EECS 583 – Class 6
More Dataflow Analysis

University of Michigan

September 24, 2018

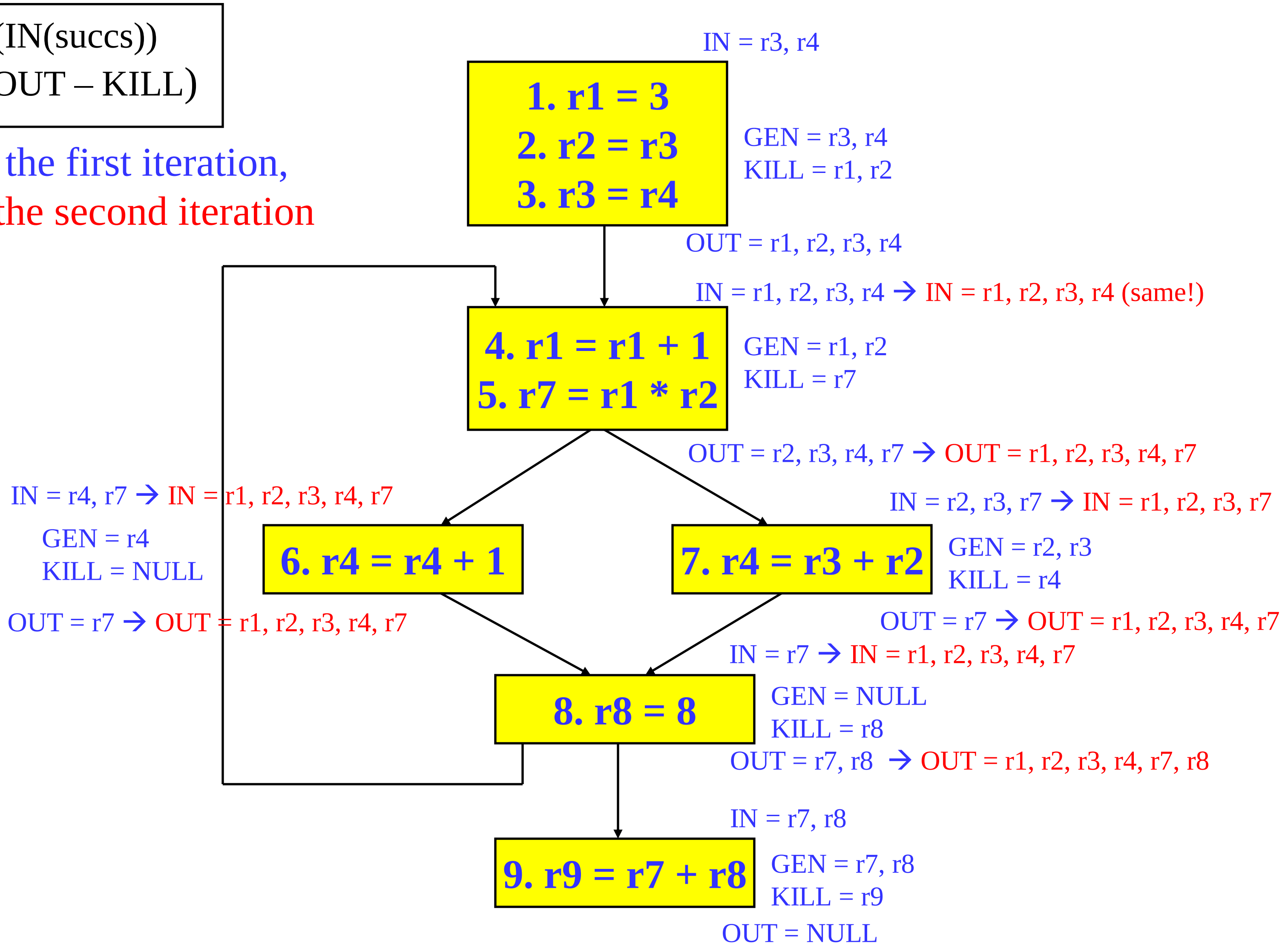
Announcements & Reading Material

- HW 1 due tonight at midnight
  - Hopefully you are done or very close to finishing
- 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 Edition 1; 9.2, 9.3 Edition 2)
- Material for Wednesday
  - “Practical Improvements to the Construction and Destruction of Static Single Assignment Form,” P. Briggs, K. Cooper, T. Harvey, and L. Simpson, *Software--Practice and Experience*, 28(8), July 1998, pp. 859-891.

Last Time: Liveness Class Problem Answer

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

Blue sets are the first iteration,
Red sets are the second iteration



IN = r3, r4

1. r1 = 3
2. r2 = r3
3. r3 = r4

GEN = r3, r4
KILL = r1, r2

OUT = r1, r2, r3, r4

IN = r1, r2, r3, r4 → IN = r1, r2, r3, r4 (same!)

4. r1 = r1 + 1
5. r7 = r1 * r2

GEN = r1, r2
KILL = r7

OUT = r2, r3, r4, r7 → OUT = r1, r2, r3, r4, r7

IN = r4, r7 → IN = r1, r2, r3, r4, r7

GEN = r4
KILL = NULL

6. r4 = r4 + 1

OUT = r7 → OUT = r1, r2, r3, r4, r7

IN = r2, r3, r7 → IN = r1, r2, r3, r7

7. r4 = r3 + r2

GEN = r2, r3
KILL = r4

OUT = r7 → OUT = r1, r2, r3, r4, r7

IN = r7 → IN = r1, r2, r3, r4, r7

8. r8 = 8

GEN = NULL
KILL = r8

OUT = r7, r8 → OUT = r1, r2, r3, r4, r7, r8

IN = r7, r8

9. r9 = r7 + r8

GEN = r7, r8
KILL = r9

OUT = NULL

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
```

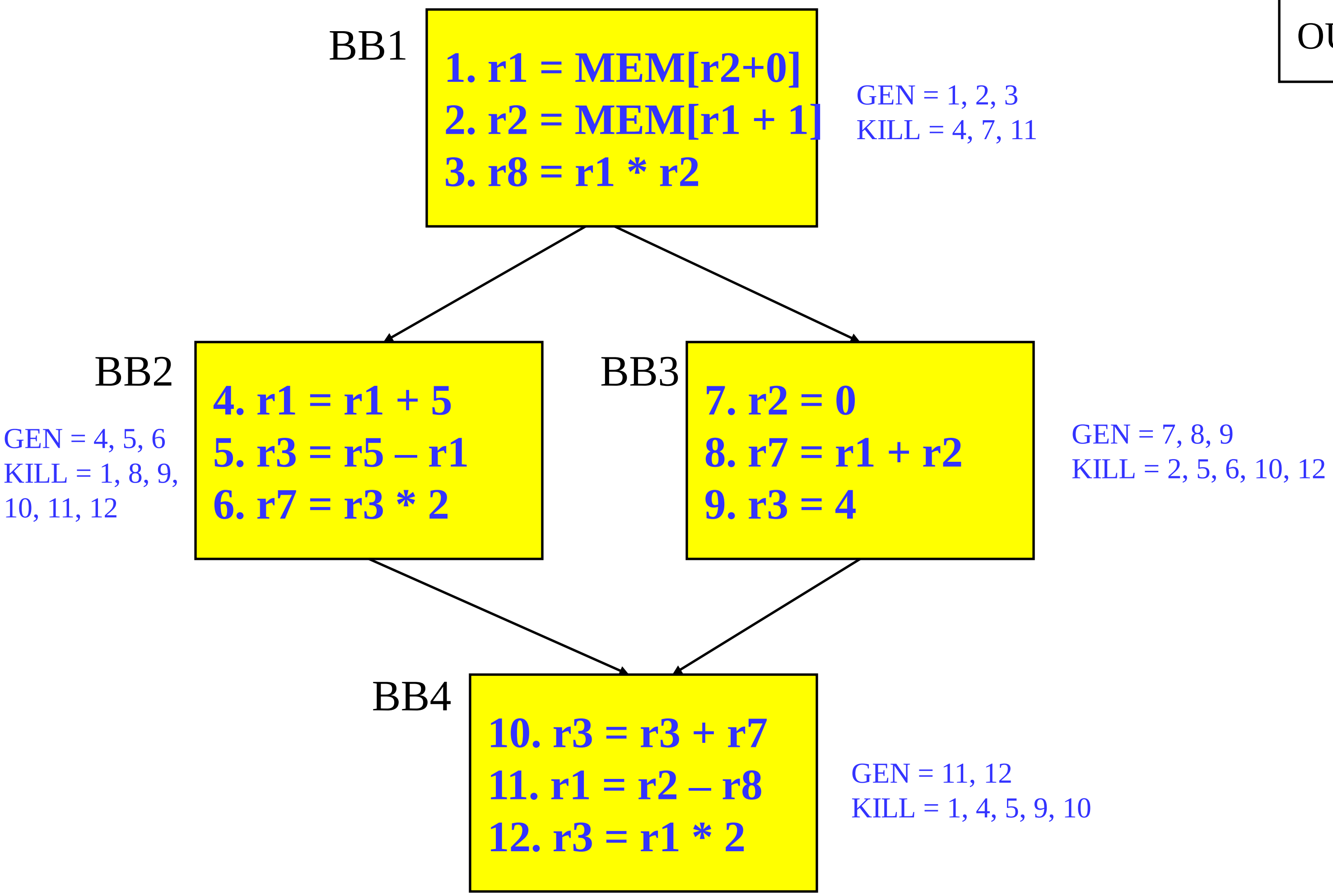
Compute Rdef IN/OUT Sets for all BBs

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)



BB1

1. r1 = MEM[r2+0]
2. r2 = MEM[r1 + 1]
3. r8 = r1 * r2

GEN = 1, 2, 3
KILL = 4, 7, 11

BB2

4. r1 = r1 + 5
5. r3 = r5 – r1
6. r7 = r3 * 2

GEN = 4, 5, 6
KILL = 1, 8, 9, 10, 11, 12

BB3

7. r2 = 0
8. r7 = r1 + r2
9. r3 = 4

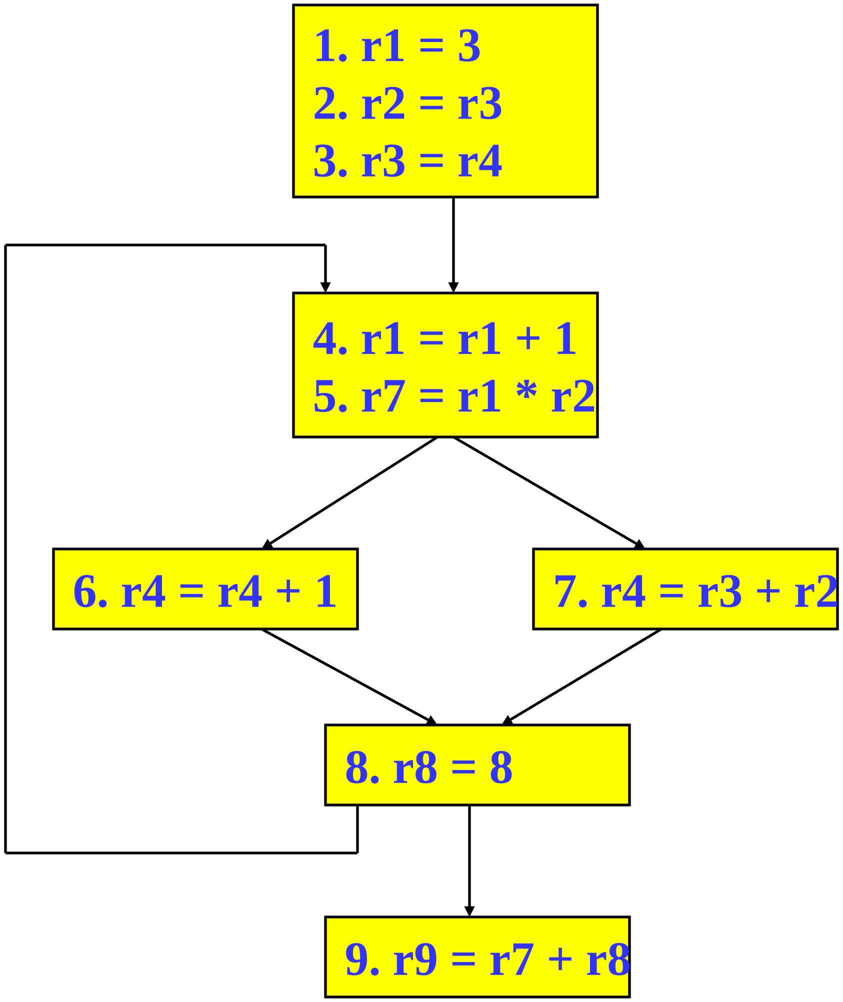
GEN = 7, 8, 9
KILL = 2, 5, 6, 10, 12

BB4

10. r3 = r3 + r7
11. r1 = r2 – r8
12. r3 = r1 * 2

GEN = 11, 12
KILL = 1, 4, 5, 9, 10

Class Problem - Rdefs



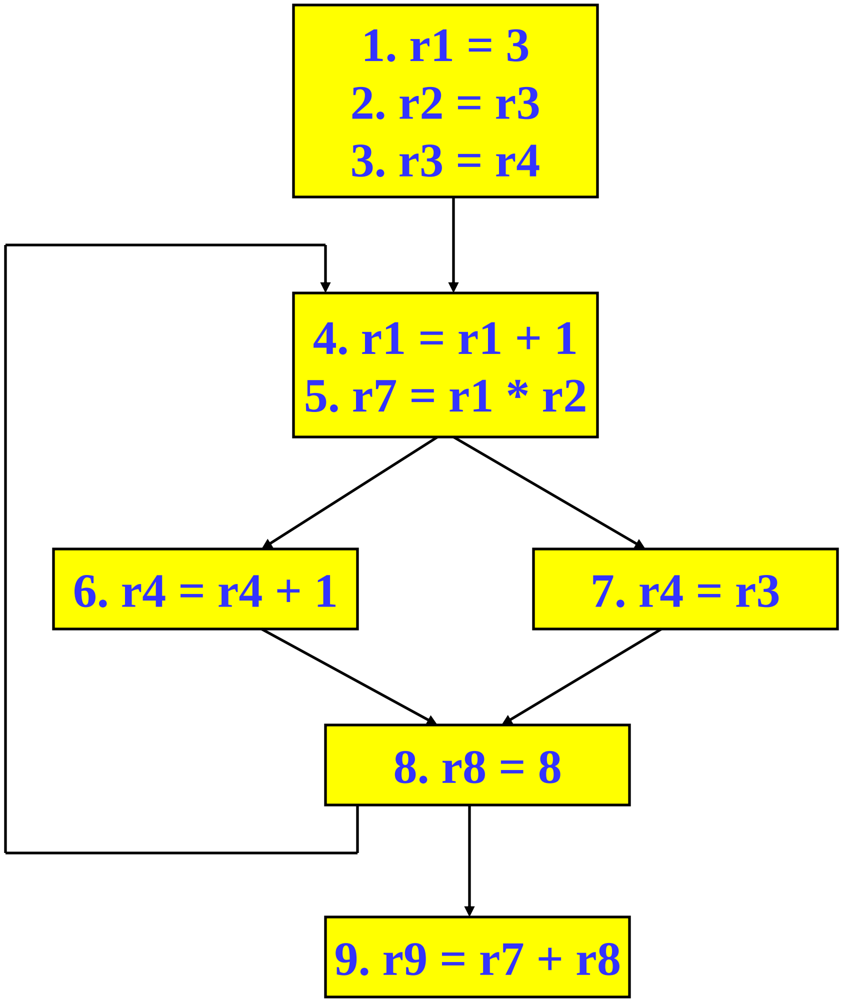
Compute reaching defs

- 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



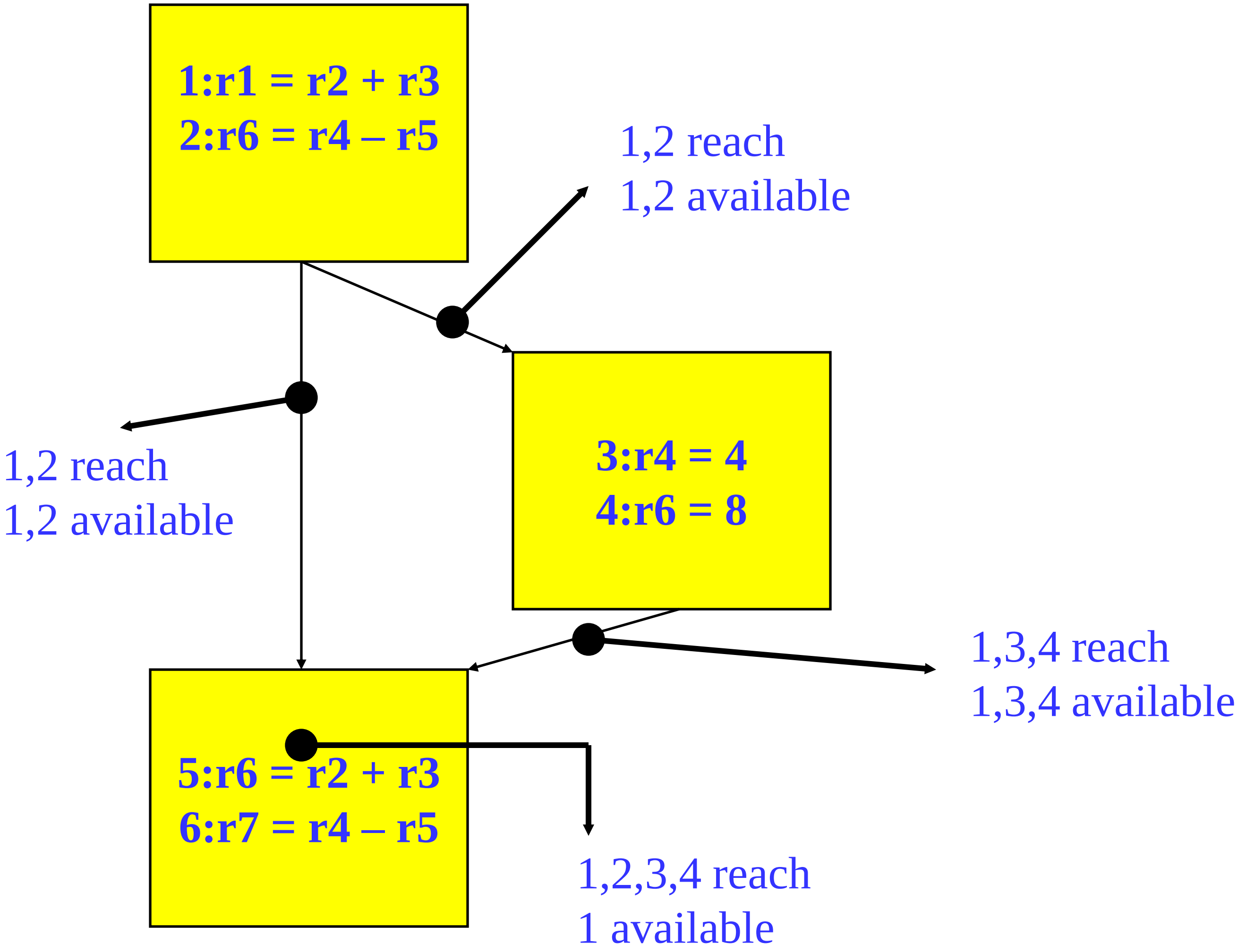
Generalizing Dataflow Analysis

- 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

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)

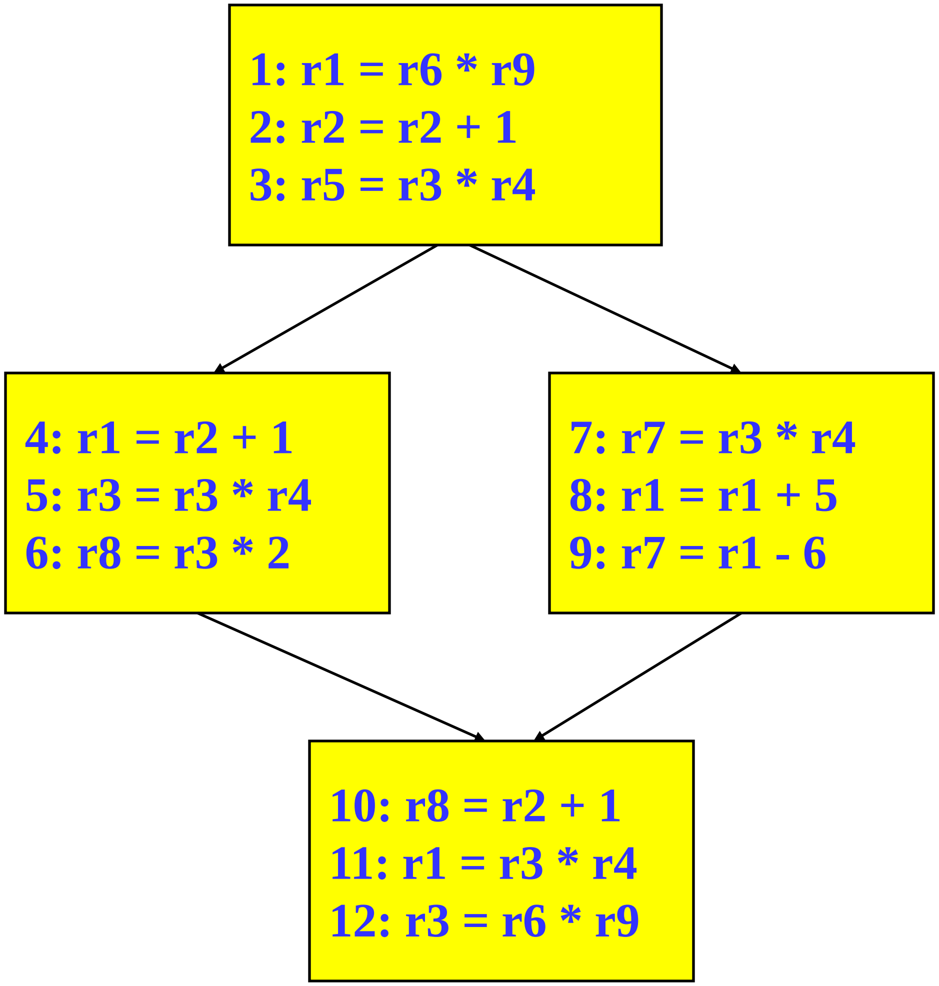
- An expression is a RHS of an operation
  - r2 = r3 + r4, r3+r4 is an expression
- An expression e is available at a point p if along all paths from e to p, e is not killed
- An expression is killed between 2 points when one of its source operands are redefined
  - r1 = r2 + r3 kills all expressions 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



```
1: r1 = r6 * r9
2: r2 = r2 + 1
3: r5 = r3 * r4
```

```
4: r1 = r2 + 1
5: r3 = r3 * r4
6: r8 = r3 * 2
```

```
7: r7 = r3 * r4
8: r1 = r1 + 5
9: r7 = r1 - 6
```

```
10: r8 = r2 + 1
11: r1 = r3 * r4
12: r3 = r6 * r9
```

Dataflow Analyses in 1 Slide

Liveness

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

Bottom-up dataflow
Any path
Keep track of variables/registers
Uses of variables → GEN
Defs of variables → KILL

Reaching Definitions/DU/UD

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

Top-down dataflow
Any path
Keep track of instruction IDs
Defs of variables → GEN
Defs of variables → KILL

Available Expressions

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

Top-down dataflow
All path
Keep track of instruction IDs
Expressions of variables → GEN
Defs of variables → KILL

Available Definitions

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

Top-down dataflow
All path
Keep track of instruction IDs
Defs of variables → GEN
Defs of variables → KILL

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)
    - What other dataflow analysis problems can be formulated?
- 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?