EECS 583 – Class 6
Dataflow Analysis

University of Michigan

September 26, 2011

Announcements & Reading Material

- Today's class
  - *Compilers: Principles, Techniques, and Tools*,
    A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988.
    (Sections: 10.5, 10.6, 10.9, 10.10)
- Material for Wednesday
  - *Compilers: Principles, Techniques, and Tools*,
    A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988,
    9.9, 10.2, 10.3, 10.7

Live Variable (Liveness) Analysis

- Defn: For each point p in a program and each variable y, determine whether y can be used before being redefined starting at p
- Algorithm sketch
  - For each BB, y is live if it is used before defined in the BB or it is live leaving the block
  - Backward dataflow analysis as propagation occurs from uses upwards to defs
- 4 sets
  - GEN = set of external variables consumed in the BB
  - KILL = set of external variable uses killed by the BB
    - equivalent to set of variables defined by the BB
  - IN = set of variables that are live at the entry point of a BB
  - OUT = set of variables that are live at the exit point of a BB

Computing GEN/KILL Sets For Each BB

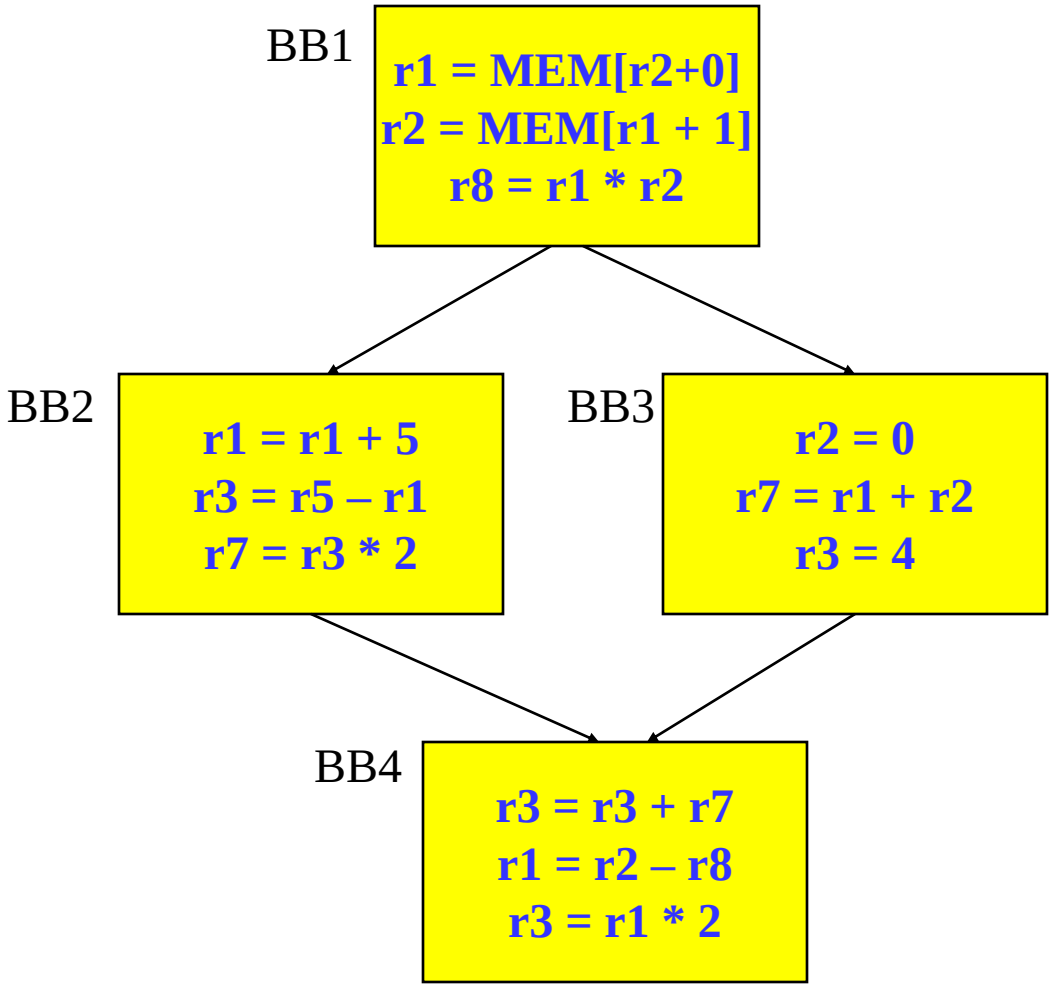
```
for each basic block in the procedure, X, do
    GEN(X) = 0
    KILL(X) = 0
    for each operation in reverse sequential order in X, op, do
        for each destination operand of op, dest, do
            GEN(X) -= dest
            KILL(X)  += dest
        endfor
        for each source operand of op, src, do
            GEN(X) += src
            KILL(X) -= src
        endfor
    endfor
endfor
```

Compute IN/OUT Sets for all BBs

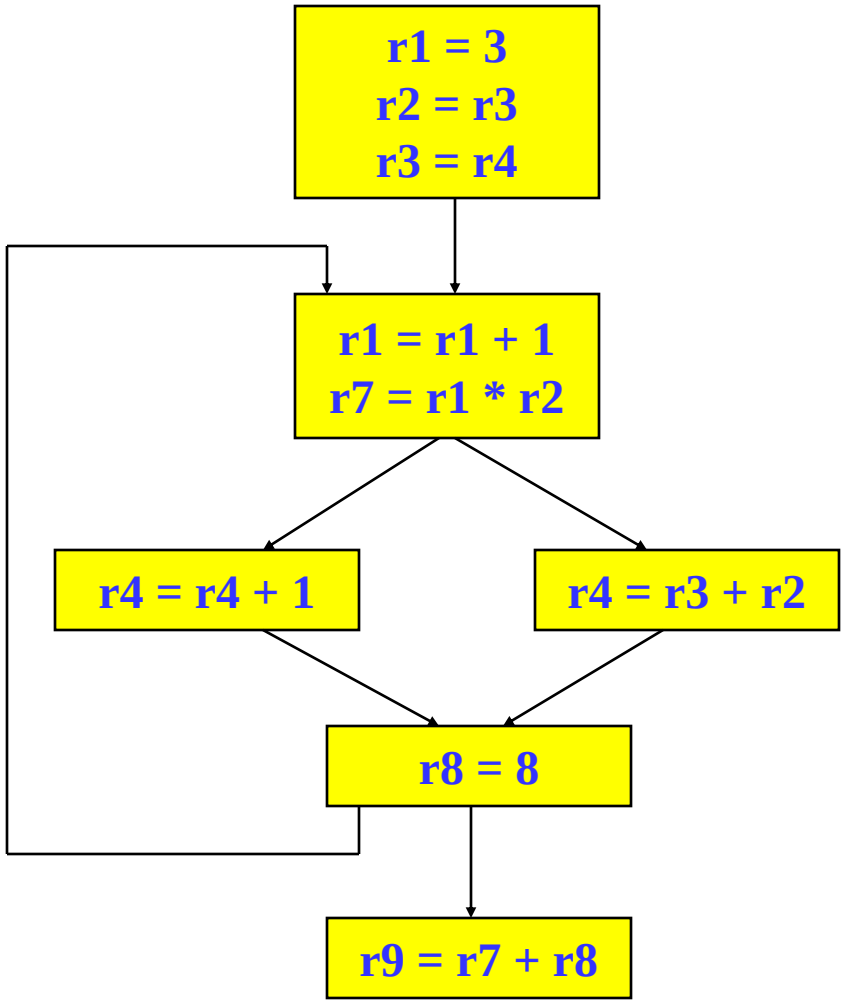
```
initialize IN(X) to 0 for all basic blocks X
change = 1
while (change) do
    change = 0
    for each basic block in procedure, X, do
        old_IN = IN(X)
        OUT(X) = Union(IN(Y)) for all successors Y of X
        IN(X) = GEN(X) + (OUT(X) – KILL(X))
        if (old_IN != IN(X)) then
            change = 1
        endif
    endfor
endfor
```

Example – Liveness Computation

OUT = Union(IN(succs))
IN = GEN + (OUT – KILL)



Class Problem



Compute liveness

- Calculate GEN/KILL for each BB
- Calculate IN/OUT for each BB

Reaching Definition Analysis (rdefs)

- A definition of a variable x is an operation that assigns, or may assign, a value to x
- A definition d reaches a point p if there is a path from the point immediately following d to p such that d is not "killed" along that path
- A definition of a variable is killed between 2 points when there is another definition of that variable along the path
  - r1 = r2 + r3 kills previous definitions of r1
- Liveness vs Reaching defs
  - Liveness → variables (e.g., virtual registers), don't care about specific users
  - Reaching defs → operations, each def is different
  - Forward dataflow analysis as propagation occurs from defs downwards (liveness was backward analysis)

Compute Rdef GEN/KILL Sets for each BB

GEN = set of definitions created by an operation
KILL = set of definitions destroyed by an operation
- Assume each operation only has 1 destination for simplicity
  so just keep track of "ops"..

```
for each basic block in the procedure, X, do
    GEN(X) = 0
    KILL(X) = 0
    for each operation in sequential order in X, op, do
        for each destination operand of op, dest, do
            G = op
            K = {all ops which define dest – op}
            GEN(X) = G + (GEN(X) – K)
            KILL(X) = K + (KILL(X) – G)
        endfor
    endfor
endfor
```

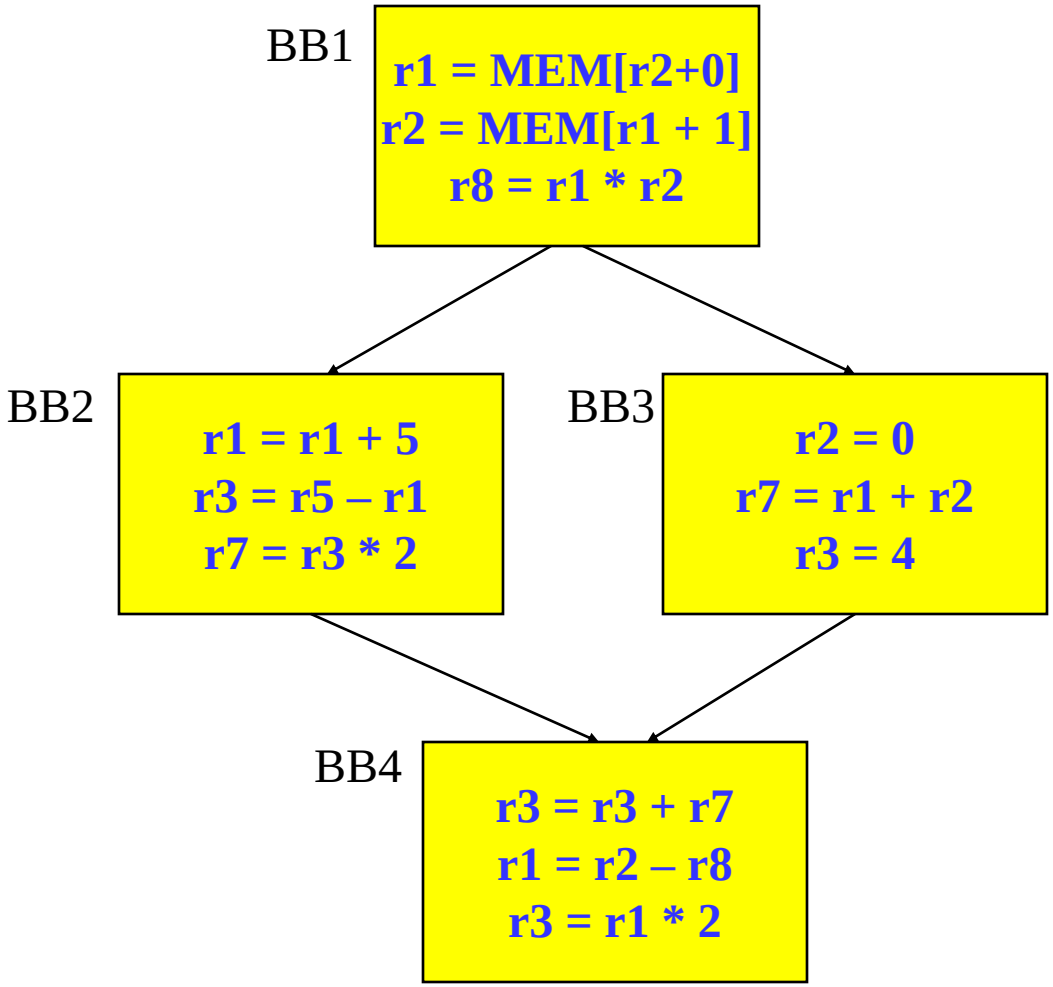
Compute Rdef IN/OUT Sets for all BBs

IN = set of definitions reaching the entry of BB
OUT = set of definitions leaving BB

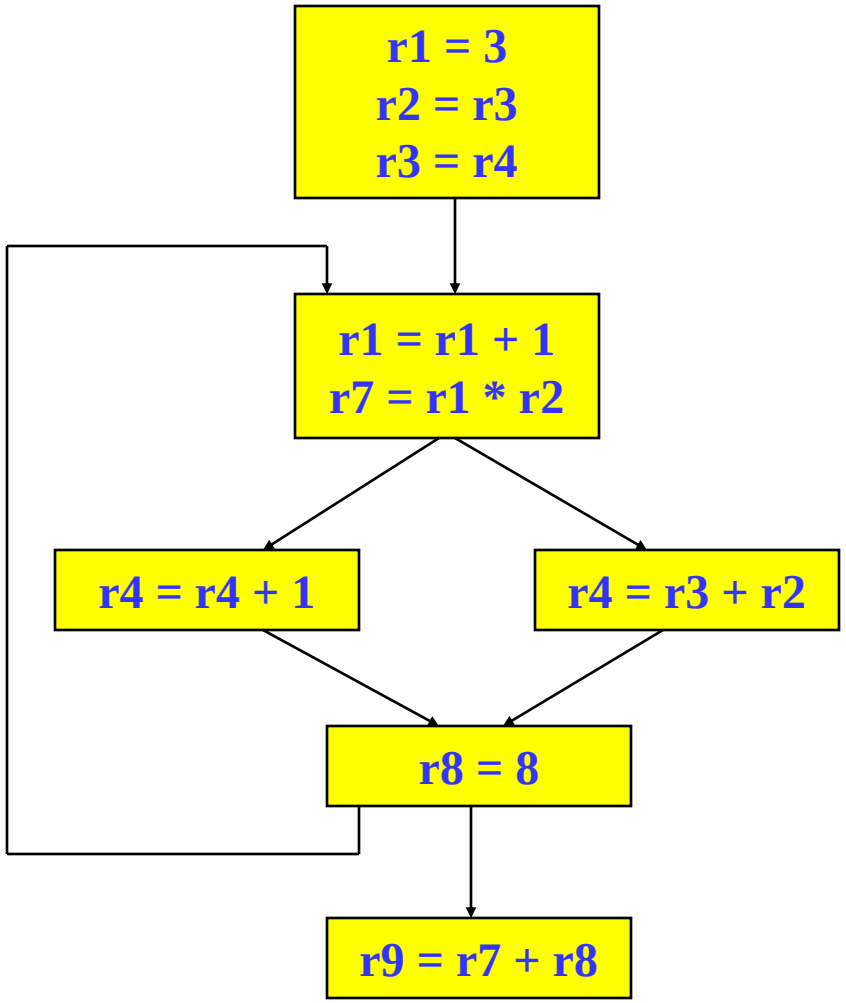
```
initialize IN(X) = 0 for all basic blocks X
initialize OUT(X) = GEN(X) for all basic blocks X
change = 1
while (change) do
    change = 0
    for each basic block in procedure, X, do
        old_OUT = OUT(X)
        IN(X) = Union(OUT(Y)) for all predecessors Y of X
        OUT(X) = GEN(X) + (IN(X) – KILL(X))
        if (old_OUT != OUT(X)) then
            change = 1
        endif
    endfor
endfor
```

Example Rdef Calculation

IN = Union(OUT(preds))
OUT = GEN + (IN – KILL)



Class Problem



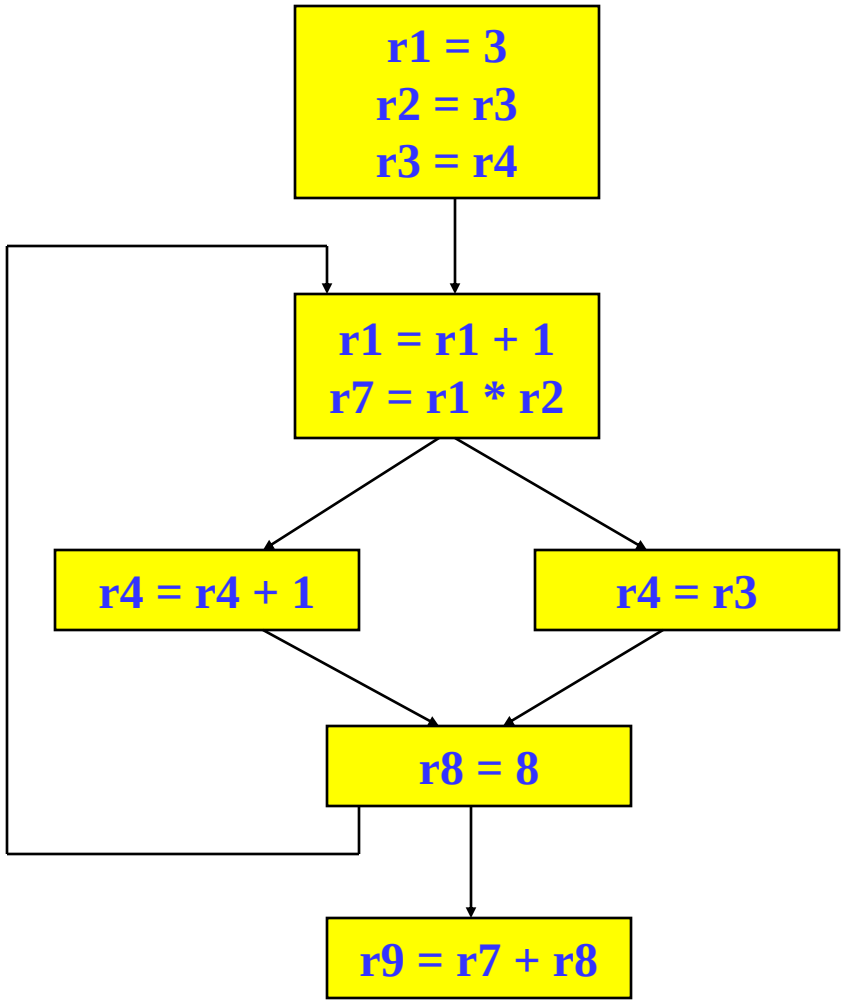
Compute reaching defs

- Calculate GEN/KILL for each BB
- Calculate IN/OUT for each BB

DU/UD Chains

- Convenient way to access/use reaching defs info
- Def-Use chains
  - Given a def, what are all the possible consumers of the operand produced
  - Maybe consumer
- Use-Def chains
  - Given a use, what are all the possible producers of the operand consumed
  - Maybe producer

Example – DU/UD Chains



Generalizing Dataflow Analysis

- Transfer function
  - How information is changed by "something" (BB)
  - OUT = GEN + (IN – KILL) /* forward analysis */
  - IN = GEN + (OUT – KILL) /* backward analysis */
- Meet function
  - How information from multiple paths is combined
  - IN = Union(OUT(predecessors)) /* forward analysis */
  - OUT = Union(IN(successors)) /* backward analysis */
- Generalized dataflow algorithm
  - while (change)
    - change = false
    - for each BB
      - apply meet function
      - apply transfer functions
      - if any changes → change = true

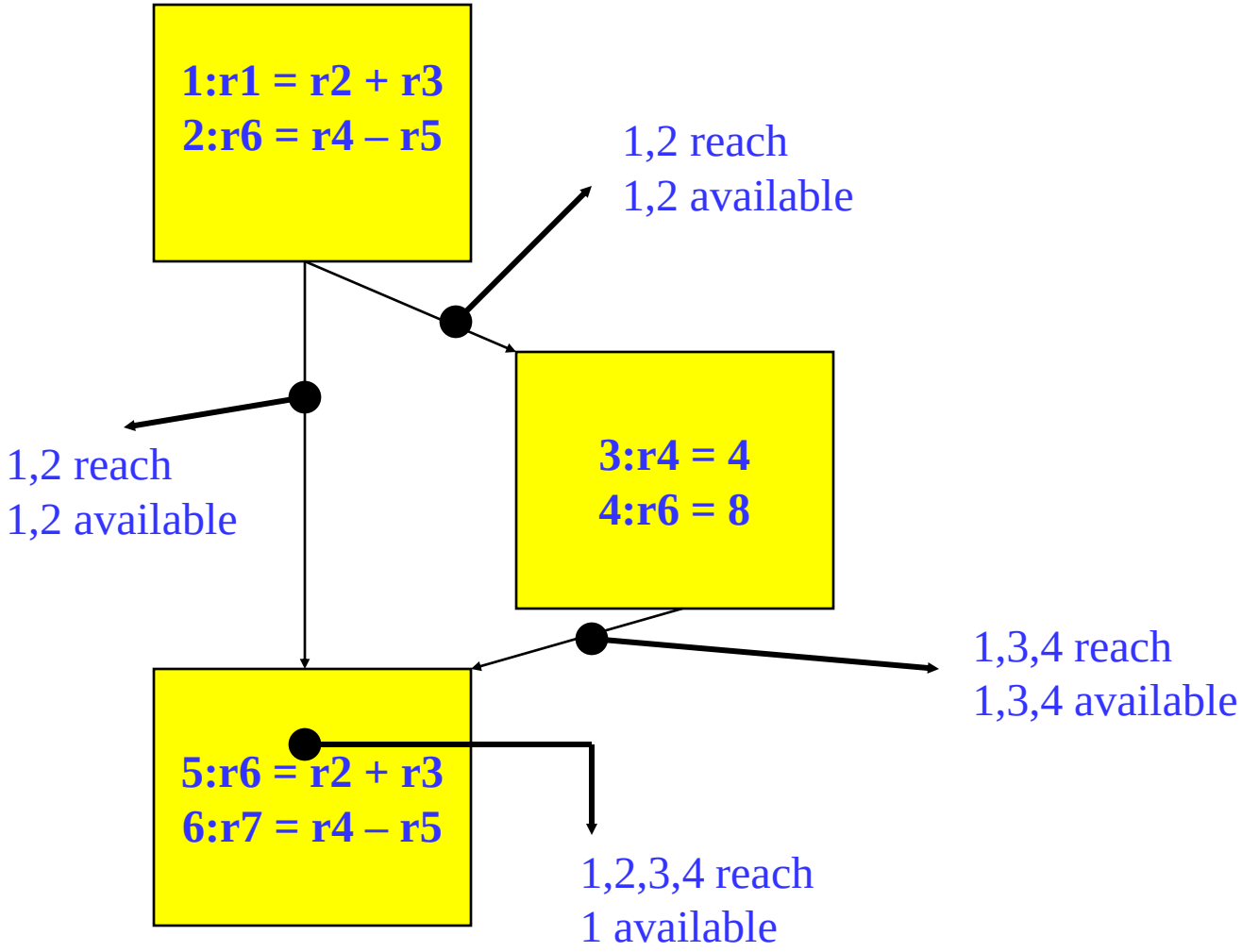
Beyond Liveness or Upward Exposed Uses

- Upward exposed defs
  - IN = GEN + (OUT – KILL)
  - OUT = Union(IN(successors))
  - Walk ops reverse order
    - GEN += dest; KILL += dest
- Downward exposed uses
  - IN = Union(OUT(predecessors))
  - OUT = GEN + (IN-KILL)
  - Walk ops forward order
    - GEN += src; KILL -= src;
    - GEN -= dest; KILL += dest;
- Downward exposed defs
  - IN = Union(OUT(predecessors))
  - OUT = GEN + (IN-KILL)
  - Walk ops forward order
    - GEN += dest; KILL += dest;

What About All Path Problems?

- Up to this point
  - Any path problems (maybe relations)
    - Definition reaches along some path
    - Some sequence of branches in which def reaches
    - Lots of defs of the same variable may reach a point
  - Use of Union operator in meet function
- All-path: Definition guaranteed to reach
  - Regardless of sequence of branches taken, def reaches
  - Can always count on this
  - Only 1 def can be guaranteed to reach
  - Availability (as opposed to reaching)
    - Available definitions
    - Available expressions (could also have reaching expressions, but not that useful)

Reaching vs Available Definitions



Available Definition Analysis (Adefs)

- A definition d is available at a point p if along all paths from d to p, d is not killed
- Remember, a definition of a variable is killed between 2 points when there is another definition of that variable along the path
  - r1 = r2 + r3 kills previous definitions of r1
- Algorithm
  - Forward dataflow analysis as propagation occurs from defs downwards
  - Use the Intersect function as the meet operator to guarantee the all-path requirement
  - GEN/KILL/IN/OUT similar to reaching defs
    - Initialization of IN/OUT is the tricky part

Compute GEN/KILL Sets for each BB (Adefs)

Exactly the same as reaching defs !!!

```
for each basic block in the procedure, X, do
   GEN(X) = 0
   KILL(X) = 0
   for each operation in sequential order in X, op, do
      for each destination operand of op, dest, do
         G = op
         K = {all ops which define dest – op}
         GEN(X) = G + (GEN(X) – K)
         KILL(X) = K + (KILL(X) – G)
      endfor
   endfor
endfor
```

Compute IN/OUT Sets for all BBs (Adefs)

```
U = universal set of all operations in the Procedure
IN(0) = 0
OUT(0) = GEN(0)
for each basic block in procedure, W, (W != 0), do
   IN(W) = 0
   OUT(W) = U – KILL(W)

change = 1
while (change) do
   change = 0
   for each basic block in procedure, X, do
      old_OUT = OUT(X)
      IN(X) = Intersect(OUT(Y)) for all predecessors Y of X
      OUT(X) = GEN(X) + (IN(X) – KILL(X))
      if (old_OUT != OUT(X)) then
         change = 1
      endif
   endfor
endfor
```

Available Expression Analysis (Aexprs)

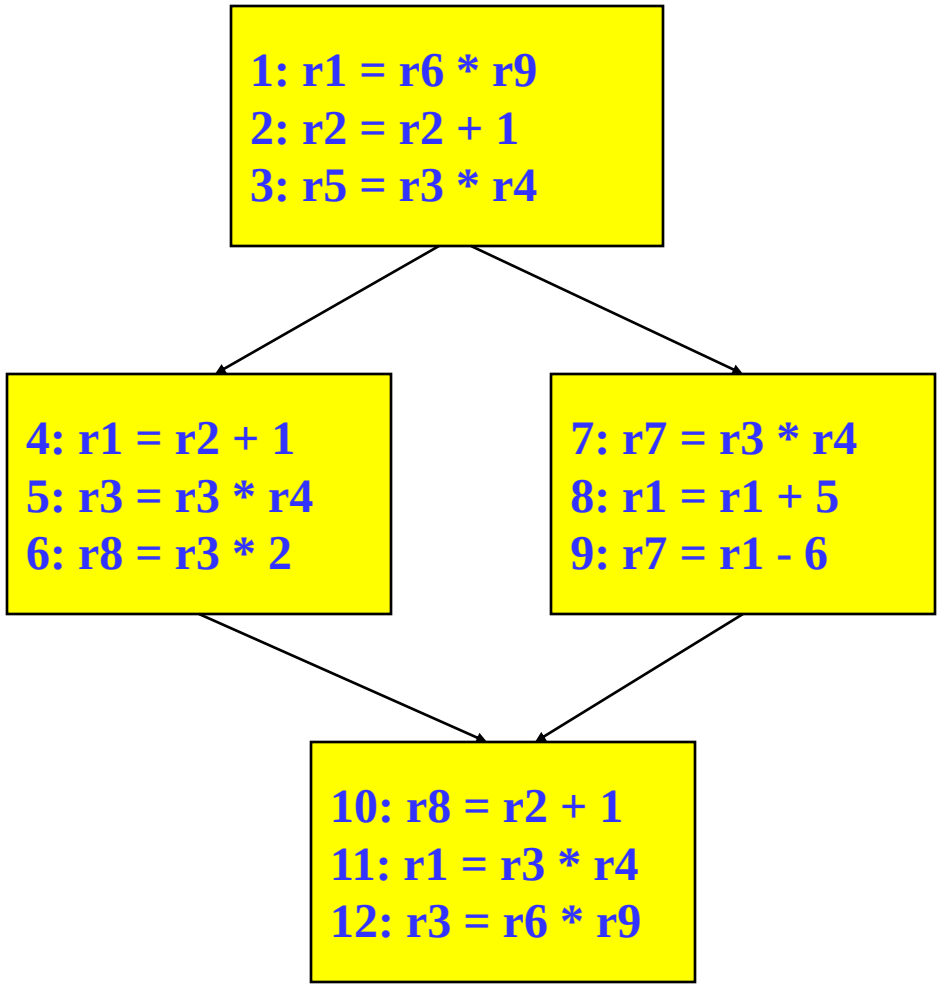
- An expression is a RHS of an operation
  - r2 = r3 + r4, r3+r4 is an expression
- An expression e is available at a point p if along all paths from e to p, e is not killed
- An expression is killed between 2 points when one of its source operands are redefined
  - r1 = r2 + r3 kills all expressions involving r1
- Algorithm
  - Forward dataflow analysis as propagation occurs from defs downwards
  - Use the Intersect function as the meet operator to guarantee the all-path requirement
  - Looks exactly like adefs, except GEN/KILL/IN/OUT are the RHS's of operations rather than the LHS's

Computation of Aexpr GEN/KILL Sets

We can also formulate the GEN/KILL slightly differently so you do not need to break up instructions like "r2 = r2 + 1".

```
for each basic block in the procedure, X, do
   GEN(X) = 0
   KILL(X) = 0
   for each operation in sequential order in X, op, do
      K = 0
      for each destination operand of op, dest, do
         K += {all ops which use dest}
      endfor
      if (op not in K)
            G = op
      else
            G = 0
      GEN(X) = G + (GEN(X) – K)
      KILL(X) = K + (KILL(X) – G)
   endfor
endfor
```

Class Problem - Aexprs Calculation



Some Things to Think About

- Liveness and rdefs are basically the same thing
  - All dataflow is basically the same with a few parameters
    - Meaning of gen/kill – src vs dest, variable vs operation
    - Backward / Forward
    - All paths / some paths (must/may)
      - So far, we have looked at may analysis algorithms
      - How do you adjust to do must algorithms?
- Dataflow can be slow
  - How to implement it efficiently?
    - Forward analysis – DFS order
    - Backward analysis – PostDFS order
  - How to represent the info?
- Predicates
  - Throw a monkey wrench into this stuff
  - So, how are predicates handled?