EECS 583 – Class 5
Dataflow Analysis

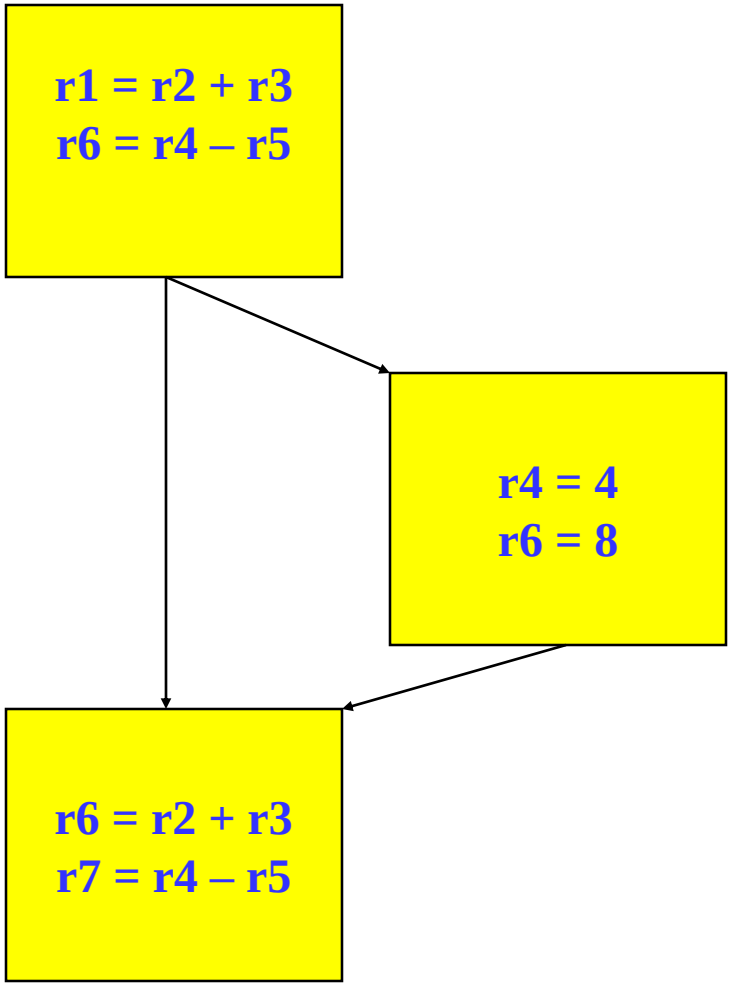
University of Michigan

January 29, 2024

Reading Material + Announcements

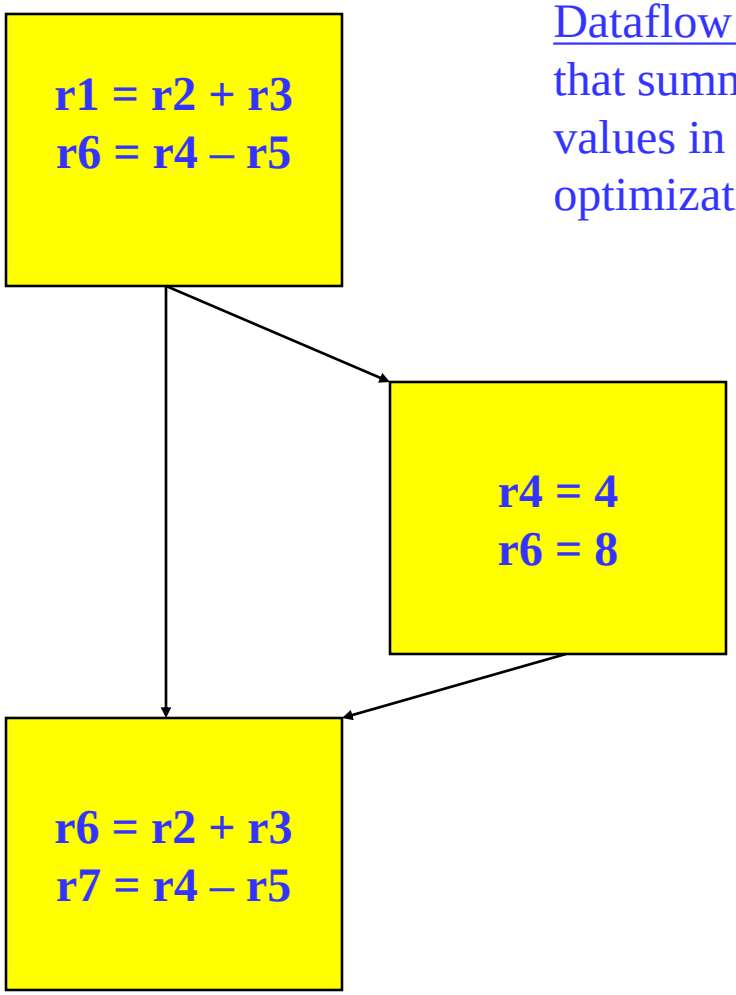
- HW 1 due tonight, midnight
  - Submit uniquename_hw1.tgz file to:
    - eecs583a.eecs.umich.edu:/hw1_submissions
  - Before asking questions: 1) Read all threads on piazza, 2) Think a bit
    - Then, post question or talk to Aditya/Yunjie if you are stuck
  - If you don't finish – Check late policy, submit whatever you have
- Today's class
  - *Compilers: Principles, Techniques, and Tools*,
    A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988.
    (Chapters: 10.5, 10.6 Edition 1; Chapters 9.2 Edition 2)
- Material for next Monday
  - *Compilers: Principles, Techniques, and Tools*,
    A. Aho, R. Sethi, and J. Ullman, Addison-Wesley, 1988.
    (Chapters: 10.5, 10.6, 10.9, 10.10 Edition 1; Chapters 9.2, 9.3 Edition 2)

Looking Inside the Basic Blocks: Dataflow Analysis + Optimization



- Control flow analysis
  - Treat BB as black box
  - Just care about branches
- Now
  - Start looking at ops in BBs
  - What's computed and where
- Classical optimizations
  - Want to make the computation more efficient
- Ex: Common Subexpression Elimination (CSE)
  - Is r2 + r3 redundant?
  - Is r4 – r5 redundant?
  - What if there were 1000 BB's
  - Dataflow analysis !!

Dataflow Analysis Introduction



Dataflow analysis – Collection of information that summarizes the creation/destruction of values in a program. Used to identify legal optimization opportunities.

Pick an arbitrary point in the program

Which VRs contain useful data values? (liveness or upward exposed uses)

Which definitions may reach this point? (reaching defns)

Which definitions are guaranteed to reach this point? (available defns)

Which uses below are exposed? (downward exposed uses)

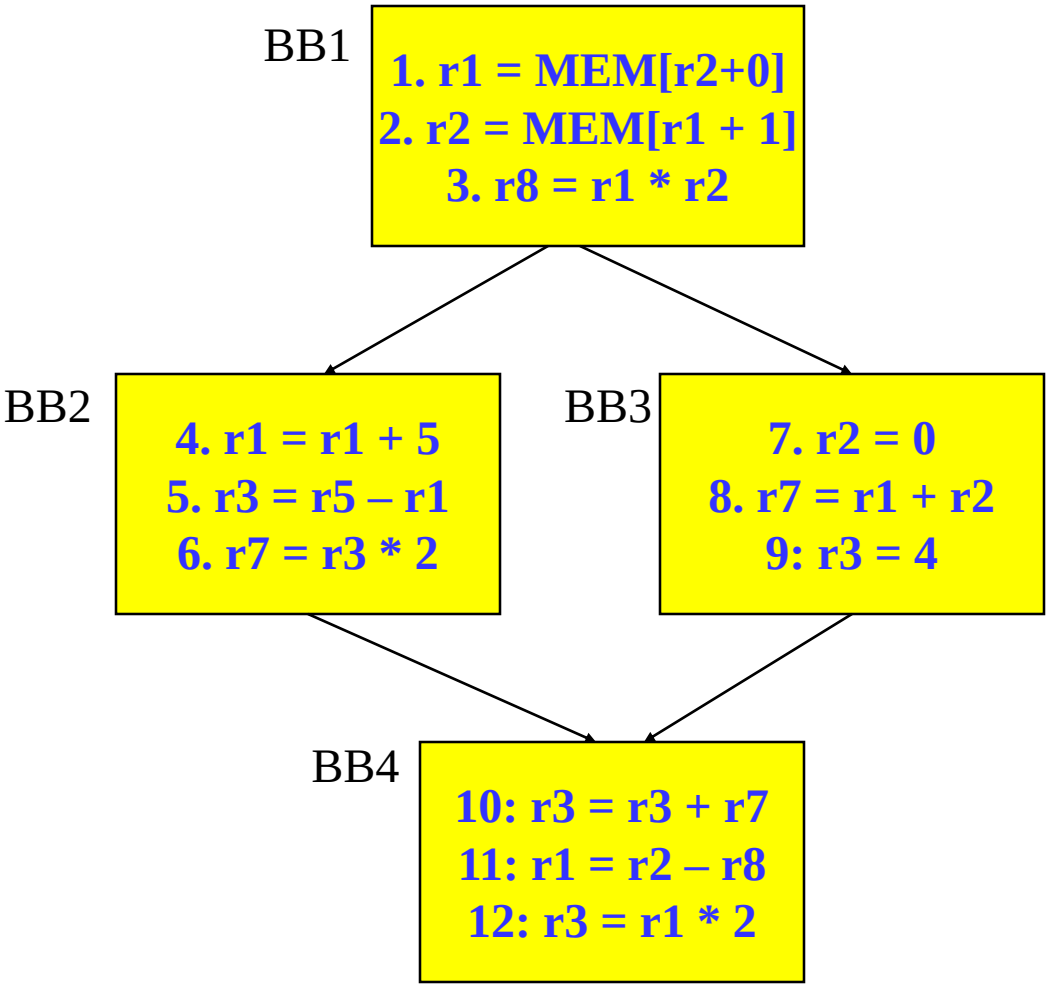
Live Variable (Liveness) Analysis

- 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
```

Example – GEN/KILL Liveness Computation

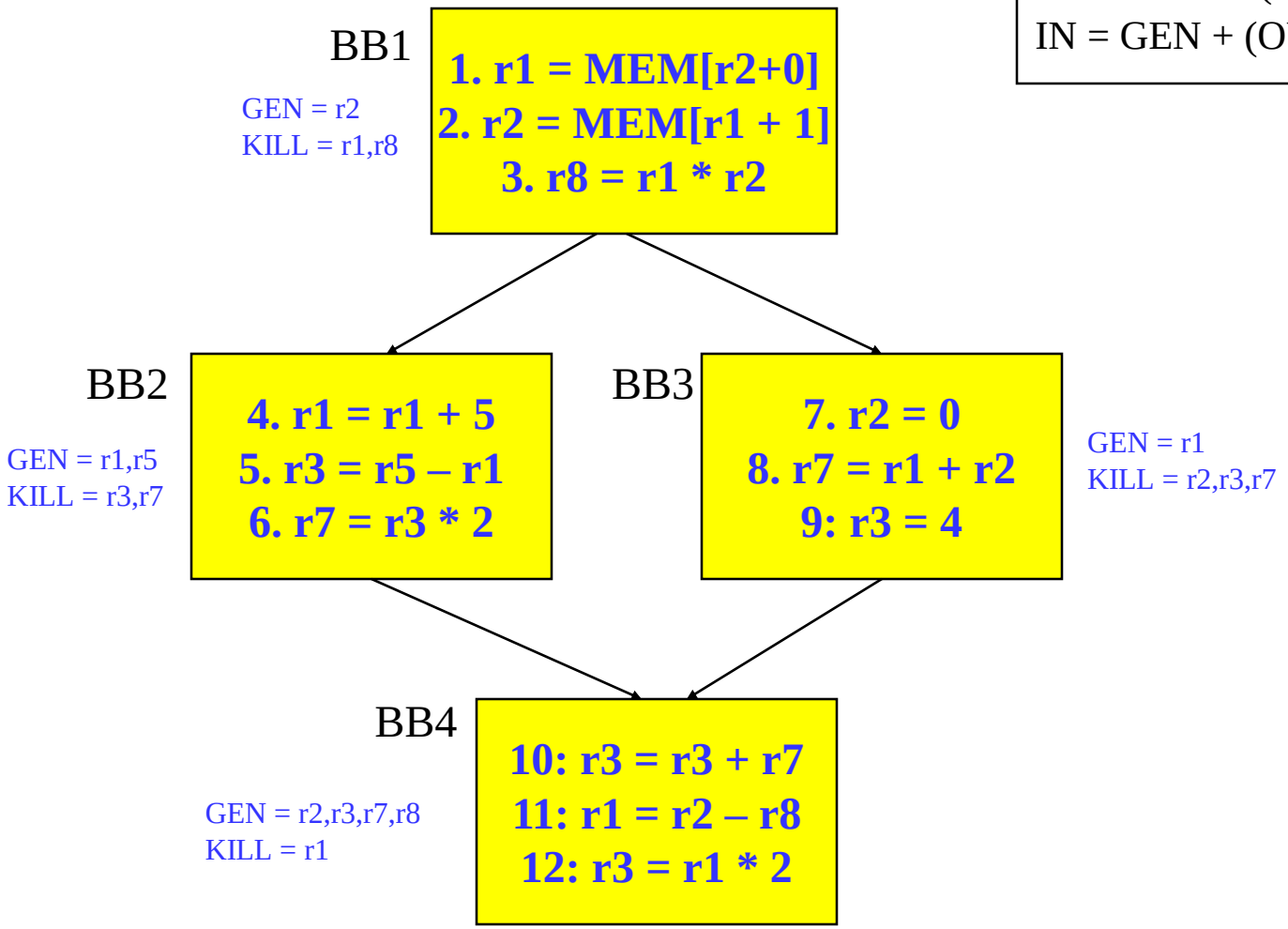


Compute IN/OUT Sets for all BBs

```
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
```

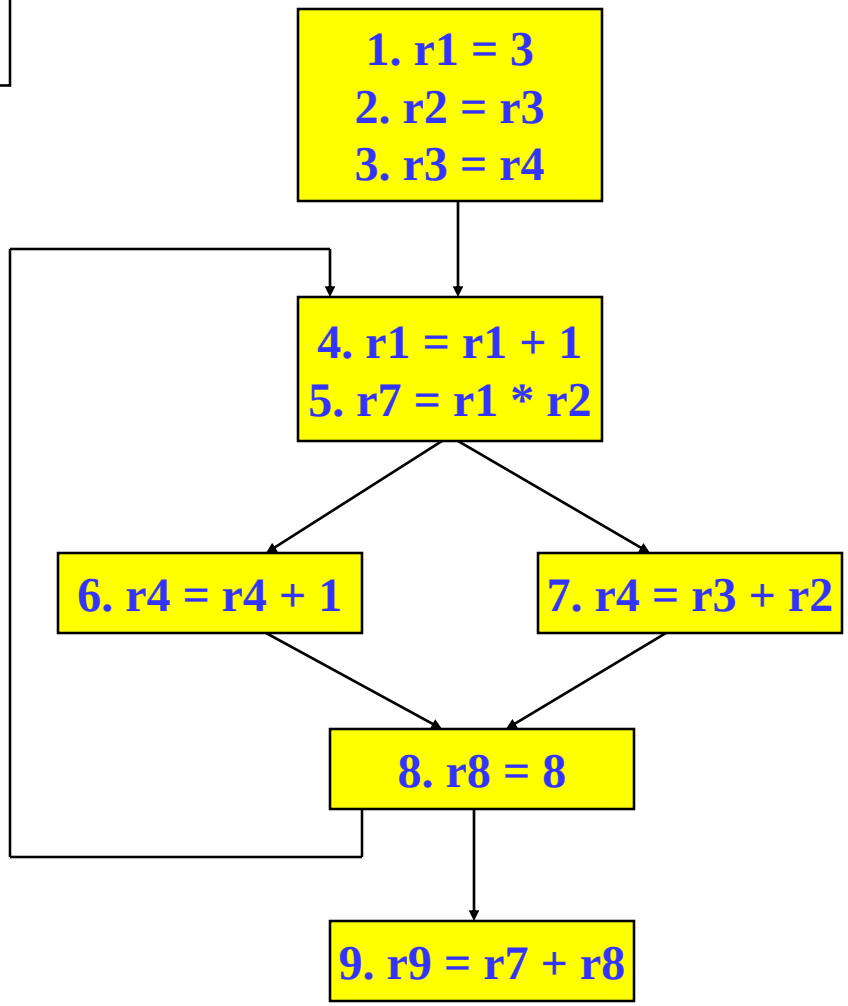
Example – Liveness Computation

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



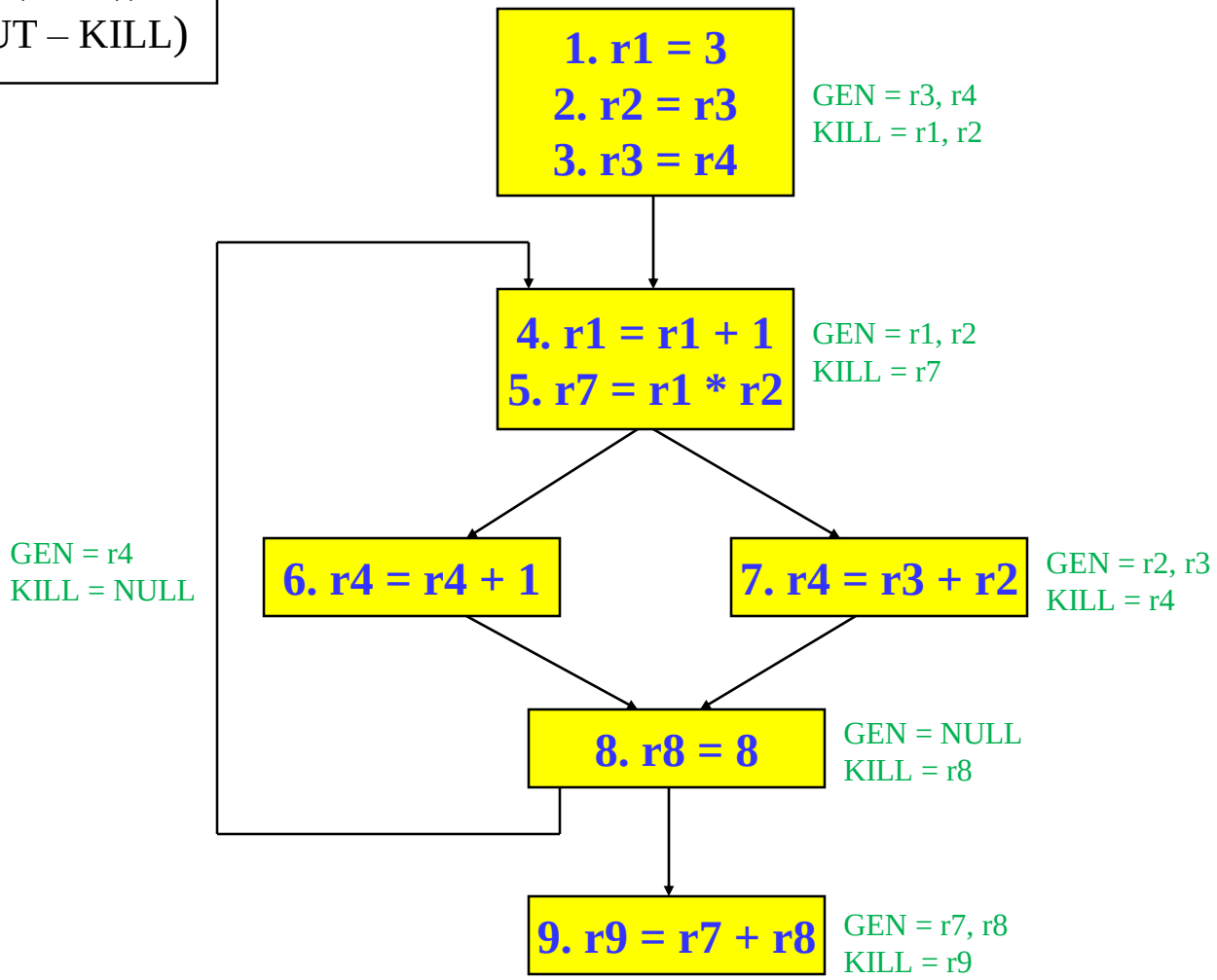
Liveness Class Problem

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



Liveness Class Problem - continued

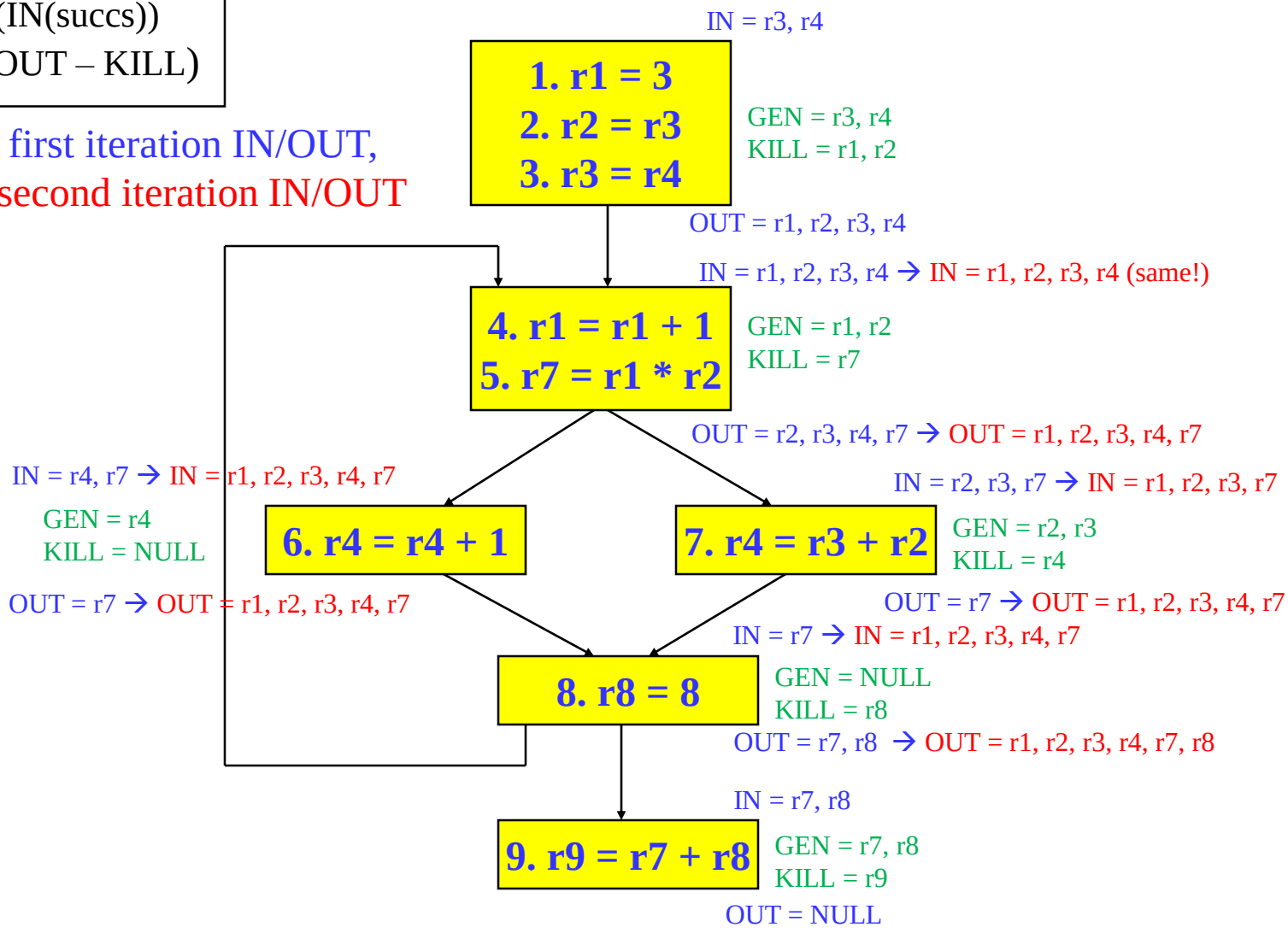
OUT = Union(IN(succs))
IN = GEN + (OUT – KILL)



Liveness Class Problem Answer

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

Blue sets are the first iteration IN/OUT,
Red sets are the second iteration IN/OUT



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)

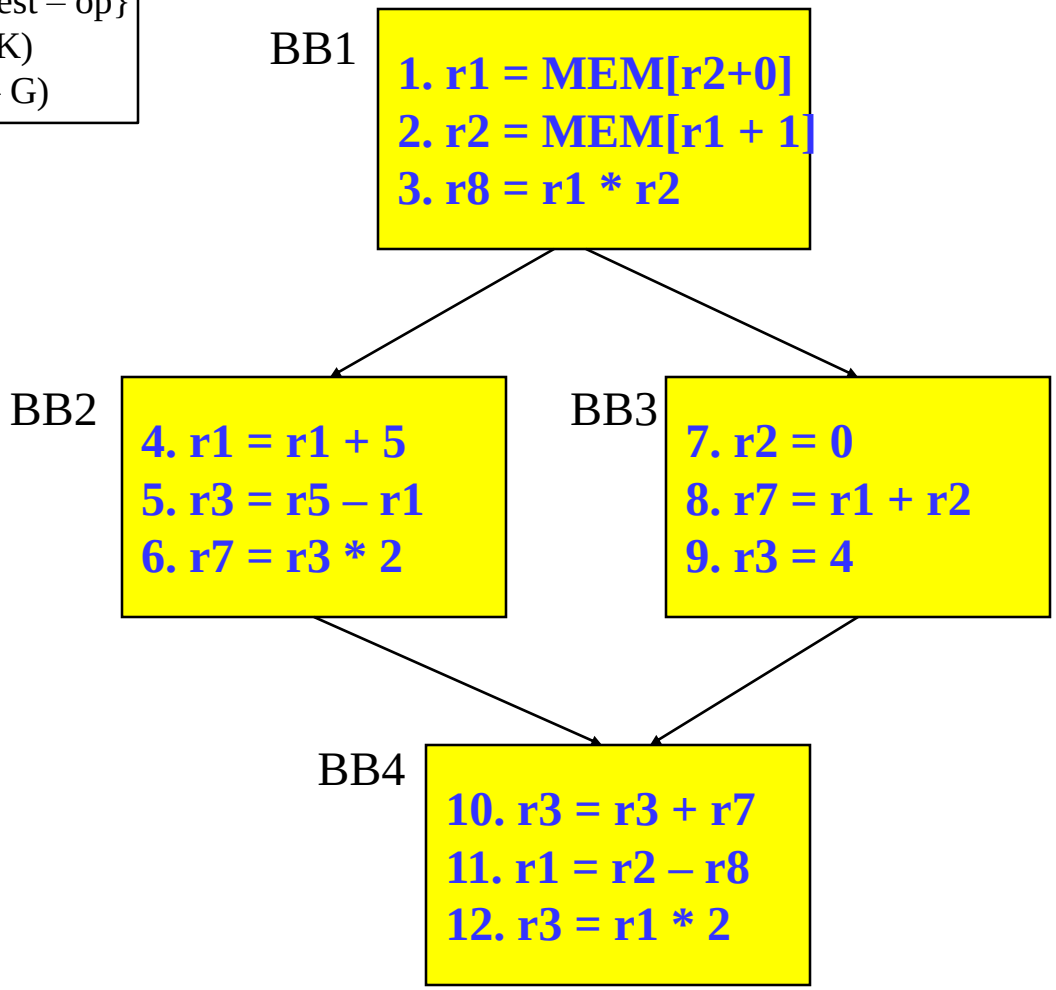
Compute Rdef GEN/KILL Sets for each BB

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
endwhile
```

Example GEN/KILL Rdef Calculation

G = op
K = {all ops which define dest – op}
GEN(X) = G + (GEN(X) – K)
KILL(X) = K + (KILL(X) – G)



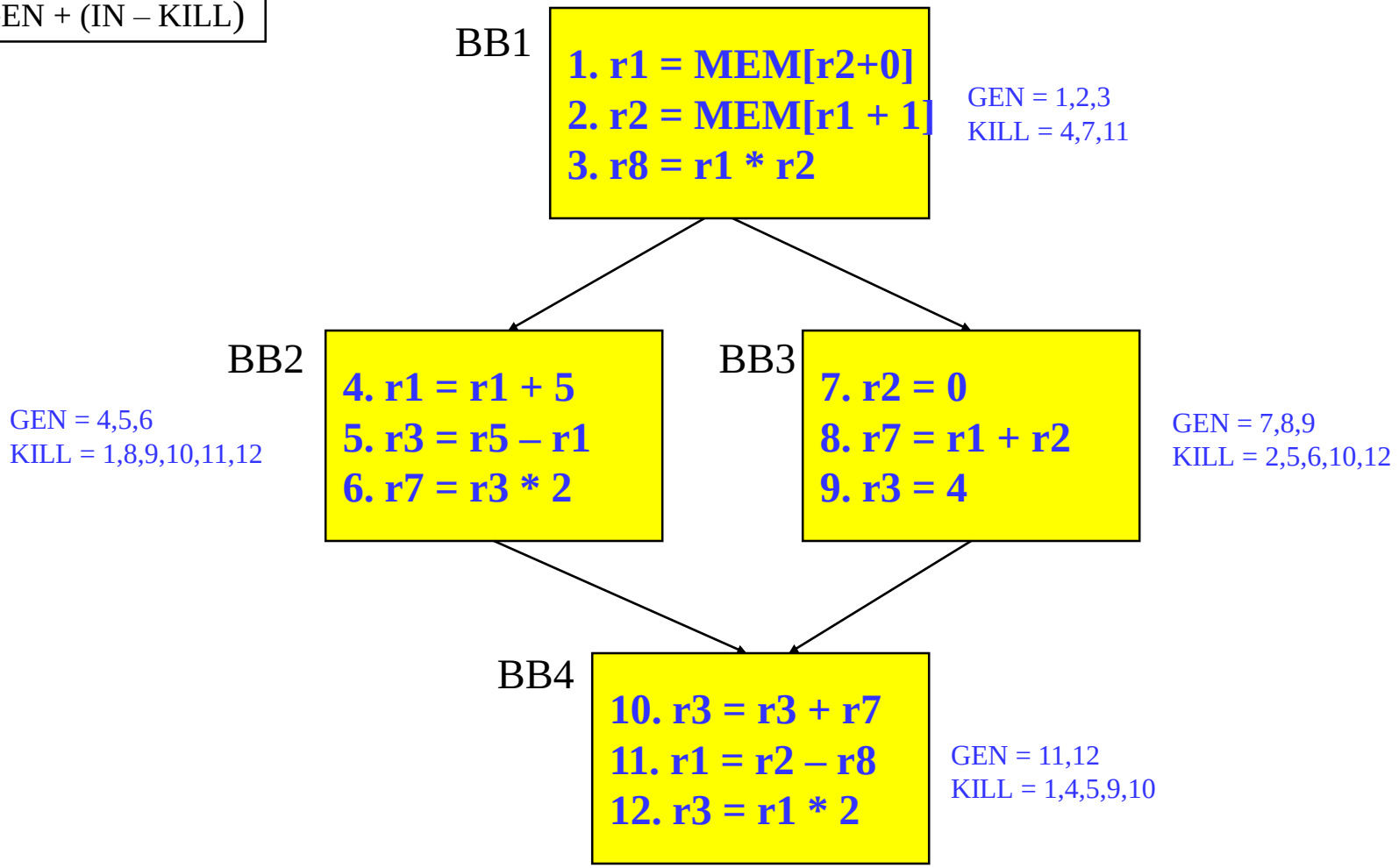
Compute Rdef IN/OUT Sets for all BBs

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
endwhile
```

Example In/Out Rdef Calculation

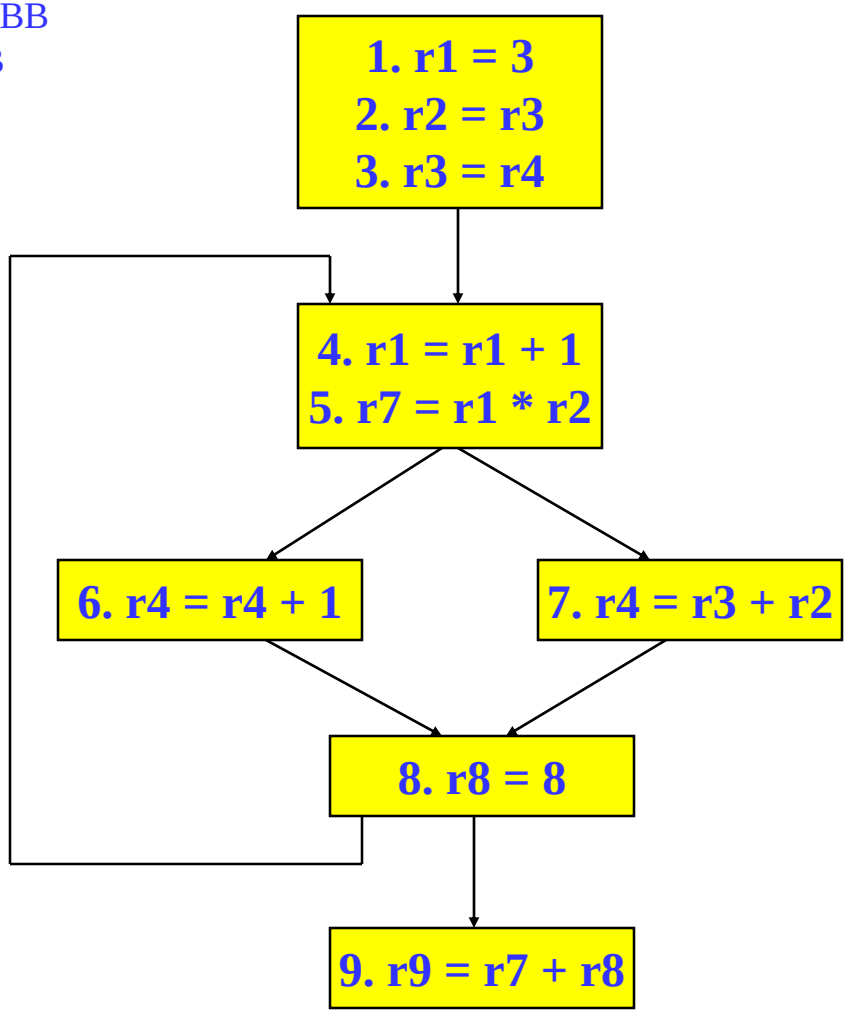
IN = Union(OUT(preds))
OUT = GEN + (IN – KILL)



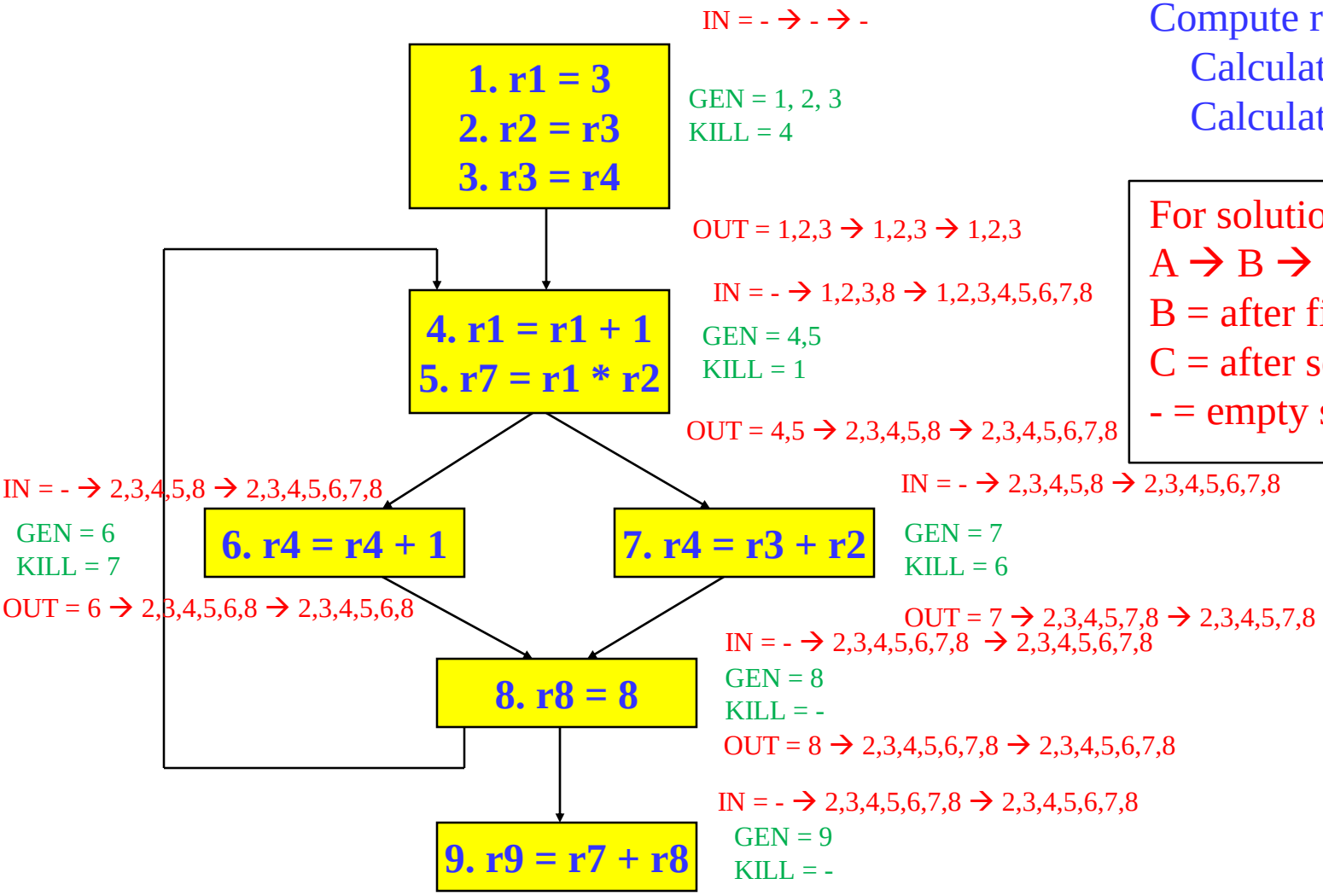
Rdefs Homework Problem

Compute reaching defs

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



Rdefs Homework Problem –**Answer**



Compute reaching defs

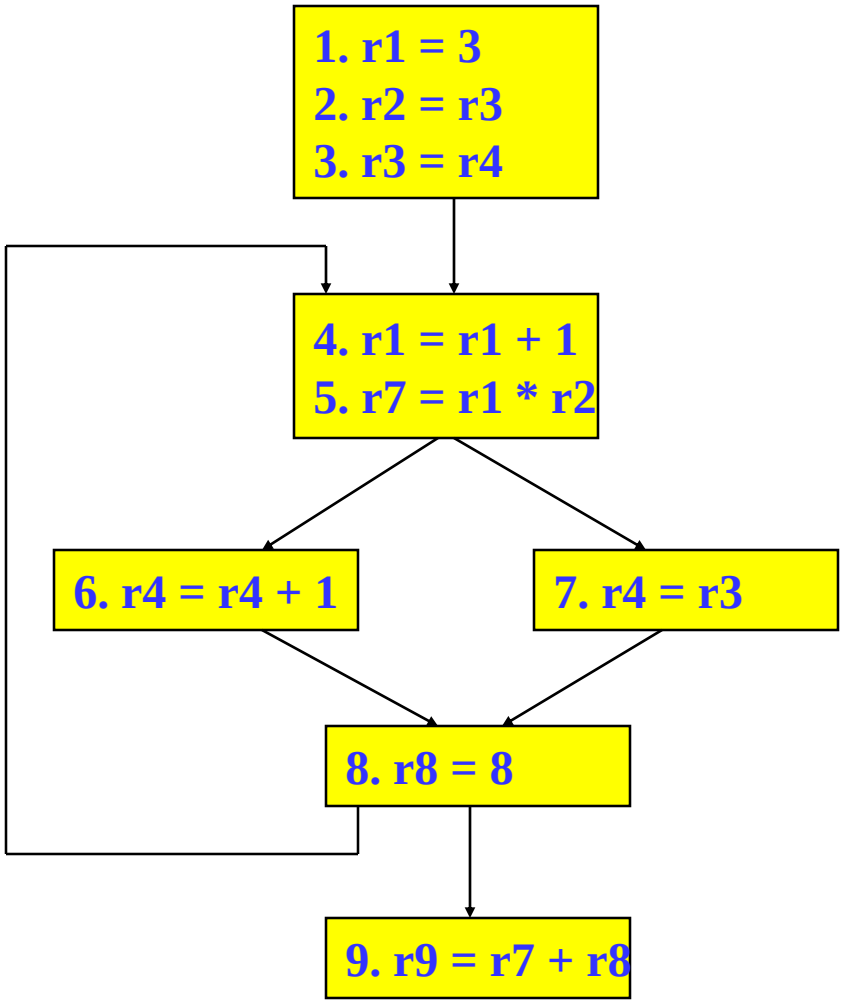
- Calculate GEN/KILL for each BB
- Calculate IN/OUT for each BB

For solution IN/OUT sets specified as A → B → C, A = initial state of sets, B = after first iteration of analysis C = after second iteration of analysis - = empty set

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



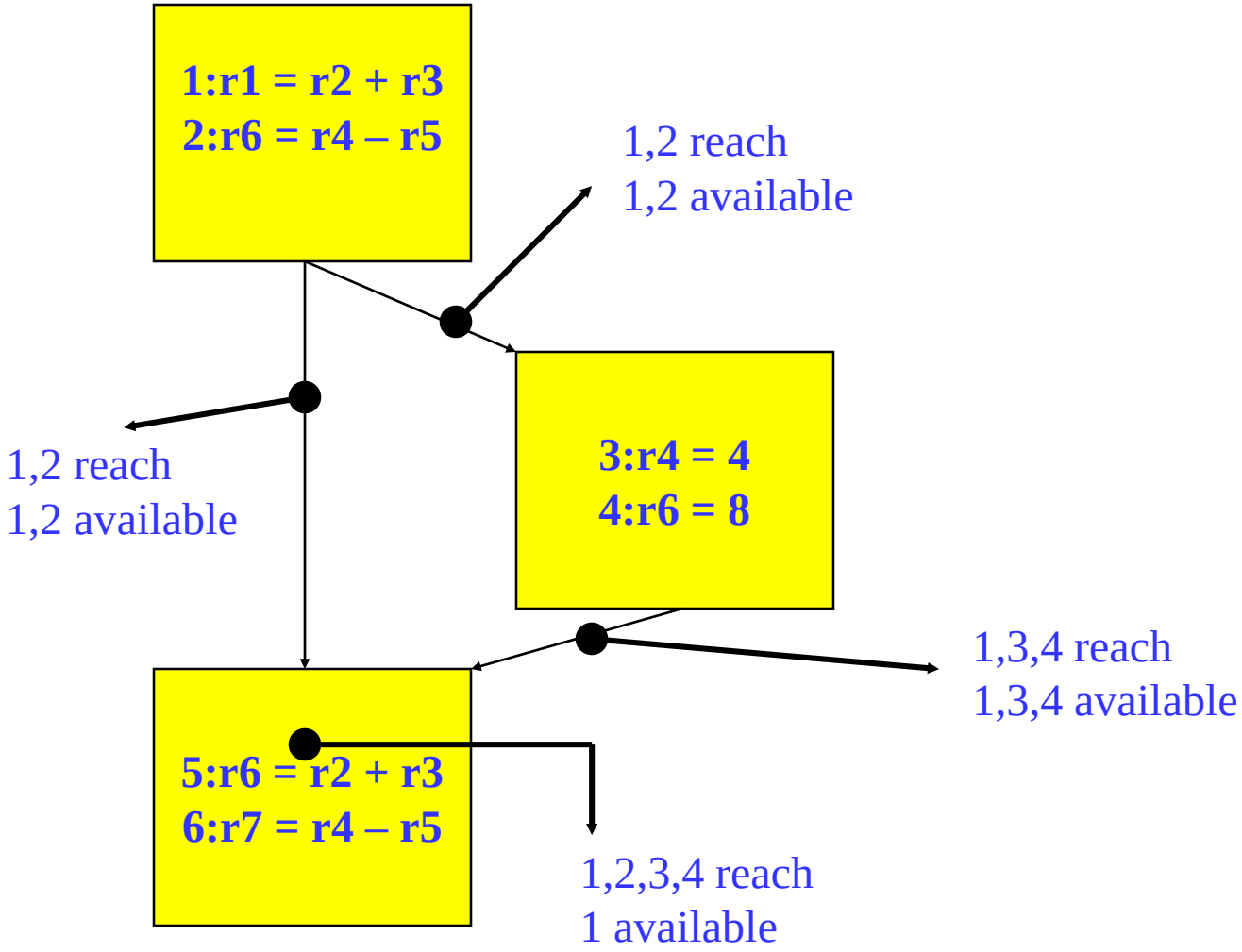
Generalizing Dataflow Analysis

- 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



TO BE CONTINUED …