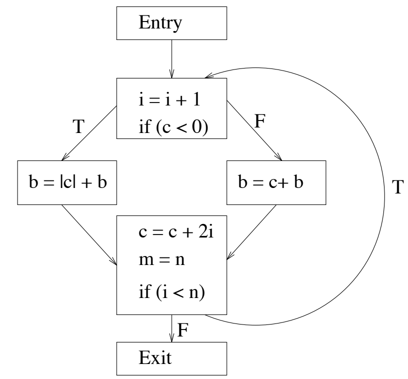
Dominators and Dependence

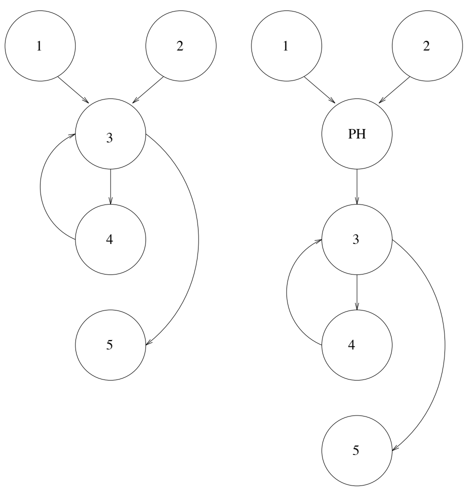
- Dominator relationships (algorithms for control flow graphs):
  - DOM
  - IDOM
  - DOM^{-1}
  - DOM!
  - DF
  - post-dominators
- Control Dependence

Motivating Example: Code Motion



- landing pad
- control dependence graph

Landing Pad (Preheader)



Dominator Relationships

Dominators

d dominates v, d DOM v, in a CFG *iff all* paths from *Entry* to v include d

$\text{DOM}(v)$ = the set of all *vertices that dominate* v

- All vertices dominate themselves, $v \in \text{DOM}(v)$.
- *Entry* dominates every vertex in the graph: $\forall v \in V : Entry \in \text{DOM}(v)$.
- reflexive, antisymmetric, and transitive

Strict Dominators

$\text{DOM!}(v) = \text{DOM}(v) - \{v\}$

- antisymmetric and transitive

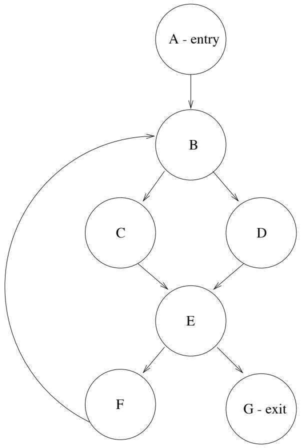
Immediate Dominator

$\text{IDOM}(v) =$ the closest, strict dominator of v

$d \text{ IDOM } v \iff d \text{ DOM! } v \wedge (\forall w \mid w \text{ DOM! } v)[w \text{ DOM } d]$

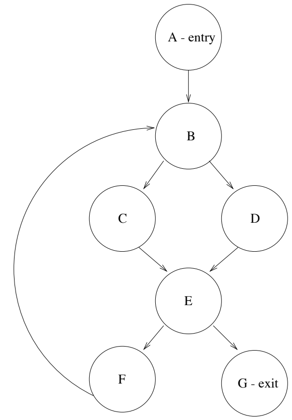
- antisymmetric

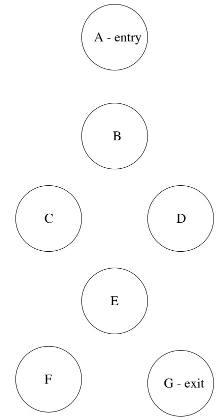
Dominators: Example



v	DOM(v)	DOM!(v)	IDOM(v)
A			
B			
C			
D			
E			
F			
G			

Dominator Tree





Dominator Relationships

Theorem: IDOM(v) is unique (*i.e.*, a singleton)

Proof: by contradiction.

Suppose c IDOM v and d IDOM v.
By definition, $c \neq v$ and $d \neq v$, so c DOM! v and d DOM! v.
By definition of IDOM:

$$(d \text{ DOM! } v) \wedge (\forall w \mid w \text{ DOM! } v)[w \text{ DOM } d]$$

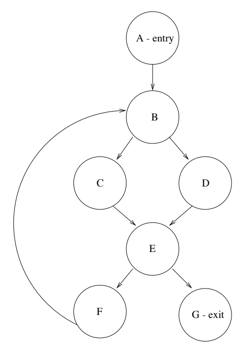
Thus, c DOM d and d DOM c, but DOM is antisymmetric, a contradiction, unless $c = d$

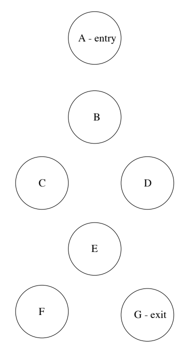
Inverse Dominators

$\text{DOM}^{-1}(v) = \{w \mid v \text{ DOM } w\}$

- reflexive, antisymmetric, and transitive

Inverse Dominators: Example





v	DOM(v)	DOM$^{-1}(v)$
A	$\{A\}$	
B	$\{A,B\}$	
C	$\{A,B,C\}$	
D	$\{A,B,D\}$	
E	$\{A,B,E\}$	
F	$\{A,B,E,F\}$	
G	$\{A,B,E,G\}$	

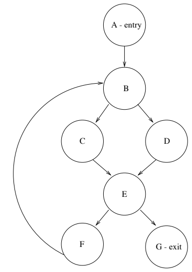
Finding Dominators: Algorithm

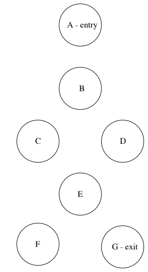
$$\text{DOM}(v) = \{v\} \cup \Big(\bigcap_{p \in PRED(v)} \text{DOM}(p) \Big)$$

$\text{DOM}(Entry) \leftarrow \{Entry\}$
foreach $v \in V - \{Entry\}$ **do** $\text{DOM}(v) \leftarrow V$ **end**
do
    changed $\leftarrow$ *false*
    foreach $v \in V - \{Entry\}$ **do**
        olddom $\leftarrow \text{DOM}(v)$
        $\text{DOM}(v) \leftarrow \{v\} \cup \Big(\bigcap_{p \in PRED(v)} \text{DOM}(p) \Big)$
        changed $\leftarrow \text{DOM}(v) \neq$ *olddom*
    end
while *changed*

Complexity: $\mathbf{O}(N^2)$

Dominator Algorithm: Example





	DOM(v) iteration: 0	1	2
A	$\{A\}$		
B	$\{A,B,C,D,E,F,G\}$		
C	$\{A,B,C,D,E,F,G\}$		
D	$\{A,B,C,D,E,F,G\}$		
E	$\{A,B,C,D,E,F,G\}$		
F	$\{A,B,C,D,E,F,G\}$		
G	$\{A,B,C,D,E,F,G\}$		

Dominance Frontier

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

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

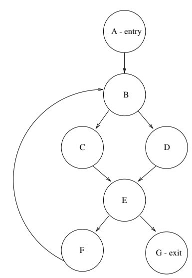
Let

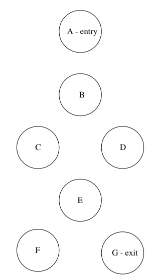
$$SUCC(S) = \bigcup_{s \in S} SUCC(s)$$
$$\text{DOM!}^{-1}(v) = \text{DOM}^{-1}(v) - \{v\}$$

Then

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

Dominance Frontier: Example





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

v	DOM$^{-1}(v)$	$SUCC$(DOM$^{-1}(v)$)
A	$\{A,B,C,D,E,F,G\}$	
B	$\{B,C,D,E,F,G\}$	
C	$\{C\}$	
D	$\{D\}$	
E	$\{E,F,G\}$	
F	$\{F\}$	
G	$\{G\}$	

v	DOM$^{-1}(v) - \{v\}$	DF(v)
A		
B		
C		
D		
E		
F		
G		

Dominance Frontier: Algorithm

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

where

$$Children(v) = \text{children of } v \text{ in dominator tree}$$
$$\text{DF}_{local}(v) = \{w \mid w \in SUCC(v) \wedge v\ \overline{\text{DOM!}}\ w\}$$
$$\text{DF}_{up}(w) = \text{the subset of DF}(w) \text{ not strictly dominated by IDOM}(w)\ (\text{IDOM}(w) = v)$$

proc *FindDF* (v) $\equiv$
  $\text{DF}(v) \leftarrow$ *empty*
  foreach $w \in Children(v)$ **do**
    FindDF (w)
    foreach $u \in \text{DF}(w)$ **do**
      if $v\ \overline{\text{DOM!}}\ u$ **then** $\text{DF}(v)$.add(u) **end**
    end
  end
  foreach $w \in SUCC(v)$ **do**
    if $v\ \overline{\text{DOM!}}\ w$ **then** $\text{DF}(v)$.add(w) **end**
  end.

Post-Dominators

Given CFG $= \langle V, E, Entry, Exit \rangle$, assume *Exit* reachable from all V:

$$\forall v \in V : v \rightarrow^* Exit$$

Post-Dominators

p post-dominates v, if all paths from v to *Exit* include p

- p PDOM $v \Rightarrow v \rightarrow^* Exit$ can be split into $v \rightarrow^* p$ and $p \rightarrow^* Exit$
- reflexive, antisymmetric, and transitive
- PDOM on CFG is the same as DOM on reverse CFG

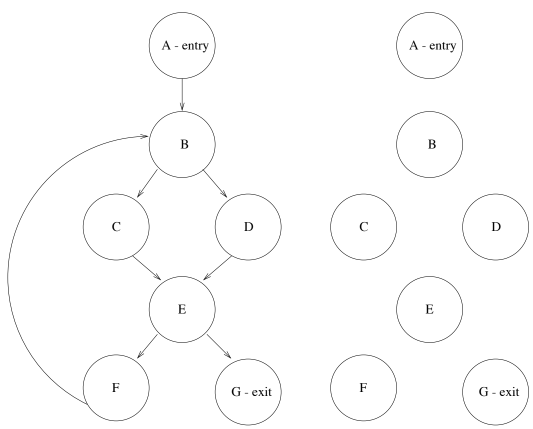
Strict Post-Dominators

- $\text{PDOM!}(v) = \text{PDOM}(v) - \{v\}$

Post-Dominance Frontier

- $\text{PDF}(v) = \{w \mid (\forall u \in SUCC(v))[w \text{ PDOM } u] \wedge (w\ \overline{\text{PDOM!}}\ v)$

Post-Dominators: Example



Control Dependence Graph (CDG)

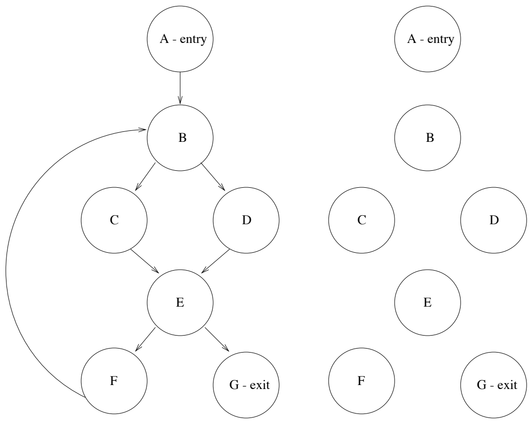
y is control dependent on x, x and y in CFG, *iff*

- $\exists\ x \rightarrow^* y$ where y post-dominates every vertex p in $x \rightarrow^* y$, $p \neq x$, and
- y does not strictly post-dominate x

$CDPRED(y) = \{x \mid y \text{ is control dependent on } x\}$
$CDSUCC(x) = \{y \mid y \text{ is control dependent on } x\}$

NB: add edge $Entry \rightarrow Exit$ in CFG

Control Dependence: Example



Next Time

Static Single Assignment

Cytron et al. *Efficiently Computing Static Single Assignment Form and the Control Dependence Graph*, *TOPLAS 13*(4):451–490, Oct 1991