Advanced Compilers

CMPSCI 710

Spring 2003

More data flow analysis

Emery Berger

University of Massachusetts, Amherst

More Data Flow Analysis

- **Last time**
  - Program points
  - Lattices
  - Max fixed point
  - Reaching definitions
- **Today**
  - Iterative Worklist Algorithm
    - actual algorithm, examples

From Last Time: Informal Description

- Define **lattice** to represent facts
- Attach meaning to lattice values
- Associate **transfer function** to each node
- Initialize values at each program point
- Iterate through program until **fixed point**

Iterative Worklist Algorithm

```
for v 2 V
  IN(v) = ∅
  OUT(v) = Gen(v)
worklist Ã V
while (worklist ≠ ∅)
  v Ã remove(worklist)
  oldout(v) = OUT(v)
  IN(v) = [p 2 PRED(v) OUT(p)
  OUT(v) = GEN(v) [ (IN(v) – KILL(v))
  if (oldout(v) ≠ OUT(v))
    worklist Ã worklist [ SUCC(v)
```


Iterative Worklist Algorithm: Analysis

- Worst-case runtime
  - Visit each basic block
    - up to |N|
    - compute successors
    - perform set operations (bit vectors)
  - Can we bound number of passes?

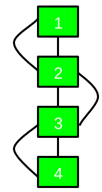
Bounding Expected Runtime

- Order matters:
  visit nodes in **reverse postorder**:
  - Nodes visited roughly after its predecessors
  - Intuition: accumulate as much info as possible before processing each node

Reverse Postorder

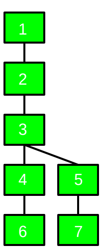
```
visit(n):
  visited(n) Ã true
  for s 2 SUCC(n)
    if not visited(s)
      visit(s)
  postorder(n) Ã count
  count Ã count + 1

count Ã 1
for each node n
  visited(n) Ã false
visit (entry)
for each node n
  rPostorder(n) Ã NumNodes – postorder(n)
```


Reverse Postorder: Examples



visited(1) = postorder(1) =
visited(2) = postorder(2) =
visited(3) = postorder(3) =
visited(4) = postorder(4) =

Reverse Postorder: Examples

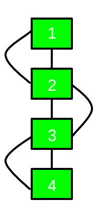


visited(1) = postorder(1) =
visited(2) = postorder(2) =
visited(3) = postorder(3) =
visited(4) = postorder(4) =
visited(5) = postorder(5) =
visited(6) = postorder(6) =
visited(7) = postorder(7) =

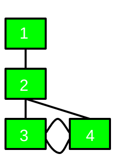
Loop Interconnectiveness

- Defined as d(G) = maximum number of back edges on any acyclic path in graph G
  - up to |N|
  - but usually · 3 and often 1
- d(G) = 1 for reducible flow graphs

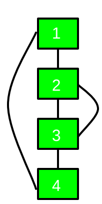
Loop Interconnectiveness: Examples



d(G) =



d(G) =

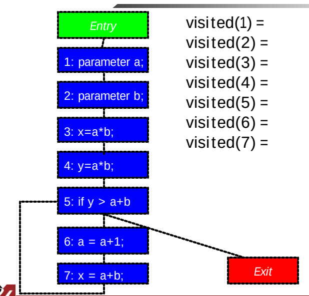


d(G) =

Iterative Worklist Algorithm, Modified

```
for v 2 V
  IN(v) = ∅
  OUT(v) = Gen(v)
worklist Ã rPostorder(V), changed Ã true
while (changed)
  changed Ã false
  for v 2 worklist
    oldout(v) = OUT(v)
    IN(v) = up 2 PRED(v) OUT(p)
    OUT(v) = GEN(v) u (IN(v) – KILL(v))
    if (oldout(v) ≠ OUT(v))
      changed Ã true
```

Reaching Definitions Example



visited(1) =
visited(2) =
visited(3) =
visited(4) =
visited(5) =
visited(6) =
visited(7) =

postorder(1) =
postorder(2) =
postorder(3) =
postorder(4) =
postorder(5) =
postorder(6) =
postorder(7) =

For Reaching Definitions

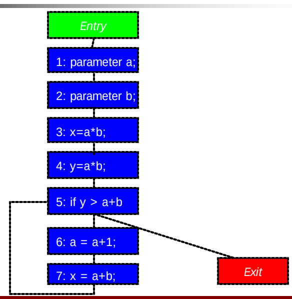
- For reaching defs, u = [
- *Gen(d: v = exp)* = {d}
  - "on exit from block d, generate new definition"
- *Kill(d: v = exp)* = *defs(v)*
  - "on exit from block d, definitions of v are killed"
- Computing *In(S)* and *Out(S)*
  - *In(S)* = [$_{P \text{ in PRED(S)}}$ *Out(P)*
  - *Out(S)* = *Gen(v)* [(*In(v)* – *Kill(v)*)
  - *Out(Entry)* = { }

Reaching Definitions Example

defs(x) = {3, 7}
defs(y) = {4}
defs(a) = {1, 6}
defs(b) = {2}

reverse postorder =
{1, 2, 3, 4, 5, 6, 7}

changed =



Iterative Worklist Algorithm: Revised Analysis

- Stabilizes in at most d(G) + 2 iterations
  - d(G) + 1 iterations to propagate data
  - 1 iteration to detect stability
  - as noted, d(G) usually · 3, often 1
- Each pass computes:
  - O(E) meets (sets of size | *defs* |)
  - O(N) other operations
- Effectively O(N) complexity
- *Note:* for backwards analysis, use postorder

Other Data Flow Problems

- Definitely uninitialized variables
  - *Gen(S)* =
  - *Kill(S)* =
  - *Out(Entry)* =
  - u =
- Possibly uninitialized variables
  - *Gen(S)* =
  - *Kill(S)* =
  - *Out(Entry)* =
  - u =

```
a = 3
b = 5
if (a == 2)
  c = 1
else
  b = 2
```


Next Time – Even More Data Flow!

- Live variable analysis
  - backwards problem
- Constant propagation
- Supplementary paper available:
  - Wegman & Zadeck, TOPLAS 1991