**Yash Vinayvanshi**
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

NAME : YASH VINAYVANSHI

PROGRAM NAME : B. TECH (COMPUTER ENGINEERING)

SEMESTER : 6

EXAMINATION ROLL. NO. : 19BCS081

UNIQUE PAPER CODE : CEN - 603

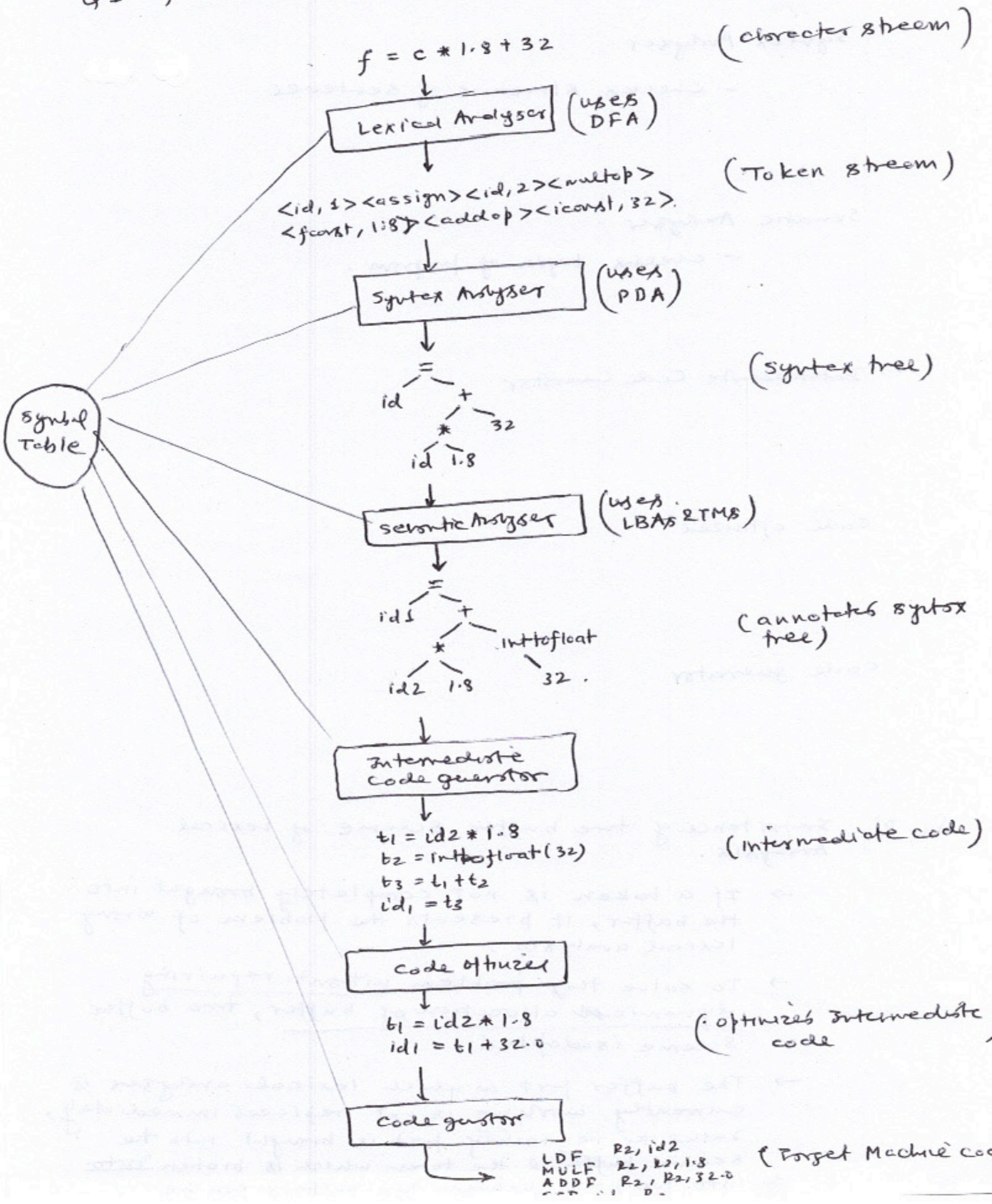
PAPER TITLE : COMPILER DESIGN

NO. OF PAGES :

DATE OF EXAM : 25/05/2022

TIME OF EXAM : 10AM - 1PM

Q1 a) PHASES OF COMPILER



Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Phase 1 : Lexical Analysis.

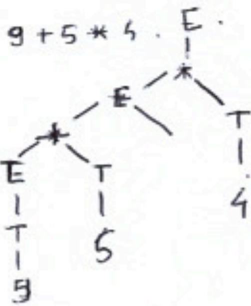
→ scans the source code character by character and groups these into tokens.
→ identifies lexical units in src code
→ classify lexical units into classes like constants, keywords etc.
→ identify a token not part of defined language
→ Enters identifiers in symbol table
→ removes comments & blank spaces.

Phase 2 : syntax analysis.

→ obtains tokens from lexical analyser
→ checks by parsing with language's grammar if the program is syntactically (structurally) correct or not.
→ reports syntax errors.
→ constructs the hierarchical structure of pgm called parse tree

eg $E \rightarrow E+T \mid E*F \mid T$
$T \rightarrow digit$

9+5*4.




Phase 3 : semantic Analysis.

→ checks if program is semantically (meaning) is correct.
→ Store type information gathered in symbol table or syntax tree.
→ Performs type checking (compatibility)

Phase 4 : Intermediate code generation.

→ generates Intermediate code from semantic representation of src pgm using SDD or SDTs
→ holds the values of attributes computed during translation

Phase 5 : code optimization.

→ To reduce redundancies and optimise code generated in ICG.
→ machine independent optimization at this stage.
- common subexpression elimination using DAGs.
- dataflow & control flow analysis for global optimiz.
- etc.

Phase 6 : code generation

- produces machine specific target code from intermediate code.
- may produce optimizations like
  - efficient register allocation
  - replacing instructions to faster equivalent instructions in instruction set

  ⋮

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q1b (ii) Sentinel Buffering Algorithm.

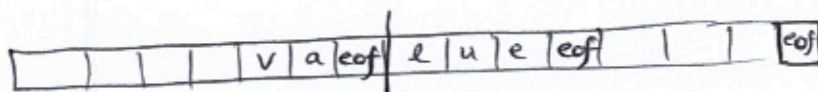
```
Algoritm : Lookahead code with sentinels.
1. forword = forword + 1.      <- one test for next character
2. if forword = eof then
3.      if forword at the end of first half then
4.           reload second half;
5.           forword = forword + 1.
6.      elseif forword at end of second half then
7.           reload first half;
8.           move forword to beginning of first half.
9.      else
10.          terminate lexical Analysis.
```

explanation

- In two buffer scheme, if the forword pointer is about to move the halfway mark, the right half is filled with new i/p characters. (to continue evaluation of incomplete identifier in buffer.)
- If forword pointer is about to move past the right end of buffer, the left half is filled with new character & forword pointer wraps around to begining of buffer
- if forword ptr ptrs to eof character, terminate ~~to continue~~.
- sentinels eof are used to mark ends of buffers & end of file & if eof is encountered test condition to check if at end of buffer or at end of file.

Advantage of sentinel Agoritum.

- Most of the time, the code performs only one test to check whether forword ptr ptrs to eof.
- only when eof is detected do we proceed to find if eof is end of buffer or end of file
- The conventional Agoritum did two tests per character to seperately identify if its end of file eof ~~for~~ or end of file



Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q2b) (i) Given grammar

$X \rightarrow uDm \mid y \mid w$.
$D \rightarrow EXF$
$E \rightarrow \epsilon$
$F \rightarrow x. \mid \epsilon$.

Rules to compute First().

1. if X is a terminal → FIRST(X) = {X}.
2. if X is nonterminal & $X \rightarrow Y_1 Y_2, \ldots Y_k$, then place 'a' in FIRST(X) if for some i, 'a' is in FIRST(Y) and ε is in all of first $(Y_1) \ldots$ FIRST$(Y_{i-1})$ ie $Y_1 \ldots Y_{i-1} \overset{*}{\Rightarrow} \varepsilon$,
   If ε is in FIRST$(Y_j)$ for all j = 1 to k then add ε to FIRST(X).
3. if $X \rightarrow \epsilon$ is a production, then add ε to FIRST(X).

Rules to compute FOLLOW().

1. Place \$ in Follow(S) where S is start symbol
   (\$ is i/p right end marker)
2. if there's a production $A \rightarrow \alpha B \beta$, then everything in FIRST(B) except ε is in FOLLOW(B).
3. if there's a production $A \rightarrow \alpha B$, or a production $A \rightarrow \alpha B \beta$ where First(β) contains ε, then everything in Follow(A) is in Follow(B).

~~FIRST~~

FIRST(X) = FIRST(u) ∪ FIRST(y) ∪ FIRST(w) = {u, y, w
FIRST (E) = FIRST (ε) = {ε}
FIRST (F) = FIRST (x) ∪ FIRST (ε) = {x, ε}
FIRST (D) = FIRST (E).
but if E → ε then
FIRST(X) = {u, y, w, ε}.

FOLLOW.

$\text{Follow}(X) = \text{FIRST}(F) \cup \{\$\}$

but if $F \to \epsilon$ then $\text{Follow}(D)$ $= \{m, x, \$\}$

$\text{Follow}(D) = \text{FIRST}(m) = \{m\}$.

$\text{Follow}(E) = \text{FIRST}(X) - \{\epsilon\} = \{u, y, \omega\}$.

$\text{Follow}(F) = \text{Follow}(D) = \{m\}$

Finally,

$\text{FIRST}(X) = \{u, y, \omega\}$

$\text{FIRST}(D) = \{u, y, \omega, \epsilon\}$.

$\text{FIRST}(E) = \{x, \epsilon\}$

$\text{FIRST}(F) = \{\epsilon\}$.

$\text{Follow}(X) = \{m, x, \$\}$.

$\text{Follow}(D) = \{m\}$.

$\text{Follow}(E) = \{u, y, \omega\}$

$\text{Follow}(F) = \{m\}$.

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

(ii) Designing LL(1) parser

given grammar

$$G: X \Rightarrow uDm \mid y \mid \omega .$$
$$D \rightarrow EXF$$
$$E \rightarrow \epsilon$$
$$F \rightarrow x \mid \epsilon$$

(i) Making grammar suitable for LL(1) ~~parsing~~ parser.

**Step 1** : removing left recursion
ie productions of type $A \rightarrow A\alpha \mid \beta \quad \alpha, \beta \in (V \cup T)$
are replaced with $A \rightarrow \beta A'$
$A' \rightarrow \alpha A' \mid \epsilon$ .

There is no left recursion in this grammar.

**Step 2** Application of left factoring.
ie productions of type $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \mid \alpha\beta_3 \mid \ldots$
are replaced with $A \rightarrow \alpha A'$
$A' \rightarrow \beta_1 \mid \beta_2 \mid \beta_3$ .

Left factoring cannot be applied in this grammar.

(ii) First() & Follow() sets already calculated.

(iii) Construction of LL(1) parsing table.

Input : Grammar G.
Output : parsing table M.

METHOD.
1. for each production $A \rightarrow \alpha$ of grammar do.
2. for each terminal a in FIRST ($\alpha$),
add $A \rightarrow \alpha$ to M[A, a].
3. if $\epsilon$ is in FIRST ($\alpha$), add $A \rightarrow \epsilon$ to M[A, b] for each terminal b in FOLLOW(A)
If $\epsilon$ is in FIRST ($\alpha$) & \$ is in FOLLOW (A)
add $A \rightarrow \alpha$ to M [A, \$].
4. Make each of undefined entries of M "error"

| | u | m | y | ω | x | $ |
|---|---|---|---|---|---|---|
| X | $X = uDm$ | | $X \to y$ | $X \to \omega$ | | |
| D | $D \to ExF$ | | $D \to ExF$ | $D \to ExF$ | | |
| E | $E \to \epsilon$ | | $E \to \epsilon$ | $E \to \epsilon$ | | |
| F | | $F \to \epsilon$ | | | $F \to x$ | |

FIRST(RHS)

1. $X \to uDm \rightsquigarrow$ FIRST(uDm) $= \{u\}$.
2. $X \to y \rightsquigarrow$ FIRST(y) $= \{y\}$
3. $X \to \omega \rightsquigarrow$ FIRST(ω) $= \{\omega\}$.
4. $D \to ExF \rightsquigarrow$ FIRST(ExF) $= \{u, y, \omega\}$.
5. $F \to x \rightsquigarrow$ FIRST(x) $= \{x\}$.

~~6 $F \to \epsilon$.~~

5. $E \to \epsilon \rightsquigarrow$ FOLLOW(E) $= \{u, y, \omega\}$
6. $F \to \epsilon$. $\rightsquigarrow$ FOLLOW(F) $= \{m\}$.

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS 081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q2C) Calculation of set of LR(1) items for CLR parser.

procedure CLOSURE(I)
begin
repeat.
for each item $[A \to \alpha . B\beta , a]$ in I, each production $B \to \gamma$ and each terminal b in FIRST($\beta$ a) s.t $[B \to . \gamma, b]$ is not in I
do.
add $[B \to . \gamma, b]$ to I.
until no more items can be added to I.
return I.

end.

procedure GOTO(I, X).
begin let J be the set of items.
$[A \to \alpha X . \beta, a]$ such that
$[A \to \alpha . X\beta, a]$ is in I.

return CLOSURE J

end.

procedure setofLR1items
C = {CLOSURE ( {S' → .S, $ } ) }
repeat
for each set of items I in C & grammar symbol X s.t GOTO(I, X) is not empty & not already in C do.
add GOTO(J, X) to C.
until no more sets of items can be added to C.
return C

end.

Yash Vinayvanshi
B.Tech Computer Engineering
Roll No: 20DCS...
FoE&T Semester ... Session 202-2...
Jamia Millia University, New Delhi - 110025

Given Grammar

1. $S \rightarrow AA$.
2. $A \rightarrow aA$
3. $A \rightarrow b$.

$\longrightarrow$

Augmented Grammar

1. $S' \rightarrow S$
2. $S \rightarrow AA$
3. $A \rightarrow aA$
4. $A \rightarrow b$

FIRST(S) = {a,
FIRST(A) = {a,
FOLLOW(S) = {\$}
FOLLOW(A) = {\$,a,

$I_0$ : $S' \rightarrow .S, \$$

$A \rightarrow \alpha . B\beta, a$
ie $S \rightarrow \epsilon . S\epsilon$
FIRST($\beta a$) = FIRST(\$) = {\$}

$S \rightarrow .AA, \$$.
$A \rightarrow .aA, a/b$.
$A \rightarrow .b, a/b$.

$A \rightarrow \alpha . B\beta, a$
$A \rightarrow \epsilon . AA, \$$.
FIRST($\beta a$) = FIRST(A\$)
= FIRST(A) = a/b

GOTO($I_0$, S) ≡ I1.
$S' \rightarrow S., \$$.

GOTO($I_0$, A) ≡ I2.

$A \rightarrow \alpha . B\beta, a$
$A \rightarrow A . A\epsilon, \$$.
FIRST($\beta a$)
= FIRST($\epsilon$,\$) = \$.

$S \rightarrow A.A, \$$.
$A \rightarrow .aA, \$$.
$A \rightarrow .b, \$$.

GOTO($I_0$, a) ≡ $I_3$

$A \rightarrow \alpha . B\beta, a$.
$A \rightarrow a . A\epsilon, a/b$.
FIRST($\beta a$)
≡ FIRST($\epsilon$ a/b) = a, b

$A \rightarrow a.A, a/b$.
$A \rightarrow .aA, a/b$.
$A \rightarrow .b, a/b$.

GOTO($I_0$, b) ≡ $I_4$.
$A \rightarrow .b, a/b$.

~~GOTO($I_1$, Nothing)~~

GOTO($I_2$, A) ≡ $I_5$
$S \rightarrow AA., \$$.

GOTO($I_2$, a) ≡ $I_6$.

$A \rightarrow \alpha . B\beta, a$
$A \rightarrow a . A\epsilon, \$$.
FIRST($\beta a$)
≡ FIRST($\epsilon$, \$)
= \$.

$A \rightarrow a.A, \$$.
$A \rightarrow .aA, \$$.
$A \rightarrow .b, \$$.

GOTO($I_2$, b) ≡ $I_7$.
$A \rightarrow b., \$$.

GOTO($I_3$, A) ≡ $I_8$.
$A \rightarrow aA., a/b$.

~~GOTO($I_3$, a) ≡ $I_4$.~~
~~GOTO($I_6$, a)~~

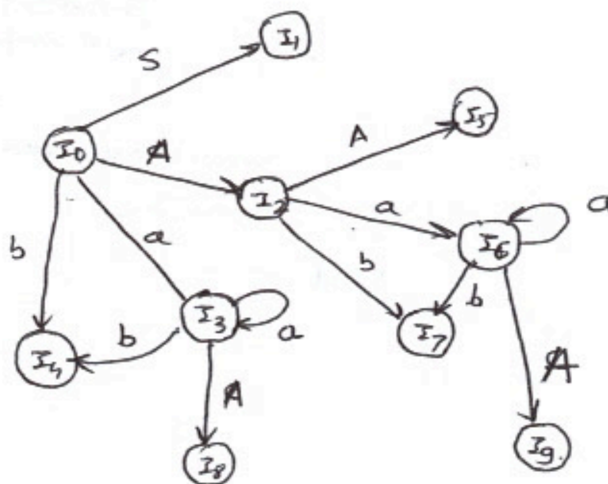
GOTO($I_3$, a) ≡ I3.

$A \rightarrow \alpha . B\beta, a$
$A \rightarrow a . A\epsilon, a/b$
FIRST($\beta a$)
≡ FIRST($\epsilon$, a/b)
= a/b.

$A \rightarrow a.A, a/b$
$A \rightarrow .aA, a/b$
$A \rightarrow .b, a/b$.

GOTO($I_3$, b) ≡ $I_4$.
GOTO($I_6$, A) ≡ I9
$A \rightarrow aA., \$$

GOTO($I_6$, a) ≡ $I_6$
GOTO($I_6$, b) ≡ $I_7$




GOTO Graph

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Algorithm : Construction of CLR parsing table.

I/p : A grammar G augmented by production $S' \to S$.

O/p : if possible, the CLR parsing funct ACTION & GOTO.

done. ✓ 1. construct $C = \{I_0, I_1, \ldots, I_n\}$, the collection of sets of LR(1) items for G

2. state i of parser is constructed from $I_i$. The parsing actions for state i are:

a) if $[A \to \alpha . a\beta, b]$ is in $I_i$ & $GOTO(I_i, a) = I_j$ then set ACTION [i, a] to shift j $(S_j)$.

b) If $[A \to \alpha ., a]$ is in $I_i$, then set ACTION [i, a] to reduce $A \to \alpha$ ($R_{A \to \alpha}$) or its index

c) if $[S' \to S., \$]$ is in $I_i$, then set ACTION [i, \$] to "accept"

3. if $GOTO(I_i, A) = I_j$ then set GOTO [i, A] = j.

4. All other entries made "error" or left blank.

5. Initial state of parse is one constructed from set containing item $[S' \to .S, \$]$.

if conflict (shift-reduce, reduce-reduce, ...) occurs using above rules then grammar is not LR(1).

1. $S' \to S$
2. $S \to AA$
3. $A \to aA$
4. $A \to b$.

| state | ACTION a | ACTION b | ACTION \$ | GOTO S | GOTO A |
|---|---|---|---|---|---|
| 0 | S3 | S4 | | 1 | 2 |
| 1 | | | acc. | | |
| 2 | S6 | S7. | | | 5 |
| 3 | S3 | S4. | | | 8 |
| 4 | R3 | R3 | | | |
| 5 | | | R1 | | |
| 6 | S6 | S7 | | | 9. |
| 7 | | | R3 | | |
| 8 | R2 | R2. | | | |
| 9 | | | R2 | | |

$I_0 \xrightarrow{a} I_3 \to S_3$

$I_2 \xrightarrow{A} I_5$

$GOTO(I_0, d) = I_4$
$A \to b.\, , a/b$.
rule 3 of grammar.
∴ R3 written at $I_4$ row in a/b.

$S' \to S., \$$ comes from $I_1$.
∴ $[I_1, \$] = Acc$

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q3a) context free grammar for arithmetic assignment.

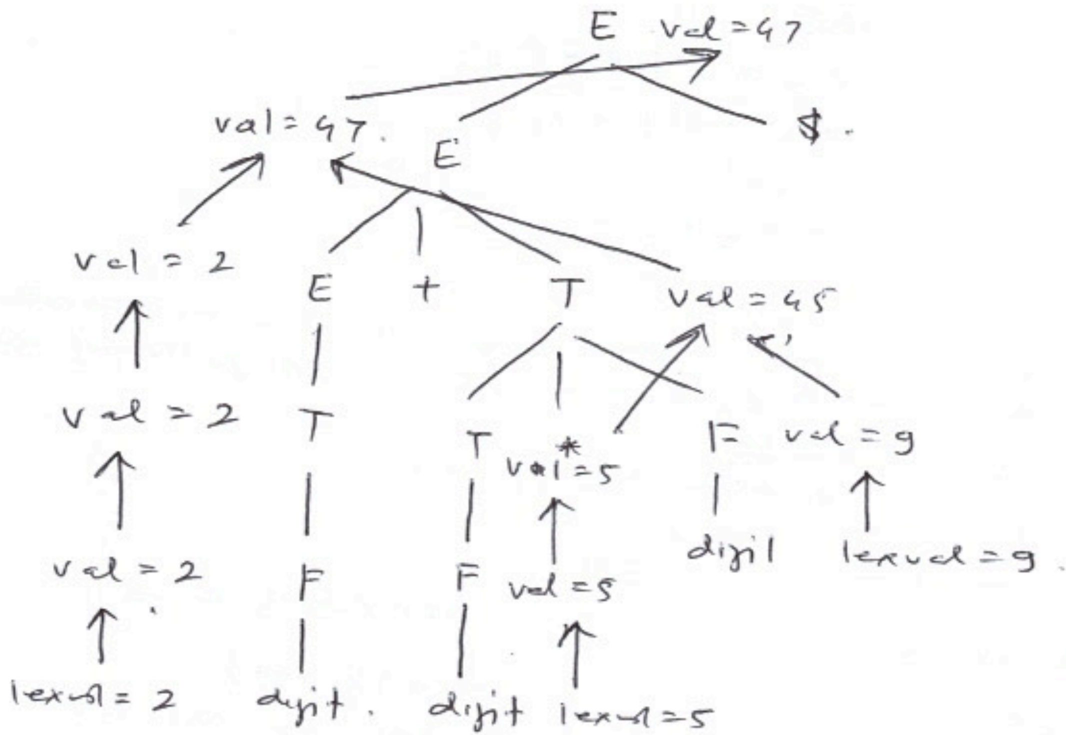
$S \rightarrow E$
$E \rightarrow E + T \mid E - T \mid T$
$T \rightarrow T * F \mid T / F \mid F$
$F \rightarrow (E)$
$F \rightarrow digit$

This grammar handles 5 operations on expression +, -, *, ÷ & ( ) and establishes following order of precedence
$+, - < *, / < ( )$.

| PRODUCTION | SEMANTIC RULE |
|---|---|
| $S \rightarrow E$ | print (E.val) |
| $E \rightarrow E_1 + T$ | $E.val = E_1.val + T.val$ |
| $E \rightarrow E_1 - T$ | $E.val = E_1.val - T.val$ |
| $E \rightarrow T$ | $E.val = T.val$ |
| $T \rightarrow T_1 * F$ | $T.val = T_1.val * F.val$ |
| $T \rightarrow T_1 / F$ | $T.val = T_1.val * F.val$ |
| $T \rightarrow F$ | $T.val = F.val$ |
| $F \rightarrow (E)$ | $F.val = E.val$ |
| $F \rightarrow digit$ | $F.val = digit.lexval$ (from lexical analysis) |

This is an S attribute definition & data flows strictly in bottom up fashion in val attributes.

eg. 2 + 5 * 9 $.



Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q3 b) Difference between syntax directed definitions & syntax directed translation

| SDDs (Attribute grammars) | SDTs (Translation scheme) |
|---|---|
| → give high level specification for translations. | → Indicates order of evaluation of semantic actions associated to a production rule. |
| Hides many implementation details such as order of evaluation of semantic actions. | ie provides information about implementation details. |
| The production rules are associated with a set of semantic actions but do not say when they'll be evaluated. | → semantic actions embedded anywhere in the bodies of productions. |

~~→ Example~~

| | |
|---|---|
| $L \to E n$ | $L.val = E.val$ |
| $E \to E_1 + T$ | $E.val = E_1.val + T.val$ |
| $E \to T$ | $E.val = T.val$ |
| $T \to T_1 * F$ | $F.val = T_1.val * F.val$ |
| $T \to F$ | $T.val = F.val$ |
| $F \to (E)$ | $F.val = E.val$ |
| $F \to digit$ | $F.val = digit.lexval$ |

(crossed out)

Example

$E \to E_1 + T \quad \{print + \}$.

for creating syntax tree.

Example.

$E \to E_1 + T \quad E.code = E_1.code \,||\, T.code \,||\, '+'$

(postfix conversion)

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q4 a). SDD to produce TAC for "for loop" in C PL.

| grammar. | semantic rule. |
|---|---|
| 1. STMT (statement) → for (E1, M E2, N E3) P STMT1 (expression) | { gen ('goto N.quad + 1') <br> $Q_1$ = nextquad. <br> gen ('if $E_2$.result == 0 goto —') <br> gen ('goto P.quad + 1'); <br> backpatch (N.quad, Q1); <br> backpatch ( $STMT_1$.next, N.quad + 1 ); <br> backpatch (P.quad, M.quad) <br> STMT.next := makelist(Q1) |
| 2. M → ∈ | { M.quad := nextquad; }. |
| 3. N → ∈ | { N.quad := nextquad; gen ('goto —'); }. |
| 4. P → ∈ | { P.quad := nextquad; gen ('goto —') }. |

— gen() writes the string.
— backpatch() fills the goto addresses in second pass.

Example. for ( E1 ; E2 ; E3 ) S.

```
        code for E1.
L1 :    code for E2. (result in T).
        goto L4
L2 :    code for E3
        goto L1.
L3 :    code for S      /* all jumps out of S goto L2 */
        goto L2
L4 :    if T == 0 goto L5
        goto L3
L5 :    /* exit */
```

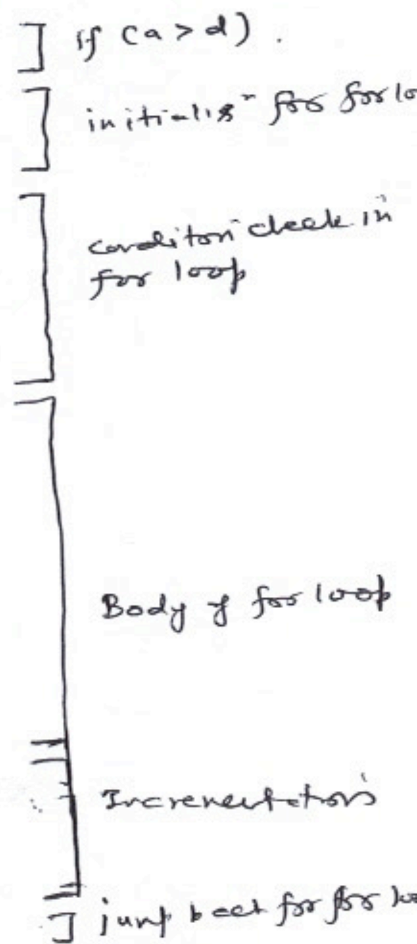
Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q 4 c).

```
if (a > d)
{
    for (a=0, d=0; a>b && c>d; a++, d++)
    {
        a = a + (b/c)^2;
        d = d - (a/b);
    }
}.
```

Short circuit 3 address code

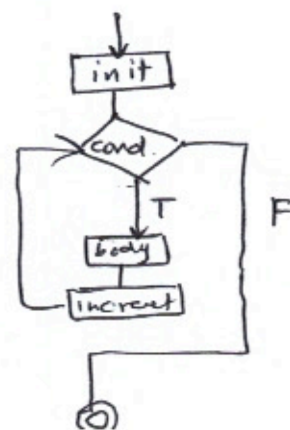
```
100 : if a > d goto 102
101 : goto 121.
102 : a = 0
103 : d = 0
104 : if a > b goto 106
105 : goto 121
106 : if c>d goto 108
107 : goto 121.
108 : t1 = b/c
109 : t2 = b/c
110 : t3 = t1 * t2
111 : t4 = a + t3.
112 : a = t4.
113 : t5 = a/b
114 : t6 = d - t5
115 : d = t6.
116 : t7 = a+1.
117 : a = t7.
118 : t8 = d+1
119 : d = t8.
120 : goto 104.
121 : ____
```




NOTE:

Flow control for for loop in C.

```
for (init; condition; increment) {
    body
}.
```



NOTE : preincrement & post increment calls a function both increment the variable but return values are different. preincrement returns incremented value, post increment returns old value (not incremented). post increment function under the hood needs to create a temporary variable in which it stores the old value of variable, increments variable & returns the temporary copy. But in this case the return values are just thrown away & original copy of variable is used.

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

## Q5b) LABELLING ALGORITHM.

optimal order means the order of independent statements in a block which yields shortest instruction sequence.

→ A simple algorithm exists to determine the optimal order in which to evaluate statements in a basic block when DAG representation of a block is tree.

→ The Labelling Algorithm is the first part of this algorithm for optimal ordering of trees.

→ Labelling algorithm labels each node of the tree bottom up with an integer that denotes the fewest number of registers required to evaluate the tree. with no stores of intermediate results.

NOTE: Machine model assumed here is, the left operand is allocated a register while the right operand is operated on left operand in the same register. No register is allocated for right operand.

**Algorithm : Labelling.**

input : expression tree

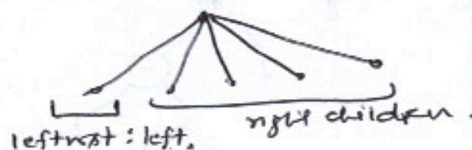
output : expression tree with labels denoting fewest no. of registers required to evaluate tree.

1) if n is a leaf then.
   if n is the leftmost child of its parent then
   $label(n) := 1$.
   else $label(n) := 0$.

2) else
   // n is internal node.
   let $n_1, n_2, \ldots, n_k$ be the children of n ordered by label ie $label(n_1) \geq label(n_2) \geq \ldots label($
   $$label(n) := \max_{1 < i < k} (label(n_i) + i - 1).$$

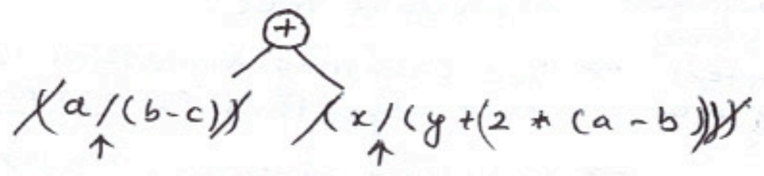
NOTE: The labelling algorithm generally visits the nodes in postorder which is most appropriate for label computation.

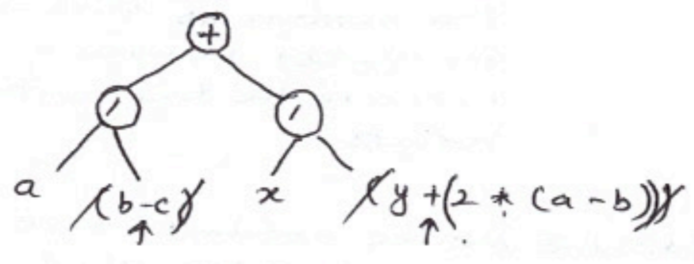
NOTE: left child in case of m-ary trees is the leftmost child

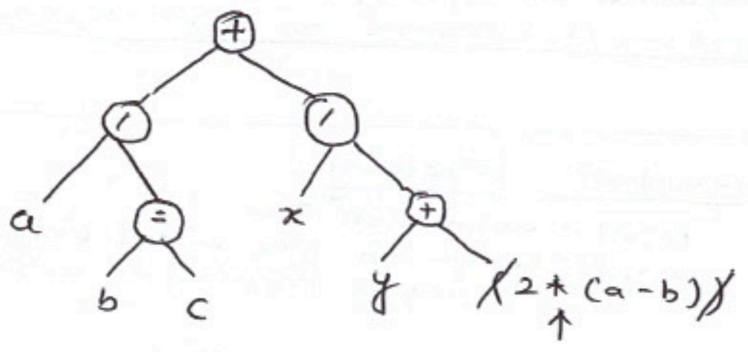


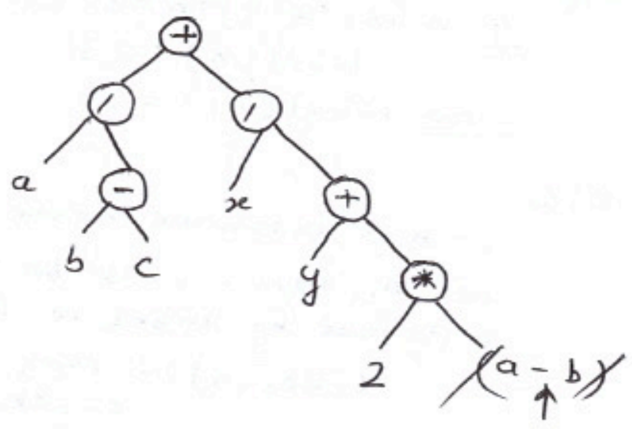
Eg. Building expression tree for given expression

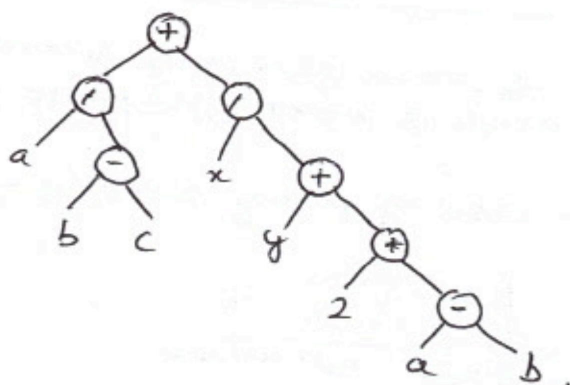
$$((a/(b-c)) + (x/(y+(2*(a-b)))))$$



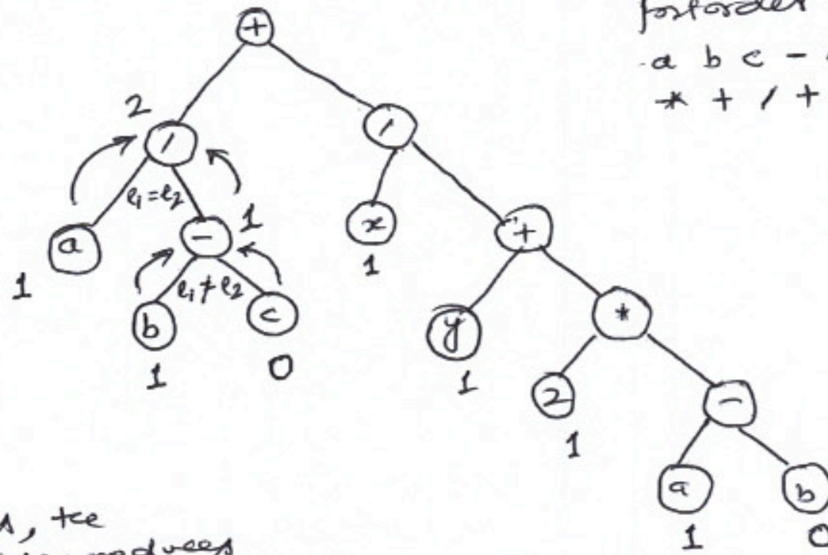



Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

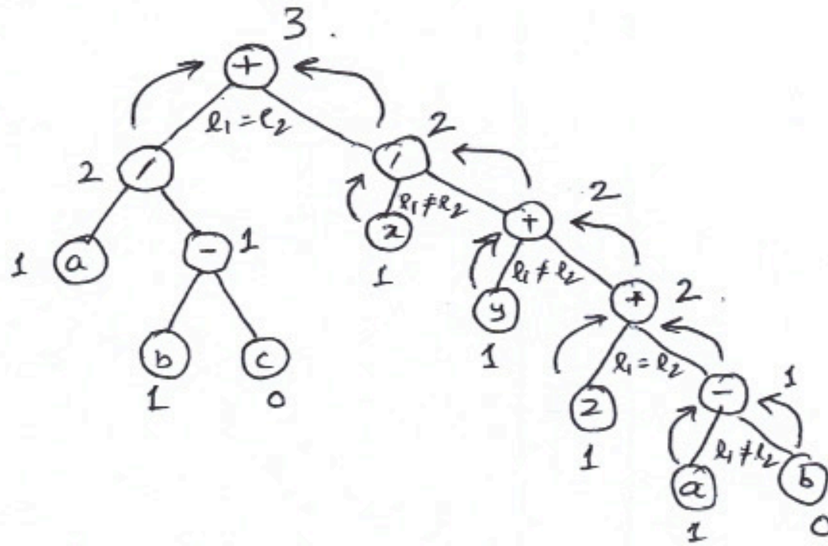
Applying labelling algorithm.

Postorder:
a b c - / x y z a b - * + / +.




For binary nodes, the general formula in Algo reduces to:

$$label(n) = \begin{cases} max(l_1, l_2) & \text{if } l_1 \neq l_2 \\ l_1 + 1 & \text{if } l_1 = l_2 \end{cases}$$




∴ Total No. of register needed = label of root = 3

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No.: 19BCS081
FoE&T Semester: VI Session: 2021-22
Jamia Millia University, New Delhi - 110025

Q5 c) i) Heuristic Node listing Agoritum for DAGs.

- It is a recording Agoritum which attempts as far as possible to make evaluation of a node immediately follow the evaluation of its left most arguement to avoid registee spilling. (ie reduce temporaries).

Algoritum : Node listing for DAGs.
input : DAG for a basic block
output : A node listing in reverse order that reduces temporaries.

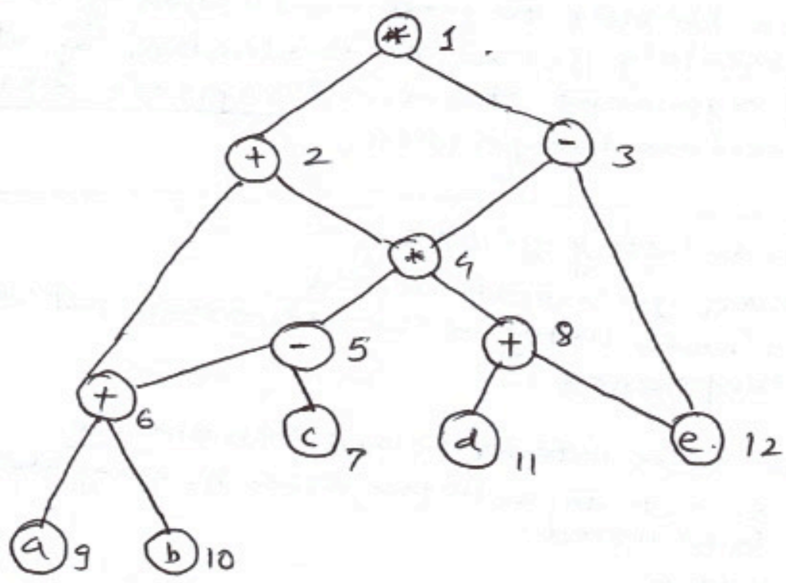
1. While unlisted interior nodes remain do
2. select an unlisted node n, all of whose parents have been listed.
3. list n.
4. While the leftmost child m of n has no unlisted parents & is not a leaf, do
5. list m;
6. n = m

(ii) Analysis .

(iii) Major reason for need of this Agoritum .

Yash Vinayvanshi
B. Tech. (Computer Engineering)
Roll No : 19BCS
Semester : Session : 202
FoET
Jamia Millia University, New Delhi - 110025

(iv) Illustration




Initially, the only node with unlisted parents is 1 ⇝ set n = 1

1 → (n=1) → 1.left = 2, 2.π = {1} (listed), list 2, {1, 2} → (n=2) → 2.left = 6, 6.π = {2, 5} (5 not listed); select another candidate, 3 (other child of 1) → 3.π = {1} (listed), list 3, {1, 2, 3}

list {1}

n = 3

3.left = 4, 4.π = {2, 3} (listed), list 4, {1, 2, 3, 4} → (n = 4) → 4.left = 5, 5.π = 4 (listed), list 5, {1, 2, 3, 4, 5} → (n = 5) → 5.left = 6, 6.π = {2, 5} (listed), list 6, {1, 2, 3, 4, 5, 6} → 6.left = 9 (leaf), 6.right = 9 (leaf), 5.right = 7 (leaf), n = 6, 4.right = 8

n = 8

8.π = {4}.
list 8.
{1, 2, 3, 4, 5, 6, 8}. ⇝ suggested order of evaluation is.
8 6 5 4 3 2 1.