

# EECS 583 – Class 6

## Dataflow Analysis II

---

*University of Michigan*

*September 18, 2023*

# Announcements & Reading Material

---

- ❖ HW 1 – Due tonight at midnight
  - » See piazza/GSIs if you are stuck
- ❖ HW 2 coming out Friday
- ❖ 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)
- ❖ Next class
  - » “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.

# Recap: Liveness vs Reaching Defs

---

## Liveness

$\text{OUT} = \text{Union}(\text{IN}(\text{succs}))$ $\text{IN} = \text{GEN} + (\text{OUT} - \text{KILL})$
--

Bottom-up dataflow

Any path

Keep track of variables/registers

Uses of variables  $\rightarrow$  GEN

Defs of variables  $\rightarrow$  KILL

## Reaching Definitions/DU/UD

$\text{IN} = \text{Union}(\text{OUT}(\text{preds}))$ $\text{OUT} = \text{GEN} + (\text{IN} - \text{KILL})$
--

Top-down dataflow

Any path

Keep track of instruction IDs

Defs of variables  $\rightarrow$  GEN

Defs of variables  $\rightarrow$  KILL

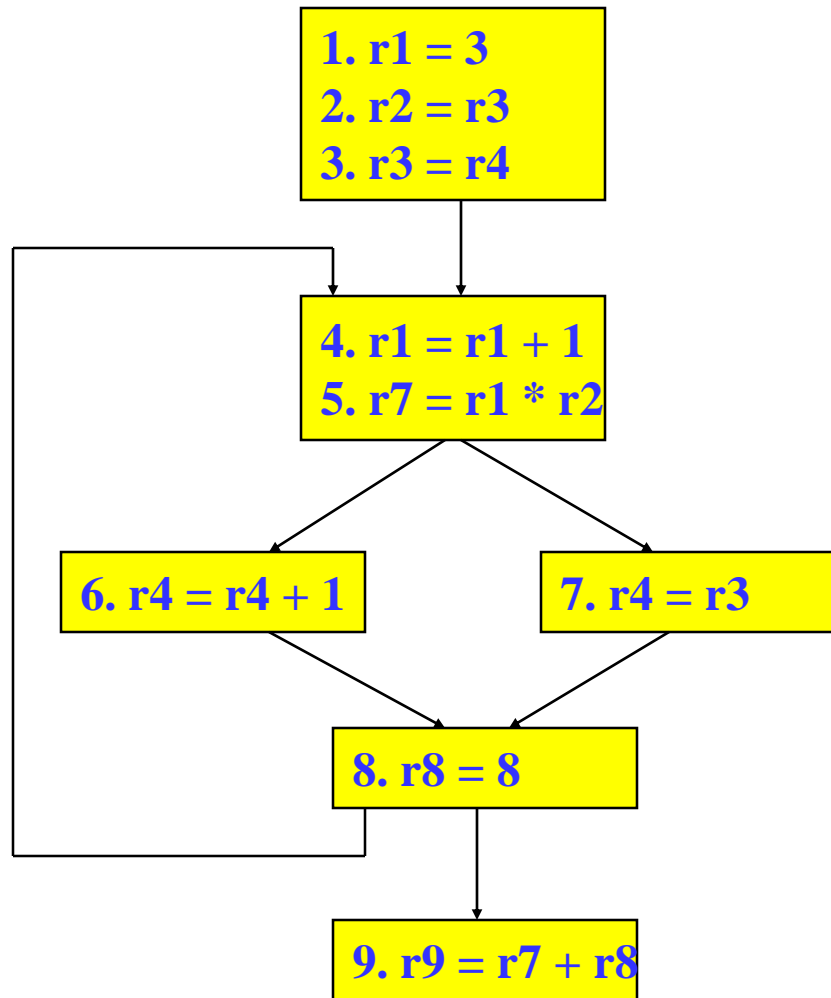
# DU/UD Chains

---

- ❖ 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, e.g., rdefs \*/
- »  $IN = GEN + (OUT - KILL)$  /\* backward analysis, e.g., liveness \*/

## ❖ Meet function

- » How information from multiple paths is combined
- »  $IN = \text{Union}(OUT(\text{predecessors}))$  /\* forward analysis \*/
- »  $OUT = \text{Union}(IN(\text{successors}))$  /\* backward analysis \*/

## ❖ Generalized dataflow algorithm

- » while (change)
  - change = false
  - for each BB
    - ◆ apply meet function
    - ◆ apply transfer functions
    - ◆ if any changes  $\rightarrow$  change = true

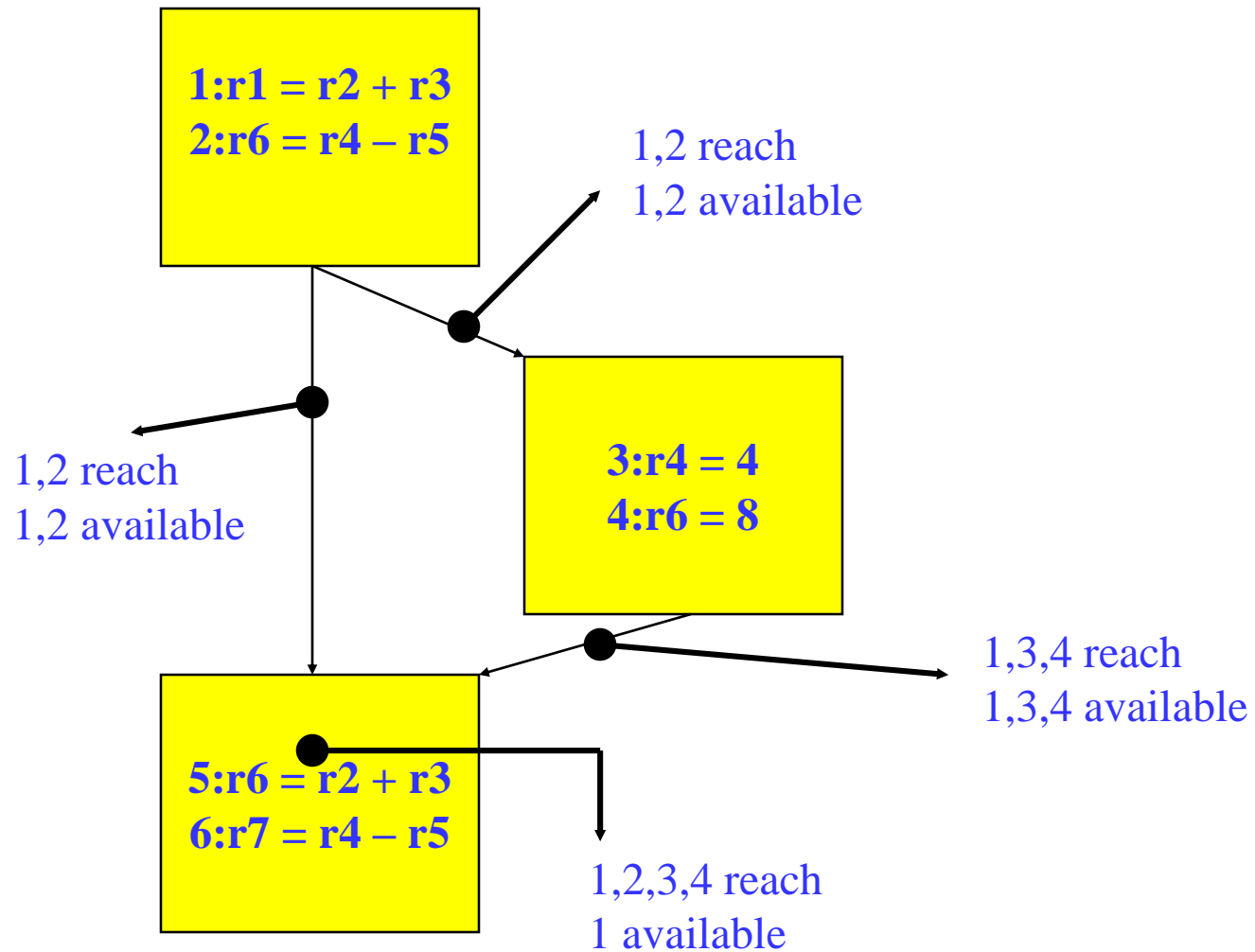
# What About All Path Problems?

---

- ❖ 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  
endwhile
```

# 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

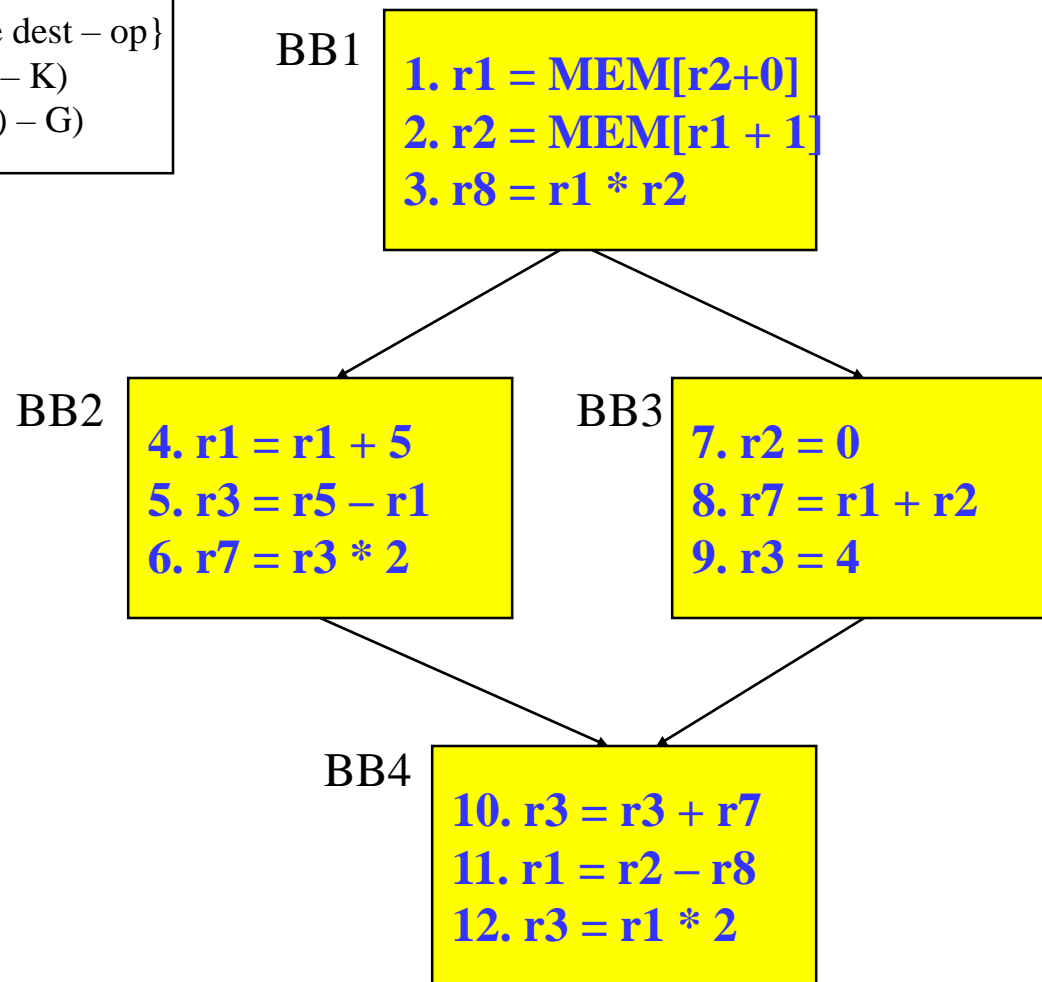
endfor

endwhile

# Example Adef Calculation

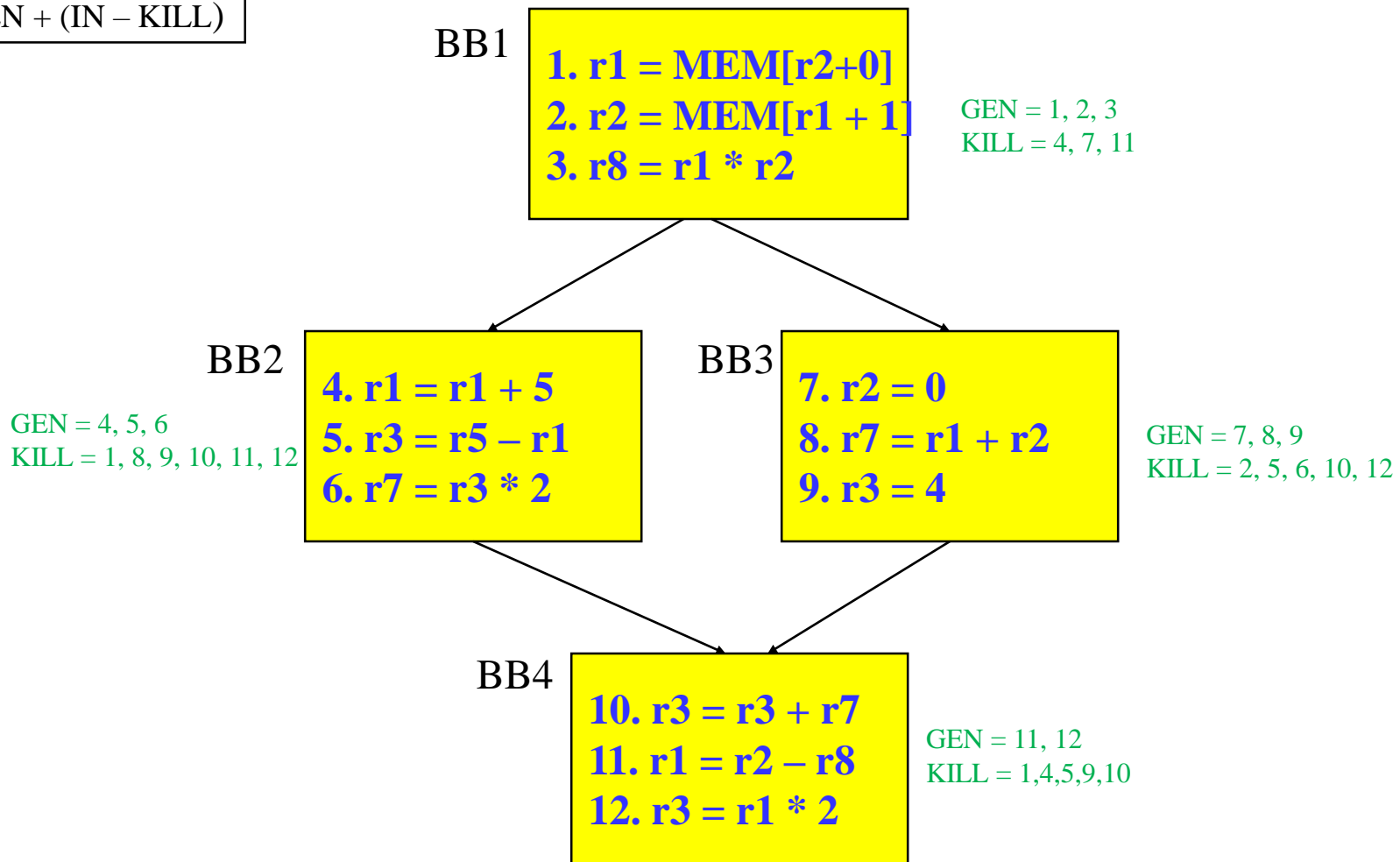
---

$G = \text{op}$   
 $K = \{\text{all ops which define dest} - \text{op}\}$   
 $\text{GEN}(X) = G + (\text{GEN}(X) - K)$   
 $\text{KILL}(X) = K + (\text{KILL}(X) - G)$



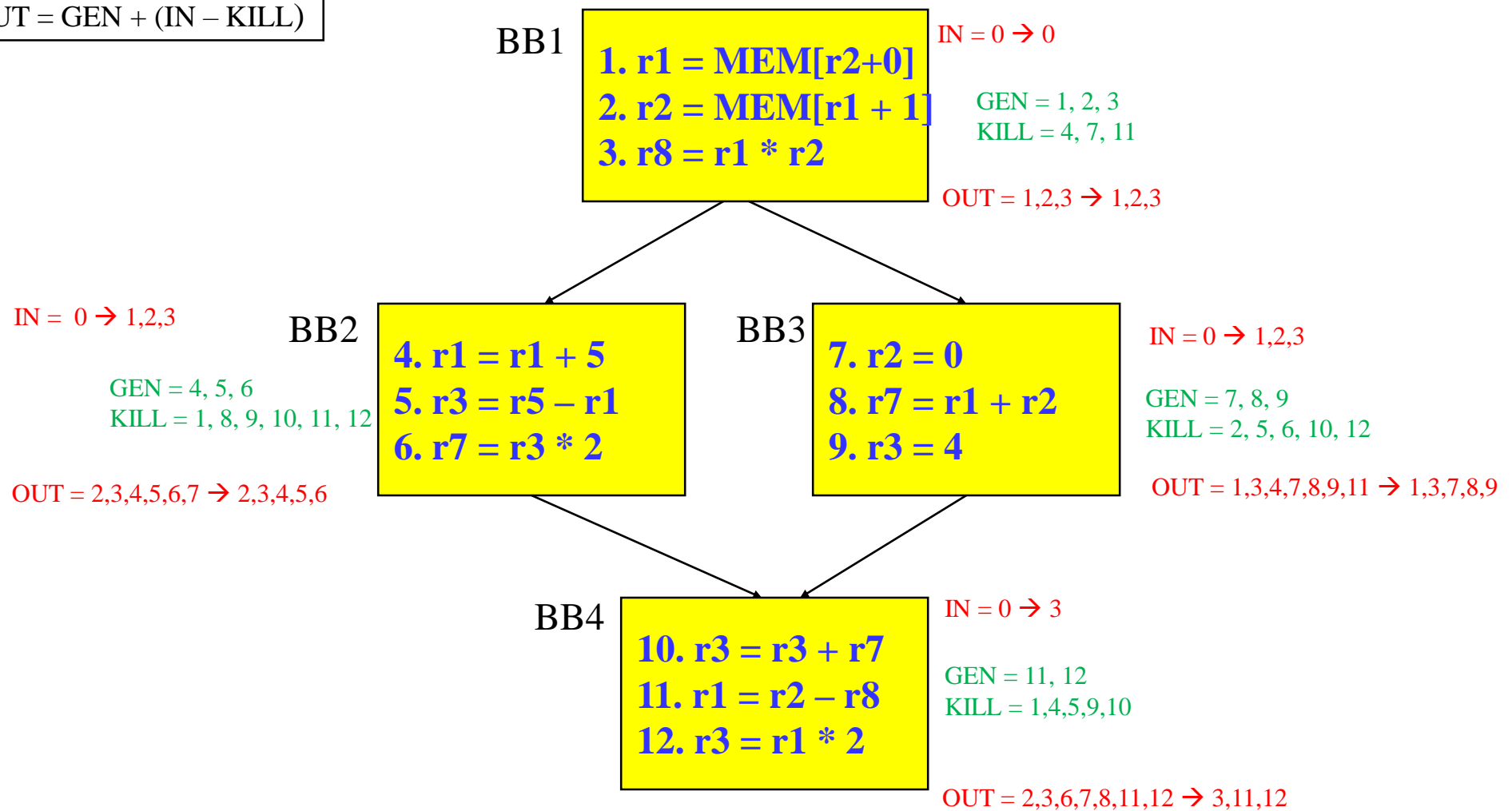
# Example Adef Calculation - Continued

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



# Example Adef Calculation - Answer

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



# 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

---

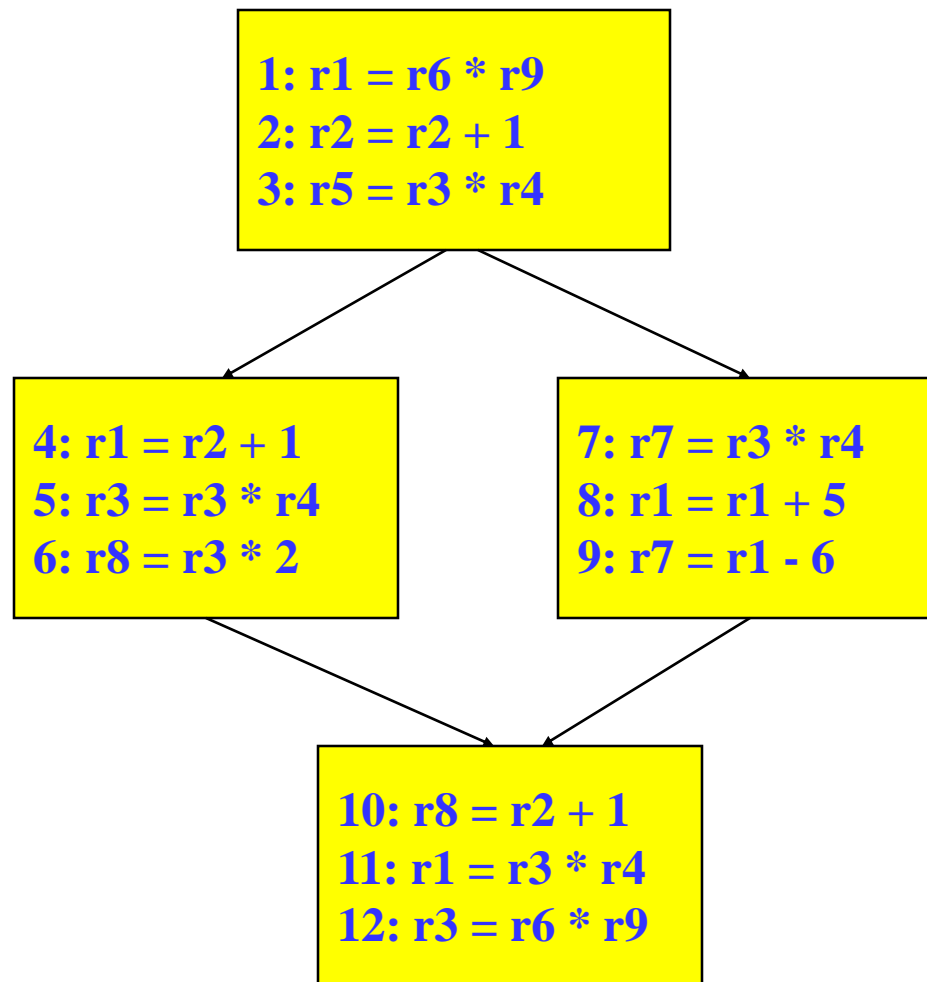
We can also formulate the GEN/KILL slightly differently so you do not need to break up instructions like “ $r2 = r2 + 1$ ”.

```
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
```



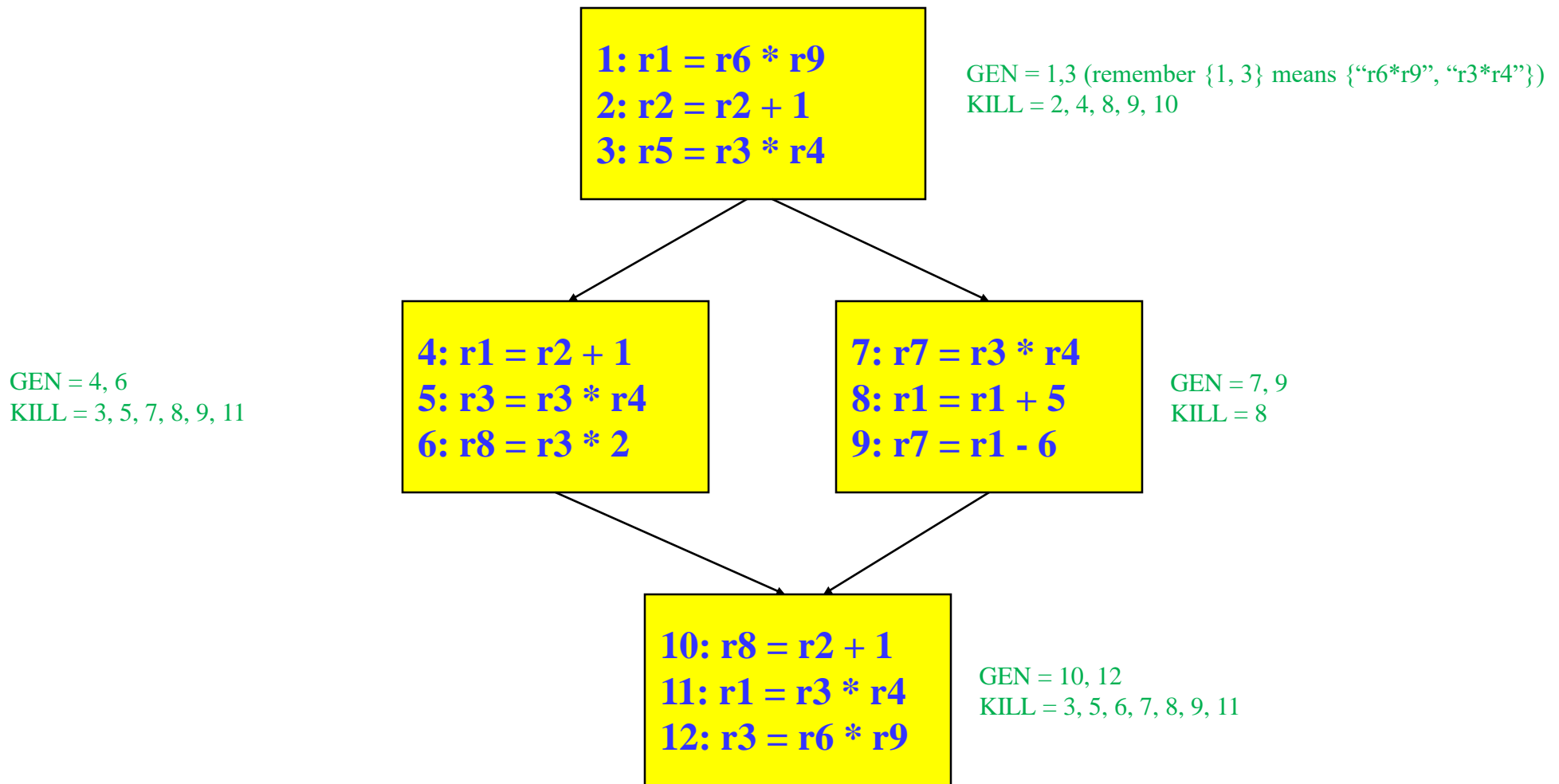
# Example Aexpr Calculation

---



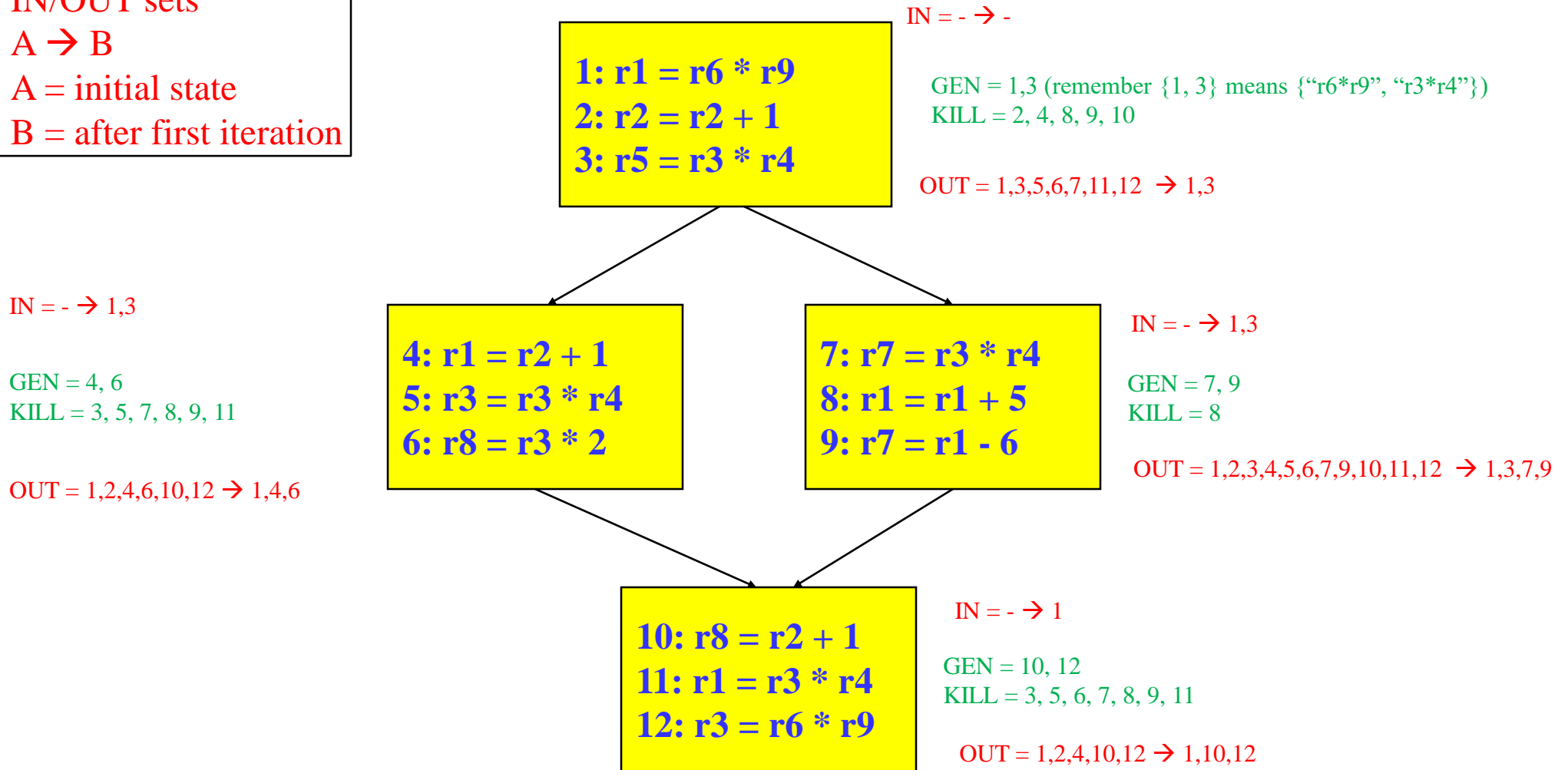
# Example Aexpr Calculation - Continued

---



# Example Aexpr Calculation - Answer

IN/OUT sets  
 $A \rightarrow B$   
 A = initial state  
 B = after first iteration



# Dataflow Summary

## 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