Back to our example

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?

We can apply the fix point theory results !

We need to check that we have

a) a continuous function on

b) a CPO with bottom

Kleene's Theorem

Point b first

Vars is the (finite) set of variables occuring in the program P.
Let N be the number of nodes of the CFG of P.

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

Point b

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

CPO with bottom?

Yes! Because it is finite

Point a

The map Live:

((**Vars**)x $\mathcal{P}$(**Vars**))^N -> ($\mathcal{P}$ (**Vars**)x$\mathcal{P}$(**Vars**))^N defined by

Live(<in_1,out_1,…,in_N,out_N>)=

<use[1]∪(out_1-def[1]), $\bigcup_{m \in post[1]}$ in_m, …., use[N]∪ (out_N -def[N]), $\bigcup_{m \in post[N]}$ in_m>

Point a

The map Live:

((**Vars**)x $\mathcal{P}$(**Vars**))^N -> ($\mathcal{P}$ (**Vars**)x$\mathcal{P}$(**Vars**))^N defined by

Live(<in_1,out_1,…,in_N,out_N>)=

<use[1]∪(out_1-def[1]), $\bigcup_{m \in post[1]}$ in_m, …., use[N]∪ (out_N -def[N]), $\bigcup_{m \in post[N]}$ in_m>

is continuous?

Yes! because it is monotone on a finite domain

Why a least fixpoint

- Live is a possible analysis,

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

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)

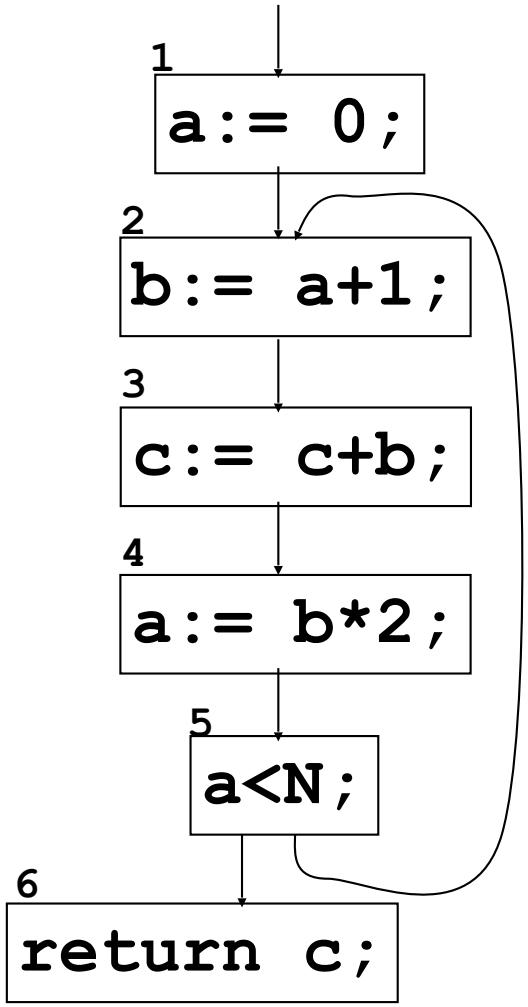
- The converse does not hold: the analysis can tell us that x is in the computed set out[n], but this does not imply that x will be necessarily live in n during some program execution
- In liveness analysis "conservative approximation" means that the analysis may erroneously derive that a variable is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

★if $x \in$ in[n] then x could be live at program point n.

★if $x \notin$ in [n] then x is definitely dead at program point n.

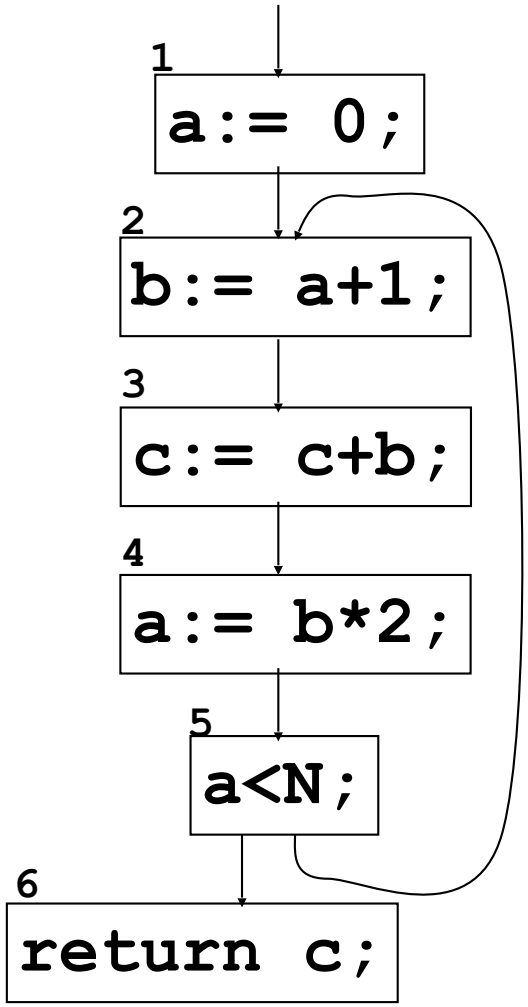
```
for all n
    in[n]:={} out[n]:={};
repeat
    for all n (1 to 6)
          in'[n]:=in[n]; out'[n]:=out[n];
          in[n]:= use[n] U (out[n] - def[n]);
          out[n]:= U { in[m] | m ∈ post[n]};
until (for all n: in'[n]=in[n] && out'[n]=out[n])
```

			Live^1		Live^2		Live^3	
	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	

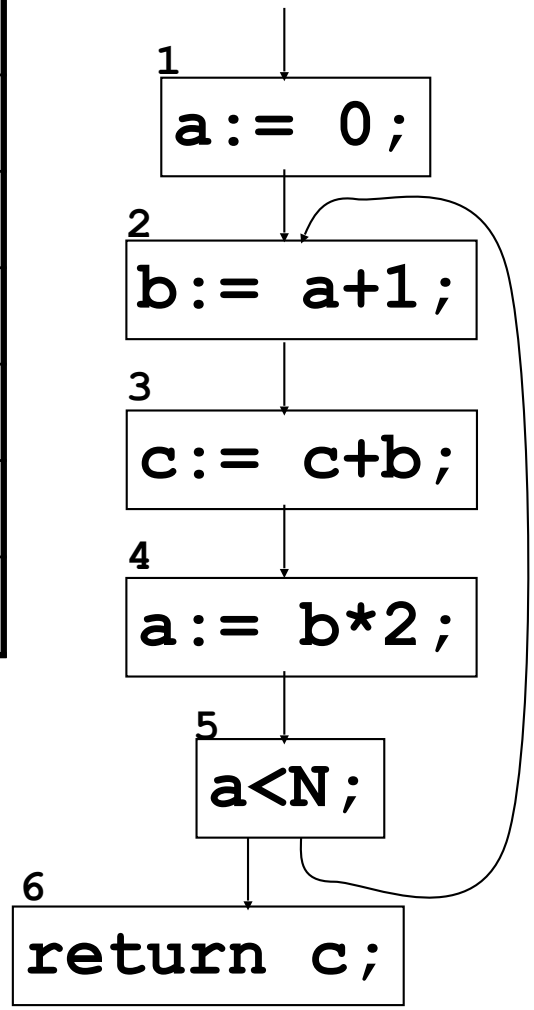


```
for all n
    in[n]:=?; out[n]:=?;
repeat
    for all n (1 to 6)
          in'[n]:=in[n]; out'[n]:=out[n];
          in[n]:= use[n] U (out[n] - def[n]);
          out[n]:= U { in[m] | m ∈ post[n]};
until (for all n: in'[n]=in[n] && out'[n]=out[n])
```

			Live^3		Live^4		Live^5	
	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	



			Live[5]		Live[6]		Live[7]	
	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])
```

			Live[1]		Live[2]		Live[3]	
	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.