Computing SSA

*Compute Dominance Frontiers*
Potential join points, based only on CFG

*Place  φ (Phi) Functions*
Join points based on actual assignments in program
Want minimal number

*Rename variables*
Each use has a unique definition point

Get minimal SSA!

---

**Dominance Frontiers**

\[ \triangleright \text{Strictly dominates} \quad \triangleright \text{Dominates} \]

\[ DF(X) = \{ Y \mid \exists Z \text{ a pred of } Y (X \triangleright Z \text{ and } X \not\triangleright Y) \} \]

\( X \) dominates a predecessor of \( Y \) but does not strictly dominate \( Y \)

\[ DF(S) = \bigsqcup \{ DF(s) \mid s \in S \} \]

![Diagram showing dominance frontiers and definitions](image)
Other Examples

Ladder graph example

<table>
<thead>
<tr>
<th>Node X</th>
<th>DF(X)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>(G,H,I,J)</td>
</tr>
<tr>
<td>B</td>
<td>(H,I,J)</td>
</tr>
<tr>
<td>C</td>
<td>(I,J)</td>
</tr>
<tr>
<td>D</td>
<td>(J)</td>
</tr>
<tr>
<td>F</td>
<td>(G)</td>
</tr>
<tr>
<td>G</td>
<td>(H)</td>
</tr>
</tbody>
</table>
Computing Dominance Frontiers

\[ DF(X) = DF_{\text{local}}(X) \cup \bigcup_{Z \text{ a child of } X} DF_{\text{up}}(Z) \]

\[ DF_{\text{local}}(X) = \{ Y \text{ a succ. of } X | \text{idom}(Y) \neq X \} \]

\[ DF_{\text{up}}(Z) = \{ Y \in DF(Z) | \text{idom}(Y) \neq X \} \]

Complexity: Linear in size of \( \bigcup \) of DF sets

Linear in size of CFG for well-structured programs

Placing Phi Functions
Placing Phi Functions

\[
\begin{align*}
V &:= \\
W &:= \\
V &:= \\
W &:= \\
V &:= \phi(V, V) \\
W &:= \phi(W, W)
\end{align*}
\]
Placing Phi Functions

DF^1(S) = DF(S)

DF^+ (S) = lim inf_{i} DF^i (S)

Let S = \{nodes with asst's to X\} \cup \{Entry\}

DF^+(S) is the set of nodes that require phi functions for variable X

Worklist algorithm, pass for each variable

- Initialize worklist with set of all assignments to variable
- For each Y on worklist, place phi function for each Z in DF(Y)
  if not already there, and place it on worklist

Complexity: (Total no. Assts. \times wt.av.(DF))
Renaming Variables

<table>
<thead>
<tr>
<th>V :=</th>
<th>W :=</th>
</tr>
</thead>
<tbody>
<tr>
<td>V :=</td>
<td>W :=</td>
</tr>
<tr>
<td>\phi(V, V)</td>
<td>\phi(W, W)</td>
</tr>
<tr>
<td>W :=</td>
<td>W :=</td>
</tr>
</tbody>
</table>

Renaming Variables

<table>
<thead>
<tr>
<th>V_0 :=</th>
<th>W_0 :=</th>
</tr>
</thead>
<tbody>
<tr>
<td>V_0 :=</td>
<td>W_0 :=</td>
</tr>
<tr>
<td>V :=</td>
<td>W :=</td>
</tr>
<tr>
<td>\phi(V, V)</td>
<td>\phi(W, W)</td>
</tr>
<tr>
<td>W :=</td>
<td>W :=</td>
</tr>
<tr>
<td>W :=</td>
<td>W :=</td>
</tr>
<tr>
<td>\phi(W, W)</td>
<td>\phi(W, W)</td>
</tr>
</tbody>
</table>

V_0 := \phi(V_0, V_1)
W_0 := \phi(W_0, W_1)
Renaming Variables
Keep stack of indices for each variable
Traversal of Dominator tree, starting from Entry
Visit Node:
    RHS Asst in Node:
        Rename with index from variable’s TOS
    LHS Asst in Node:
        Create new index, rename, and push
φ function in jth successor of Node:
    Rename jth variable in φ with TOS index
Visit all children of Node in Dom tree
Pop stack for each LHS asst in Node

SSA Construction Example
Entry
a := b := c := 1
L1: if () go to L3
    a := a + 1
    b := b + 1
goto L4
L3: a := a + 2
    b := b + 2
L4: c := c + (a * b)
    if () then goto L1
    print(c)
Exit