Spring 2010, CS 255/455 Homework 1

Feb. 24, 2010

due 2pm Monday March 1st at the start of the class

Question 1 Value numbering (20 points):

1. Give the definition of extended basic block
2. Give an example of redundancy that cannot be removed by value numbering.

Answer:

Question 2 Live Analysis (20 points):

The present GCC compiler can identify uninitialized variable uses when it is invoked with the optimization flag on. Given the example C program file, uninit.c,

```
int f(int b) {
  int i;
  return b==0?  i:  0;
}
```

When compiled by `gcc -c -O -Wuninitialized uninit.c`, the compiler generates
`uninit.c:3:  warning:  .i.  is used uninitialized in this function`

The warning is generated by data flow analysis, in particular, Live analysis. In fact, GCC cannot support the flag -Wuninitialized if -O is not specified.

1. Describe an algorithm to identify the set of variables that may be used without initialization. To make the problem simpler, your augmented analysis is required to identify only the name of the variables, not the location of the uninitialized use.

2. **(455 required, 255 extra credit)** Data flow analysis conservatively assumes that all control flow paths through a control flow graph may happen, which leads to imprecision when some control flow paths are actually impossible due to program logic. Show that the -Wuninitialized feature in GCC may produce false positives by giving an example program in which no variable can be used before initialization but the compiler analysis still generates a warning of a possible use of uninitialized data.

Answer:

Question 3 SSA (20 points):

Give the control flow graph of the following pseudo code. List the basic steps of SSA (static-single assignment) form construction. Convert the code into the SSA form. Remember to convert the code into a control flow graph first.

```
s = 1
r = 1
j = 2
while (j<n) {
   s = s + r
   j++
   if (j==n) {
     r = s
     break
   }
   r = s + r
   j++
}
print r
```

Answer:

Question 4 Constant Propagation (20 points):

(455 required, 255 extra credit)

1. Design an iterative solution for constant propagation, which takes in a def-use graph and identifies variable assignments that always produce constants. Show that your algorithm converges. Hint: analyze each variable assignment and represent its "constancy" by values in a 3-tier lattice, where the top is maybe constant, the middle are constant values, and the bottom is not constant.

2. Can your algorithm discover all constants? If not, give an example where your algorithm fails to discover a constant assignment.

Answer: