \to aC \cdot$ , $\$$

are merged as $I_{89}$

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

LALR parsing table :

| State | ACTION | | | GOTO | |
|---|---|---|---|---|---|
| | a | b | $ | S | C |
| 0 | $S_{36}$ | $S_{47}$ | | 1 | 2 |
| 1 | | | Acc | | |
| 2 | $S_{36}$ | $S_{47}$ | | | 5 |
| 36 | $S_{36}$ | $S_{47}$ | | | 89 |
| 47 | $R_3$ | $R_3$ | $R_3$ | | |
| 5 | | | $R_1$ | | |
| 89 | $R_2$ | $R_2$ | $R_2$ | | |

| LR(1) Stack | I/P buffer | Action |
|---|---|---|
| $0 | a baab $ | T[0,a] = S36 |
| $0a 36 | baab $ | T[36,b] = S47 |
| $0a36b47 | aab $ | T[47,a] = R3 |
| $0a36C<br>T[36,C] = 89 | | |
| $0b36C89 | aab $ | T[89,a] = R2 |
| $0C<br>T[0,C] = 2 | aab $ | T[2,a] = S36 |
| $0C2 | ab $ | T[36,a] = S36 |
| $0C2a36 | b $ | T[36,b] = S47 |
| $0C2a36a36b47 | $ | T[47,$] = R3 |
| $0C2a36<br>a36C<br>T[36,C] = 89 | $ | T[89,$] = R2 |
| $0C2a36a36C89 | | |
| $0C2a36C<br>T[36,C] = 89 | $ | T[89,$] = R2 |
| $0C2b36C89 | $ | T[5,$] = R1 |
| $0C2C5 | $ | T[1,$] = Accept ✓ |
| $0S1 | | |

aboab

CLR

| Stack | I/p buffer | Action |
|---|---|---|
| $ 0 | a b a a b $ | T[0, a] = S3 |
| $ 0 a 3 | b a a b $ | T [3, b] = S4 |
| $ 0 a 3 b 4 | a a b $ | T [4, a] = R3 |
| $ 0 a 3 C 8 | a a b $ | T [8, a] = R2 |
| $ 0 C 2 | a a b $ | T [2, a] = S6 |
| $ 0 C 2 a 6 | a b $ | T [6, a] = S6 |
| $ 0 C 2 a 6 a 6 | b $ | T [6, b] = S7 |
| $ 0 C 2 a 6 a 6 b 7 | $ | T [7, $] = R3 |
| $ 0 C 2 a 6 a 6 C 9 | $ | T [9, $] = R2 |
| $ 0 C 2 a 6 C 9 | $ | T [9, $] = R2 |
| $ 0 C 2 C 5 | $ | T [5, $] = R1 |
| $ 0 S 1 | $ | T [1, $] = Accept |

20 mark

6) Prove that foll CFG is not SLR(1) but CLR(1) if so, build CALR parsing table for grammar & also parse the string w = *id = *id

S → L = R | R

L → *R | id

R → L

| FIRST | Follow |
|---|---|
| S = {*, id} | S = {$} |
| L = {*, id} | L = {=, $} |
| R = {*, id} | R = {=, $} |

Soln,

not SLR(1) is prev. Solved

(1) Augmented Grammar

$S' \rightarrow S$

$S \rightarrow L = R$ - 1
$S \rightarrow R$ - 2
$L \rightarrow * R$ - 3
$L \rightarrow id$ - 4
$R \rightarrow L$ - 5

(2) CLOSURE ($S' \rightarrow S$)

$I_0$
- $S' \rightarrow . S$ , $
- $S \rightarrow . L = R$ , $
- $S \rightarrow . R$ , $
- $L \rightarrow . * R$ , =/$
- $L \rightarrow . id$ , =/$
- $R \rightarrow . L$ , $

GOTO ($I_0$, S)

$I_1$ { $S' \rightarrow S.$ , $

GOTO ($I_0$, L)

$I_2$
- $S \rightarrow L . = R$ , $
- $R \rightarrow L.$ , $

GOTO ($I_0$, R)

$I_3$ { $S \rightarrow R.$ , $

GOTO ($I_0$, *)

$I_4$
- $L \rightarrow * . R$ , =/$
- $R \rightarrow . L$ , =/$
- $L \rightarrow . * R$ , =/$
- $L \rightarrow . id$ , =/$

GOTO ($I_0$, id)

$I_5$ { $L \rightarrow id.$ , =/$

GOTO ($I_2$, =)

$I_6$
- $S \rightarrow L = . R$ , $
- $R \rightarrow . L$ , $
- $L \rightarrow . * R$ , $
- $L \rightarrow . id$ , $

GOTO ($I_4$, L)

$I_7$ { $R \rightarrow L.$ , =/$

GOTO ($I_4$, R)

$I_8$ { $L \rightarrow * R.$ , =/$

GOTO ($I_4$, *)

$I_4$
- $L \rightarrow * . R$ , =/$
- $R \rightarrow . L$ , =/$
- $L \rightarrow . * R$ , =/$
- $L \rightarrow . id$ , =/$

GOTO ($I_4$, id)

$I_5$ { $L \rightarrow id.$ , =/$

GOTO ($I_6$, L)

$I_9$ { $R \rightarrow L.$ , $

GOTO ($I_6$, R)

$I_{10}$ { $S \rightarrow L = R.$ , $

GOTO (I6, *)

$I_{11}$ { L → * . R , $
R → . L, $, L → . * R, $, L → . id, $

GOTO (I6, id)

$I_{12}$ { L → id . , $

GOTO ($I_{11}$, id)

$I_{12}$ { L → id . , $

GOTO ($I_{11}$, R)

$I_{13}$ { L → * R . , $

GOTO ($I_{11}$, L)

$I_4$ { R → L . , $

GOTO ($I_{11}$, *)

$I_{11}$ { L → * . R , $
R → . L , $
L → . * R, $ L → . id , $

Parsing table (CLR)

| | ACTION | | | | GOTO | | |
|---|---|---|---|---|---|---|---|
| States | = | * | id | $ | S | L | R |
| 0 | | S4 | S5 | | 1 | 2 | 3 |
| 1 | | | | Acc | | | |
| 2 | S6 | | | $R_5$ | | | |
| 3 | | | | $R_2$ | | | |
| 4 | | S4 | S5 | | | 7 | 8 |
| 5 | $R_4$ | | | $R_4$ | | | |
| 6 | | $S_{11}$ | $S_{12}$ | | | 9 | 10 |
| 7 | $R_5$ | | | $R_5$ | | | |
| 8 | $R_3$ | | | $R_3$ | | | |
| 9 | | | | $R_5$ | | | |
| 10 | | | | $R_1$ | | | |
| 11 | | $S_{11}$ | $S_{12}$ | | | 9 | 13 |
| 12 | | | | $R_4$ | | | |
| 13 | | | | $R_3$ | | | |

Merge states with common 1st values :

$I_A$ & $I_{11}$

$L \to *.R$ , =/$

$R \to .L$ , =/$

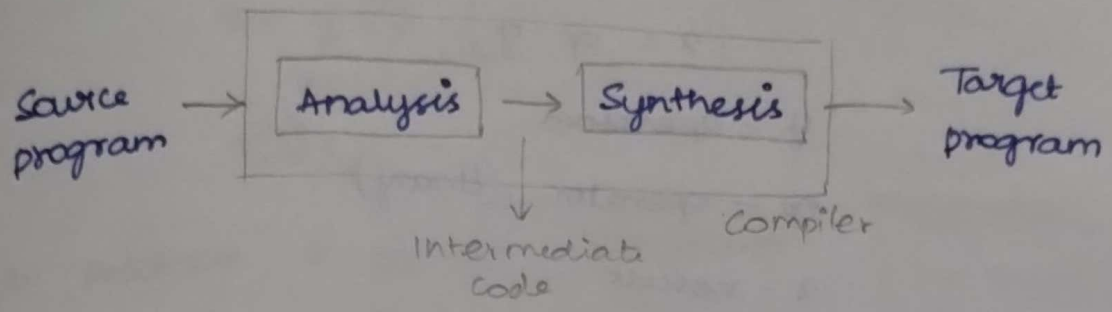
$L \to .*R$ , =/$

$L \to .id$ , =/$

$I_5$ & $I_{12}$ :

$L \to id.$ , =/$

# Intermediate Code Generation




## Benefits of ICG

1. Portability - Producing a target code for different architecture
2. Retargeting
3. Perform pre-optimization (m/c independent)

## Intermediate Languages

Intermediate representations can be

1. POSTFIX notation
2. Syntax tree
3. Three addr. code (TAC) statements

## Types of TAC statements

1. Assignment statement of the form

$$x = y \text{ op } z$$

$y, z$ - operands

$x$ - result

op - operator (binary)

2. Assignment

$x = op\ y$

y - operand

op - operator (Unary)

x - result

3. Copy statement of the form

$x = y$

4. Unconditional jump

goto L

L - label

5. Conditional jump

if E then goto $L_1$

goto $L_2$

6. Procedural call statement

$P\ (n_1, n_2, \ldots n_k)$

can be represented as

param $n_1$,

param $n_2$,

⋮

param $n_k$

Call $P, k$

7. Indexed statements

$x = y[i]$

$x[i] = y$

b. Address & pointer statements

$x = \&y$ ; $x = *y$

$*x = y$

8) Write the 3 addr. code strtr for the following expressions.

assignment - right to left
operations - left to right

i) $x = a + b * c$

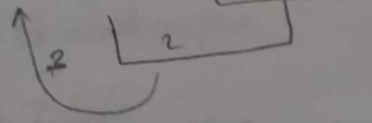
ii) $x = (a*a) + (b*b) + (2*a*b)$

iii) $s = (b*b) - 4 * a * c$

iv) $y = -b + c * -b + c$

Soln,

i) $x = a + b * c$

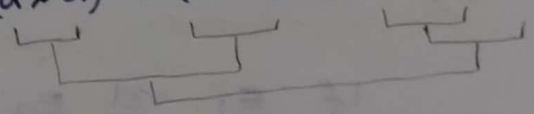



$t_0 = b * c$

$t_1 = a + t_0$

$x = t_1$

ii) $x =$
$(a*a) + (b*b) + (2*a*b)$



$t_0 = a * a$

$t_1 = b * b$

$t_2 = 2 * a$

$t_3 = t_2 * b$

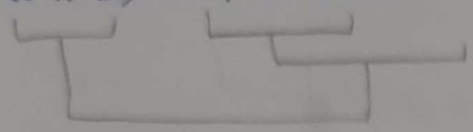
$t_4 = t_0 + t_1$

$t_5 = t_4 + t_3$

$x = t_5$

iii) $s =$

$(b * b) - 4 * a * c$



$t_0 = b * b$

$t_1 = 4 * a$

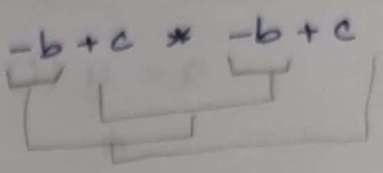
$t_2 = t_1 * c$

$t_3 = t_0 - t_2$

$s = t_3$

iv) $y =$

$-b + c * -b + c$



$t_0 = \text{UMINUS } b$

$t_1 = \text{UMINUS } b$

$t_2 = c * t_1$

$t_3 = t_0 + t_2$

$t_4 = t_3 + c$

$y = t_4$

## TAC statement for Boolean Expressions

Result of Expression → True (1)
Result of Expression → False (0)

$E \rightarrow E_1 \text{ rel } E_2 \mid id \mid num$

$rel \rightarrow < \mid > \mid == \mid <= \mid >= \mid != \mid \not< \mid \not>$

### TAC Statement

If $E_1$ rel $E_2$ then goto True label
goto False label

$(a < b)$

TAC Stmt:

If $a < b$ then goto $L_1$
goto $L_2$

(a>b) && (a>c)

If a>b then goto L1
goto Lexit

L1: If a>c then goto Ltrue
goto Lexit

(a==b) || (c>b)

If a==b then goto Ltrue
goto L1

L1: If c>b then goto Ltrue
goto Lexit

(a<b) && (b!=c) || (c<=d)

L1 L2

If (a<b) then goto L1
goto L2

L1: If (b!=c) then goto Ltrue
goto L2

L2: If (c<=d) then goto Ltrue
goto Lexit

→ precedence :) not, and, or

## Implementation of TAC statements

Three ways

- Quadruple
- Triple
- Indirect triple

### Quadruple

Each TAC stmt is repr. as record with 4 fields.

op, arg1, arg2 and result

op - operator
arg1, arg2 - operands
result - result of comp.

$x = a + b * 10$

$t_0 = b * 10$

$t_1 = a + t_0$

$x = t_1$

Quadruple

| op | arg1 | arg2 | result |
|---|---|---|---|
| * | b | 10 | $t_0$ |
| + | a | $t_0$ | $t_1$ |
| = | $t_1$ | - | x |

## Triple Implementation

3 fields

op - operator

arg1, arg2 - operands

Triple

| | op | arg1 | arg2 |
|---|---|---|---|
| (0) | * | b | 10 |
| (1) | + | a | [0] |
| (2) | = | [1] | |

## Indirect Triple

List of ptrs to a triple structure

| Stmt no | Addr |
|---|---|
| (0) | 100 |
| (1) | 101 |
| (2) | 102 |

Triple

| | op | arg1 | arg2 |
|---|---|---|---|
| 100 | * | b | 10 |
| 101 | + | a | [0] |
| 102 | = | [1] | |
| ⋮ | | | |
| 150 | | | |

Q) $S = n * s_1 * (a+b) / (x-y)$

Compute TAC Stmt & Implement it by 3 ways (Quadruple, Triple, Indirect triple)

Soln,

$S =$

$n * S_1 * (a+b) / (a-y)$

$t_0 = a+b$

$t_1 = a-y$

$t_2 = n * S_1$

$t_3 = t_2 * t_0$

$t_4 = t_3 / t_1$

$S = t_4$

Quadruple

| Op | arg1 | arg2 | result |
|---|---|---|---|
| + | a | b | $t_0$ |
| - | a | y | $t_1$ |
| * | n | $S_1$ | $t_2$ |
| * | $t_2$ | $t_0$ | $t_3$ |
| / | $t_3$ | $t_1$ | $t_4$ |
| = | $t_4$ | - | S |

Triple

| | op | arg1 | arg2 |
|---|---|---|---|
| (0) | + | a | b |
| (1) | - | a | y |
| (2) | * | n | $S_1$ |
| (3) | * | [2] | [0] |
| (4) | / | [3] | [1] |
| (5) | = | [4] | |

→ same but with diff. addresses

Indirect Triple

| | op | arg1 | arg2 |
|---|---|---|---|
| (100) | + | a | b |
| (101) | - | a | y |
| (102) | * | n | $S_1$ |
| (103) | * | [2] | [0] |
| (104) | / | [3] | [1] |
| (105) | = | [4] | |

| Stmt no | Addr |
|---|---|
| (0) | 100 |
| (1) | 101 |
| (2) | 102 |
| (3) | 103 |
| (4) | 104 |
| (5) | 105 |