Where Do We Place $\phi$-Functions?

**Basic Rule**
- If two distinct (non-null) paths $x \rightarrow z$ and $y \rightarrow z$ converge at node $z$, and nodes $x$ and $y$ contain definitions of variable $v$, then a $\phi$-function for $v$ is inserted at $z$.

$$v_1 := \ldots \quad v := \phi(v_1, v_2) \quad \ldots v_3 \ldots$$

Approaches to Placing $\phi$-Functions

**Minimal**
- As few as possible subject to the basic rule.

**Briggs-Minimal**
- Same as minimal, except $v$ must be live across some edge of the CFG.

**Pruned**
- Same as minimal, except dead $\phi$-functions are not inserted.

What’s the difference between Briggs Minimal and Pruned SSA?

Machinery for Placing $\phi$-Functions

Recall Dominators
- $d \text{ dom } i$ if all paths from entry to node $i$ include $d$.
- $d \text{ sdom } i$ if $d \text{ dom } i$ and $d \neq i$.

**Dominance Frontiers**
- The dominance frontier of a node $d$ is the set of nodes that are “just barely” not dominated by $d$; i.e., the set of nodes $n$, such that
  - $d$ dominates a predecessor $p$ of $n$, and
  - $d$ does not strictly dominate $n$.
- $DF(d) = \{ n \mid \exists p \in \text{pred}(n), d \text{ dom } p \text{ and } d \not\text{ sdom } n \}$.

**Notational Convenience**
- $DF(S) = \bigcup_{s \in S} DF(s)$.
**Dominance Frontier Example**

\[ DF(d) = \{ n \mid \exists p \in \text{pred}(n), d \text{ dom } p \text{ and } d \not\text{ sdom } n \}\]

\[ \text{Dom}(5) = \{5, 6, 7, 8\} \]

\[ DF(5) = \{4, 5, 12, 13\} \]

Nodes in \text{Dom}(5)

What’s significant about the Dominance Frontier?
In SSA form, definitions must dominate uses

**Dominance Frontier Example II**

\[ DF(d) = \{ n \mid \exists p \in \text{pred}(n), d \text{ dom } p \text{ and } d \not\text{ sdom } n \}\]

\[ \text{Dom}(5) = \{5, 6, 7, 8\} \]

\[ DF(5) = \{4, 5, 13\} \]

Nodes in \text{Dom}(5)

In this new graph, node 4 is the first point of convergence between the entry and node 5, so do we need a \( \phi \) function at node 13?

**SSA Exercise**

\[ DF(8) = \{10\} \]

\[ DF(9) = \{10\} \]

\[ DF(2) = \{6\} \]

\[ DF(8,9) = \{10\} \]

\[ DF(0) = \{6\} \]

\[ DF(2,8,9,10) = \{6,10\} \]

**Dominance Frontiers Revisited**

Suppose that node 3 defines variable x

\[ DF(3) = \{5\} \]

\[ x \in \text{Def}(3) \]

Do we need to insert a \( \phi \) function for x anywhere else?

Yes. At node 6. Why?
Dominance Frontiers and SSA

Let
- \( DF_1(S) = DF(S) \)
- \( DF_{i+1}(S) = DF(S \cup DF_i(S)) \)

Iterated Dominance Frontier
- \( DF_{\infty}(S) \)

Theorem
- If \( S \) is the set of CFG nodes that define variable \( v \), then \( DF_{\infty}(S) \) is the set of nodes that require \( \phi \)-functions for \( v \)

Algorithm for Inserting \( \phi \)-Functions

for each variable \( v \)
WorkList \( \leftarrow \emptyset \)
EverOnWorkList \( \leftarrow \emptyset \)
AlreadyHasPhiFunc \( \leftarrow \emptyset \)

for each node \( n \) containing an assignment to \( v \)
WorkList \( \leftarrow \) WorkList \( \cup \) \( \{ n \} \)
EverOnWorkList \( \leftarrow \) WorkList

while WorkList \( \neq \emptyset \)
Remove some node \( n \) for WorkList
for each \( d \in DF(n) \)
if \( d \notin \text{AlreadyHasPhiFunc} \)
Insert a \( \phi \)-function for \( v \) at \( d \)
AlreadyHasPhiFunc \( \leftarrow \) AlreadyHasPhiFunc \( \cup \) \( \{ d \} \)
If \( d \notin \text{EverOnWorkList} \)
WorkList \( \leftarrow \) WorkList \( \cup \) \( \{ d \} \)
EverOnWorkList \( \leftarrow \) EverOnWorkList \( \cup \) \( \{ d \} \)

Put all defs of \( v \) on the worklist
Insert at most one \( \phi \) function per node
Process each node at most once

Variable Renaming

Basic idea
- When we see a variable on the LHS, create a new name for it
- When we see a variable on the RHS, use appropriate subscript

Easy for straightline code
\[
\begin{align*}
\text{x} &= \text{y} \\
\text{x} &= \text{y} \\
\text{x} &= \text{y} \\
\text{x} &= \text{y} \\
\text{x} &= \text{y} \\
\text{x} &= \text{y} \\
\end{align*}
\]

Use a stack when there’s control flow
- For each use of \( x \), find the definition of \( x \) that dominates it

Dominance Tree Example

The dominance tree shows the dominance relation

\[
\text{CFG} \\
\text{Dominance Tree}
\]
Variable Renaming (cont)

Data Structures
- Stacks[v] ∀ v
  Holds the subscript of most recent definition of variable v, initially empty
- Counters[v] ∀ v
  Holds the current number of assignments to variable v; initially 0

Auxiliary Routine

procedure GenName(variable v)
  i := Counters[v]
  push i onto Stacks[v]
  Counters[v] := i + 1

Use the Dominance Tree to remember the most recent definition of each variable

Variable Renaming Algorithm

procedure Rename(block b)
  if b previously visited return
  for each φ-function p in b
    GenName(LHS(p)) and replace v with v_i, where i = Top(Stack[v])
  for each statement s in b (in order)
    for each variable v ∈ RHS(s)
      replace v by v_i, where i = Top(Stacks[v])
    for each variable v ∈ LHS(s)
      GenName(v) and replace v with v_i, where i = Top(Stack[v])
  for each s ∈ succ(b) (in CFG)
    j ← position in s’s φ-function corresponding to block b
    for each φ-function p in s
      replace the j-th operand of RHS(p) by v_i, where i = Top(Stack[v])
    for each s ∈ child(b) (in DT)
      Rename(s)
  for each t ∈ child(b) (in DT)
    if t is a φ-function or statement t in b
      for each variable v_i ∈ LHS(t)
        Pop(Stack[v])
      Unwind stack when done with this node

compilation

Transformation from SSA Form

Proposal
- Restore original variable names (i.e., drop subscripts)
- Delete all φ-functions

Complications
- What if versions get out of order?
  (simultaneously live ranges)

Alternative
- Perform dead code elimination (to prune φ-functions)
- Replace φ-functions with copies in predecessors
- Rely on register allocation coalescing to remove unnecessary copies