Def-Use and Use-Def Chains

Application of Reaching Defs

Auxiliary data structures like CFG

Ex. Def-Use Chains

```
X :=
if (...) then X :=     else X:=
:=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 :=
  :=X
X :=
  :=X
if (...) then X :=     else X:=
 :=X
```

```
X 1:=
  :=X1
X2 :=
  :=X 2
if (...) then X3 :=     else X4:=
X 5:=  Φ(X3, X4)
  :=X5
```

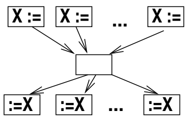
SSA with Def-Use chains

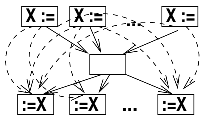
More compact Def-Use representation

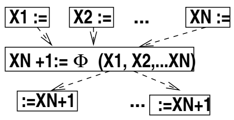
Advantages of SSA

More compact Def-Use representation

Ex. CFG is O(N) **Def-use chains O(N^2)**







SSA with Def-use chains O(N)

More Advantages of SSA

More powerful optimizations

Value Numbering
Program Equivalence
Constant Propagation

Faster optimizations

Constant Propagation
Code Motion

Increased Parallelism

Used in real compilers!

IBM, Tera, Apollo,...

SSA Loop Example

```
(1)     Read(N)
(2)     I := 1

(3)L2: If(I > N)goto L3
(4)         A(I):= A(I) + 1
(5)         I := I + 1
(6)     goto L2
(7) L3:Print(A(N))
```

```
(1)     Read(N)
(2)     I1 := 1
   L2:I3 := Φ (I1,I2)
(3)     If(I3 > N)goto L3
(4)         A(I3):= A(I3) + 1
(5)         I2 := I2 + 1
(6)     goto L2
(7) L3:Print(A(N))
```

Φ makes merge of values explicit

Qu: How construct SSA form?

Computing SSA

Compute Dominance Frontiers

Potential join points, based only on CFG

Place ϕ (Phi) Functions

Join points based on actual assignments in program

Want minimal number

Rename variables

Each use has a unique defintion point

Get minimal SSA!

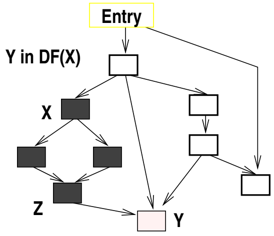
Dominance Frontiers

$\gg$ *Strictly dominates* $\quad\quad \underline{\gg}$ *Dominates*

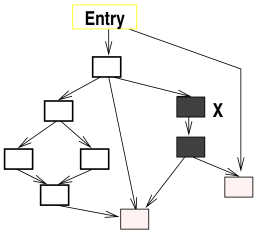
$\mathrm{DF}(X) = \{\, Y \mid \exists\, Z$ a pred of Y $(X \underline{\gg} Z$ and $X \not\gg Y)\}$

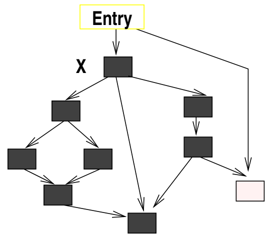
X dominates a predecessor of Y but does not strictly dominate Y

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

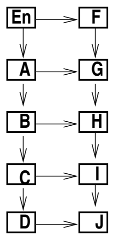


Other Examples





Ladder graph example

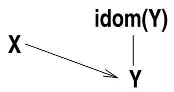


Node X	DF(X)
A	{G,H,I,J}
B	{H,I,J}
C	{I,J}
D	{J}
F	{G}
G	{H}

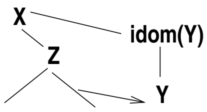
Computing Dominance Frontiers

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

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



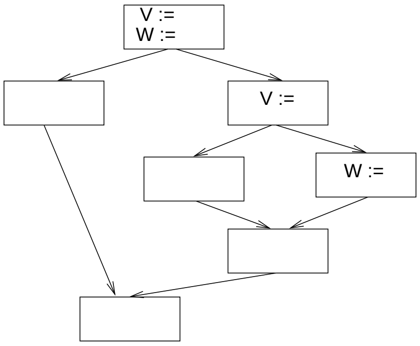
$$DF_{up}(Z) = \{Y \text{ in } DF(Z) \mid idom(Y) \neq X\}$$



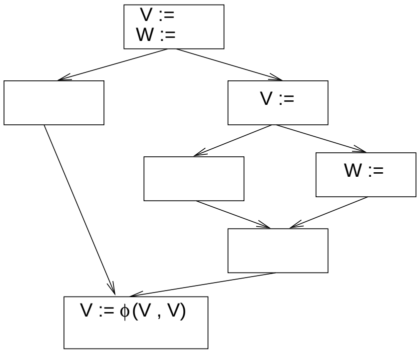
Complexity: Linear in size of $\bigcup$ *of DF sets*

Linear in size of CFG for well-structured programs

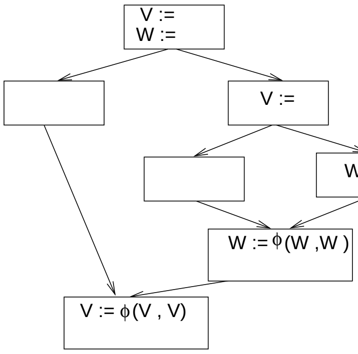
Placing Phi Functions



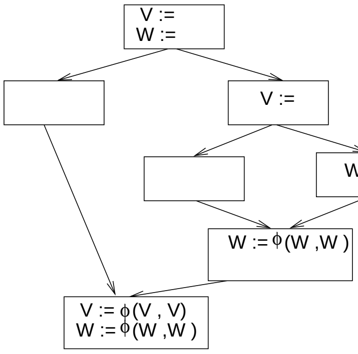
Placing Phi Functions



Placing Phi Functions



Placing Phi Functions



Placing Phi Functions

$DF^1(S) = DF(S)$ $\quad\quad DF^{i+1}(S) = DF(S \cup DF^i(S))$

$DF^+(S) = \lim_i DF^i(S)$

Let S = {nodes with asst's to X} $\cup$ {Entry}

$DF^+(S)$ is the set of nodes that require phi functions for variable X

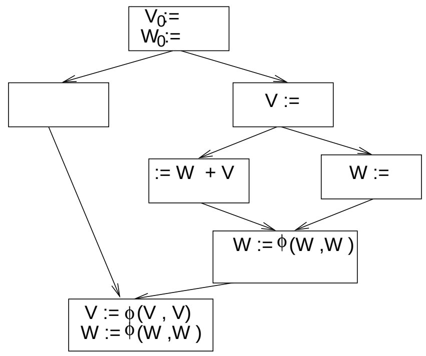
Worklist algorithm, pass for each variable

Initialize worklist with set of all assignments to variable

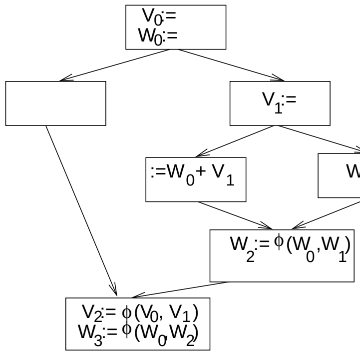
For each Y on worklist, place phi function for each Z in DF(Y) if not already there, and place it on worklist

Complexity: (Total no. Assts. x wt.av.(DF))

Renaming Variables



Renaming Variables



Renaming Variables

Keep stack of indices for each variable

Traversal of Dominator tree, starting from Entry

Visit Node:

RHS Asst in Node:

Rename with index from variable's TOS

LHS Asst in Node:

Create new index, rename, and push

ϕ ***function in jth successor of Node:***

Rename jth variable in ϕ with TOS index

Visit all children of Node in Dom tree

Pop stack for each LHS asst in Node