Advanced Compilers

CMPSCI 710

Spring 2003

Dominators, etc.

Emery Berger

University of Massachusetts, Amherst

Dominators, etc.

- Last time
  - Live variable analysis
    - backwards problem
  - Constant propagation
    - algorithms
    - def-use chains
- Today
  - SSA-form
  - dominators

Def-Use Chains: Problem

```
switch (j)
  case x: i Ã 1; break;
  case y: i Ã 2; break;
  case z: i Ã 3; break;
switch (k)
  case x: a Ã i; break;
  case y: b Ã i; break;
  case z: c Ã i; break;
```

- worst-case size of graph = O(D*U) = $O(N^2)$

SSA Form

- **Static single assignment**
  - each assignment to variable gets unique name
  - all uses reached by that assignment are renamed
  - exactly one def per use
  - *sparse* program representation:
    use-def chain = (variable, [use_1, use_2…])

SSA Transformations

- New variable for each assignment, rename uses

```
v1 Ã 4
   Ã v1+5
v2 Ã 6
   Ã v2+7
```

- Easy for straight-line code, but what about control flow?

F -Functions

- At each join, add special assignment: "ϕ function":
  - operands indicate which assignments reach join
  - j_{th} operand = j_{th} predecessor
- If control reaches join from j_{th} predecessor, then value of ϕ(R,S,…) is value of j_{th} operand

SSA Transformation, F function

```
if P
  then v Ã 4
  else v Ã 6
/* use v */
```

```
if P
  then v1 Ã 4
  else v2 Ã 6
v3 Ã φ(v1, v2)
/* use v3 */
```


SSA Example II

```
v Ã 1
while (v < 10)
  v Ã v + 1
```

```
1: v Ã 1
2: if (v < 10)
3:   v Ã v + 1
4:   goto 2
```


SSA Example III

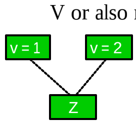
```
switch (j)
  case x: i Ã 1; break;
  case y: i Ã 2; break;
  case z: i Ã 3; break;
switch (k)
  case x: a Ã i; break;
  case y: b Ã i; break;
  case z: c Ã i; break;
```

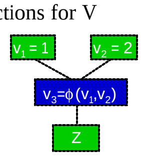

Placing F functions

- Safe to put ϕ functions for every variable at every join point
- But:
  - inefficient – not necessarily sparse!
  - loses information
- **Goal: minimal f nodes, subject to need**

F Function Requirement

- Node Z *needs f* function for V if:
  - Z is **convergence point** for two paths originating at different nodes
  - both originating nodes contain assignments to V or also need ϕ functions for V





Minimal Placement of F functions

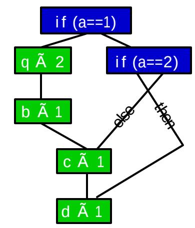
- Naïve computation of need is expensive: must examine all triples in graph
- Can be done in O(N) time
  - Relies on **dominance frontier** computation [Cytron et al., 1991]
  - Also can be used to compute **control dependence graph**

Control Dependence Graph

- Identifies conditions affecting statement execution
- Statement is **control dependent** on branch if:
  - one edge from branch *definitely* causes statement to execute
  - another edge *can* cause statement to be skipped

Control Dependence Example

```
if (a == 1)
  q Ã 2
else
  if (a == 2)
    goto B
  goto A
b Ã 1
A: c Ã 1
B: d Ã 1
```



Dominators

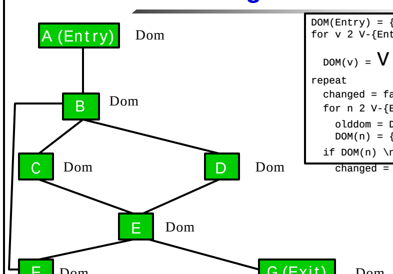
- Before we do dominance frontiers, we need to discuss other dominance relationships
- x **dominates** y (x dom y)
  - in CFG, all paths to y go through x
  - **Dom(v)** = set of all vertices that dominate v
  - *Entry* dominates every vertex
    - Reflexive: *a dom a*
    - Transitive: *a dom b, b dom c ! a dom c*
    - Antisymmetric *a dom b, b dom a ! b=a*
- **Notice: in SSA form, a definition dominates its use**

Finding Dominators

- *Dom(v) = {v}* Å$_{p\ 2\ PRED(v)}$ *Dom(p)*
- Algorithm:

```
DOM(Entry) = {Entry}
for v 2 V-{Entry}
  DOM(v) = V
repeat
  changed = false
  for n 2 V-{Entry}
    olddom = DOM(n)
    DOM(n) = {n} [ (Åp 2 PRED(n) DOM(p))
  if DOM(n) ≠ olddom
    changed = true
```


Dominator Algorithm Example



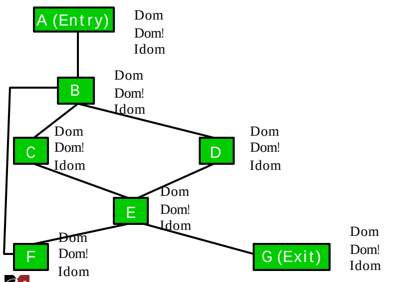

```
DOM(Entry) = {Entry}
for v 2 V-{Entry}

  DOM(v) = V
repeat
  changed = false
  for n 2 V-{Entry}
    olddom = DOM(n)
    DOM(n) = {n} [ (Åp 2 PRED(n) DOM(p))
  if DOM(n) \neq olddom
    changed = true
```

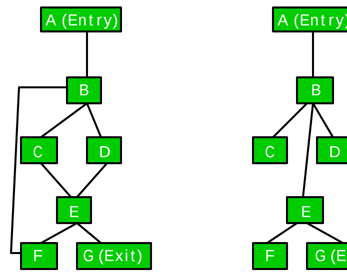

Other Dominators

- **Strict dominators**
  - *Dom!(v) = Dom(v)* – {v}
  - antisymmetric & transitive
- **Immediate dominator**
  - *Idom(v)* = closest strict dominator of v
  - d *Idom* v if
    d *Dom!* v and 8w 2 *Dom!(v)*: w *Dom* d
  - antisymmetric
  - *Idom* induces tree

Dominator Example



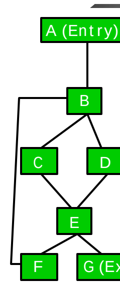
Dominator Tree



Inverse Dominators

- $Dom^{-1}(v)$ = set of all vertices dominated by v
- reflexive, antisymmetric, transitive

Inverse Dominator Example



- *Dom(A):* {A}
  Dom^{-1}(A):
- *Dom(B):* {A,B}
  Dom^{-1}(B):
- *Dom(C):* {A,B,C}
  Dom^{-1}(C):
- *Dom(D):* {A,B,D}
  Dom^{-1}(D):
- *Dom(E):* {A,B,E}
  Dom^{-1}(E):
- *Dom(F):* {A,B,E,F}
  Dom^{-1}(F):
- *Dom(G):* {A,B,E,G}
  Dom^{-1}(G):

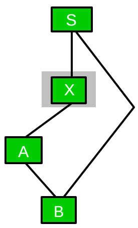
Finally: Dominance Frontiers!

- The **dominance frontier** DF(X) is set of all nodes Y such that:
  - X dominates a predecessor of Y
  - But X does not strictly dominate Y
    - $DF(X) = \{Y \mid (\exists P \in PRED(Y): X\ Dom\ P) \wedge \neg\ X\ Dom!\ Y)\}$

Why Dominance Frontiers

- **Dominance frontier criterion:**
  - **if node x contains def of a, then any node z in DF(x) *needs* a ϕ function for a**
- intuition:
  at least two non-intersecting paths converge to z, and one path must contain node strictly dominated by x

Dominance Frontier Example

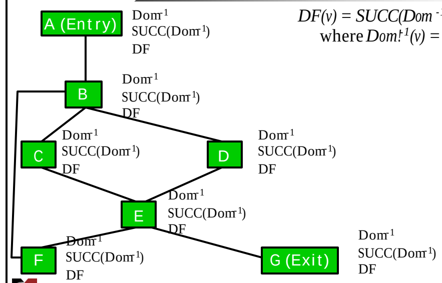


node y is in dominance frontier of node x if: x dominates predecessor of y but does not strictly dominate y

DF(X) = {Y|(9 P 2 PRED(Y): X Dom P) Æ q X Dom! Y)

Dominance Frontier Example

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



Next Time

- Computing dominance frontiers
- Computing SSA form