

Dataflow I: Dataflow Analysis

EECS 483 – Lecture 23
University of Michigan
Monday, November 27, 2006

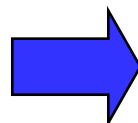
Announcements and Reading

- ❖ Project 3 – should have started work on this
- ❖ Schedule for the rest of the semester
 - » Today – Dataflow analysis
 - » Wednes 11/29 – Finish dataflow, optimizations
 - » Mon 12/4 – Optimizations, start on register allocation
 - » Wednes 12/6 – Register allocation, Exam 2 review
 - » Mon 12/11 – Exam 2 in class
 - » Wednes 12/13 – No class (Project 3 due)
- ❖ Reading for today's class
 - » 10.5, 10.6. 10.10, 10.11

From Last Time - Class Problem

Maximally optimize the control flow of this code

L1: if (a < b) goto L11
L2: goto L7
L3: goto L4
L4: stuff4
L5: if (c < d) goto L15
L6: goto L2
L7: if (c < d) goto L13
L8: goto L12
L9: stuff 9
L10: if (a < c) goto L3
L11: goto L9
L12: goto L2
L13: stuff 13
L14: if (e < f) goto L11
L15: stuff 15
L16: rts



L1: if (a < b) goto L9
L2: if (c < d) goto L13 goto L2
L4: stuff4
L5: if (c < d) goto L15
L6: if (c < d) goto L13 goto L2
L9: stuff 9
L10: if (a < c) goto L4
L11: goto L9
L13: stuff 13
L14: if (e < f) goto L9
L15: stuff 15
L16: rts

Blocks 7, 8, 12 are unreachable
Block 3 is empty

Dataflow Analysis Introduction

$r1 = r2 + r3$
 $r6 = r4 - r5$

$r4 = 4$
 $r6 = 8$

$r6 = r2 + r3$
 $r7 = r4 - r5$

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 defs)

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

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
 - » **USE** = set of external variables consumed in the BB
 - » **DEF** = set of variables defined in 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

Liveness Example

$r1 = r2 + r3$
 $r6 = r4 - r5$

$r2, r3, r4, r5$ are all live as they are consumed later, $r6$ is dead as it is redefined later

$r4 = 4$
 $r6 = 8$

$r4$ is dead, as it is redefined. So is $r6$. $r2, r3, r5$ are live

$r6 = r2 + r3$
 $r7 = r4 - r5$

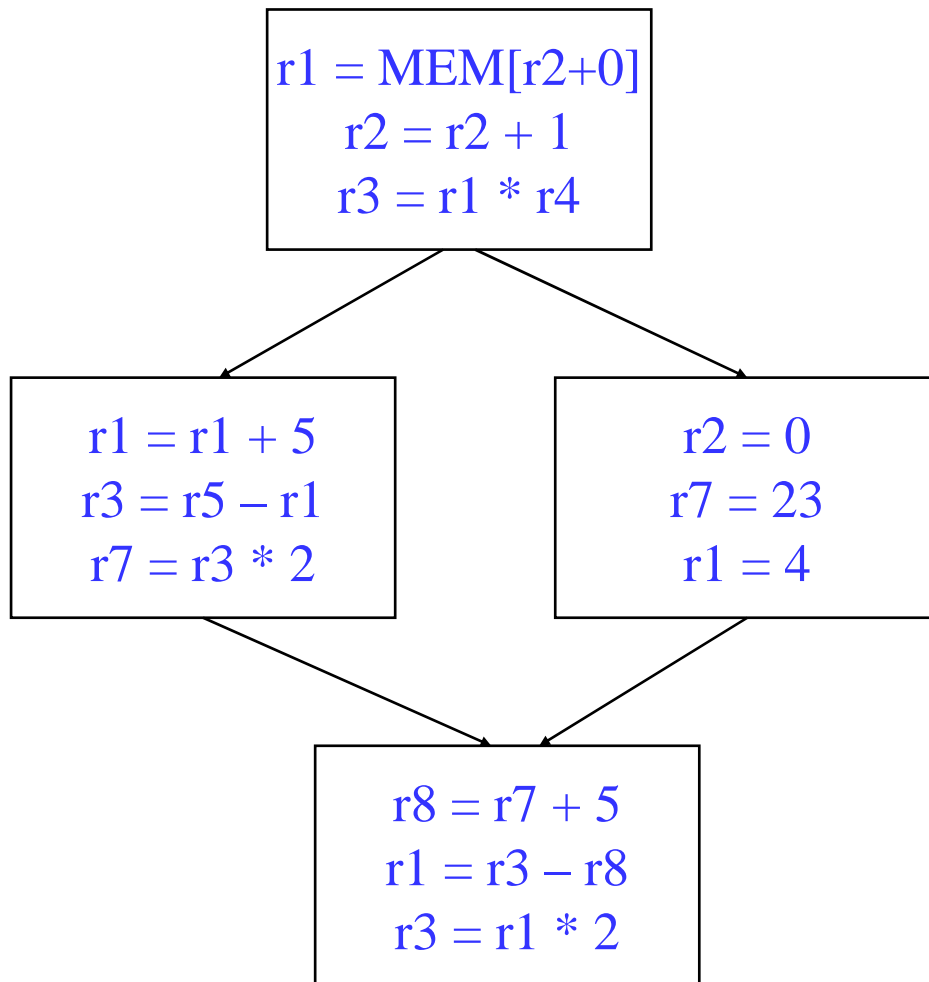
What does this mean? $r6 = r4 - r5$ is useless, it produces a dead value !!
Get rid of it!

Compute USE/DEF Sets For Each BB

```
for each basic block in the procedure, X, do  
  DEF(X) = 0  
  USE(X) = 0  
  for each operation in sequential order in X, op, do  
    for each source operand of op, src, do  
      if (src not in DEF(X)) then  
        USE(X) += src  
      endif  
    endfor  
    for each destination operand of op, dest, do  
      DEF(X) += dest  
    endfor  
  endfor  
endfor
```

def is the union of all the LHS's
use is all the VRs that are used before defined

Example USE/DEF Calculation



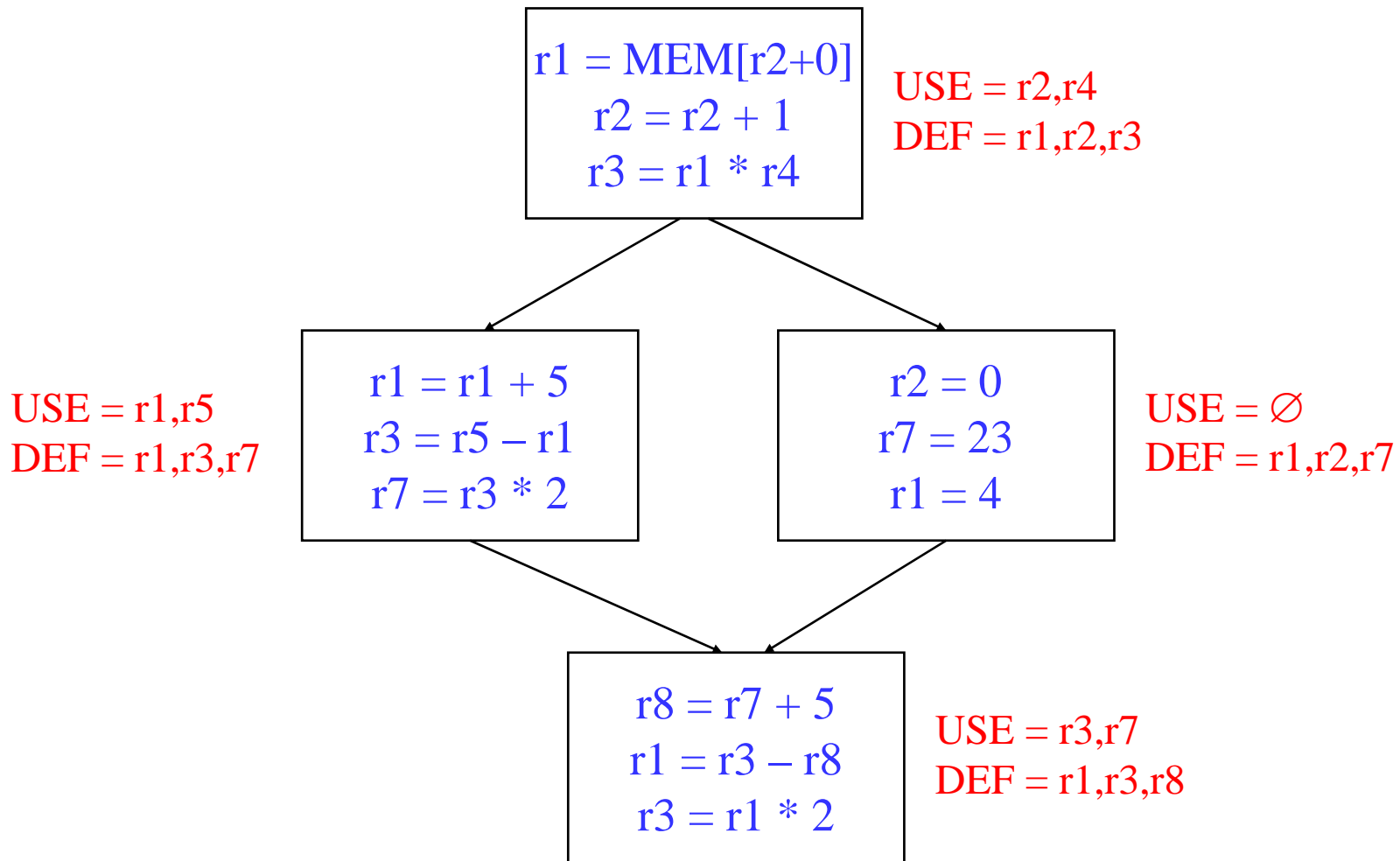
Compute IN/OUT Sets For All BBs

```
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) = USE(X) + (OUT(X) - DEF(X))
    if (old_IN != IN(X)) then
      change = 1
    endif
  endfor
endfor
```

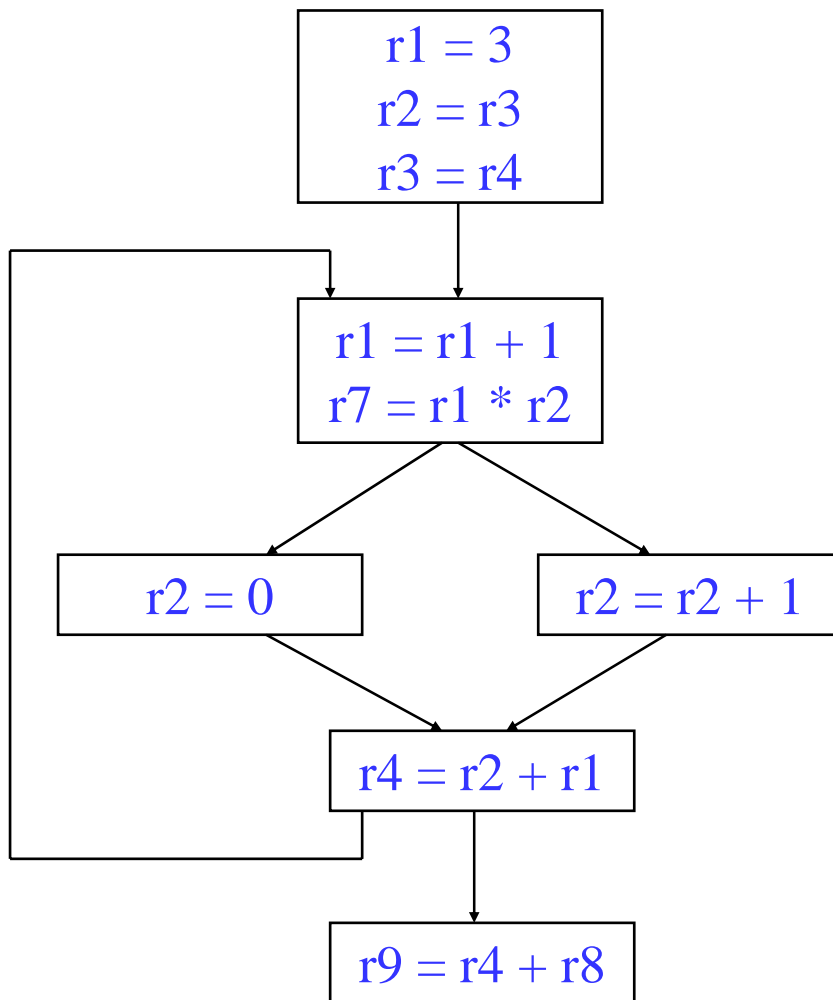
IN = set of variables that are live when the BB is entered

OUT = set of variables that are live when the BB is exited

Example IN/OUT Calculation



Class Problem

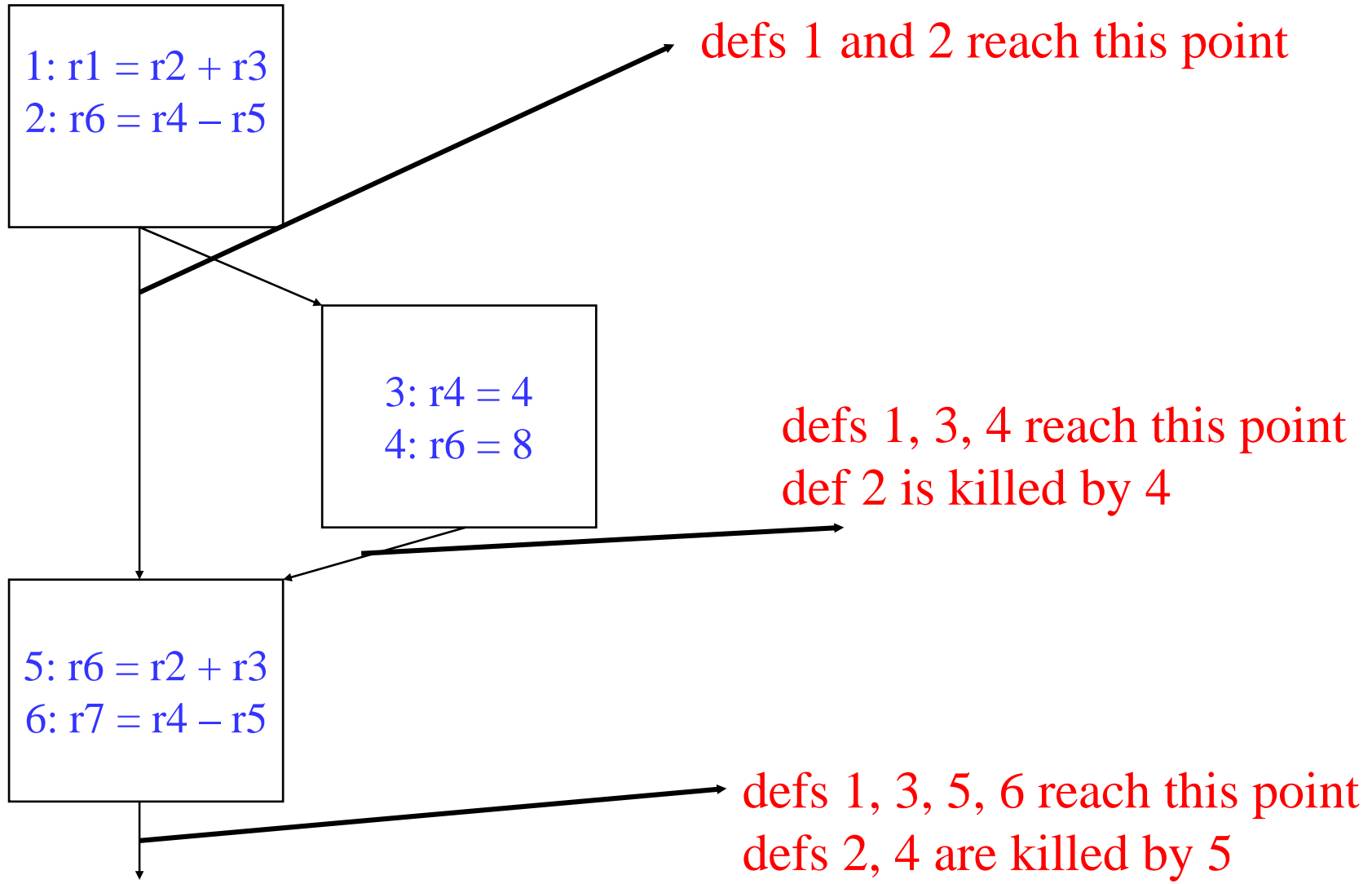


Compute liveness, ie
calculate USE/DEF
calculate IN/OUT

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 $r1$

Reaching Defs Example



Reaching Definition Analysis (rdefs)

❖ Algorithm sketch

- » Forward dataflow analysis as propagation occurs from defs downwards

❖ 4 sets

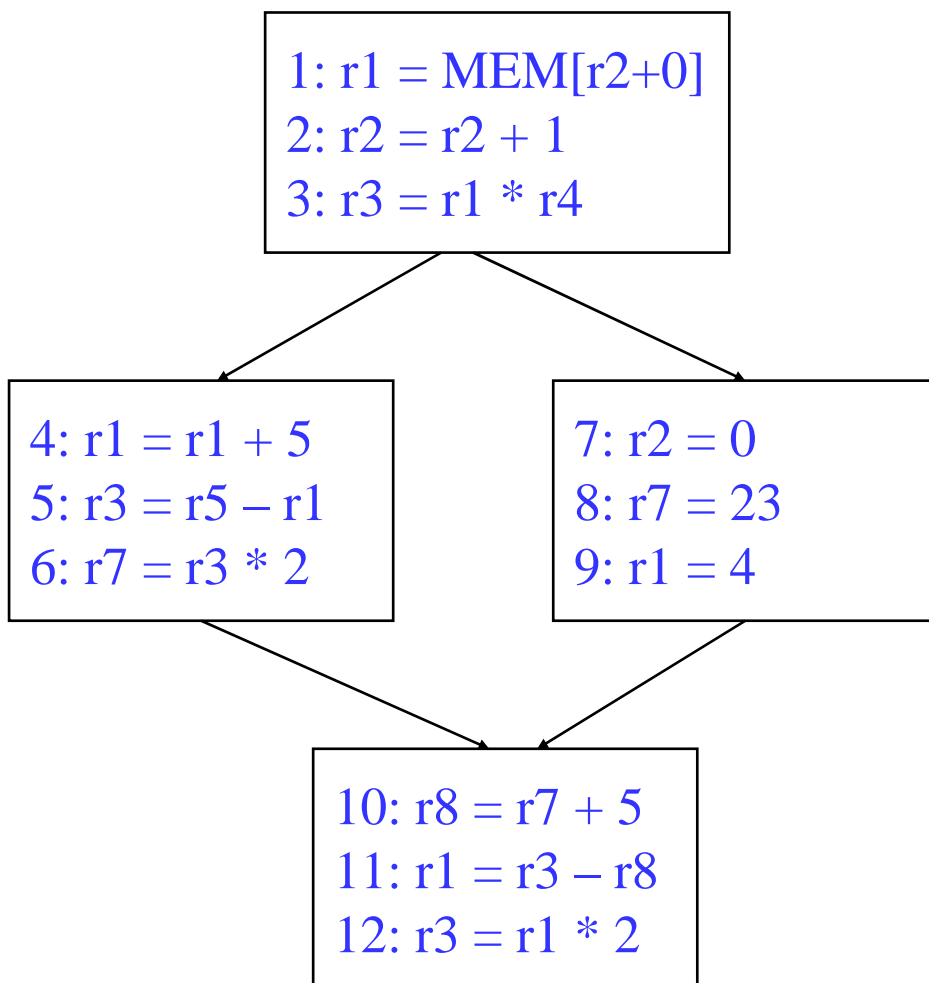
- » **GEN** = set of definitions generated in the BB
(operations not registers like liveness !!)
- » **KILL** = set of definitions killed in the BB
- » **IN** = set of definitions reaching the BB entry
- » **OUT** = set of definitions reaching the BB exit

Compute Rdef 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 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
```

gen = set of definitions created by an operation
kill = set of definitions destroyed by an operation
→ Assume each operation only has 1 destination
so just keep track of “ops”.

Example Rdef GEN/KILL Calculation

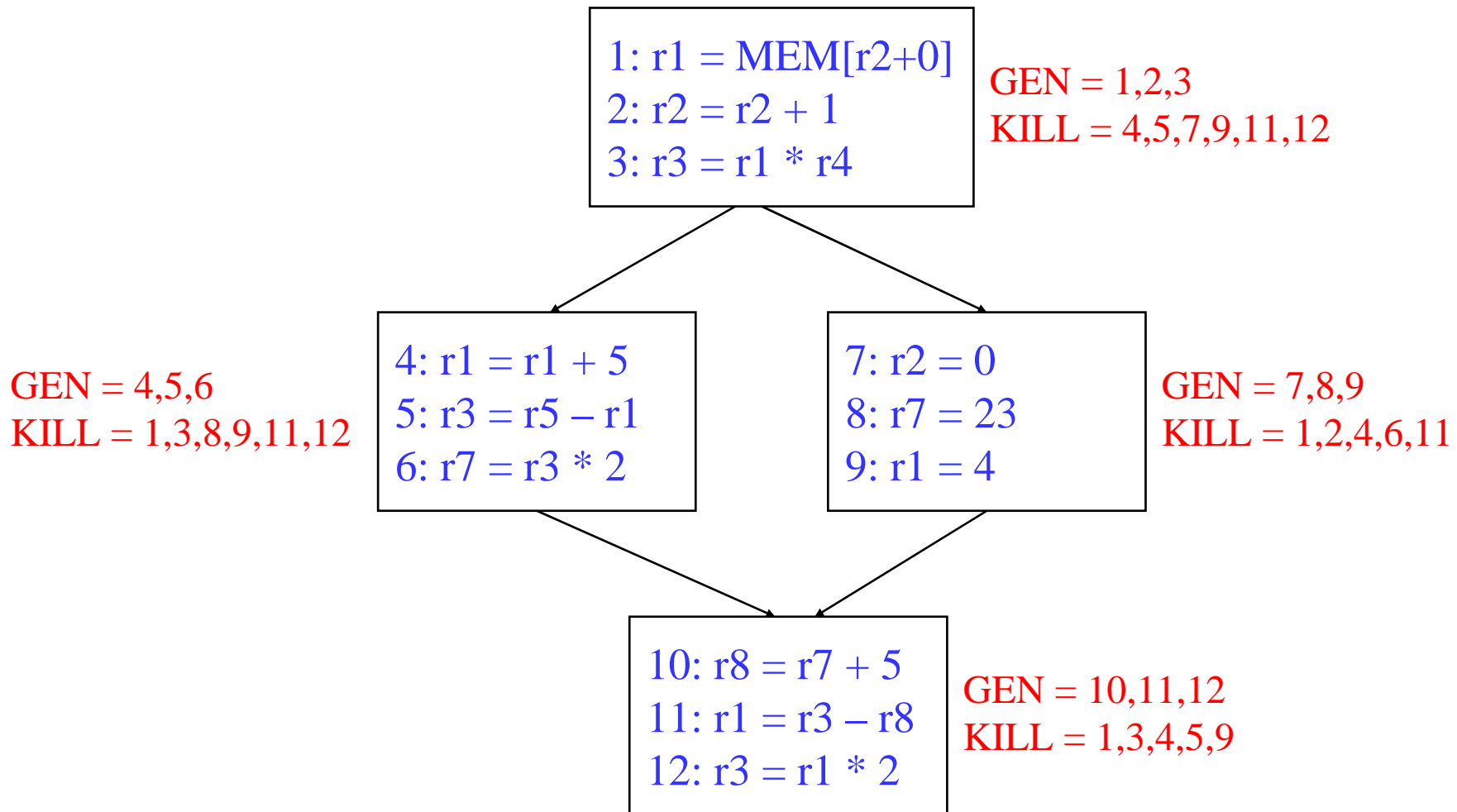


Compute Rdef IN/OUT Sets for all BBs

```
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 =  $OUT(X)$ 
     $IN(X) = \text{Union}(OUT(Y))$  for all predecessors  $Y$  of  $X$ 
     $OUT(X) = GEN(X) + (IN(X) - KILL(X))$ 
    if (old_OUT  $\neq$   $OUT(X)$ ) then
      change = 1
    endif
  endfor
endfor
```

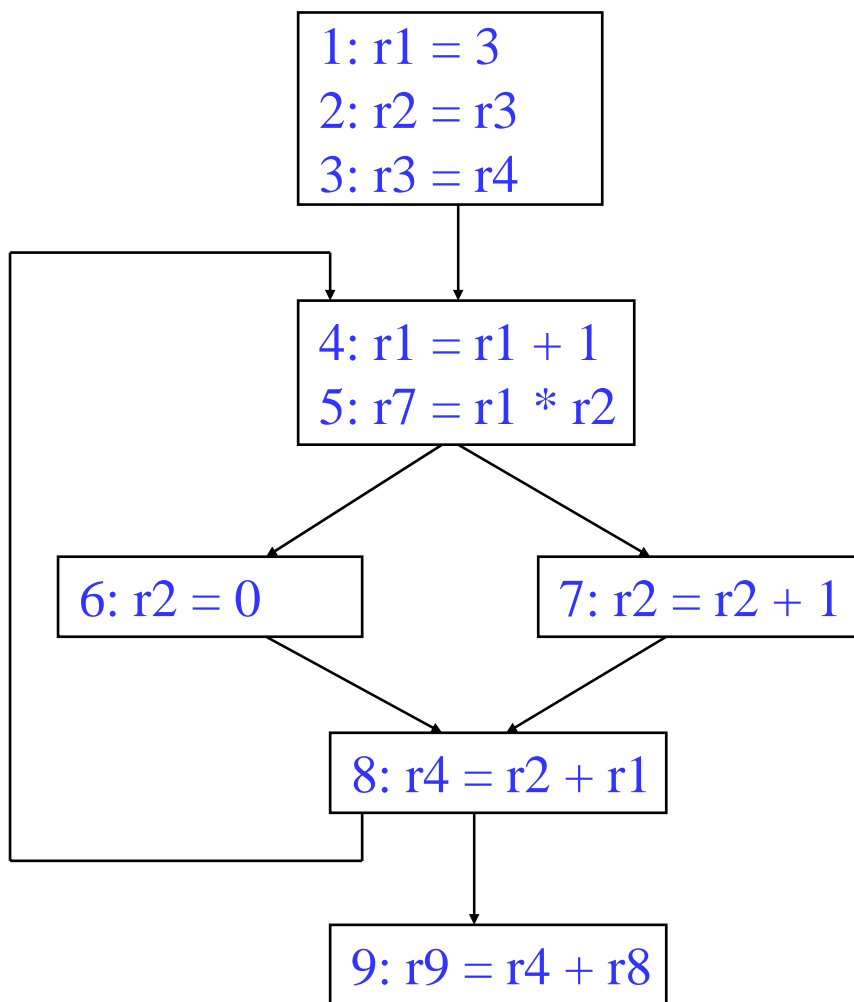
IN = set of definitions reaching the entry of BB
OUT = set of definitions leaving BB

Example Rdef IN/OUT Calculation



Class Problem

Reaching definitions
Calculate GEN/KILL
Calculate IN/OUT

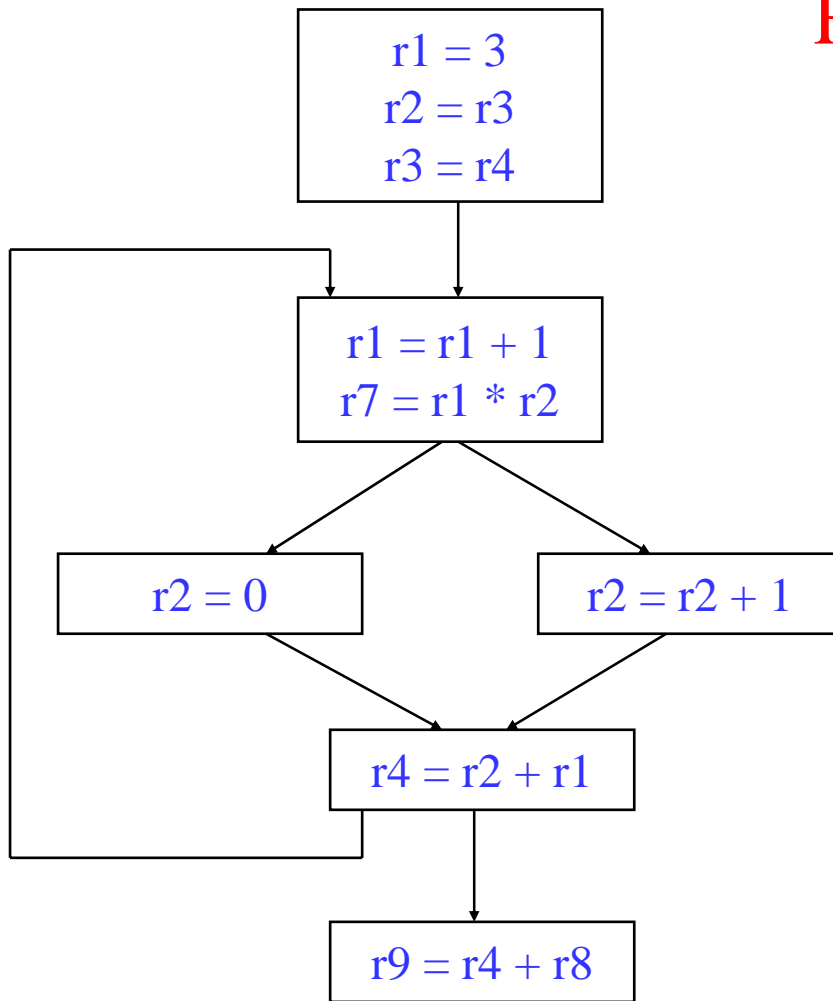


DU/UD Chains

- ❖ 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

Class Problem

Find the DU/UD Chains



Some Things to Think About

- ❖ Liveness and reaching defs are basically the same thing!!!!!!!!!!!!!!!!!!!!
 - » All dataflow is basically the same with a few parameters
 - Meaning of gen/kill (use/def)
 - 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?
 - » How to represent the info?

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
- » $IN = \text{Union}(OUT(\text{predecessors}))$ forward analysis
- » $OUT = \text{Union}(IN(\text{successors}))$ backward analysis
- » Note, this is only for “any path

Generalized Dataflow Algorithm

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

Liveness Using GEN/KILL

❖ Liveness = upward exposed uses

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

Example - Liveness with GEN/KILL

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

BB1

$r1 = \text{MEM}[r2+0]$
 $r2 = r2 + 1$
 $r3 = r1 * r4$

$up_use_GEN(1) = r2, r4$
 $up_use_KILL(1) = r1, r3$

$up_use_GEN(2) = r1, r5$
 $up_use_KILL(2) = r3, r7$

BB2

$r1 = r1 + 5$
 $r3 = r5 - r1$
 $r7 = r3 * 2$

BB3

$r2 = 0$
 $r7 = 23$
 $r1 = 4$

$up_use_GEN(3) = 0$
 $up_use_KILL(3) = r1, r2, r7$

BB4

$r3 = r3 + r7$
 $r1 = r3 - r8$
 $r3 = r1 * 2$

$up_use_GEN(4.3) = r3, r7, r8$ $up_use_KILL(4.3) = r1$
 $up_use_GEN(4.2) = r3, r8$ $up_use_KILL(4.2) = r1$
 $up_use_GEN(4.1) = r1$ $up_use_KILL(4.1) = r3$

Beyond Liveness (Upward Exposed Uses)

❖ Upward exposed defs

- » $IN = GEN + (OUT - KILL)$
- » $OUT = \text{Union}(IN(\text{successors}))$
- » Walk ops reverse order
 - $GEN += \text{dest}; KILL += \text{dest}$

❖ Downward exposed defs

- » $IN = \text{Union}(OUT(\text{predecessors}))$
- » $OUT = GEN + (IN - KILL)$
- » Walk ops forward order
 - $GEN += \text{dest}; KILL += \text{dest};$

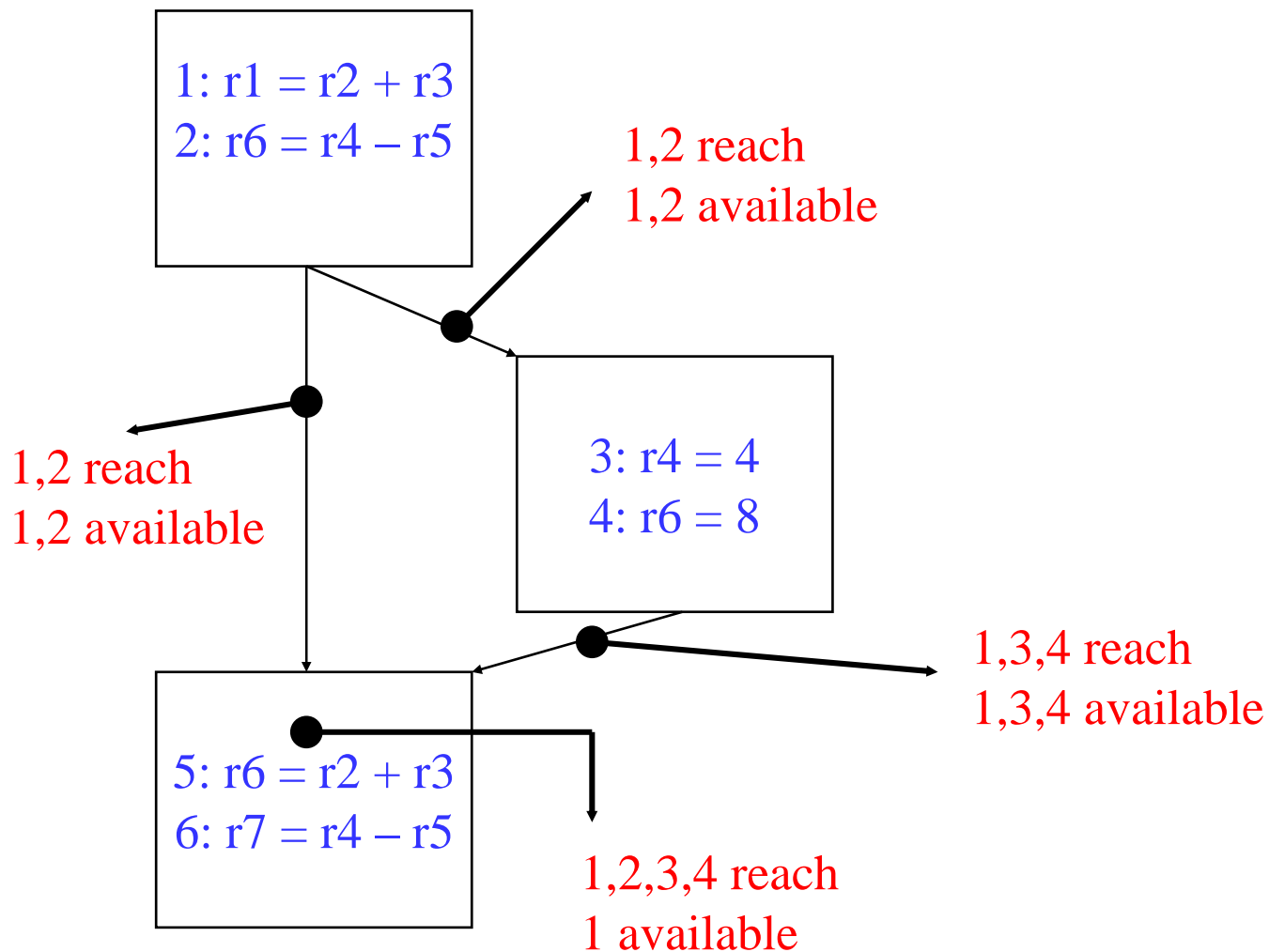
❖ Downward exposed uses

- » $IN =$
 $\text{Union}(OUT(\text{predecessors}))$
- » $OUT = GEN + (IN - KILL)$
- » Walk ops forward order
 - $GEN += \text{src}; KILL -= \text{src};$
 - $GEN -= \text{dest}; KILL += \text{dest};$

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 Adef GEN/KILL Sets

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 Adef IN/OUT Sets

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

Example: Adefs Calculation

