8/2/23

★ Predictive Top Down Parser

How to calculate FIRST()

FIRST(α) → set of these terminals with which the strings derivate from α start

If $\alpha = XYZ$, then FIRST(α) is computed as follows.

① FIRST(α) = FIRST(XYZ) = $\{X\}$, if X is terminal

② FIRST(α) = FIRST(XYZ) = $\{X\}$, if X doesn't derive to an empty string

FIRST(X) doesn't contains ϵ

③ If FIRST(X) contains ϵ, then

FIRST(α) = FIRST(XYZ) = FIRST(X) $- \{\epsilon\}$ $\cup$ FIRST(YZ)

Date
Page

Q Consider the grammar ~~S → ACB / CbB / Ba~~

$S \to ACB / CbB / Ba$

$A \to da / BC$

$B \to g / \epsilon$

$C \to h / \epsilon$

Calculate FIRST for each production

Sol FIRST(S) → FIRST(ACB) ∪ FIRST(CbB) ∪ FIRST(Ba) — (1)

FIRST(A) → FIRST(da) ∪ FIRST(BC) — (2)

FIRST(B) → FIRST(g) ∪ FIRST(ε)
→ {g, ε}

FIRST(C) → FIRST(h) ∪ FIRST (ε)
→ {h, ε}

(2) FIRST(A) → FIRST (da) ∪ FIRST(BC)
→ {d} ∪ FIRST(BC) — (3)
→ {d} ∪ {g, h, ε} → {d, g, h, ε}

FIRST(BC) → FIRST(B) - {ε} ∪ FIRST(C)
→ {g, ε} - {ε} ∪ {h, ε}
→ {g, h, ε}

(1) FIRST (Ba) → FIRST(B) - {ε} ∪ FIRST(a)
→ {g, ε} - {ε} ∪ {a}
→ {g, a}

$FIRST(CbB) \rightarrow FIRST(C) - \{\epsilon\} \cup FIRST(bB)$
$\rightarrow \{h, \epsilon\} - \{\epsilon\} \cup \{b\}$
$\rightarrow \{h, b\}$

① $FIRST(ACB) \rightarrow FIRST(A) - \{\epsilon\} \cup FIRST(CB)$
$\rightarrow \{d, g, h, \epsilon\} - \{\epsilon\} \cup \{h, g, \epsilon\}$
$\rightarrow \{d, g, h, \epsilon\}$

$FIRST(CB) \rightarrow FIRST(C) - \{\epsilon\} \cup FIRST(B)$
$\rightarrow \{h, \epsilon\} - \{\epsilon\} \cup FIRST(B)$
$\cancel{\rightarrow \{h, \epsilon\} -}$
$\rightarrow \{h, g, \epsilon\}$

$FIRST(S) \rightarrow \{a, b, d, g, h, \epsilon\}$

Q Consider the following grammar calculate the FIRST for each and every production

$S \rightarrow ABC$
$A \rightarrow a / \epsilon$
$B \rightarrow r / \epsilon$
$C \rightarrow b / \epsilon$

$FIRST(S) \rightarrow \cancel{A} FIRST(ABC)$ — ①

$FIRST(A) \rightarrow FIRST(a) \cup FIRST(\epsilon)$
$\rightarrow \{a, \epsilon\}$

Date
Page

$FIRST(B) \rightarrow FIRST(r) \cup FIRST(\epsilon)$
$\rightarrow \{r, \epsilon\}$

$FIRST(C) \rightarrow FIRST(b) \cup FIRST(\epsilon)$
$\rightarrow \{b, \epsilon\}$

① $FIRST(S) \rightarrow FIRST(ABC)$
$\rightarrow FIRST(A) - \{\epsilon\} \cup FIRST(BC)$

$FIRST(BC) \rightarrow FIRST(B) - \{\epsilon\} \cup FIRST(C)$
$\rightarrow \{r, \epsilon\} - \{\epsilon\} \cup \{b, \epsilon\}$
$\rightarrow \{r, b, \epsilon\}$

① $FIRST(S) \rightarrow \{a, \epsilon\} - \{\epsilon\} \cup \{r, b, \epsilon\}$
$\rightarrow \{a, r, b, \epsilon\}$

Q $F \rightarrow XYZa$
$X \rightarrow x / \epsilon$
$Y \rightarrow y / \epsilon$
$Z \rightarrow z / \epsilon$

$FIRST(F) \rightarrow FIRST(XYZa)$ — ①

$FIRST(X) \rightarrow FIRST\{x\} \cup FIRST\{\epsilon\}$
$\rightarrow \{x\} \cup \{\epsilon\}$
$\rightarrow \{x, \epsilon\}$

$FIRST(Y) \rightarrow FIRST(y) \cup FIRST(\epsilon)$

$\rightarrow \{y\} \cup \{\epsilon\}$

$\rightarrow \{y, \epsilon\}$

$FIRST(Z) \rightarrow FIRST(z) \cup FIRST(\epsilon)$

$\rightarrow \{z\} \cup \{\epsilon\}$

$\rightarrow \{z, \epsilon\}$

$FIRST(XYZa) \rightarrow FIRST(X) - \{\epsilon\} \cup \{YZa\}$

$\rightarrow \{x, \epsilon\} - \{\epsilon\} \cup FIRST(Y) - \{\epsilon\} \cup FIRST(Za)$

$\rightarrow \{x\} \cup \{y\} \cup \{z\} \cup \{a\}$

$\rightarrow \{x, y, z, a\}$

15/2/24

*

15/02/23

Vidyalekhan
DATE
PAGE

* Predictive Table Driver Parser

Consider the following grammer

$S \rightarrow aAb$
$A \rightarrow cd \,/\, ef \,/\, \epsilon$ - ?
calculate first()

FIRST(S) $\longrightarrow$ FIRST (aAb) $\longrightarrow \{a\}$ $\rightarrow$ {S, a}

FIRST (A) $\rightarrow$ FIRST (cd) ∪ FIRST (ef) $\rightarrow$ $A \rightarrow cd$ = {A, c}
$\rightarrow$ {c, e} ~~A~~ $\rightarrow$ ef = {A, e}

Table

	a	b	c	d	e	f	$
S	$S \rightarrow aAb$						
A			$A \rightarrow cd$		$A \rightarrow ef$		

We use follow() function for finding the epsalon table if any production contain epsalon(ϵ).

* How to Calculate

The derivation by $A \rightarrow \epsilon$ is a right choice when the parser is on worge of the non-terminal A and the next input symbols happens to be a terminal which can occur immediately following

A in any string occuring on the right side of production. It is will late to the expansion of $A \rightarrow \epsilon$ and the next lead in the parse tree will be considered which is labeled by the symbol immediate following A and therefore may match next input symbol. Therefore we conclude that $A \rightarrow \epsilon$ is to be added in the table at [A, b] for every small b immediately follows A in any of the production in a right hand string. To compute the set of such terminals we make the use of function FOLLOW(A) where, A is a non-terminal and as defined below

* Rules -

FOLLOW (A) $\longrightarrow$ Set of Terminal that immediately follow A in any string occuring on the right side of production of grammer.

If $A \rightarrow \alpha B \beta$ is a production, then follow(B) will be computed as

Rule 1) FOLLOW (B) = FOLLOW (β) if first (β) does not contain ϵ

Rule 2) FOLLOW (B) = ~~FOLLOW~~ FIRST (β) - $\{\epsilon\}$ $\cup$ FOLLOW (A) if first (β) contains ϵ

17/02/23

FOLLOW (S) $\longrightarrow$ {\$}
When S is not getting the RHS of production rule

eg 1) S $\longrightarrow$ aAb
A $\longrightarrow$ c
FOLLOW (A) $\longrightarrow$ FIRST(b)
$\longrightarrow$ {b}

S $\longrightarrow$ aAbc
A $\longrightarrow$ c
FOLLOW (A) $\longrightarrow$ FIRST (bc)
$\longrightarrow$ {b}

eg 2) S $\longrightarrow$ aAB
A $\longrightarrow$ d
B $\longrightarrow$ c/ϵ

FOLLOW (S) $\rightarrow$ {\$}
FOLLOW(A) $\rightarrow$ FIRST (B) - {ϵ} U FOLLOW (S) — (1)
FIRST (B) $\rightarrow$ { c, ϵ }
FOLLOW (A) $\rightarrow$ { c, $\cancel{\epsilon}$ } - { $\cancel{\epsilon}$ } U {\$}
$\longrightarrow$ { c, \$ }

- Consider the following grammer & generate the predictive parsing table

S $\longrightarrow$ aABb
A $\longrightarrow$ c/ϵ
B $\longrightarrow$ d/ϵ

FIRST(S) $\rightarrow$ FIRST (aABb) $\longrightarrow$ {a} (S, a)
FIRST (A) $\rightarrow$ FIRST(c) U FIRST (ϵ)
$\rightarrow$ { c, ϵ }
FIRST (B) $\rightarrow$ FIRST (d) U FIRST(ϵ)
$\longrightarrow$ { d, ϵ }

$S \rightarrow aABb$

FOLLOW(A) $\rightarrow$ FIRST(Bb) — ①

FIRST(Bb) $\rightarrow$ FIRST(B) $\cup$ FIRST(b)

$\rightarrow$ FIRST(B) $- \{\epsilon\}$ $\cup$ FIRST(b)

$\rightarrow \{d, \cancel{\epsilon}\} - \{\cancel{\epsilon}\} \cup \{b\}$

$\rightarrow \{d, b\}$

1) ⇒ FOLLOW(A) $\rightarrow \{d, b\}$ (A→d), (A,b)

FOLLOW(B) $\rightarrow$ FIRST(b)

$\rightarrow \{b\}$

(B,b)

Predictive parsing Table

	a	b	c	d	$
S	$S \rightarrow aABb$				
A		$A \rightarrow \epsilon$	$A \rightarrow c$	$A \rightarrow \epsilon$	
B		$B \rightarrow \epsilon$		$B \rightarrow d$	

20/02/23
1st class

- Consider the grammer

$F \rightarrow XYZa$

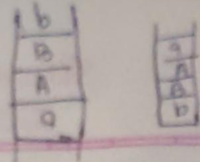
$X \rightarrow x \mid \epsilon$

$Y \rightarrow y \mid \epsilon$

$Z \rightarrow z \mid \epsilon$

1) Find the FIRST() & FOLLOW()
2) Generate the parsing table.

⇒ as per the schedule of 13/02/23. for ~~A~~ calculating of FIRST



$F \to XYZa$

FOLLOW (X) $\to$ FIRST (YZa)

$\to \{y, z, a\}$

FOLLOW (Y) $\to$ FIRST (Za)

$\to \{z, a\}$

FOLLOW (Z) $\to$ FIRST (a)

$\to \{a\}$

Predictive parsing Table

	x	y	z	a	$
F	$F \to XYZa$	$F \to XYZa$	$F \to XYZa$	$F \to XYZa$	
X	$X \to x$	$X \to \epsilon$	$X \to \epsilon$	$X \to \epsilon$	
Y		$Y \to y$	$Y \to \epsilon$	$Y \to \epsilon$	
Z			$Z \to z$	$Z \to \epsilon$	

20/02/23
2nd class

- consider the grammer

  $S \to aABb$

  $A \to c / \epsilon$

  $B \to d / \epsilon$

1) string $\omega \to acdb$ 2) $\omega \to ab$

The string is parse or not by stacking method / ~~Stack in~~ top-down parser with stack?

2)

1) $\omega \rightarrow acdb$

Stack	Unspent I/P	Moves
\$ S	acdb \$	$S \rightarrow aABb$
\$ bBAa	acdb \$	Pop a
\$ bBA	cdb \$	$A \rightarrow c$
\$ bBc	cdb \$	Pop c
\$ bB	db \$	$B \rightarrow d$
\$ bd	db \$	pop d
\$ b	b \$	pop b
\$	\$	String is parse.

2) $\omega \rightarrow ab$

Stack	Unspent I/P	Moves
\$ S	ab \$ ~~acdb \$~~	$S \rightarrow aABb$
~~\$ aABb~~	~~a~~	
\$ bBAa	ab \$	pop a
\$ bBA	b \$	$A \rightarrow \epsilon$
\$ bBϵ	ϵ b \$	pop ϵ
\$ bB	b \$	$B \rightarrow \epsilon$
\$ b	b \$	pop b
\$	\$	String is parse

* LL(1) — parser.

LL(1) is top-down parser

LL(1)

- left to right scan
- left most derivative parser
- length of ~~look ahead~~ is one

If the predictive table contain only single production in single cell.

→ Algorithm

For every pair of production.

$A \rightarrow \alpha / \beta$

$FIRST(\alpha) \cap FIRST(\beta) = \phi$

{

grammer is LL(1)

}

21/02/23

- Test whether the grammer is LL(1) or not and construct a predictive parsing table for it.

$S \rightarrow AaAb / BbBa$

$A \rightarrow \epsilon$

$B \rightarrow \epsilon$

→ $S \rightarrow \underbrace{AaAb}_{\alpha} / \underbrace{BbBa}_{\beta}$

$FIRST(\alpha) \cap FIRST(\beta) = \phi$ — (1)

$FIRST(AaAb) \rightarrow FIRST(A) - \{\epsilon\} \cup FIRST(aAb)$

$\rightarrow \{\epsilon\} - \{\epsilon\} \cup \{a\}$

$\rightarrow \{a\}$

21/2/23

Q Test whether the grammar is LL(1) or not and construct a predictive parsing table for it.

$S \rightarrow AaAb \,/\, BbBa$
$A \rightarrow \epsilon$
$B \rightarrow \epsilon$

Sol $\text{FIRST}(\alpha) \cap \text{FIRST}(\beta) = \{\Phi\}$ — ①

$\text{FIRST}(AaAb) \rightarrow \text{FIRST}(A) - \{\epsilon\} \cup \text{FIRST}(aAb)$
$\rightarrow \{\epsilon\} - \{\epsilon\} \cup \{a\}$
$\rightarrow \{a\}$

$\text{FIRST}(BbBa) \rightarrow \text{FIRST}(B) - \{\epsilon\} \cup \text{FIRST}(bBa)$
$\rightarrow \{\epsilon\} - \{\epsilon\} \cup \{b\}$
$\rightarrow \{b\}$

① ~~FIRST~~ $\{a\} \cap \{b\} = \{\Phi\}$

$\text{FIRST}(A) \rightarrow \{\epsilon\}$
$\text{FIRST}(B) \rightarrow \{\epsilon\}$

$\text{FIRST}(S) \rightarrow \cancel{Aa}\ \text{FIRST}(AaAb) \cup \text{FIRST}(BbBa)$
$\rightarrow \{a\} \cup \{b\}$
$\rightarrow \{a, b\}$

FOLLOW(S) → {$}

FOLLOW(A) → FIRST(aAb) → {a} | FIRST(b) → {b}

FOLLOW(A) → {a, b} | → {a, b}

FOLLOW(B) → FIRST(bBa) → {b} | FIRST(a) → {a}

FOLLOW(B) → {b, a}

	a	b	$
S	S→AaAb	S→BbBa	
A	A→ε	A→ε	
B	B→ε	B→ε	

Q

* Elimination of left Recursion

Q S → aIJh

I → Ib Se / c

J → KLKr / ε

K → d / ε

L → p / ε

Sol FIRST(S) → FIRST(a) − {ε} ∪ FIRST(IJh)

→ {a} − {ε} ∪

Consider the gramma

* **Elimination of left recursion**

If a grammar contains a pair of production of the form

$A \rightarrow A\alpha / \beta$, then the grammar is called left recursive grammar.

Q Consider the grammar containing left recursion

$A \rightarrow A\alpha / \beta$

where β doesn't begin with A.

To eliminate the left recursion replace of the pair of production with

$A \rightarrow \beta A'$

$A' \rightarrow \alpha A' / \epsilon$

Q $\rightarrow$ ~~SOI~~

$S \rightarrow aJh$

$I \rightarrow cI'$

$I' \rightarrow bSeI' / \epsilon$

$J \rightarrow KLKr / \epsilon$

$K \rightarrow d / \epsilon$

$L \rightarrow p / \epsilon$

FIRST(S) → {a}
FIRST(I) → {c}
FIRST(I') → {b} ∪ {ϵ} → {b, ϵ}
FIRST(J) → {d, p, r, ϵ}
FIRST(K) → {d, ϵ}
FIRST(L) → {p, ϵ}

FOLLOW(S) → {\$}
FOLLOW(I) → {d, p, r, h}
FOLLOW(I') → {d, p, r, h
FIRST(J) → {h}
FIRST(K) → {p, d, r}
FIRST(L) → {d, r}

23/2/23

Unit 3

Intermediate Code Representation form

① Postfix $A + B \rightarrow ab+$

② Syntax Tree

③ Three address code $c = a+b$

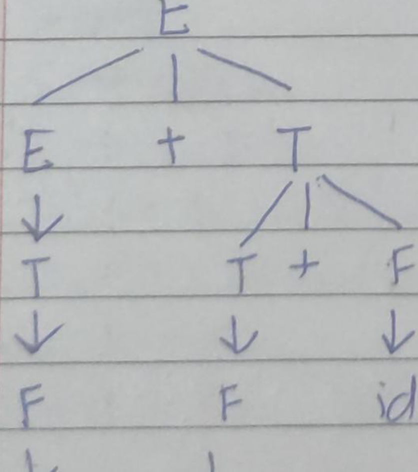
② * Representation of Syntax Tree

Ex
$E \rightarrow E + T / F$
$T \rightarrow T * F / F$
$F \rightarrow (E) / id$

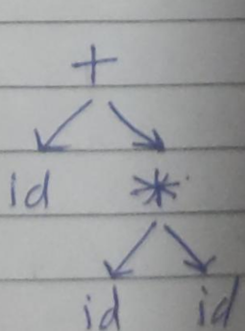
$w = id + id * id$

Draw parse tree & syntax tree

Sol



id + id * id

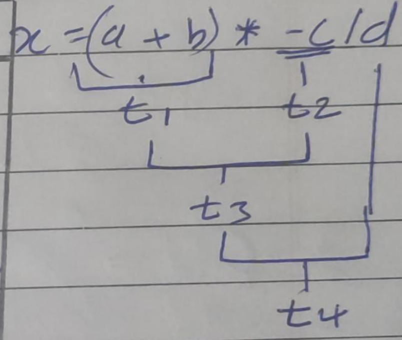


③ Three address code
i) Quadruple
ii) Triple
iii) Indirect triple

$$x = (a+b) * -c/d$$

Q i) Quadruple Representation

	Operator	operand1	op2	result
(1)	+	a	b	t_1
(2)	−	c		t_2
(3)	*	t_1	t_2	t_3
(4)	/	t_3	d	t_4
(5)	=	x	t_4	



ii) Triple Representation

~~operator operand1 operand2~~

	Operator	operand1	operand2
(1)	+	a	b
(2)	−	c	
(3)	*	(1)	(2)
(4)	/	(3)	d
(5)	=	x	(4)

Indirect Triple

Pointer
(1)
(2)
(3)
(4)
(5)

vimp 13 Mark

Q Consider the following grammar

$E \rightarrow E + T / T$

$T \rightarrow T * F / F$

$F \rightarrow id$

find the canonical collection of set of LR(0) items

Sol $E \rightarrow E + T$

$E \rightarrow T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow id$

step I :- Argumented grammar

$S \rightarrow E$

$E \rightarrow E + T$

$T \rightarrow T * F$

$T \rightarrow F$

$F \rightarrow id$

Step II :- Closure $(S \rightarrow \cdot E)$

$$= \left\{ \begin{array}{l} S \rightarrow \cdot E \\ E \rightarrow \cdot E + T \\ T \rightarrow \cdot T * F \\ T \rightarrow \cdot F \\ F \rightarrow \cdot id \end{array} \right\} \rightarrow \underline{I0}$$

Date
Page

Transitions States

$S \to \cdot E \quad S \to E\cdot$

$E \to \cdot E+T \quad E \to E\cdot+T \quad E \to E+\cdot T \quad E \to E+T\cdot$

$\cancel{T\to}\ E \to \cdot T \quad E \to T\cdot$

$T \to \cdot T*F \quad T \to T\cdot*F \quad T \to T*\cdot F \quad T \to T*F\cdot$

$T \to \cdot F \quad T \to F\cdot$

$F \to \cdot id \quad F \to id\cdot$

~~find the canonical~~

Step 3

$goto(I_0, E) \to \begin{Bmatrix} S \to E\cdot \\ E \to E\cdot+T \end{Bmatrix} \to I_1$

$goto(I_0, T) \to \begin{Bmatrix} E \to T\cdot \\ T \to T\cdot*F \end{Bmatrix} \to I_2$

$goto(I_0, F) \to \{T \to F\cdot\} \to I_3$

$goto(I_0, +) \to \phi$

$goto(I_0, *) \to \phi$

$goto(I_0, id) \to \{F \to id\cdot\} \to I_4$

Date
Page

~~goto(I~~

goto(I_1, E) → ϕ

goto(I_1, T) → ϕ

goto (I_1, F) → ϕ

goto(I_1, +) → $\left\{\begin{array}{l} E \to E + \cdot T \\ T \to \cdot T * F \\ T \to \cdot F \\ F \to \cdot id \end{array}\right\}$ I_5

goto (I_1, *) → ϕ

goto(I_1, id) → ϕ

goto(I_2, E) → ϕ

goto (I_2, T) → ϕ

goto (I_2, F) → ϕ

goto (I_2, +) → ϕ

goto(I_2, *) → $\left\{\begin{array}{l} T \to T * \cdot F \\ F \to \cdot id \end{array}\right\}$ I_6

goto (I_2, id) → ϕ

goto (I3, E) → ϕ
goto (I3, T) → ϕ
goto (I3, F) → ϕ
goto (I3, +) → ϕ
goto (I3, *) → ϕ
goto (I3, id) → ϕ

goto(I4, E) → ϕ
goto(I4, T) → ϕ
goto(I4, F) → ϕ
goto (I4, +) → ϕ
goto(I4, *) → ϕ
goto(I4, id) → ϕ

goto(I5, E) → ϕ
goto (I5, T) → $\left\{\begin{array}{l} E \to E+T\cdot \\ T \to T\cdot *F \end{array}\right\}$ — I7
goto (I5, +) → ϕ
goto(I5, *) → ϕ
goto (I5, id) → { F → id· } → I4
goto(I5, id) → { T → F· } → I3

goto(I6, F) → { T → T*F· } ← I8
goto(I6, id) → { F → id· } – I4
goto (I7, *) → $\left\{\begin{array}{l} T \to T*\cdot F \\ F \to \cdot id \end{array}\right\}$ → I6

I8 → ϕ (for all)

6/3/23

Q Construct ~~&~~ an LL(1) parsing table for the following grammar

$S \rightarrow aBDh$
$B \rightarrow cC$
$C \rightarrow bC / \epsilon$
$D \rightarrow EF$
$E \rightarrow g / \epsilon$
$F \rightarrow f / \epsilon$

Sol $FIRST(F) \rightarrow \{f, \epsilon\}$
$FIRST(E) \rightarrow \{g, \epsilon\}$

$FIRST(D) \rightarrow \{f, g, \epsilon\}$
$FIRST(C) \rightarrow \{b, \epsilon\}$
$FIRST(B) \rightarrow \{c\}$
$FIRST(S) \rightarrow \{a\}$

$FOLLOW(S) \rightarrow \{\$\}$
$FOLLOW(B) \rightarrow FIRST(Dh)$
$\rightarrow FIRST(D) - \{\epsilon\} \cup FIRST(h)$
$\rightarrow \{f, g, h\}$

$FOLLOW(C) \rightarrow FOLLOW(B)$
$\rightarrow \{f, g, h\}$

$FOLLOW(D) \rightarrow FIRST(h)$
$\rightarrow \{h\}$

~~FIRST(E) ⇒~~ 1 ∪(FOLLOW)D

FOLLOW(E) → FIRST(F) − {ε} ~~∪FOLLOW(D)~~

→ {f} ∪ {h}

→ {f, h}

FOLLOW(F) → FOLLOW(D)

→ {h}

	a	b	c	f	g	h	$
S	S→aBDh						
B			B→cC				
C		C→bC		C→ε	C→ε	C→ε	
D				D→EF	D→EF		
E				E→ε	E→g	E→ε	
F				F→f		F→ε	

Q Generate the ~~$~~ LL(1) parsing table

$S_1 \rightarrow S\#$

$S \rightarrow qABC$

$A \rightarrow a / bbD$

$B \rightarrow a / \epsilon$

$C \rightarrow b / \epsilon$

$D \rightarrow c / \epsilon$

FI

Q. LL(1) Parsing table

$S_1 \rightarrow S\#$

$S \rightarrow q / ABC$

$A \rightarrow a / bbD$

$B \rightarrow a / \epsilon$

$C \rightarrow b / \epsilon$

$D \rightarrow c / \epsilon$

Sol $\text{FIRST}(B) \rightarrow \{a, \epsilon\}$

$\text{FIRST}(C) \rightarrow \{b, \epsilon\}$

$\text{FIRST}(D) \rightarrow \{c, \epsilon\}$

$\text{FIRST}(A) \rightarrow \{a, b\}$

$\text{FIRST}(S) \rightarrow \{q\}$

$\text{FIRST}(S_1) \rightarrow \text{FIRST}(S) \cup \text{FIRST}(\#)$

$\rightarrow \{q, \#\}$

$\text{FOLLOW}(S_1) = \{\$\}$

$\text{FOLLOW}(S) \rightarrow \text{FIRST}(\#) \rightarrow$

$\rightarrow \{\#\}$

$\text{FOLLOW}(A) \rightarrow \text{FIRST}(BC)$

$\rightarrow \text{FIRST}(B) - \{\epsilon\} \cup \text{FIRST}(C)$

$\rightarrow \{a, \epsilon\} - \{\epsilon\} \cup \{b, \epsilon\} \cup \text{FOLLOW}(S)$

$\rightarrow \{a, b, \#\}$

FOLLOW(B) → FIRST(C) − {ε} ∪ FOLLOW(S)
→ {b, #}

FOLLOW(C) → FOLLOW(S).
→ {#}

FOLLOW(D) → FOLLOW(A)
→ {a, b, #}

	a	b	c	q	#	$
S_1				$S_1 \to q\#$	$S_1 \to q\#$	
S				$S \to qABC$		
A	$A \to a$	$A \to bbD$				
B	$B \to a$	$B \to \epsilon$			$B \to \epsilon$	
C		$C \to b$			$C \to \epsilon$	
D	$D \to \epsilon$	$D \to \epsilon$	$D \to c$		$D \to \epsilon$	

Date
Page

21/3/23

Q Consider the following grammar

$S \to AaAb$ — (1)

$S \to BbBa$ — (2)

$A \to \epsilon$ — (3)

$B \to \epsilon$ — (4)

Solve using LR(0)

Sol step I - Argumented grammar

~~Ans~~

$S_1 \to S$

$S \to Aa\,Ab$

$S \to Bb\,Ba$

$A \to \epsilon$

$B \to \epsilon$

step 2 - Closure $(S_1 \to \cdot S)$

$$\to \left\{ \begin{array}{l} S_1 \to \cdot S \\ S \to \cdot AaAb \\ S \to \cdot BbBa \\ A \to \cdot \\ B \to \cdot \end{array} \right\} \quad \begin{array}{l} I_0 \\ R_K \to R_3 \\ R_K \to R_4 \end{array}$$

goto $(I_0, S_1) \to \{ S_1 \to S \cdot \}$ I_1

goto $(I_0, A) \to \{ S \to A \cdot aAb \}$ I_2

goto $(I_0, B) \to \{ S \to B \cdot bBa \}$ I_3

goto $(I_0, a) \rightarrow \phi$

goto $(I_0, b) \rightarrow \phi$

$I_1 \rightarrow \phi$ {all entries)

goto $(I_2, a) \rightarrow \left\{ \begin{array}{l} S \rightarrow Aa \cdot Ab \\ A \rightarrow \cdot \end{array} \right\} I_4$

goto $(I_3, b) \rightarrow \left\{ \begin{array}{l} S \rightarrow Bb \cdot Ba \\ B \rightarrow \cdot \end{array} \right\} I_5$

goto $(I_4, A) \rightarrow \{S \rightarrow AaA \cdot b\}\ I_6$

goto $(I_5, B) \rightarrow \{S \rightarrow BbB \cdot a\}\ I_7$

goto $(I_6, b) \rightarrow \{S \rightarrow AaAb \cdot\}\ I_8$

goto $(I_7, a) \rightarrow \{S \rightarrow BbBa \cdot\}\ I_9$

~~FOLLOW($)~~ —

FOLLOW(S) → ($)
FOLLOW (A) → (a, b)
FOLLOW (B) → (a, b)

	Action Table			Go to Table		
	a	b	$	S	A	B
I_0	R_3/R_4	R_3/R_4		1	2	3
I_1			Accept			
I_2	S_4					
I_3		S_5				
I_4	R_3	R_4			6	
I_5	R_3	R_4				7
I_6		S_3				
I_7	S_9					
I_8			R_1			
I_9			R_2			

Q LR(1) parser

$S \to AaAb$

$S \to BbBa$

$A \to \epsilon$

$B \to \epsilon$

Sol Step 1 → Argumented grammar

$S_1 \to S$

$S \to AaAb$

$S \to BbBa$

$A \to \epsilon$

$B \to \epsilon$

step 2 : closure $(S_1 \rightarrow \cdot S)$

$$\rightarrow \left\{ \begin{array}{l} S_1 \rightarrow \cdot S, \$ \\ S \rightarrow \cdot AaAb, \$ \\ S \rightarrow \cdot BbBa, \$ \\ A \rightarrow \cdot, a \\ B \rightarrow \cdot, b \end{array} \right\} I_0$$

$goto(I_0, S) \rightarrow \{ S_1 \rightarrow S\cdot, \$ \}$ I_1

$goto(I_0, A) \rightarrow \{ S \rightarrow A\cdot aAb, \$ \}$ I_2

$goto(I_0, B) \rightarrow \{ S \rightarrow B\cdot bBa, \$ \}$ I_3

$goto(I_0, a) \rightarrow \phi$

$goto(I_0, b) \rightarrow \phi$

$I_1 \longrightarrow \phi$ (for all)

$$goto(I_2, a) \rightarrow \left\{ \begin{array}{l} S \rightarrow Aa\cdot Ab \\ A \rightarrow \cdot R3 \end{array} \right\} I_4$$

$$goto(I_3, b) \rightarrow \left\{ \begin{array}{l} S \rightarrow Bb\cdot Ba \\ B \rightarrow \cdot \end{array} \right\} I_5$$

$goto(I_5, B) \rightarrow \{S \rightarrow BbB \cdot a\}$ I_7

$goto(I_6, b) \rightarrow \{S \rightarrow AaAb\underset{R_1}{\cdot}\}$ I_8

$goto(I_7, a) \rightarrow \{S \rightarrow BbBa \cdot\}$ I_9

	Action Table			Goto Table		
	a	b	$	S	A	B
I0	R3/R4	R3/R4		1	2	3
I1			Accept			
I2	S4					
I3		S5				
I4	R3	R4			6	
I5	R3	R4				7
I6		S8				
I7	S9					
I8		$	R_1			
I9			R_2			

Date
Page

27/3/23

☆ LALR Parser
Look ahead LR parser

Q consider the grammar
$S \to AA$ —(1)
$A \to aA$ —(2)
$A \to b$ —(3)
Construct the parsing table with LALR

Sol step 1: Argumented grammar
$S_1 \to S$
$S \to AA$
$A \to aA$
$A \to b$

step 2: Closure $(S_1 \to \cdot S)$

$$= \left\{ \begin{array}{l} S_1 \to \cdot S, \$ \\ S \to \cdot AA, \$ \\ A \to \cdot aA, a/b \\ A \to \cdot b, a/b \end{array} \right\} I_0$$

goto$(I_0, S) \to \{ S_1 \to S\cdot, \$ \} I_1$

$$\text{goto}(I_0, A) \to \left\{ \begin{array}{l} S \to A \cdot A, \$ \\ A \to \cdot aA, \$ \\ A \to \cdot b, \$ \end{array} \right\} I_2$$

Date ________
Page ________

$$goto(I_0, a) \rightarrow \left\{\begin{array}{l} A \rightarrow a \cdot A, \; ab/b \\ A \rightarrow \cdot aA, \; a/b \\ A \rightarrow \cdot b, \; a/b \end{array}\right\} I_3$$

$$goto(I_0, b) \rightarrow \{ A \rightarrow b \cdot, \; a/b \} \; I_4$$

$$goto(I_2, A) \rightarrow \{ S \rightarrow AA \cdot, \; \$ \} \rightarrow I_5$$

$$goto(I_2, a) \rightarrow \left\{\begin{array}{l} A \rightarrow a \cdot A, \; \$ \\ A \rightarrow \cdot aA, \; \$ \\ A \rightarrow \cdot b, \; \$ \end{array}\right\} \rightarrow I_6$$

$$goto(I_2, b) \rightarrow \{ A \rightarrow b \cdot, \; \$ \} \rightarrow I_7$$

$$goto(I_3, A) \rightarrow \{ A \rightarrow aA \cdot, \; a/b \} \rightarrow I_8$$

$$goto(I_3, a) \rightarrow \left\{\begin{array}{l} A \rightarrow a \cdot A, \; a/b \\ A \rightarrow \cdot aA, \; a/b \\ A \rightarrow \cdot b, \; a/b \end{array}\right\} \rightarrow I_3$$

$$goto(I_3, b) \rightarrow \{ A \rightarrow b \cdot, \; a/b \} \rightarrow I_4$$

$$goto(I_6, A) \rightarrow \{ A \rightarrow aA \cdot, \; \$ \} \rightarrow I_9$$

$$goto(I_6, a) \rightarrow \left\{\begin{array}{l} A \rightarrow a \cdot A, \; \$ \\ A \rightarrow \cdot aA, \; \$ \end{array}\right. \rightarrow I_6$$

	Action Table			Goto	Table
I_0	a	b	$	S	A
I_1	S_3	S_4		1	2
I_2			Accept		
I_{36}	S_6	S_7			5
I_{47}	S_{36}	S_{47}			89
I_5	R_3	R_3	R_3		
I_{89}			R_1		
	R_2	R_2	R_2		

	Action Table				GoTo Table		
	+	*	id	$	E	T	F
I_0			S4		1	2	3
I_1	S5			Accept			
I_2	R2	S6		R2			
I_3	R4	R4		R4			
I_4	R5	R5		R5			
I_5			S4			7	3
I_6			S4				8
I_7	R1	S6		R1			
I_8		R3		R3			

---x---

Unit: 3

* Syntax Directed Translation Schema :-

Phases of compiler :-

1. lexical analyzer
2. Syntax analyzer
3. Intermediate code generation
4. Code optimization
5. Code generation

SDTS = (1) production with grammar symbol
(2) Semantic rule with attribute

Production	Semantic rule
$E_1 \rightarrow E + T$	$E_1.val \rightarrow E.val + T.val$
Symbol	attribute

SDTS :-

We know that every programming language construct (A language construct is syntatically allowable part of program that may be form from one or more lexical token in accordance with the rule of programming language). So it is very imp. to know the rules of the translation of programming construct whenever a construct encounters in programming language it is evaluated translated according to symentic rule defined in that particular programming language.

The translation may be generation one of intermediate code, object code, or adding info. in simple table about the construct time.

Syntatic structure of programming language specify the property of construct. The context free grammer is used to specify the syntatic structure of programming language for that we add a attribute with the grammer symbol & semantic rules with the production to make translation of construct easy.

The syntax analyzer directs the whole process during the parsing of the source code:

(1) All the lexical analyzer whenever the syntax analyzer wants another token.

(2) Perform the action of semantic analyzer.

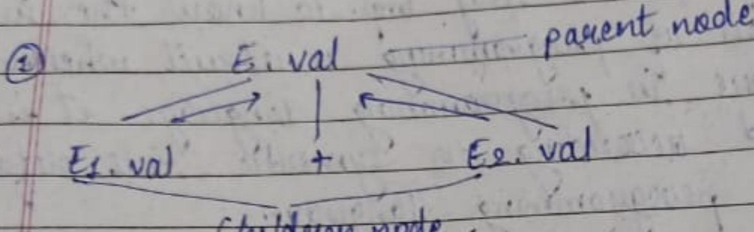
(3) " " " " intermediate code generator

<u>Attribute</u>: are associated information with the language

construct by attaching them to the grammer symbol representing that construct.

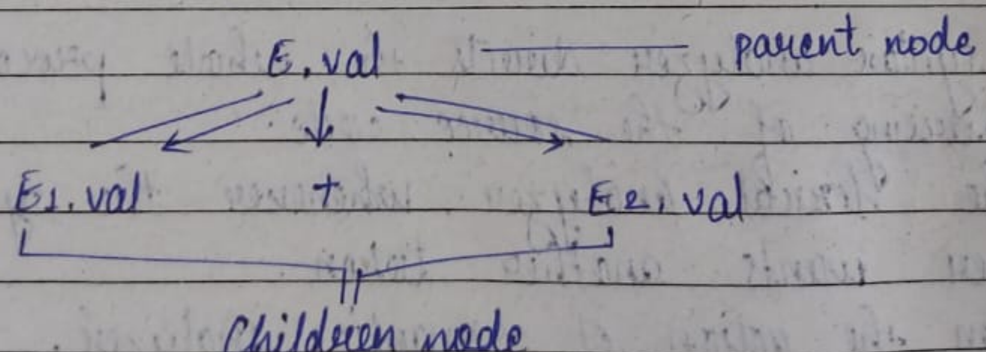
Types of attributes :-
1. Synthesized attribute
2. Inherited attribute

(1)



An attribute at a node is ~~syntasized~~ synthesized if its value is computed from the attributed value of the children of that node in the parse tree

(2) An attribute at a node is inherited if its value is computed from its attribute value at sibling or parent of that node in the parse tree



Q For the given grammar

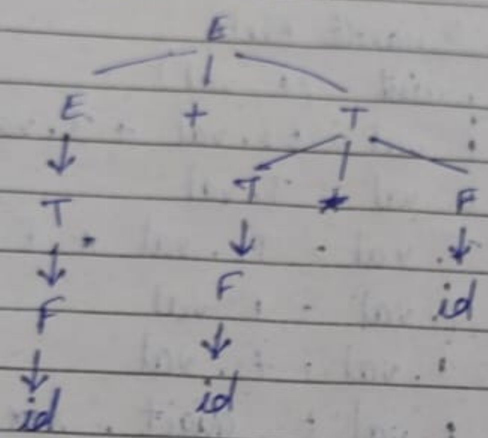
$E \rightarrow E + T \,/\, T$

$T \rightarrow T * F \,/\, F$

$F \rightarrow (E) \,/\, id$

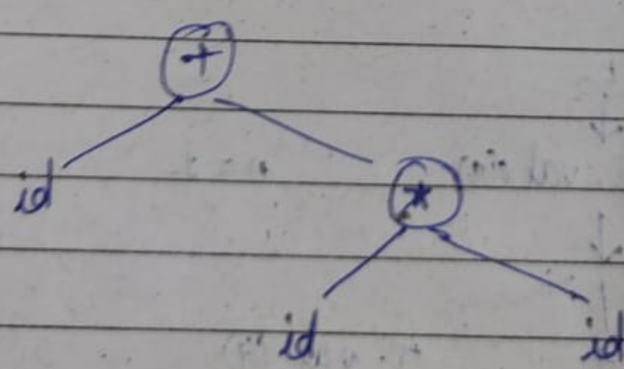
Construct parse tree & syntax tree for string $w = id + id * id$

Sol^n Parse Tree



Syntax tree

$id + (id * id)$



mknode =

op	left op	right op

* Annotated Parse Tree :—

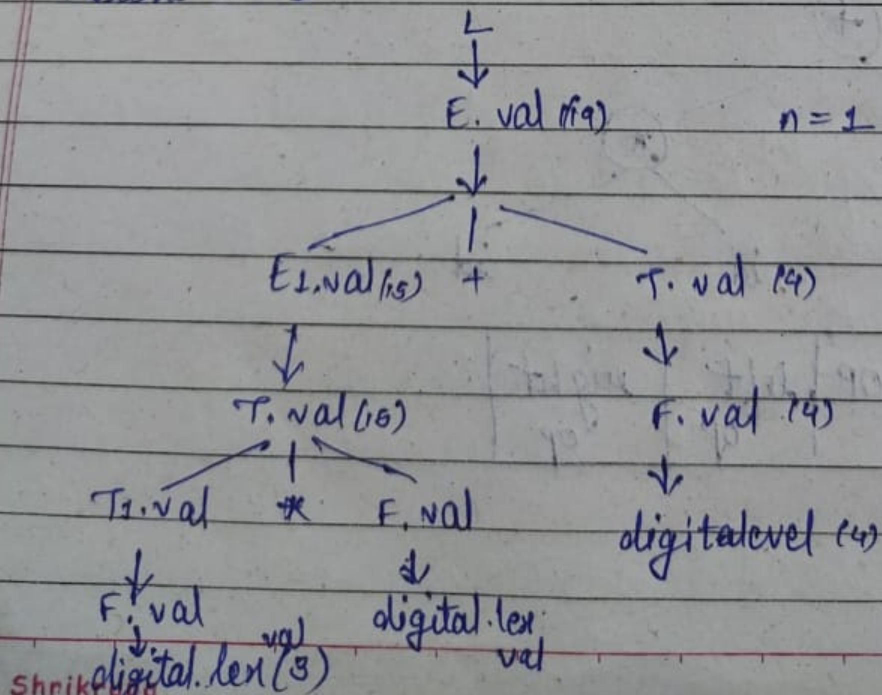
A parse tree showing the values of the attributes at each node is called an annotated parse tree.

Q Consider the following semantic rule for the given production

Production	Semantic Rule
$L \rightarrow En$	print (E. 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
$T \rightarrow F$	T. val = F. val
$F \rightarrow (E)$	F. val = E. val
$F \rightarrow$ digit	F. val = digit. lex val

Draw the annotated parse tree for expression
3 * 5 + 4n

Sol^n: Assume n = 1



29/3/23

(3) Three Address Code

i) Quadruple Representation

$$a = \underbrace{\underbrace{\underbrace{-b}_{t_1} * \underbrace{(c+d)}_{t_2}}_{t_3} / e}_{t_4}$$

	operator	oprand 1	oprand 2	result
(1)	$-$	b		t_1
(2)	$+$	c	d	t_2
(3)	$*$	t_1	t_2	t_3
(4)	$/$	t_3	e	t_4
(5)	$=$	t_4		a

ii) Triple Representation

	operator	oprand 1	oprand 2
(1)	$-$	b	
(2)	$+$	c	d
(3)	$*$	(1)	(2)
(4)	$/$	(3)	e
(5)	$=$	(4)	a

Date
Page

iii) Indirect Triple Representation

(1)	
(2)	
(3)	
(4)	
(5)	

Q Write a quadruple, triple representation for the following equation

$$-(a+b) * (c+d) + (a+b+c)$$

Sol

	oprator	oprand 1	oprand 2	Result
(1)	+	a	b	t_1
(2)	+	c	d	t_2
(3)	*	t_1	t_2	t_3
(4)	+	t_1	c	t_4
(5)	-	t_4	t_3	t_5

	oprator	prand 1	oprand 2
(1)	+	a	b
(2)	+	c	d
(3)	*	(1)	(2)
(4)	+	(1)	c
(5)	-	(4)	(3)

Date
Page

5/4/23

★ Syntax directed translation

Scheme for
① boolean expression
② If - then - else statement

① Boolean Expression

Q Translate the given expression into three address code.
$x < y$ and $a > b$ into boolean exp

sol
0) if $(x < y)$ goto 3
1) $t = 0$
2) goto 4
3) t_1
4) if $(a > b)$ goto 7
5) $t_2 = 0$
6) goto 8
7) $t_2 = 1$
8) $t_3 = t_1$ and t_2

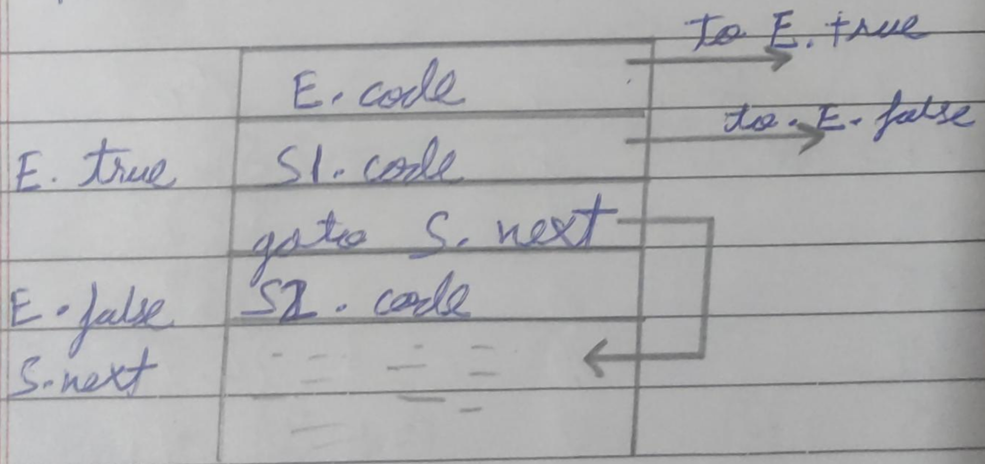
Q

QB(2A) if $x < y$ then $a = b + c$ else $p = q + r$

Sol

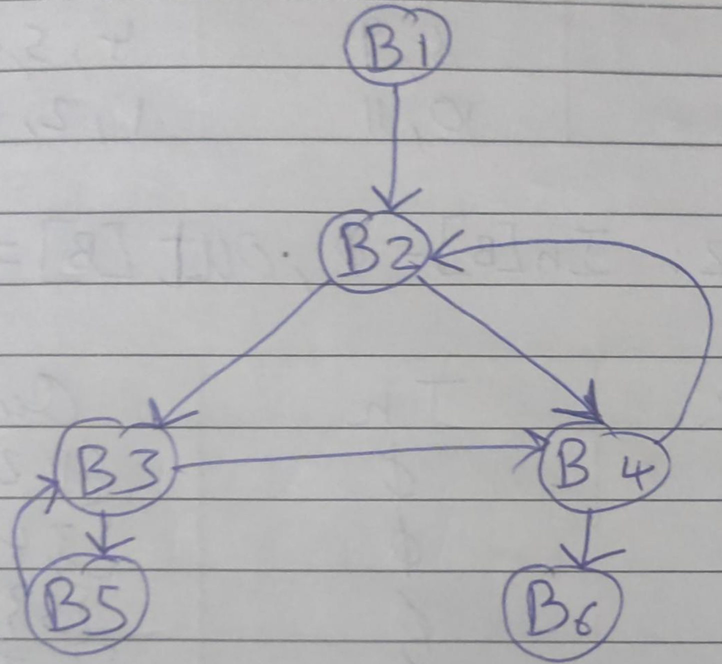
0) if $(x < y)$ goto 2
1) goto 5
2) $t1 = b + c$
3) $a = t1$
4) goto S.next
5) $t2 = q + r$
6) $p = t2$

```
if (a>b)
{ S1
}
else
{ S2
}
```



for $S \rightarrow$ if E then S1 else S2

Q Find In & Out for each block



B1 : (1) b=1
(2) c=2
B2 : (3) a=b+c
(4) d=a-b
B3 : (5) d=c+d
B4 : (6) c = b+c
(7) e = ~~a~~ a-b
B5 : (8) d = b+c
(9) e = e+1
B6 : (10) b = c+d
(11) c = b-d

	In	out
B1	0	B2
B2	B1, B4	B3, B4
B3	B2, B5	B4, B5
B4	B2, B3	B2, B6
B5	B3	B3
B6	B4	0

Step 1:- Calculate gen and kill

Block	gen	kill
B1	1, 2	6, 10, 11
B2	3, 4	5, 8
B3	5	4, 8
B4	6, 7	2, 9, 11
B5	8, 9	4, 5, 7
B6	10, 11	1, 2, 6

Step 2: In[B] = ϕ, out [B] = gen[B]

Block	In	Out
B1	ϕ	{1, 2}
B2	ϕ	{3, 4}
B3	ϕ	{5}
B4	ϕ	{6, 7}
B5	ϕ	{8, 9}
B6	ϕ	{10, 11}

Step 3: out (B) = In (B) − Kill[B] ∪ gen[B]

Block	In[B]	Out[B]
B1	ϕ	{1, 2}
B2	{1, 2, 6, 7}	{1, 2, 3, 4, 6}
B3	{3, 4, 8, 9}	{3, 5, 9}
B4	{3, 4, 5}	{3, 4, 5, 6, 7}
B5	{5}	{8, 9}
B6	{6, 7}	{7, 10, 11}

out[B3] =

Step 4: out(B) = In[B] − kill(B) ∪ gen[B]

Block	In[B]	Out B
B1	∅	{1, 2}
B2	{1, 2, 3, 4, 6, 7}	{1, 2, 3, 4, 6, 7}
B3	{1, 2, 3, 4, 6, 8, 9}	{1, 2, 3, 5, 6, 7, 9}
B4	{1, 2, 3, 4, 5, 6, 9}	{1, 3, 4, 5, 6, 7}
B5	{3, 5, 9}	{3, 8, 9}
B6	{3, 4, 5, 6, 7}	{3, 4, 5, 7, 10, 11}

Step 5: Out(B) = In[B] − Kill[B] ∪ gen[B]

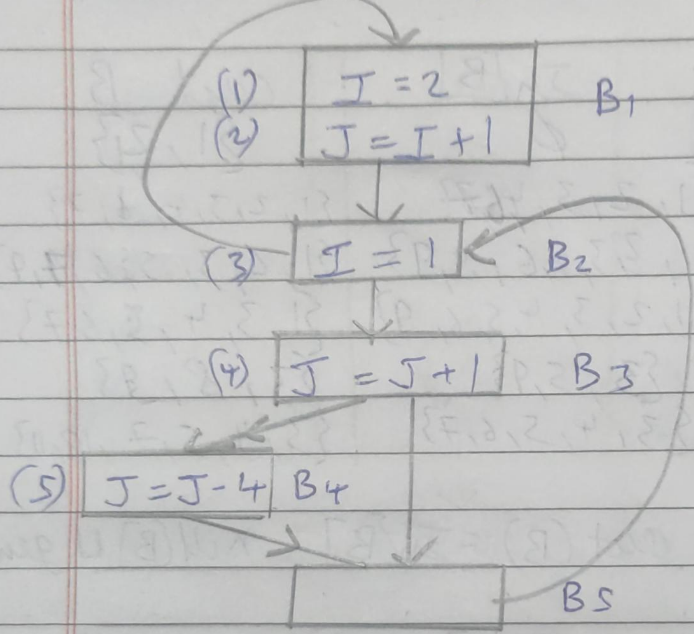
Block	In[B]	out[B]
B1	∅	{1, 2}
B2	{1, 2, 3, 4, 5, 6, 7}	{1, 2, 3, 4, 6, 7}
B3	{1, 2, 3, 4, 6, 7, 8, 9}	{1, 2, 3, 5, 6, 7, 9}
B4	{1, 2, 3, 4, 5, 6, 7, 9}	{1, 3, 4, 5, 6, 7}
B5	{1, 2, 3, 5, 6, 7, 9}	{1, 2, 3, 6, 8, 9}
B6	{1, 3, 4, 5, 6, 7}	{3, 4, 5, 7, 10, 11}

Step 6: Out B) = In[B] − kill[B] ∪ gen[B]

Block	In[B]	Out[B]
B1	∅	{1, 2}
B2	{1, 2, 3, 4, 5, 6, 7}	{1, 2, 3, 4, 6, 7}
B3	{1, 2, 3, 4, 6, 7, 8, 9}	{1, 2, 3, 5, 6, 7, 9}
B4	{1, 2, 3, 4, 5, 6, 7, 9}	{1, 3, 4, 5, 6, 7}
B5	{1, 2, 3, 5, 6, 7, 9}	{1, 2, 3, 6, 8, 9}
B6	{1, 3, 4, 5, 6, 7}	{3, 4, 5, 7, 10, 11}

18/4/23

Q. Solve the data flow equation for the following flow graph.



Compute IN & Out of each block

Ans

	In	Out
B1	B2	B2
B2	B1, B5	B1, B3
B3	B2	B4, B5
B4	B3	B5
B5	B3 B4	B2

Step 1 : Gen & Kill

Block	Gen	Kill
B1	{1, 2}	{3, 4, 5}
B2	{3}	{1}
B3	{4}	{2, 5}
B4	{5}	{2, 4}
B5	ϕ	ϕ

Step 2 : In [B] : ϕ , OUT [B] = GEN [B]

Block	In [B]	Out [B]
B1	ϕ	{1,2}
B2	ϕ	{3}
B3	ϕ	{4}
B4	ϕ	{5}
B5	ϕ	ϕ

Step 3 : In[B] = ~~Out~~ [B] , Out [B] = In [B] - Kill [B] U GEN [B]

Block	In [B]	Out [B]
B1	{1, 2}	{1, 2}
B2	{3}	3
B3	{4}	
B4	{5}	
B5	ϕ	

Date
Page

Step 3: Out[B] = In[B] - kill[B] ∪ GEN[B]

Block	In [B]	Out [B]
B1	{3}	{1, 2}
B2	{1, 2}	{2, 3}
B3	{3}	{3, 4}
B4	{4}	{[illegible], 5}
B5	{4, 5}	{4, 5}

Step 4: Out[B] = In[B] - kill[B] ∪ Gen[B]

Block	In[B]	Out[B]
B1	{2, 3}	{1, 2}
B2	{1, 2, 4, 5}	{2, 3, 4, 5}
B3	{2, 3}	{3, 4}
B4	{3, 4}	{3, 5}
B5	{[illegible], 3, 4, [illegible] 5}	{3, 4, 5}

Step 5: out[B] = In[B] - kill(B) ∪ GEN[B]

Block	In[B]	Out[B]
B1	{2, 3, 4, 5}	{1, 2}
B2	{1, 2, 3, 4, 5}	{2, 3, 4, 5}
B3	{2, 3, 4, [illegible] 5}	{3, 4}
B4	{3, 4}	{3, 5}
B5	{3, 4, 5}	{3, 4, 5}