\[ DF(x) = DF_{\text{local}}(x) \cup \bigcup \{ DF_{\text{up}}(z) \mid z \text{ a child of } x \} \]
\[ \{ y \mid y \in \text{succ}(x) \land \text{idom}(y) \neq x \} \]

<table>
<thead>
<tr>
<th>node</th>
<th>( DF_{\text{local}} )</th>
<th>( DF_{\text{up}} )</th>
<th>( DF(x) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>{63}</td>
<td>{3}</td>
<td>{63}</td>
</tr>
<tr>
<td>6</td>
<td>{33}</td>
<td>{33}</td>
<td>{33}</td>
</tr>
<tr>
<td>4</td>
<td>{3}</td>
<td>{3}</td>
<td>{3}</td>
</tr>
<tr>
<td>7</td>
<td>{83}</td>
<td>{83}</td>
<td>{83}</td>
</tr>
<tr>
<td>3</td>
<td>{3}</td>
<td>{3}</td>
<td>{3}</td>
</tr>
<tr>
<td>2</td>
<td>{83}</td>
<td>{83}</td>
<td>{83}</td>
</tr>
<tr>
<td>8</td>
<td>{83}</td>
<td>{83}</td>
<td>{83}</td>
</tr>
<tr>
<td>1</td>
<td>{3}</td>
<td>{33}</td>
<td>{33}</td>
</tr>
<tr>
<td>9</td>
<td>{2x3}</td>
<td>{2x3}</td>
<td>{2x3}</td>
</tr>
<tr>
<td>0</td>
<td>{13}</td>
<td>{13}</td>
<td>{13}</td>
</tr>
</tbody>
</table>

\(2\)
3.5 φ-node insertion

(part 1, followed by renaming of var)

(part 2)

given: CFG, DF for all nodes

for each variable V: list of blocks (nodes in FG) that define V

ASSIGN(V)

idea: insert φ nodes for ASSIGN, add more φ nodes (possibly) based on φ-node insertion

has-φ-node(X) - for a block X in FG true

if X contains φ node for V

worklist W - set of nodes that must be processed

added-to-worklist(X) - for block X true if X has already been added to W (for V)
Remark: An efficient implementation of the predicate can be based on integers
- no need to use bit vectors with N entries (N = # blocks in CFG)

Use counter to keep track of insertion of blocks when processing

```c
int count = 0;
for all node x \in CFG \\
    has_phi_node(x) = count;
    atw(x) = count;
```

worklist w = {3}
for all variables $V$:

```cpp
count++; 

w = ASSIGN(V); // init worklist

for all $x \in ASSIGN(V)$:
    atw(x) = count;

while ($w \neq 13$) do:
    pick $B \in w$, remove $B$ from $w$
    for all nodes $Y \in DF(B)$:
        if (has-$\phi$-node ($Y$) < count) then
            insert $\phi$ node into $Y$;
            has-$\phi$-node ($Y$) = count;
            if (atw($Y$) < count) then
                add $Y$ to worklist;
                atw($Y$) = count;
        else if no $\phi$ node
        // for all blocks $Y$ in DF(B)
```

⑤ ③ ②
\( \text{assign}(a) = \{0, 4\} \)
\( \text{assign}(b) = \{0, 2, 6\} \)

\( \text{count} = 0 \)
\( \text{has 1-node}(\_\_\_) = 0 \)
\( \text{atw}(\_\_\_) = 0 \)

\( \text{for} \quad v = a \)
\( \quad \text{count} = 1 \)
\( \quad \text{w} = \{0, 4\} \)
\( \quad \text{atw}(0) = 1; \)
\( \quad \text{atw}(4) = 1; \)

\( \bullet \) pick a block \( B \) from \( w \)
\( \quad \boxed{B = 0} \quad \text{w} = \{0, 4\} \)
\( \quad \text{DF}(0) = \{x\} \)
\( \quad \text{insert } \phi \text{ node in } x \)
\( \quad \text{has } \phi \text{-node}(x) = 1 \)
\( \quad \text{atw}(x) = 1 \)
\( \quad \text{w} = \{4, x\} \)

\( \bullet \) pick a block \( B \)
\( \quad \boxed{B = 4} \)
\( \text{DF}(4) = \{?\} \)
\( \text{insert } \phi \text{ node} \)
\( \text{has } \phi \text{-node}(3) = 1 \)
\( \text{atw}(3) = 1 \)
\( \text{w} = \{x, 3\} \)

\( \bullet \) pick a node
\( \quad \boxed{B = x} \)

\( \bullet \) pick a node
\( \quad \boxed{B = 3} \quad \text{w} = 23 \)
\( \quad \text{DF}(3) = \{8\} \)
\( \text{insert } \phi \text{ node} \)
\( \quad \text{w} = \{8\} \)
\[ B = \emptyset \quad w = \lambda \]
\[
\text{DF}(\emptyset) = \{ 9 \}
\text{insert a } \emptyset \text{ node}
\]
\[
w = \{ 9 \}
\]
\[ B = 9 \quad w = \{ 3 \}
\]
\[
\text{DF}(9) = \{ x^3 \}
\quad x \text{ has a } \emptyset \text{ node}
\]
\[ v = b \]
3.6 Renaming the variables

(part 2): given a CFG with φ nodes,

determine correct version of variables

on the RHS, the LHS, in φ-node.

Consider this setup:

dom by B

dom to C

\[
\begin{aligned}
A & : a \\
B & : a \\
C & : a \\
D & : a \\
E & : a \\
\end{aligned}
\]

\[
\begin{aligned}
A & : B \\
C & : \ldots \\
F & : D \\
\end{aligned}
\]
So, inside region dominated by B:

use version of "a" defined in B

inside ... ... by C:

use version of "c" defined in C

If you process C(E,D) before processing A:

use stale/version set in B

- restore versions to those set in B

on a stack
stack of versions (integer) [array of stacks, one for each variable]

counter = 0
stack = \bot \ (empty) [for each v]

process (X) {

for all stmts S in X do \\
  if S is a regular assignment stmt \\
    for each V used in the RHS do \\
      replace V by V_i (= Top.Stack[P(V)])

  for each V used in the LHS do \\
    if V = ....
      i = counter;
      push i on the stack[P(V)]
      replace V by V_i;
      counter ++;

}
for all successors \( S \) of \( \ast \) (in the CGT) do

// check if there is a \( \phi \) node
// if there is a \( \phi \) node for variable \( V \)
let \( X \) be the \( K \)-th predecessor of \( S \).
// replace \( V \) in \( K \)-th position at \( \phi \) node
\[ V = \phi(V, \ldots, V_i, \ldots, V) \]

// push \( K \)-th

for all successors \( S \)
for all children \( C \) of \( \ast \) (in CGT) do

\[ \text{push}(C) \]

for all assignments \( A \) in \( \lambda \) do

\[ \text{if } V \text{ is used on the LHS of } A \]
\[ \text{pop}(\text{Stack}[V]) \] /* remove version for stack */
\( \text{proof (0)} \)

\[
\text{stack}[a] = (0, 1) \\
\text{stack}[b] = (0, 1)
\]

\( \text{proof (1)} \)

1. First successor of 1 is node 3

1 is 1st predecessor of 3

\( a_1 = \phi(a_0, a_2) \)

\( \text{proof (3)} \)

\( \text{proof (4)} \)

\( a_1 = \ldots \)

\[
\text{stack}[a] = (1, 0, 1)
\]

\( \text{proof (5)} \)

\( \text{proof (6)} \)

\[
\text{stack}[b] = (1, 0, 1)
\]
6 is 2nd predecessor of 3

\[ a = \phi(a_0, a_1) \]

\[ a_2 = \phi(a_0, a_1) \]

skew \( (a) = (2, 9 \uparrow) \)
Comments

. Easy to do this for all variables in
  parent
  - visit each node of CFB only once

. Easy to deal with expressions (e.g.
  if branch/ if stmt): there is no
  visit node starting until root of OT
  (START)