Computing Static Single Assignment (SSA) Form

Overview

- What is SSA?
- Advantages of SSA over use-def chains
- “Flavors” of SSA
- Dominance frontiers revisited
- Inserting $\phi$-nodes
- Renaming the variables
- Translating out of SSA form

What is SSA?

- Each assignment to a variable is given a unique name
- All of the uses reached by that assignment are renamed
- Easy for straight-line code

\[
\begin{align*}
V & \leftarrow 4 & V_0 & \leftarrow 4 \\
& \leftarrow V + 5 & & \leftarrow V_0 + 5 \\
V & \leftarrow 6 & V_1 & \leftarrow 6 \\
& \leftarrow V + 7 & & \leftarrow V_1 + 7
\end{align*}
\]

What about control flow?

\[\implies \phi\text{-nodes}\]
What is SSA?

CS 380C Lecture 4 3 Static Single Assignment
What is SSA?

\[ B_1 \quad I \leftarrow 1 \]

\[ B_2 \quad I \leftarrow I + 1 \]

\[ B_1 \quad I_0 \leftarrow 1 \]

\[ B_2 \quad I_1 \leftarrow \phi(I_2, I_0) \]

\[ I_2 \leftarrow I_1 + 1 \]
Advantages of SSA over use-def chains

- More compact representation
- Easier to update?
- Each USE has only one definition
- Definitions are explicit merging of values definitions may still reach multiple φ-node
“Flavors” of SSA

Where do we place $\phi$-nodes?

**Condition:**
If two non-null paths $X \rightarrow^* Z$ and $Y \rightarrow^* Z$ converge at node $Z$, and nodes $X$ and $Y$ contain assignments to $V$ (in the original program), then a $\phi$-node for $V$ must be inserted at $Z$ (in the new program).

**minimal**
As few as possible subject to condition

**Briggs-minimal**
Invented by Preston Briggs
As few as possible subject to condition, and $V$ must be live across some basic block

**pruned**
As few as possible subject to condition, and no dead $\phi$-nodes
Dominance Frontiers Revisited

The *dominance frontier* of $X$ is the set of nodes $Y$ s.t. $X$ dominates a predecessor of $Y$, but $X$ does not strictly dominate $Y$.

$$DF(X) = \{Y \mid \exists P \in \text{pred}(Y), (X \text{ DOM } P \text{ and } X \not\text{ DOM! } Y)\}$$

If $X$ appears on every path from *entry* to $Y$, then $X$ *dominates* $Y$ ($X \text{ DOM } Y$).

If $X \text{ DOM } Y$ and $X \neq Y$, then $X$ *strictly dominates* $Y$ ($X \text{ DOM! } Y$).

The *immediate dominator* of $Y$ (IDOM($Y$)) is the closest strict dominator of $Y$.

IDOM($Y$) is $Y$'s parent in the *dominator tree*.
Dominance Frontier Example

\[
\begin{align*}
A &= \\
A &= A = \\
DF(9) &= \\
DF\{8, 9\} &= \\
DF(10) &= \\
\quad = \\
DF(8) &= \\
\quad = \\
DF(2) &= \\
DF\{8, 9\} &= \\
DF(10) &= \\
\quad = \\
DF\{2, 8, 9, 10\} &= \\
\end{align*}
\]
Iterated Dominance Frontier

Extend the dominance frontier mapping from nodes to sets of nodes:

\[ \text{DF}(\mathcal{L}) = \bigcup_{X \in \mathcal{L}} \text{DF}(X) \]

The *iterated* dominance frontier \( \text{DF}^+(\mathcal{L}) \) is the limit of the sequence:

\[ \begin{align*}
\text{DF}_1 &= \text{DF}(\mathcal{L}) \\
\text{DF}_{i+1} &= \text{DF}(\mathcal{L} \cup \text{DF}_i)
\end{align*} \]

**Theorem 1**

The set of nodes that need \( \phi \)-nodes for any variable \( V \) is the iterated dominance frontier \( \text{DF}^+(\mathcal{L}) \), where \( \mathcal{L} \) is the set of nodes with assignments to \( V \).
Inserting φ-nodes

for each variable \( V \)
   \( HasAlready \leftarrow \emptyset \)
   \( EverOnWorkList \leftarrow \emptyset \)
   \( WorkList \leftarrow \emptyset \)
   for each node \( X \) containing an assignment to \( V \)
      \( EverOnWorkList \leftarrow EverOnWorkList \cup \{ X \} \)
      \( WorkList \leftarrow WorkList \cup \{ X \} \)
   end for
while \( WorkList \neq \emptyset \)
   remove \( X \) from \( WorkList \)
   for each \( Y \in DF(X) \)
      if \( Y \notin HasAlready \)
         insert a φ-node for \( V \) at \( Y \)
         \( HasAlready \leftarrow HasAlready \cup \{ Y \} \)
      if \( Y \notin EverOnWorkList \)
         \( EverOnWorkList \leftarrow EverOnWorkList \cup \{ Y \} \)
         \( WorkList \leftarrow WorkList \cup \{ Y \} \)
   end for
end while
endfor
Renaming the variables

Data Structures

Stacks array of stacks, one for each original variable \( V \)
   The subscript of the most recent definition of \( V \)
   Initially, \( \text{Stacks}[V] = \text{EmptyStack}, \forall V \)

Counters an array of counters, one for each original variable
   The number of assignments to \( V \) processed
   Initially, \( \text{Counters}[V] = 0, \forall V \)

procedure GenName(Variable \( V \))
   \( i \leftarrow \text{Counters}[V] \)
   replace \( V \) by \( V_i \)
   Push \( i \) onto \( \text{Stacks}[V] \)
   \( \text{Counters}[V] \leftarrow i + 1 \)

Rename - a recursive procedure

- Walks the dominator tree in preorder
- Initially, call Rename(entry)

CS 380C Lecture 4 11 Static Single Assignment
Renaming the variables

procedure Rename(Block X)

// first process ϕ-nodes
for each ϕ-node P in X
    GenName(LHS(P))

// then process statements in block X
for each statement A in X
    for each variable V ∈ RHS(A)
        replace V by V_i, where i = Top(Stacks[V])
    for each variable V ∈ LHS(A)
        GenName(V)

// then update any ϕ-functions in CFG successors of X
for each Y ∈ SUCC(X)
    j ← position in Y’s ϕ-nodes corresponding to X
    for each ϕ-node P in Y
        replace the j^th operand of RHS(P) by V_i
            where i = Top(Stacks[V])

// recursively visit children of X in dominator tree
for each Y ∈ SUCC(X)
    Rename(Y)

// when backing out of X, pop variables defined in X
for each ϕ-node or statement A in X
    for each V_i ∈ LHS(A)
        Pop (Stacks[V])
What happens to Stacks during Renaming?

\[
\begin{align*}
V &\leftarrow \\
\vdots \\
V &\leftarrow \\
\vdots \\
V &\leftarrow \\
\end{align*}
\]

Stacks  

\begin{align*}
\text{Before} & \\
V & \rightarrow i \rightarrow \ldots \rightarrow 0
\end{align*}

Stacks  

\begin{align*}
\text{After} & \\
V & \rightarrow i+3 \rightarrow i+2 \rightarrow i+1 \rightarrow i \\
\vdots & \rightarrow 0
\end{align*}
Computing SSA Form

- Compute dominance frontiers
- Insert $\phi$-nodes
- Rename variables

**Theorem 2**

Any program can be put into minimal SSA form using this algorithm.

**Translating Out of SSA Form**

- Restore original names to variables
- Delete all $\phi$-nodes
- Replace $\phi$-nodes with copies in predecessors
Translating Out of SSA Form

\[
\begin{align*}
B_1 & \text{ if } (\ldots) \\
B_2 & X_0 \leftarrow 5 \\
B_3 & X_1 \leftarrow 3 \\
B_4 & X_2 \leftarrow \phi(X_0, X_1) \\
& \quad Y \leftarrow X_2
\end{align*}
\]
Next Time

Static Single Assignment

- Induction variables (standard vs. SSA)
- Loop Invariant Code Motion with SSA