Control Flow Analysis

Last Time

- Constant propagation
- Dominator relationships

Today

- Static Single Assignment (SSA) - a sparse program representation for data flow
- Dominance Frontier

Computing Static Single Assignment (SSA) Form

Overview

- What is SSA?
- Advantages of SSA over use-def chains
- “Flavors” of SSA
- Dominance frontier
- Control dependence
- 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?

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

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
  \( \phi \)-node merge together multiple definitions
  definitions may reach multiple \( \phi \)-node
"Flavors" of SSA

Where do we place $\phi$-nodes?

**Condition:**
If two non-null paths $X \to Z$ and $Y \to Z$ 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

Motivating Example

Where do you put the $\phi$-nodes?
**Dominance Frontiers**

**Intuitively:** The dominance frontier indicates a join point of control flow where two or more potential definitions can come together.

DF(v) dominance frontier of v is a set.
DF(v) includes w iff
- v dominates some predecessor of w
- v does not strictly dominate w

\[
DF(v) = \{ w | \exists u \in \text{PRED}(w) \text{ [ } v \text{ DOM } u \text{ ] } \land \text{ } v \text{ DOM! } w \}\]

Recall:
- If X appears on every path from entry to Y, then X dominates Y (X DOM Y).
- If X DOM Y and X ≠ Y, then X strictly dominates Y (X 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**

**Intuitively:** The dominance frontier indicates a join point of control flow where two or more potential definitions can come together.

$DF(v)$ dominance frontier of $v$ is a set. $DF(v)$ includes $w$ iff
- $v$ dominates some predecessor of $w$
- $v$ does not strictly dominate $w$

$$DF(v) = \{w | \exists u \in PRED(w) \ [v \ DOM u] \land v \ DOM! w\}$$

**Algorithm:**
- `procedure FindDF(v)
  forall v
    if (the number of predecessors of $v \geq 2$) then
      forall predecessors $p$ of $v$
        runner = $p$
        while (runner $\neq$ IDOM($v$))
          add $v$ to $DF$ (runner)
          runner = IDOM(runner)
      endwhile
    endif
  endfor
endprocedure`

**Dominance Frontier Example**

```
|   1   |   7   |
|  2 A= |  8 A= |
|  3   |  9 A= |
|  4   | 10   |
|  5   |  6   |
|   A= | 11   |

DF(1) =  1 2 3 4  5  6  7  8  9 10
DF(2) =  3 4  5  6  7  8
DF(3) =  4  6  7  8
DF(4) =  5  6  7
DF(5) =  6
```

CS 380C Lecture 7  11   Static Single Assignment
CS 380C Lecture 7 12   Static Single Assignment
**Iterated Dominance Frontier**

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

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

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

\[ DF_1 = DF(\mathcal{L}) \\
DF_{i+1} = DF(\mathcal{L} \cup DF_i) \]

**Theorem 1**

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

---

**Inserting \( \phi \)-nodes**

```plaintext
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 \( \phi \)-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 control flow graph in preorder
- Initially, call **Rename**(entry)
Renaming the variables

procedure Rename(Block X)
    if X visited return
    for each φ-node P in X
        GenName(LHS(P))
    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)
    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])
    for each Y ∈ SUCC(X)
        Rename(Y)
    for each φ-node or statement A in X
        for each V_i ∈ LHS(A)
            Pop (Stacks[V])

What happens to Stacks during Renaming?

Before

Stacks

V

... i

0

......

After

Stacks

V

... +3

+2

+1

i

... 0

......
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
Static Single Assignment

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

Cytron et al. Dominance Frontier Algorithm

let \( SUCC(S) = \bigcup_{s \in S} SUCC(s) \)

\[ \text{DOM}^{-1}(v) = \text{DOM}^{-1}(v) - v, \text{ then} \]

\[ \text{DF}(v) = SUCC(\text{DOM}^{-1}(v)) - \text{DOM}^{-1}(v) \]

\[ \text{DF}(v) = \text{DF}_{\text{local}}(v) \bigcup_{c \in \text{Child}(v)} \text{DF}_{\text{up}}(c) \]

Child\((v)\): children of \(v\) in the dominator tree

\( \text{DF}_{\text{local}}(v) = \{w | w \in SUCC(v) \land \text{DOM}^{-1}(w) = v\} \)

\( \text{DF}_{\text{up}}(w) \) is the subset of \( \text{DF}(w) \) that is not strictly dominated by \( \text{IDOM}(w) \) (\( \text{IDOM}(w) = v \)).

Algorithm:

\begin{verbatim}
    procedure FindDF(v)
        DF(v) = empty
        for w \in DomChild(v) do
            FindDF(w)
            for u in DF(w) do
                if not( v DOM! u ) then
                    add u to DF(v)
            endfor
        endfor
        for w in SUCC(v) do
            if not( v DOM! w ) then
                add w to DF(v)
        endfor
    endprocedure
\end{verbatim}