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 ϕ-nodes
- Renaming the variables
- Translating out of SSA form

R. Cytron, J. Ferrante, B. K. Rosen, M. N. Wegman, and F. K. Zadeck, "Efficiently Computing Static Single Assignment Form and the Control Dependence Graph", *ACM TOPLAS* 13(4), October, 1991, pp. 451–490.

What is SSA?

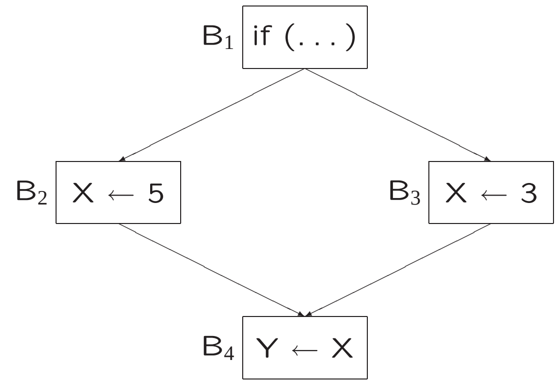
- Each assignment to a variable is given a unique name
- All of the uses reached by that assignment are renamed
- Easy for straight-line code

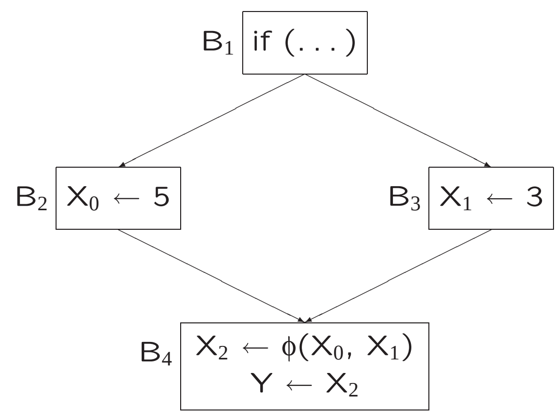
V	$\leftarrow 4$	V_0	$\leftarrow 4$
	$\leftarrow V + 5$		$\leftarrow V_0 + 5$
V	$\leftarrow 6$	V_1	$\leftarrow 6$
	$\leftarrow V + 7$		$\leftarrow V_1 + 7$

What about control flow?

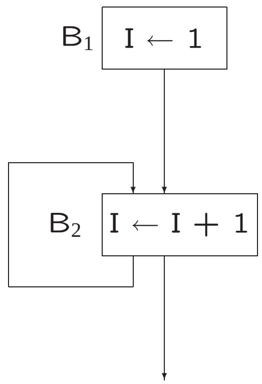
$\Longrightarrow$ ϕ-nodes

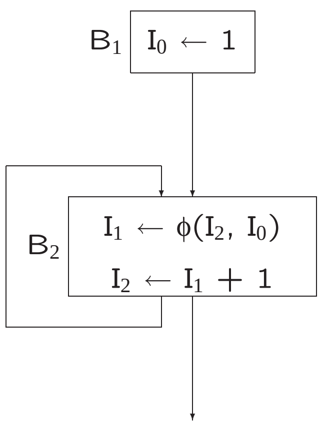
What is SSA?





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
  ϕ-node merge together multiple definitions
  definitions may reach multiple ϕ-node

"Flavors" of SSA

Where do we place ϕ-nodes?

Condition:

If two non-null paths $X \xrightarrow{+} Z$ and $Y \xrightarrow{+} Z$ converge at node Z, and nodes X and Y contain assignments to V (in the original program), then a ϕ-node for V must be inserted at Z (in the new program).

minimal

As few as possible subject to condition

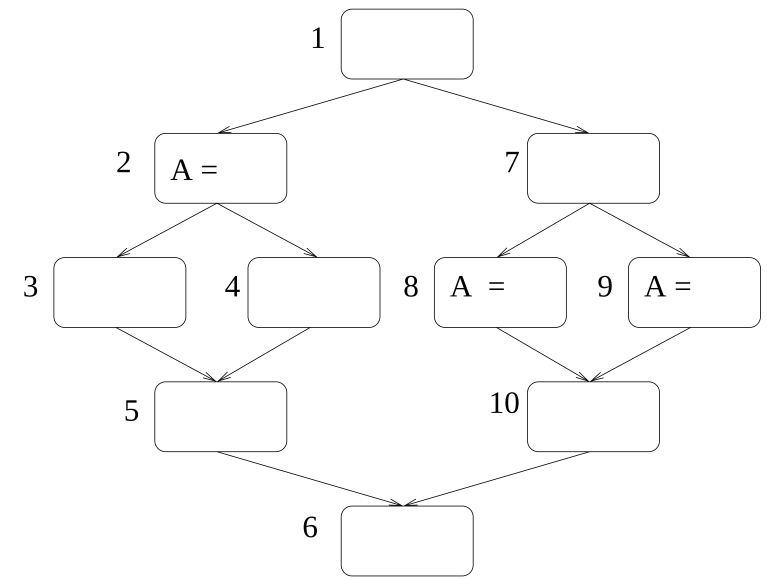
Briggs-minimal Invented by Preston Briggs

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 ϕ-nodes

Motivating Example



Where do you put the ϕ-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

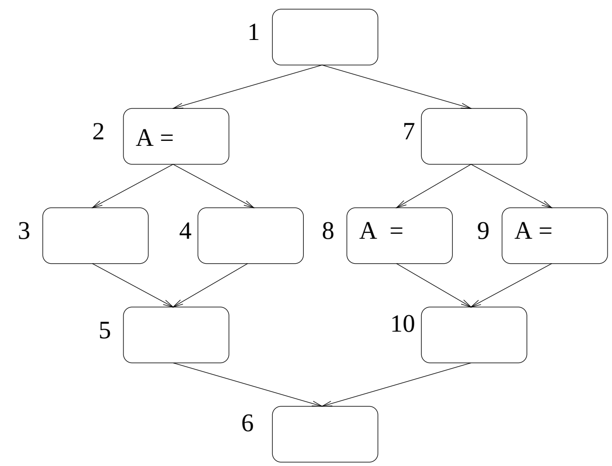
- v dominates some predecessor of w
- v does not strictly dominate w

$$DF(v) = \{w \mid (\exists u \in \text{PRED}(w))\ [v \text{ DOM } u] \wedge v\ \overline{\text{DOM!}}\ w\}$$

Remember:

- If X appears on every path from *entry* to Y, then X *dominates* Y (X DOM Y).
- If X DOM Y and X $\neq$ 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 Example



DF(8) =

DF(9)=

DF(2)=

DF({8,9}) =

DF(10) =

DF({2,8,9,10}) =

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

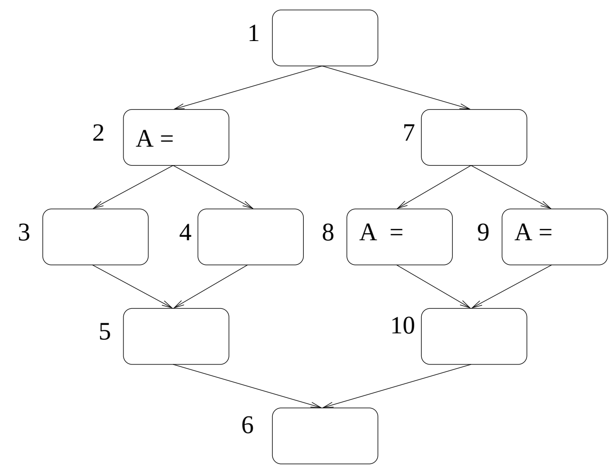
- v dominates some predecessor of w
- v does not strictly dominate w

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

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
        endfor
    endif

Dominance Frontier Example



DF(1) =   DF(6) =

DF(2)=   DF(7) =

DF(3)=   DF(8) =

DF(4) =   DF(9) =

DF(5) =   DF(10) =

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:

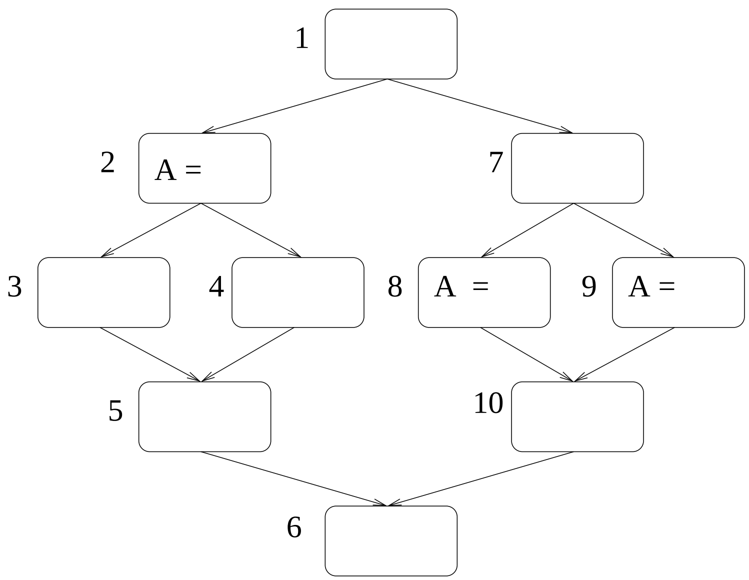
$$DF_1 = DF(\mathcal{L})$$
$$DF_{i+1} = DF(\mathcal{L} \bigcup DF_i)$$

Theorem 1
The set of nodes that need ϕ-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 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* $\bigcup \{X\}$
        WorkList $\leftarrow$ *WorkList* $\bigcup \{X\}$
    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* $\bigcup \{Y\}$
                if Y $\notin$ *EverOnWorkList*
                    EverOnWorkList $\leftarrow$ *EverOnWorkList* $\bigcup \{Y\}$
                    WorkList $\leftarrow$ *WorkList* $\bigcup \{Y\}$
        end for
    end while
endfor

Inserting ϕ-node Example



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, Stacks[V] = EmptyStack, $\forall$ V

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

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

Rename - a recursive procedure

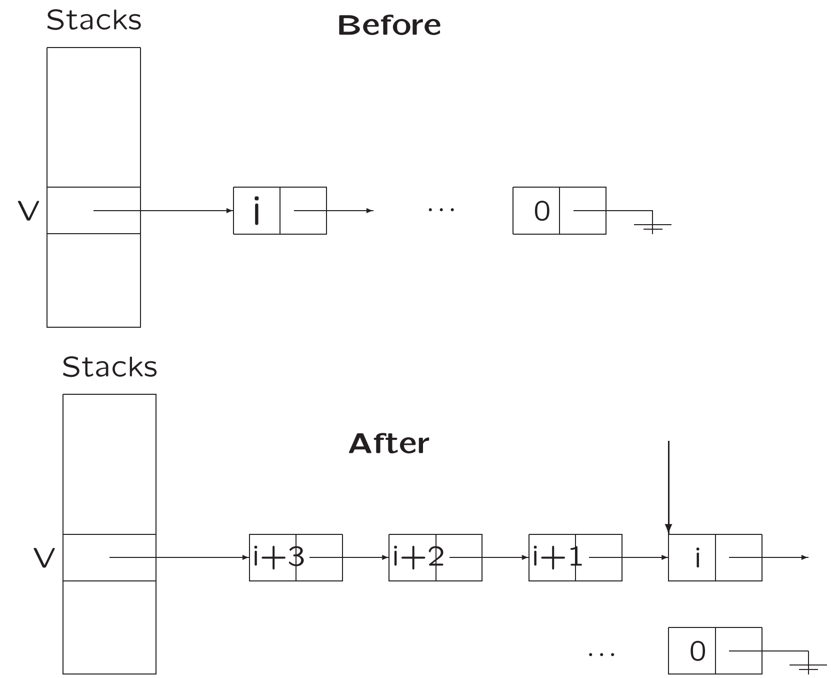
- Walks the control flow graph in preorder
- Initially, call Rename(*entry*)

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?

V ←
⋮
V ←
⋮
V ←



Computing SSA Form

Compute dominance frontiers

Insert ϕ-nodes

Rename variables

Theorem 2

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

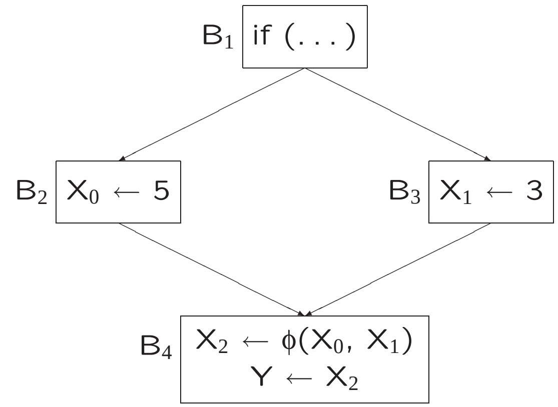
Translating Out of SSA Form

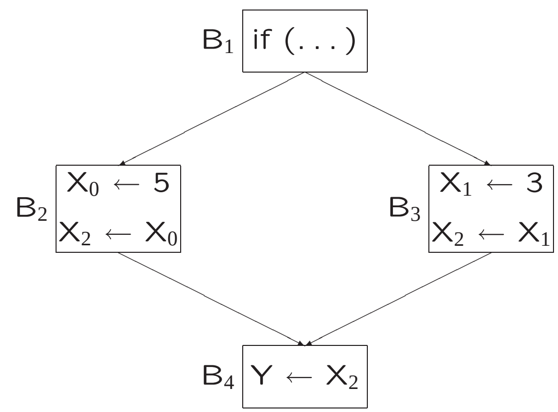
Restore original names to variables

Delete all ϕ-nodes

Replace ϕ-nodes with copies in predecessors

Translating Out of SSA Form





Next Time

Static Single Assignment

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

Cytron et al. Dominance Frontier Algorithm

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

$\text{DOM!}^{-1}(v) = \text{DOM}^{-1}(v)$ - v, then

$$\text{DF}(v) = \text{SUCC}(\text{DOM}^{-1}(v)) - \text{DOM!}^{-1}(v)$$

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

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

$\text{DF}_{local}(v) = \{w | w \in \text{SUCC}(v)\ v\overline{\text{DOM!}}w\}$

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

Algorithm:

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