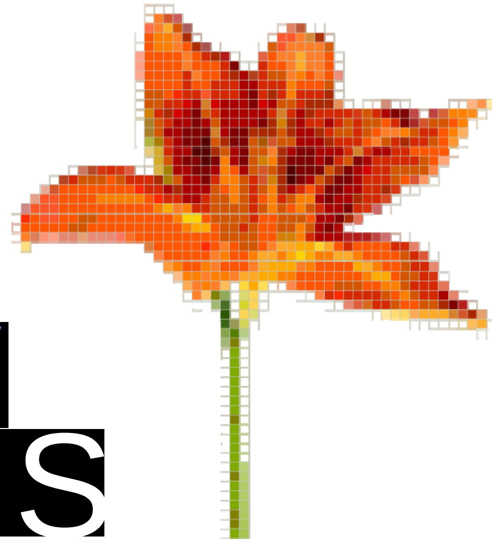
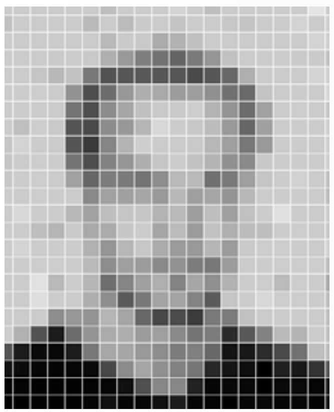
DATA-FLOW ANALYSIS



[HOW TO ANALYZE LANGUAGES AUTOMATICALLY]



Claus Brabrand

(((**brabrand@itu.dk**)))

Associate Professor, Ph.D.

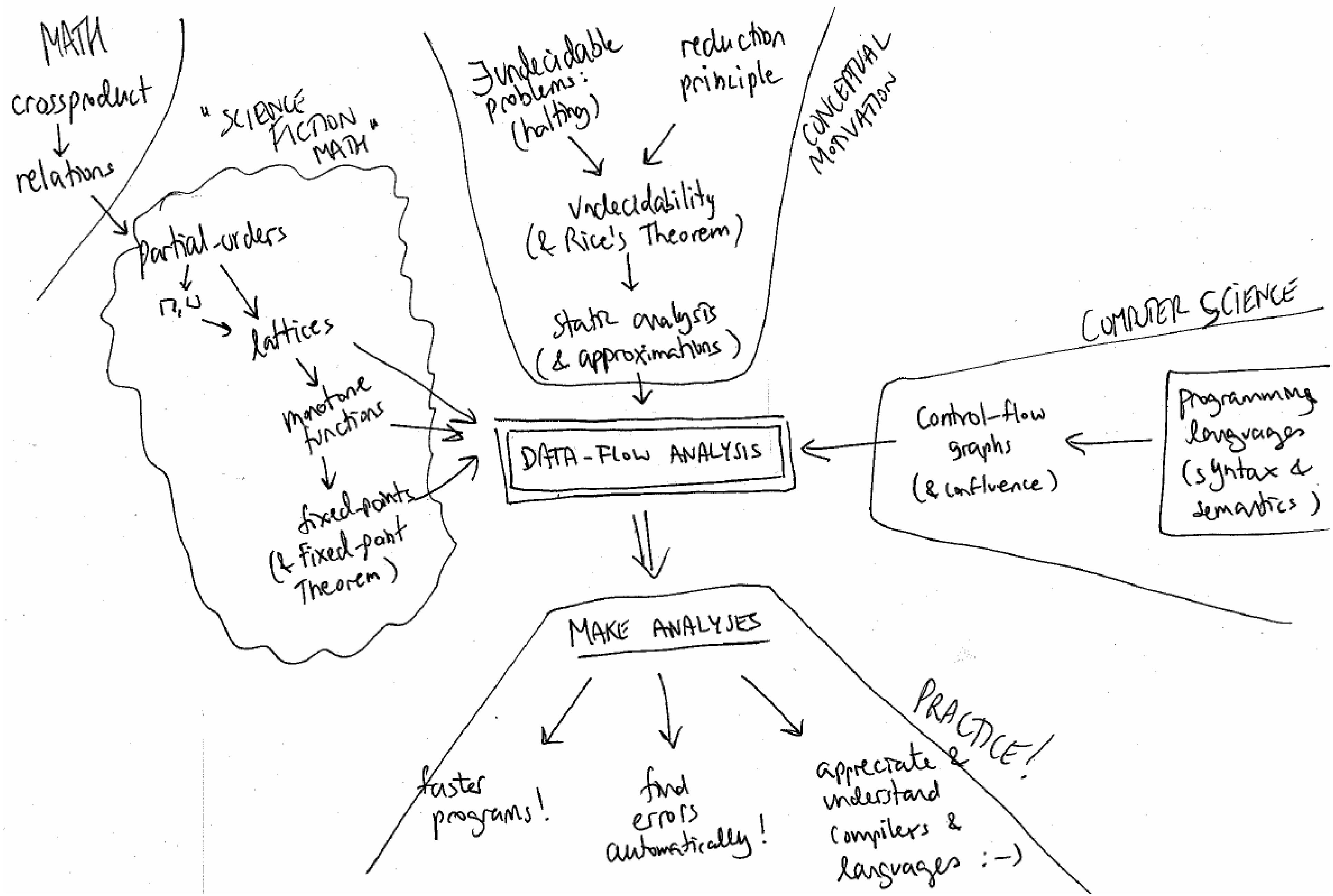
(((Programming, Logic, and Semantics)))

IT University of Copenhagen

⊢ Agenda

- Quick recap (of everything so far):
- *"Putting it all together"* → ***Data-Flow Analysis***
- Fixed-Point Iteration Strategies (3x)
- "Sign Analysis"
- "Constant Propagation Analysis"
- "Initialized Variables Analysis"
- Set-Based Analysis Framework
- WORKSHOP

3 Example Data-Flow Analyses



MATH
crossproduct
relations
"SCIENCE FICTION MATH"
partial-orders
⊓, ⊔
lattices
monotone functions
fixed-points
(& Fixed-point Theorem)
∃ Undecidable problems: (halting)
reduction principle
CONCEPTUAL MOTIVATION
Undecidability
(& Rice's Theorem)
static analysis
(& approximations)
DATA-FLOW ANALYSIS
COMPUTER SCIENCE
Control-flow graphs
(& confluence)
Programming languages
(syntax & semantics)
MAKE ANALYSES
PRACTICE!
faster programs!
find errors automatically!
appreciate & understand compilers & languages :-)

⊢ Agenda

- Quick recap (of everything so far):
- *"Putting it all together"* → ***Data-Flow Analysis***
- Fixed-Point Iteration Strategies (3x)
- "Sign Analysis"
- "Constant Propagation Analysis"
- "Initialized Variables Analysis"
- Set-Based Analysis Framework
- WORKSHOP

3 Example Data-Flow Analyses (covering "Sign Analysis", "Constant Propagation Analysis" and "Initialized Variables Analysis")

⊢ All you need is...:

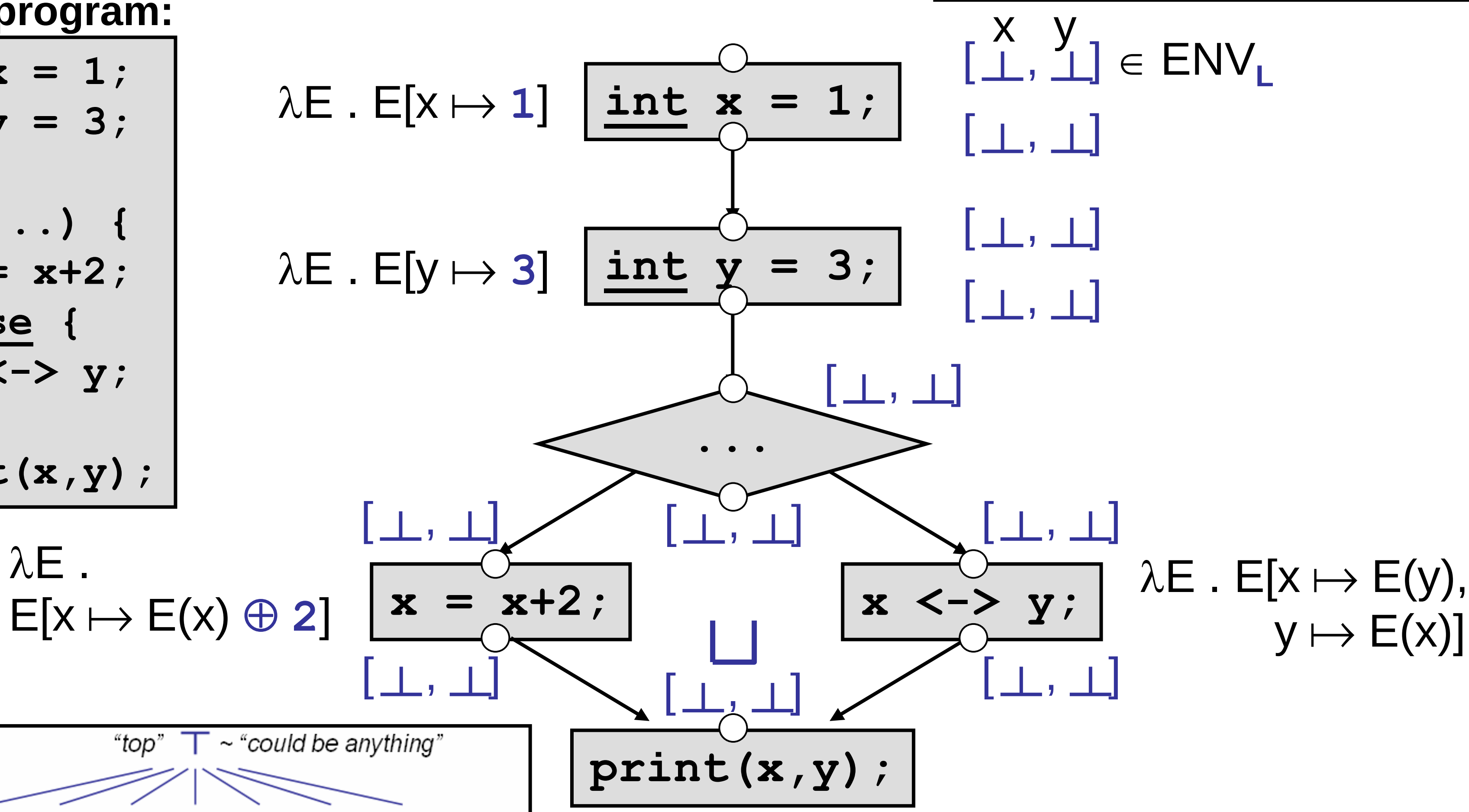
We (only) need 3 things:

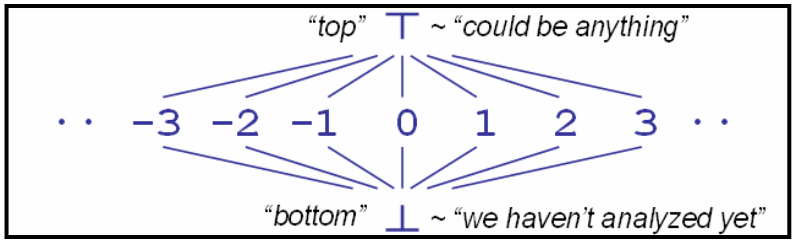
- A ***control-flow graph***
- A ***lattice***
- ***Transfer functions***

Given program:

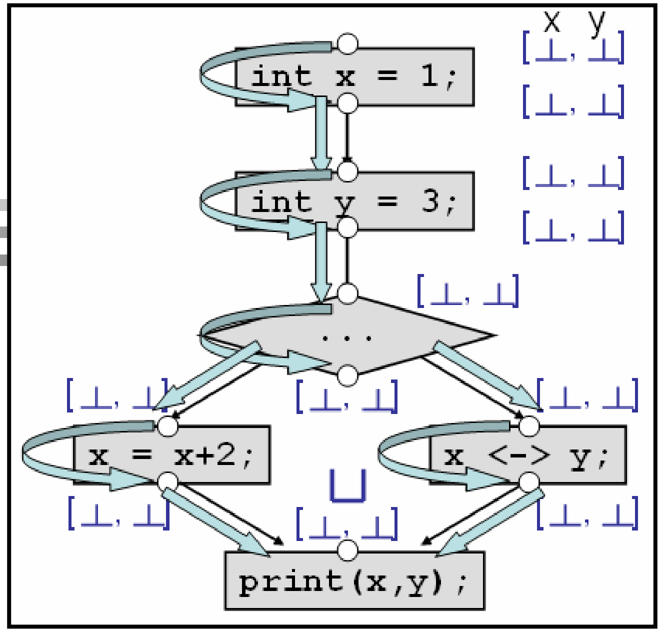
```
int x = 1;
int y = 3;

if (...) {
  x = x+2;
} else {
  x <-> y;
}
print(x,y);
```

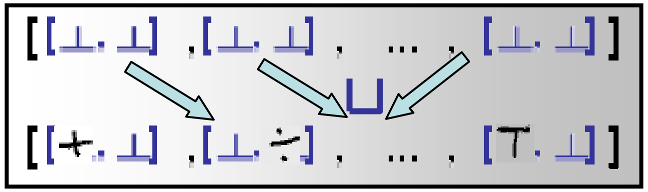




⊢ Solve Equations :-)

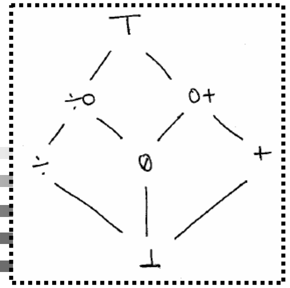


- One ***big*** lattice:
  - E.g., $(L^{|VAR|})^{|PP|}$
- 1 ***big*** abstract value vector:
  - [[⊥, ⊥] , [⊥, ⊥] , ... , [⊥, ⊥]] $\in (L^{|VAR|})^{|PP|}$
- 1 ***big*** transfer function:
  - **T** : $(L^{|VAR|})^{|PP|} \to (L^{|VAR|})^{|PP|}$



- Compute fixed-point (simply):
  - Start with bottom value vector ($\perp_{(L^{|VAR|})^{|PP|}}$)
  - Iterate transfer function '**T**' (until nothing changes)
  - Done; print out (or use) solution…! **:-)**

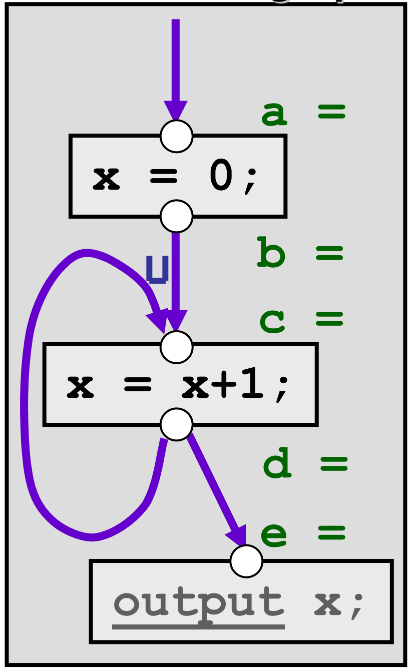
⊢ The Entire Process :-)



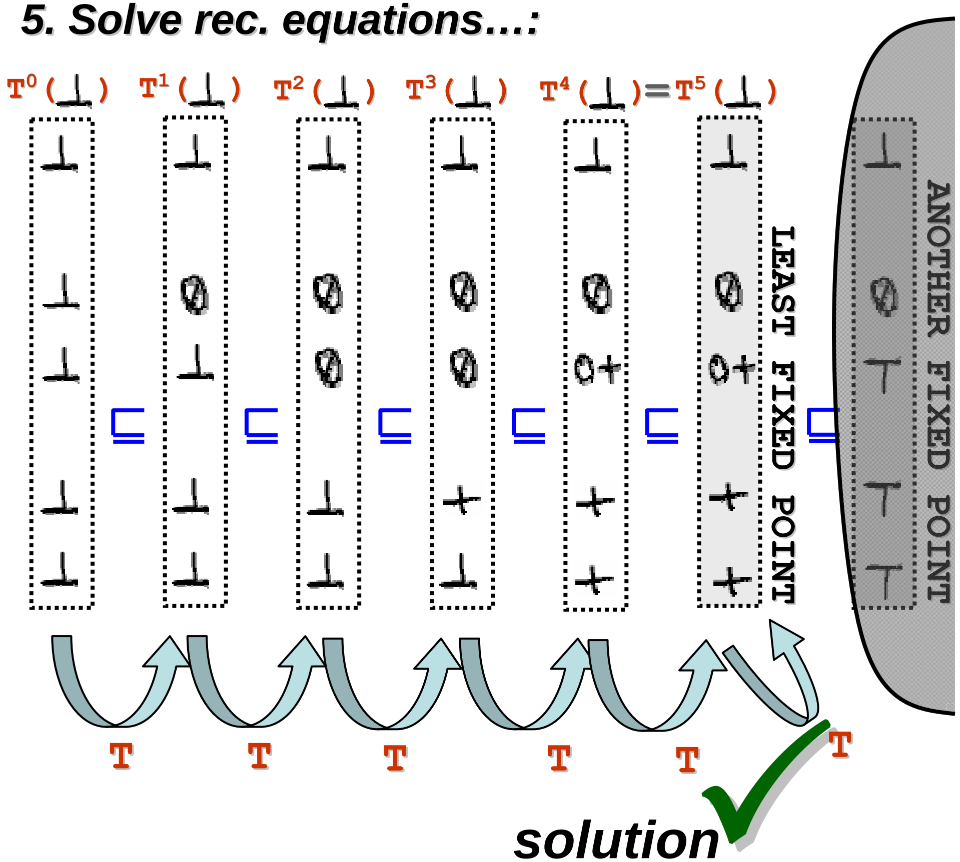
Program:

```
x = 0;
do {
   x = x+1;
} while (...);
output x;
```

1. Control-flow graph:



5. Solve rec. equations...:



3. Recursive equations:

$a = \bot$
$b = f_{x=0}(a)$
$c = b \sqcup d$
$d = f_{x=x+1}(c)$
$e = d$

≀

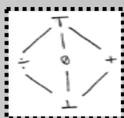
2. Transfer functions:

$f_{x=0}(\ell) = 0$
$f_{x=x+1}(\ell) = \ell \oplus_L +$

4. one "big" transfer function:

$T((a,b,c,d,e)) = (\bot, f_{x=0}(a), b \sqcup d, f_{x=x+1}(c), d)$

...over a "big" power-lattice:

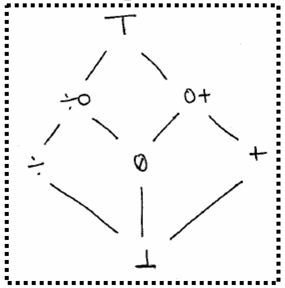


$|VAR|*|PP| = 1*5 = 5$

⊢ Exercise:

- Repeat this process for program (of two vars):
  -
```
x = 1;
y = 0;
while (v>w) {
  x <-> y;
}
y = y+1;
```

…using lattice:



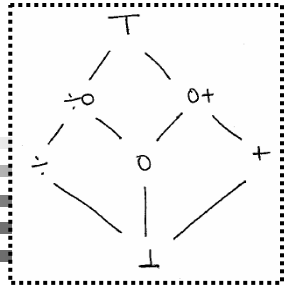
- i.e., *determine*…:
  - **1**) ***Control-flow graph***
  - **2**) ***Transfer functions***
  - **3**) ***Recursive equations***
  - **4**) ***One ”big” transfer function***
  - **5**) ***Solve recursive equations*** **:-)**

⊢ Agenda

- Quick recap (of everything so far):
- *"Putting it all together" → **Data-Flow Analysis***
- Fixed-Point Iteration Strategies (3x)
- "Sign Analysis"
- "Constant Propagation Analysis"
- "Initialized Variables Analysis"
- Set-Based Analysis Framework
- WORKSHOP

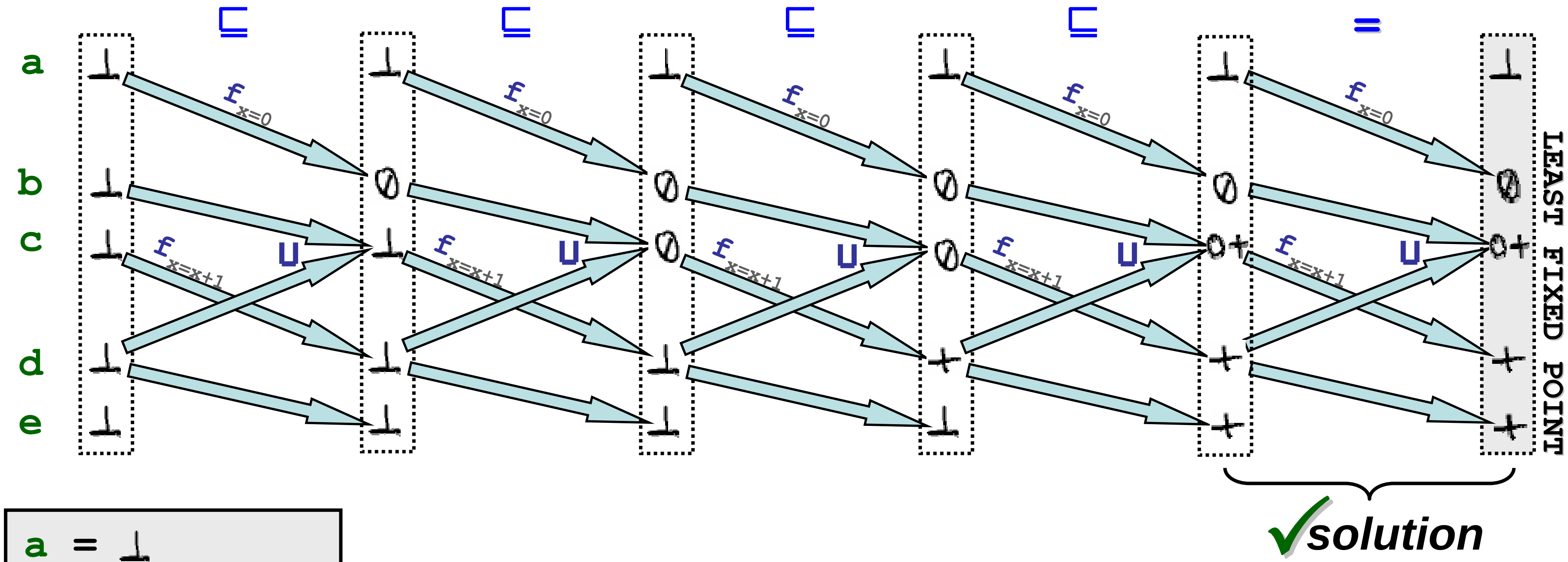
3 Example Data-Flow Analyses

Naïve Fixed-Point Algorithm



Naïve Fixed-Point Algorithm:

- *…uses intermediate ”results” from **previous** iteration:*

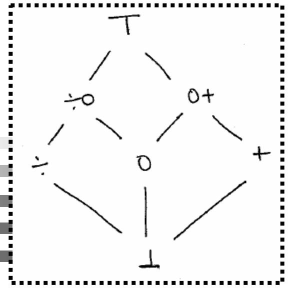


$a = \bot$
$b = f_{x=0}(a)$
$c = b \sqcup d$
$d = f_{x=x+1}(c)$
$e = d$

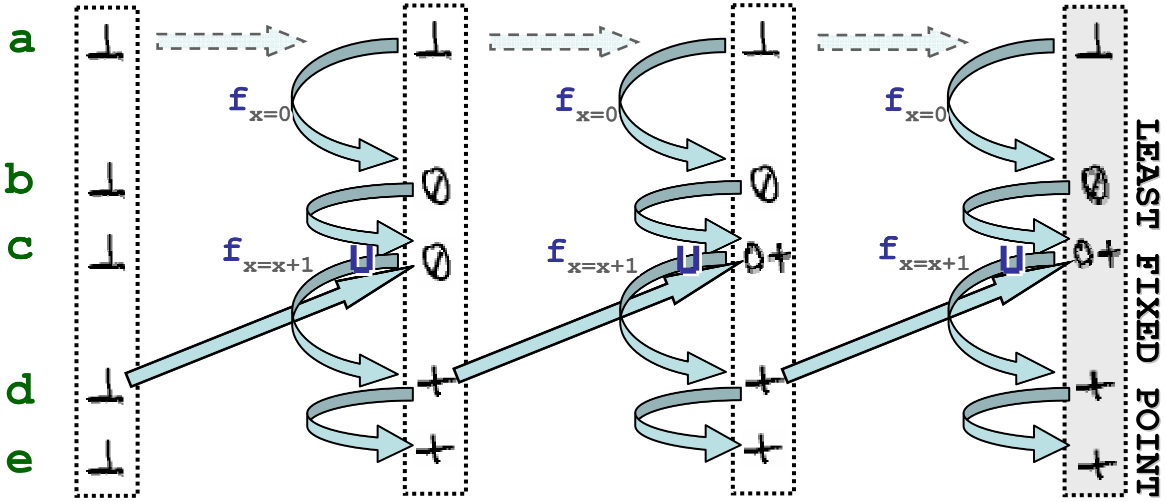
Slow!

$f_{x=0}(\ell) = +$
$f_{x=x+1}(\ell) = \ell \oplus_L +$

⊢ Chaotic Iteration Algorithm



- ***Chaotic Iteration Algorithm*:**
  - *…exploits "**forward nature**" of program control flow:*



✓*solution*

```
a = ⊥
b = f_{x=1}(a)
c = b ⊔ d
d = f_{x=x+1}(c)
e = d
```

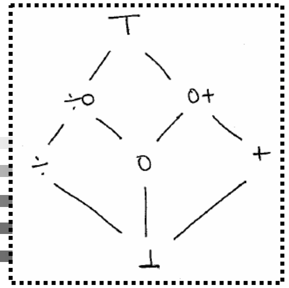
Faster!

(always uses "latest" results)

$f_{x=1}(\ell) = +$

$f_{x=x+1}(\ell) = \ell \oplus_L +$

⊢ Work-list Algorithm



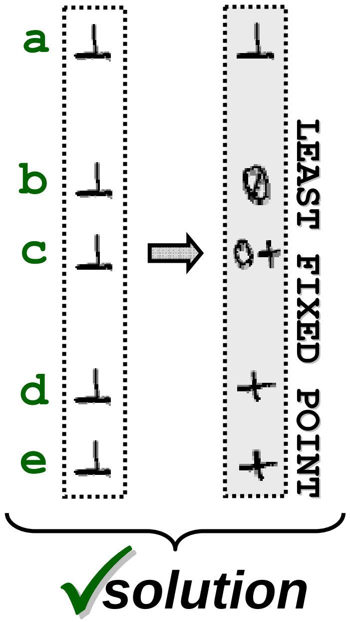
- ***Work-list Algorithm:***
  - *…uses a "**queue**" to control (optimize) computation:*

Initialize queue with start point:
Q := [a]

Pop top element from queue and (re-)compute it; IF it changed THEN enqueue all points that depend on it's value (if it isn't already on the queue)

Stop when queue is empty

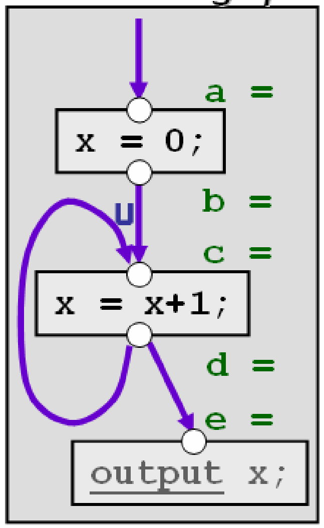
```
a = ⊥
b = f_x=1 (a)
c = b ⊔ d
d = f_x=x+1 (c)
e = d
```



Fastest!
(in general)

Queue:
[a]
↓
[b]
↓
[c]
↓
[d,c]
↓
[c,e]
↓
⋮
↓
[]

Control-flow graph:



$f_{x=1}(\ell) = +$
$f_{x=x+1}(\ell) = \ell \oplus_L +$

⊢ Agenda

- Quick recap (of everything so far):
- *"Putting it all together"* → ***Data-Flow Analysis***
- Fixed-Point Iteration Strategies (3x)
- "Sign Analysis"
- "Constant Propagation Analysis"
- "Initialized Variables Analysis"
- Set-Based Analysis Framework
- WORKSHOP

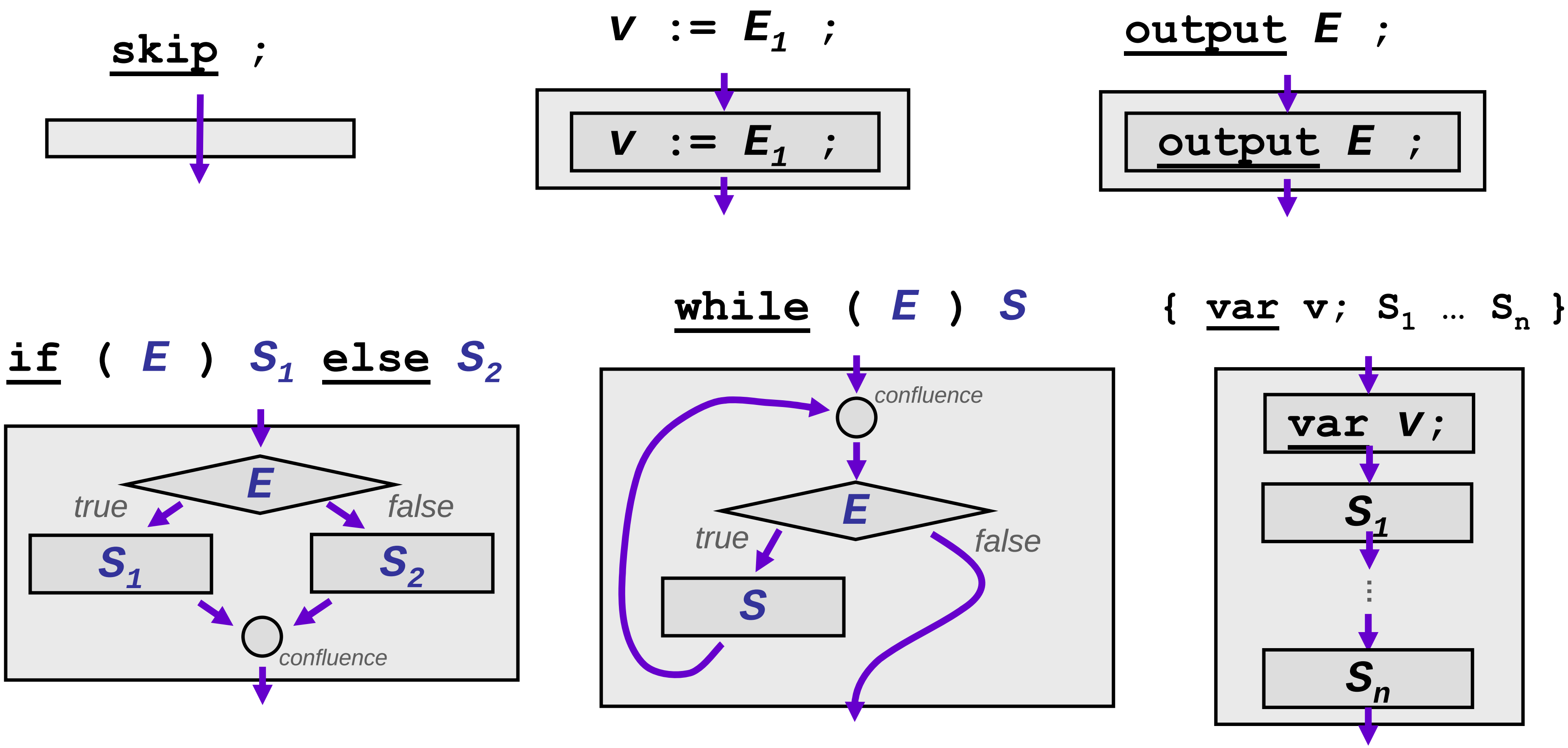
3 Example Data-Flow Analyses

⊢ The Language 'C--'

- Syntactic Categories:
  - *Expressions* (**E** ∈ **EXP**):
    -
    ```
    E  :  n  |  v  |  E + E'  |  - E
          |  E * E'  |  E == E'  |  input
    ```
  - *Statements* (**S** ∈ **STM**):
    -
    ```
    S  :  skip ;  |  v := E ;  |  output E ;
          |  if E then S else S'
          |  while E do S  |  { var v; S1 ... Sn }
    ```
    - (…assume we only have integer variables '**x**', '**y**', '**z**')

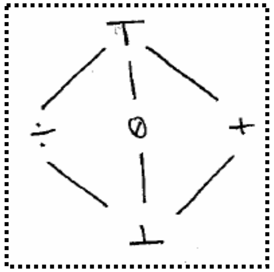
⊢ Control-Flow Graph (for 'C--')

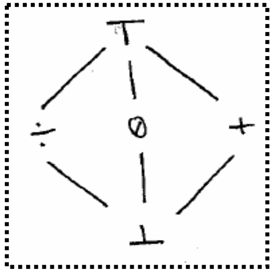
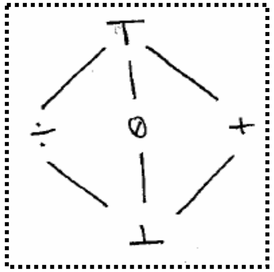
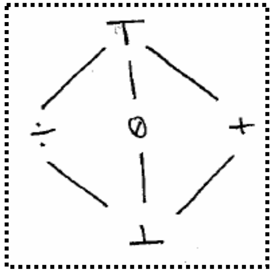
- Inductively defined ***control-flow graph*:**



⊢ Sign Analysis: Lattice

- Lattice:

  L_{SIGN}: 

- $ENV_{Lattice}$:  ×  ×  $\approx L_{SIGN}^{|VAR|}$ $\approx VAR \rightarrow L_{SIGN}$

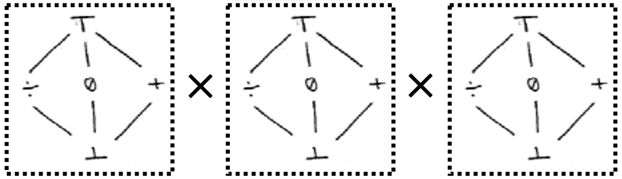
  ↑ `x`   ↑ `y`   ↑ `z`

- Confluence operator:
  - $\sqcup$ = '($\sqcup$, $\sqcup$, $\sqcup$)' *(pairwise)*

    ↑ `x`   ↑ `y`   ↑ `z`

Sign Analysis: Transfer F's

- Transfer Functions:



$\lambda Env \,.\, Env[x \mapsto \top]$

```
var x;
```

$\lambda Env \,.\, Env$

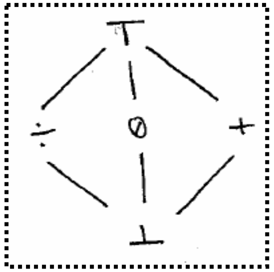
```
output … ;
```

$\lambda Env \,.\, Env[x \mapsto sign(Env, \textbf{Exp})]$

```
x := Exp ;
```

Inductive definition of '*sign*' in the syntactic structure of Exp

- Syntax: `E : n | input | v | E + E' | E * E' | E == E' | - E`



  - *sign*(*Env*, **n**) = +
  - *sign*(*Env*, **input**) = ⊤
  - *sign*(*Env*, **v**) = *Env*(**v**)
  - *sign*(*Env*, E_1+E_2) = *sign*(*Env*, E_1) $\oplus_L$ *sign*(*Env*, E_2)
  - *sign*(*Env*, **-E**) = 0 $\ominus_L$ *sign*(*Env*, **E**)
  - …

+	⊥	0	-	+	?
⊥	⊥	⊥	⊥	⊥	⊥
0	⊥	0	-	+	?
-	⊥	-	-	?	?
+	⊥	+	?	+	?
?	⊥	?	?	?	?

-	⊥	0	-	+	?
⊥	⊥	⊥	⊥	⊥	⊥
0	⊥	0	+	-	?
-	⊥	-	?	-	?
+	⊥	+	+	?	?
?	⊥	?	?	?	?

*	⊥	0	-	+	?
⊥	⊥	0	⊥	⊥	⊥
0	0	0	0	0	0
-	⊥	0	+	-	?
+	⊥	0	-	+	?
?	⊥	0	?	?	?

==	⊥	0	-	+	?
⊥	⊥	⊥	⊥	⊥	⊥
0	⊥	+	0	0	?
-	⊥	0	?	0	?
+	⊥	0	0	?	?
?	⊥	?	?	?	?

Exercise:

- Come up with a ***program*** the analysis…
- **A)**
  - ***can*** analyse precisely
- **B)**
  - ***can't*** analyse precisely

Agenda

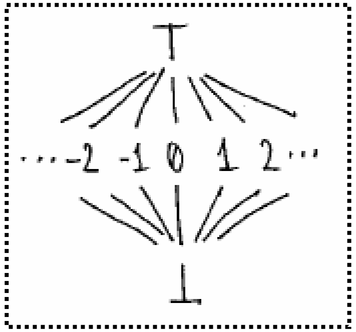
- Quick recap (of everything so far):
- *"Putting it all together"* → ***Data-Flow Analysis***
- Fixed-Point Iteration Strategies (3x)
- "Sign Analysis"
- "Constant Propagation Analysis"
- "Initialized Variables Analysis"
- Set-Based Analysis Framework
- WORKSHOP

3 Example Data-Flow Analyses

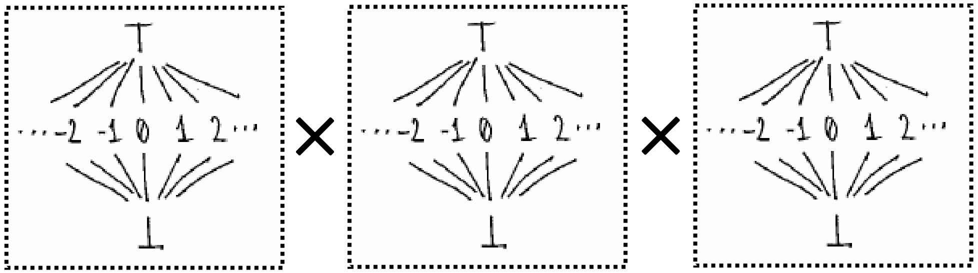
Const Propagation: Lattice

- Lattice:

L_{NUM}:



- $ENV_{Lattice}$:



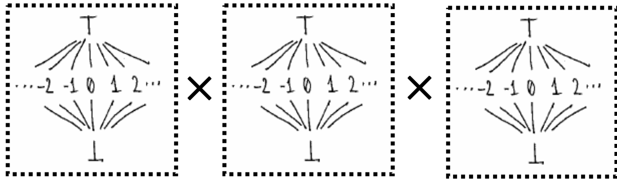
$\approx \; L_{NUM}^{|VAR|}$
$\approx \; VAR \to L_{NUM}$

- Confluence operator:
  - $\bigsqcup$ = '($\sqcup$, $\sqcup$, $\sqcup$)' *(pairwise)*

    ↑ ↑ ↑
    x y z

⊢ Const Propagation: Transfer F's

- Transfer Functions:



$\lambda Env \,.\, Env[x \mapsto \top]$ — `var x;`

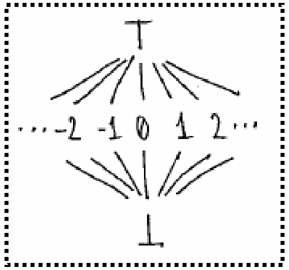
$\lambda Env \,.\, Env$ — `output … ;`

$\lambda Env \,.\, Env[x \mapsto eval(Env, Exp)]$ — `x := Exp ;`

Inductive definition of '*eval*' in the syntactic structure of Exp

- Syntax:

```
E  :  n  | input  |  v  |  E + E'
      |  E * E'  |  E == E'  |  - E
```



- *eval*(*Env*, **n**) = n
- *eval*(*Env*, **input**) = ⊤
- *eval*(*Env*, **v**) = *Env*(**v**)
- *eval*(*Env*, $\mathbf{E_1}$+$\mathbf{E_2}$) = *eval*(*Env*, $\mathbf{E_1}$) $\oplus_L$ *eval*(*Env*, $\mathbf{E_2}$)
- *eval*(*Env*, **-E**) = 0 $\ominus_L$ *eval*(*Env*, **E**)
- …

$$\mathbf{n} \oplus_L \mathbf{m} = \begin{cases} \top & \text{, if } \mathbf{n} = \top \vee \mathbf{m} = \top \\ \bot & \text{, if } \mathbf{n} = \bot \vee \mathbf{m} = \bot \\ \mathbf{r} & \text{, o/w (where } \mathbf{r} = \mathbf{n} + \mathbf{m}\text{)} \end{cases}$$

…i.e.:

Lplus	⊥	…	-1	0	+1	…	⊤
⊥	⊥	…	⊥	⊥	⊥	…	⊥
⋮	⋮	⋱	⋮	⋮	⋮		⋮
-1	⊥	…	-2	-1	0	…	⊤
0	⊥	…	-1	0	1	…	⊤
+1	⊥	…	0	1	2	…	⊤
⋮	⋮		⋮	⋮	⋮	⋱	⋮
⊤	⊥	…	⊤	⊤	⊤	…	⊤

Exercise:

- Come up with a ***program*** the analysis…
- **A)**
  - ***can*** analyse precisely
- **B)**
  - ***can't*** analyse precisely

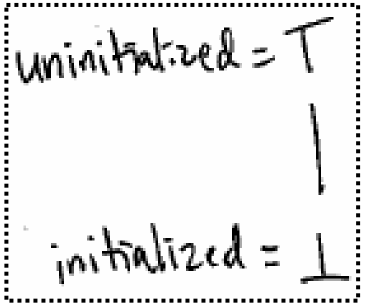
⊢ Agenda

- Quick recap (of everything so far):
- *"Putting it all together"* → ***Data-Flow Analysis***
- Fixed-Point Iteration Strategies (3x)
- "Sign Analysis"
- "Constant Propagation Analysis"
- "Initialized Variables Analysis"
- Set-Based Analysis Framework
- WORKSHOP

3 Example Data-Flow Analyses

⊢ Initialized Variables Analysis

- Lattice:

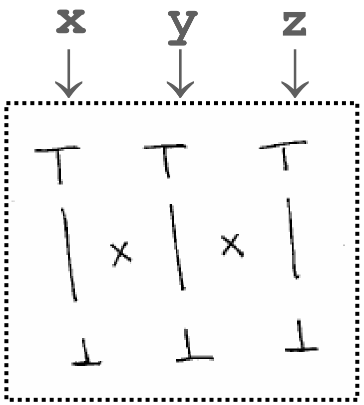


← *possibly uninitialized*

Note: It's always "safe" to answer "too high"

← *definitely initialized*

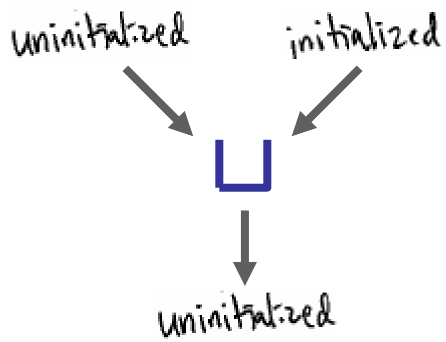
- ENV$_{Lattice}$:



- Confluence operator:
  - ⊔ = '(⊔,⊔,⊔)' *(pairwise)*

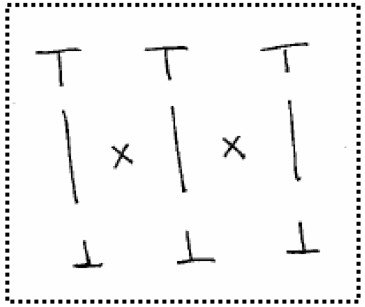
    ↑ ↑ ↑

    x y z



⊢ Initialized Variables Analysis

- Transfer Functions:



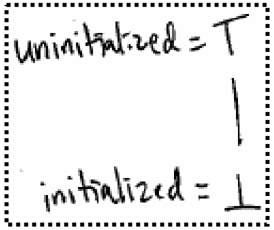
$\lambda Env \,.\, Env[x \mapsto \top]$ — **`var x;`**

$\lambda Env \,.\, Env$ — **`output … ;`**

$\lambda Env \,.\, Env[x \mapsto init(Env, \mathbf{Exp})]$ — **`x := … ;`**

⊢ Inductive definition of '*init*' in the syntactic structure of Exp

- Syntax: `E : n | input | v | E + E' | E * E' | E == E' | - E`

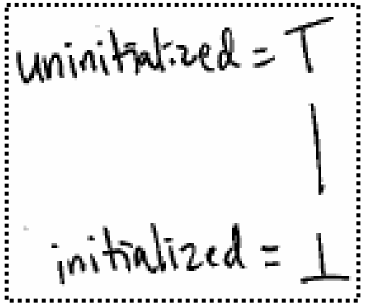


  - *init*(*Env*, **n**) = $\bot$
  - *init*(*Env*, **input**) = $\bot$
  - *init*(*Env*, **v**) = *Env*(**v**)
  - *init*(*Env*, $\mathbf{E_1 + E_2}$) = *eval*(*Env*, $\mathbf{E_1}$) $\oplus_L$ *eval*(*Env*, $\mathbf{E_2}$)
  - *init*(*Env*, **-E**) = $\bot \ominus_L$ *eval*(*Env*, **E**)
  - …

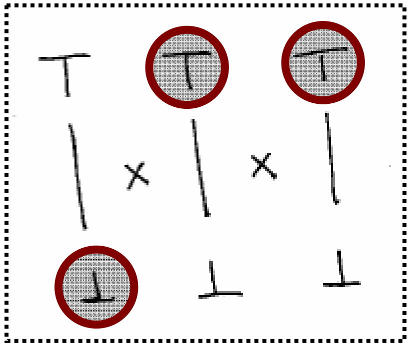
$\oplus_L$	$\bot$	$\top$
$\bot$	$\bot$	$\top$
$\top$	$\top$	$\top$

Note: Isomorphism!

- With value lattice:

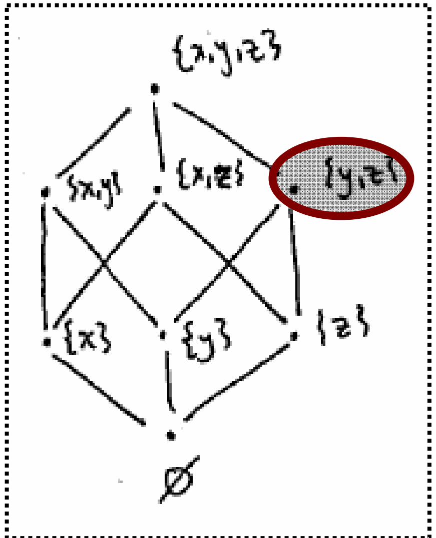


- ENV-lattice is isomorphic to:
  - …for every program point:



'**x**' 'y' 'z'

≈*isomorphic*

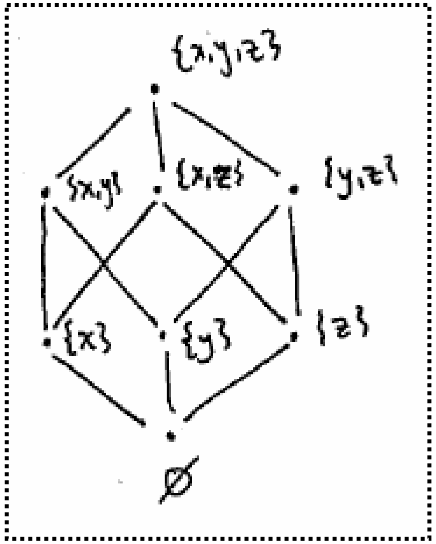


$Init_{VARS} \subseteq \{\mathbf{x},\mathbf{y},\mathbf{z}\}$

Vars that are possibly uninitialized

Initialized Variables Analysis (Revisited)

- ENV-lattice:

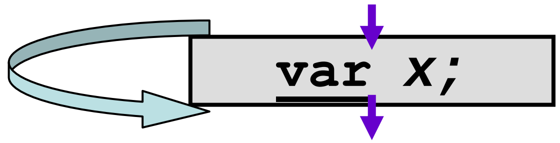


Vars that are possibly uninitialized

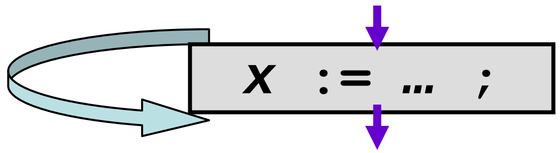
- Confluence operator:
  - $\sqcup$ = ' $\cup$ ' *(i.e., set union)*

- Transfer Functions:

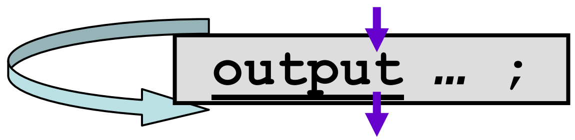
$\lambda S \,.\, S \cup \{\mathbf{x}\}$



$\lambda S \,.\, S \setminus \{\mathbf{x}\}$



$\lambda S \,.\, S$



Exercise:

- Come up with a ***program*** the analysis…
- **A)**
  - ***can*** analyse precisely
- **B)**
  - ***can't*** analyse precisely

⊢ Agenda

- Quick recap (of everything so far):
- *"Putting it all together"* → ***Data-Flow Analysis***
- Fixed-Point Iteration Strategies (3x)
- "Sign Analysis"
- "Constant Propagation Analysis"
- "Initialized Variables Analysis"
- Set-Based Analysis Framework
- WORKSHOP

3 Example Data-Flow Analyses

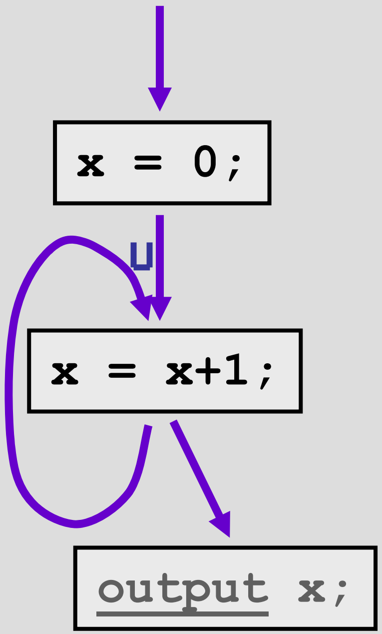
Set-based analyses...

$\{may, must\} \times \{forwards, backwards\}$

⊢ Forwards vs. Backwards?

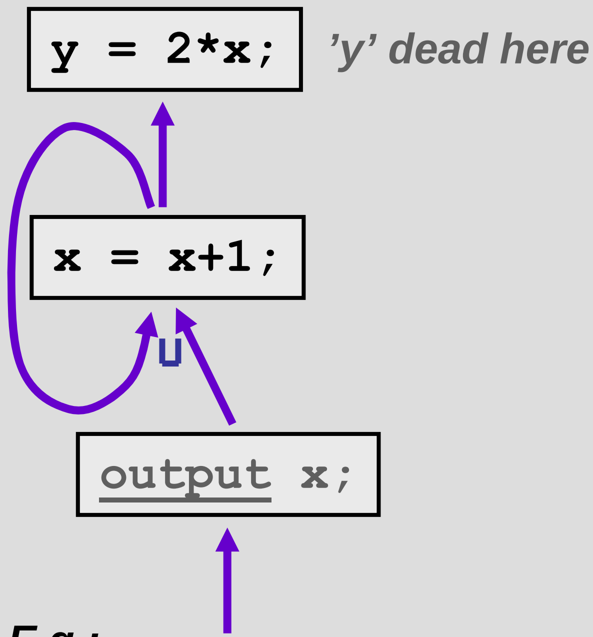
Analyze info that depends on ***past behavior***

- What you have seen:
  - Forwards:



Analyze info that depends on ***future behavior***

- Some analyses…:
  - Backwards:

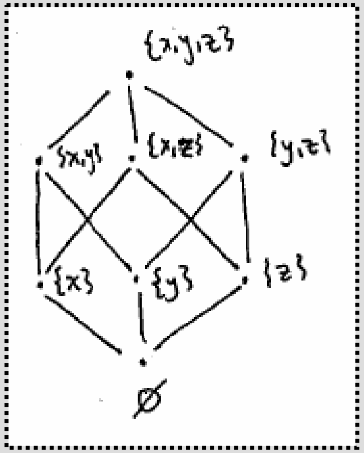


E.g.:
- Live Variables
- Very Busy Expressions

⊢ May vs. Must?

Analyze info that ***may possibly*** be true ∀paths

- What you have seen:

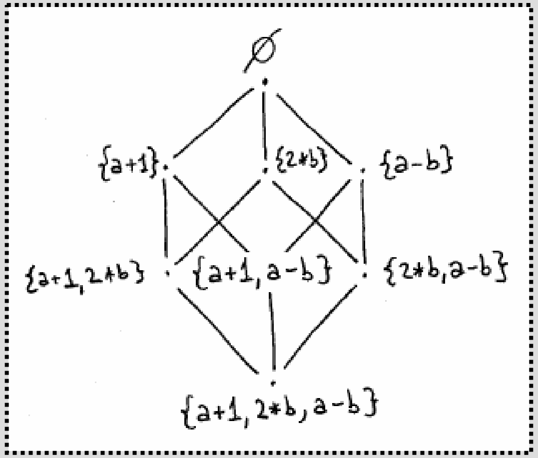


E.g.:
- Uninitialized Variables

- Confluence:
  - ⊔ = ' ∪ ' *(set union)*
- Partial order:
  - ⊑ = ' ⊆ ' *(sub-set-eq)*

Analyze info that ***must definitely*** be true ∀paths

- Some analyses…:



E.g.:
- Initialized Variables
- Available Expressions
- Very Busy Expressions

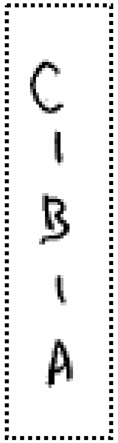
- Confluence:
  - ⊔ = ' ∩ ' *(set intersection)*
- Partial order:
  - ⊑ = ' ⊇ ' *(super-set-eq)*

⊢ DUALITY: Lattice

$(M,\subseteq)$ ⇔ $(W,\supseteq)$

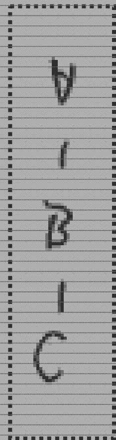
Lattice ***Lattice***

Lattice ***M***:



- Confluence:
  - $\sqcup_M$
- Partial order:
  - $\sqsubseteq_M$

"Stand on head": ***W***



- Confluence:
  - $\sqcup_W = \sqcap_M$
- Partial order:
  - $\sqsubseteq_W = \sqsupseteq_M$

DUALITY: Framework

What we have done:

- Approximation:
  - ”safe” to throw away info upwards
- With confluence:
  - $\sqcup$ (least upper bound)
- Least fixed point:
  - $\mathtt{lfp}(f) = \bigsqcup_{i \geq 0} f^i(\bot)$
- Computed as:
  - $\bot \sqsubseteq f(\bot) \sqsubseteq f(f(\bot)) \sqsubseteq \ldots$

”Stand on head”:

- Approximation:
  - ”safe” to throw away info downwards
- With confluence:
  - $\sqcap$ (greatest lower bound)
- Greatest fixed point:
  - $\mathtt{gfp}(f) = \bigsqcap_{i \geq 0} f^i(\top)$
- Computed as:
  - $\top \sqsupseteq f(\top) \sqsupseteq f(f(\top)) \sqsupseteq \ldots$

⊢ Agenda

- Quick recap (of everything so far):
- *"Putting it all together"* → ***Data-Flow Analysis***
- Fixed-Point Iteration Strategies (3x)
- "Sign Analysis"
- "Constant Propagation Analysis"
- "Initialized Variables Analysis"
- Set-Based Analysis Framework
- WORKSHOP

3 Example Data-Flow Analyses

⊢ WORKSHOP

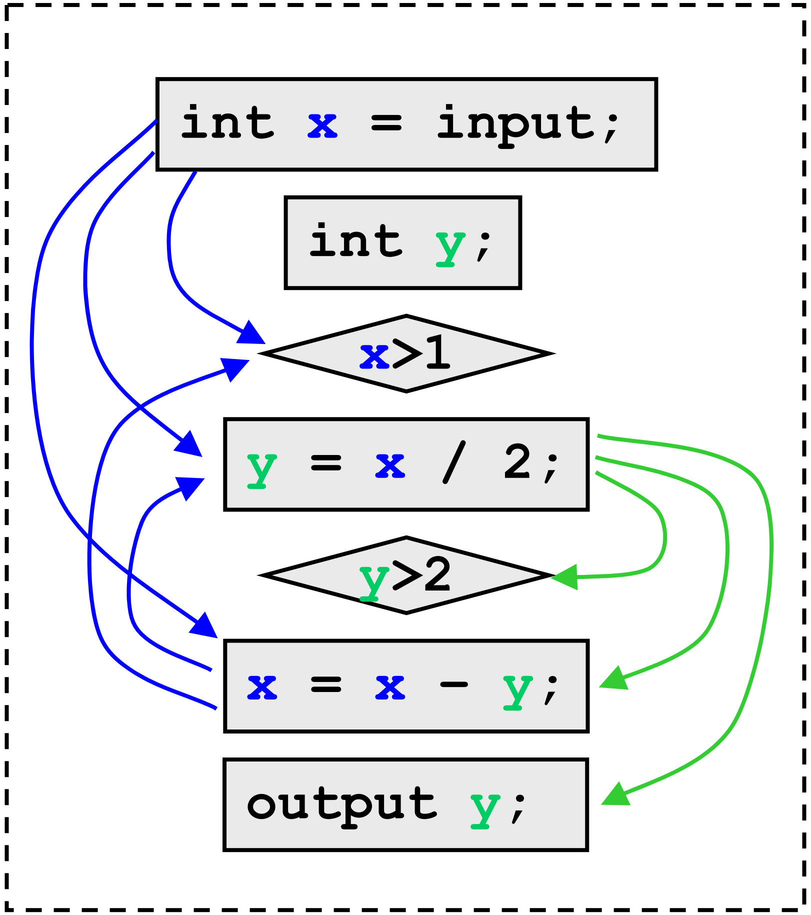
- ***Reaching Definitions:***
  - '↓' (*forward*), '∪' (*may* / *smallest set*)
- ***Live Variables:***
  - '↑' (*backward*), '∪' (*may* / *smallest set*)
- ***Available Expressions:***
  - '↓' (*forward*), '∩' (*must* / *largest set*)
- ***Very Busy Expressions:***
  - '↑' (*backward*), '∩' (*must* / *largest set*)

Reaching Definitions

- Reaching Definitions:
  - The ***reaching definitions*** (for a given program point) are those ***assignments*** that ***may have defined*** the current ***vals*** of ***vars***

- Example:
  - ```
    int x = input;
    int y;
    while (x>1) {
      y = x / 2;
      if (y>2) x = x - y;
    }
    output y;
    ```

DEF-USE graph:


    ```

WHY do we do this?

> *"Learning takes place through the* ***active behavior*** *of the student: it is what (s)he does that (s)he learns, not what the teacher does."*
>
> -- Ralph W. Tyler (1949)

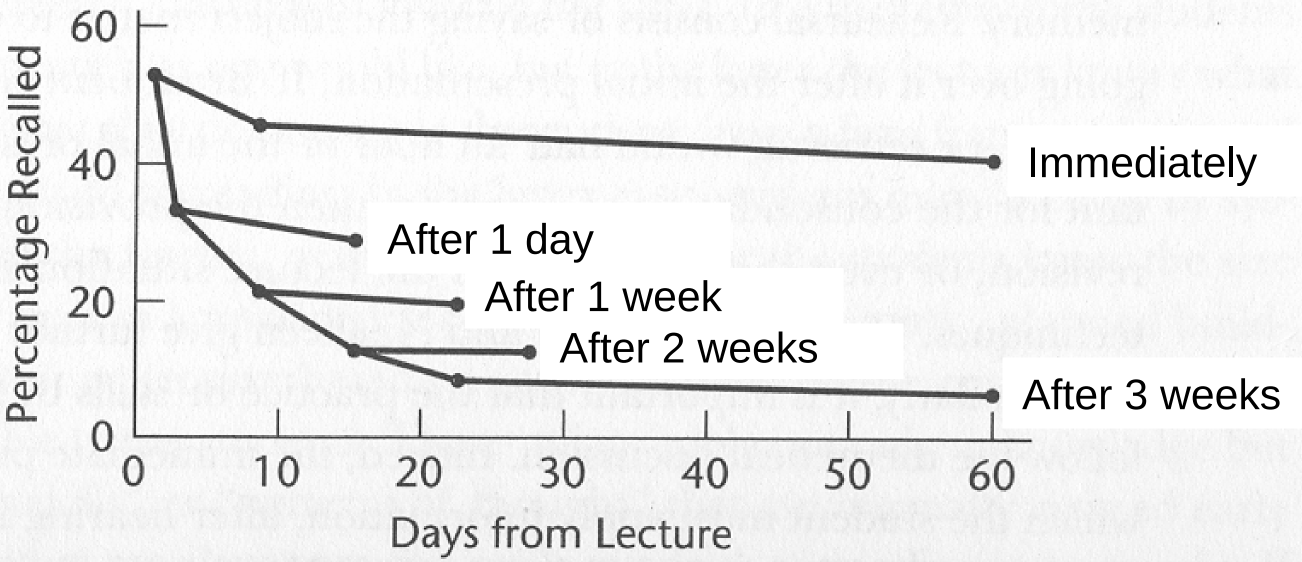
⊢ WORKSHOP

- **1)** Define the ***problem***
- ~~**2)** Show that the problem is ***undecidable***~~
- **3)** Define a ***Lattice***
  - Check that it is a lattice (and explain how)
- **4)** Define ***monotone transfer functions***
  - Check that they are monotone (and explain how)
- **5)** Pick a program the analysis ***can analyze***
  - Make a "The Entire Process" diagram (cf. slide #5)
- **6)** Repeat **5)** for program the analysis ***can't***…
- **7)** ***Explain*** possible uses of the analysis

Now, please: 3' recap

- Please spend 3' on thinking about and writing down the main ideas and points from the lecture – ***now!:***

FIGURE 2.5. THE VALUE OF REHEARSAL FOLLOWING A LECTURE.



Source: Adapted from Bassey (1968).