Common Sub-expression Elimination

66.648 Compiler Design Lecture (04/14/97)

Computer Science

Rensselaer Polytechnic

Lecture Outline

- Definition Use (d-u) Chains
- Available Expression Analysis
- Global Common Sub-expression elimination
- Examples
- Administration

Basic Idea of d-u chain

The d-u chain for a definition (or assignment of value to a variable) is a list of possible uses of the definition I.e., a list of all read accesses to the variable that can retrieve the value stored by the definition.

Motivation

- One application of du-chains is to identify defs with empty du-chains I.e, those defs have no uses. Such defs can be removed to obtain dead-store elimination. (such defs are not live variables.)
- du-chains can be used to calculate data-dependencies.
- Du-chains can be used for faster algorithms for constant propagation and other data flow analysis algorithms.

Computing du-chains

Du-chains can be computed by an algorithm similar to live variable analysis. In, out, def and use are sets of variables in live variable analysis. In computing du-chains, we use sets of triplets. A triplet is a (variable, Ilstmt#, operand#).

Example

a=1;

if (x < y) { b = 2; c=a +b;}

else { b = 4; c= b-1;}

d=a+c;

e=b+c;

Example

Algorithm

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

Each element is a triple.

This is the same live variable algorithm.

Ex: [7] d:=a+c [8] d:=b+d

If out[b] = {}

in[b]={(a.7,1),(c,7,2),(b,8,1)} why didn't d appear in the in[b]?

More definitions

The du-chain for a def d of variable X in basic block b is then computed according to the following cases:

1) There is a following def,d’, of variable X in basic block B ==> du-chain(d) = set of triples of X between d and d’.

2) Otherwise,

==> du-chain(d) = (set of triples of X in out[b]) U (set of triples of X that follow d in B)

Examples

Example of computing du-chain SETS

d	du-chain(d)
def of a in [1]	{ (a,4,1),(a,7,1)}
def of b in [3]	{(b,4,2),(b,8,1)}
def of c in [4]	{(c,7,2),(c,8,2)}
def of b in [5]	{(b,6,1),(b,8,1)}
def of c in [6]	{(c,7,2),(c,8,2)}

Elimination of Global Common Sub-Expression

Motivation: Redundant occurrences of cse can be eliminated at compile-time, thus saving the cost of computing this at run-time. The savings obtained by eliminating cse is proportional to the number of times the basic block containing that statement is executed.

Example

```
if (x < y) { t1 = 4*I; …}
   else { t2=4*I;…}
t3=4*I;
```

The statement t3:=4*I, need not re-evaluate 4*I provided I is not red-defined in the conditional branches.

Available Expression Analysis

An expression x op y is available at point p if every directed path from start node to p evaluates x op y and after last such evaluation there is no re-assignment to x or y.

A basic block generates expression x op y if it evaluates x op y and does not subsequently redefine x or y.

A basic block kills expression x op y, if it assigns to x or y and does not subsequently recompute x op y.

Available Expression in Basic Block

Available expression at point p = available expression at previous point p' - ae killed between p and p' U ae generated between p' and p.

e.g.	Gen expr	killed expr	avai expr
a :=b+c	{b+c}		{b+c}
b:=a-d	{a-d}	{b+c}	{a-d}
c:=b+c	{}	{b+c}	{a-d}
d:=a-d	{}	{a-d}	{}

Algorithm

In[b], out[b] containing available expressions

e_gen[b], e_kill[b], set of available and generated expressions killed in a block.

Out[b]= e_gen[b]   U (in[b]-e_kill(b))

in[b] = intersection (out(p)), where p is the set of all predecessors of block b.

in[starting_basic_block] = {}

Algorithm

In[b1] ={}

out[b1]=e_gen[b1]

for each basic block b != b1 out[b] = universal set of expression - e_kill[b]

change=true;

while change { change=false; for each basic block b !=b1 in dfs order for each variable do

in[b]=universal set of expressions. For each predecessor p of b do in[b]=in[b] intersect out[p]

Algorithm- contd

Old_out=out[b]

out[b]= ie_gen[b] U (in[b]-e_kill[b])

if out[b] != old_out change =true;

}

Algorithm for Global cse

1) Perform available expression analysis

2) For each basic block b do

For each expression e in B that is available at the start of B and is not killed prior to e's position in B do

a) perform a reverse dfs backwards from b till we visit all ancestors that evaluate e.

b) Generate a new temporary t'1 and replace the evaluation of e in B by the use of t'.

Algorithm for Global cse

C) For each ancestor block A that evaluates e do

replace the last evaluation of e in a with t'=e and t=t'

end for

end for

end for

Example

If (x<y) { t'=I*4*I; t1=t',…}

else { t'=I*4;t2=t' … }

t3=t'

Comments and Feedback

Project 3 is out. Please start working. PLEASE do not wait for the due date to come.

We are in chapters 9 and 10. Please read that chapter and read the relevant portion of Java. Please keep studying this material and work exercises.