

CS 671 Graduate Seminar

Challenge Problem 2

Sample Solution

UD/DU Chains and Constant Propagation

Extend constant propagation algorithm discussed in class to only propagate constants from code regions that are not dead code.

Here is the outline of one possible algorithm:

Add a “field” or flag *reachable* to every node in the CFG. Initially, all nodes with the exception of the start node are marked not reachable, i.e., *reachable* = *false*. Propagate constants along DU chains only if the statement that generates the definition is marked *reachable*. Statements are marked reachable through an rPOSTORDER walk of the CFG. If a branch is reached with a conditional that is constant, the rPOSTORDER walk will only consider nodes within the branch that will be executed. Nodes in the other branch are not visited, and therefore will never be marked reachable.

Lattices and MFP

1. Show that the bit-vector problem of Reaching Definitions is distributive, i.e., show that for propagation function f and lattice elements x and y , $f(x \wedge y) = f(x) \wedge f(y)$.

The propagation function for an arbitrary basic block has the form

$$f(x) = \text{GEN} \cup (x - \text{KILL})$$

The confluence operator \wedge is set union \cup for the Reaching Definitions problem.

$$\begin{aligned} f(x \cup y) &= \\ \text{GEN} \cup ((x \cup y) - \text{KILL}) &= \\ \text{GEN} \cup \text{GEN} \cup (x - \text{KILL}) \cup (y - \text{KILL}) &= \\ \text{GEN} \cup (x - \text{KILL}) \cup \text{GEN} \cup (y - \text{KILL}) &= \\ f(x) \cup f(y) \end{aligned}$$

2. Prove that for all fixpoints $p = f(p)$, $f^k(0) \leq p$, where f is a monotone function over a lattice L .

p is a fixpoint, i.e., $\forall t: f^t(p) = p$.
Since 0 is the bottom element, $0 \leq p$ holds.
Due to monotonicity, $f^k(0) \leq f^k(p) = p$

Iterative Algorithms

1. For a rapid bit-vector problem, give the shortest node list and the longest node list in order to reach the maximal fixed point.

For rapid data flow problems, propagating data flow information along any acyclic path in the CFG is sufficient in order to reach the fixpoint. This fact is used by the nodelist iterative algorithm.

For our formulation of the data flow equations (by visiting node n we actually apply the propagation functions of all its predecessors/successors), we do not have to enumerate all acyclic paths as a subsequence in the node list. Basically, the sources of back edges are special cases here. If visiting node n only implies that its own propagation function is evaluated, and then, if a change of information happened, all successors/predecessors are inserted in the worklist, we need to propagate along all acyclic paths.

So, there are a set of possible “shortest” paths here.

a shortest node list: a c e g i c e g b d f h j l d f h j l k

a shortest node list: a c e g i c e g b h f j l d f h j l d k

a shortest node list: a c e g i c e g b h f j l d f h j d k

There is no *longest* node list. You can construct an arbitrary long node list by including redundant node visits in the list.

2. Modify the worklist algorithm by introducing a node numbering and node selection criteria from the worklist such that the number of nodes visited is minimized in order to compute the maximal fix point for a rapid bit-vector problem (e.g.: Reaching Definitions).

For a rapid problem, propagating data flow information along acyclic paths is sufficient. Compute an rPOSTORDER node numbering that follows the execution order of the nodes and is biased towards enumerating nodes of inner loops first. Modify the worklist algorithm to select as the next to be processed node in the worklist the one with the lowest rPOSTORDER number. An example numbering is given in the figure below.

Show the sequence of nodes visited for the control flow graph.

processing sequence: 1 2 3 4 5 2 3 4 5 2 6 7 8 9 8 9 10 11 7 8 9 10 11 7 12

