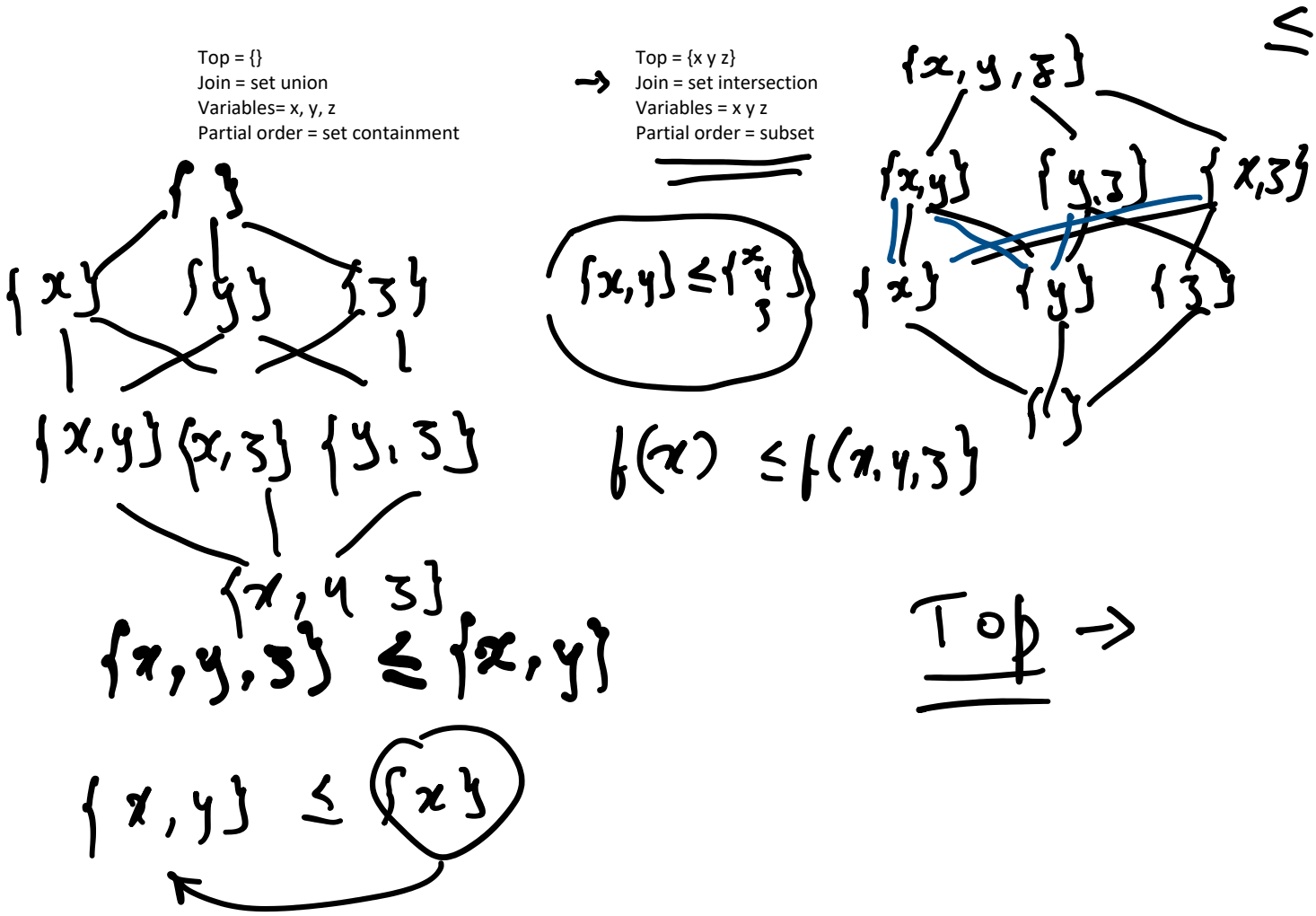
Lattice

13 March 2022 11:35

Top = {}
Join = set union
Variables= x, y, z
Partial order = set containment

→

Top = {x y z}
Join = set intersection
Variables = x y z
Partial order = subset

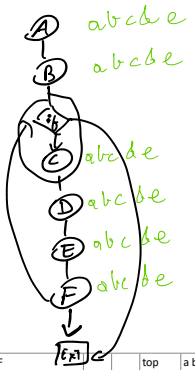


Worklist algorithm

13 March 2022 10:43

Place entry node in the worklist.
Init state to top
While the worklist is not empty:
Pop a node from the worklist
Join all its preds
Apply the transfer function for that node.
Place its successors who changed in the worklist.

→ A: a=1
B: c=0
for i=1 ... 10 {
C: b=2
D: d= a+b
E: e=b+c
F: c=4



i	A	B	C	D	E	F			top	abcde
0	abcde	abcde	abcde	abcde	abcde	abcde			bottom	{}
1	a	ab	abc	abcd	abcde	a1b2c4d3e2				
			ab	abd	abd	abd				
			ab	abd	abd	abdc				

{a=1}

→ { } ∪ {a=1}

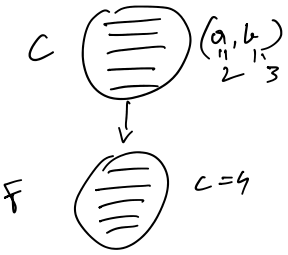
{a=1}

{a=1}

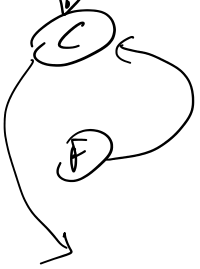
f(B, a=1) → {a=1, c=0}

f(C, (a=1, c=0)) → { (a=1), (c=0) }

f(D,



Transfer fun B & F



f(B, S) = S ∪ {B}

1) information from predecs

2) previous state

f(C, (a=1, c=0))

```
s=0
For I = 1: n
   x=v[i]
   s= s+x
```

out

Transfer func. →

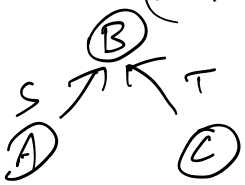
join → ∪ (comp'in)

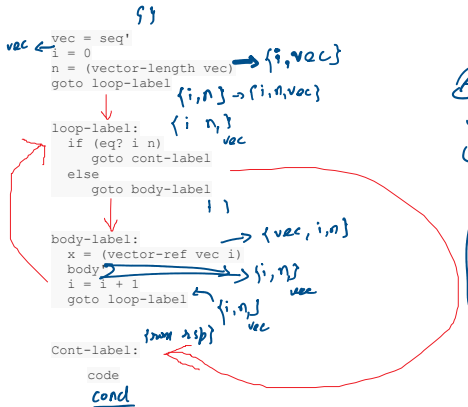
x=e {S-x} ∪ V(e)

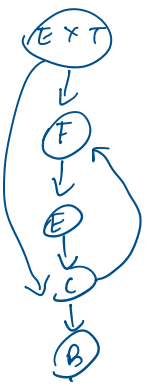
(S' = (S ∪ T) ....) → S' → (S ∪ T)

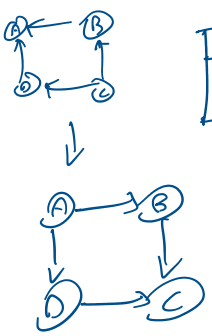
T

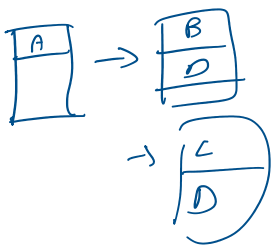
S' = {S ∪ T}

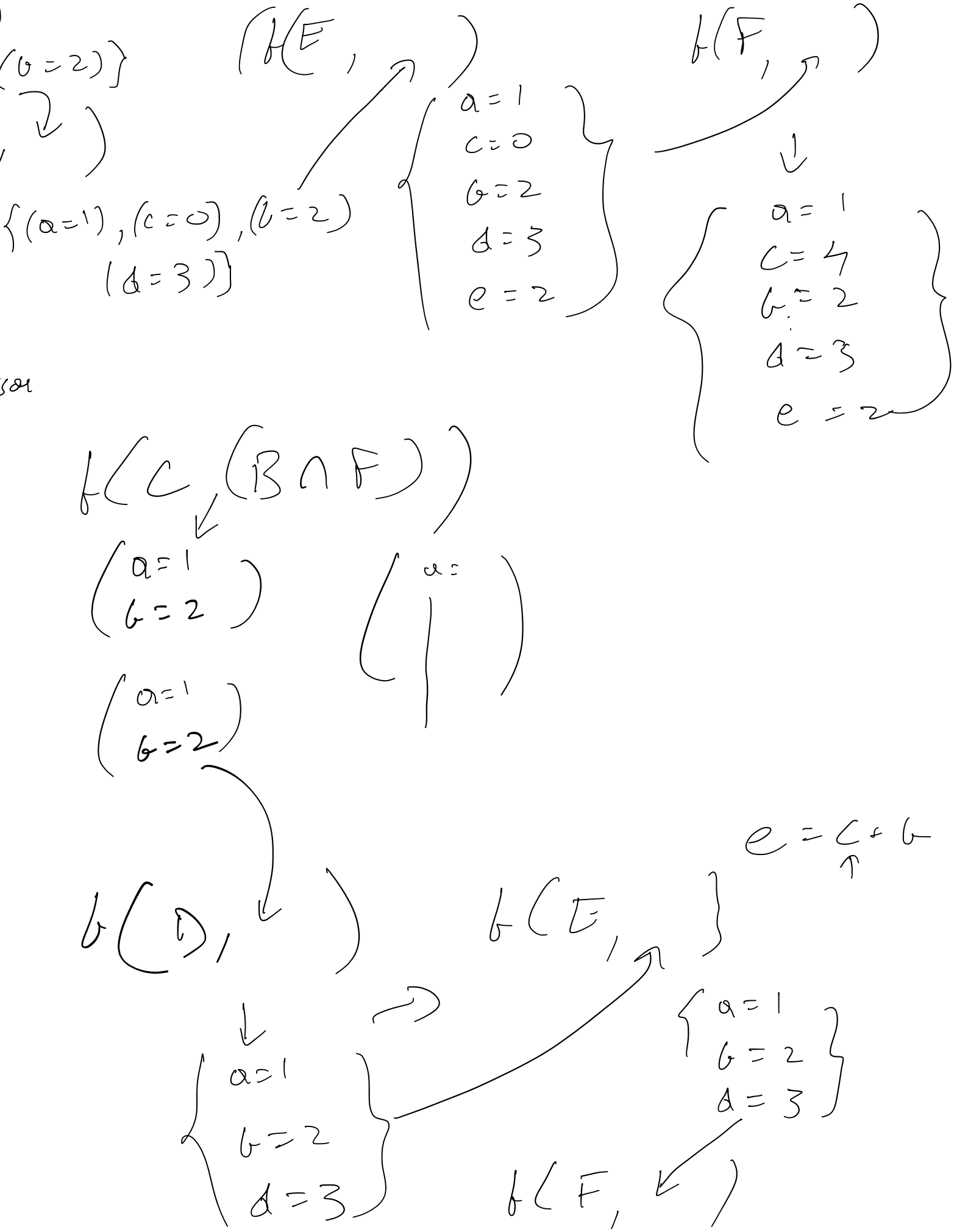


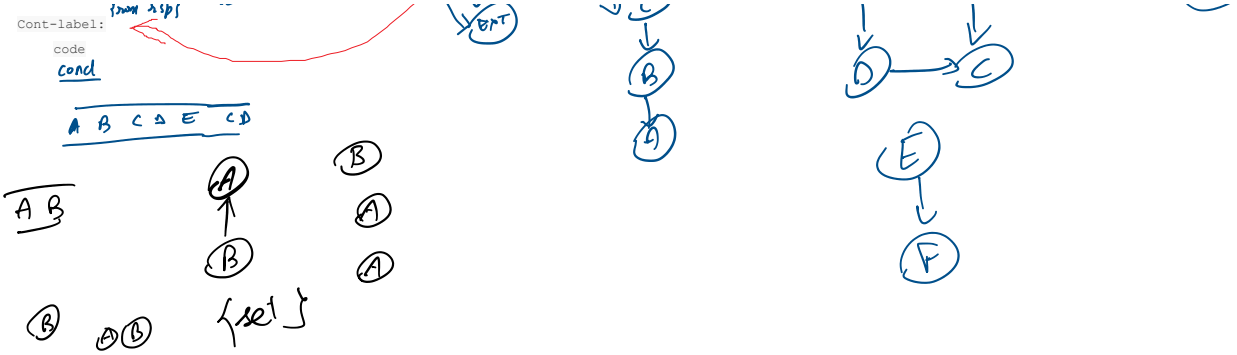


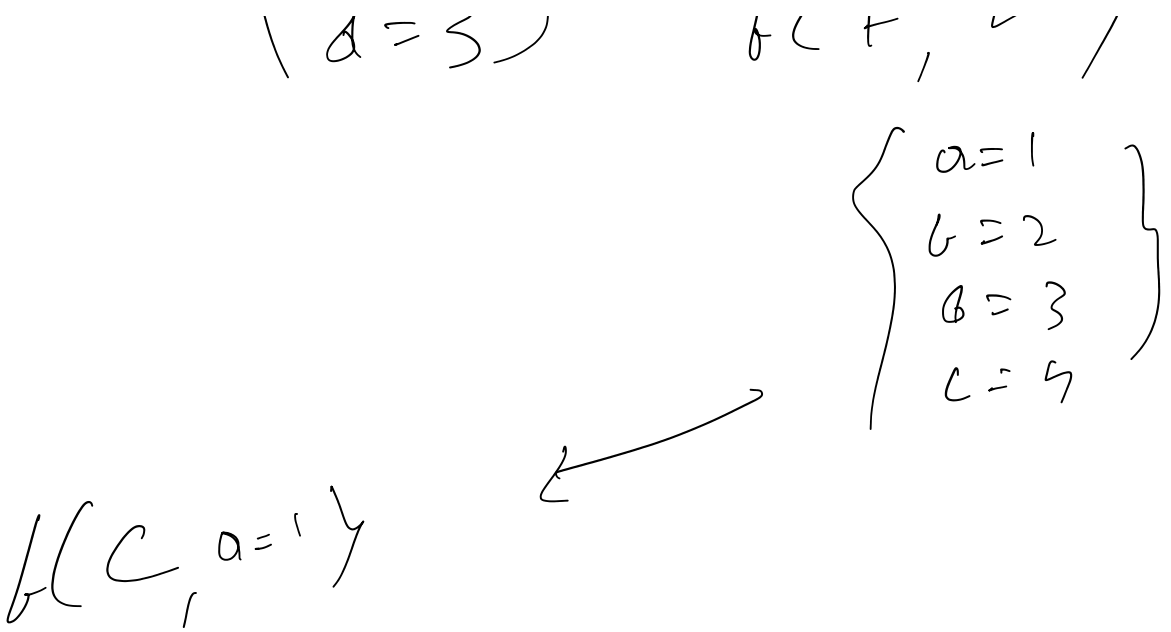







(b=2)}
{(a=1),(c=0),(b=2)
(d=3)}
f(E,
a=1
c=0
b=2
d=3
e=2
f(F,
a=1
c=4
b=2
d=3
e=2
f(C,(B∩F))
a=1
b=2
a=
f(D,
a=1
b=2
f(E,
e=c+b
a=1
b=2
d=3
f(F,


Cont-label:
code
concl
A B C D E C D
A B
{set}


a=1
b=2
a=3
c=4

While loop

13 March 2022 11:40

Begin, set, while : void types (unlike +, let)
Set!: variable lifetime is end of the program
Begin: executes the set of expressions

While exp exp
Begin es exp: exp is the last expression that is returned