Final Exams

Q1

Initial Thoughts Q1

If the variable is inside contitional (or nested conditinals) then those conditionals resolve to true otherwise the execution would not reach that point. Therefore, those conditionals can be removed as they must be true always for that variable at that program point.

Only those conditionals (control dependance) must remain that modify the value of the variable, outside of the final scope of the variable.

Answer Q1

- Check the type of the basic block at slice criterian
- case if.body:
  - repeat until out of if.body
    - check if this basic block modifies variable (mark this instruction)
    - go to predecessor assume br conditional is always true

Then for other basic blocks we must traverse and keep track of instructions that modify this variable, marking all instructions that contain the variables at the phi instruction definition for slice criterian.

Q2

Initial Thoughts Q2

If the slice criterion is withen a loop.body and the loop.body contains instruction/s that modify variable then each iteration of the loop will have a different value.

Follow all predecessors recursively, keep list of visited blocks that influence the variable. Follow all successors recursively, keep list of visited blocks that influence the variable. If both sets intersection is not the null set we can conclude we are inside a loop that modifies the variable.

This idea can be expanded for nested loops. The scope will only be nested further if there are more basic blocks that are part of a loop body (that can be identified by the above method using a stack of basic blocks) that influence the variable.

Answer Q2

- Check the type of the basic block
  - loop.body

        * for each instruction in loop.body
            · compute intersection between pred and succ
            · check if slice criterian is modified (mark this instruction)
            · determine scope
            · mark instructions in pred as necessary if scope is in loop.body that modifies variable

Q3

Initial Thoughts Q3

func visit_basic_block(basic_block, stack)

- ps = number of predecesors of current basic block
- $phis$ = number of phi instructions in current basic block
- $n = ps * phis$
- $Stack < Instruction.PHI > stackarr[n]$; // Stack data structure array that stores phi instructions
- bb_pred_list = basic block predecesor list
- $i = 0$;
- foreach $inst$ in basic block:
    - if $inst$ is a Instruction.PHI then
        * new int $pss = ps$; // pss is a dynamic integer variable with ps as value
        * while j in range(0, ps)
            · $stackarr[(i * ps) + j].push(tuple(inst, \&pss))$
            · spawn thread of func visit_basic_block with params $bb\_pred\_list[j]$ and $stackarr[(i * ps) + j]$
    - if $inst$ is a Instruction.Switch or Instruction.Br
        * $my\_tuple = stack.peek()$;
        * if $(snd(my\_tuple) > 0)$
            · $*(snd(my\_tuple)) - -$; // assume mutex safe
            · destroy thread
        * else
            · $phi\_inst = stack.pop()$
            · $phi\_inst.add\_predicate(inst.conditional)$
    - $i + +$;

Main function

- Start with last instruction

Answer Q3

See algorithm.

Q4

Initial Thoughts Q4

The concept of static single assignment form stems from divergent branches that involve the same variable. So that when these divergent branches converge at basic block the variable can have different values. So the phi instruction occurs at join points. Divergent edges, i.e. multiple paths from the same basic block arise from branch instructions. Unconditional branching is not applicable as only one branch is taken.

Answer Q4

So, there must be a conditional.

Q5

Initial Thoughts Q5

```
int F(int a, int b, int c) {
  x = a;
  while (x < b) {
    x += c * c;
  }
  return x;
}
```

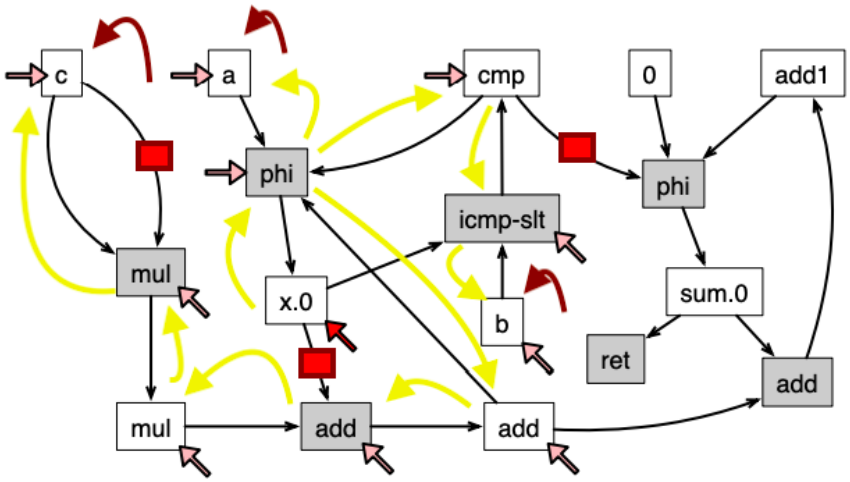

Figure 1: q5

1. Traverse the dependance graph starting with the x.0 node.
2. Visit each predecesor, keep a list of visited nodes.

3. Stop when you reach NULL, or when you reach a visited node again.
4. At the end of a dependance graph see the free variables associated with the phi instruciton of our variable, in this case (a,b,c).

Answer Q5

a Q5

1. NULL
2. revisit a node

b Q5 The backward traversal of the dependance graph shows the 3 variables that are needed by our candidate variable.

Q6

Initial Thoughts Q6

Edges that go to a dead basic block All successor edges from dead basic blocks

Answer Q6

a Q6 Edges that go to a dead basic block. All successor edges from dead basic blocks.

b Q6 If a live basic block (a) has a successor that goes through a dead basic block (b) back to a alive basic block (c) then that region will be squashed. I.e. the region between a and c, all edges will be deleted and a new edge from a to c will be created.