The \( \Phi \) function

- \( \Phi \) merges multiple definitions along multiple control paths into a single definition.
- At a BB with \( p \) predecessors, there are \( p \) arguments to the \( \Phi \) function.
  \[ x_{\text{new}} \leftarrow \Phi(x_1, x_1, x_1, \ldots, x_p) \]
- How do we choose which \( x_i \) to use?
  - Most compiler writers don’t really care!
  - If we care, use moves on each incoming edge
    (Or, as in pegasus use a mux)

"Implementing" \( \Phi \)

```
\[
c_1 \leftarrow 12 \\
\text{if (i)} \\
\]

```

```
\[
a_1 \leftarrow x + y \\
b_1 \leftarrow a_1 + x \\
a_3 \leftarrow a_1 \\
c_3 \leftarrow c_1 \\
a_3 \leftarrow \Phi(a_1, a_2) \\
c_3 \leftarrow \Phi(c_1, c_2) \\
a_4 \leftarrow c_3 + a_3 \\
\]
```

Trivial SSA

- Each assignment generates a fresh variable.
- At each join point insert \( \Phi \) functions for all live variables.

```
\[
x \leftarrow 1 \\
y \leftarrow 1 \\
\]
```

```
\[
x_1 \leftarrow x \\
y_1 \leftarrow x \\
x_2 \leftarrow \Phi(x_1, x_1) \\
y_3 \leftarrow \Phi(y_1, y_2) \\
z_1 \leftarrow y_3 + x_2 \\
\]
```

Way too many \( \Phi \) functions inserted.
Minimal SSA

- Each assignment generates a fresh variable.
- At each join point insert Φ functions for all variables with multiple outstanding defs.

```
x ← 1
y ← x
y ← 2
z ← y + x

x1 ← 1
y1 ← x2
y2 ← 2
y3 ← Φ(y1, y2)
z1 ← y3 + x1
```

Another Example

```
a ← 0
b ← a + 1
c ← c + b
a ← b * 2
a < N

return c
```

Another Example

```
a ← 0

Notice use of c1
b ← a + 1
c ← c + b
a ← b * 2
a < N

return c
```

When do we insert Φ?

```
a ← 0

a3 ← Φ(a1, a2)
c3 ← Φ(c1, c2)
b2 ← a3 + 1
c2 ← c3 + b2
a2 ← b2 * 2
a2 < N

 CFG
```

If there is a def of a in block 5, which nodes need a Φ()?
When do we insert $\Phi$?

- Insert a $\Phi$ function for variable $A$ in block $Z$ iff:
  - $A$ was defined more than once before (i.e., $A$ defined in $X$ and $Y$ AND $X \neq Y$)
  - $Z$ is the first block that joins the paths from $X$ to $Z$ and $Y$ to $Z$

- Entry block implicitly defines all vars
- Note: $A = \Phi(...)$ is a def of $A$
When do we insert $\Phi$?

Def-use property of SSA
- If $x_i$ is used in $x \leftarrow \Phi(..., x_i, ...)$, then
  NO BBs in any path from $BB(x_i)$ to $BB(\Phi)$
  include def of $x$ except $BB(X_i)$ and $BB(\Phi)$
- If $x$ is used in $y \leftarrow ... x ...$, then no BBs in path
  from $BB(x)$ to $BB(y)$ define $x$ except $BB(x)$

Another way to say this:
Definitions dominate uses

Dominance Property of SSA
- In SSA definitions dominate uses.
  - If $x_i$ is used in $x \leftarrow \Phi(..., x_i, ...)$, then
    $BB(x_i)$ dominates ith pred of $BB(\Phi)$
  - If $x$ is used in $y \leftarrow ... x ...$, then
    $BB(x)$ dominates $BB(y)$
- Use this for an efficient alg to convert to SSA
A little side trip

- Computing dominators

- $d \text{ dom } n$ iff every path from $s$ to $n$ goes through $d$
- $n \text{ dom } n$ for all $n$
- Some definitions:
  - immediate dominator: $d \text{ idom } n$ iff
    - $d \neq n$
    - $d \text{ dom } n$
    - $d$ doesn't dominate any other dominator of $n$
  - strictly dominates: $s \text{ sdom } n$ iff
    - $s \text{ dom } n$
    - $s \neq n$

Examples

- $d \text{ dom } n$ iff every path from Entry to $n$ contains $d$.
  - $1 \text{ dom } 1$ ; $1 \text{ dom } 2$ ; $1 \text{ dom } 3$ ; $1 \text{ dom } 4$ ;
  - $2 \text{ dom } 2$ ; $2 \text{ dom } 3$ ; $2 \text{ dom } 4$ ; $3 \text{ dom } 3$ ;
  - $4 \text{ dom } 4$

- $s \text{ strictly dominates } n$, $(s \text{ sdom } n)$, iff $s \text{ dom } n$ and $s \neq n$.
  - $1 \text{ sdom } 2$ ; $1 \text{ sdom } 3$ ; $1 \text{ sdom } 4$ ;
  - $2 \text{ sdom } 3$ ; $2 \text{ sdom } 4$

- $d \text{ immediately dominates } n$, $d=\text{idom}(n)$, iff $d \text{ sdom } n$ and there is no node $x$ such that $d \text{ dom } x$ and $x \text{ dom } n$.
  - $1 \text{ idom } 2$ ; $2 \text{ idom } 3$ ; $2 \text{ idom } 4$

Properties of dominators

- $\text{idom}(n)$ is unique
- The dominance relation is a partial ordering; that is, it is reflexive, anti-symmetric and transitive:
  - reflexive:
    - $x \text{ dom } x$
  - anti-symmetric:
    - $x \text{ dom } y$ and $y \text{ dom } x \rightarrow x = y$
  - transitive:
    - $x \text{ dom } y$ and $y \text{ dom } z \rightarrow x \text{ dom } z$

The dominator tree

- One can represent dominators in a CFG as a tree of immediate dominators.
- In dominator tree, edge from parent to child if parent idom child in the CFG
- The set of dominators of a node are the nodes from the root to the node.

(adapted from: http://www.eecg.toronto.edu/~voss/ece540/)
Computing Dominators

- $d \text{ dom } n$ iff every path from $s$ to $n$ goes through $d$
- Note: $n \text{ dom } n$ for all $n$

- If $s \text{ dom } d \land d \neq n \land p_i \in \text{pred}(n) \land d \text{ dom } p_i,$ then $d \text{ dom } n$

- How can we use this?

Simple iterative alg

- $\text{dom(Entry)} = \text{Entry}$
- for all other nodes, $n,$ $\text{dom}(n) = \text{all nodes}$ changed = true
- while (changed) {
  changed = false
  for each $n, n \neq \text{Entry}$ {
    old = $\text{dom}(n)$
    $\text{dom}(n) = \{n\} \bigcup \bigcap_{p \in \text{pred}(n)} \text{dom}(p)$
    if (dom($n$) != old) changed = true
  }
}

(adapted from: http://www.eecg.toronto.edu/~voss/ece540/)

Computing Dominators

- $d \text{ dom } n$ iff every path from $s$ to $n$ goes through $d$
- Note: $n \text{ dom } n$ for all $n$

- If $s \text{ dom } d \land d \neq n \land p_i \in \text{pred}(n) \land d \text{ dom } p_i,$ then $d \text{ dom } n$
- $\text{dom}(n) = \{n\} \bigcup \bigcap_{p \in \text{pred}(n)} \text{dom}(p)$

Example

- $\text{DOM (Entry)} = \{\text{Entry}\}$
- $\text{DOM (1)} = \{\text{Entry,1}\}$
- $\text{DOM (2)} = \{\text{Entry,1,2}\}$
- $\text{DOM (3)} = \{\text{Entry,1,2,3}\}$
- $\text{DOM (4)} = \{\text{Entry,1,2,3,4}\}$
- $\text{DOM (5)} = \{\text{Entry,1,2,3,4,5}\}$
- $\text{DOM (6)} = \{\text{Entry,1,2,3,4,5,6}\}$
- $\text{DOM (7)} = \{\text{Entry,1,2,3,4,5,7}\}$
- $\text{DOM (8)} = \{\text{Entry,1,2,3,4,5,8}\}$
- $\text{DOM (9)} = \{\text{Entry,1,2,3,4,9}\}$
- $\text{DOM (10)} = \{\text{Entry,1,2,10}\}$

(Borrowed from: http://www.eecg.toronto.edu/~voss/ece540/)
Finding immediate dominators

- idom(n) dominates n, isn't n, and, doesn't strictly dominate any other sdom n
- Init idom(n) to nodes which sdom n
- foreach x ∈ idom(n)
  - foreach y ∈ idom(n) - {x}
    - if (y ∈ sdom(x)) idom(n)=idom(n)-{y}

Example (immediate dominators)

DOMd (1) = {Entry}
DOMd (2) = {Entry, 1}
DOMd (3) = {Entry, 1, 2}
DOMd (4) = {Entry, 1, 2, 3}
DOMd (5) = {Entry, 1, 2, 3, 4}
DOMd (6) = {Entry, 1, 2, 3, 4, 5}
DOMd (7) = {Entry, 1, 2, 3, 4, 5}
DOMd (8) = {Entry, 1, 2, 3, 4, 5}
DOMd (9) = {Entry, 1, 2, 3, 4}
DOMd (10) = {Entry, 1, 2}

Dominance Property of SSA

- In SSA definitions dominate uses.
  - If x_i is used in x ← Φ(..., x_i, ...), then BB(x_i) dominates ith pred of BB(PHI)
  - If x is used in y ← ... x ..., then BB(x) dominates BB(y)
- Use this for an efficient alg to convert to SSA

Dominance

x strictly dominates w (s dom w) iff x dom w AND x ≠ w

If there is a def of a in block 5, which nodes need a Φ()?
Dominance Frontier

The dominance Frontier of a node $x = \{ w \mid x \text{ dom pred}(w) \text{ AND } !(x \text{ sdom } w)\}$

$\text{x strictly dominates } w (s \text{ sdom } w) \text{ iff } x \text{ dom } w \text{ AND } x \neq w$

Computing $\text{DF}(n)$

c is an example of the successors of n not strictly dominated by n

n idom a
n idom b
!n idom c

x is in $\text{DF}[a]$, but !(idom(x) dom x)

n idom a
n idom b
!n idom c
Computing the Dominance Frontier

The dominance Frontier of a node \( x = \{ w \mid x \text{ dom pred}(w) \text{ AND } !(x \text{ sdom } w) \} \)

\[
\text{compute-DF}(n) \\
S = \{} \\
\text{foreach node } y \text{ in succ}[n] \\
\quad \text{if idom}(y) \neq n \\
\quad S = S \cup \{ y \} \\
\text{foreach child of } n, c, \text{ in D-tree} \\
\quad \text{compute-DF}(c) \\
\quad \text{foreach } w \text{ in DF}[c] \\
\quad \quad \text{if } !n \text{ dom } w \\
\quad \quad S = S \cup \{ w \} \\
\text{DF}[n] = S
\]

Using DF to compute SSA

- place all \( \Phi() \)
- Rename all variables

Using DF to Place \( \Phi() \)

- Gather all the defsites of every variable
- Then, for every variable
  - foreach defsite
    - foreach node in DF(defsite)
      - if we haven't put \( \Phi() \) in node put one in
        - If this node didn't define the variable before: add this node to the defsites

- This essentially computes the Iterated Dominance Frontier on the fly, inserting the minimal number of \( \Phi() \) neccesary
Renaming Variables

• Walk the D-tree, renaming variables as you go
• Replace uses with more recent renamed def
  - For straight-line code this is easy
  - If there are branches and joins?

Renaming Variables

• Walk the D-tree, renaming variables as you go
• Replace uses with most recent renamed def
  - For straight-line code this is easy
  - If there are branches and joins use the closest def such that the def is above the use in the D-tree
• Easy implementation:
  - for each var: rename (v)
  - rename(v): replace uses with top of stack
    at def: push onto stack
    call rename(v) on all children in D-tree
    for each def in this block pop from stack

```
rename
foreach var a
  a.count = 0
  a.stack = empty
  a.stack.push(0)
rename(entry)
renamer
  foreach s in block n
    if s isn't Φ
      foreach use of x in S
        replace x with x.stack.top()
      foreach def of x in S
        i = ++x.count
        x.stack.push(i)
        replace x with x
  foreach y ∈ succ(n)
    j = pred # of n in y
    foreach Φ in y
      i <- var-j.stack.top()
      replace var-j with var-j.i
      foreach child X of n in D-tree: rename(X)
      foreach def, x, in S: x.stack.pop()
```
Compute D-tree

1. \( i \leftarrow 1 \)
2. \( j \leftarrow 1 \)
3. \( k \leftarrow 0 \)
4. \( k < 100? \)
5. \( j < 20? \)
6. \( j \leftarrow i \)
7. \( k \leftarrow k + 1 \)
8. \( j \leftarrow k \)
9. \( k \leftarrow k + 2 \)
10. return \( j \)
11. \( j \leftarrow i \)
12. \( k \leftarrow k + 1 \)
13. \( j \leftarrow k \)
14. \( k \leftarrow k + 2 \)

Compute Dominance Frontier

1. \( i \leftarrow 1 \)
2. \( j \leftarrow 1 \)
3. \( k \leftarrow 0 \)
4. \( k < 100? \)
5. \( j < 20? \)
6. \( j \leftarrow i \)
7. \( k \leftarrow k + 1 \)
8. \( j \leftarrow k \)
9. \( k \leftarrow k + 2 \)
10. return \( j \)
11. \( j \leftarrow i \)
12. \( k \leftarrow k + 1 \)
13. \( j \leftarrow k \)
14. \( k \leftarrow k + 2 \)
15. DFs

Insert \( \Phi() \)

1. \( \{ \} \)
2. \( \{ i, j, k \} \)
3. \( \{2\} \)
4. \( \{\} \)
5. \( \{7\} \)
6. \( \{j,k\} \)
7. \( \{\} \)

DFs

1. \( \text{orig}[n] \)
2. \( \{\} \)
3. \( \{2\} \)
4. \( \{\} \)
5. \( \{7\} \)
6. \( \{j,k\} \)
7. \( \{\} \)

var i: \( W=\{1\} \)

var j: \( W=\{1,5,6\} \)

DF{1}, DF{5}
\textbf{Insert } \Phi() \textbf{ } \\

1 \{ \} \quad 1 \{ i,j,k \} \\
2 \{2\} \quad 2 \{ \} \\
3 \{2\} \quad 3 \{ \} \\
\text{defsites}[v] \\
4 \{\} \quad 4 \{ i \} \\
5 \{7\} \quad 5 \{ j,k \} \\
6 \{7\} \quad 6 \{ j,k \} \\
7 \{2\} \quad 7 \{ \} \\

\text{j} \leftarrow \Phi(j,j) \\
\text{k} < 100? \\
\text{j} \leftarrow \Phi(j,j) \\
\text{DFs} \\
\text{k} \leftarrow \Phi(k,k) \\
k < 100? \\
j \leftarrow \Phi(j,j) \\
\text{DFs} \\
\text{var } j: \text{ W} =\{1,5,6\} \\
\text{DF(1), DF(5), DF(6)}
Rename Vars

\[
\begin{array}{c}
i \leftarrow 1 \\
j \leftarrow 1 \\
k \leftarrow 0 \\
j \leftarrow \Phi(j,j) \\
k \leftarrow \Phi(k,k) \\
k < 100? \\
j < 20? \\
\text{return } j \\
j \leftarrow i \\
k \leftarrow k + 1 \\
k \leftarrow k + 2 \\
j \leftarrow \Phi(j,j) \\
k \leftarrow \Phi(k,k)
\end{array}
\]

Rename Vars

\[
\begin{array}{c}
i_1 \leftarrow 1 \\
j_1 \leftarrow 1 \\
k_1 \leftarrow 0 \\
j \leftarrow \Phi(j,j) \\
k \leftarrow \Phi(k,k) \\
k < 100? \\
j < 20? \\
\text{return } j \\
j \leftarrow i \\
k \leftarrow k + 1 \\
k \leftarrow k + 2 \\
j \leftarrow \Phi(j,j) \\
k \leftarrow \Phi(k,k)
\end{array}
\]
**Rename Vars**

1. \( i_1 \leftarrow 1 \)
2. \( j_1 \leftarrow 1 \)
3. \( k_1 \leftarrow 0 \)
4. \( j_2 \leftarrow \Phi(j_1,j) \)
5. \( k_2 \leftarrow \Phi(k_1,k) \)
6. \( k_2 < 100? \) - return
7. \( j < 20? \) - return
8. \( j_3 \leftarrow i_2 \)
9. \( k_3 \leftarrow k_2 + 1 \)
10. \( j_4 \leftarrow k_2 \)
11. \( k_4 \leftarrow k_2 + 2 \)
12. \( j_5 \leftarrow \Phi(j_3,j_4) \)
13. \( k_5 \leftarrow \Phi(k_3,k_4) \)

** SSA Properties**

- Only 1 assignment per variable
- definitions dominate uses

**Constant Propagation**

- If \( "v \leftarrow c" \), replace all uses of \( v \) with \( c \)
- If \( "v \leftarrow \Phi(c,c,c)" \) replace all uses of \( v \) with \( c \)

W <- list of all defs
while !W.isEmpty {
    Stmt S <- W.removeOne
    if S has form \( "v \leftarrow \Phi(c,...,c)" \)
        replace S with \( V \leftarrow c \)
    if S has form \( "v \leftarrow c" \) then
        delete S
    foreach stmt U that uses \( v \),
        replace \( v \) with \( c \) in \( U \)
    W.add(U)
}
Other stuff we can do?

- Copy propagation
  delete “x ← Φ(y)” and replace all x with y
  delete “x ← y” and replace all x with y
- Constant Folding
  (Also, constant conditions too!)
- Unreachable Code
  Remember to delete all edges from unreachable block