Data Flow Analysis

Chen Ding

CS255/455 Advanced Programming Systems
Spring 2009

Global Analysis

- What do we analyze?
  - names, values, or locations?
- What is different from local or super-local analysis?
  - merge points
    - why are they needed?
    - why are they difficult?
  - two examples

Global Redundancy Elimination (GRE)

- Expression and program point
  - An expression is a computation that produces a new value
  - A program point can be the point before or after a statement and before or after a basic block
- Expression definition, kill, and availability
  - An expression e is defined at point p in CFG if its value is computed at p
  - Expression e is killed at point p if one of its operands is redefined at p
  - Expression e is available at p if every path leading to p has a prior definition and e is not killed in between

```
a = b - c
...
b = a + d
...
c = b - c
...
d = a + d
```


Meet-over-all-path Solutions (MOP)

- Control flow graph
  - any block may reach any other block
    - finite graph
    - infinite (and infinitely long) paths
  - may provide a value
  - may kill a variable
- What can we call truth?
  - meet-over-all-path (MOP)
    - invariant properties in all paths

Data Flow Analysis

- Data flow analysis
  - solving "a set of equations, posed over a graphical representation of the code to discover facts about what can occur when program runs"
- Three steps
  - build the control flow graph
  - gather local information
  - solve iteratively

AVAIL Analysis (Step 2)

- DEExpr(n): expressions defined but not killed in block n
- ExprKill(n): expressions in the procedure killed in block n

```
VarKill ← ∅
DEExpr(n) ← ∅
for i = k to 1
    assume operation o_i is "x ← y op z"
    if (y ∉ VarKill) and (z ∉ VarKill)
      then add "y + z" to DEExpr(n)
    VarKill ← VarKill ∪ {x}
ExprKill(n) ← ∅
for each expression e in the procedure
    for each variable v ∈ e
      if v ∈ VarKill
        then ExprKill(n) ← ExprKill(n) ∪ {e}
```

A:
```
e = b + 18
...
u = e + f
```

B:
```
u = e + f
...
u = b
```


source: *EAC figure 8.8*

AVAIL Analysis

- Local sets
  - DEExpr(n): exprs defined but not killed in block n
  - ExprKill(n): exprs killed in block n
- MOP sets
  - AVAIL(n): exprs available at the start of block n
    - defined at or before all its predecessors
    - not killed after the definition

★ Is e+f available?

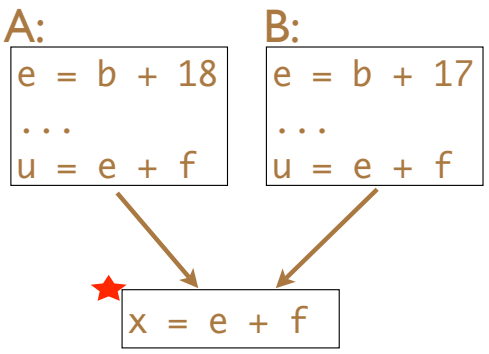
DEExpr(A) =

ExprKill(A) =

DEExpr(B) =

ExprKill(B) =

AVAIL(★) =



★ Is e+f available?

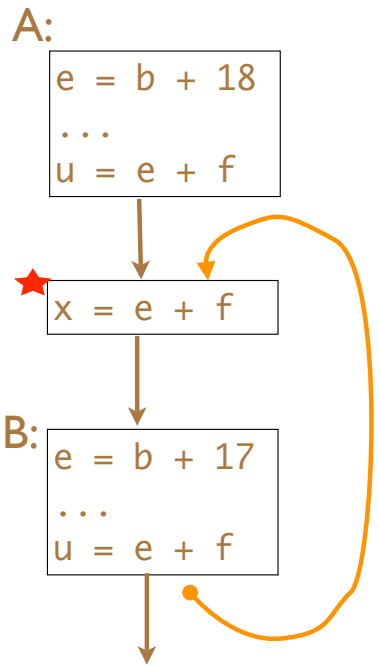
DEExpr(A) =

ExprKill(A) =

DEExpr(B) =

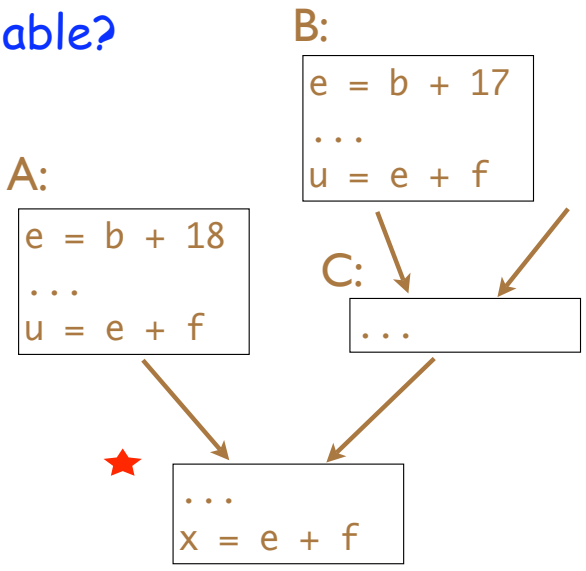
ExprKill(B) =

AVAIL(★) =



★ Is e+f available?

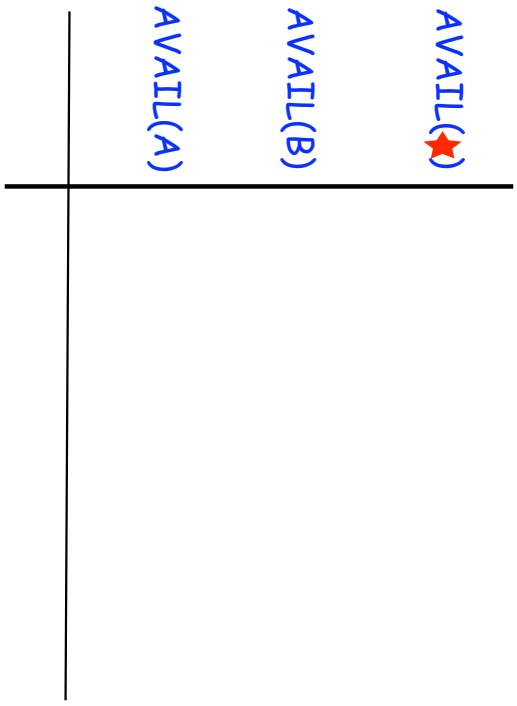
AVAIL(★) =



Solving Recursive Equations

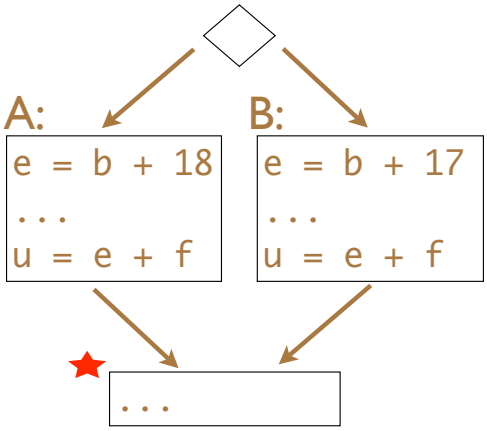
- Example
  - solve $\cos(x) = x$ for x
- Solution?
- Any problems with the solution?
  - hint: at least three

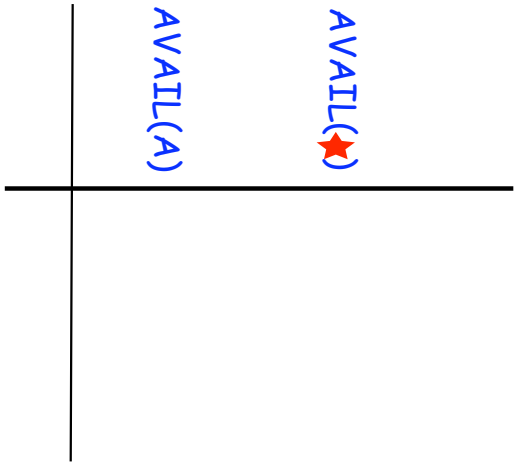
AVAIL Algorithm

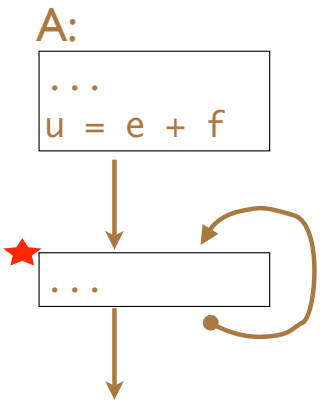


DEExpr(A) DEExpr(B)

ExprKill(A) ExprKill(B)







- How many choices for initial AVAIL(n)?
- What are the implications for the resulting fixed point?