15-745: Spring 2011

Homework 3

Salil Joshi
salilj@andrew.cmu.edu

Cyrus Omar
cyrus@cmu.edu

1 Dead Code Elimination using FVA

We used the dataflow framework developed in the previous homework to implement faint variable analysis (FVA). The results from this analysis are used to eliminate dead code.

1.1 FVA

FVA is carried out using a backward pass. Thus we subclass our `Dataflow`, passing it the template paramter `false`. The in and out sets are `BitVector`s, with one bit for each variable in the function. If the bit corresponding to a variable is set (i.e. true) then the variable is faint.

The meet operator simply takes the bitwise $\wedge$ of the bitvectors, thus computing the intersection of the sets represented by the BitVectors. This is because a variable is faint if and only if it is faint along *every* path following its definition. The top element is thus a `BitVector` with all bits set, and bottom is one with all bits unset.

Initially, we mark *all* variables as faint. The transfer function then marks a variable as faint if it used in an instruction with a non-faint LHS or a function call or control flow instruction. We cannot eliminate function calls (and thus we cannot mark their arguments as faint) even if the LHS is faint because the called function might have side effects.

For store instructions, the value operand is marked as not faint if the pointer operand is not faint, *or* if it was not locally allocated. Note that a store to an address given by a faint variable should *not* be removed if the variable is a global variable or an argument, because the effects of this store may be used by code outside the function being analysed. For simplicity, we ignore aliasing.

1.2 Dead Code Elimination

After FVA analysis is complete, we know that all instructions (almost; exceptions below) that define a faint variable can be removed. Further, removing these instructions is safe because all uses of the defined variable can only be in instructions defining faint variables and thus these uses too will be removed.

There are two exceptions to this rule:

- Function Calls
- Stores to global variables or arguments

We cannot remove these instructions as they may have effects outside the function even if they are useless within the function.

So, elimination is done by iterating through all instructions in the function (backwards, to ensure that uses are removed before definitions) and erasing instructions defining faint variables, unless they meet the two criteria outlined above.

1.3 Evaluation