

Static Single Assignment (SSA)

What is **SSA**?

- each assignment to a variable is given a unique name
- all of the uses reached by that assignment are renamed

Example:

$v := 4;$	$v_1 := 4;$
$:= v + 5$	$:= v_1 + 5$
$v := 6;$	$v_2 := 6;$
$:= v + 7$	$:= v_2 + 7$

Why is this useful?

- representation explicitly connects definitions and uses
- more compact representation than DU and UD chains
- merging of values is explicit

⇒ makes many transformations easier

cs671, spring'08

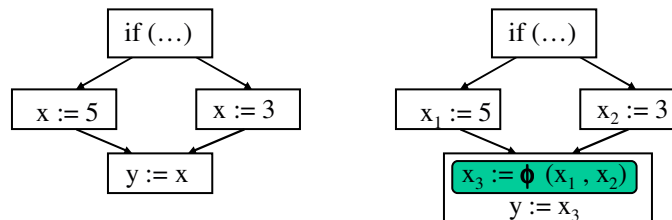
Handling Multiple Reaching Definitions

ϕ -functions (or ϕ -nodes)

For some CFG node N and variable v a function of the form

$$v := \phi(v_{p_1}, v_{p_2}, \dots),$$

where each v_{p_i} corresponds to the definition of v reaching this point through node N 's predecessor p_i .



cs671, spring'08

Handling Multiple Reaching Definitions

Where do we place ϕ -functions ?

Intuitively: At the first point where paths in the CFG merge that have distinct definitions for the same variable.

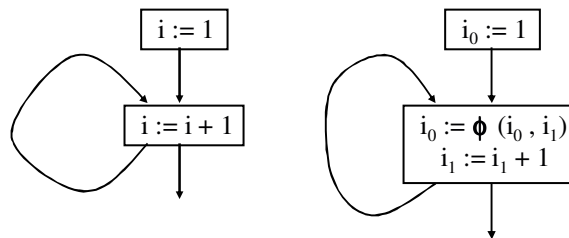
Formally: If there exist two non-null paths $X \rightarrow^* Z$, and $Y \rightarrow^* Z$ that converge for the first time at node Z , and nodes X and Y contain assignments to variable v (in the original program), then a ϕ -function must be inserted at Z (in the new program).

Placement of ϕ -function subject to this condition yields "*minimal*" SSA form.

cs671, spring'08

Handling Multiple Reaching Definitions

Another Example: loops



cs671, spring'08

Computing SSA Form

1. Insert ϕ -functions
 - (a) Build dominator tree
 - (b) Compute the dominance frontier and their closure
 - (c) Rename variables
2. Translate back to SSA form.

“The SSA graph is simply the graph built by adding def-use and/or use-def chains to the program in SSA form during SSA construction”.

R. Cytron et al., “Efficiently computing static single assignment form and the control dependence graph”, ACM Transactions on Programming Languages and Systems (TOPLAS), 13(4), Oct. 1991.

cs671, spring'08

Insert ϕ -functions: Dominators

Review.

If X appears on every path from ENTRY node to Y , then
 X *dominates* Y ($X \geq Y$)

If $X \geq Y$ and $X \neq Y$, then
 X *strictly dominates* Y ($X \gg Y$)

The *immediate dominator* Y ($\text{idom}(Y)$) is the closest strict dominator of Y

$\text{idom}(Y)$ is Y 's parent in the *dominator tree*

cs671, spring'08

Dominance Frontiers

The *dominance frontier* of node X is the set of nodes Y such that

X dominates a predecessor of Y , but
 X does not strictly dominate Y

$$DF(X) = \{ Y \mid \exists Z \in \text{Pred}(Y), X \geq Z \text{ and } \text{not}(X \gg Y) \}$$

The dominance frontier can be subdivided into two components:

$$DF_{\text{local}}(X) = \{ Y \in \text{Succ}(X) \mid \text{not}(X \gg Y) \}$$

$$DF_{\text{up}}(X) = \{ Y \in DF(Z) \mid Z \in \text{Children}(X) \text{ and } \text{not}(X \gg Y) \}$$

Then

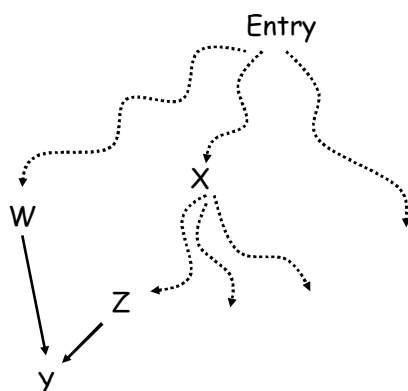
$$DF(X) = DF_{\text{local}}(X) \cup DF_{\text{up}}(X)$$

Succ = immediate successors in the CFG

Children = descendents in the dominator tree

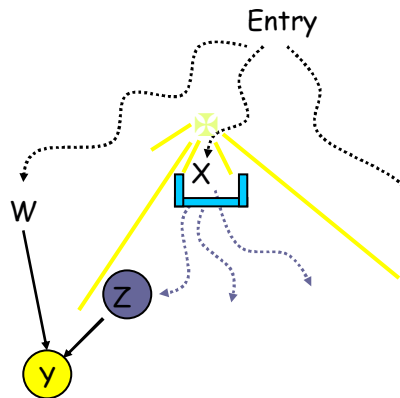
cs671, spring'08

Intuition for Dominance Frontiers



cs671, spring'08

Intuition for Dominance Frontiers



Barry Rosen, IBM TJ Watson:
"Place an opaque shade at node X
with a light source at X.
Consider all dark edges incident on a
lighted node. Their targets form the
dominance frontier for X."

Dominance frontier are points just
beyond the region a node
dominates.

cs671, spring'08

Dominance Frontiers Algorithm

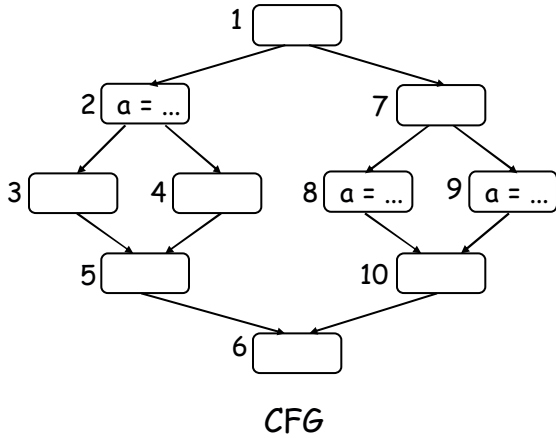
```
for each X in a bottom-up traversal of the  
dominator tree
```

```
DF(X) :=  $\phi$   
for each Y  $\in$  Succ(X) /* local */  
  if idom(Y)  $\neq$  X then  
    DF(X) := DF(X)  $\cup$  { Y }  
  
for each Z  $\in$  Children(X) /* up */  
  for each Y  $\in$  DF(Z)  
    if idom(Y)  $\neq$  X then  
      DF(X) := DF(X)  $\cup$  { Y }
```

Succ = immediate successors in the CFG
Children = descendants in the dominator tree

cs671, spring'08

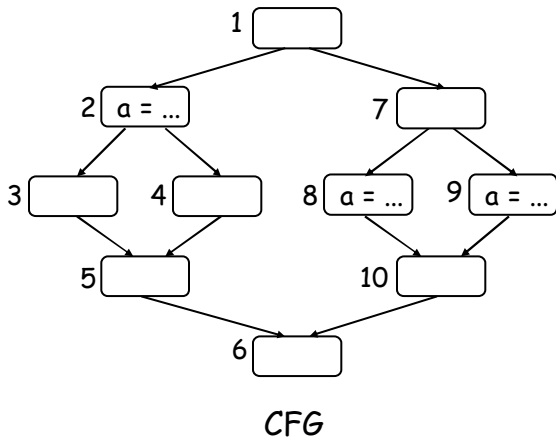
Dominance Frontiers Example



cs671, spring'08

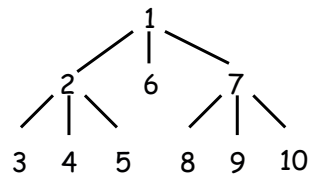
Dominator Tree

Dominance Frontiers Example



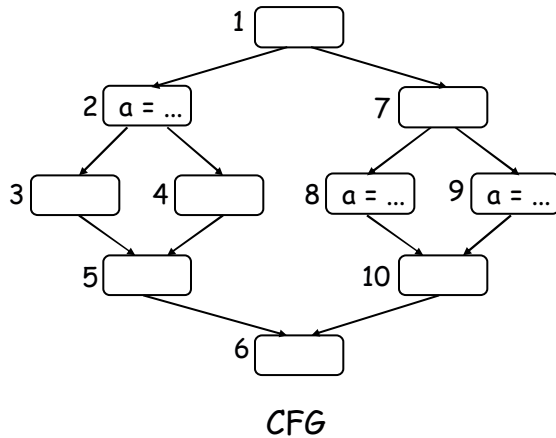
cs671, spring'08

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

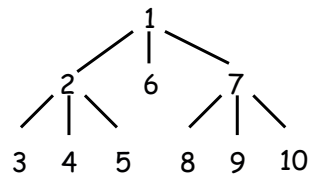


Dominator Tree

Dominance Frontiers Example



- DF(3) = { 5 }
- DF(4) = { 5 }
- DF(5) = { 6 }
- DF(2) = { 6 }
- DF(6) = ϕ
- DF(8) = { 10 }
- DF(9) = { 10 }
- DF(10) = { 6 }
- DF(7) = { 6 }
- DF(1) = ϕ



cs671, spring'08

Dominance Frontiers Closure

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

$$DF(L) = \bigcup_{X \in L} DF(X)$$

The *iterated dominance* frontier $DF^+(L)$ is the limit of the sequence:

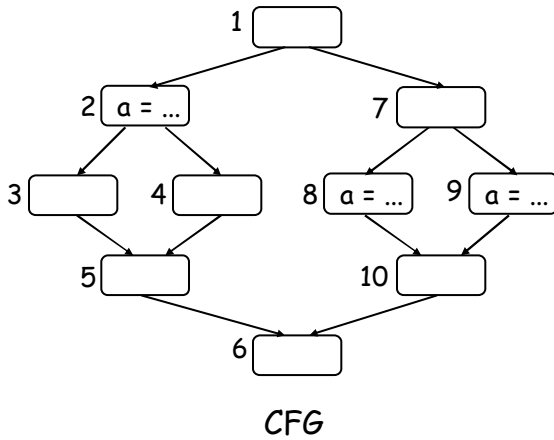
$$\begin{aligned}
 DF_1 &= DF(L) \\
 DF_{i+1} &= DF(L \cup DF_i) \quad (\text{i.e., a closure}) \\
 &[\text{same as } DF(L) \cup DF(DF(L)) \cup DF(DF(DF(L))) \dots]
 \end{aligned}$$

Theorem

The set of nodes that need ϕ -nodes for any variable v is the iterated dominance frontier $DF^+(L)$, where L is the set of nodes with assignments to v .

cs671, spring'08

Dominance Frontiers Example



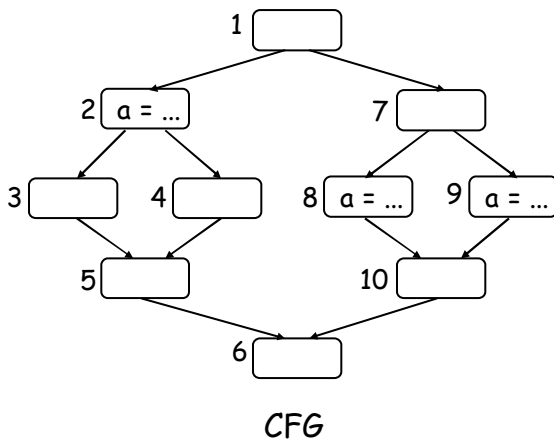
$DF(\{8, 9\}) = \{10\}$
 $DF(10) = \{6\}$
 $DF(6) = \phi$
 $DF(2) = \{6\}$
 $DF(5) = \{6\}$

$DF^+(\{8, 9\})$
 $=$

$DF^+(\{2, 8, 9\})$
 $=$

cs671, spring'08

Dominance Frontiers Example



$DF(\{8, 9\}) = \{10\}$
 $DF(10) = \{6\}$
 $DF(6) = \phi$
 $DF(2) = \{6\}$
 $DF(5) = \{6\}$

$DF^+(\{8, 9\})$
 $= DF^+(\{8, 9, 10\})$
 $= DF^+(\{8, 9, 10, 6\})$
 $= \{6, 10\}$

$DF^+(\{2, 8, 9\})$
 $= DF^+(\{2, 8, 9, 10\})$
 $= DF^+(\{2, 8, 9, 10, 6\})$
 $= \{6, 10\}$

cs671, spring'08

Algorithm for Inserting ϕ -nodes

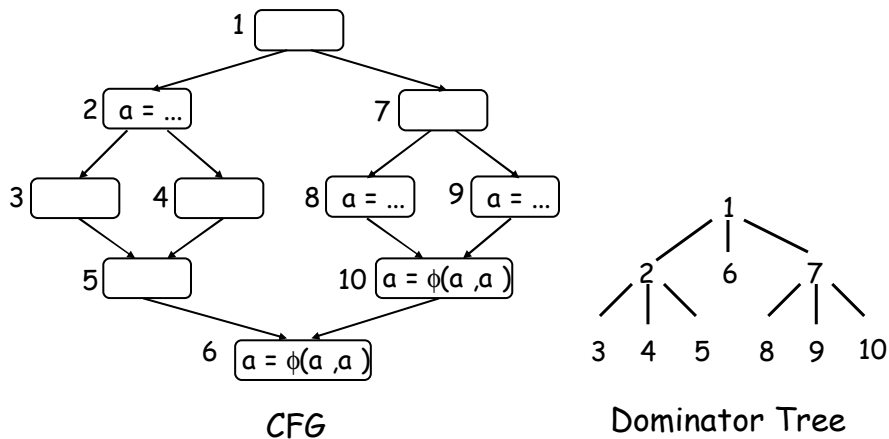
```

for each variable v
  HasAlready := {}
  WorkList := {}
  for each node X containing an assignment to v
    WorkList := WorkList  $\cup$  { X }

while WorkList  $\neq$  {}
  remove X from WorkList
  for each Y  $\in$  DF( X )
    if Y  $\notin$  HasAlready then
      insert a  $\phi$ -node for v at Y
      HasAlready := HasAlready  $\cup$  { Y }
      WorkList := WorkList  $\cup$  { Y }
  
```

cs671, spring'08

Algorithm for Inserting ϕ -nodes



cs671, spring'08

Computing SSA Form

Complete Algorithm:

- compute the dominance frontier
- insert ϕ -nodes
- rename variables

Theorem:

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

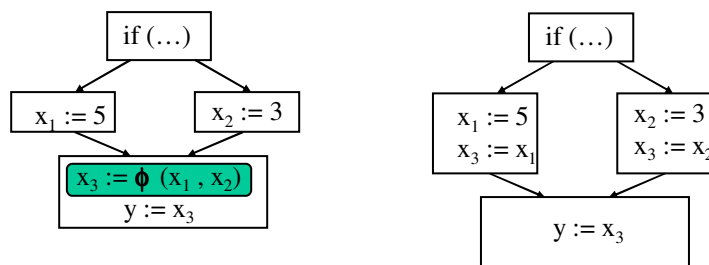
- Optimizations: [fewer ϕ -nodes than "minimal"]
- eliminate any dead ϕ -nodes (requires live analysis)
 - pruned - overall, i.e., across entire procedure
 - semi-pruned - dead on exit of basic block

cs671, spring'08

Computing SSA Form

Translation back to SSA form

- replace ϕ -nodes with copies in CFG predecessors
- delete all ϕ -nodes



cs671, spring'08

SSA Form: Renaming of Variables

Data structures used for renaming variables

Stacks: array of stacks, one for each original variable v

- contains subscript of most recent definition of v
- initially, $Stacks[v] = \text{empty}$ for all variables v
- restores "context" during pre-order walk of dominator tree

Counters: array of integers, one entry for each original variable v

- keeps track of number of processed assignments to v
- initially, $Counters[v] = 0$ for all variables v
- are monotonically increasing

cs671, spring'08

SSA Form: Renaming of Variables

Procedure **GenName**(variable v):

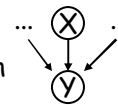
$i := Counters[v]$; replace v by v_i ; push i onto $Stacks[v]$;
 $Counters[v] := i + 1$;

Procedure **Rename** (CFG node X)

- processes nodes in CFG based on preorder in dominator tree
- initially, call **Rename**(entry)

Function **WhichPred** (CFG node X , CFG node Y) return integer

- integer telling which predecessor of Y in CFG is X .
- The j -th operand of a ϕ -function corresponds to the j -th predecessor of Y numbering in-edges from left to right



Convention: each assignment statement S has the form
 $LHS(S) := RHS(S)$

cs671, spring'08

SSA Form: Renaming of Variables

Procedure Rename (CFG node X)

for each ϕ -node P in X do
 GenName(LHS(P))

for each statement S in X
 for each variable $v \in \text{RHS}(S)$ do
 replace v by v_i , where $i = \text{top}(\text{Stacks}[v])$
 for each variable $v \in \text{LHS}(S)$ do GenName(v)

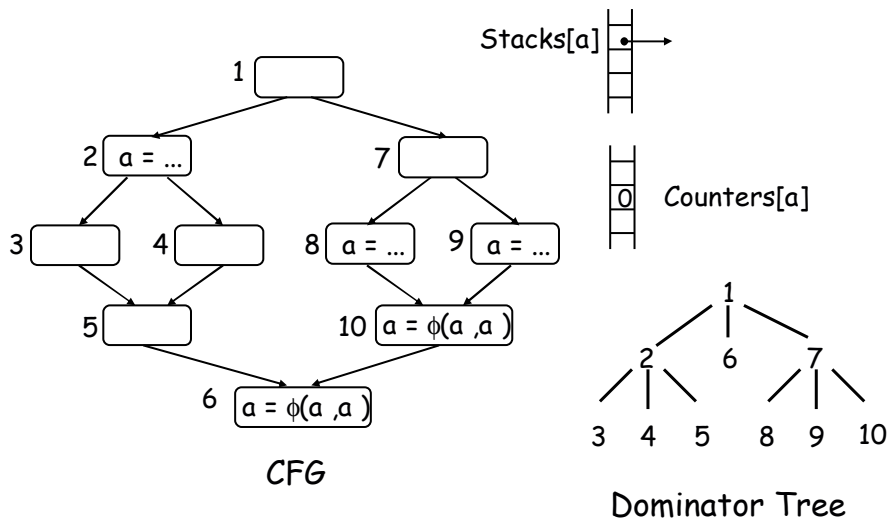
for each $Y \in \text{Succ}(X)$ do
 $j := \text{WhichPred}(X, Y)$
 for each ϕ -node P in Y do
 replace the j^{th} operand of RHS(P) by v_i , where $i = \text{top}(\text{Stacks}[v])$

for each $Z \in \text{Children}(X)$ do Rename(Z)

for each ϕ -node or statement S in X
 for each variable $v_i \in \text{LHS}(S)$ do pop Stacks[v]

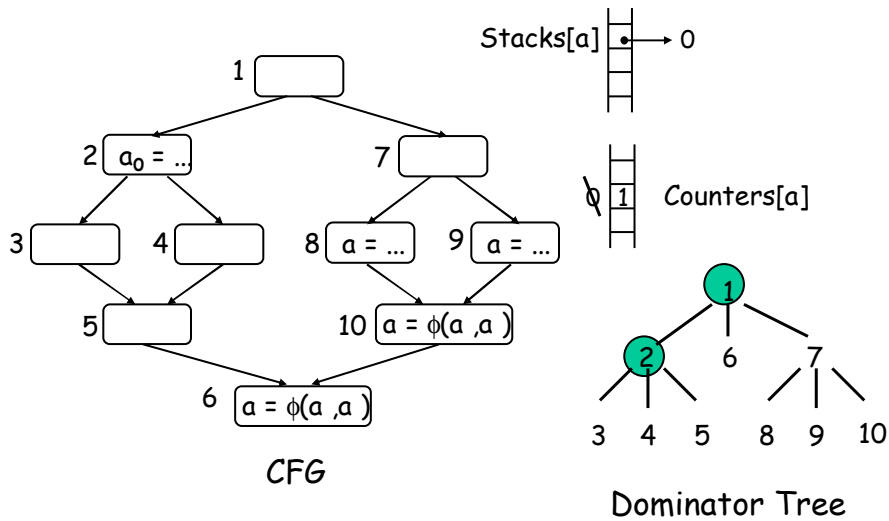
cs671, spring'08

SSA Form: Renaming of Variables



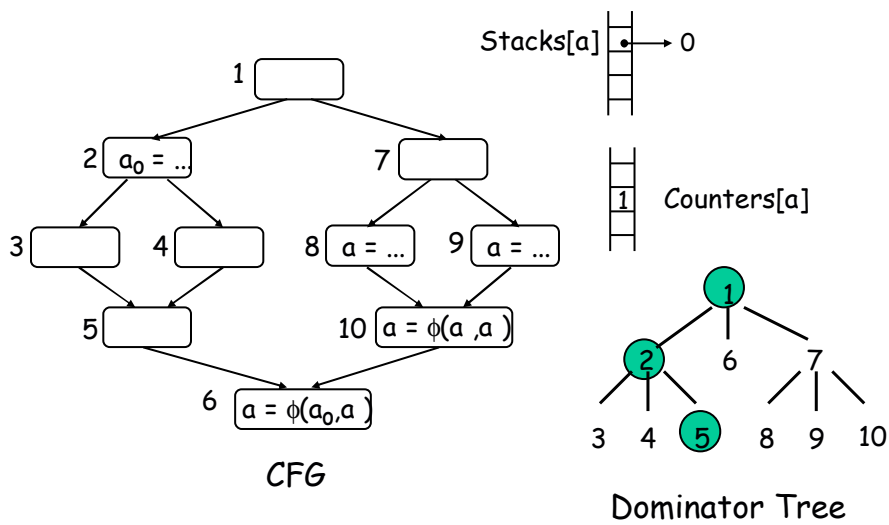
cs671, spring'08

SSA Form: Renaming of Variables



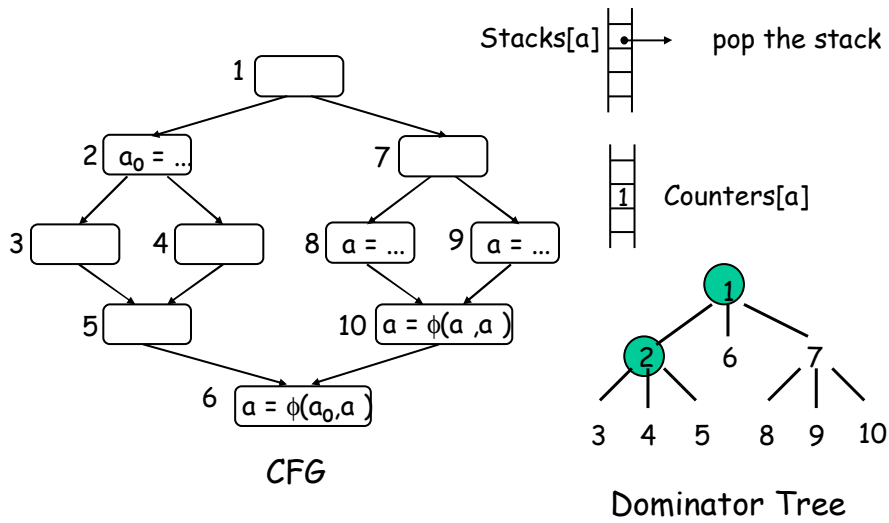
cs671, spring'08

SSA Form: Renaming of Variables



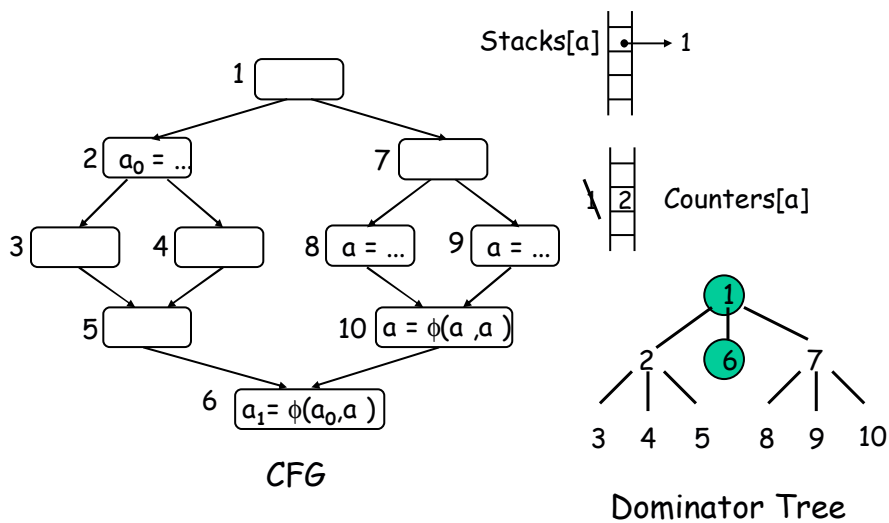
cs671, spring'08

SSA Form: Renaming of Variables



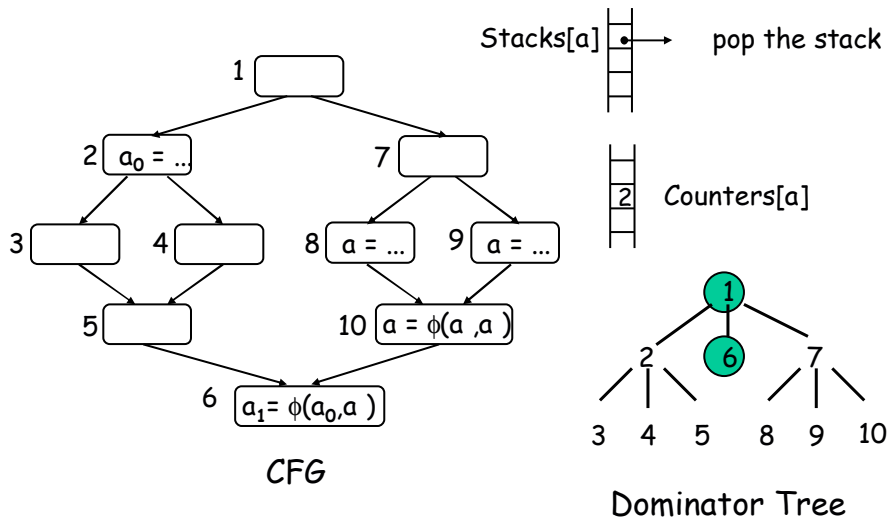
cs671, spring'08

SSA Form: Renaming of Variables



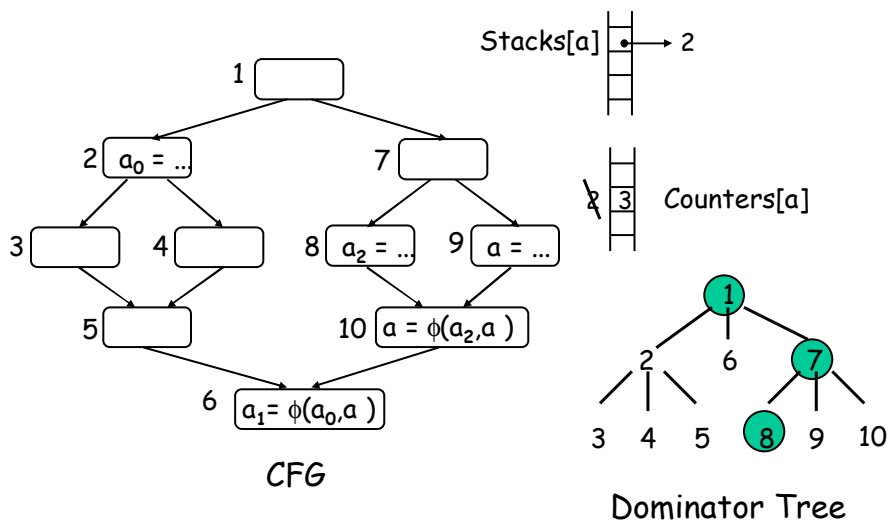
cs671, spring'08

SSA Form: Renaming of Variables



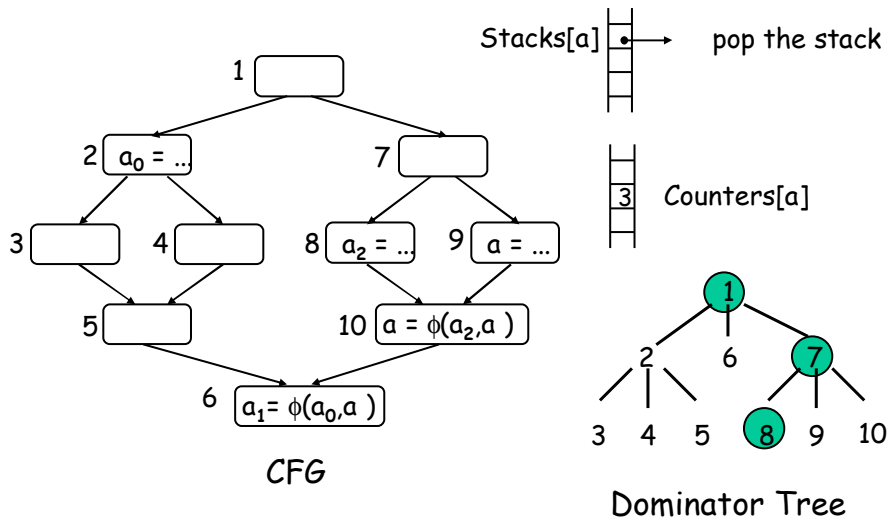
cs671, spring'08

SSA Form: Renaming of Variables



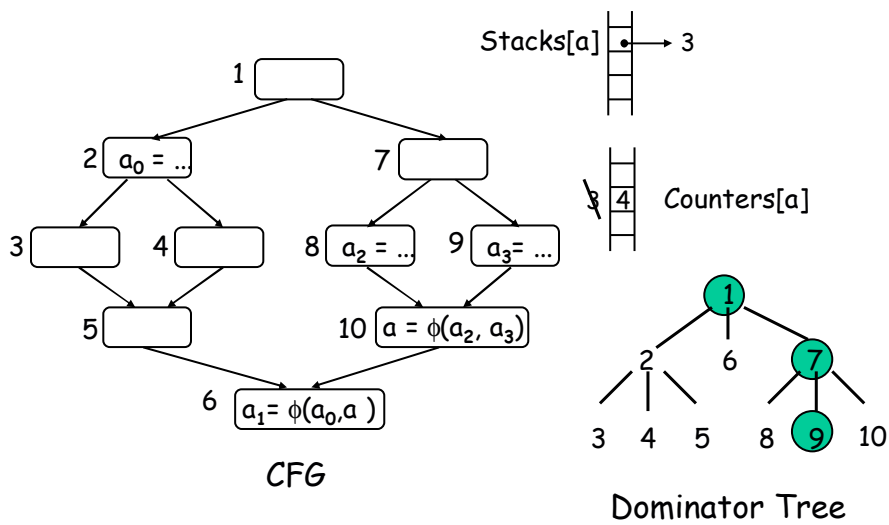
cs671, spring'08

SSA Form: Renaming of Variables



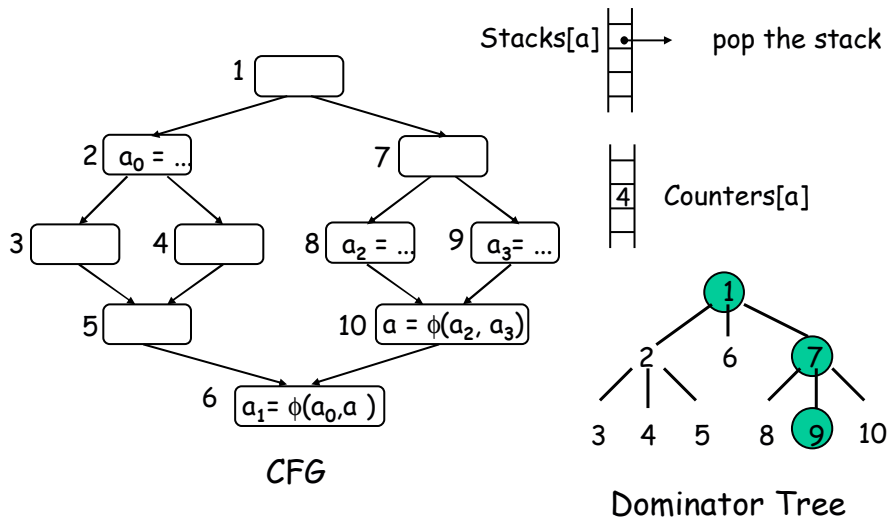
cs671, spring'08

SSA Form: Renaming of Variables



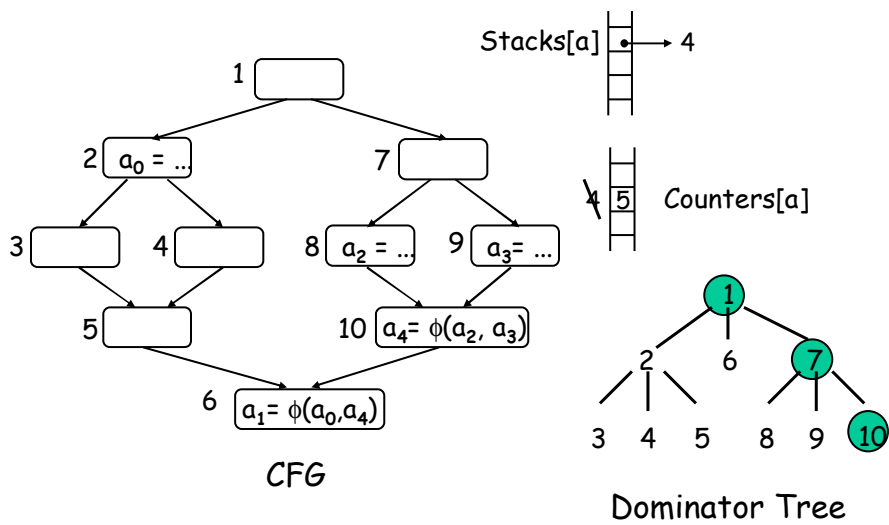
cs671, spring'08

SSA Form: Renaming of Variables



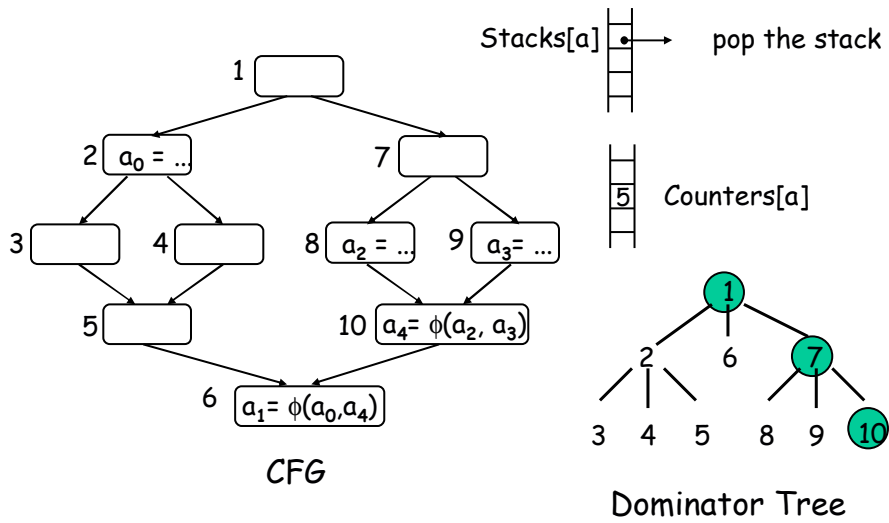
cs671, spring'08

SSA Form: Renaming of Variables

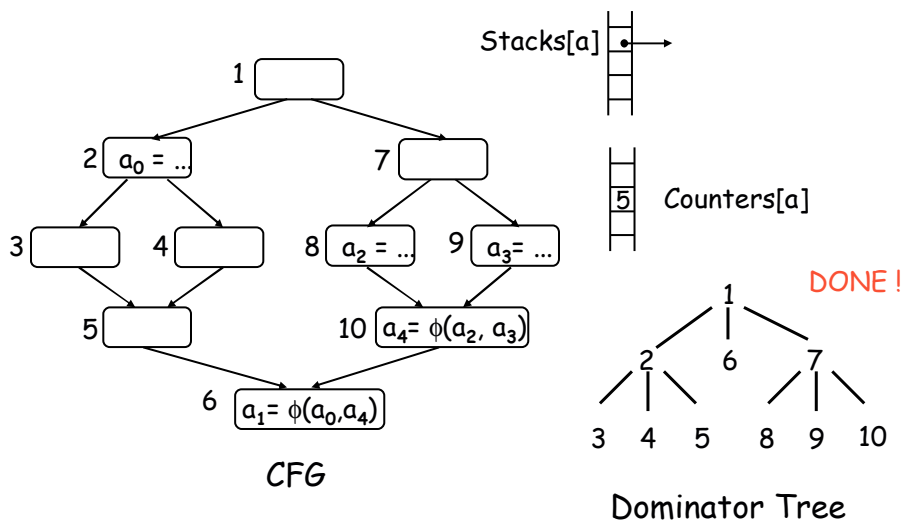


cs671, spring'08

SSA Form: Renaming of Variables

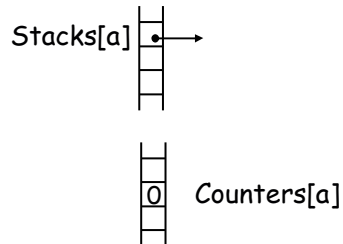


SSA Form: Renaming of Variables



SSA Form: Renaming of Variables

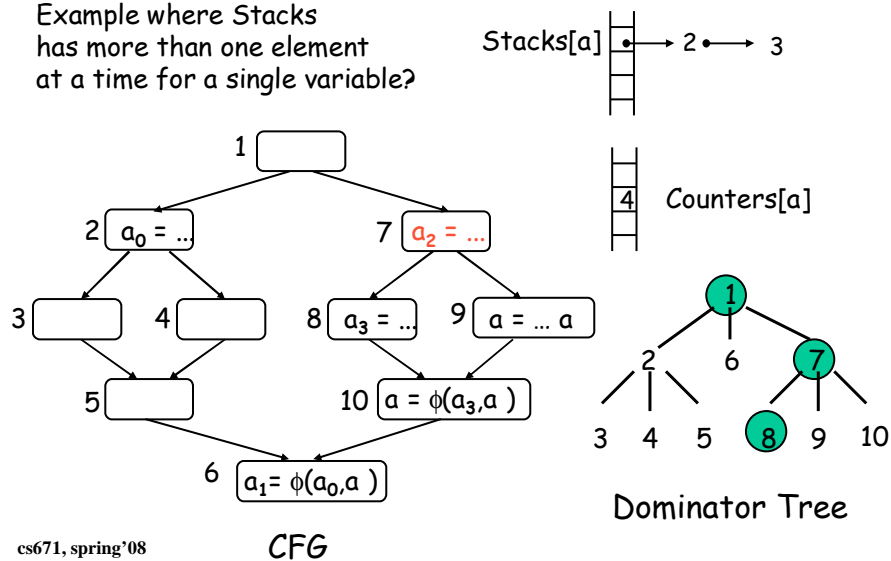
Example where Stacks has more than one element at a time for a single variable?



cs671, spring'08

SSA Form: Renaming of Variables

Example where Stacks has more than one element at a time for a single variable?



cs671, spring'08

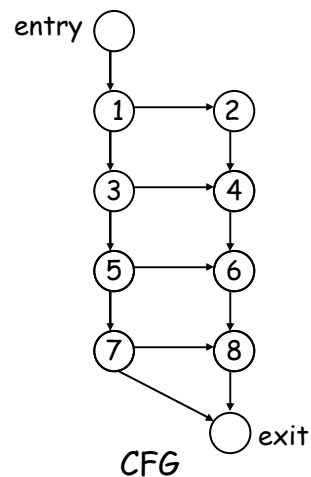
Relation to Control Dependence

Dominance frontiers on reverse CFG yield control dependence relation

- Y is control dependent on X means that there is a logical test at X whose outcome determines if Y will be executed for sure (one branch), or may not be executed (other branch)
- Y *postdominates* Z if Y appears on every path from node Z to unique EXIT node (analogous to dominator on the reverse CFG)
- Y is control dependent on X iff
 - \exists path (X, Z_1, \dots, Z_k, Y) s.t. $\forall Z_i \neq X, Y$ postdominates Z_i
 - X is not postdominated by Y
- Y is control dependent on X in CFG iff $X \in DF(Y)$ on reverse CFG

cs671, spring'08

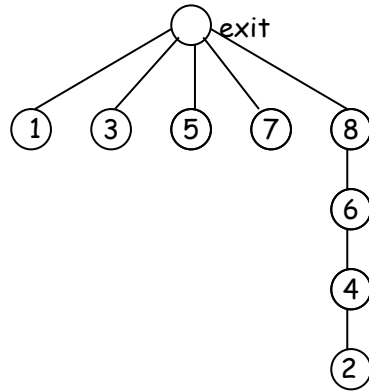
Relation to Control Dependence



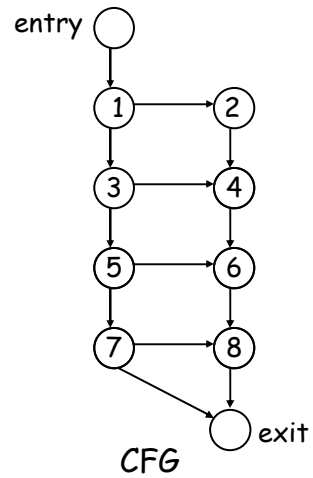
Postdominator Tree

cs671, spring'08

Relation to Control Dependence



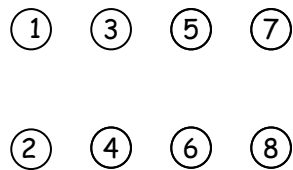
Postdominator Tree



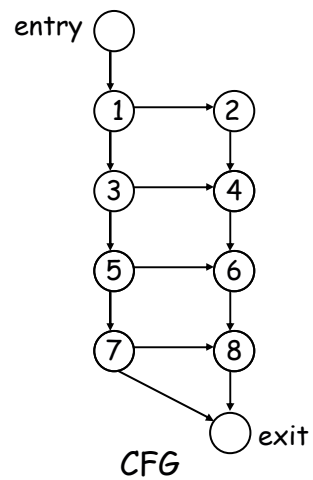
CFG

cs671, spring'08

Relation to Control Dependence



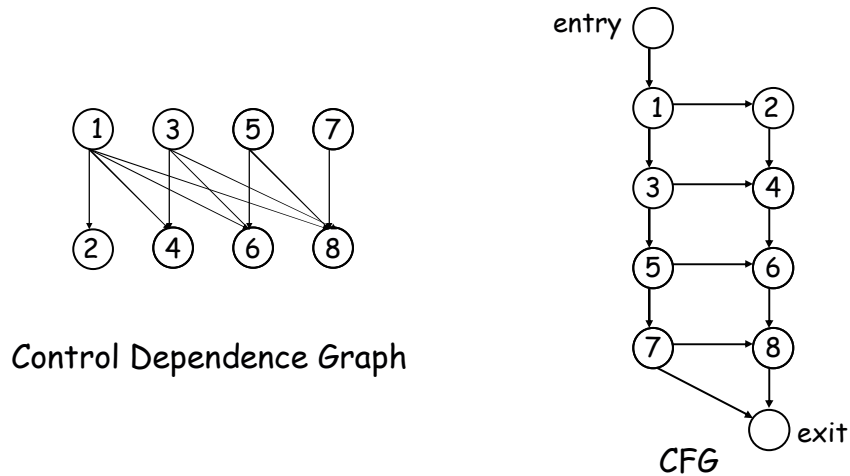
Control Dependence Graph



CFG

cs671, spring'08

Relation to Control Dependence



cs671, spring'08

Observations and Experimental Results [Cytron et al.]

- There are faster ways of computing dominance frontiers(DJ graphs)
- Size of control dependence graph varies linearly with program size in practice, quadratic in theory
- Size of dominance frontier varies linearly with program size
- Claim that number of ϕ -nodes varies linearly with program size

V. Shreedhar and G. Gao., "Computing ϕ -nodes in linear time using DJ graphs",
IJ Programming Languages, 1995.

cs671, spring'08