# Flow Analysis

66.648 Compiler Design Lecture (04/08/98)

Computer Science

Rensselaer Polytechnic

# Lecture Outline

- Flow Analysis
- Examples
- Administration

# Control Flow Graph

A control flow graph (CFG) or a flow graph is a directed multigraph in which nodes are basic blocks and edges represent flow of control (branches or fall -through execution).

The basic block whose leader is the first Initial Leader statement is called the start node.

There is a directed edge from Basic block b1 to basic block block b2:

1) There is a branch from the last statement of B1 to the first statement of B2.

# Control Flow Graph

2) Control flow can fall through B1 to B2 because B2 immediately follows B1 and B1 does not end in a transfer statement.

# Example

```
Public class test {
public static void main(String argsv[]) {
int I,j,k,counter;
 I = 78;
j = 89;
k = j;
for (counter=1;counter<I;counter++) k=k+j;
}
```

# Example

```
I = 78
j = 89
k = j
counter=1
loop: if (counter>I) goto  out
k=k+j
counter=counter+1
goto Loop
out:
```

# Comment

The control flow graph may contain cycles. There can also be multiple edges going from one basic block to another. One distinguishes these edges by placing labels on the edges.

# Live Variable Analysis

A point in a control flow graph is a code position immediately before a basic block (basic block entry) or immediately after a basic block (basic block exit).

A variable X is live at point P (or, more precisely, the value if variable X is live at point P) if the value of X at point P may be used along some directed path in a Control Flow Graph(CFG) that starts at P.

in[B] = set of variables live on entry to block B

# Live Variable Analysis

Out[B] = set of variables live on exit to block B.

Motivation: if the value of a variable is not live (dead), then that variable need not be stored in memory or in a register. That register can be used to store some other variable.

# Example

# Definition and Uses of def and use

Def[B] = set of variables V, such that V is defined in block B before any instruction in B uses V.

use[B] =set of variables V, such that V is used in block B before any instruction in B uses V.

Recursive equations for in and out.

In[B] = use[B] U (out[B] - def[b])

Out[B]= U in[s] where s is a successor block of B.

We want to find the fixed points(or the smallest in

and out sets ] satisfying the above two equations.

# Algorithm for computing In and Out

This is an iterative algorithm

for each basic block B do in[b] = {} ;

change=true;

while change do {

change=fales; for each basic block do {

out[b]={};

for each successor S of B do out[b]=out[b]U in[s]

oldin= in[B];

# Algorithm for computing In and Out

In[B] = use[B] U (out[B]-def[B])

if in[B] != oldin { change = true;}

}

}

This algorithm will converge.

# Simple Register Assignment

The following simple register assignment strategy can improve the quality of code generated.

Global Scheme:

1) Select a fixed number of variables/temporaries from the program's intermediate language for register assignment.

Selection is based on the variables that are used in the inner loops. For each variable x, the amount of savings will be

sum over basic blocks in a loop( use(x)+2*live(x));

# Intuition

What is the saving by keeping a variable in a register during the duration of a loop. If a variable is live at the beginning of a block and at the exit of a block (inside a loop), then we save two units- one for loading and the other for storing. For each use of that variable, we save one unit.

# Example

Example 9.6 page 543

# Outer Loops

Once the variables are assigned registers in the inner loop, use a similar strategy to assign variables to registers.

One can formulate register allocation as a graph coloring problem, by constructing a register interference graph. Nodes of this graph are variables and there is an edge from one node to another, if one variable is live, where the other variable is defined.

# Ordering the Computations

Reordering a computation can affect the cost of the resulting code.

Eg:

t1 := a+b

t2:=c+d

t3:=e-t2

t4:=t1-t3

What will be the resulting code size if we have two

registers.

# Rearrange the order

T2:=c+d

t3:=e-t2

t1:=a+b

t4:=t1-t3

How much cost you save. This is because of

the register spilling.

Qn: Can you write an expression that needs

3 registers no matter what order.

# Example

# Comments and Feedback

Projects 3 and 4 are out. Please start working. PLEASE do not wait for the due date to come.

We are in chapter 9. Please read that chapter and read the relevant portion of Java. Please keep studying this material and work exercises.