# How can we compute a solution to 1 and 2?

1. $in[n] = use[n] \cup (out[n] - def[n])$
2. $out[n] = \cup \{in[m] \mid m \in post[n]\}$

We need to compute a fix point

- but how can we be sure that such fix-points exist?

  It depends on the domain and on the function!

1. in[n] = use[n] ∪ (out[n] - def[n])
2. out[n] = ∪ {in[m] | m ∈ post[n]}

# Which is our domain?

Our objects:

Given a node we need to compute the set in and the set out (sets of variables)

- Let **Vars** be the finite set of variables that occur in the program P to analyze. We consider all possible subsets: $\mathcal{P}$**(Vars)**

Given a node we will need a set for in and a set for out: $\mathcal{P}$**(Vars)x** $\mathcal{P}$**(Vars)**

But we have **N** nodes, one for each node of the **CFG** so our domain will be

( $\mathcal{P}$(Vars)x $\mathcal{P}$(Vars))$^N$ : N-tuples of pairs of subsets of **Vars**

The order : $\subseteq^{2N}$

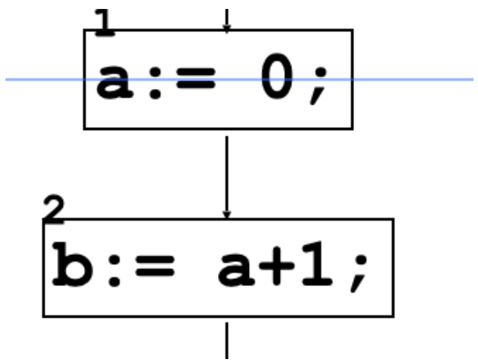
$$< in_1^1, out_1^1, \ldots, in_N^1, out_N^1 > \subseteq^{2N} < in_1^1, out_1^1, \ldots, in_N^1, out_N^1 > \text{ iff}$$

$$in_i^1 \subseteq in_i^2 \text{ and } out_i^1 \subseteq out_i^2$$

# Example

**Vars ={a,b} N=2.**

$\langle(\mathcal{P}(\textbf{Vars})\times\mathcal{P}(\textbf{Vars}))^2, \subseteq^4\rangle$ is a finite domain.



# Our domain

$\langle (\mathcal{P}(\textbf{Vars}) \times \mathcal{P}(\textbf{Vars}))^N, \subseteq^{2N} \rangle$

CPO with bottom?

It is a CPO because it is finite

bottom?

1. in[n] = use[n] ∪ (out[n] - def[n])
2. out[n] = ∪ {in[m] | m ∈ post[n]}

# Which is our function?

The map Live:

$( \mathcal{P}(\mathbf{Vars}) \times \mathcal{P}(\mathbf{Vars}) )^N \to ( \mathcal{P}(\mathbf{Vars}) \times \mathcal{P}(\mathbf{Vars}) )^N$ defined by

$Live(\langle in_1, out_1, \ldots, in_N, out_N \rangle) =$

$\langle use[1] \cup (out_1 - def[1]), \bigcup_{m \in post[1]} in_m, \ldots, use[N] \cup (out_N - def[N]), \bigcup_{m \in post[N]} in_m \rangle$

# Is it continuous?

The map Live:

$$( (\textbf{Vars}) \times \mathcal{P}(\textbf{Vars}) )^N \to (\mathcal{P}(\textbf{Vars}) \times \mathcal{P}(\textbf{Vars}) )^N$$ defined by

$$Live(\langle in_1, out_1, \ldots, in_N, out_N \rangle) =$$

$$\langle use[1] \cup (out_1 - def[1]), \bigcup_{m \in post[1]} in_m, \ldots, use[N] \cup (out_N - def[N]), \bigcup_{m \in post[N]} in_m \rangle$$

is continuous?

Yes! because it is monotone on a finite domain

# In conclusion

The map Live:

$(\mathcal{P}(\textbf{Vars}) \times \mathcal{P}(\textbf{Vars}))^N \rightarrow (\mathcal{P}(\textbf{Vars}) \times \mathcal{P}(\textbf{Vars}))^N$ defined by

$Live(\langle in_1, out_1, \ldots, in_N, out_N \rangle) =$

$$\langle use[1] \cup (out_1 - def[1]), \bigcup_{m \in post[1]} in_m, \ldots, use[N] \cup (out_N - def[N]), \bigcup_{m \in post[N]} in_m \rangle$$

is a monotonic (and therefore continuous) function on the finite CPO $\langle (\mathcal{P}(\textbf{Vars}) \times \mathcal{P}(\textbf{Vars}))^N, \subseteq^{2N} \rangle$ and therefore Live has

a least fixpoint

# Why a least fixpoint

- Live is a possible analysis,

$$in[n] \supseteq live\text{-}in[n] \text{ and } out[n] \supseteq live\text{-}out[n]$$

i.e., if a variable x will be really live in a node n during some program execution then x belongs to in[n] of all the fixpoints of the function Live

All fixpoints of the equation system is an over-approximation of really live variables.

We want the least fixpoint ( more precise over approximations)

# Conservative Approximation

- How to interpret the output of this static analysis?
- Correctness tells us that:

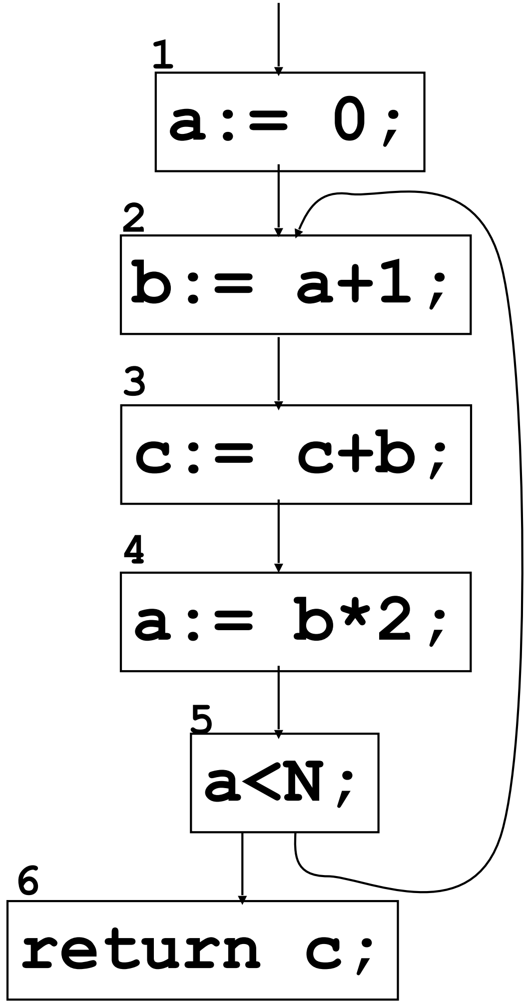
**in[n] ⊇ live-in[n]** and **out[n] ⊇ live-out[n]**

If the variable x will be really live in some node n during some program execution then x belongs to in[n] of all the fixpoints of the function Live (least fixpoint)

- The converse does not hold: the analysis can tell us that x is in the computed set out[n], but this does not imply that x will be necessarily live in n during some program execution
- In liveness analysis "conservative approximation" means that the analysis may erroneously derive that a variable is live, while the analysis is not allowed to erroneously derive that a variable is "dead" (i.e., not live).
  - ★if x ∈ in[n] then x could be live at program point n.
  - ★if x∉in [n] then x is definitely dead at program point n.

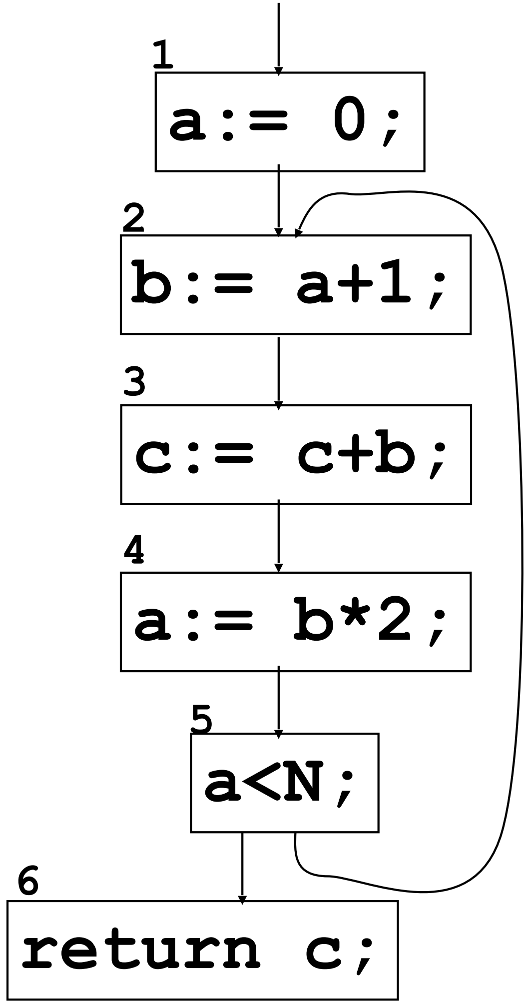
```
for all n
    in[n]:={} out[n]:={};
repeat
    for all n (1 to 6)
          in'[n]:=in[n]; out'[n]:=out[n];
          in[n]:= use[n] U (out[n] - def[n]);
          out[n]:= U { in[m] | m ∈ post[n]};
until (for all n: in'[n]=in[n] && out'[n]=out[n])
```

| | | | Live1 | | Live2 | | Live3 | |
|---|---|---|---|---|---|---|---|---|
| | use | def | in | out | in | out | in | out |
| 1 | | a | | | | a | | a |
| 2 | a | b | a | | a | b c | a c | b c |
| 3 | b c | c | b c | | b c | b | b c | b |
| 4 | b | a | b | | b | a | b | a |
| 5 | a | | a | a | a | a c | a c | a c |
| 6 | c | | c | | c | | c | |

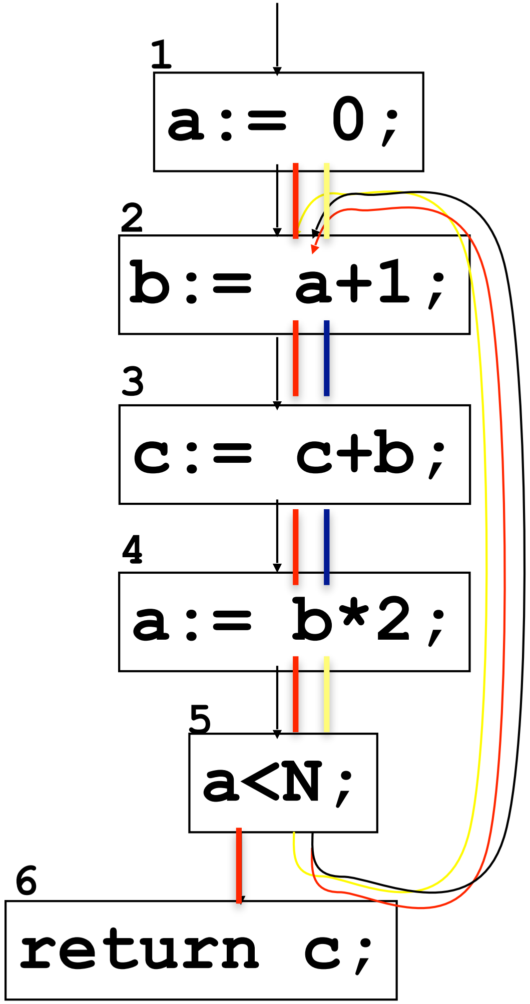


```
for all n
   in[n]:=?; out[n]:=?;
repeat
   for all n (1 to 6)
        in'[n]:=in[n]; out'[n]:=out[n];
        in[n]:= use[n] U (out[n] - def[n]);
        out[n]:= U { in[m] | m ∈ post[n]};
until (for all n: in'[n]=in[n] && out'[n]=out[n])
```

| | | | Live3 | | Live4 | | Live5 | |
|---|---|---|---|---|---|---|---|---|
| | use | def | in | out | in | out | in | out |
| 1 | | a | | a | | a c | c | a c |
| 2 | a | b | a c | b c | a c | b c | a c | b c |
| 3 | b c | c | b c | b | b c | b | b c | b |
| 4 | b | a | b | a | b | a c | b c | a c |
| 5 | a | | a c | a c | a c | a c | a c | a c |
| 6 | c | | c | | c | | c | |



| | | | Live5 | | Live6 | | Live7 | |
|---|---|---|---|---|---|---|---|---|
| | use | def | in | out | in | out | in | out |
| 1 | | a | c | a c | c | a c | c | a c |
| 2 | a | b | a c | b c | a c | b c | a c | b c |
| 3 | b c | c | b c | b | b c | b c | b c | b c |
| 4 | b | a | b c | a c | b c | a c | b c | a c |
| 5 | a | | a c | a c | a c | a c | a c | a c |
| 6 | c | | c | | c | | c | |




The algorithm thus gives the following output:

out[1]={a,c}, out[2]={b,c}, out[3]={b,c}, out[4]={a,c}, out[5]={a,c}

In this case, the output of the analysis is precise

# Improvement

In this iterative computation, observe that we have to wait for the next iteration in order to exploit the new information computed for in and out on the nodes.

By a suitable reordering of the nodes and by first computing out[n] and then in[n], we are able to converge to the fixpoint in just 3 iteration steps.

```
for all n
   in[n]:=?; out[n]:=?;
repeat
   for all n (6 to 1)
         in'[n]:=in[n]; out'[n]:=out[n];
         out[n]:= U { in[m] | m ∈ post[n]};
         in[n]:= use[n] U (out[n] - def[n]);
until (for all n: in'[n]=in[n] && out'[n]=out[n])
```

```
for all n
   in[n]:=?; out[n]:=?;
repeat
   for all n (6 to 1)
        in'[n]:=in[n]; out'[n]:=out[n];
        out[n]:= U { in[m] | m ∈ post[n]};
        in[n]:= use[n] U (out[n] - def[n]);
until (for all n: in'[n]=in[n] && out'[n]=out[n])
```

| | | | Live1 | | Live2 | | Live3 | |
|---|---|---|---|---|---|---|---|---|
| | use | def | out | in | out | in | out | in |
| 6 | c | | | c | | c | | c |
| 5 | a | | c | a c | a c | a c | a c | a c |
| 4 | b | a | a c | b c | a c | b c | a c | b c |
| 3 | b c | c | b c | b c | b c | b c | b c | b c |
| 2 | a | b | b c | a c | b c | a c | b c | a c |
| 1 | | a | ac | c | ac | c | ac | c |

# Backward Analysis

As shown by the previous example, Live Variable Analysis is a "backward" analysis. This means that information propagates "backward" from terminal nodes to initial nodes:

1. in[n] can be computed from out[n];
2. out[n] can be computed from in[m] for all the nodes m that are successors of n.

# Application: Dead Code Elimination

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   t2 := t3;
   while j <= m do
      t1 := t3 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1;
   t3 := t3 + (m+1)
od
```

dead variable

```
i := 0;
t3 := 0;
while i <= n do
   j := 0;
   while j <= m do
      t1 := t3 + j;
      temp := Base(A) + t1;
      Cont(temp) := Cont(Base(B) + t1)
                  + Cont(Base(C) + t1);
      j := j+1
   od;
   i := i+1;
   t3 := t3 + (m+1)
od
```

# Reaching Definitions (Reaching Assignment) Analysis

One of the more useful data-flow analysis

```
d1 : y := 3
d2 : x := y
```

d1 is a reaching definition for d2

```
d1 : y := 3
d2 : y := 4
d3 : x := y
```

d1 is no longer a reaching definition for d3, because d2 kills its reach: the value defined in d1 is no longer available and cannot reach d3

A definition d at point i reaches a point p if there is a path from the point i to p such that d is not killed (redefined) along that path

# Reaching definitions

This information is very useful

- The compiler can  know whether x is a constant at point p
- The debugger can tell whether is possible that x is an undefined variable at point p

# Reaching definitions

- Given a program point n, which definitions are actual – not successively overwritten by a different assignment – when the execution reaches n?

  And when the execution leaves n?

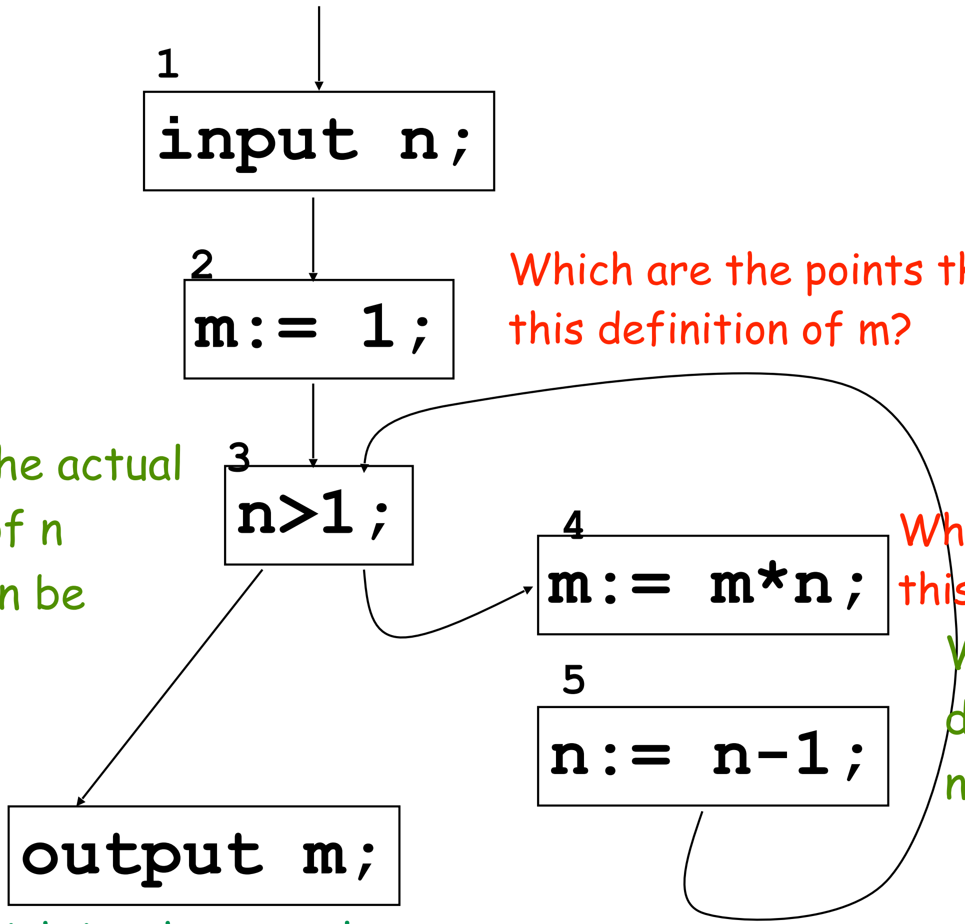
- A program point may clearly "generate" new definitions

- A program point n may "kill" a definition:

  if n is an assignment x:=exp then n kills all the assignments to the variable x which are actual in input to n

- We are thus interested in computing input and output reaching definitions for any program point

# The intuition: the factorial of n




Which are the points that are reached by this definition of m?

Which is the actual definition of n here? Can n be initialised?

Which are the points that are reached by this definition of m?

Which are the actual definition of n and m ? Can n or m be initialised?

Which is the actual definition of m here? Can m be initialised?

# Formalization of the reaching definition property

- The property can be represented by sets of pairs:
  $\{(x,p) \mid x \in$ **Vars**, p is a program point$\} \in \mathcal{P}($**Vars x Points**$)$
  where (x,p) means that the variable x is assigned at
  program point p
- For each program point, this dataflow analysis computes a set of such pairs
- The meaning of a pair (x,p) in the set for a program point q is that the assignment of x at point p is actual at point q
- ? is a special symbol that we add to **Points** and we use to represent the fact that a variable x is not initialized.
- The set $\iota = \{(x,?) \mid x \in$ **Vars**$\}$ therefore denotes that all the program variables are not initialized.

# The domain for Reaching Definitions Analysis

**Vars** is the (finite) set of variables occuring in the program P.

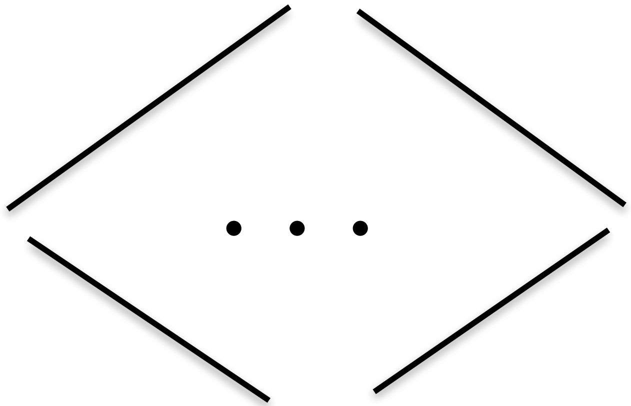
Let N be the number of nodes of the CFG of P.

Let **Points={?,1,...N}.**

$$\langle (\mathcal{P}(\textbf{Vars} \times \textbf{Points}) \times \mathcal{P}(\textbf{Vars} \times \textbf{Points}))^{N}, \subseteq^{2N} \rangle$$

- Example Vars={a,b} e N=2

$$< S = \{(a,?),(a,1),(b,?),(b,1)\}, S, S, S >$$



$$< \emptyset, \emptyset, \emptyset, \emptyset >$$

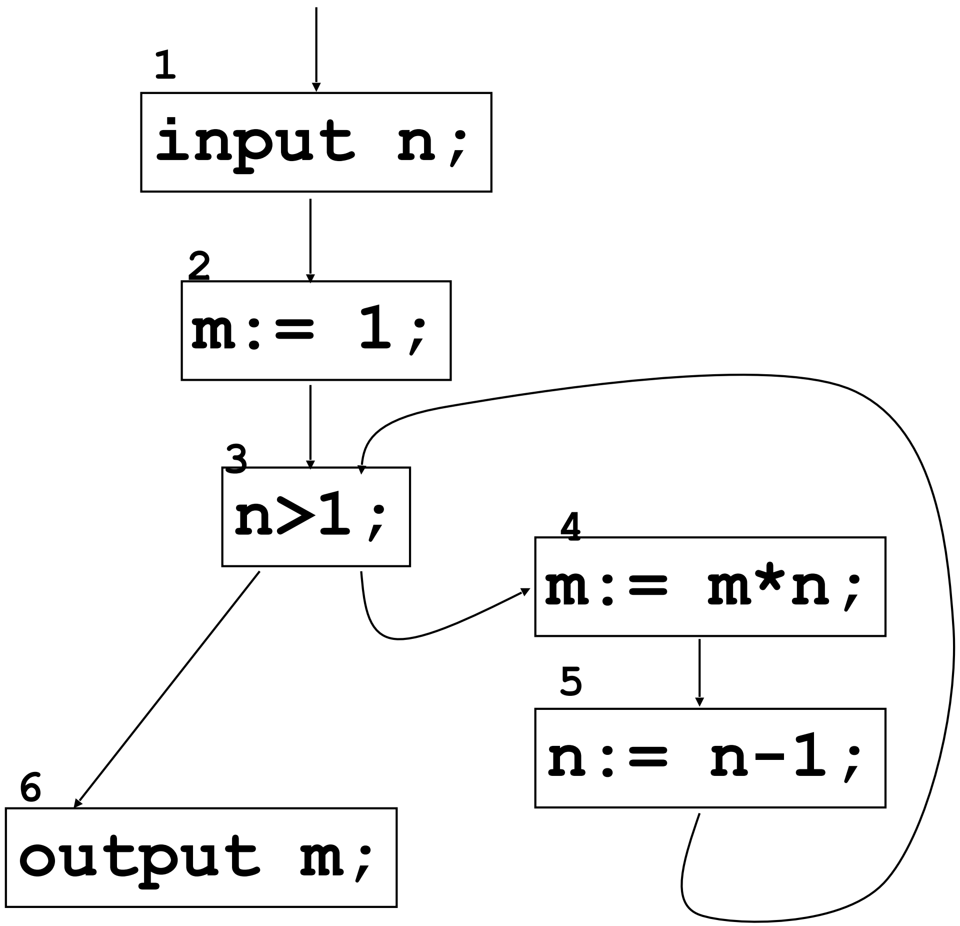
# Specification

- $kill_{RD}[p] = \begin{cases} \{(x,q) \mid q \in \textbf{Points} \text{ and } \{x\}=def[q]\} & \text{if } \{x\}=def[p] \\ \emptyset & \text{if } \emptyset=def[p] \end{cases}$

- $gen_{RD}[p] = \begin{cases} \{(x,p)\} & \text{if } \{x\}=def[p] \\ \emptyset & \text{if } \emptyset=def[p] \end{cases}$

As usual, def[p] = {x} when the command in the point p is an assignment x:=exp

# Kill and Gen




| | $kill_{RD}$ | $gen_{RD}$ |
|---|---|---|
| 1 | | |
| 2 | (m,?)(m,2)(m,4) | (m,2) |
| 3 | | |
| 4 | (m,?)(m,2)(m,4) | (m,4) |
| 5 | (n,?) (n,5) | (n,5) |
| 6 | | |

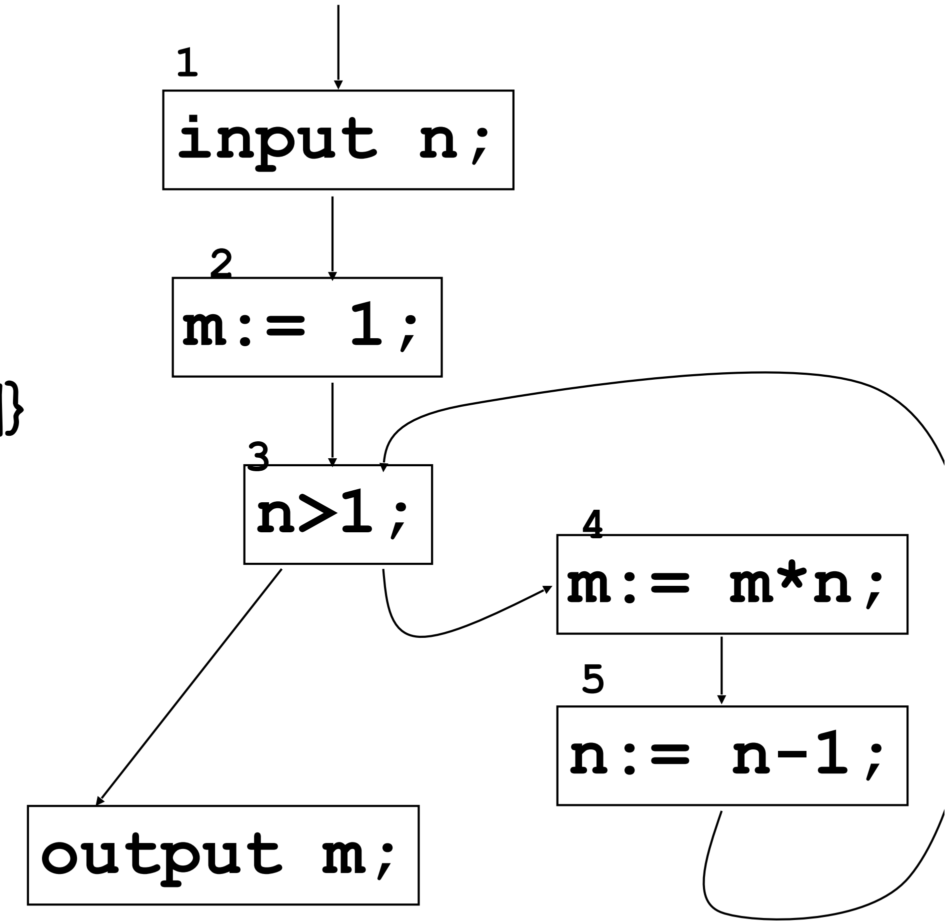
# Specification

- Reaching definitions analysis is specified by equations:

$$RD_{entry}(p) = \begin{cases} \{(x,?) \mid x \in VARS\} & \text{if p is initial} \\ \bigcup\{RD_{exit}(q) \mid q \in pre[p]\} & \text{if p is not initial} \end{cases}$$

$RD_{exit}(p) =$

$(RD_{entry}(p) \setminus kill_{RD}[p]) \cup gen_{RD}[p]$



# The solution of the previous system

Once again the solution for the equations in the previous system requires the existence of a fix point

We can apply the Kleene theorem if we have

a) a continuous function on

b) a CPO with bottom

# Point b

$$\langle (\mathcal{P}(\text{Vars} \times \text{Points}) \times \mathcal{P}(\text{Vars} \times \text{Points}))^{N}, \subseteq^{2N} \rangle$$

is a CPO with bottom?

It is a CPO because it is finite

Bottom?

# Point a: the function

The map Reach:

$<\mathcal{P}(Vars \times Points) \times \mathcal{P}(Vars \times Points)>^N \to <\mathcal{P}(Vars \times Points) \times \mathcal{P}(Vars \times Points)>^N$

defined by

(assuming 1 is the only initial node)

$Reach(<RDentry_1, RDexit_1, \ldots, RDentry_N, RDexit_N>) =$

$< \{(x,?) \mid x \text{ in } VARS\}, (RD_{entry1} \setminus kill_{RD}[1]) \cup gen_{RD}[1],$

$\cup\{RD_{exit2} \mid m \text{ in } pre[2]\}, (RD_{entry2} \setminus kill_{RD}[2]) \cup gen_{RD}[2],$

….,

$\cup\{RD_{exitm} \mid m \text{ in } pre[N]\}, (RD_{entryN} \setminus kill_{RD}[N]) \cup gen_{RD}[N] >$

# Point a

Reach(<$RDentry_1$,$RDexit_1$,….,$RDentry_N$,$RDexit_N$>)=

< {(x,?) | x in VARS}, ($RD_{entry1} \backslash kill_{RD}[1]$ ) U $gen_{RD}[1]$,

U{$RD_{exit2}$ |m in pre[2]} , ($RD_{entry2} \backslash kill_{RD}[2]$ ) U $gen_{RD}[2]$

….,

U{$RD_{exitm}$ |m in pre[N]} , ($RD_{entryN} \backslash kill_{RD}[N]$ ) U $gen_{RD}[N]$ >

$kill_{RD}(1)$={(a,?)}, $gen_{RD}(1)$={(a,1)}

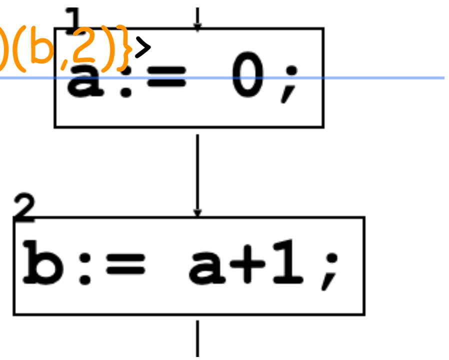
$kill_{RD}(2)$={(b,?)}, $gen_{RD}(2)$={(b,2)}

- Example

Reach(<{(a,?)}, {}, {},{} )=<{(a,?)(b,?)},{(a,1)(b,?)},{(a,1)(b,?)},{(a,1)(b,2)}>

Reach(<{(a,?)(a,2)},{(a,2)},{},{(b,1)}>)=

<{(a,?)(b,?)},{(a,1)(b,?)},{(a,1)(b,?)},{(a,1),(b,2)}>




Note that Reach is monotone!

Since it is monotone on a finite domain then it is continuous

# Why a least fix point

RD analysis is possible,

if an assignment x:=a in some point q is really actual in entry to some point p then

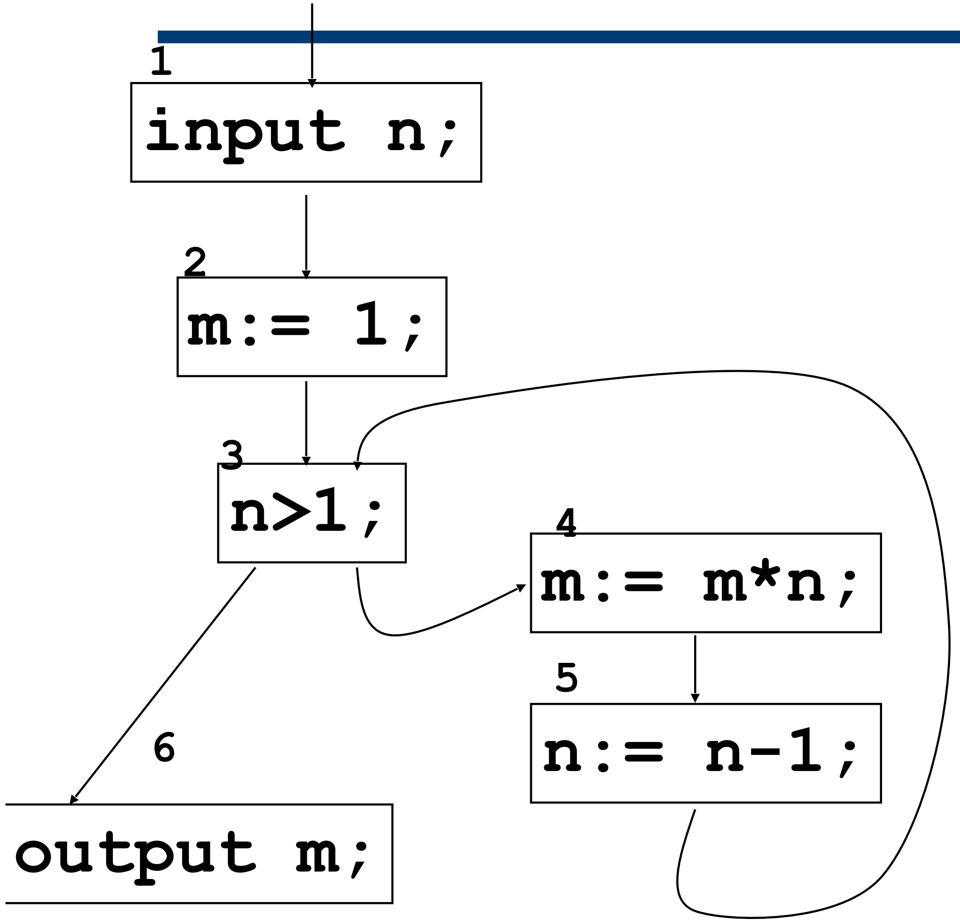
$(x,q) \in RD_{entry}(p)$

The vice versa does not hold

All fixpoints of the above equation system is an over-approximation of really reaching definitions.

Computing the least fixpoint gives a more precise over approximation

# First iteration:




| | kill | gen |
|---|---|---|
| 2 | (m,?)(m,2) (m,4) | (m,2) |
| 4 | (m,?)(m,2) (m,4) | (m,4) |
| 5 | (n,?) (n,5) | (n,5) |

$RD_{entry}(1)= \{(n,?),(m,?)\}$

$RD_{exit}(1) = \{(n,?),(m,?)\}$

$RD_{entry}(2)= \{(n,?),(m,?)\}$

$RD_{exit}(2)= \{(n,?),(m,2)\}$

$RD_{entry}(3)=\{(n,?),(m,2)\}$

$RD_{exit}(3)=\{(n,?),(m,2)\}$

$RD_{entry}(4)= \{(n,?),(m,2)\}$

$RD_{exit}(4)= \{(n,?), (m,4)\}$

$RD_{entry}(5)= \{(n,?),(m,4)\}$

$RD_{exit}(5)= \{(n,5),(m,4)\}$

$RD_{entry}(6)= \{(n,?),(m,2)\}$

$RD_{exit}(6)= \{(n,?),(m,2)\}$

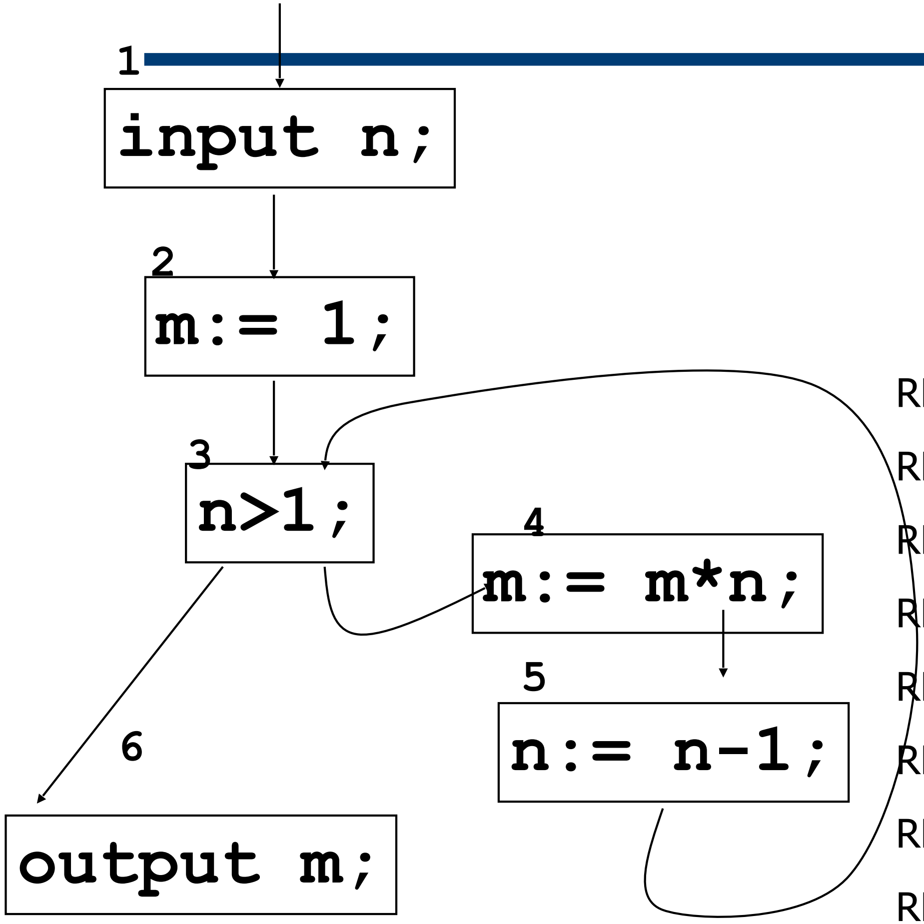
$RD_{entry}(p) =\{(x,?) \mid x \text{ in } Vars\}$, if p is initial

$RD_{entry}(p) =\cup\{RD_{exit}(q) \mid q \text{ in } pre[p]\}$, otherwise

$RD_{exit}(p) = (RD_{entry}(p) \setminus kill_{RD}[p]) \cup gen_{RD}[p]$

# Second iteration:

| | | |
|---|---|---|
| 2 | (m,?)(m,2) (m,4) | (m,2) |
| 4 | (m,?)(m,2) (m,4) | (m,4) |
| 5 | (n,?) (n,5) | (n,5) |




$RD_{entry}(p) = \{(x,?) \mid x \text{ in Vars}\}$, if p is initial

$RD_{entry}(p) = \cup\{RD_{exit}(q) \mid q \text{ in } pre[p]\}$, otherwise

$RD_{exit}(p) = (RD_{entry}(p) \setminus kill_{RD}[p]) \cup gen_{RD}[p]$

| | |
|---|---|
| $RD_{entry}(1) = \{(n,?),(m,?)\}$ | $RD_{entry}(1) = \{(n,?),(m,?)\}$ |
| $RD_{exit}(1) = \{(n,?),(m,?)\}$ | $RD_{exit}(1) = \{(n,?),(m,?)\}$ |
| $RD_{entry}(2) = \{(n,?),(m,?)\}$ | $RD_{entry}(2) = \{(n,?),(m,?)\}$ |
| $RD_{exit}(2) = \{(n,?),(m,2)\}$ | $RD_{exit}(2) = \{(n,?),(m,2)\}$ |
| $RD_{entry}(3) = \{(n,?),(m,2)\}$ | $RD_{entry}(3) = \{(n,?),(m,2),(n,5)(m,4)\}$ |
| $RD_{exit}(3) = \{(n,?),(m,2)\}$ | $RD_{exit}(3) = \{(n,?),(m,2),(n,5)(m,4)\}$ |
| $RD_{entry}(4) = \{(n,?),(m,2)\}$ | $RD_{entry}(4) = \{(n,?),(m,2),(n,5)(m,4)\}$ |
| $RD_{exit}(4) = \{(n,?), (m,4)\}$ | $RD_{exit}(4) = \{(n,?),(n,5)(m,4)\}$ |
| $RD_{entry}(5) = \{(n,?),(m,4)\}$ | $RD_{entry}(5) = \{(n,?),(n,5)(m,4)\}$ |
| $RD_{exit}(5) = \{(n,5),(m,4)\}$ | $RD_{exit}(5) = \{(n,5),(m,4)\}$ |
| $RD_{entry}(6) = \{(n,?),(m,2)\}$ | $RD_{entry}(6) = \{(n,?),(m,2),(n,5)(m,4)\}$ |
| $RD_{exit}(6) = \{(n,?),(m,2)\}$ | $RD_{exit}(6) = \{(n,?),(m,2),(n,5)(m,4)\}$ |

fix point!

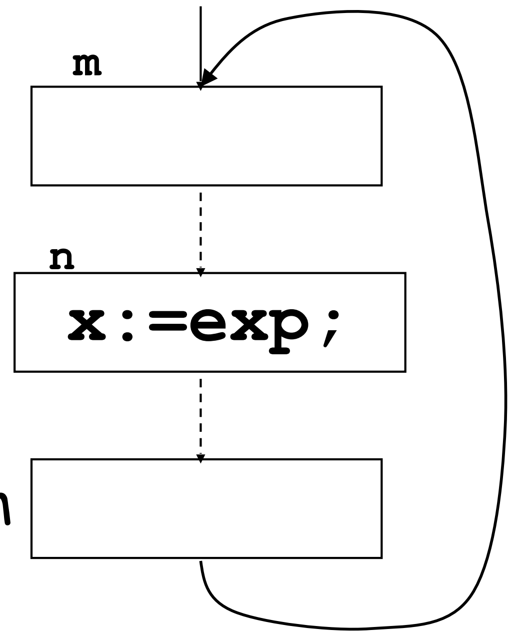
# RD analysis

- RD analysis is forward and possible,
  i.e., if an assignment x:=a in some point q is really actual in entry to some point p then
  $(x,q) \in RD_{entry}(p)$ (while the vice versa does not hold).

How can we use this?

-If the analysis tells us that a variable is undefined then it is
-Loop invariant code motions

# Application: Loop invariant code motion




Consider a loop where:

1. m is the entry point
2. an inner point n contains an assignment x:=exp
3. if for any variable y occurring in exp (i.e. $y \in$ **vars**(exp)) and for any program point p, we have that

$$(y,p) \in RD_{\texttt{entry}}(m) \iff (y,p) \in RD_{\texttt{entry}}(n)$$

then, the assignment x:=exp can be correctly moved out as preceding the entry point of the loop

# Loop-invariant code motion

```
y:=3; z:=5;
for(int i=0; i<9; i++) {
  x = y + z;
  a[i] = 2*i + x;
}
```

```
y:=3; z:=5;
x = y + z;
for(int i=0; i<9; i++) {
  a[i] = 2*i + x;
}
```

# Available Expressions Analysis

Let p be a program point. For each execution path ending in p, we want track the expressions that have already been evaluated and then not modified.

These are called available expressions

# Example

x:=a+b;

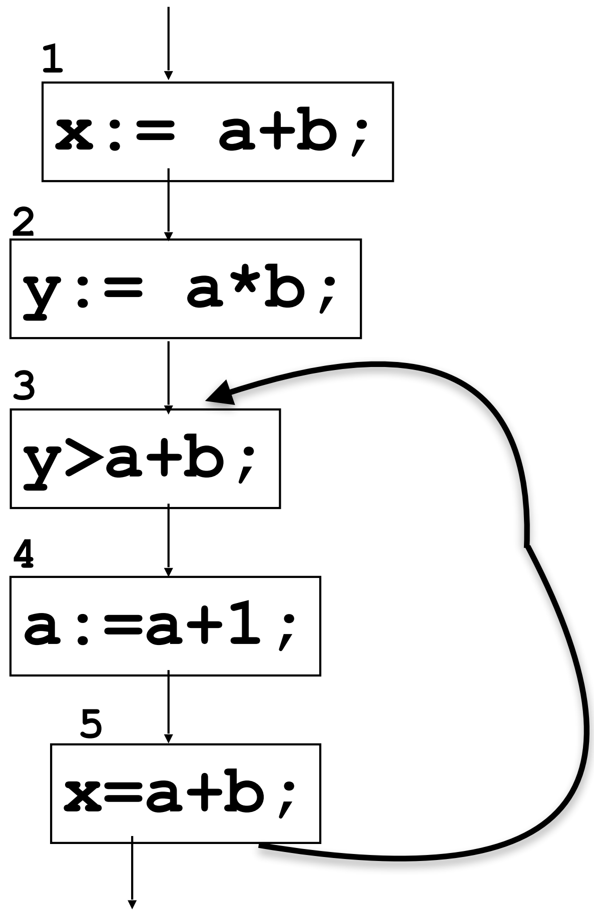
y:=a*b;

while y>a+b

do (a:=a+1;

x:=a+b;)

when the execution reaches 3, the expression a+b is available, since it has been previously evaluated (in point 1 for the first iteration of the while-loop and in point 5 for the next iterations) and does not need to be evaluated again in 3




- This analysis can be therefore used to avoid re-evaluations of available expressions

# The domain

Let **E**={ e | e is a sub-expressions/expression appearing in P}

Let N be the number of nodes of the CFG of P

$\langle (\mathcal{P}(\mathbf{E}) \times \mathcal{P}(\mathbf{E}))^N, \subseteq^{2N} \rangle$ is a finite domain

## $Kill_{AE}$ and $Gen_{AE}$

- An expression e in E is killed in a program point p (e is in $kill_{AE}(p)$) if a variable occurring in e is modified (i.e., it is defined by some assignment) by the command in p.

$$kill_{AE}([x:=e']^p) = \{e \text{ in } E \mid x \in \mathbf{vars}(e)\}$$

- An expression e is generated in a program point p (e is in $gen_{AE}(p)$) if e is evaluated in p and no variable occurring in e is modified in p.

$$gen_{AE}([x:=e]^p) = \{e\} \qquad \text{if } x \notin \mathbf{vars}(e),$$

$$gen_{AE}([x:=e]^p) = \emptyset \qquad \text{if } x \in \mathbf{vars}(e);$$

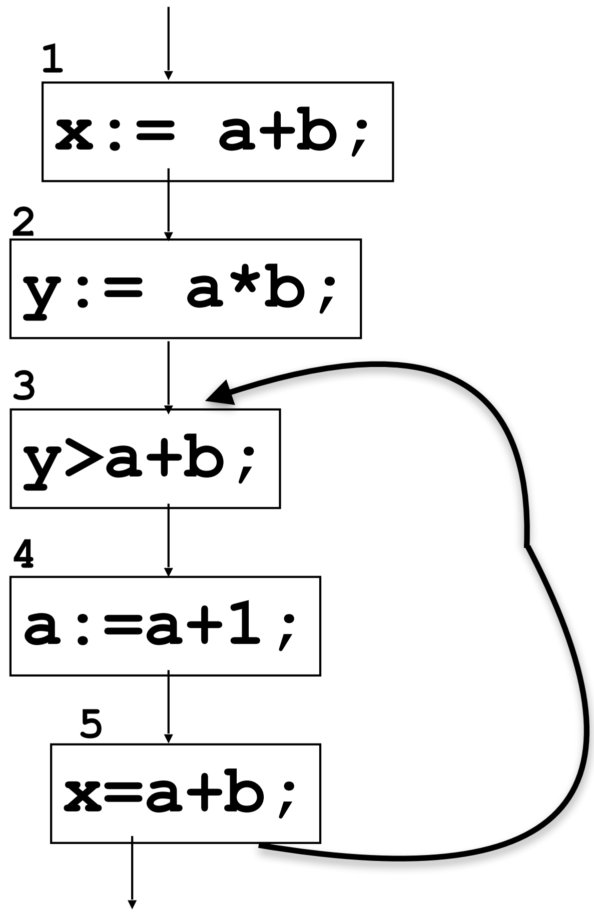
$gen_{AE}([e1>e2]^p) = expr(\{e1, e2\})$ where expr(S) returns the subset of S that are expressions

# Example

x:=a+b; y:=a*b; while y>a+b do (a:=a+1; x:=a+b)

**E** = {a+b, a*b, a+1}

| n | $kill_{AE}(n)$ | $gen_{AE}(n)$ |
|---|---|---|
| 1 | ∅ | {a+b} |
| 2 | ∅ | {a*b} |
| 3 | ∅ | {a+b} |
| 4 | {a+b, a*b,a+1} | ∅ |
| 5 | ∅ | {a+b} |



# Specification

- Available expressions analysis is specified by the following equations, for any program point p:

$$AE_{entry}(p) = \begin{cases} \varnothing & \text{if } p \text{ is initial} \\ \cap\{AE_{exit}(q) \mid q \in pre[p]\} & \text{otherwise} \end{cases}$$

$$AE_{exit}(p) = (AE_{entry}(p) \setminus kill_{AE}(p)) \cup gen_{AE}(p)$$

# Point a and b to apply Kleene Theorem

To find a solution to the previous equation system we need to apply Kleene Theorem

b) $\langle (\mathcal{P}(E) \times \mathcal{P}(E))^N, \subseteq^{2N} \rangle$ is a finite domain therefore is a CPO, moreover, it has a bottom element

a) The map $(\mathcal{P}(E) \times \mathcal{P}(E))^N \rightarrow (\mathcal{P}(E) \times \mathcal{P}(E))^N$ defined by

(assuming 1 is the only initial node)

$AE(\langle AE_{entry1}, AE_{exit1}, \ldots, AE_{entryN}, AE_{exitN} \rangle) =$

$\langle \emptyset, (AE_{entry1} \setminus kill_{AE}(1)) \cup gen_{AE}(1),$

$\cap\{AE_{exitq} \mid q \text{ in } pre[2]\}, (AE_{entry2} \setminus kill_{AE}(2)) \cup gen_{AE}(2),$

.....

$\cap\{AE_{exitq} \mid q \text{ in } pre[N]\}, (AE_{entryN} \setminus kill_{AE}(N)) \cup gen_{AE}(N) \rangle$

# Point a

a) The map

$AE(\langle AE_{entry1}, AE_{exit1}, \ldots, AE_{entryN}, AE_{exitN} \rangle) =$

$\langle \emptyset, (AE_{entry1} \setminus kill_{AE}(1)) \cup gen_{AE}(1),$

$\cap\{AE_{exitq} \mid q \text{ in } pre[2]\}, (AE_{entry2} \setminus kill_{AE}(2)) \cup gen_{AE}(2),$

.....

$\cap\{AE_{exitq} \mid q \text{ in } pre[N]\}, (AE_{entryN} \setminus kill_{AE}(N)) \cup gen_{AE}(N) \rangle$

is monotone on the finite domain $\langle (\mathcal{P}(\mathbf{E}) \times \mathcal{P}(\mathbf{E}))^N, \subseteq^{2N} \rangle$

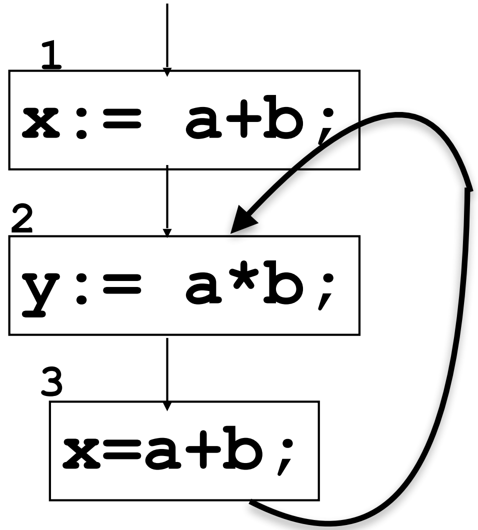
- Example

$AE(\langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle) =$

$\langle \emptyset, \{a+b\}, \{\}, \{a*b\}, \{a*b\}, \{a+b, a*b\} \rangle$

$AE(\langle \emptyset, \{a+b\}, \{\}, \{a*b\}, \{a*b\}, \{a+b, a*b\} \rangle) =$

$\langle \emptyset, \{a+b\}, \{a+b\}, \{a+b, a*b\}, \{a+b, a*b\}, \{a+b, a*b\} \rangle$



# Which fix point?

AE is a definite analysis:

if $e \in AE_{entry}(p)$ then e is really available in entry to p

the converse does not hold

- Any fixpoint of the above equation system is an under-approximation of really available expressions.

Between all fix points, we are thus interested in computing the greatest fixpoint (the more precise approximation)

Also, observe that this is a forward analysis.

# Computing the greatest fix point

The starting point, for all n
$AE_{entry}(n)=AE_{exit}(n)=\{a+b,a*b,a+1\}$

x:=a+b; y:=a*b; while y>a+b do (a:=a+1; x:=a+b)

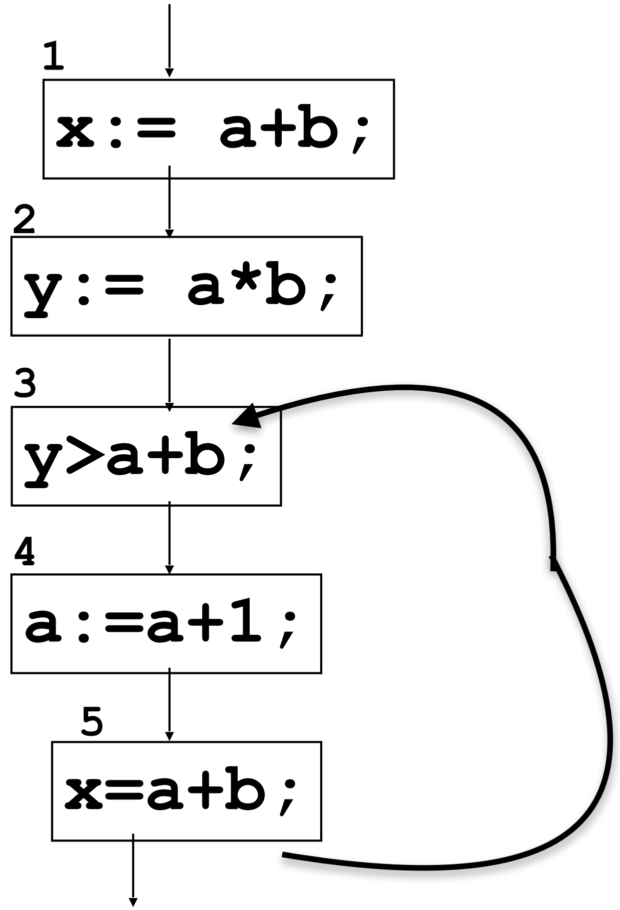
**E** = {a+b, a*b, a+1}

$AE_{entry}(p)=\emptyset$ if p is initial

$AE_{entry}(p)=\bigcap\{AE_{exit}(q) \mid q \text{ in } pre[p]\}$

$AE_{exit}(p) = (AE_{entry}(p) \setminus kill_{AE}(p)) \cup gen_{AE}(p)$

| n | $kill_{AE}(n)$ | $gen_{AE}(n)$ |
|---|---|---|
| 1 | $\emptyset$ | {a+b} |
| 2 | $\emptyset$ | {a*b} |
| 3 | $\emptyset$ | {a+b} |
| 4 | {a+b, a*b,a+1} | $\emptyset$ |
| 5 | $\emptyset$ | {a+b} |




$AE_{entry}(1)=\emptyset$ $AE_{exit}(1)=\{a+b\}$

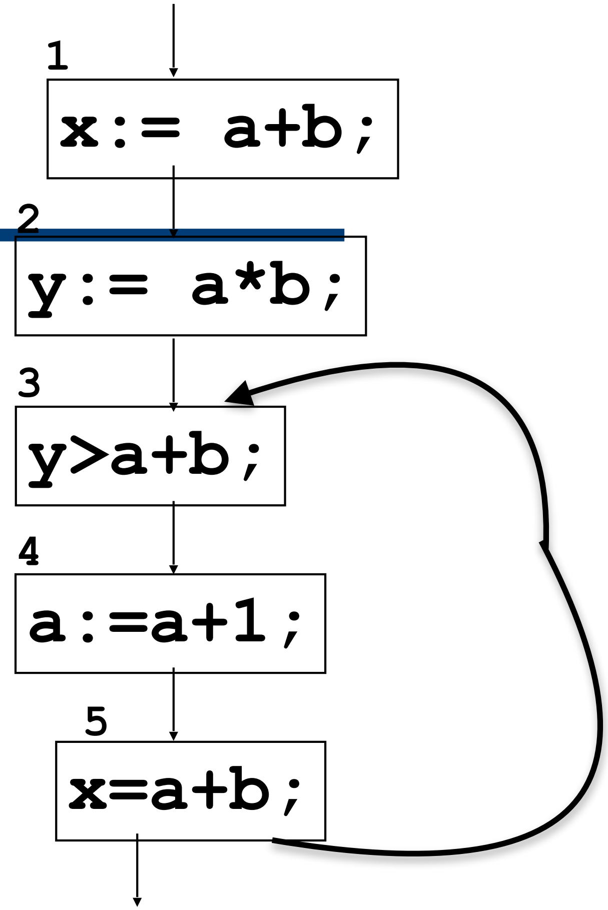
$AE_{entry}(2)=\{a+b\}$ $AE_{exit}(2)=\{a+b,a*b\}$

$AE_{entry}(3)=\{a+b,a*b\}$ $AE_{exit}(3)=\{a+b,a*b\}$

$AE_{entry}(4)=\{a+b,a*b\}$ $AE_{exit}(4)=\{\}$

$AE_{entry}(5)=\{\}$ $AE_{exit}(5)=\{a+b\}$

# Second iteration




$AE_{entry}(p)=\varnothing$ if p is initial

$AE_{entry}(p)= \bigcap\{AE_{exit}(q) \mid q \text{ in } pre[p]\}$

$AE_{exit}(p) = (AE_{entry}(p) \setminus kill_{AE}(p)) \cup gen_{AE}(p)$

Previous iteration

| n | $AE_{entry}(n)$ | $AE_{exit}(n)$ |
|---|---|---|
| 1 | ∅ | {a+b} |
| 2 | {a+b} | {a+b, a*b} |
| 3 | {a+b,a*b} | {a+b,a*b} |
| 4 | {a+b,a*b} | ∅ |
| 5 | ∅ | {a+b} |

$AE_{exit}(1)= AE_{entry}(1) \cup \{a+b\}$

$AE_{exit}(2)= AE_{entry}(2) \cup \{a*b\}$

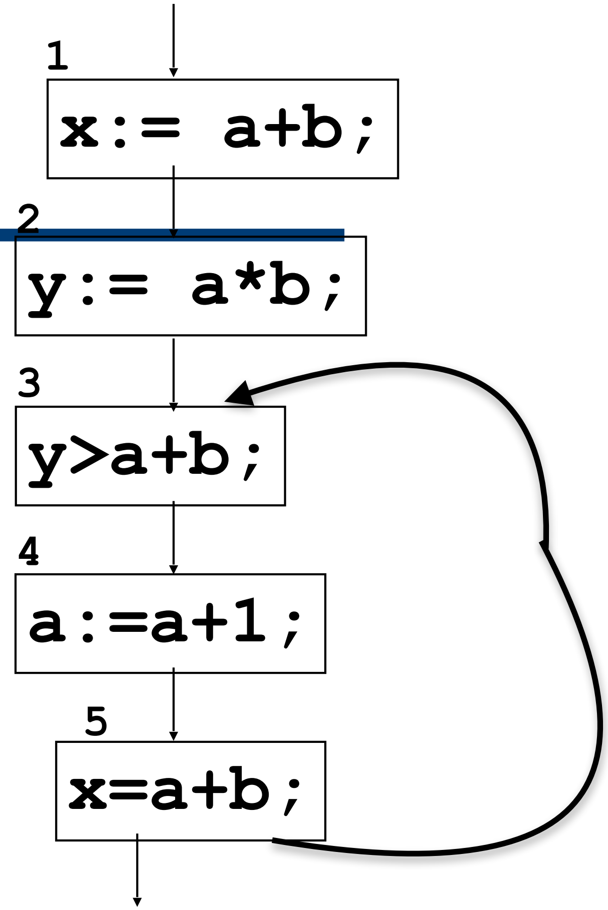
$AE_{exit}(3)= AE_{entry}(3) \cup \{a+b\}$

$AE_{exit}(4)= AE_{entry}(4) - \{a+b, a*b, a+1\}$

$AE_{exit}(5)= AE_{entry}(5) \cup \{a+b\}$

| n | $AE_{entry}(n)$ | $AE_{exit}(n)$ |
|---|---|---|
| 1 | ∅ | {a+b} |
| 2 | {a+b} | {a+b, a*b} |
| 3 | {a+b} | {a+b} |
| 4 | {a+b} | ∅ |
| 5 | ∅ | {a+b} |

# Third iteration and Greatest Fixpoint




$AE_{entry}(p)=\varnothing$ if p is initial

$AE_{entry}(p)= \bigcap\{AE_{exit}(q) \mid q \text{ in } pre[p]\}$

$AE_{exit}(p) = (AE_{entry}(p) \setminus kill_{AE}(p)) \cup gen_{AE}(p)$

Previous iteration

| n | $AE_{entry}(n)$ | $AE_{exit}(n)$ |
|---|---|---|
| 1 | ∅ | {a+b} |
| 2 | {a+b} | {a+b, a*b} |
| 3 | {a+b} | {a+b} |
| 4 | {a+b} | ∅ |
| 5 | ∅ | {a+b} |

$AE_{exit}(1)= AE_{entry}(1) \cup \{a+b\}$

$AE_{exit}(2)= AE_{entry}(2) \cup \{a*b\}$

$AE_{exit}(3)= AE_{entry}(3) \cup \{a+b\}$

$AE_{exit}(4)= AE_{entry}(4) - \{a+b, a*b, a+1\}$

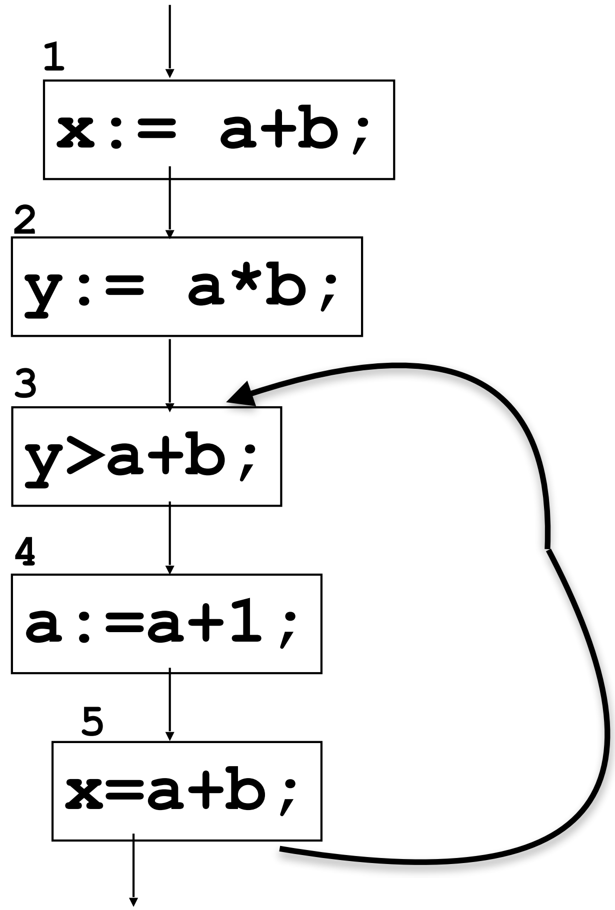
$AE_{exit}(5)= AE_{entry}(5) \cup \{a+b\}$

| n | $AE_{entry}(n)$ | $AE_{exit}(n)$ |
|---|---|---|
| 1 | ∅ | {a+b} |
| 2 | {a+b} | {a+b, a*b} |
| 3 | {a+b} | {a+b} |
| 4 | {a+b} | ∅ |
| 5 | ∅ | {a+b} |

# Result

x:=a+b; y:=a*b; while y>a+b do (a:=a+1; x:=a+b)

| n | $AE_{entry}(n)$ | $AE_{exit}(n)$ |
|---|---|---|
| 1 | ∅ | {a+b} |
| 2 | {a+b} | {a+b, a*b} |
| 3 | **{a+b}** | {a+b} |
| 4 | {a+b} | ∅ |
| 5 | ∅ | {a+b} |



# Application: Common Subexpression Elimination

A[i,j]=B[i,j]+C[i,j]

```
i := 0;
while i <= n do
   j := 0;
   while j <= m do
      temp := Base(A) + i*(m+1) + j;
      Cont(temp) := Cont(Base(B) + i*(m+1) + j)
                  + Cont(Base(C) + i*(m+1) + j);
      j := j+1
   od;
   i := i+1
od
```

first computation

re-computations

```
t1 := i * (m+1) + j;
temp := Base(A) + t1;
Cont(temp) := Cont(Base(B)+t1)
            + Cont(Base(C)+t1);
```

# A Dataflow Analysis Framework

- The above dataflow analyses (Reaching Definitions, Available Expressions, Live Variables) reveal many similarities.
- One major advantage of a unifying framework of dataflow analysis lies in the design of a generic analysis algorithm that can be instantiated in order to compute different dataflow analyses.

# Catalogue of Dataflow Analyses

| | *Possible Analysis*<br>**Semantics ⊆ Analysis** | *Definite Analysis*<br>**Analysis ⊆ Semantics** |
| --- | --- | --- |
| *Forward*<br>**in[n] ⇛ out[n]**<br>**pre ⇛ post** | Reaching definitions | Available expressions |
| *Backward*<br>**out[n] ⇛ in[n]**<br>**post ⇛ pre** | Live variables | **Very busy expressions** |