

# CSC2/455 Software Analysis and Improvement

## Dominators and SSA Form - II

Sreepathi Pai

URCS

February 11, 2019

# Outline

Review

Dominator Trees and Frontiers

Minimal SSA Form

Emitting code for SSA form

Postscript

# Outline

Review

Dominator Trees and Frontiers

Minimal SSA Form

Emitting code for SSA form

Postscript

# Static Single Assignment Form

- ▶ Simple algorithm to generate SSA form
  - ▶ Introduce  $\phi$  functions
  - ▶ Rename variables using Reaching Definitions
- ▶ Algorithm can generate excessive  $\phi$  functions
  - ▶ TODAY: Use dominance frontiers to place the minimal number of  $\phi$  functions

# Outline

Review

Dominator Trees and Frontiers

Minimal SSA Form

Emitting code for SSA form

Postscript

## Recall: Dominators

- ▶ A node  $n$  in the CFG dominates a node  $m$  iff:
  - ▶  $n$  is on all paths from entry to  $m$
  - ▶ by definition, a node  $n$  always dominates itself
  - ▶ if  $n \neq m$ , then  $n$  *strictly* dominates  $m$
- ▶ Computed using a dataflow-style analysis
  - ▶ Each node annotated with a set of its dominators

# Dominance Frontiers

- ▶ The dominance frontier of a node  $n$  ( $DF(n)$ ) is a *set* of nodes
- ▶ A node  $m \in DF(n)$  iff:
  - ▶  $n$  does not strictly dominate  $m$
  - ▶  $n$  dominates  $q$  where  $q \in \text{pred}(m)$
- ▶ Note that dominance frontiers only contain *join* nodes
  - ▶ I.e. nodes with multiple predecessors
- ▶ Computing the dominance frontier of each node:
  - ▶ Iterative Data-flow analysis?

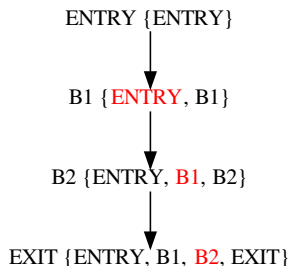
# Dominance Frontiers: Direct algorithm

Direct calculation of dominance frontiers using *dominator trees*.



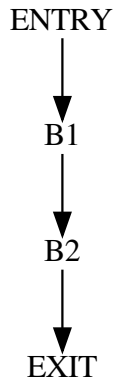
# Immediate Dominators

- ▶ The *immediate* dominator of a node  $m$  ( $IDOM(m)$ ) is the node  $n$ :
  - ▶ such that  $n$  strictly dominates  $m$ , and
  - ▶  $n$  does not strictly dominate  $o$  where  $o \in (DOM(m) - \{m\})$
  - ▶ in some sense,  $n$  is the “closest” dominator in the CFG to  $m$ .
- ▶ By definition, ENTRY has no immediate dominator

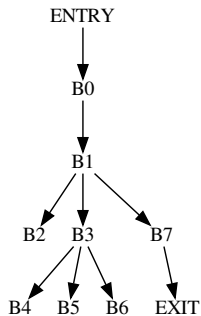
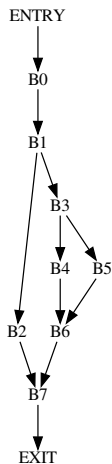


# Dominator Trees

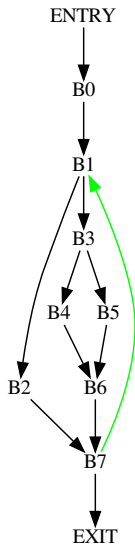
- ▶ Note that each node in the CFG can have only one immediate dominator
  - ▶ Can you see why?
- ▶ Create a graph  $G = (V, E)$ , where:
  - ▶  $V$  is the set of basic blocks
  - ▶ There is an edge  $(n, m)$  in  $E$  if  $n$  is the immediate dominator of  $m$  (i.e.  $IDOM(m) = n$ )



## Example: CFG and its dominator tree



Which node is the immediate dominator of B1?



# Computing the dominance frontier

- ▶ Find all join nodes in CFG, e.g.  $j$
- ▶ For all nodes  $n$  that dominate predecessors of  $j$ 
  - ▶ If  $n$  does not strictly dominate  $j$ , add  $j$  to  $DF(n)$
- ▶ This last step can be operationalized as:
  - ▶ Start from a predecessor  $p$  in the dominator tree
  - ▶ Add  $m$  to  $DF(p)$
  - ▶ Move up the tree and repeat until you reach  $IDOM(j)$

# Outline

Review

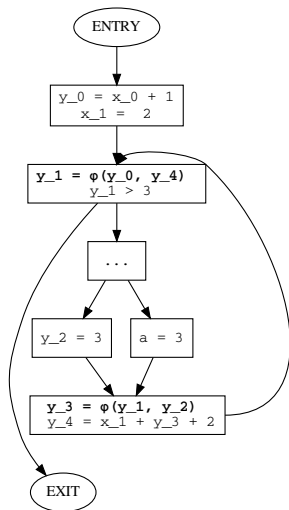
Dominator Trees and Frontiers

**Minimal SSA Form**

Emitting code for SSA form

Postscript

## Example: Non-redundant $\phi$ functions



## Placing $\phi$ functions

- ▶ For each definition  $d$  in basic block  $n$ :
  - ▶ Place a  $\phi$  function for  $d$  in all nodes  $m$  where  $m \in DF(n)$
  - ▶ Note that each  $\phi$  function is also a definition!
  - ▶ Repeat, until no more  $\phi$  functions need to be inserted
- ▶ This is the minimal number of  $\phi$  functions for a definition  $d$ 
  - ▶ Can we reduce the overall number of  $\phi$  functions?
- ▶ (Figure 9.9 in Cooper and Turczon)



# Other optimizations

- ▶ Dead definitions
  - ▶ Definitions that are not read (i.e. overwritten) do not need  $\phi$  functions
- ▶ Two forms:
  - ▶ *Semi-pruned SSA* form, using “globals” names (those variables that are live in to a block)
  - ▶ *Pruned SSA* form, using `LIVEOUT` information

# Outline

Review

Dominator Trees and Frontiers

Minimal SSA Form

**Emitting code for SSA form**

Postscript

# Renaming variables

- ▶ SSA form introduced “subscripts” for each variable
- ▶ Should we drop them when generating code?

```
a_0 = x_0 + y_0
```

```
b_0 = a_0
```

```
a_1 = 17
```

```
c_0 = a_0
```

## Problem with dropping subscripts

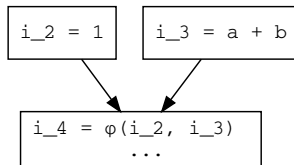
```
a = x + y  
b = a  
a = 17  
c = a    # WRONG!
```

# Handling subscripts

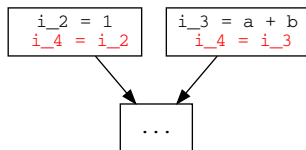
- ▶ Each definition becomes a new variable
  - ▶ I.e. Do NOT drop subscripts
- ▶ Preserves data dependences
  - ▶ Esp. important when we aggressively move code from basic blocks (e.g. very busy expressions, loop invariant code motion, etc.)

## Code for $\phi$ functions

- ▶ Introduce copies along each incoming edge to a join node



## Inserting appropriate copies along incoming edges



## Critical edges

- ▶ Executing  $\phi$  functions by inserting copies into predecessor blocks is not always correct
- ▶ If the predecessor of a block containing  $\phi$  functions has multiple successors, then the  $\phi$  function may execute when it shouldn't
  - ▶ This *may* be harmless, but not always
- ▶ Edges connecting such predecessors to the block contain the  $\phi$  function are called *critical* edges
- ▶ Such edges need to be *split* by inserting a block on that edge
- ▶ See the discussion in Cooper and Turczon for more details and an example



# More complications

- ▶ Excessive copies
  - ▶ Copy propagation into  $\phi$  functions
  - ▶ Note args in resulting  $x_1 = \phi(x_0, y_1)$   $\phi$  functions are for different variables

# Outline

Review

Dominator Trees and Frontiers

Minimal SSA Form

Emitting code for SSA form

Postscript

# References

- ▶ Chapter 9 of Cooper and Turczon
  - ▶ Section 9.2.1
  - ▶ Section 9.3