Advanced Compilers

CMPSCI 710

Spring 2003

Data flow analysis

Emery Berger

University of Massachusetts, Amherst

Data flow analysis

- Framework for proving facts about program at each **point**
  - Point: entry or exit from block (or CFG edge)
  - Lots of "small facts"
  - Little or no interaction between facts
- Based on **all paths** through program
  - Includes **infeasible** paths

Infeasible Paths Example

```
a = 1;
if (a == 0) {
  a = 1;
}
if (a == 0) {
  a = 2;
}
```

- Infeasible paths never actually taken by program, regardless of input
- Undecidable to distinguish from feasible

Data Flow-Based Optimizations

- **Dead variable elimination**
  - a = 3; *print* a; x = 12; *halt*) a = 3; print a; *halt*
- **Copy propagation**
  - x = y; … *use of* x) …*use of* y
- **Partial redundancy**
  - a = 3*c + d; b = 3*c) b = 3*c; a=b+d
- **Constant propagation**
  - a = 3; b = 2; c = a+b) a = 3; b = 2; c = 5

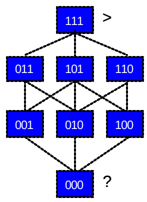
Data Flow Analysis

- Define **lattice** to represent facts
- Attach meaning to lattice values
- Associate **transfer function** to each node
  - (f:L ! L)
- Initialize values at each program point
- Iterate through program until **fixed point**

Lattice-Related Definitions

- **Meet function**: u
  - commutative and associative
  - x u x = x
- Unique **bottom** ? and **top** > element
  - x u ? = ?
  - x u > = x
- Ordering: x v y *iff* x u y = x
- Function f is **monotone** if 8 x, y:
  - x v y implies f(x) v f(y)

Bit-Vector Lattice



111 u 111 =
011 u 111 =
000 u 101 =
100 u 011 =
011 u 110 =
001 u 110 =

- Meet = bit-vector logical and

Constant Propagation Lattice

>
… -2 -1 0 1 2 …
?

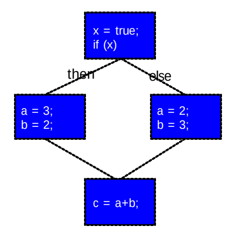
- Meet rules:
  - a u > = a
  - a u ? = ?
  - constant u constant = constant *(if equal)*
  - constant u constant = ? *(if not equal)*
- Define obvious transfer functions for arithmetic

Iterative Data Flow Analysis

- Initialize non-entry nodes to >
  - Identity element for meet function
- If node function is monotone:
  - Each re-evaluation of node moves *down* the lattice, if it moves at all
- If height of lattice is finite, must terminate

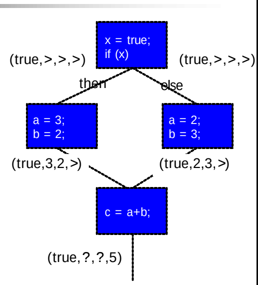
Constant Propagation Example I

- Two choices of "point":
  - Compute at edges
    - maximal information
  - Compute on entry
    - must "meet" data from all incident edges
    - loses information



Constant Propagation Example II

- Vector for (x,a,b,c)
- Init values to >
- Iterate forwards



Accuracy: MOP vs. MFP

- We *want* "meet-over-all-paths" solution, but paths can be infinite if there are loops
- Best we can do in general:
  - **Maximum Fixed Point** solution = largest solution, ordered by v, that is fixed point of iterative computation
  - Provides "most information"
  - More conservative than MOP

Distributive Problems

- f is **distributive** *iff*
  - f(x u y) = f(x) u f(y)
  - Doing meet early doesn't reduce precision
- Non-distributive problems:
  - constant propagation
- Distributive problems:
  - MFP = MOP
  - reaching definitions, live variables

Reaching Definitions

- **Definition**: each assignment to variable
- *defs(v)* represents set of all definitions of *v*
- Assume all variables scalars
  - No pointers
  - No arrays
- A definition **reaches** given point if 9 path to that point such that variable *may* have value from definition

Data Flow Functions

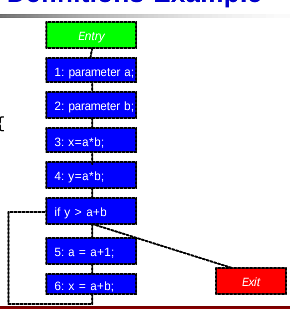
- *Kill(S)*: facts not true after *S* just because they were true before
  - *example*: redefinition of variable (assignment)
- *Gen(S)*: facts true after *S* regardless of facts true before *S*
  - *example*: assigned values not killed in *S*
- *In(S)*: dataflow info on entry to *S*
  - *In(S)* = [$_{p\ 2\ \mathrm{PRED}(S)}$ *Out*(p)
  - *example*: definitions that reach *S*
- *Out(S)*: dataflow info on exit from *S*
  - *Out(S)* = *Gen*(S) [*(In(S) – Kill(S))*
  - *example*: reaching defs after *S*

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)*
  - If *S* has one predecessor *P*, *In(S)* = *Out(P)*
  - Otherwise: *In(S)* = u$_{P\ \mathrm{in}\ \mathrm{PRED}(S)}$*Out(P)*
  - *Out(Entry)* = { }

Reaching Definitions Example

```
parameter a;
parameter b;
x = a*b;
y = a*b;
while (y > a+b) {
   a = a+1;
   x = a+b;
}
```



defs(x) =
defs(y) =
defs(a) =
defs(b) =

Analysis Direction

- **Forwards analysis**:
  - start with Entry, compute towards Exit
- **Backwards analysis**:
  - start with Exit, compute towards Entry
  - *In(S)* = *Gen(S)* [*(Out(S) – Kill(S))*
  - *Out(S)* = u$_{F\ in\ SUCC(S)}$ *In(F)*
- Backwards problems:
  - Live variables: which variables might be read before overwritten or discarded

Next Time

- More data flow analysis