NTNU – Trondheim
Norwegian University of
Science and Technology

Dataflow Analysis Framework: Summary and precision

We have looked at

- Live Variables
- Available Expressions
- Reaching Definitions
- Copy Propagation
  - as instances of a general dataflow analysis method
  - as points in a control flow graph
  - as data flow equations that associate sets with the points
  - as positions in a partial order (lattice) of possible sets
- Today, we'll add one more (Constant Folding) and look at how good our iterative solution is

Constant Folding (and propagation)

- The domain we're after is *pairs of variables, and their constant values.*
  - Obviously, not every variable will *have* a constant value, more in a minute
- Forward analysis
  - Traces paths from a point where a variable may be constant, to any point where we have determined that it isn't
- An intersection meet operator (of sorts)
  - A constant value must be the same along every path, otherwise it isn't very constant

Three levels of information

- We can say three things about the constant-ness of a variable X
  1) *X may be a constant, but we haven't found its value yet*
  2) *X may be a constant, its value has only been 36* (or some other number)
  3) *X is not constant, we've seen changes in its value*
- We can order these observations according to how much we've found out about X:

X = ⊤	← Can't say anything about X yet ("least precise knowledge")
X = 21	← X is 21 somewhere in the program
X = ⊥	← X is not 21 everywhere ("most precise knowledge")

NTNU – Trondheim
Norwegian University of Science and Technology

The program logic

- An assignment of a constant to a variable (v=c) generates that pair as a possibly constant value

  gen [l] = {v=c}

- It also destroys the possibility that v is any other constant than c

  kill [l] = {v=n where n ≠ c}

- An assignment of an expression (v=u+w) generates a possibly constant value if all its terms are constant

  gen [l] = {v=k}   kill [l] = { v=n where n ≠ k}

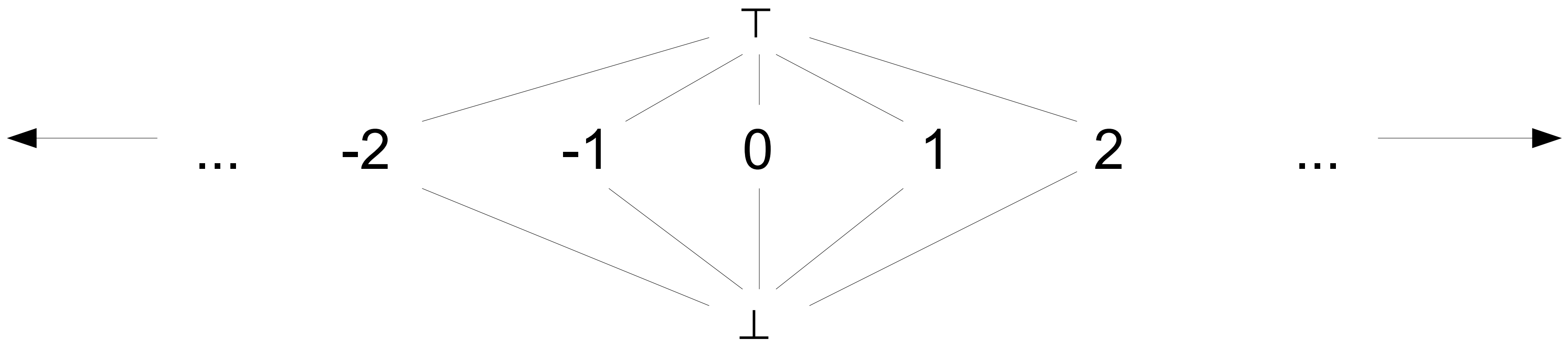
  k=u+w if u,w are constants

  k= ⊥ if u or w are ⊥   (known to be not-constant)

  k= T otherwise

If we draw the three levels

- There is an infinity of constants
- "X=36" is as informative as "X=21", but taken together, they say that X is neither 36 nor 21 *always*
- A lattice of more and less informative levels becomes

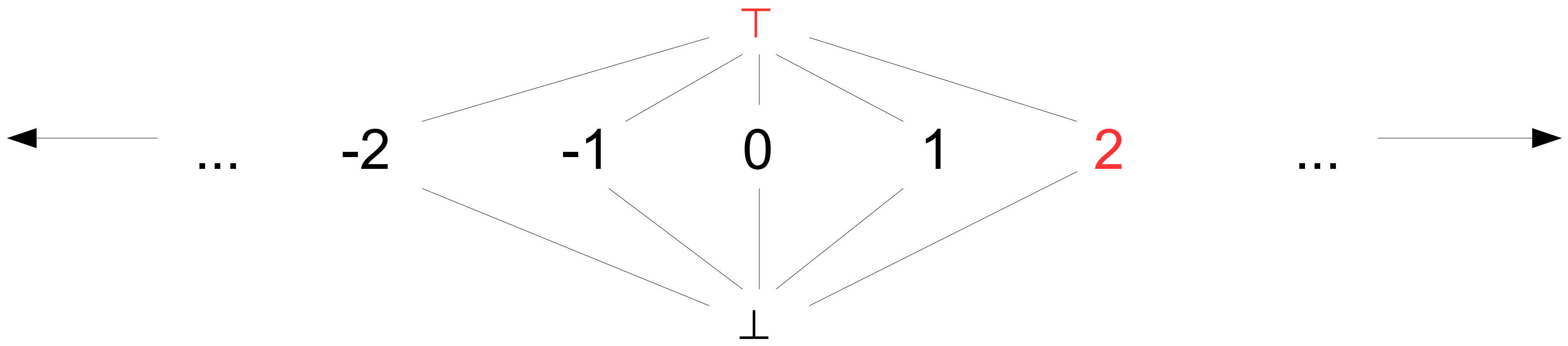


(It's infinitely wide, but has finite height)

NTNU – Trondheim
Norwegian University of
Science and Technology

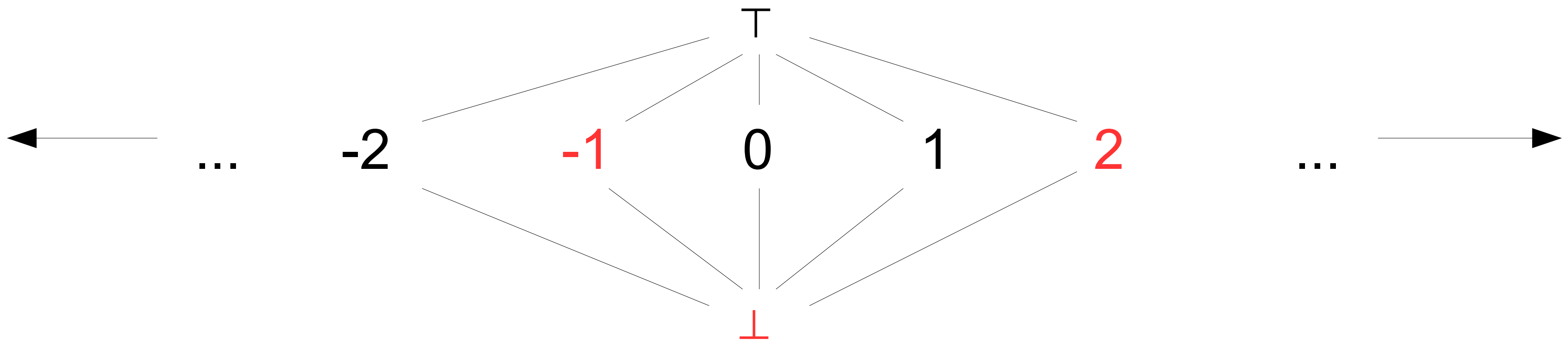
When X=⊤ meets X=2

- One set of observations haven't seen any value for X
- The other has only seen that X = 2
- X could be the constant 2
- {X=⊤} ⊓ {X=2} gives {X=2} (greatest lower bound in the order)



When X=-1 meets X=2

- One set of observations have only seen that X=-1
- The other has only seen that X = 2
- X can’t be a constant, there are two different values
- {X=-1} ⊓ {X=2} gives {X=⊥}   (greatest lower bound in the order)



Part of a meet operator

- This ordering relation of

  $\bot \sqsubseteq$ *(numbers)* $\sqsubseteq \top$

  and the meet operator

  $p \sqcap q = \text{glb}(p, q)$ *(in our constants-lattice)*

  gives how to handle multiple observations about one variable
  - The p-s and q-s here are set elements like “X=64”, “X=⊥”, “X=⊤”, *et cetera*.
  - Those all talk about one variable
  - “Y=27”, “Y=13”, “Y=⊤” are positions in a separate lattice, which describes the constant-ness of Y

    (that has the exact same structure)

When there are more variables

- The domain of the Constant Folding analysis is sets of bindings to values

  $\{v_1=c_1,\ v_2=c_2,\ v_3=c_3,\ldots\}$

  where the c-s are $\perp$, $\top$, or numbers

- Between two program points, the transfer function then takes us between

  $\{v_1=c_1,\ v_2=c_2,\ v_3=c_3,\ \ldots\}$

  and

  $\{v_1'=c_1',\ v_2=c_2'\ v_3=c_3',\ \ldots\}$

- Can we confidently say that

  $\{v_1=c_1,\ v_2=c_2,\ v_3=c_3,\ \ldots\} \sqsupseteq \{v_1'=c_1',\ v_2=c_2'\ v_3=c_3',\ \ldots\}$

  so that the transfer function will work towards a guaranteed, finite goal?

NTNU – Trondheim
Norwegian University of
Science and Technology

Products of lattices

- Lattices are partial orders, they consist of a set, and an order
  (which fulfills the constraint that all subsets have a g.l.b. and l.u.b.)
- The sets have Cartesian products
  $L_1 \times L_2 = \{ (x,y) \mid x \in L_1, y \in L_2 \}$
  $L_1 \times L_2 \times L_3 = \{ (x,y,z) \mid x \in L_1, y \in L_2, z \in L_3 \}$
  ...and so on…
- If $L_1, \dots L_n$ are (complete) lattices, their Cartesian product is a (complete) lattice as well, with the order defined so that the n-tuples
  $(y_1, y_2, \dots, y_n) \sqsupseteq (x_1, x_2, \dots, x_n)$
  if and only if
  $y_1 \sqsupseteq x_1, y_2 \sqsupseteq x_2, \dots, y_n \sqsupseteq x_n$
- In other words, if we apply a monotonic function to all the elements in the n-tuple from a lattice product, the n-tuples preserve the same order

The whole meet operator

- When two control paths meet up, their respective const-information sets might be something like

  $\{x = 3, y = \top, z = 5\}$

  and

  $\{x = 3, y = 2, z = \bot\}$

- The CF meet operator applies the constant-glb relation to all pairs

  $\{x = 3, y = \top, z = 5\}$

  $\sqcap \{x = 3, y = 2, z = \bot\}$

  $= \{x = 3, y = 2, z = \bot\}$

  $\mathrm{glb}(3,3) = 3$, $\mathrm{glb}(\top,2) = 2$, $\mathrm{glb}(5,\bot) = \bot$

NTNU – Trondheim
Norwegian University of
Science and Technology

Convergence

- The whole CF lattice is ordered by the relation from the constant-lattices of each of its variables
- The meet op. (glb) of the constant-ness states of one variable is monotonic
  - It never goes from “X = 24” to “X is still unknown” ($\top$)
  - It never goes from “X is not constant” ($\perp$) to “X is 62” either
- Therefore, the combination of individual meets for all the variables is monotonic also
  - Same rationale, it’s not going to go from a “more specific” point

    $\{x = 3, y = 2, z = \perp\}$

    to a “less specific” point like

    $\{x = 3, y = 2, z = 5\}$

    because that’s not what comes out of $\{z = \perp\} \sqcap \{z = 5\}$

The analyses we have seen

- Ok… to recap what we know about all this stuff now
  - Domains are made up of elements that represent information from the source code, they are sets of

    Live variables (Liveness)

    Pairs of variables (Copy Propagation)

    Expressions (Available Expressions)

    Definitions / assignments (Reaching Definitions)

    Constant-information about variables (Constant Folding)

Transfer functions

- Descriptions of how statements affect the sets at program points before and after

LV:	lv before = { lv after – var. defined } ⋃ { var. used }
CP:	copies after = { copies before - copies ruined } ⋃ { copies made }
AE:	expr. after = { expr. before – expr. ruined } ⋃ { expr. evaluated }
RD:	defs after = { defs before – defs overwritten } ⋃ { defs made }
CF:	const after = { const before – non-const found} ⋃ { const made }

or, with more conventional notation

LV:	in[I] = { out[I] - def(I) } ⋃ use(I)	(Backward)
CP:	out[I] = { in[I] – kill(I) } ⋃ gen(I)	(Forward)
AE:	out[I] = { in[I] – kill(I) } ⋃ gen(I)	(Forward)
RD:	out[I] = { in[I] – kill(I) } ⋃ gen(I)	(Forward)
CF:	out[I] = { in[I] – kill(I) } ⋃ gen(I)	(Forward)

(what each analysis kills and generates follows from how the instructions affect its domain)

Meet operators

- Descriptions of how to combine control flow paths, when they cross

LV:	$\cup$	(variables used along any path)
CP:	$\cap$	(copies made along every path)
AE:	$\cap$	(expressions available along every path)
RD:	$\cup$	(definitions coming from any path)
CF:	$\sqcap_{CF}$	(glb relation from constant-ness lattices)

NTNU – Trondheim
Norwegian University of
Science and Technology

Monotonicity

- Guarantee that iterating over the data flow equations take program points strictly toward one end of the domain's order
- The contributions from instructions are static, the source code doesn't change during analysis
- The meet operators only contribute in one direction

LV:	$x \cup y$	is glb in power set lattice of variables
CP:	$x \cap y$	is glb in power set lattice of copies
AE:	$x \cap y$	is glb in power set lattice of expressions
RD:	$x \cup y$	is glb in power set lattice of definitions
CF:	$x \sqcap_{CF} y$	is glb in the product of constant-lattices we discussed

- <u>None of these analyses will run forever</u>

NTNU – Trondheim
Norwegian University of
Science and Technology

Ups and downs

- Up until this point, I waved my hands at the beginning and pointed out that we can arrange our lattice orders
  - With ∅ at the bottom and the set all elements at the top
  - With ∅ at the top and the set of all elements at the bottom
  - With g.l.b. and l.u.b. determining the direction when points are combined
  - An idea of a “Top” (⊤) and “Bottom” (⊥)
  - Some matching, vague notion of “more” and “less” program information

  and suggested that all of these can be rearranged as a matter of notation
- I have played fast and loose with this because we haven’t said anything where it matters
  - Same kind of nuisance as talking about stacks that grow into lower addresses, it’s disruptive to stop and remember that up is down and plus is minus every 2 minutes

Making a choice

- Consistency matters more in an overview, so let's standardize it a bit
- Choose the top $\top$ to be the most an analysis can hope for
- Choose the meet operator $\sqcap$ to be the greatest lower bound of a lattice subset
- Choose the bottom $\perp$ to be the worst outcome

Why choose these?

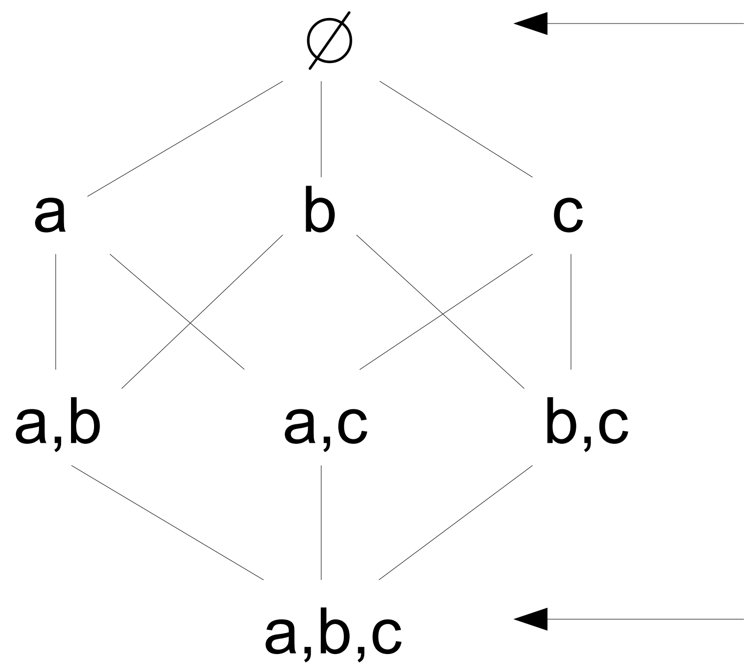
- The book draws with up/down in these directions (Fig. 9.22, p.622)
- We need a convention before discussing "precision"

- On the other hand
  - Several fixed points can solve the same system of constraint equations
  - The one that our iterative method finds is called the *maximal fixed point*
  - It is "maximal" in the sense of being at the end of a chain of states which is as long as possible
  - Paradoxically, that puts it closest to the order point called "bottom"

    (sigh)
  - That's the way it goes

Interpretations from top to bottom

For live variables:



Most useful:
No variables are needed

Most careful:
All variables are needed

NTNU – Trondheim
Norwegian University of Science and Technology

Interpretations from top to bottom

For available expressions:

$$\begin{tikzcd}
 & e_1,e_2,e_3 \arrow[dl, dash] \arrow[d, dash] \arrow[dr, dash] & & \text{Most useful: All expressions can be re-used} \arrow[ll] \\
e_1,e_2 \arrow[d, dash] \arrow[dr, dash] & e_1,e_3 \arrow[dl, dash] \arrow[dr, dash] & e_2,e_3 \arrow[dl, dash] \arrow[d, dash] & \\
e_1 \arrow[dr, dash] & e_2 \arrow[d, dash] & e_3 \arrow[dl, dash] & \\
 & \emptyset & & \text{Most careful: No expressions can be re-used} \arrow[ll]
\end{tikzcd}$$

Several solutions

- As a trivial example, take the "program" x = y+z, and consider liveness
  - We get 1 constraint equation: in = {out – x} U {y,z}
- Start from out = {x,y,z}

```
        {y,z} are live here
    x = y + z
        {x,y,z} are live here
```

- Start from out = {}

```
        {y,z} are live here
    x = y + z
        {} is live here
```

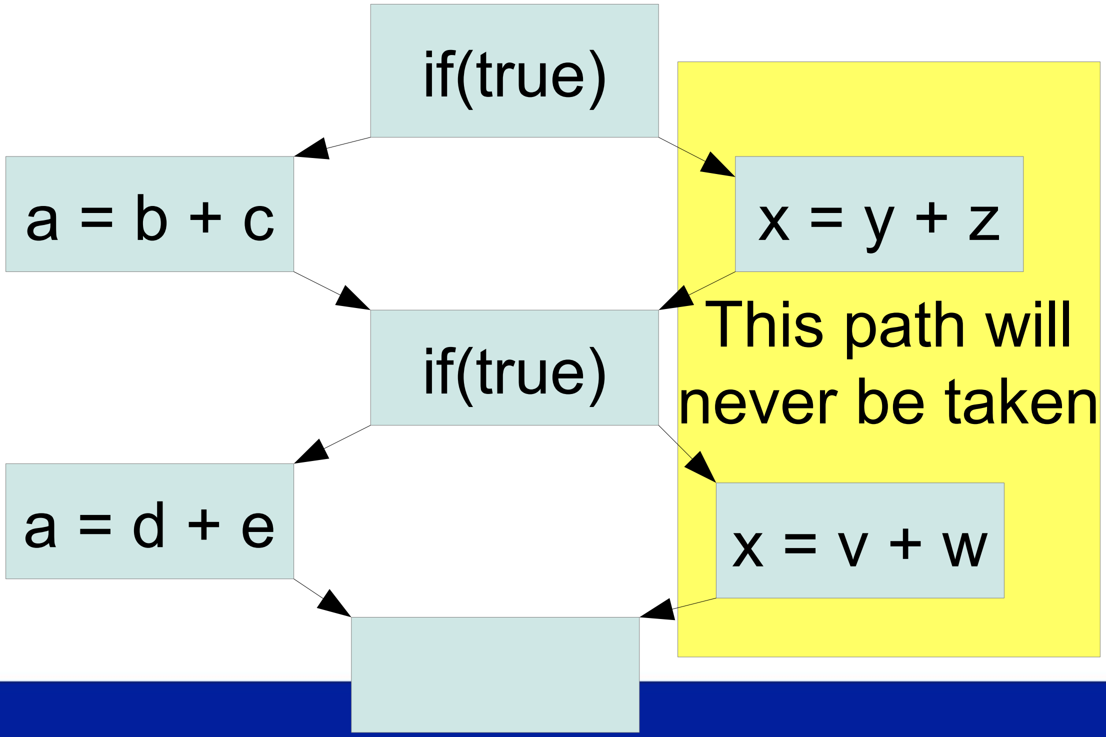
- These are both solutions to the data flow equation
- Apply the constraints again, nothing changes in either case

NTNU – Trondheim
Norwegian University of
Science and Technology

What's the *best* solution?

- That would be the one which captures what the program actually does:

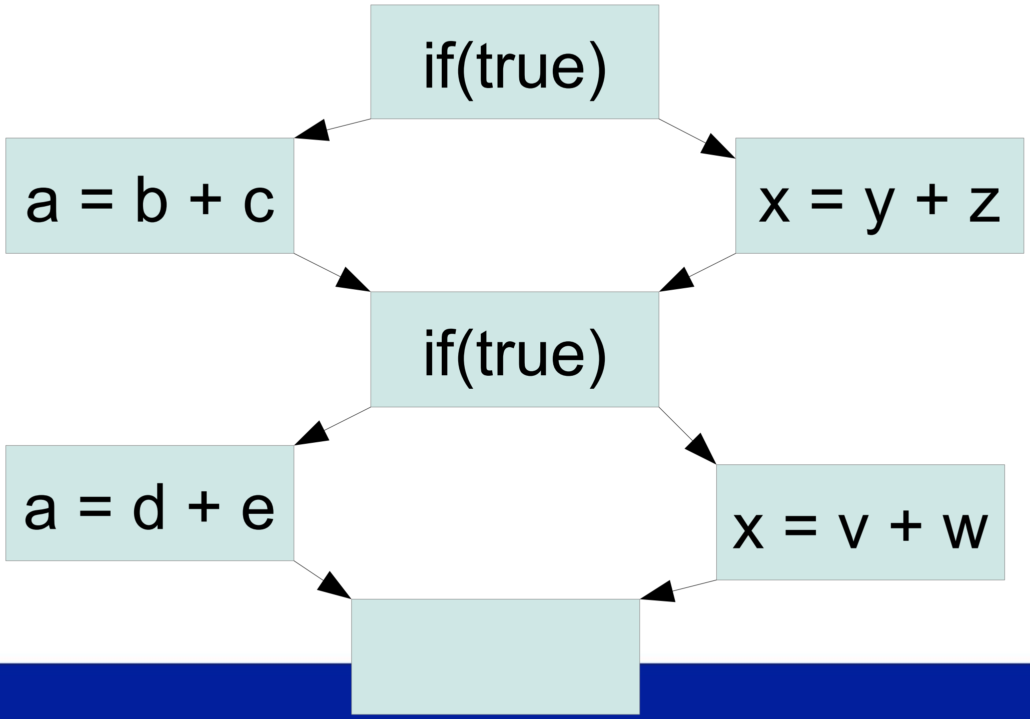
{b,c,d,e} live
{v,w,y,z} dead



Which solution does the framework suggest?

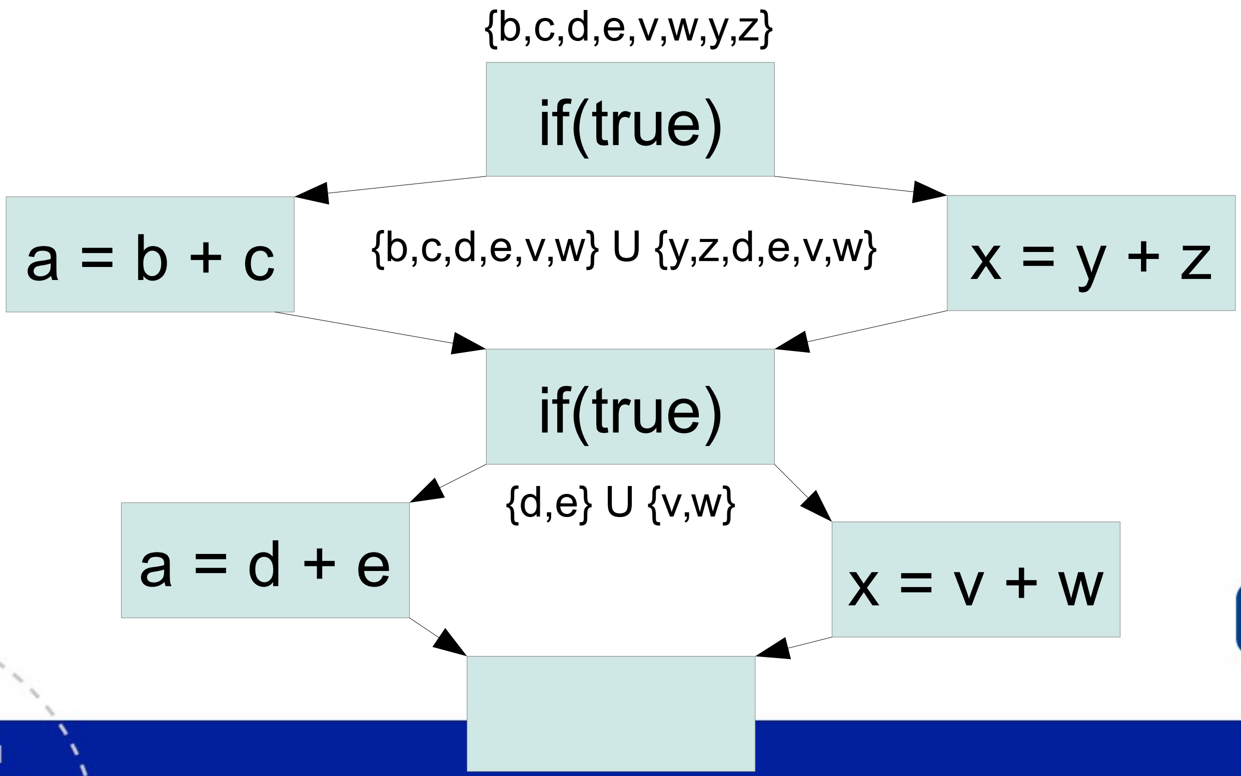
- That’s the one which comes from considering the meet operator applied to all possible paths

{b,c,d,e} U {b,c,v,w} U {y,z,d,e} U {y,z,v,w} = {b,c,d,e,v,w,y,z}



Which solution do we compute?

- The one that comes from starting every point at ⊤, and iterating with ⊓ until there's no change



Names for those

- In order, we can call them

IDEAL	(The one that accurately reflects the code)
Meet-Over-Paths	(The one that considers every path)
Maximal Fixed Point	(The one we get by iterating from $\top$)

- IDEAL is the *most precise* solution, because it would tell us exactly what the program means
  - Sadly, that can not be computed automatically
- MOP would be as close as we could get by static inspection
  - Trace every possible execution individually, apply $\sqcap$ between all
  - Sadly, we can’t compute that either
  - “Every possible execution” includes going through every (dynamically determined) loop once, two times, three times, … and on to infinity

Their relationship

- The solution we *do* get (the way we've been working), is the MFP

  The iterations for a point go through a descending chain

  $\top \sqsupseteq F(\top) \sqsupseteq F(F(\top)) \sqsupseteq \ldots \sqsupseteq$ MFP (← where we stop iterating)
- This is excessively careful
  - It combines paths as soon as possible, thereby losing precision
  - We'll see in a minute
- It's safe

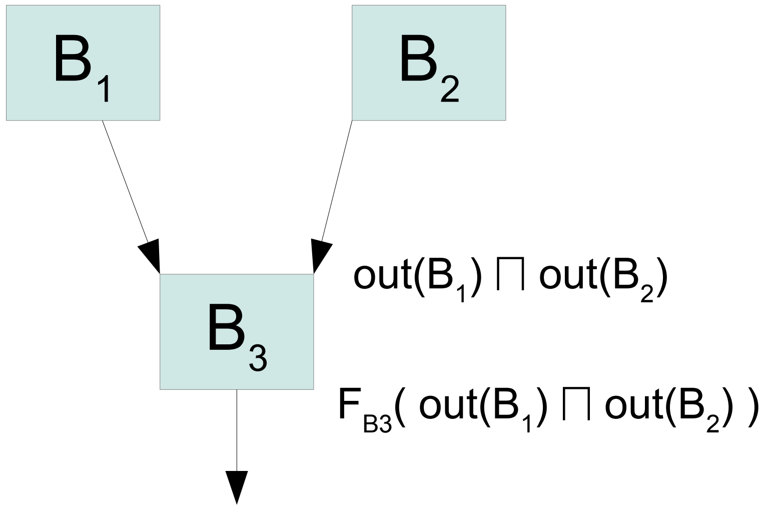
  MOP $\sqsupseteq$ MFP
  - They're often the same (as in all our examples so far)
  - When they differ, MOP is closer to the most useful end of the order
- MOP applies constraints along paths never taken, when there are any

  IDEAL $\sqsupseteq$ MOP $\sqsupseteq$ MFP

NTNU – Trondheim
Norwegian University of
Science and Technology

MFP evaluation

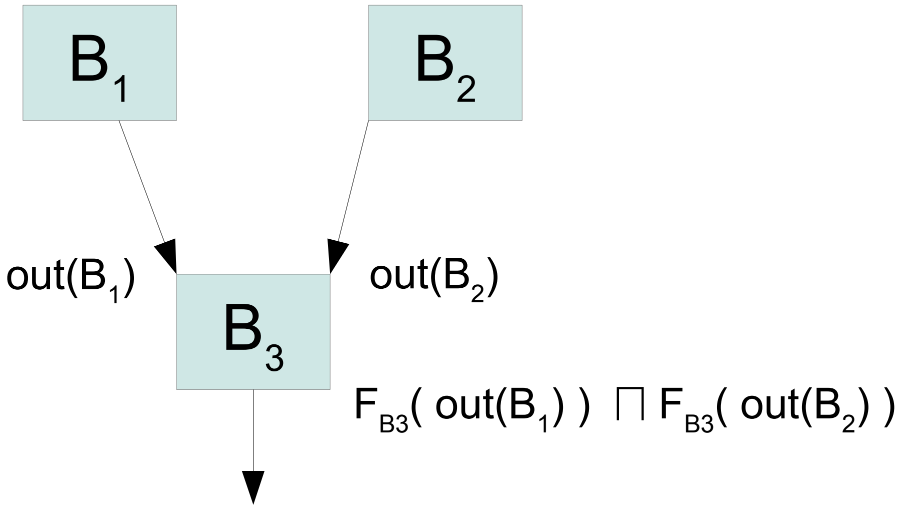
- MFP computes the function of B_3 on the combination of out(B_1) and out(B_2)



B_1
B_2
B_3
$out(B_1) \sqcap out(B_2)$
$F_{B3}(out(B_1) \sqcap out(B_2))$

MOP evaluation

- MOP computes the function of B_3 by combining
  - B_3s effect on out(B_1)
  - B_3s effect on out(B_2)


B_1
B_2
out(B_1)
out(B_2)
B_3
$F_{B3}(\text{ out}(B_1)\text{ }) \sqcap F_{B3}(\text{ out}(B_2)\text{ })$

NTNU – Trondheim
Norwegian University of Science and Technology

Distributivity

- If F is a distributive function *wrt.* $\sqcap$, then

  F (x $\sqcap$ y) = F(x) $\sqcap$ F(y)

  (that's the definition of distributive)
- When the function representing an analysis has this property, then the MFP solution (we can compute) is the same as the MOP solution (we can't compute)
- When
  - the function is just adding and removing elements to sets
  - the operator is just simple combinations of set elements

  distributivity follows

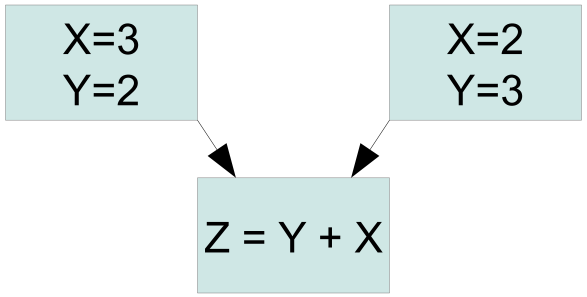
  If F is something like "delete element x", then practically by common sense,

  F({x,y,z} U {v,w,x}) = {v,w,y,z}

  F({x,y,z}) U F({v,w,x}) = {y,z} U {v,w} = {v,w,y,z}

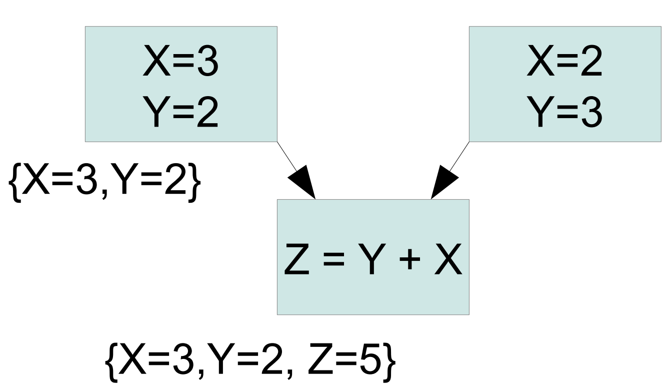
Distributivity *vs* Constant Folding

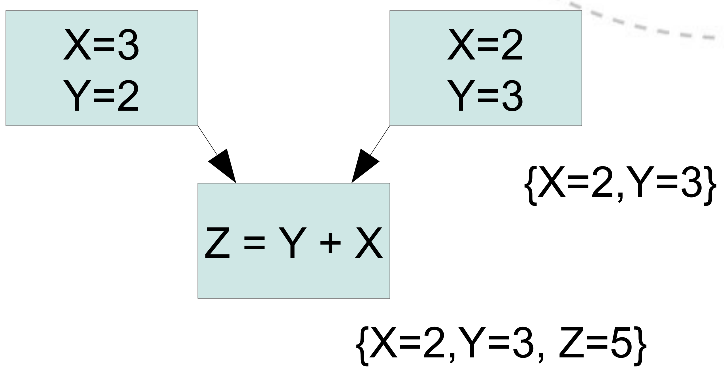
- LV, CP, AE, RD all give MFP=MOP, because their functions are distributive *wrt.* their respective union/intersection meet operators
- The constant-detecting scheme is <u>not</u> distributive *wrt.* its funny meet operator
- Witness:



Distributivity *vs* Constant Folding

- By Meet-Over-Paths, here are the two paths





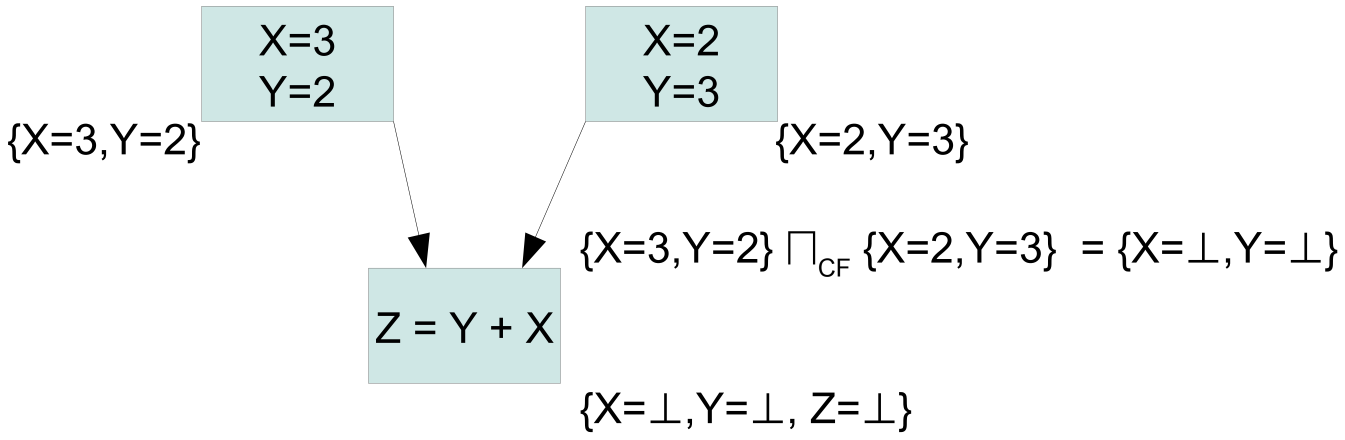
This gives the MOP solution

$$\{X=3,Y=2,\ Z=5\} \sqcap_{CF} \{X=2,Y=3,\ Z=5\} \ = \ \{X=\perp,\ Y=\perp,\ \underline{\mathbf{Z = 5}}\ \}$$

NTNU – Trondheim
Norwegian University of
Science and Technology

Distributivity *vs* Constant Folding

- The Maximal Fixed Point solution is less informative, it misses that Z=5 regardless of which way it's calculated



X=3
Y=2
X=2
Y=3
{X=3,Y=2}
{X=2,Y=3}
{X=3,Y=2} $\sqcap_{CF}$ {X=2,Y=3} = {X=⊥,Y=⊥}
Z = Y + X
{X=⊥,Y=⊥, Z=⊥}