Application: Dead Code Elimination

dead variable

```
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
```

```
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?

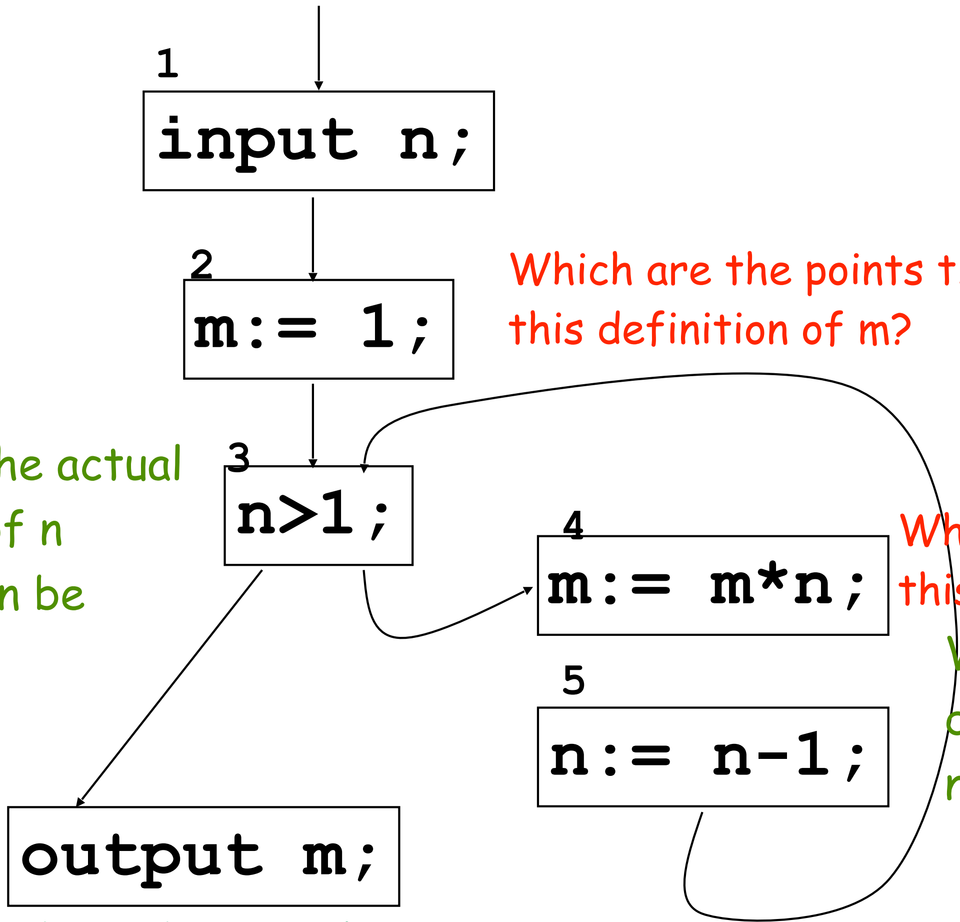
- A program point may clearly "generate" new definitions

- 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.

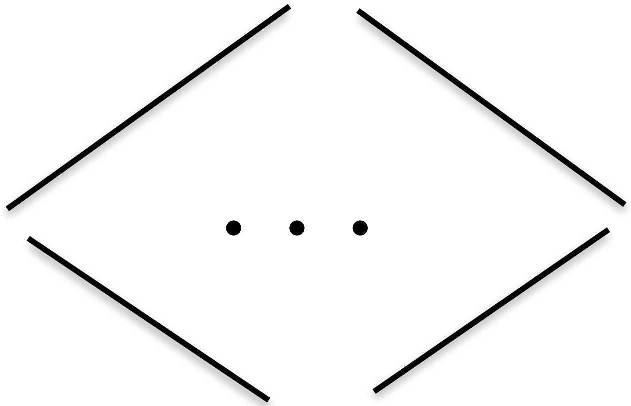
Let N be the number of nodes of the CFG of P.

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

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

- Example Vars={a,b} e N=1

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



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

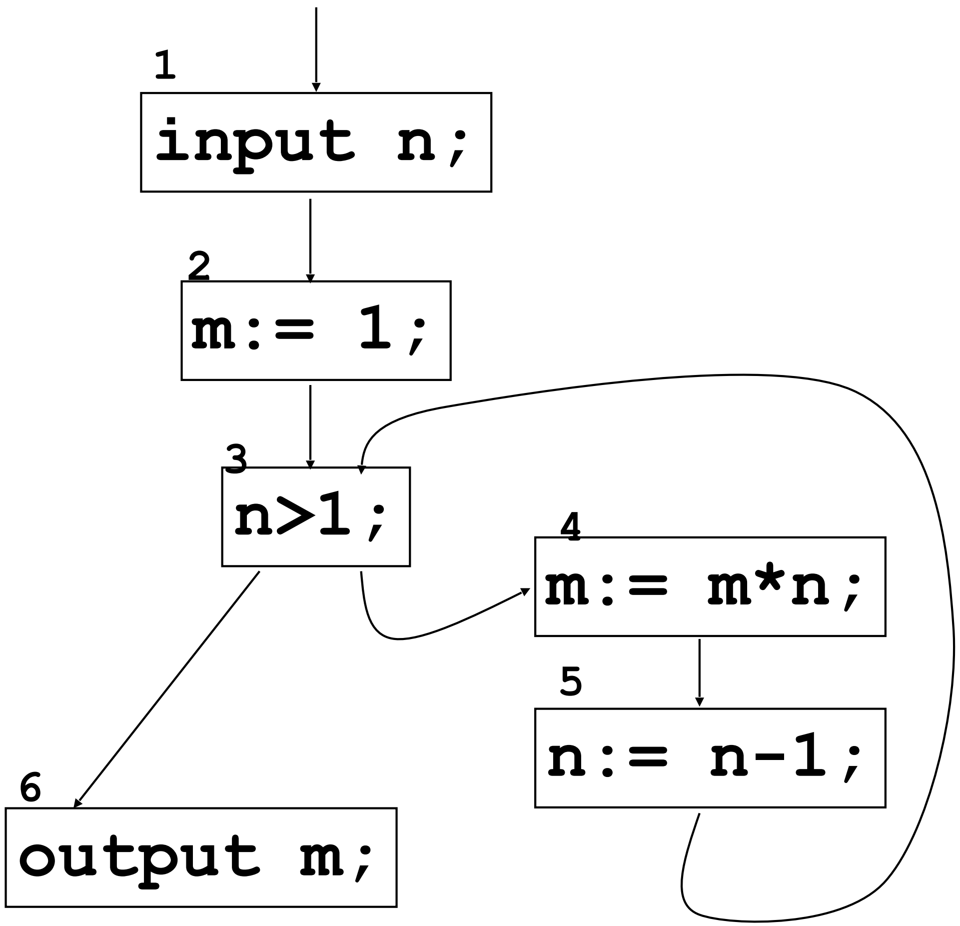
Specification

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

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

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

Kill and Gen



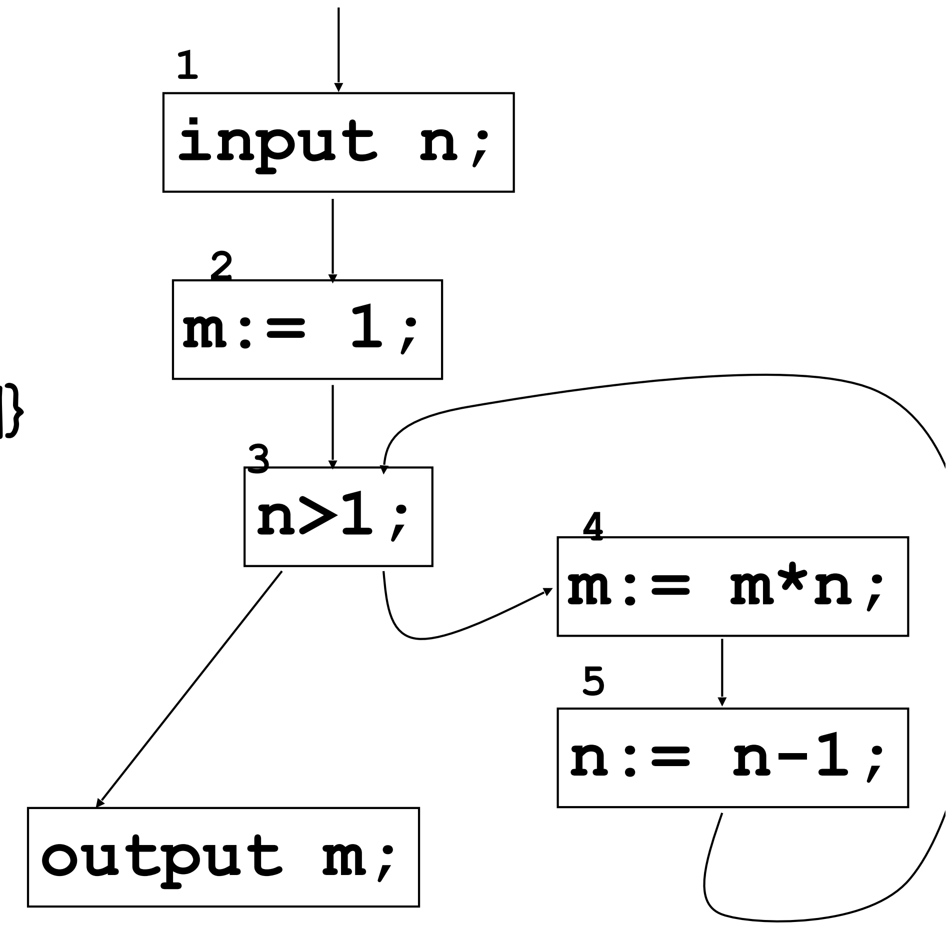
	$kill_{RD}$	gen_{RD}
1		
2	(m,?)(m,2) (m,4)	(m,2)
3		
4	(m,?)(m,2) (m,4)	(m,4)
5	(n,?) (n,5)	(n,5)
6		

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 are require 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?

Yes! Because it is finite

Point a: the map

The map Reach:

$<\mathcal{P}(\text{Vars} \times \text{Points}) \times \mathcal{P}(\text{Vars} \times \text{Points}))^N \rightarrow <\mathcal{P}(\text{Vars} \times \text{Points}) \times \mathcal{P}(\text{Vars} \times \text{Points}))^N$

defined by

(assuming 1 is the only initial node)

$\text{Reach}(<\text{RDentry}_1, \text{RDexit}_1, \ldots, \text{RDentry}_N, \text{RDexit}_N>) =$

$< \{(x,?) \mid x \text{ in VARS}\}, \text{RD}_{\text{entry1}} \backslash \text{kill}_{\text{RD}}[1]) \cup \text{gen}_{\text{RD}}[1],$

$\cup\{\text{RD}_{\text{exit2}} \mid m \text{ in pre}[2]\}, \text{RD}_{\text{entry2}} \backslash \text{kill}_{\text{RD}}[2]) \cup \text{gen}_{\text{RD}}[2],$

....,

$\cup\{\text{RD}_{\text{exitm}} \mid m \text{ in pre}[N]\}, \text{RD}_{\text{entryN}} \backslash \text{kill}_{\text{RD}}[N]) \cup \text{gen}_{\text{RD}}[N] >$

Point a

$Reach(\langle RDentry_1, RDexit_1, \ldots, RDentry_N, RDexit_N \rangle) =$

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

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

$\ldots,$

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

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

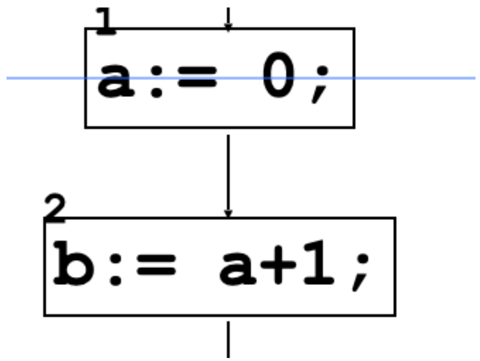
$kill_{RD}(2)=\{(b,?)\}$, $gen_{RD}(2)=\{(b,2)\}$

- Example

$Reach(\langle \{\}\{\}\{\}\{\} \rangle) = \langle \{(a,?)(b,?)\}\{(a,1)(b,?)\}\{(a,1)(b,?)\}\{(a,1)(b,2)\} \rangle$

$Reach(\langle \{(a,?)(b,?)\}\{(a,1)(b,?)\}\{(a,1)(b,?)\}\{(a,1)(b,2)\} \rangle) =$

$\langle \{(a,?)(b,?)\}\{(a,1)(b,?)\}\{(a,1)(b,?)\}\{(a,1)(b,2)\} \rangle$



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

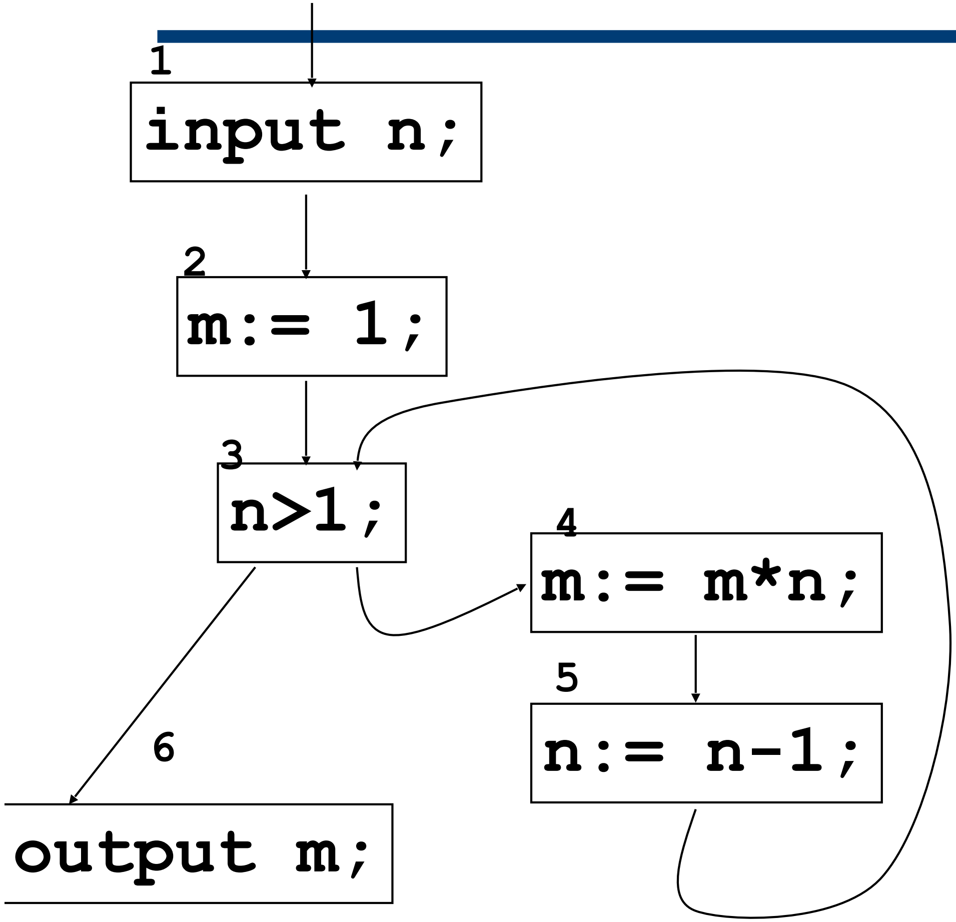
$(x,q) \in RD_{entry}(p)$

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:



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

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

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

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

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

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

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

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

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

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

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

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

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

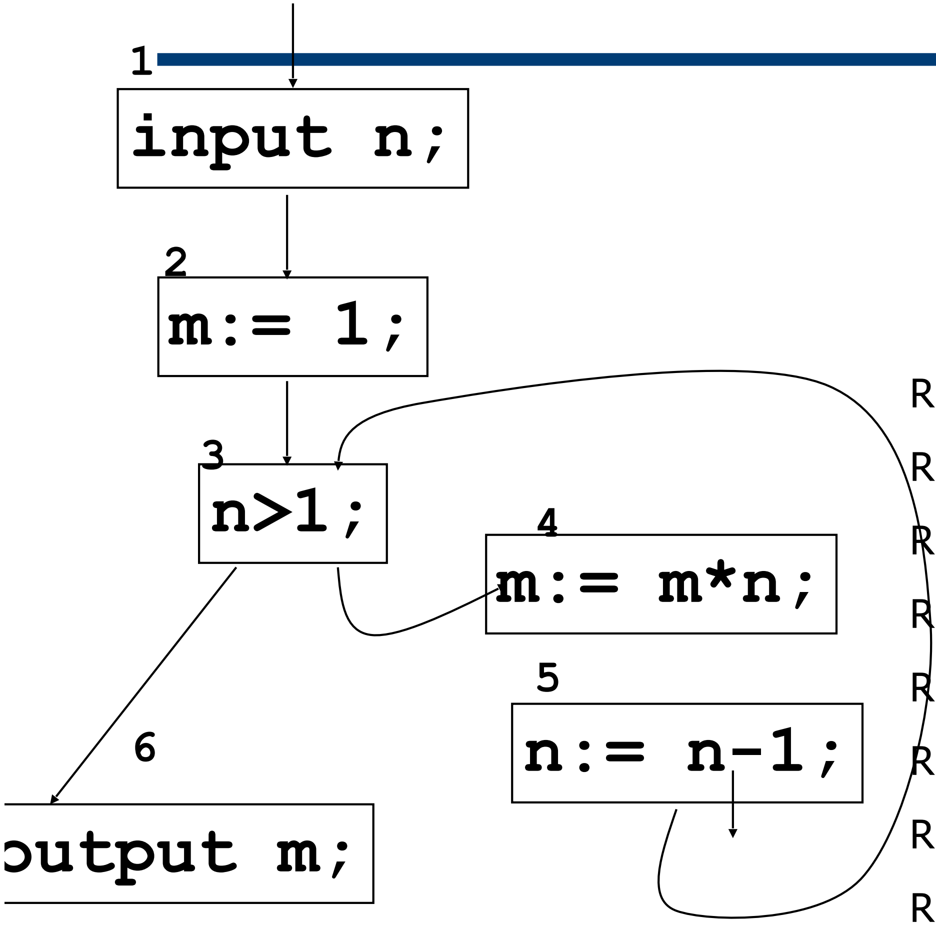
$RD_{entry}(p) = \{(x,?) \mid x \text{ in } Vars\}$, if p is initial

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



$RD_{entry}(1)$= {(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!

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

$RD_{entry}(p)$ =U{$RD_{exit}(q)$ | q in pre[p]}, otherwise

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

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

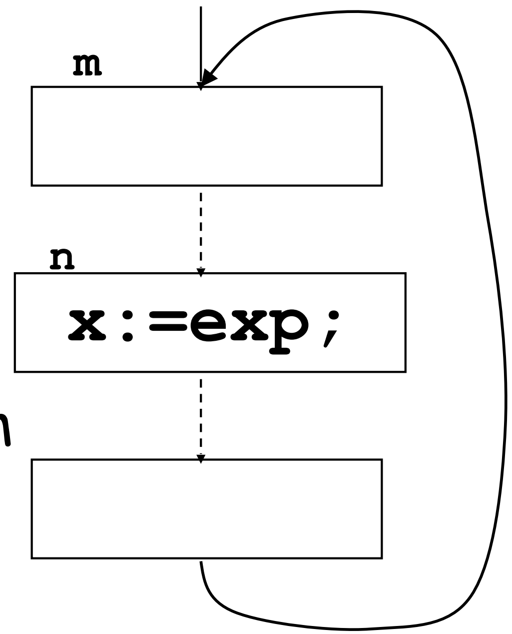
Application: Loop invariant code motion

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

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

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 the expressions that have already been evaluated and then not modified.

These are called available expressions

Example

x:=a+b;

y:=a*b;

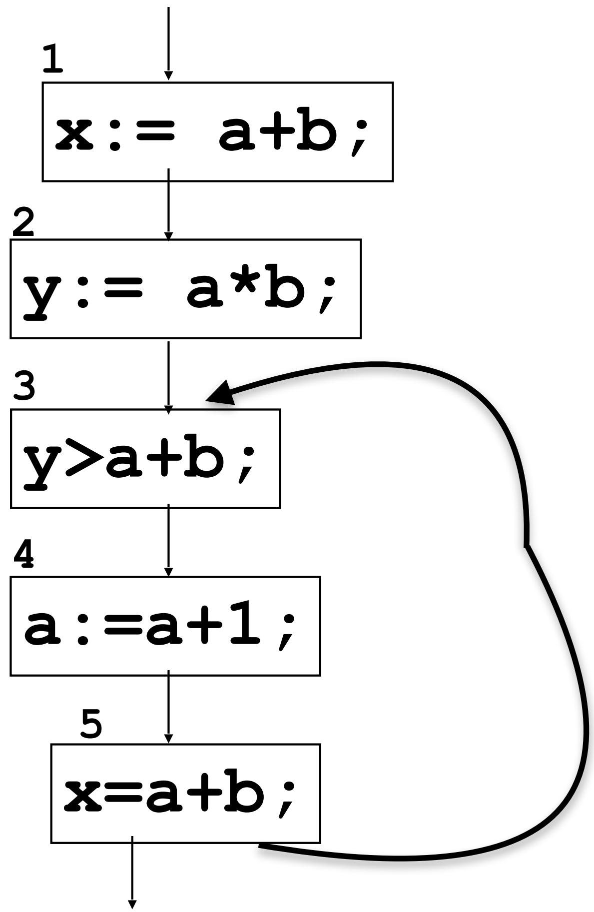
while y>a+b

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.

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

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

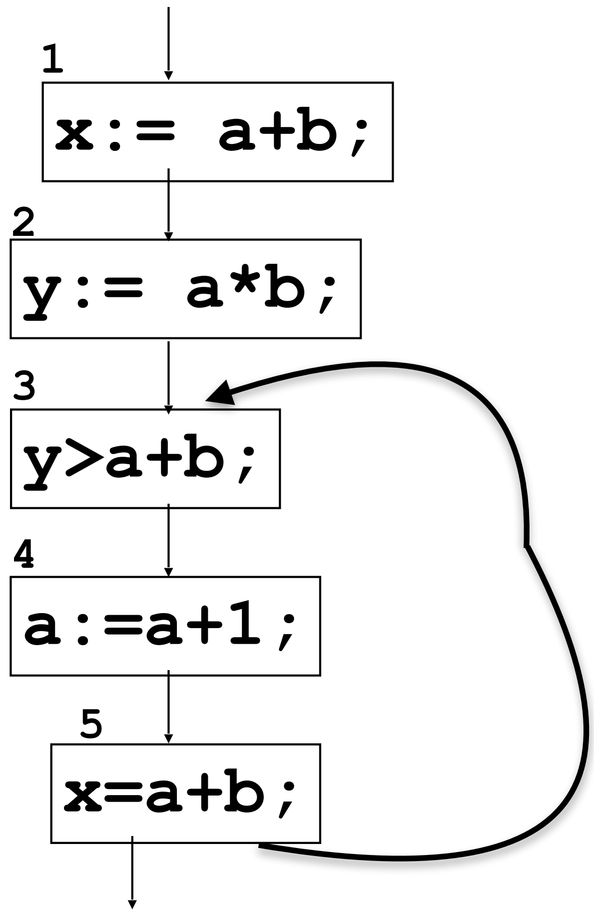
$$gen_{AE}(S)^p = \mathbf{exps}(S) \qquad \text{if } S \; x:=e$$

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:

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

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

Point a and b to apply Kleene Theorem

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

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

a) The map $(\mathcal{P}(\mathbf{E}) \times \mathcal{P}(\mathbf{E}))^N \to (\mathcal{P}(\mathbf{E}) \times \mathcal{P}(\mathbf{E}))^N$ 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

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

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

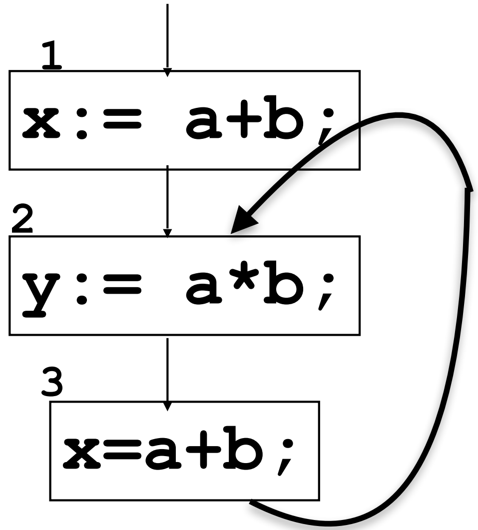
$\cap\{AE_{exitq} \mid q \text{ in } pre[2]\}, (AE_{entry2} \setminus kill_{AE}(2)) \cup gen_{AE}(2),$

.....

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

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

- Example



AE(<∅,∅,∅,∅,∅,∅>)=

<∅,{a+b}, {}, {a*b}, {a*b}, {a+b, a*b}>

AE(<∅,{a+b}, {}, {a*b}, {a*b}, {a+b, a*b}>)=

<∅,{a+b}, {a+b}, {a+b,a*b}, {a+b,a*b}, {a+b,a*b}>

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

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

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

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

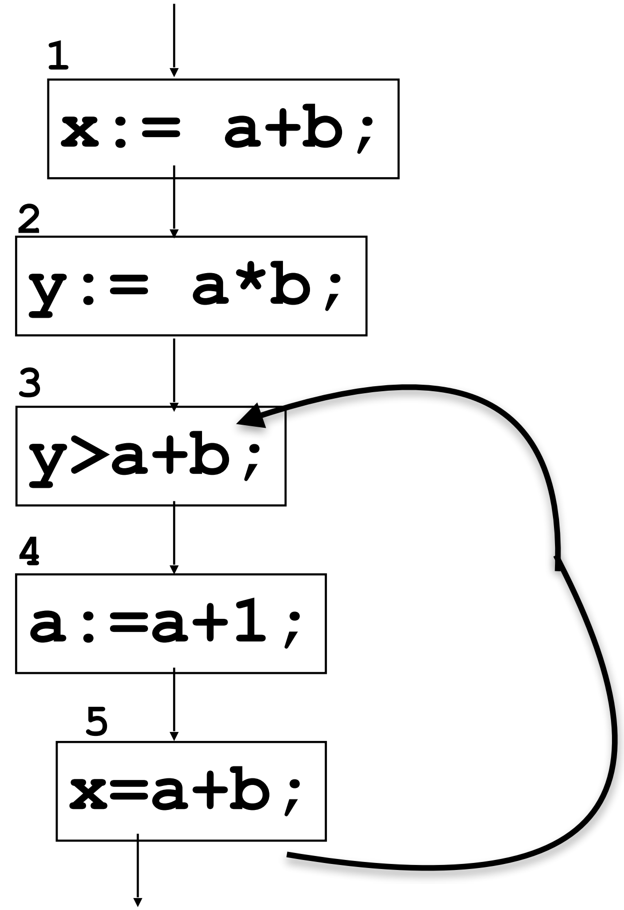
$AE_{entry}(1)=\emptyset$ $AE_{exit}(1)=\{a+b\}$

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

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

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

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

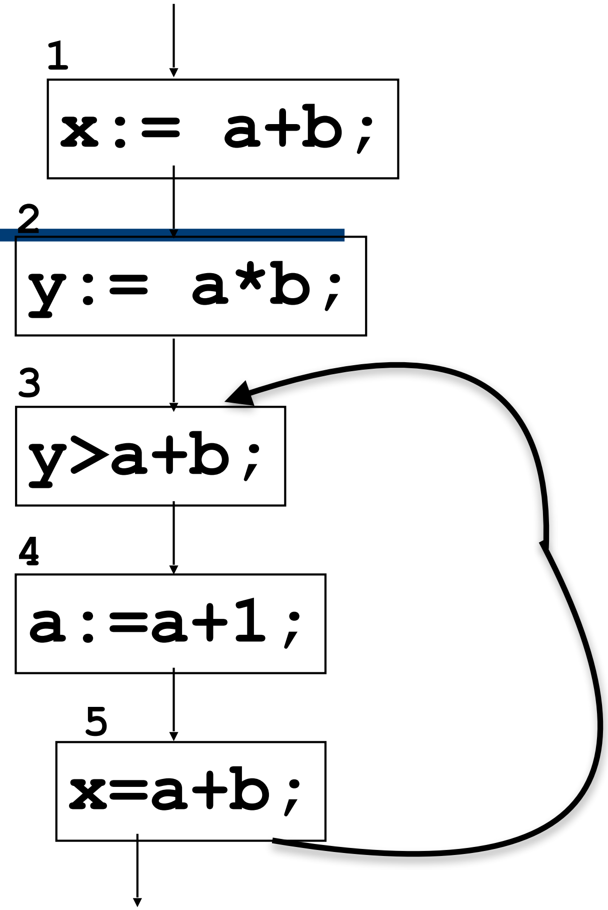


Second iteration

$AE_{entry}(p)=\emptyset$ 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	$AE_{entry}(n)$	$AE_{exit}(n)$
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}

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

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

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

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

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

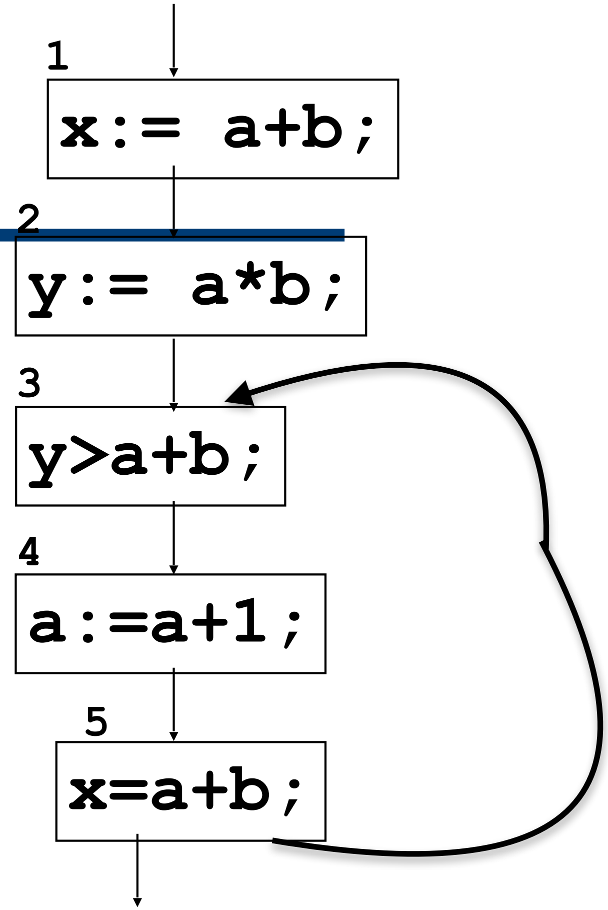
n	$AE_{entry}(n)$	$AE_{exit}(n)$
1	∅	{a+b}
2	{a+b}	{a+b, a*b}
3	{a+b}	{a+b}
4	{a+b}	∅
5	∅	{a+b}

Third iteration and Greatest Fixpoint

$AE_{entry}(p)=\emptyset$ 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	$AE_{entry}(n)$	$AE_{exit}(n)$
1	$\emptyset$	{a+b}
2	{a+b}	{a+b, a*b}
3	{a+b}	{a+b}
4	{a+b}	$\emptyset$
5	$\emptyset$	{a+b}

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

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

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

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

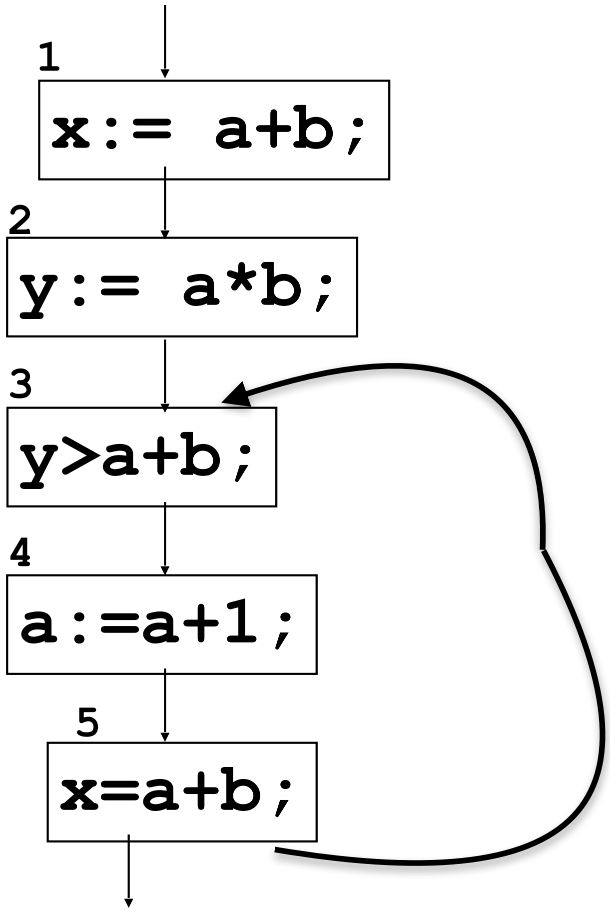
$AE_{exit}(5)= AE_{entry}(5) \cup \{a+b\}$

n	$AE_{entry}(n)$	$AE_{exit}(n)$
1	$\emptyset$	{a+b}
2	{a+b}	{a+b, a*b}
3	{a+b}	{a+b}
4	{a+b}	$\emptyset$
5	$\emptyset$	{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 **Semantics** $\subseteq$ **Analysis**	*Definite Analysis* **Analysis** $\subseteq$ **Semantics**
Forward **in[n]** ⇒ **out[n]** **pre** ⇒ **post**	Reaching definitions	Available expressions
Backward **out[n]** ⇒ **in[n]** **post** ⇒ **pre**	Live variables	**Very busy expressions**