Def-Use and Use-Def Chains

Application of ReachingDefs

Auxiliary data structures like CFG

Ex. DefUse Chains

\[ X := \]
\[ \text{if (.) then } X := \quad \text{else } X := \]
\[ \Rightarrow X \]

Optimizations may operate on Def-Use chains

Can bypass CFG entirely

Static Single Assignment Form (SSA)

Each (static) assignment to a variable renamed

All of the uses reached by the assignment renamed

Results in a single def that reaches every use

Ex.

\[ X := \]
\[ \Rightarrow X \]
\[ X := \]
\[ \Rightarrow X \]
\[ \text{if (.) then } X := \quad \text{else } X := \]
\[ \Rightarrow X \]

SSA with Def-Use chains

Def-Use chains

More compact Def-Use representation
Advantages of SSA

More compact Def-Use representation

Ex. CFG is $O(N)$

More powerful optimizations
- Value Numbering
- Program Equivalence
- Constant Propagation

Faster optimizations
- Constant Propagation
- Code Motion

Increased Parallelism

Used in real compilers!
- IBM Jikes, Sun, Compaq Swift Java, Tera MTA, Scale,...
SSA Loop Example

\[(1) \quad \text{Read}(N) \]
\[2) \quad i := 1 \]
\[3) \quad \text{if } i > N \text{ goto } L3 \]
\[4) \quad a(i) := a(i) + 1 \]
\[5) \quad i := i + 1 \]
\[6) \quad \text{goto } L2 \]
\[7) \quad \text{Print}(A(0)) \]
\[8) \quad \text{L3: Print}(A(0)) \]

\( \Phi \) makes merge of values explicit

Qu: How construct SSA form?

Computing SSA

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

**Place \( \Phi \) (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**

- **Strictly dominates** $\gg$ **Dominates**
- $DF(X) = \{ Y | \exists Z \text{ a pred of } Y (X \gg Z \text{ and } X \not\gg Y)\}$
- $X$ dominates a predecessor of $Y$ but does not strictly dominate $Y$

$$DF(S) = \bigcup_{s \in S} DF(s)$$

**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>E</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>{G}</td>
</tr>
<tr>
<td>G</td>
<td>{H}</td>
</tr>
</tbody>
</table>

### Computing Dominance Frontiers

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

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

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

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

*Linear in size of CFG for well-structured programs*
Placing Phi Functions

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

\[ \text{DF}^1(S) = \text{DF}(S) \]
\[ \text{DF}^{i+1}(S) = \text{DF}(S \cup \text{DF}^i(S)) \]
\[ \text{DF}^+(S) = \lim_{i \to \infty} \text{DF}^i(S) \]

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

\( \text{DF}^+(S) \) 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 \( \text{DF}(Y) \)
  - if not already there, and place it on worklist

**Complexity:** \( (\text{Total no. Assts. \times wt.av.}(\text{DF})) \)

Renaming Variables

\[ V := \phi(V, V) \]
\[ W := \phi(W, W) \]
Renaming Variables

Keep stack of indices for each variable

Traversal of Dominator tree, starting from Entry
Visit Node:
  \textbf{RHS Asst in Node:}
  Rename with index from variable’s TOS
  \textbf{LHS Asst in Node:}
  Create new index, rename, and push
  \textit{function in \textit{j}th successor of Node:}
  Rename \textit{j}th variable in \textit{\phi} with TOS index

Visit all children of Node in Dom tree
Pop stack for each LHS asst in Node