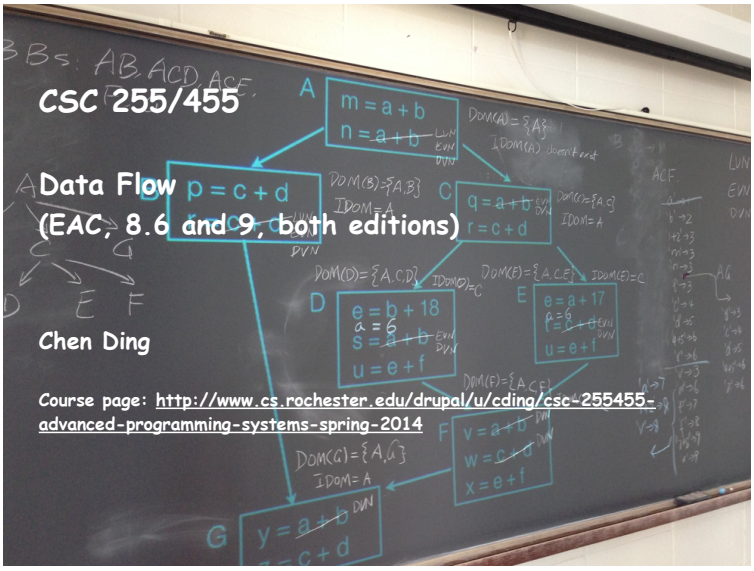


Problems of Value Numbering

- Merging values
  - Inconsistent values
    - value numbers differ in different blocks
  - Redundant computation
    - but not the same value
- Computing the dominator sets

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
- Assumptions
  - different names, different variable
  - single name space

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 (but before) 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 Framework

- 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"
  - solving all path problems of a graph
    - "truth"
- 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 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}

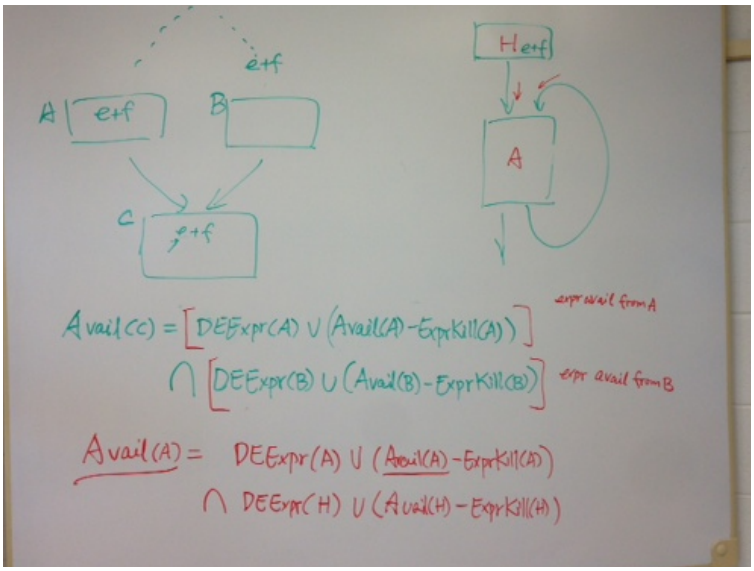
A:
```
e = b + 18
...
u = e + f
```

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


source: *EAC figure 8.8*

AVAIL Analysis

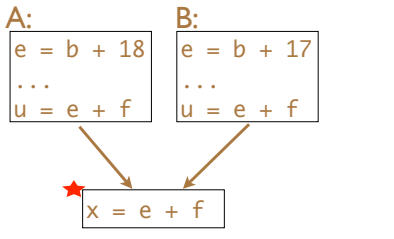
- Local sets
  - DEExpr(n): exprs defined but not killed in block n
    - downward exposed expressions
  - ExprKill(n): exprs killed in block n
- MOP sets
  - AVAIL(n): exprs available at the start of block n
    - defined at or before all its predecessors
    - not killed after the definition

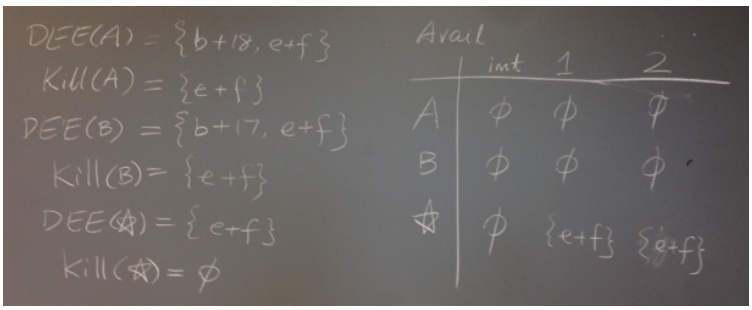


Solving Recursive Equations

- Example
  - solve cos(x) = x for x
- Solution?
- Problems?
  - hint: at least three

```
>> def solve(x)
>>   y = Math.cos(x)
>>   return x if (x-y).abs < 0.0001
>>   return solve(y)
>> end
>> z = solve(1)
=> 0.739130176529671
>> Math.cos(z)
=> 0.739054790746917
```





Review Questions

- What is the purpose of data flow analysis?
  - why is it called a framework?
- What are the three steps?
- What are the local sets computed for AVAIL?
- What is the data flow equation for AVAIL?
  - can you prove correctness and completeness?
- Can you draw the two basic cases of control flow merge?