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 $n$

Notational Convenience
- $\text{DF}(S) = \bigcup_{s \in S} \text{DF}(s)$

Dom(5) = \{5, 6, 7, 8\}
DF(5) = \{4, 5, 13\}

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?

Dominance Frontier Example

DF(d) = \{n | \exists p \in \text{pred}(n), d \text{ dom } p \text{ and } d \neq \text{sdom } n\}

Dom(5) = \{5, 6, 7, 8\}
DF(5) = \{4, 5, 12, 13\}

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

SSA Exercise

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

v := \ldots
v := \ldots
v := \phi(v_3, v_4)

v := \phi(v_3, v_2)

v := \phi(v_3, v_4)

v := \phi(v_3, v_2)
**Dominance Frontiers Revisited**

Suppose that node 3 defines variable x

$$\text{DF}(3) = \{5\}$$

Do we need to insert a $\phi$-function for x anywhere else? Yes. At node 6. Why?

**Dominance Frontiers and SSA**

Let
- $$\text{DF}_x(S) = \text{DF}(S)$$
- $$\text{DF}_{x+1}(S) = \text{DF}(S \cup \text{DF}_x(S))$$

**Iterated Dominance Frontier**
- $$\text{DF}_x(S)$$

**Theorem**
- If S is the set of CFG nodes that define variable v, then $$\text{DF}_x(S)$$ 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
- Put all defs of v on the worklist
- WorkList $\leftarrow$ WorkList $\cup$ \{n\}
- EverOnWorkList $\leftarrow$ WorkList
- while WorkList $\neq$ $\emptyset$
- Remove some node n from WorkList
- for each d $\in$ DF(n)
- if d $\notin$ AlreadyHasPhiFunc
- Insert a $\phi$-function for v at d
- Insert at most one $\phi$-function per node
- AlreadyHasPhiFunc $\leftarrow$ AlreadyHasPhiFunc $\cup$ \{d\}
- if d $\notin$ EverOnWorkList
- WorkList $\leftarrow$ WorkList $\cup$ \{d\}
- EverOnWorkList $\leftarrow$ EverOnWorkList $\cup$ \{d\}
- 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**

- $x = x$
- $x = x$

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

- $x = x$
- $x_n = x_{n-1}$
- $x_{n-1} = x_{n-2}$

**Traverse the dominance tree**
The dominance tree shows the dominance relation

---

### Variable Renaming Algorithm

```plaintext
procedure Rename(block b)
  if b previously visited return  // Call Rename(entry-node)
  for each 
    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 + Top(Stack[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 \( \Phi \)-function corresponding to block b
      for each \( \Phi \)-function p in s
        replace the \( j \)th operand of RHS(p) by v_{j+1}, where i = Top(Stack[v])
        Rename(s)
      for each \( \Phi \)-function or statement t in b
        Recurse using Depth First Search
      Unwind stack when done with this node
  for each \( \Phi \)-function or statement t in b
    Pop(Stack[v])
```

---

### Variable Renaming (cont)

**Data Structures**
- \( \text{Stacks}[v] \forall v \)
  - Holds the subscript of most recent definition of variable \( v \), initially empty
- \( \text{Counters}[v] \forall v \)
  - Holds the current number of assignments to variable \( v \); initially 0

**Auxiliary Routine**

```plaintext
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

---

### Transformation from SSA Form

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

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

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

---
**Backward Analyses vs. Forward Analyses**

For forward data-flow analysis, at phi node apply meet function.

For backward data-flow analysis?

\[ v_2 := \phi(v_0, v_1) \]

\[ \ldots v_2 \ldots \]

---

**Static Single Information Form (SSI)**

Ananian’s Masters Thesis, 1997 MIT

**Concepts**

SSA construction
- Place phi nodes
- Variable renaming

Transformation from SSA to executable code depends on the optimizations dead-code elimination and copy propagation

Backward data-flow analyses can use SSI modification to SSA

---

**Next Time**

Assignments
- HW1 due

Lecture
- Using SSA for program optimization