KUMARI AHSHA

USCID: 9488013423

HW2

Flow → Block | Block, Block || Flow.val = Block.val

$Block_0$ → Block, Assign || $Block_0$.cost = Block.cost + Assign.cost
$Block_0$.val = Block.val

→ Assign || $Block_0$.cost = Assign.cost
$Block_0$.val = Assign.val

→ Expr || $Block_0$.val = Expr.val

$Expr_0$ → $Expr_1$ + Term || $Expr_0$.val = $Expr_1$.val + Term.val
$Expr_0$.cost = $Expr_1$.cost + Term.cost

→ $Expr_1$ - Term || $Expr_0$.val = $Expr_1$.val - Term.val
$Expr_0$.cost = $Expr_1$.cost - Term.cost

→ Term || $Expr_0$.val = Term.val
$Expr_0$.cost = Term.cost

→ BExpr || Expr.val = BExpr.val
Expr.cost = BExpr.cost

BExpr → Expr1 relop Expr2 || BExpr.cost = Expr1.cost + cost(RELOP) + Expr2.cost

→ Assign || BExpr.val = Assign.val
BExpr.cost = Assign.cost

→ true || BExpr.val = 1

→ false || BExpr.val = 0

relop → > | < | = | >= | <= | == | !=

Assign → id = Expr || Assign.val = id.val
Assign.cost = COST(load) + Expr.co

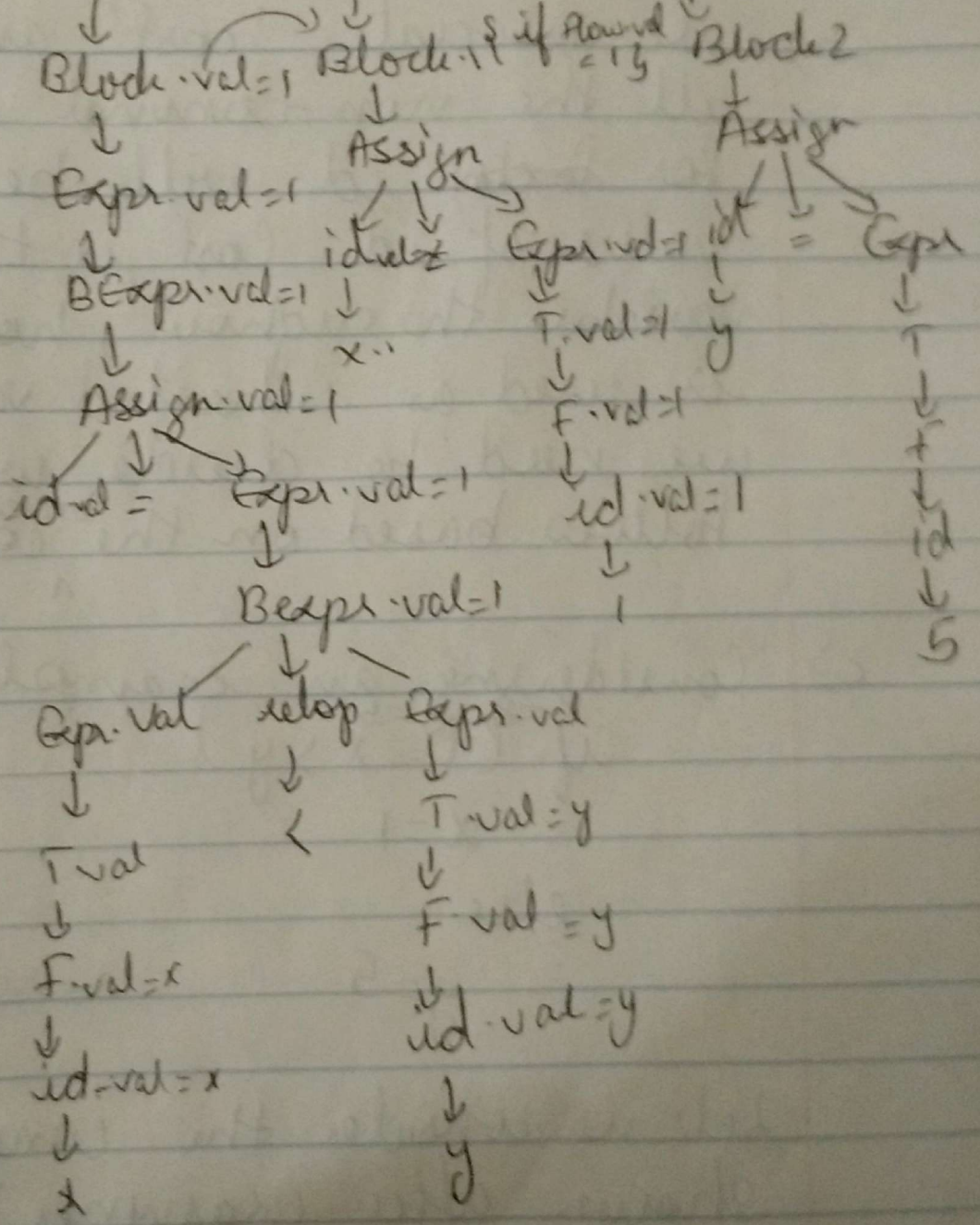
$Term_0 \rightarrow Term_1 * factor$ || $Term_0.cost = Term_1.cost + factor.cost$
$Term_0.val = Term_1.val / Factor.val$

$Term_1 / Factor$ || $Term_0.cost = Term_1.cost + Factor.cost$
$Term_0.val = Term_1.val / Factor.val$

factor || $Term_0.cost = Factor.cost$
$Term_0.val = Factor.val$

factor → id / number || Factor.cost = COST(load)
Factor.val = id.val
Factor.val = number.val

Flow.val = (depending on value of flow)



So, cost flows completely bottom to up as Synthesized where as considering "value" as Inherited attribute as shown above

The Beaps considered here is the boolean expression considering it of type true, false or $a > b$ or $x = a > b$; so according to question it comes out as if (x) or if (a>b) then block of statements

b.) We associate 'cost' and 'val' with all the non terminal symbols. Value for ~~factor~~ id will be the lexical value of id. Cost is the synthesized variable throughout, however val is used as inherited variable when we need to decide which block to follow based on the boolean condition.

c.) Considering an example

```
x=& y=5;
if (t= x > y)
    x = 1
else
    y = 5
```

Let us consider the complete tree as shown after grammer for explanation of cost & value as shown in part a

Considering load tracking, now, we have a central variable for cost

Block → Block Assign
→ Assign — Cost ← 0

Assign → id = E — cost ← cost + COST(load)

E → E + T — cost ← cost + COST(add)

E → E − T — cost ← cost + COST(sub)

E → T

T → T * f — cost ← cost + COST(mul)

T → T + f — cost ← cost + COST(add)

T → f

f → id — cost ← cost + COST(loadI)

f → number

BExpr → E_1 relop E_2 — cost ← cost + COST(relop)

Cost(loadI) = 1
cost(load) = 2
COST(store) = 2
COST(add) = 2
COST(div) = 2
COST(relop) = 1

```
i ← hash(identifier);
if (Table[i].loaded = false)
{
    cost ← cost + COST(load);
    Table[i].loaded ← true;
}
}
```

d.)

1.) If we have limited number of temporaries we can do code optimization before implementing it. Moreover we can use stack and directed acyclic graphs for optimization

2.) If we have unlimited number of temporaries we can use SSA and write a clear implementation for. That will help in finding the part of code which is useless/useful. It also helps in estimation of values.

3.) If we have limited number of temporaries and cyclic control flow we can use stacks and DAG's. We can also reuse the registers depending on the system for cyclic control flow.

2)

$b_1 = 0$;
$d_1 = 1$;
$a_1 = \ldots$;
$i_1 = \ldots$;
L1: $i_2 = \phi(i_1, i_3)$
if ($i_2 <= 0$) goto Lbreak;
$d_2 = 1$
$b_2 = \phi(b_1, b_3)$
$b_3 = b_2 + 1$
$d_3 = 1$
$i_3 = i_2 - 1$
if ($i_3 < 0$) {
$a_2 = \ldots$;
$d_4 = 1$;
goto Lbreak;
}
$b_4 = 0$
goto L1
Lbreak: $a_3 = \phi(a_1, a_2)$
$b_5 = \phi(b_1, b_3, b_4)$
$x_1 = a_3$
$y_1 = b_5$
$d_5 = \phi(d_1, d_3, d_4)$

In this problem d is always 1 so its being reassigned without any use. This information is useful

Q3)

name: main		parent: ←	
kind	symbol	type	size
var	a	integer	4
var	b	integer	4
var	c	integer	4

name: f1		parent: →	
kind	symbol	type	size
param	w	integer	4
param	x	integer	4

name: f2		parent →	
kind	symbol	type	size
param	y	integer	4
param	z	integer	4
var	a	integer	4

name: f3		parent →	
kind	symbol	type	size
param	m	integer	4
param	n	integer	4
var	x	integer	4
var	y	integer	4

name: f4		parent →	
kind	symbol	type	size
param	k	integer	4

For line 19 we refer to table of f2, it takes the value of z from its own table, for value of c it looks at the table of its parent ie. main. The tables are activation records of each function.

84.)

x = sinh(x)

```
double sinh(double x){
  int i, j;
  double y = 0.0;
  for (i=1, j=0; j<N; i+=2, j++)
  {
    y += pow(x,i) / fact(i);
  }
  return y;
}
```

```
t1 = x
putparam t
call sinh, 1
```

```
sinh : getparam x
       y = 0.0
       i = 1
       j = 0
L0:    t1 = j < N
       if not t1 goto L
       putparam i
       t2 = call fact, 1
       put param x
       putparam i
       t3 = call pow, 2
       t4 = t2 $ / t3
       y = y + t4
L1 ; return y
```

Optimizedion

```
double sinh(double x)
{ int i,j;
  double y = 0.0;
  int fact = 1;
  i = 1;
  for (j=0; j<N; j++) {
    y += x / fact;
    x = x * x * x;
    fact = fact * (i+1) * (i+2);
    i += 2; }}
```

//Use of marker is not necessarily required

5.) S → select {code} for id in explist ;
Code → var = fname (paramlist) ;
explist$_0$ → val , explist$_1$;
explist$_0$ → val
id → val
val → float
fname → string
paramlist$_0$ → char, paramlist$_1$
paramlist$_0$ → char
var → string

S.code = append (gen ('goto L0:'), Code.code,
gen ('L0:'));
Code.code = append (gen ('L1:'), gen (var '='
fname), gen ('-'Li), gen('('
paramlist')';), gen ('goto end'))
queue.push ((paramlist[0], Li));

for element in explist {
while (queue not empty)
{
(Vi, Li) = queue.pop;
S.code = append (S.code, gen('if a=Vi
goto Li'));
}
S.code = append (S.code, gen('goto L'
paramlist.size+1));