Note-2 $\boxed{\text{If } L \text{ is non empty, then } \frac{\Sigma^*}{L} = \Sigma^*}$ If L is empty then $\frac{\Sigma^*}{L} = \{\}$. (No matching)

Note-3 $\boxed{\text{If } L \text{ is non-empty, then } \frac{L}{\Sigma^*} = \text{all the prefixes of } L.}$

$$\frac{a}{(a+b)^*} = \epsilon, a$$

$$\frac{TOC}{(A+B+\cdots Z)^*} = \epsilon, T, TO, TOC = \text{all the prefixes of } L.$$

Code Optimization

* Loop optimization
* Strength Reduction
* Redundency Elimination
* Dead code elimination
* Constant folding
* Copy propagation
* Algebric simplification

Loop Optimization

⇓

(1) Loop invariant (Code motion)

(2) Loop unrolling (Decreasing testcases)

(3) Loop jamming (Loop combine)

1. Loop Invariant

```
i=0
while(i <= 10,00,000)
{
  x = Sin(A) * Cos(B) * i
  i = i+1;
}
```

Sin(A) * Cos(B) → not varying (invariant)

⇓ take invariant code outside the while loop.

```
i = 0
t = Sin(A) * Cos(B)
while ( i ≤ 1,00,000)
  {
    x = t * i;
    i = i + 1;
  }.
```

2 Loop Unrolling (Decreasing test cases).

```
while ( i ≤ 10,00,000)
  {
    x[i] = i;
    i++;
  }.
```

10,00,000 test cases ⇒

```
while (i ≤ 10,00,000)
{
  x[i] = i;
  i++;
  x[i] = i;
  i++;
}
```

5,00,000 test cases.

3 Loop Jamming (Loop Combine).

```
i = 1;
while (i ≤ 10,00,000)
 {
   x = a + b * i;
   i = i + 1;
 }
```

10,00,000 Comp

```
i = 1;
while ( i ≤ 10,00,000)
 {
   y = c + d * i
   i++;
 }.
```

10,00,000 Comp

$$\boxed{\begin{array}{l} i = 1 \\ \text{while } (i \le 10,00,000) \\ \{ \\ \quad x = a + b * i; \\ \quad y = c + d * i; \\ \qquad i{+}{+}; \\ \}. \end{array}}$$

$\boxed{\text{Strength Reduction}}$:- Replacing costlier operation by less cost operation or replacing lower speed operator to the higher speed operator

Exp:— n ⇒ *
⇓ ⇓
2 * n left shift

4 * i ⇒ * (lesstime)
⇓ ⇓
i + i + i + i +

Constant Folding :-

Fold all the constants and give one equivalent value.

$a = b + \boxed{5+10+15+25}$

$\Downarrow$

$a = b + \boxed{55}$

Copy Propagation :-

Unnecissarily don't propagate the constant by copying one by one into another variable.

Exp:-

PI = 3.14
x = PI
y = x * 100
Z = 100
a = y/Z

Redundancy Elimination :-

Use DAG Data Structure

A = b+C
B = 2+b+3+C
C = C+1+b

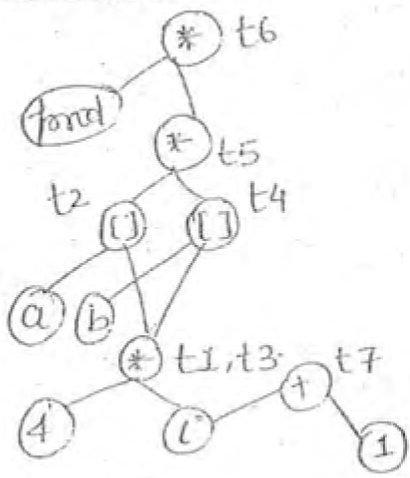
$\Downarrow$

A = b+C
B = 5+A
C = A+1

Exp-2

t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = b[t3]
t5 = t2 * t4
t6 = prod * t5
t7 = i + 1

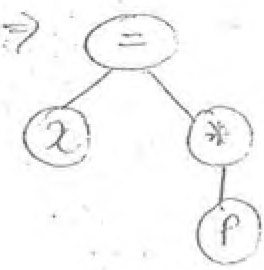
$\Rightarrow$



$\Downarrow$

t3 = 4 * i
t2 = a[t3]
t4 = b[t4]
t5 = t2 * t4
t6 = prod * t5
t7 = i + 1

x = *P $\Rightarrow$



Dead Code Elimination :-

Exp

x = t1 ;
a[t1] = t2
b[t2] = a[t1]
printf (b[t2])

$\Rightarrow$

a[t1] = t2
b[t2] = a[t1]
printf(b[t2])

$\Rightarrow$ x is not at all useful.

Algebric Simplification

$A = A*1$
$B = B+0$ } ⇒ don't use these type of operators

GATE Problems

<u>Q1</u> Consider the following C pgm-

```
for(i=1; i<N; i++)
{
  for(j=1; j<N; j++)
  {
    if(i%2)
    {
      x+= 4*j + 5*i
      y+= 7+4*j;
    }
```

Then which one of the following is false:-

a) above pgm contain loop invariant.

b) above pgm contain common subexpression elimination.

c) above code contain strength reducti

d) None of the above.

- Common subexpression = $4*j$
- Strength Reduction = $j+j+j+j$
- ```
  for(i=1; i<N; i++)
      if(i%2)
  for(j=1; j<N; j++)
  {
      x+= 4*j + 5*i
      y+= 7+4*j
  }
  ```

<u>Q1</u> Multiplication of a positive integer by a power of 2, can be replaced by left shift, which executes faster on most of the processors. This is an example of:-

a) Loop unrolling

b) strength Reduction

c) Dead code Reduction

d) None of above
  ```

Q19 $i = 1, j = 0;$ for the above pgm, involving integers i, j, n, which one of the following is loop invariant:-

while ($j \le n$)

{

$i = 2 * i$

$j = j + 1;$ } N+1

}

⇓

$i = (2)^{j+1}$

i) $i = j + 1$

ii) $i = (j+1)^2$

iii) $j = 2^i$

iv) $i = 2^{j+1}$

Q4.20 $S \to AB | CA$

$B \to BC | AB$

$A \to a$

$C \to aB | b$

Sol^n Reduced form

① Eliminate all the states or variables which are not reachable from start symbol.

↳ $S \to AB | CA$

$B \to BC | AB$

$A \to a$

$C \to aB | b$

② Eliminate those variables and productions, which are unnecessary

$S \to \cancel{AB} | CA$

$\cancel{B \to BC | AB}$

$A \to a$

$C \to \cancel{aB} | b$

⇒

$S \to CA$

$A \to a$

$C \to b$

ϕ LA

Syntax

Semantic

I.C.G.

C.O.

T.C.G.

Chapter No. 5

RUN TIME ENVIRONMENT

Environment
(Binding)

a — 5000 memory location

⇒ variable will be allocated to the multiple location at runtime. Variable will not changed.

f1() → f2() → f3()

(Activation record)

Actual
Return add
Local variable
Temporary variable
non-local
address of calling fun
m/c status

⇒ activation Record

(This are the information that should be to f1() before the control is going to f1() → f2())

Control stack

f3()
f2()
f1()

⇒ all the current active function of the system in same order.

⇒ All of activation record first enter to the control stack

Storage Allocation

i) Static storage Allocation
ii) Stack storage Allocation
iii) Heap storage allocation

→ memory created only once (static variable) = Compilation time memory allocation
⇓
Can't be allocated at run time

* Static storage Allocation

→ Memory is allocated at compilation time only
→ Bindings do not change at run time
→ One activation record for procedure
→ Recursion is not supported
→ Size of the object must be known as compile time itself (one time allocation of add)
→ Data structures can not be created dynamically (not allocated and deallocated dynamically)

Exp:-

F(1)
F(2)
F(3)

2 Stack Storage Allocation:-

→ Whenever is a function is called, activation record is created and pushed it into the stack.

→ Whenever a function ends, activation record is poped out from the stack.

→ At the time of running memory location will change.

→ Locals are bound to new activation record.

F(1)
F(2)
F(3)

⇒ 6

Disadvantage:-

Locals can not retained when activation ends i.e. function is over.

3 Heap Allocation

* Allocations and deallocation may be done in any order

Code Optimization
Runtime Environment } ⇒ Book

DBMS
(1) NF
(2) Relational Algebra
(3) Transaction

CN
(1) Data link layer
→ Stop n wait
→ Go back n
→ Selective reject } → Token ring, → Jamming } →
(2) N/W layer
→ Subnetting
→ Supernetting
(3) Transport layer

OS
→ process magmt
→ Scheduling
→ Synchronization
→ memory magmt
→ Page replacement
→ Disk scheduling

Algo
$\boxed{\text{Notes}}$

DS
$\boxed{\text{Notes}}$

Compiler
$\boxed{}$

Digital
Complete.

CO
i) Pipelining
ii) Addressing modes
iii) Memory magmt
iv) Floating point

Maths
→ Matrices
→ Graph
→ Reln and funn
→ Lattices
→ Group theory

Apti

(4) Web Technology
XML, HTML } W3school.com

(5) S/W Engg
→ Cyclomatic
→ Cocomo model

COMPILER

(42)