a handle of $\alpha\beta w$ in the position following α

Handle Pruning:

* A right most derivation in reverse can be obtained by handle pruning.
* i.e) If w is a sentence of the grammar to parse then $w = \gamma_n$ where γ_n is the nth right sentential form of right most derivation

$$S \Rightarrow \gamma_0 \underset{rm}{\Rightarrow} \gamma_1 \underset{rm}{\Rightarrow} \gamma_2 \Rightarrow \dots \Rightarrow \gamma_{n-1} \underset{rm}{\Rightarrow} \gamma_n$$

$$= w$$

The Closure Operation:

* If I is a set of items for a grammar G then closure (I) is the set of items constructed from I by the two rules
  1) Initially every item in I is added to closure (I)
  2) If $A \rightarrow \alpha \cdot B\beta$ is in closure (I) and $B \rightarrow \gamma$ is a production then add the item $B \rightarrow \cdot\gamma$ to I, if it is not already there. We apply this rule until no more new items can be added to closure (I).

The Goto Operation:

* goto (I , X) where

  I → is the set of items

  X → is a grammar symbol.

* goto (I , X) is defined to be the closure of all items $A \rightarrow \alpha X \cdot \beta$ such that $A \rightarrow \alpha \cdot X \beta$ is in I.

Applications of DAG:

1) We automatically detect common subexpressions
2) We can determine which identifiers have their values used in the block
3) We can determine which statements compute values that could be used outside the block
4) Another important use to which the dag may be put is to reconstruct a simplified list of quadruples taking advantages of common sub expression and not performing assignments of the form $x = y$ unless necessary
5) If there are additional attached identifiers $y_1, y_2, \ldots, y_K$ for a node n whose values are also needed outside the block, we assign to them with statements

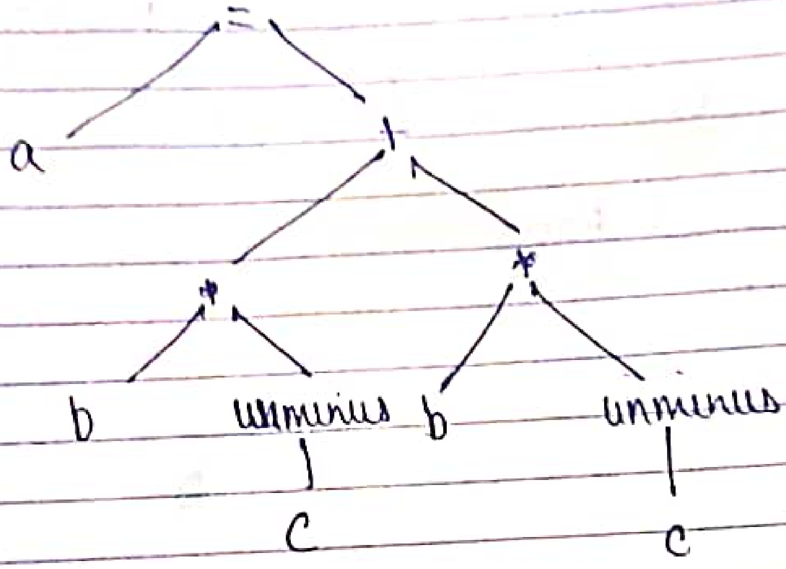
$$y_1 = x, \quad y_2 = x, \quad \ldots \quad y_K = x$$

If n has no attached identifiers we create a new temporary name to hold the value of n.

(am)

Cal. 2

4) Syntax tree => $a = b * -c + b * -c$



5) Three address code:

$a = b * \text{unminus } c + b * \text{unminus } c$

temp1 = unminus c

temp2 = b * temp1

temp3 = unminus c

temp4 = b * temp3

temp5 = temp2 + temp4

a = temp5

6) a) Linear representation - postfix Notation

b) graphical " " - Syntax tree & DAG

c) Virtual Machine " " - Static Machine code

d) 3-address Statements - 1) Quadruples
2) Triples
3) Indirect Triples

Deadcode Elimination:

* A variable is live at a point in a program if its value can be used subsequently. Otherwise it is dead at that point.

* For eg)

```
debug = false
   :
if (debug)
  print ...
```

* Each time the program reaches the statement the value of the debug is false

* The print statement is dead because it cannot be reached.

* copy propagation followed by dead code elimination removes the assignment to x and transforms into

$a[t_2] = t_5$

$a[t_4] = t_3$

goto B_2

1 Code motion:

* Code motion moves code outside a loop.
* A modification that decreases the amount of code in a loop is code motion.
* This transformation takes an expression that yields the same result independent of the number of times a loop is executed. (a loop invariant computation) and places the expression before the loop.
* For eg) evaluation of limit-2 is a loop invariant computation in

while (i < = limit - 2)

code motion will result

t = limit - 2

while (i < = t)

... elimination & Reduction in strength

BASIC BLOCKS AND FLOW GRAPHS:

* A graph representation of three address statements called a flowgraph, is useful for understanding code generation algorithms
* Nodes in the flow graph represent computation and edges represent the flow of control

Basic blocks:

* A basic block is a sequence of consecutive statements in which flow of control enters at the beginning and leaves at the end without halt or possibility of branching except at the end.

* Example:

$$
\begin{aligned}
t_1 &= a * a \\
t_2 &= a * b \\
t_3 &= 2 * t_2 \\
t_4 &= t_1 + t_2 \\
t_5 &= b * b \\
t_6 &= t_4 + t_5
\end{aligned}
$$

* A three address statements $x = y + z$ is said to <u>define x</u> and to <u>use</u> (or reference) y and z.
* A name in a basic block is said to be live at a given point if its value is used after that point in the program.

Flow graphs:

* We can add the flow-of-control information to the set of basic blocks making up a program by constructing a directed graph called a flow graph
* The nodes of the flow graph are basic blocks
* One node is distinguished as initial, it is the block whose leader is the first statement.
* There is a directed edge from Block B_1 to block B_2 if B_2 can immediately follow B_1 in some execution sequence. that is if
  1) there is a conditional or unconditional jump from the last statement of B_1 to the first statement of B_2 or
  2) B_2 immediately follow B_1 in the order of the program, and B_1 does not end in an unconditional jump.
* B_1 is a predecessor of B_2
* B_2 is a successor of B_1

+ Example:

```
prod = 0
i = 1
```
B_1

```
t1 = 4 * i
t2 = a[t1]
t3 = 4 * i
t4 = a[t3]
t5 = t2 * t4
t6 = prod + t5
prod = t6
t7 = i+1
i = t7
if i<=20 goto B2
```
B_2

Loops:

* A loop is a collection of nodes in such that

statements

Register and Address Descriptors:

- The code generation algorithm uses descriptors to keep track of registers contents and addresses for names.

1. A register descriptor keeps track of what is currently in each register. It is consulted whenever a new register is needed. We assume that initially the register descriptor shows that all registers empty.
2. An address descriptor keeps track of the location where the current value of the name can be found at run time. The location might be a register, a stack location, a memory address or some set of these.

Instruction Cost:

* The cost of an instruction to be one plus the cost associated with the source and destination address modes. (indicated as added cost in the table for address modes above).

* Example:

* The statement a = b+c can be implemented by many different instruction sequences.

1. MOV b, R0
   ADD c, R0
   MOV R0, a

   cost = 6

2. MOV b, a
   ADD c, a

   cost = 6

Assuming R0, R1, and R2 contain the addresses of a, b, and c respectively

3. MOV *R1 *R0
   ADD *R2 *R0

   cost = 2

Assuming R_1 and R_2 contain the values of b and c respectively, and that the value of b is not needed after the assignment

4) ADD R2 R1
   MOV R1 a

Evaluation Order

picking up the best Order } = NP- complete problem

• Instruction cost = 1 + cost associated with the source or destination

$$a = b + c \Rightarrow \begin{array}{llll} & 1 & 1 & 0 \\ \text{MOV} & b & R_0 & = (1+1+0) = 2 \\ \text{ADD} & c & R_0 & = (1+1+0) = 2 \\ \text{MOV} & R_0 & a & = (1+0+1) = 2 \\ & & & \overline{\quad 6 \quad} \end{array}$$