

# 15-819: Program Analysis

---

## Dataflow Analysis

Jonathan Aldrich

# Outline

---

- **Introduction to Dataflow Analysis**
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm
- Example Dataflow Analyses
  - Constant Propagation
  - Reaching Definitions
  - Live Variable Analysis

# Motivation: Dataflow Analysis

---

- Catch interesting errors
  - Non-local:  $x$  is null,  $x$  is written to  $y$ ,  $y$  is dereferenced
- Optimize code
  - Reduce run time, memory usage...
- Soundness required
  - Safety-critical domain
    - Assure lack of certain errors
  - Cannot optimize unless it is proven safe
    - Correctness comes before performance
- Automation required
  - Dramatically decreases cost
  - Makes cost/benefit worthwhile for far more purposes

# Dataflow analysis

---

- Tracks value flow through program
- Can distinguish order of operations
  - Did you read the file after you closed it?
  - Does this null value flow to that dereference?
- Differs from AST walker
  - Walker simply collects information or checks patterns
  - Tracking flow allows more interesting properties
- Abstracts values
- Chooses abstraction particular to property
  - Is a variable null?
  - Is a file open or closed?
  - Could a variable be 0?
  - Where did this value come from?

# Zero Analysis

---

- Could variable  $x$  be 0?
  - Useful to know if you have an expression  $y/x$
  - In C, useful for null pointer analysis
- Program semantics
  - Stack environment  $\eta$  maps every variable to an integer
- Semantic abstraction
  - $\sigma$  maps every variable to non zero (NZ), zero(Z), or maybe zero (MZ)
  - Abstraction function for integers  $\alpha_{ZI}$ :
    - $\alpha_{ZI}(0) = Z$
    - $\alpha_{ZI}(n) = NZ$  for all  $n \neq 0$
  - We may not know if a value is zero or not
    - Analysis is always an approximation
    - Need MZ option, too

# Zero Analysis Example

---

```
x := 10;  
y := x;  
z := 0;  
while y > -1 do  
  x := x / y;  
  y := y-1;  
  z := 5;
```

$\sigma = []$

# Zero Analysis Example

---

```
x := 10;  
y := x;  
z := 0;  
while y > -1 do  
  x := x / y;  
  y := y-1;  
  z := 5;
```

$\sigma = []$

$\sigma = [x \mapsto \alpha_{z_1}(10)]$

# Zero Analysis Example

---

```
x := 10;  
y := x;  
z := 0;  
while y > -1 do  
  x := x / y;  
  y := y-1;  
  z := 5;
```

$\sigma = []$

$\sigma = [x \mapsto \text{NZ}]$

# Zero Analysis Example

---

```
x := 10;  
y := x;  
z := 0;  
while y > -1 do  
  x := x / y;  
  y := y-1;  
  z := 5;
```

$$\sigma = []$$
$$\sigma = [x \mapsto \text{NZ}]$$
$$\sigma = [x \mapsto \text{NZ}, y \mapsto \sigma(x)]$$

# Zero Analysis Example

---

```
x := 10;
y := x;
z := 0;
while y > -1 do
  x := x / y;
  y := y-1;
  z := 5;
```

$\sigma = []$

$\sigma = [x \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}]$

# Zero Analysis Example

---

$x := 10;$

$y := x;$

$z := 0;$

while  $y > -1$  do

$x := x / y;$

$y := y - 1;$

$z := 5;$

$\sigma = []$

$\sigma = [x \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \alpha_{z|}(0)]$

# Zero Analysis Example

---

```
x := 10;  
y := x;  
z := 0;  
while y > -1 do  
  x := x / y;  
  y := y-1;  
  z := 5;  
  
σ = []  
σ = [x ↦ NZ]  
σ = [x ↦ NZ, y ↦ NZ]  
σ = [x ↦ NZ, y ↦ NZ, z ↦ Z]
```

# Zero Analysis Example

---

```
x := 10;
y := x;
z := 0;
while y > -1 do
  x := x / y;
  y := y-1;
  z := 5;

```

$\sigma = []$

$\sigma = [x \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

# Zero Analysis Example

---

```
x := 10;
y := x;
z := 0;
while y > -1 do
  x := x / y;
  y := y-1;
  z := 5;

```

$\sigma = []$

$\sigma = [x \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

# Zero Analysis Example

---

```
x := 10;
y := x;
z := 0;
while y > -1 do
  x := x / y;
  y := y-1;
  z := 5;

```

$\sigma = []$

$\sigma = [x \mapsto NZ]$

$\sigma = [x \mapsto NZ, y \mapsto NZ]$

$\sigma = [x \mapsto NZ, y \mapsto NZ, z \mapsto Z]$

$\sigma = [x \mapsto NZ, y \mapsto NZ, z \mapsto Z]$

$\sigma = [x \mapsto MZ, y \mapsto NZ, z \mapsto Z]$

$\sigma = [x \mapsto MZ, y \mapsto MZ, z \mapsto Z]$

# Zero Analysis Example

---

```
x := 10;
y := x;
z := 0;
while y > -1 do
  x := x / y;
  y := y-1;
  z := 5;

```

$\sigma = []$

$\sigma = [x \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{NZ}]$

# Zero Analysis Example

---

```
x := 10;  
y := x;  
z := 0;  
while y > -1 do  
  x := x / y;  
  y := y-1;  
  z := 5;
```

```
 $\sigma = []$   
 $\sigma = [x \mapsto NZ]$   
 $\sigma = [x \mapsto NZ, y \mapsto NZ]$   
 $\sigma = [x \mapsto NZ, y \mapsto NZ, z \mapsto Z]$   
 $\sigma = [x \mapsto MZ, y \mapsto MZ, z \mapsto MZ]$   
 $\sigma = [x \mapsto MZ, y \mapsto NZ, z \mapsto Z]$   
 $\sigma = [x \mapsto MZ, y \mapsto MZ, z \mapsto Z]$   
 $\sigma = [x \mapsto MZ, y \mapsto MZ, z \mapsto NZ]$ 
```

# Zero Analysis Example

---

```
x := 10;
y := x;
z := 0;
while y > -1 do
  x := x / y;
  y := y-1;
  z := 5;

```

$\sigma = []$

$\sigma = [x \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{MZ}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{MZ}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{NZ}]$

# Zero Analysis Example

---

```
x := 10;
y := x;
z := 0;
while y > -1 do
  x := x / y;
  y := y-1;
  z := 5;

```

$\sigma = []$

$\sigma = [x \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}]$

$\sigma = [x \mapsto \text{NZ}, y \mapsto \text{NZ}, z \mapsto \text{Z}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{MZ}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{MZ}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{MZ}]$

$\sigma = [x \mapsto \text{MZ}, y \mapsto \text{MZ}, z \mapsto \text{NZ}]$

# Zero Analysis Example

---

```
x := 10;  
y := x;  
z := 0;  
while y > -1 do  
  x := x / y;  
  y := y-1;  
  z := 5;
```

```
 $\sigma = []$   
 $\sigma = [x \mapsto NZ]$   
 $\sigma = [x \mapsto NZ, y \mapsto NZ]$   
 $\sigma = [x \mapsto NZ, y \mapsto NZ, z \mapsto Z]$   
 $\sigma = [x \mapsto MZ, y \mapsto MZ, z \mapsto MZ]$   
 $\sigma = [x \mapsto MZ, y \mapsto MZ, z \mapsto MZ]$   
 $\sigma = [x \mapsto MZ, y \mapsto MZ, z \mapsto MZ]$   
 $\sigma = [x \mapsto MZ, y \mapsto MZ, z \mapsto NZ]$ 
```

**Nothing more happens!**

# Zero Analysis Termination

---

- The analysis values will not change, no matter how many times we execute the loop
  - Proof: our analysis is deterministic
  - We run through the loop with the current analysis values, none of them change
  - Therefore, no matter how many times we run the loop, the results will remain the same
  - Therefore, we have computed the dataflow analysis results for any number of loop iterations

# Zero Analysis Termination

---

- The analysis values will not change, no matter how many times we execute the loop
  - Proof: our analysis is deterministic
  - We run through the loop with the current analysis values, none of them change
  - Therefore, no matter how many times we run the loop, the results will remain the same
  - Therefore, we have computed the dataflow analysis results for any number of loop iterations
- Why does this work
  - If we simulate the loop, the data values could (in principle) keep changing indefinitely
    - There are an infinite number of data values possible
    - Not true for 32-bit integers, but might as well be true
      - Counting to  $2^{32}$  is slow, even on today's processors
  - Dataflow analysis only tracks 2 possibilities!
    - So once we've explored them all, nothing more will change
    - This is the secret of abstraction
- We will make this argument more precise later

# Using Zero Analysis

---

- Visit each division in the program
- Get the results of zero analysis for the divisor
- If the results are definitely zero, report an error
- If the results are possibly zero, report a warning

# Outline

---

- Introduction to Dataflow Analysis
- **Dataflow Analysis Frameworks**
  - **Lattices**
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm
- Example Dataflow Analyses
  - Constant Propagation
  - Reaching Definitions
  - Live Variable Analysis

# Defining Dataflow Analyses

---

- **Lattice**
  - Describes program data abstractly
  - Abstract equivalent of stack or heap contents
- **Abstraction function**
  - Maps concrete value to lattice element
- **Flow functions**
  - Describes how abstract data changes
  - Abstract equivalent of statement/expression semantics
- **Control flow graph**
  - Determines how abstract data propagates from statement to statement
  - Abstract equivalent of program semantics

# Definitions

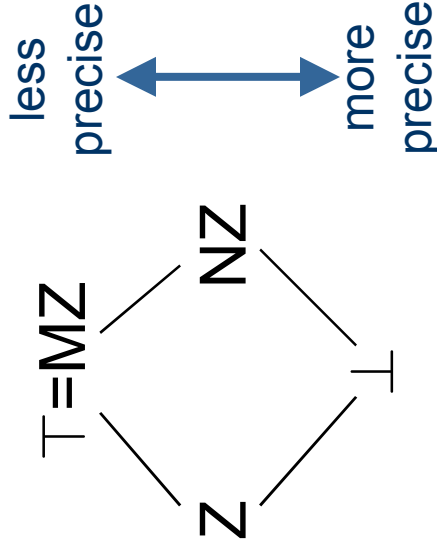
---

- Partial order – a relation  $\sqsubseteq$  that is:
  - Reflexive ( $l \sqsubseteq l$ )
  - Transitive ( $l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_3 \Rightarrow l_1 \sqsubseteq l_3$ )
  - Antisymmetric ( $l_1 \sqsubseteq l_2 \wedge l_2 \sqsubseteq l_1 \Rightarrow l_1 = l_2$ )
- Upper bound  $l$  of a set  $L$ 
  - $\forall l' \in L : l' \sqsubseteq l$
- Least upper bound / join operator  $\sqcup$ 
  - An upper bound  $l \sqsubseteq$  all other upper bounds
- Lattice – a partially ordered set where every subset has a least upper bound

# Lattices in Program Analysis

---

- A lattice is a tuple  $(L, \sqsubseteq, \sqcup, \perp, \top)$ 
  - $L$  is a set of abstract elements
  - $\sqsubseteq$  is a partial order on  $L$ 
    - Means *at least as precise as*
  - $\sqcup$  is the least upper bound of two elements
    - Must exist for every two elements in  $L$
    - Used to merge two abstract values
  - $\perp$  (bottom) is the least element of  $L$ 
    - Means we haven't yet analyzed this yet
    - Will become clear later
  - $\top$  (top) is the greatest element of  $L$ 
    - Means we don't know anything

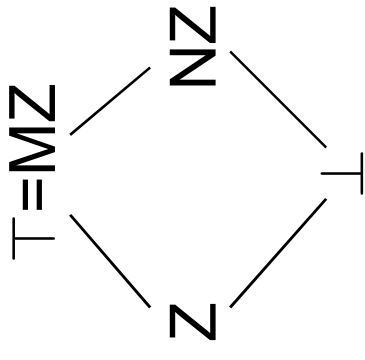


- $L$  may be infinite
  - Typically should have finite height
    - All paths from  $\perp$  to  $\top$  should be finite
    - We'll see why later

# Zero Analysis Lattice

---

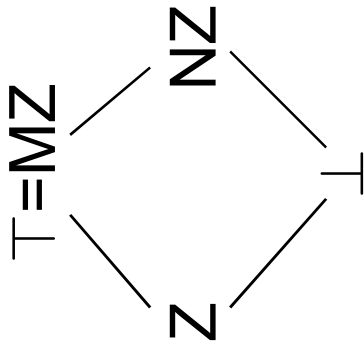
- Integer zero lattice
  - $L_{ZI} = \{\perp, Z, NZ, MZ\}$
  - $\perp \sqsubseteq Z, \perp \sqsubseteq NZ, NZ \sqsubseteq MZ, Z \sqsubseteq MZ$ 
    - $\perp \sqsubseteq MZ$  holds by transitivity
  - $\sqcup$  defined as join for  $\sqsubseteq$ 
    - $x \sqcup y = z$  iff
      - $z$  is an upper bound of  $x$  and  $y$
      - $z$  is the least such bound
  - Obeys laws:  $\perp \sqcup \mathcal{X} = \mathcal{X}, \top \sqcup \mathcal{X} = \top, \mathcal{X} \sqcup \mathcal{X} = \mathcal{X}$
  - Also  $Z \sqcup NZ = MZ$
- $\perp = \perp$
- $\forall \mathcal{X}. \perp \sqsubseteq \mathcal{X}$
- $\top = MZ$
- $\forall \mathcal{X}. \mathcal{X} \sqsubseteq \top$



# Zero Analysis Lattice

---

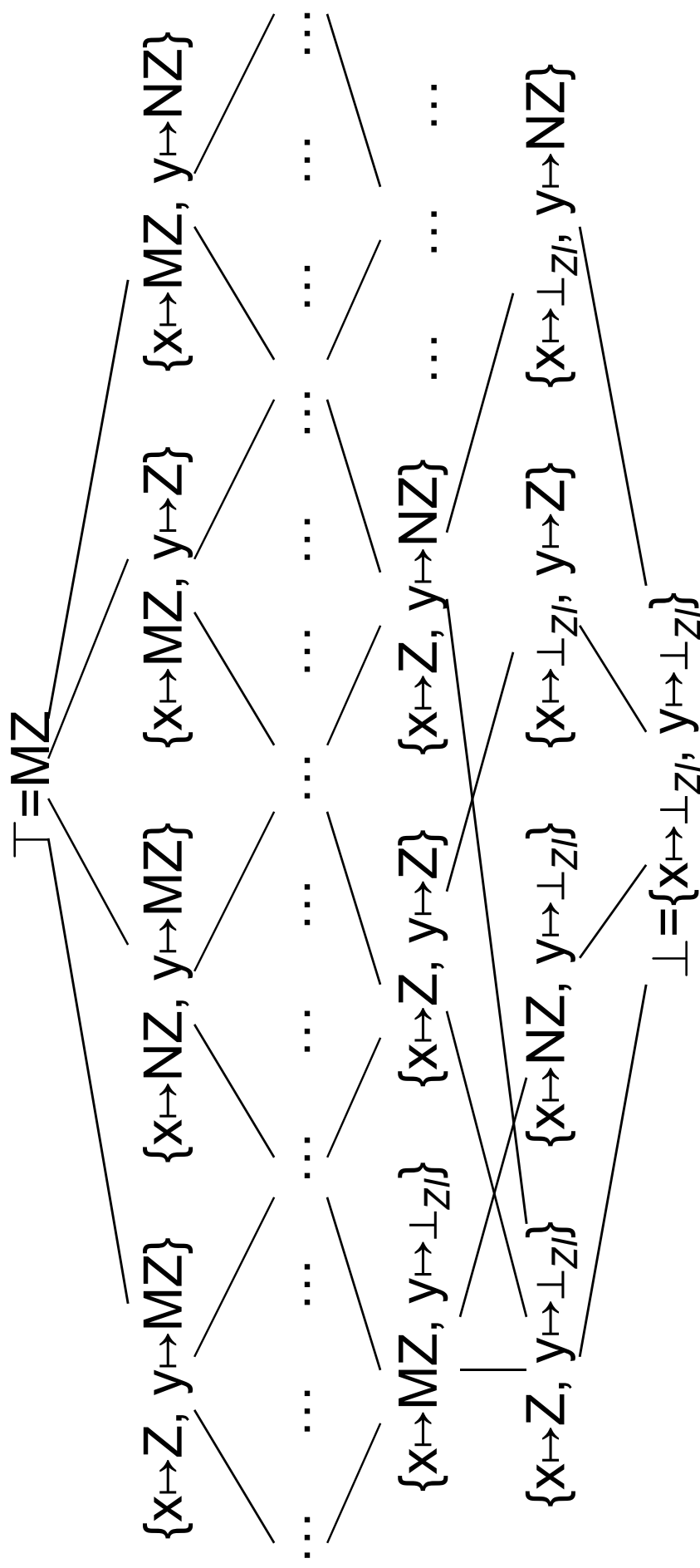
- Integer zero lattice
  - $L_{ZI} = \{\perp, Z, NZ, MZ\}$
  - $\perp \sqsubseteq Z, \perp \sqsubseteq NZ, NZ \sqsubseteq MZ, Z \sqsubseteq MZ$
  - $\sqcup$  defined as join for  $\sqsubseteq$
  - $\perp = \perp$
  - $T = MZ$
- Program lattice is a *tuple lattice*
  - $L_Z$  is the set of all maps from **Var** to  $L_{ZI}$
  - $\sigma_1 \sqsubseteq_Z \sigma_2$  iff  $\forall x \in \mathbf{Var} . \sigma_1(x) \sqsubseteq_{ZI} \sigma_2(x)$
  - $\sigma_1 \sqcup_Z \sigma_2 = \{x \mapsto \sigma_1(x) \sqcup_{ZI} \sigma_2(x) \mid x \in \mathbf{Var}\}$
  - $\perp = \{x \mapsto \perp_{ZI} \mid x \in \mathbf{Var}\}$
  - $T = \{x \mapsto T_{ZI} \mid x \in \mathbf{Var}\} = \{x \mapsto MZ \mid x \in \mathbf{Var}\}$
  - Can produce a tuple lattice from *any* base lattice
    - Just define as above



# Tuple Lattices Visually

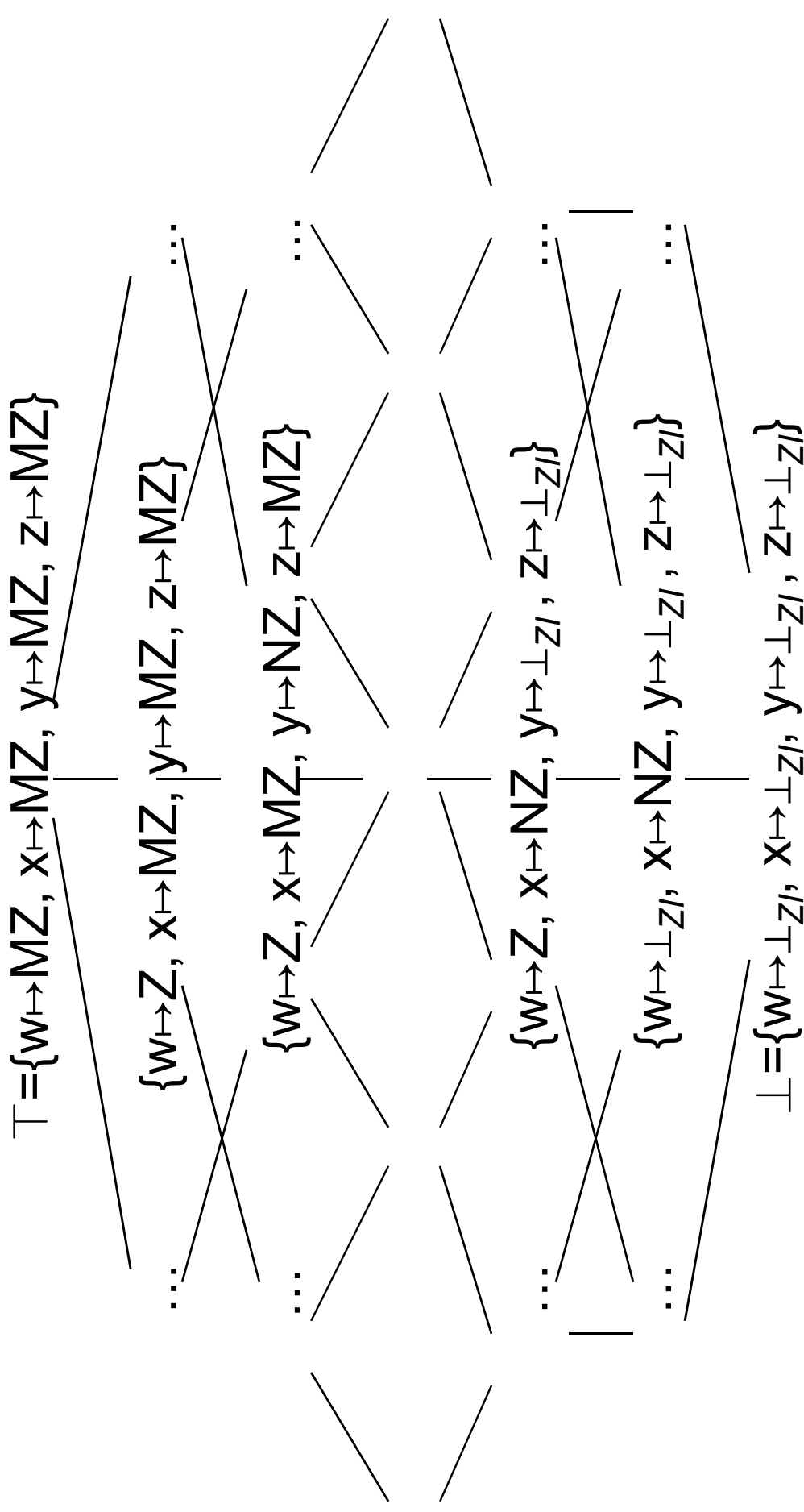
---

- For  $\text{Var} = \{x, y\}$



# One Path in a Tuple Lattice

---



## Quick Quiz

---

- Consider the following two tuple lattice values:  $[x \mapsto Z, y \mapsto MZ]$  and  $[x \mapsto MZ, y \mapsto NZ]$ 
  - How do the two compare in the lattice ordering for zero analysis?
- What is the join of these two tuple lattice values?

# Outline

---

- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm
- Example Dataflow Analyses
  - Constant Propagation
  - Reaching Definitions
  - Live Variable Analysis

# Abstraction Function

---

- Maps each concrete program state to a lattice element
  - For tuple lattices, the function can be defined for values and lifted to tuples
- Integer Zero abstraction function  $\alpha_{Z_I}$ :
  - $\alpha_{Z_I}(0) = Z$
  - $\alpha_{Z_I}(n) = NZ$  for all  $n \neq 0$
- Zero Analysis abstraction function  $\alpha_{Z_A}$ :
  - $\alpha_{Z_A}(\eta) = \{x \mapsto \alpha_{Z_I}(\eta(x)) \mid x \in \mathbf{Var}\}$
  - This is just the tuple form of  $\alpha_{Z_I}(n)$ 
    - Can be done for any tuple lattice

# Outline

---

- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - **Control flow graphs**
  - Flow functions
  - Worklist algorithm
- Example Dataflow Analyses
  - Constant Propagation
  - Reaching Definitions
  - Live Variable Analysis

# Control Flow Graph (CFG)

---

- Shows order of statement execution
  - Determines where data flows
- Decomposes expressions into primitive operations
  - Typically one CFG node per “useful” AST node
    - constants, variables, binary operations, assignments, if, while...
  - Loops are written out
    - Form a loop in the CFG
  - Benefit: analysis is defined one operation at a time

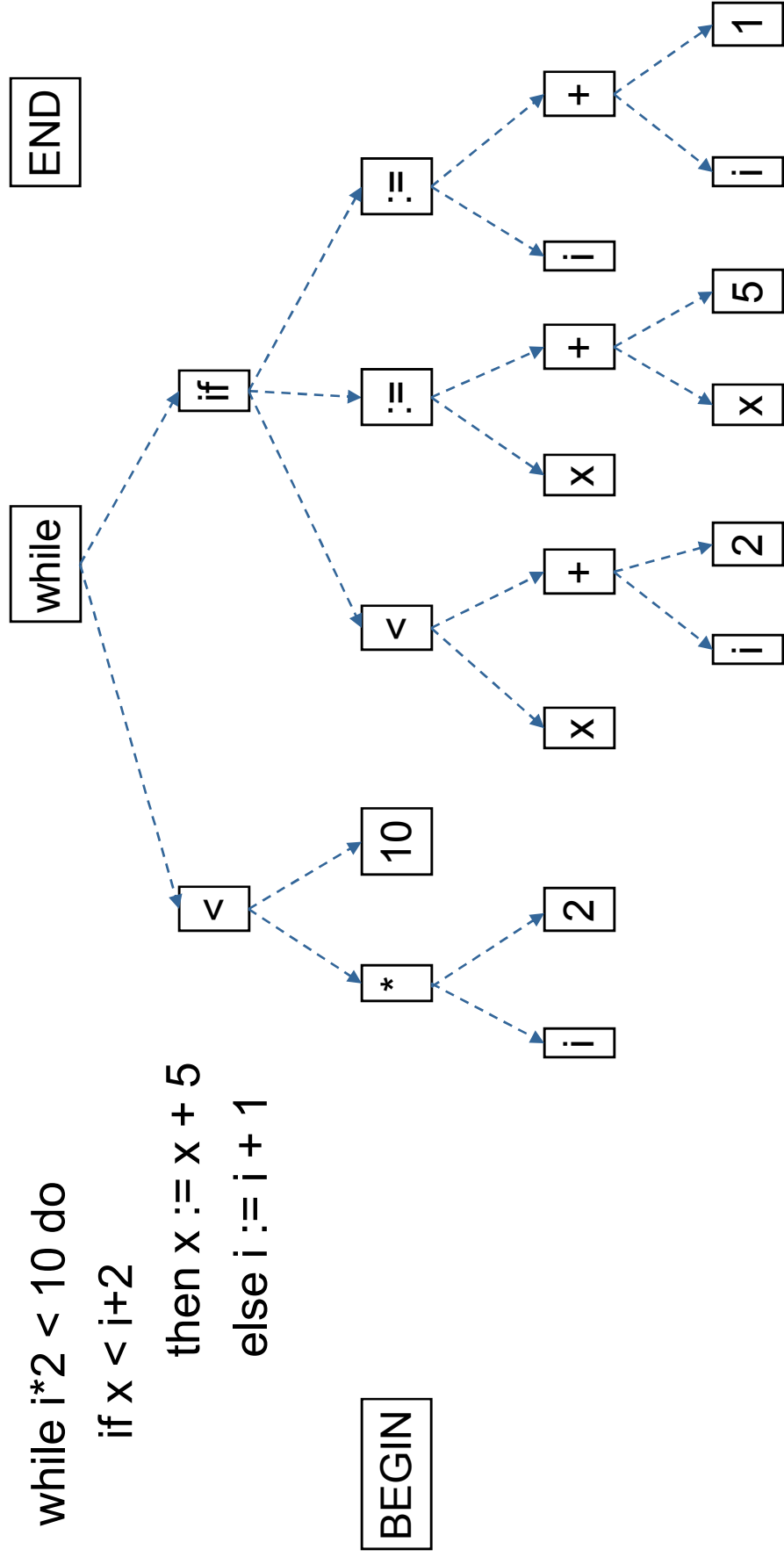
# Intuition for Building a CFG

---

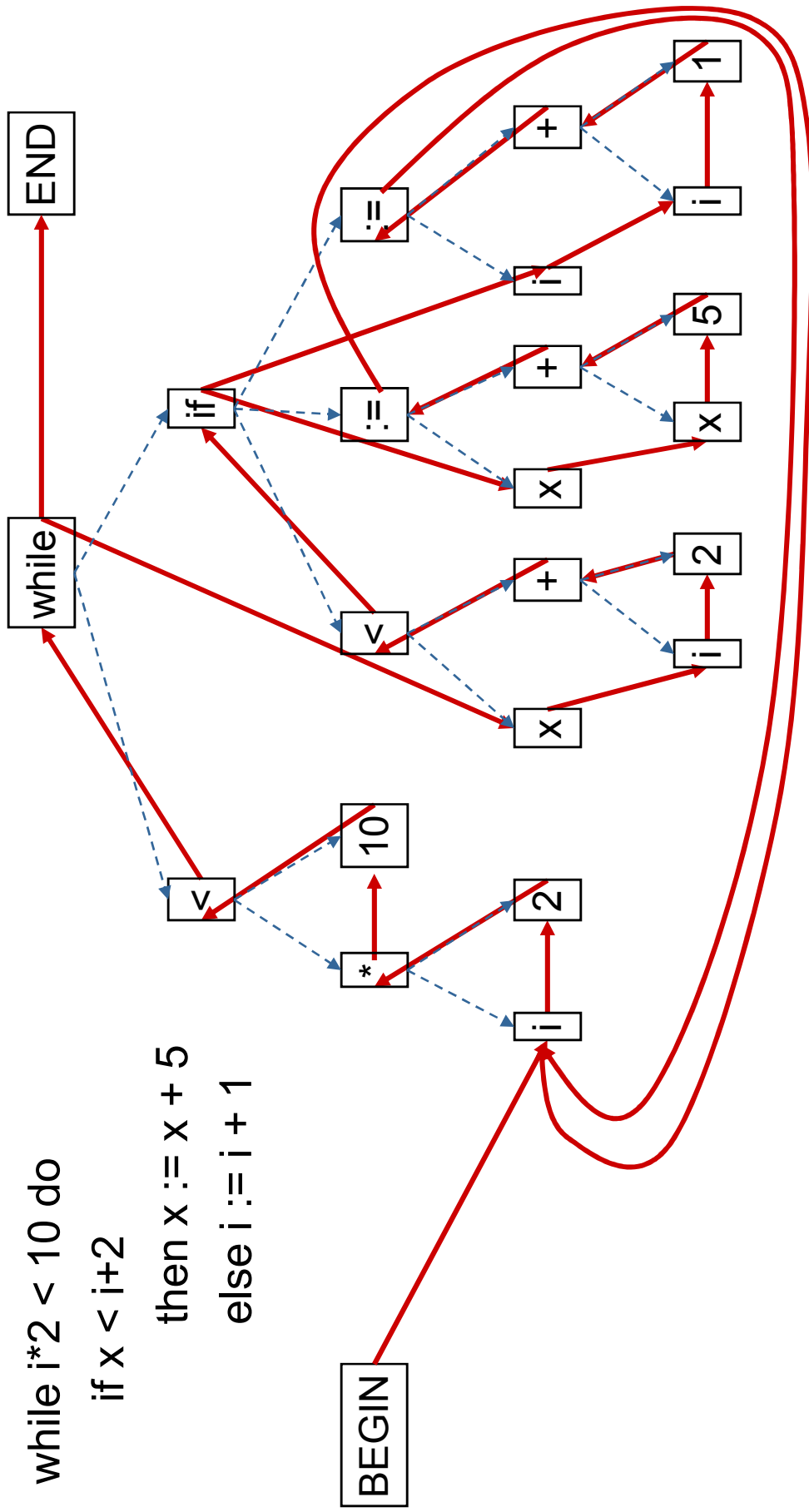
- Connect nodes in order of operation
  - Defined by language
- Java order of operation
  - Expressions, assignment, sequence
    - Evaluate subexpressions left to right
    - Evaluate node after children (postfix)
  - While, If
    - Evaluate condition first, then if/while
    - if branches to else and then
    - while branches to loop body and exit

# Control Flow Graph Example

```
while i*2 < 10 do  
  if x < i+2  
    then x := x + 5  
    else i := i + 1
```



# Control Flow Graph Example



# Quick Quiz

---

- Draw a CFG for the following program:

1:  $x := 0$

2:  $y := 1$

3: if ( $z == 0$ )

4:      $x := x + y$

5: else  $y := y - 1$

6:  $w := y$

# Outline

---

- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - **Flow functions**
  - Worklist algorithm
- Example Dataflow Analyses
  - Constant Propagation
  - Reaching Definitions
  - Live Variable Analysis

# Flow Functions

---

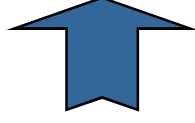
- Compute dataflow information after a statement from dataflow information before the statement
  - Formally, map a lattice element and a CFG node to a new lattice element
- $f_{\text{analysis}}(\sigma, [\textit{operation}]) = \sigma'$ 
  - $f$  is just a flow function
  - $\textit{analysis}$  is the name of our analysis
  - $\sigma$  is the old lattice
  - $\textit{operation}$  is what we are transferring over
  - $\sigma'$  is the new lattice

# Three Address Code

---

- Analysis performed on 3-address code
- inspired by 3 addresses in assembly language:  
add x,y,z
- Convert complex expressions to 3-address code
- Each subexpression represented by a temporary variable
- $x+3*y \rightarrow t_1:=3; t_2:=t_1*y; t_3:=x+t_2$

```
if (x == foo()) {  
    y = z * x + 5 / w;  
}
```



```
t1 = foo();  
t2 = x == t1;  
if (t2) {  
    t3 = z * x;  
    t4 = 5 / w;  
    y = t3 + t4;  
}
```

# While3Addr

---

- copy  $x = y$
- binary op  $x = y \text{ op } z$  ( $\text{op} \in \{+, -, *, /, \dots\}$ )
- literal  $x = n$
- unary op  $x = \text{op } y$  ( $\text{op} \in \{-, !, ++, \dots\}$ )
- label *lab*
- jump *jump lab*
- branch *btrue x lab*

# Zero Analysis Flow Functions

---

- $f_{ZA}(\sigma, [x := y]) = [x \mapsto \sigma(y)] \sigma$
- $f_{ZA}(\sigma, [x := n]) =$  if  $n == 0$   
    then  $[x \mapsto Z] \sigma$   
    else  $[x \mapsto NZ] \sigma$
- $f_{ZA}(\sigma, [x := \dots]) = [x \mapsto MZ] \sigma$
  
- $f_{ZA}(\sigma, / * \text{any non-assignment} */) = \sigma$

# Zero Analysis Flow Functions

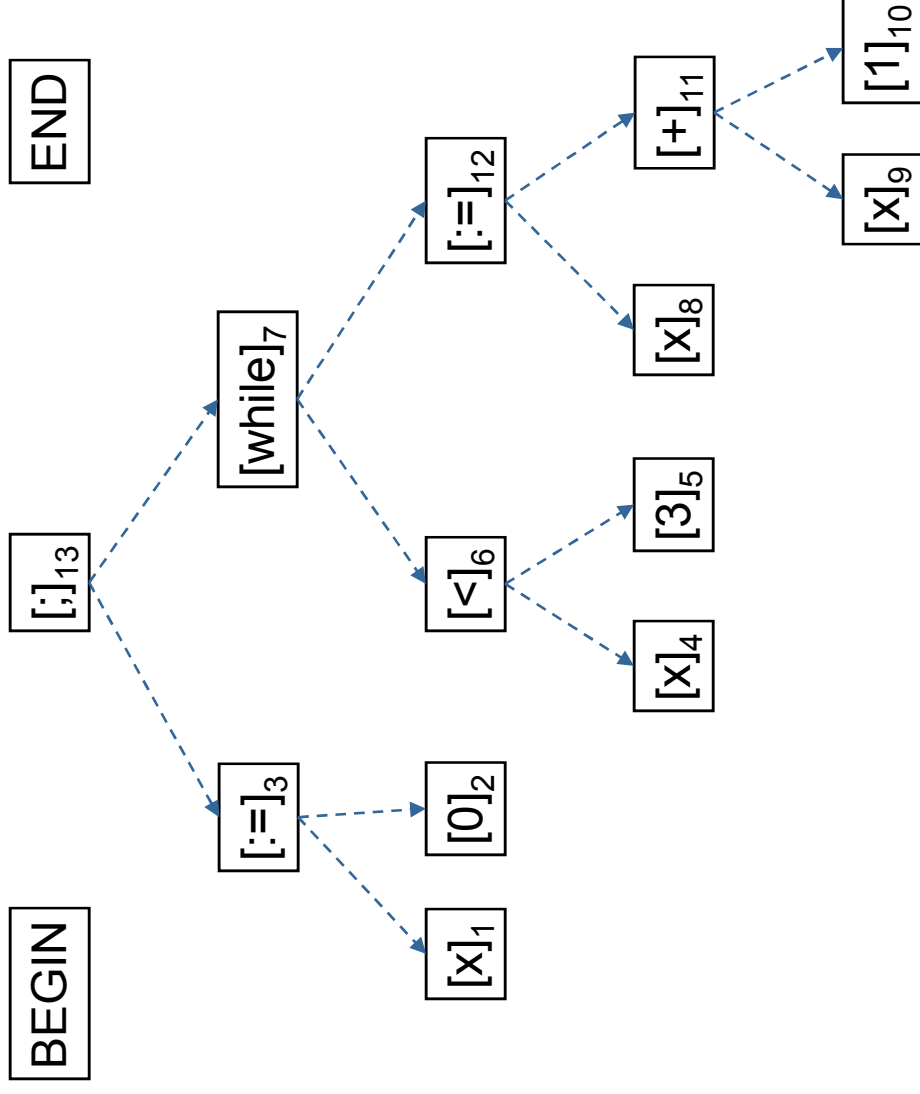
---

- $f_{ZA}(\sigma, [x := y]) = [x \mapsto \sigma(y)] \sigma$
- $f_{ZA}(\sigma, [x := n]) =$  if  $n == 0$   
    then  $[x \mapsto Z] \sigma$   
    else  $[x \mapsto NZ] \sigma$
- $f_{ZA}(\sigma, [x := \dots]) = [x \mapsto MZ] \sigma$ 
  - Could be more precise!  
 $f_{ZA}(\sigma, [x := a * b]) =$   
    if  $\sigma(a) = Z \parallel \sigma(b) = Z$   
    then  $[x \mapsto Z] \sigma$   
    else  $[x \mapsto MZ] \sigma$
- $f_{ZA}(\sigma, /* \text{any non-assignment} */) = \sigma$

# Zero Analysis Example

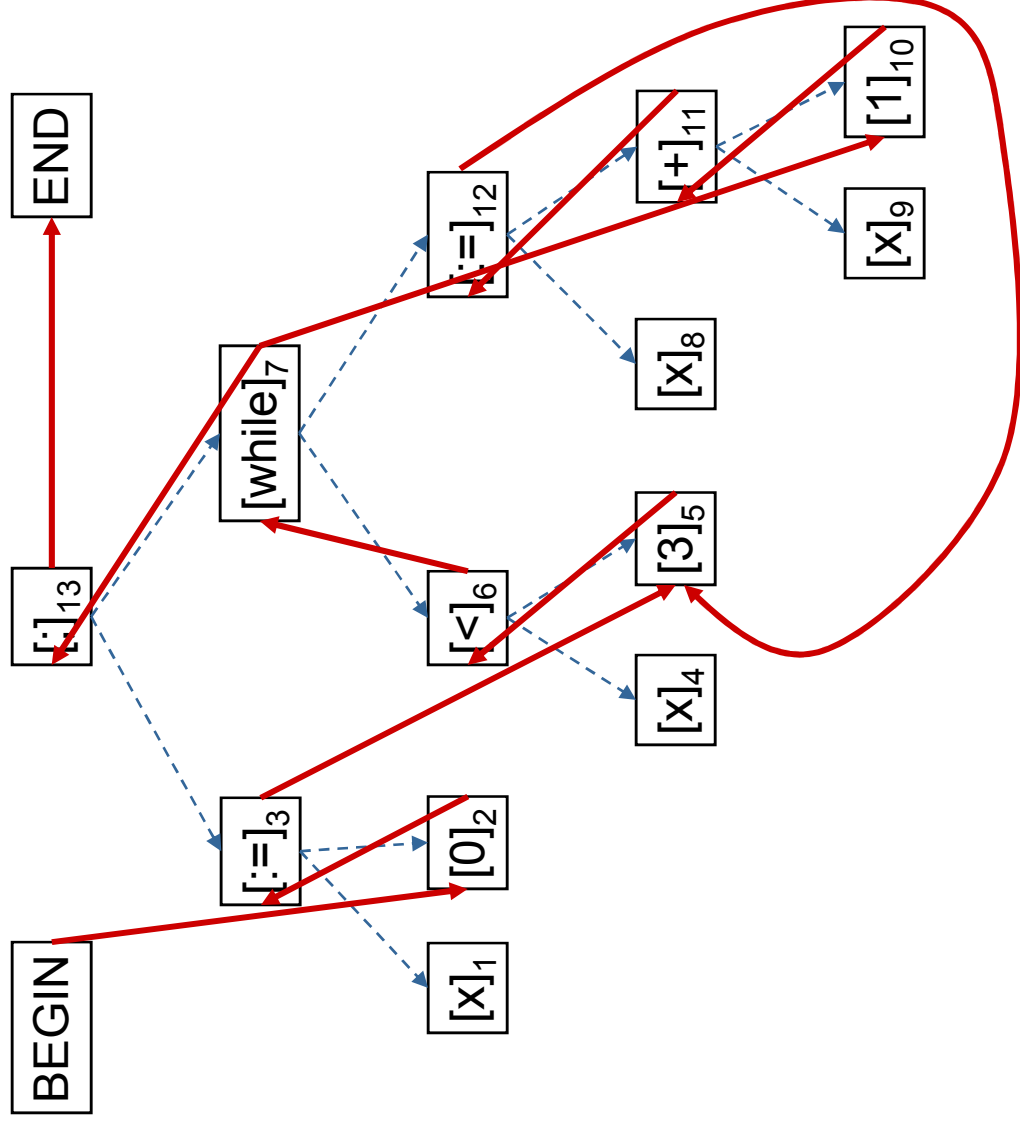
---

```
x := 0;  
while x > 3 do  
  x := x+1
```



# Zero Analysis Example

```
x := 0;  
while x > 3 do  
  x := x+1
```



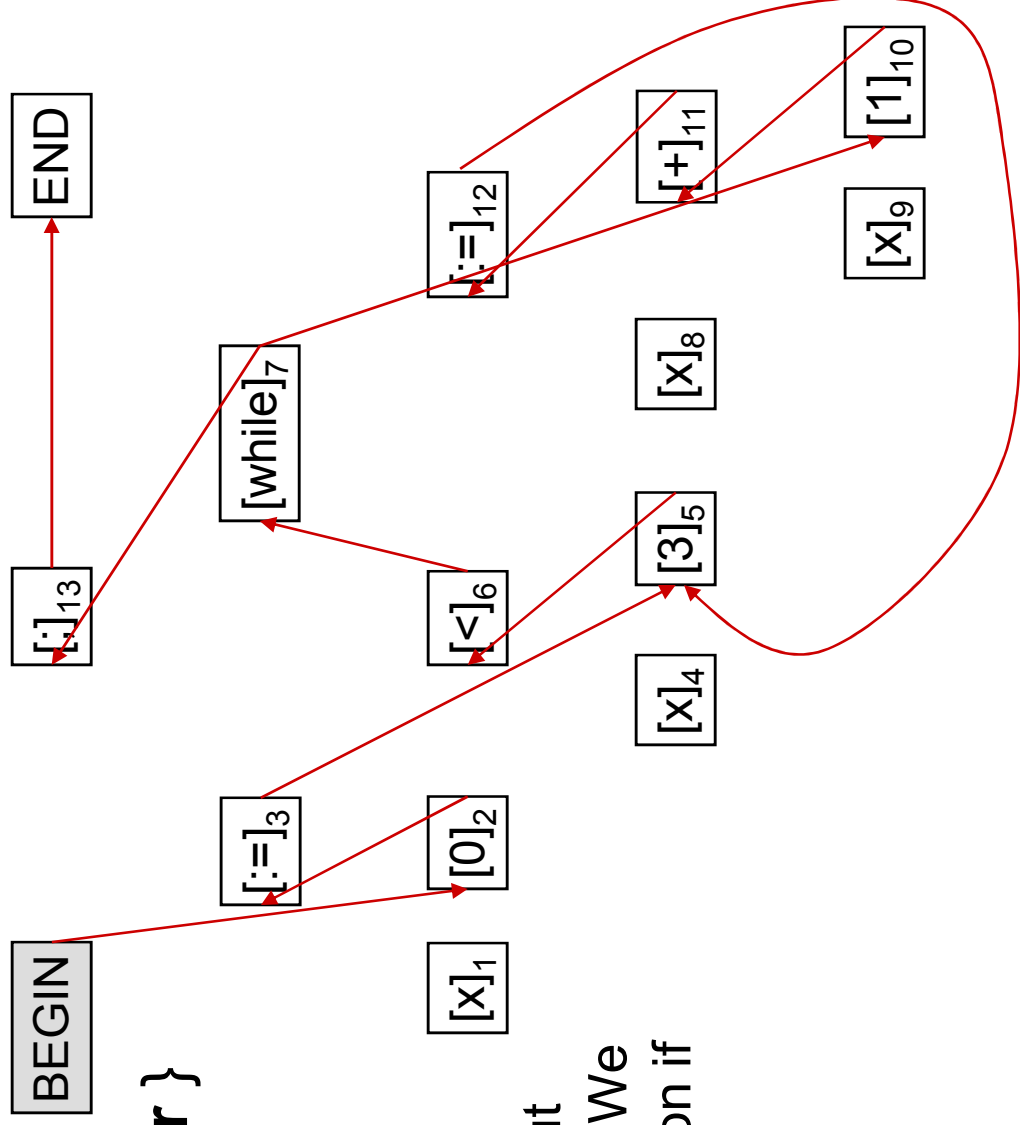
# Zero Analysis Example

Initial dataflow

$$\sigma_\iota = \{ x \mapsto MZ \mid x \in \mathbf{Var} \}$$

Intuition:

We know nothing about initial variable values. We could use a precondition if we had one.



# Zero Analysis Example

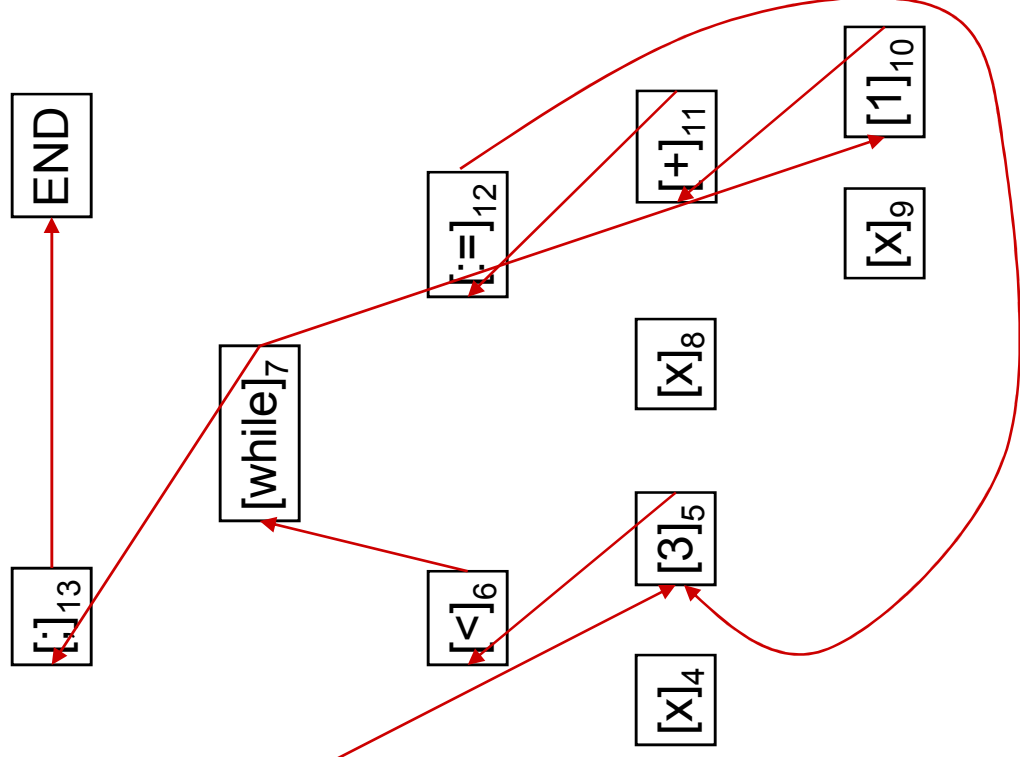
$\sigma_1 = \{ x \mapsto \text{MZ} \mid x \in \text{Var} \}$  BEGIN

$\sigma_2 = f_{ZA}(\sigma_1, [t_2 := 0])$   
 $= [t_2 \mapsto Z] \sigma_1$

$f_{ZA}(\sigma, [x := n]) =$   
 if  $n == 0$

then  $[x \mapsto Z] \sigma$

else  $[x \mapsto \text{NZ}] \sigma$



# Zero Analysis Example

$$\sigma_1 = \{x \mapsto MZ \mid x \in \mathbf{Var}\} \text{BEGIN}$$

$$\sigma_2 = [t_2 \mapsto Z] \sigma_1$$

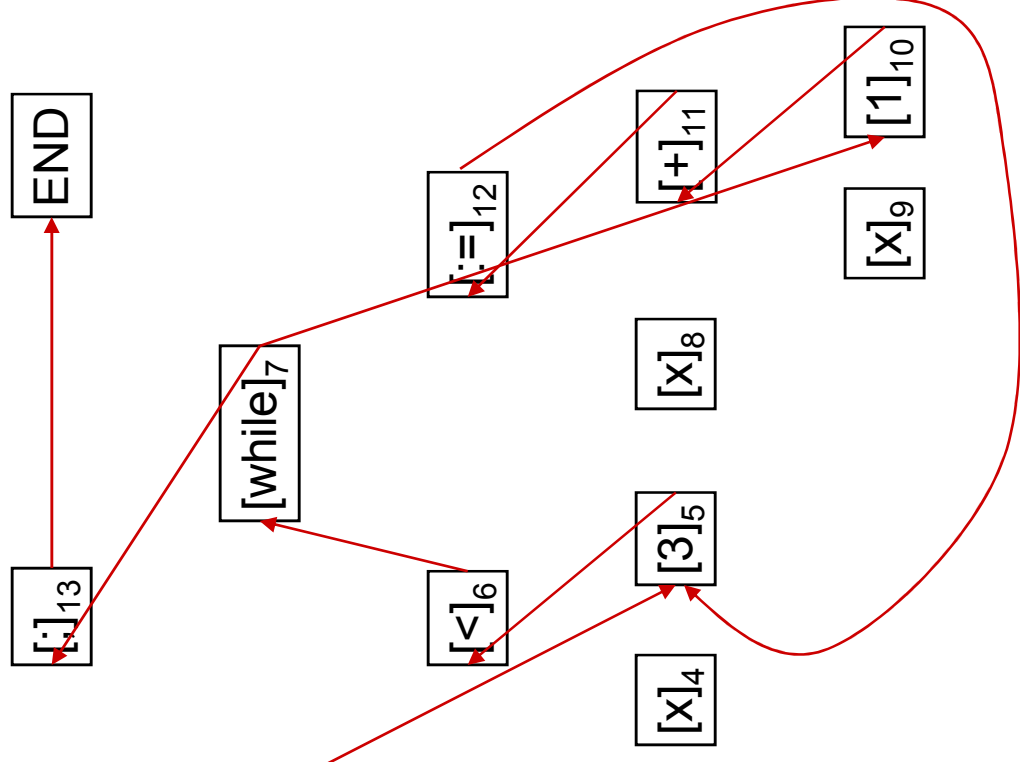
$$\sigma_3 = f_{ZA}(\sigma_2, [x := t_2])$$

$$= [x \mapsto \sigma_2(t_2)] \sigma_2$$

$$= [x \mapsto Z] \sigma_2$$

$$= [x \mapsto Z, t_2 \mapsto Z] \sigma_1$$

$$f_{ZA}(\sigma, [x := y]) = [x \mapsto \sigma(y)] \sigma$$



# Zero Analysis Example

$\sigma_{\perp} = \{x \mapsto MZ \mid x \in \text{Var}\}$  BEGIN

$\sigma_3 = [x \mapsto Z, t_2 \mapsto Z] \sigma_{\perp}$

Input to  $[3]_5$  comes from

$[:=]_3$  and  $[:=]_{12}$

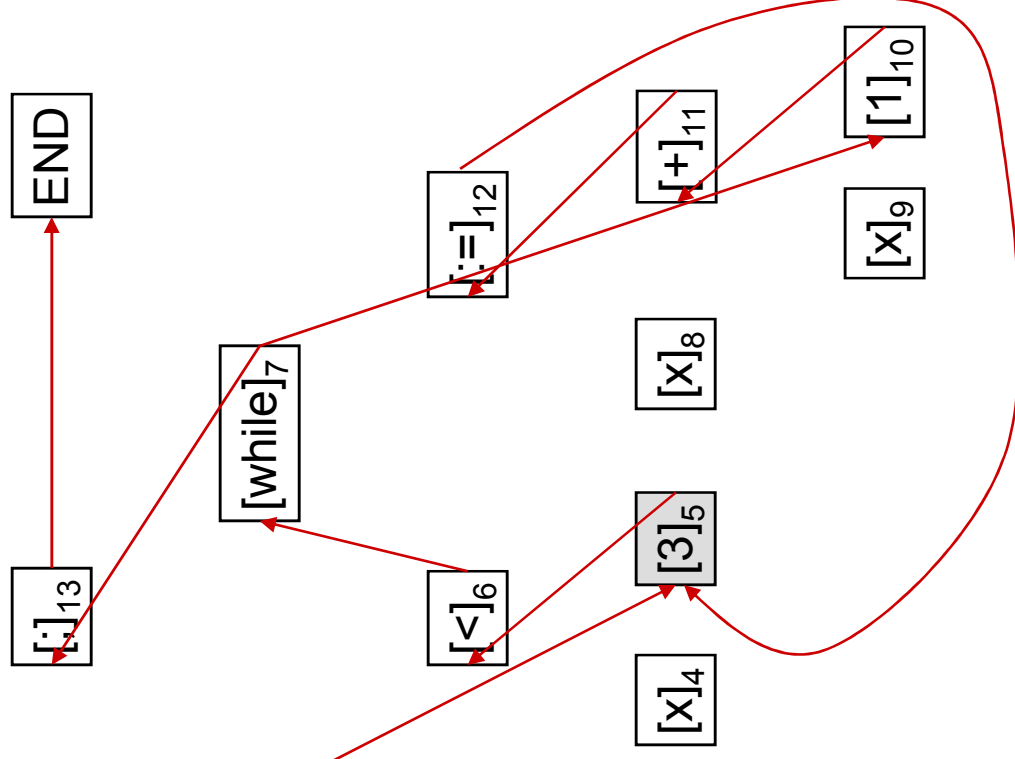
Input should be  $\sigma_3 \sqcup \sigma_{12}$

What is  $\sigma_{12}$ ?

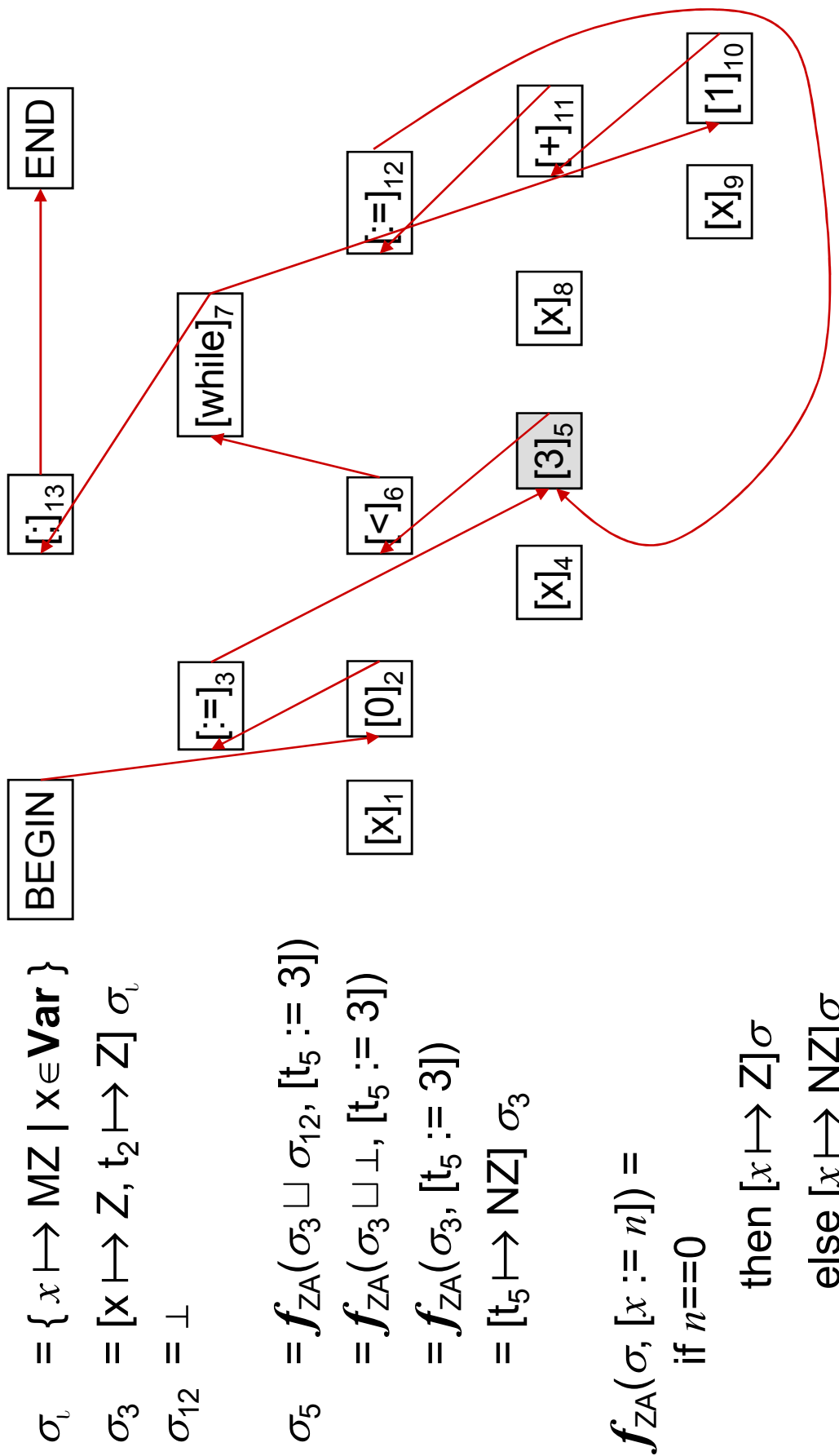
Solution: assume  $\perp$

Benefit:  $\sigma_3 \sqcup \perp = \sigma_3$

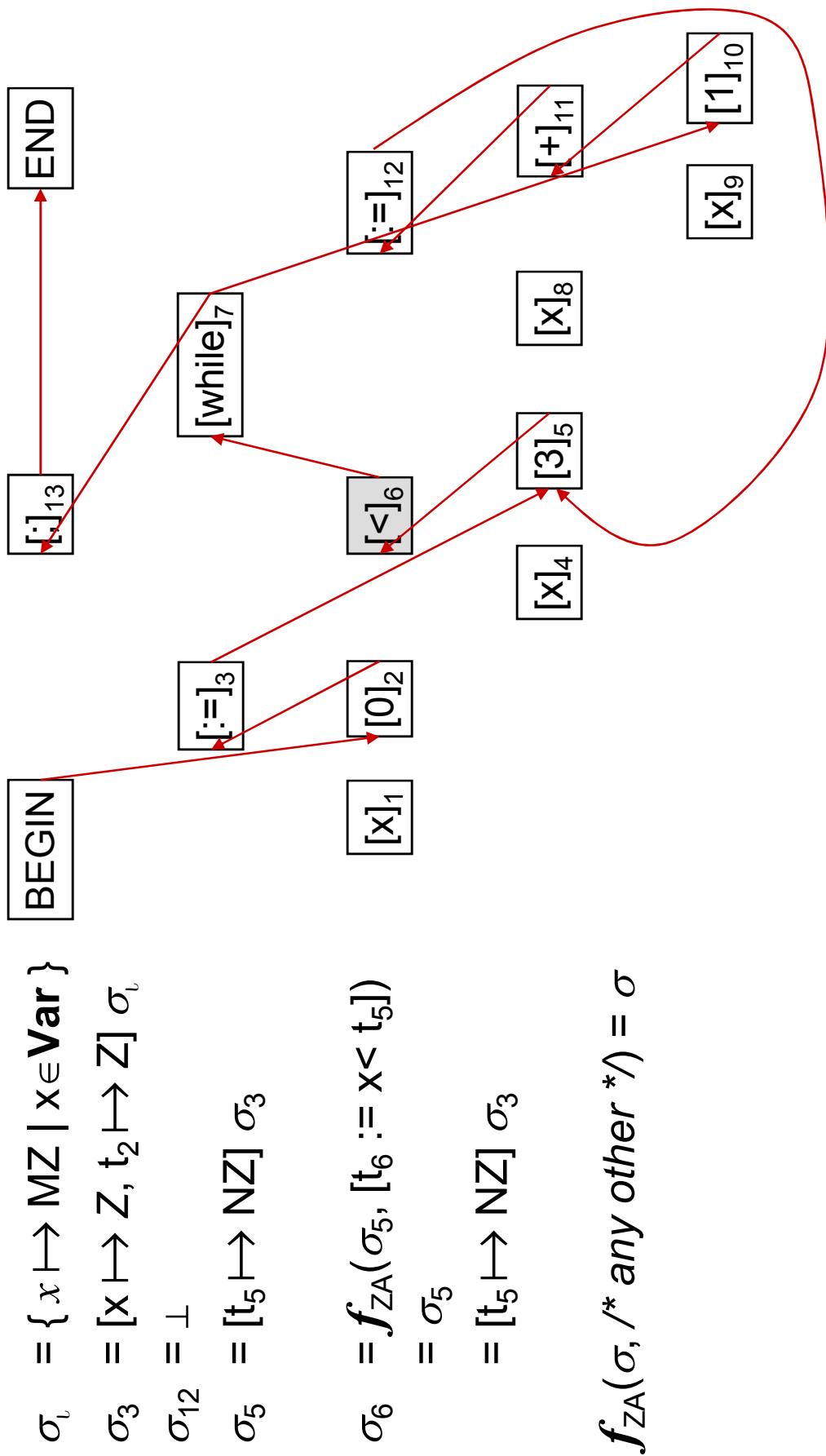
Same result as ignoring  
back edge first time



# Zero Analysis Example



# Zero Analysis Example



$$\sigma_l = \{ x \mapsto MZ \mid x \in \mathbf{Var} \}$$

$$\sigma_3 = [x \mapsto Z, t_2 \mapsto Z] \sigma_l$$

$$\sigma_{12} = \perp$$

$$\sigma_5 = [t_5 \mapsto NZ] \sigma_3$$

$$\begin{aligned} \sigma_6 &= f_{ZA}(\sigma_5, [t_6 := x < t_5]) \\ &= \sigma_5 \end{aligned}$$

$$= [t_5 \mapsto NZ] \sigma_3$$

$$f_{ZA}(\sigma, /* \text{ any other } */) = \sigma$$

# Zero Analysis Example

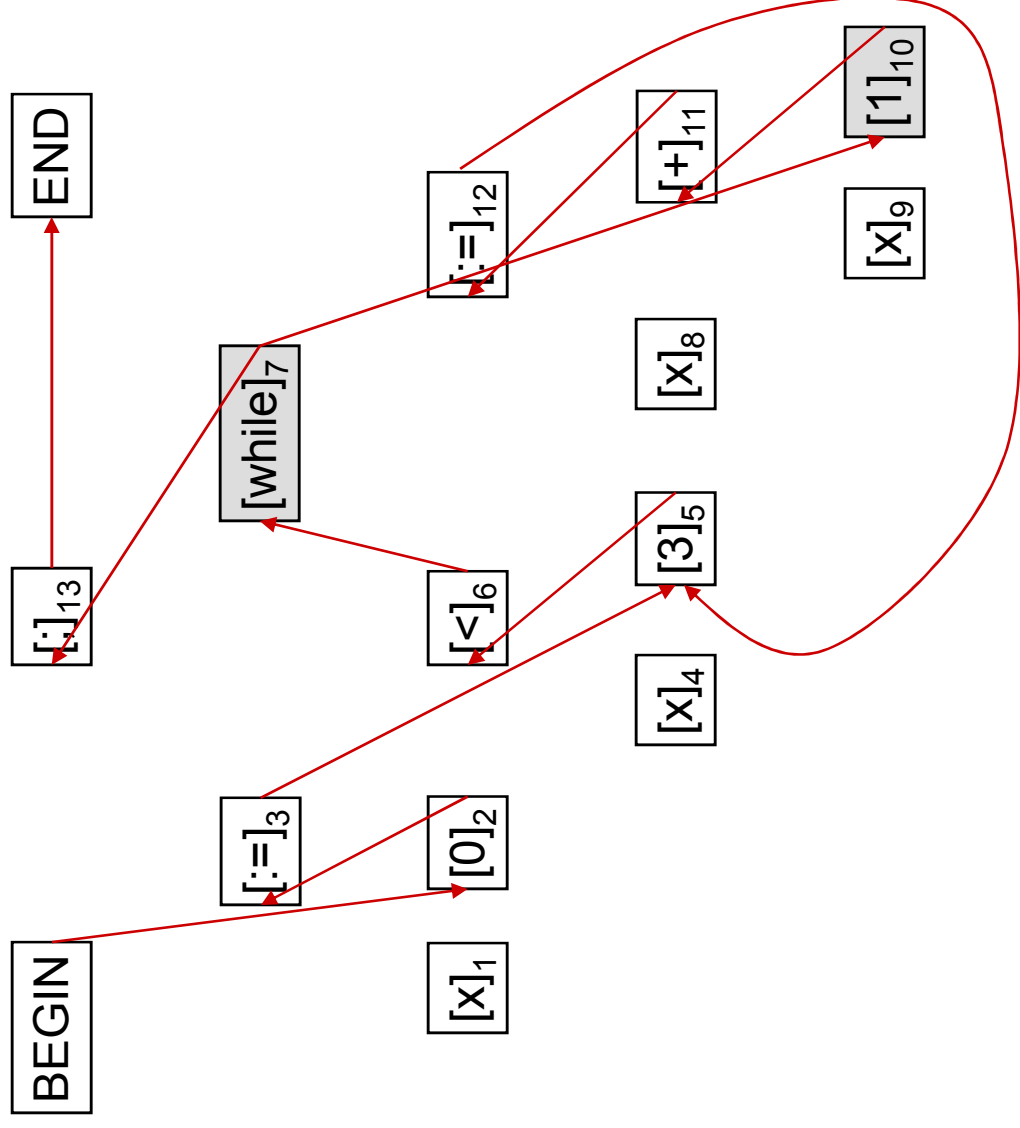
$$\sigma_l = \{ x \mapsto MZ \mid x \in \mathbf{Var} \}$$

$$\sigma_3 = [x \mapsto Z, t_2 \mapsto Z] \sigma_l$$

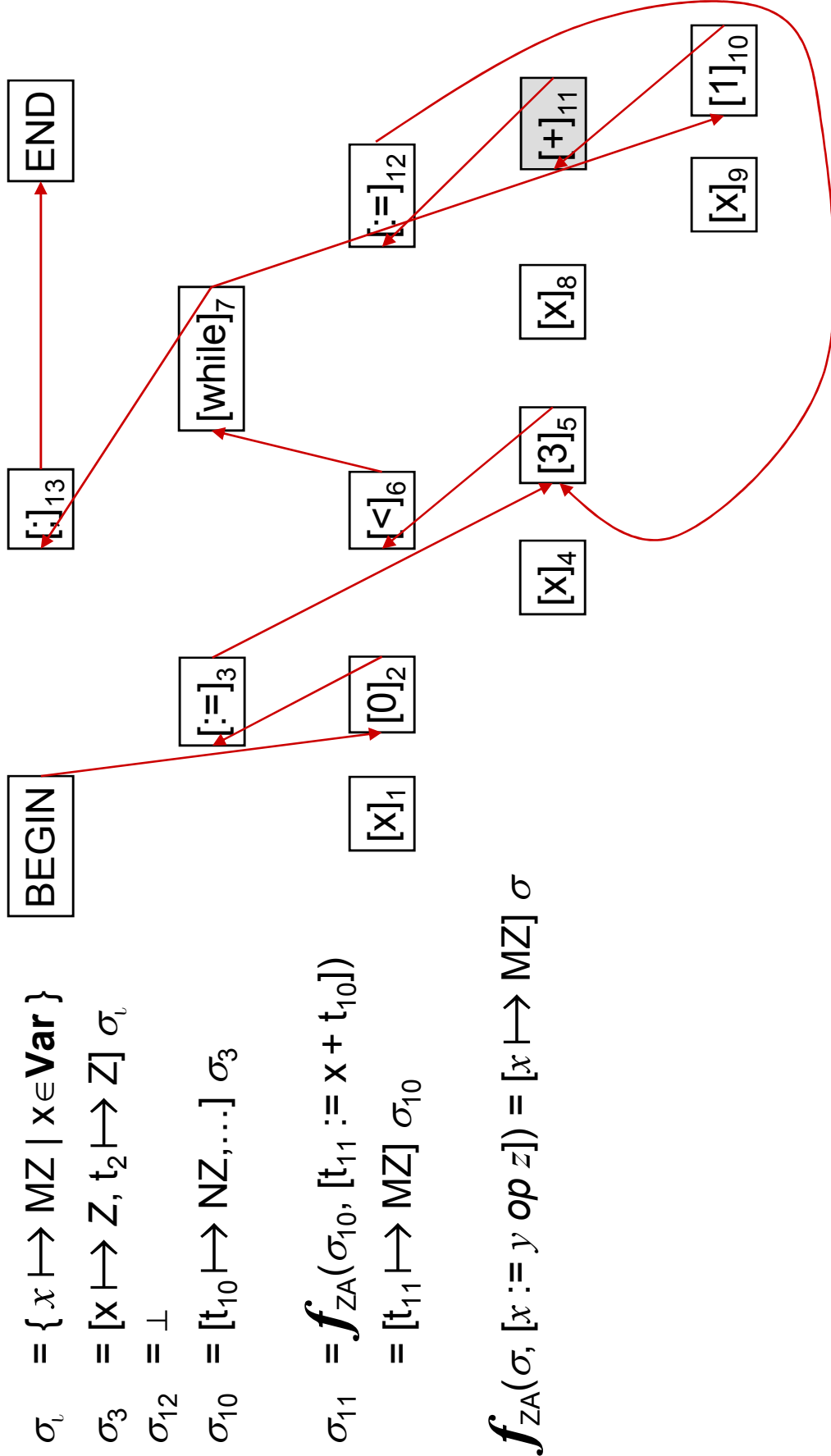
$$\sigma_{12} = \perp$$

$$\sigma_6 = [t_5 \mapsto NZ] \sigma_3$$

Skipping similar nodes...



# Zero Analysis Example



$$\sigma_l = \{x \mapsto MZ \mid x \in \mathbf{Var}\}$$

$$\sigma_3 = [x \mapsto Z, t_2 \mapsto Z] \sigma_l$$

$$\sigma_{12} = \perp$$

$$\sigma_{10} = [t_{10} \mapsto NZ, \dots] \sigma_3$$

$$\sigma_{11} = f_{ZA}(\sigma_{10}, [t_{11} := x + t_{10}])$$

$$= [t_{11} \mapsto MZ] \sigma_{10}$$

$$f_{ZA}(\sigma, [x := y \text{ op } z]) = [x \mapsto MZ] \sigma$$

# Zero Analysis Example

$\sigma_l = \{x \mapsto MZ \mid x \in \mathbf{Var}\}$  BEGIN

$\sigma_3 = [x \mapsto Z, t_2 \mapsto Z] \sigma_l$

$\sigma_{12} = \perp$

$\sigma_{11} = [t_{10} \mapsto NZ, t_{11} \mapsto MZ, \dots] \sigma_3$

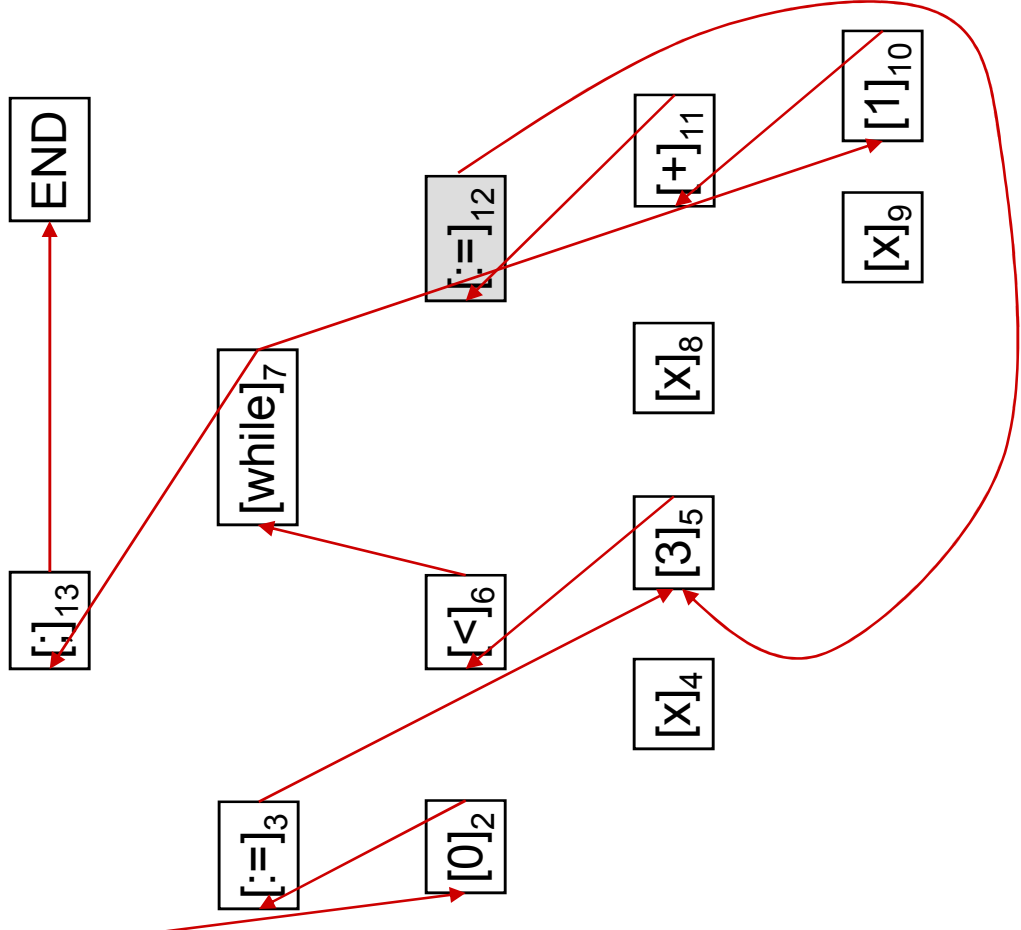
$\sigma_{12} = f_{ZA}(\sigma_{11}, [x := t_{11}])$

$= [x \mapsto \sigma_{11}(t_{11})] \sigma_{11}$

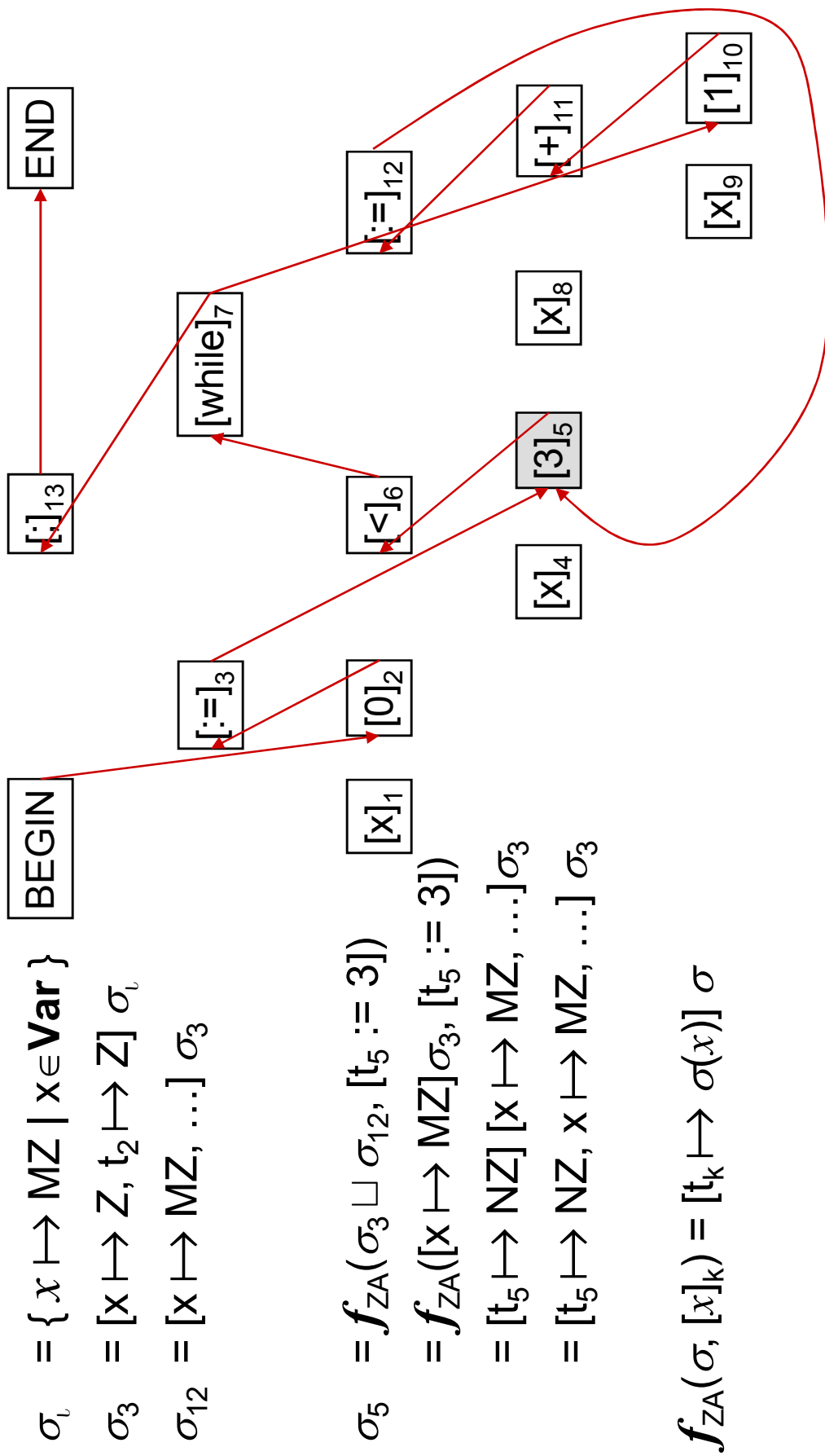
$= [x \mapsto MZ] \sigma_{11}$

$= [x \mapsto MZ, \dots] \sigma_3$

$f_{ZA}(\sigma, [x := y]) = [x \mapsto \sigma(y)] \sigma$



# Zero Analysis Example



$$\begin{aligned}
 \sigma_l &= \{x \mapsto MZ \mid x \in \mathbf{Var}\} \\
 \sigma_3 &= [x \mapsto Z, t_2 \mapsto Z] \sigma_l \\
 \sigma_{12} &= [x \mapsto MZ, \dots] \sigma_3 \\
 \sigma_5 &= f_{ZA}(\sigma_3 \sqcup \sigma_{12}, [t_5 := 3]) \\
 &= f_{ZA}([x \mapsto MZ] \sigma_3, [t_5 := 3]) \\
 &= [t_5 \mapsto NZ] [x \mapsto MZ, \dots] \sigma_3 \\
 &= [t_5 \mapsto NZ, x \mapsto MZ, \dots] \sigma_3
 \end{aligned}$$

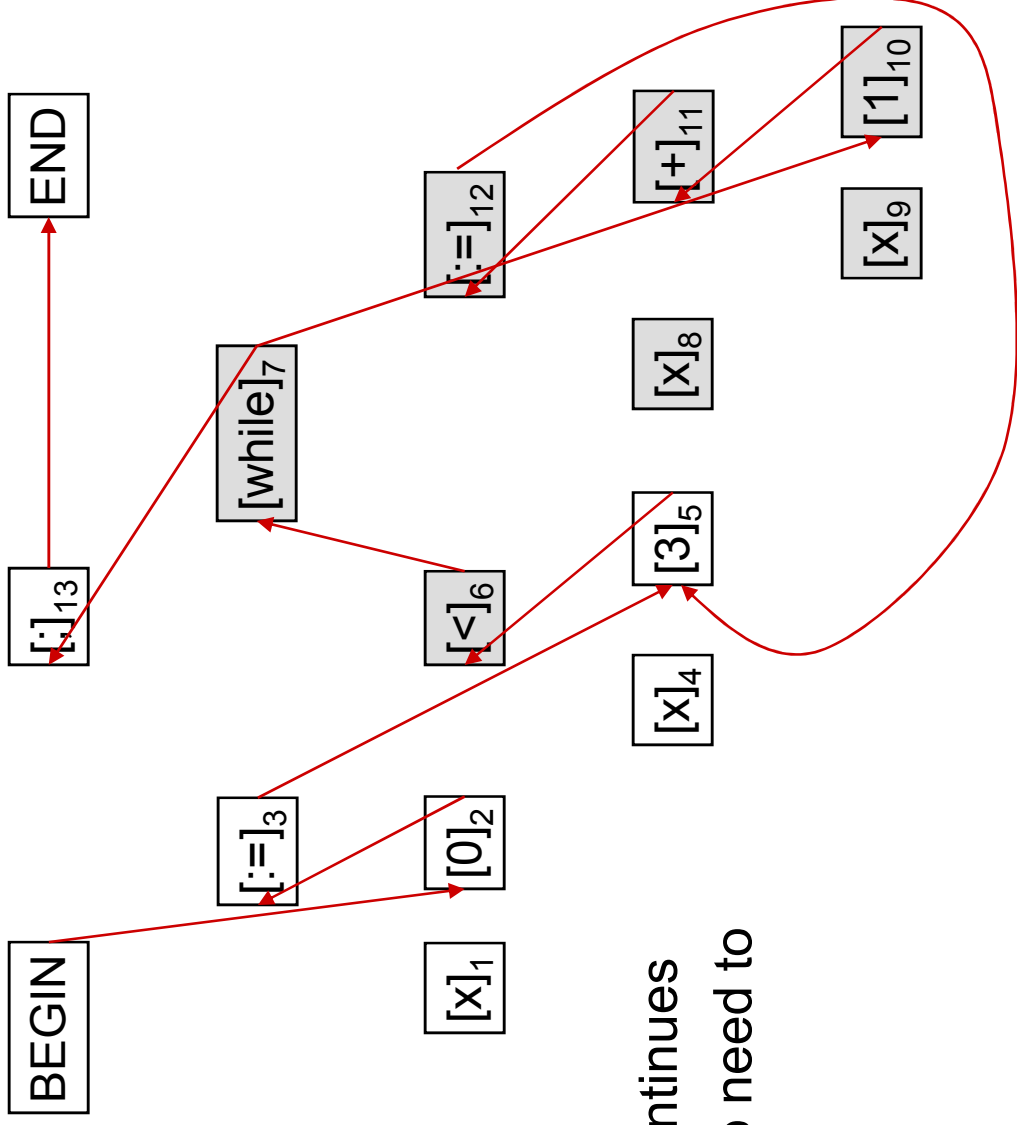
$$f_{ZA}(\sigma, [x]_k) = [t_k \mapsto \sigma(x)] \sigma$$

# Zero Analysis Example

$\sigma_l = \{ x \mapsto MZ \mid x \in \mathbf{Var} \}$

$\sigma_3 = [x \mapsto Z, t_2 \mapsto Z] \sigma_l$

$\sigma_{12} = [x \mapsto MZ, \dots] \sigma_3$

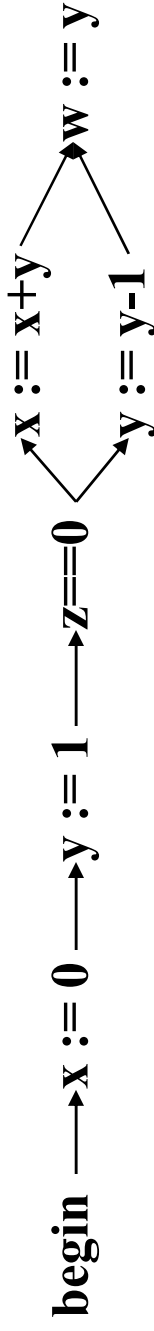


Propagation of  $x \mapsto MZ$  continues  
 $\sigma_{12}$  does not change, so no need to  
 iterate again

# Quick Quiz

---

- Explain in detail how the dataflow lattice value for after the statement  $w := y$  is computed, using the CFG below as your point of reference.



- Answer:

# Outline

---

- Introduction to Dataflow Analysis
- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - **Worklist algorithm**
- Example Dataflow Analyses
  - Constant Propagation
  - Reaching Definitions
  - Live Variable Analysis

# Kildall's Dataflow Analysis Worklist Algorithm

---

```
worklist = new Set();
for all node indexes i do
    results[i] =  $\perp_A$ ;
    results[entry] =  $\iota_A$ ;
    worklist.add(all nodes);

while (!worklist.isEmpty()) do
    i = worklist.pop();
    before =  $\bigwedge_{k \in \text{pred}(i)} \text{results}[k]$ ;
    after =  $f_A(\text{before}, \text{node}(i))$ ;
    if (!(after  $\sqsubseteq$  results[i]))
        results[i] = after;
    for all  $k \in \text{succ}(i)$  do
        worklist.add(k);
```

Ok to just add entry node  
if flow functions cannot  
return  $\perp_A$  (examples will  
assume this)

Pop removes the most  
recently added element  
from the set (performance  
optimization)

# Example of Worklist

---

$[a := 0]_1$	Position	Worklist	a	b
$[b := 0]_2$	0	1	MZ	MZ

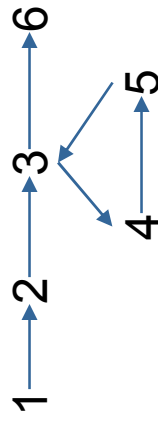
while  $[a < 2]_3$  do

$[b := a]_4;$

$[a := a + 1]_5;$

$[a := 0]_6$

Control Flow Graph



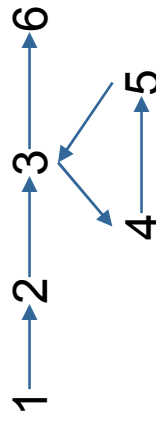
# Example of Worklist

```

[a := 0]1
[b := 0]2
while [a < 2]3 do
  [b := a]4;
  [a := a + 1]5;
[a := 0]6

```

Control Flow Graph



Position	Worklist	a	b
0		MZ	MZ
1	1	Z	MZ
2	2	Z	Z
3	3	Z	Z
4	4,6	Z	Z
5	5,6	MZ	Z
6	3,6	MZ	Z
3	4,6	MZ	MZ
4	5,6	MZ	MZ
5	3,6	MZ	MZ
3	4,6	MZ	MZ
4	6	Z	MZ

# Quick Quiz

---

Show how the worklist algorithm given in class operates on the program given, by filling in the table below.

```
1: x := 0
2: y := 1
3: if (z == 0)
4:     x := x + y
5: else y := y - 1
6: w := y
```

Position	Worklist	x	y	w
0				

# Worklist Algorithm Performance

---

```
worklist = new Set();
for all node indexes i do
    input[i] =  $\perp_A$ ;
input[entry] =  $\iota_A$ ;
worklist.add(all nodes);

while (!worklist.isEmpty()) do
    i = worklist.pop();
    after =  $f_A$ (input[i], node(i));
    for all  $k \in \text{succ}(i)$  do
        newinput = input[k]  $\sqcup$  after
        if (!(newinput  $\sqsubseteq$  input[k]))
            input[k] = newinput;
            worklist.add(k);
```

# Worklist Algorithm Performance

---

```
worklist = new Set();
for all node indexes i do
    input[i] =  $\perp_A$ ;
input[entry] =  $\iota_A$ ;
worklist.add(all nodes);

while (!worklist.isEmpty()) do
    i = worklist.pop();
    after =  $f_A$ (input[i], node(i));
    for all  $k \in \text{succ}(i)$  do
        newinput = input[k]  $\sqcup$  after
        if (!(newinput  $\sqsubseteq$  input[k]))
            input[k] = newinput;
            worklist.add(k);
```

- How many times might a node get added to the worklist?
  - The node's input must increase each time
  - The number of increases is bound by the height  $h$  of the lattice
- How many times do statements execute?
  - While loop executes  $h$  times for each node  $n$  ( $h*n$  total)
  - Statements in inner loop execute  $h$  times for each successor edge ( $h*e$  total, where  $e \geq n$ )
- Assume operation cost is  $c$ 
  - Then performance is  $O(h*e*c)$
  - Often  $h$ ,  $e$ , and  $c$  are bounded by  $n$ . So we get  $O(n^3)$
  - Good enough to run on a function, but not on the whole program

# Outline

---

- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm
- **Example Dataflow Analyses**
  - **Constant Propagation**
  - Reaching Definitions
  - Live Variable Analysis
- Dataflow Analysis Correctness
  - Termination
  - Soundness

# Constant Propagation

---

- Goal: determine which variables hold a constant value:

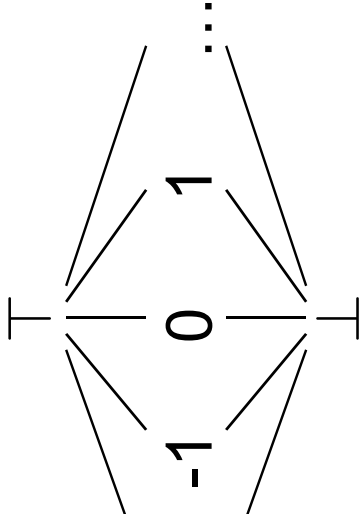
```
x := 3;  
y := x+7;  
if b  
  then z := x+2  
  else z := y-5;  
w := z-2
```

- What is w?
- Useful for optimization, error checking
- Zero analysis is a special case

# Constant Propagation Definition

---

- Constant lattice  $(L_C, \sqsubseteq_C, \sqcup_C, \perp, \top)$ 
  - $L_C = \mathbf{Integer} \cup \{\perp, \top\}$
  - $\forall n \in \mathbf{Integer} : \perp \sqsubseteq_C n \ \&\& \ n \sqsubseteq_C \top \ \dots$
- Constant propagation lattice
- Tuple lattice formed from above lattice
- See notes on zero analysis for details



- Abstraction function:
  - $\alpha_C(n) = n$
  - $\alpha_{CP}(\eta) = \{ x \mapsto \alpha_C(\eta(x)) \mid x \in \mathbf{Var} \}$
- Initial data:
  - $\iota_{CP} = \{ x \mapsto \top \mid x \in \mathbf{Var} \}$

# Constant Propagation Definition

---

- $f_{CP}(\sigma, [x := y]) = [x \mapsto \sigma(y)] \sigma$
- $f_{CP}(\sigma, [x := n]) = [x \mapsto n] \sigma$
- $f_{CP}(\sigma, [x := y \text{ op } z]) = [x \mapsto (\sigma(y) \text{ op } \sigma(z))]$   
 $\sigma$
- $n \text{ op } \top = n \text{ op } m$
- $n \text{ op } \top = \top$
- $\top \text{ op } m = \top$
- *Note: we could define for  $\perp$  too, but we won't actually ever see  $\perp$  during analysis*
- $f_{CP}(\sigma, /* \text{ any other } */) = \sigma$

# Constant Propagation Example

---

```
[x := 3]1;  
[y := x+7]2;  
if [b]3  
  then [z := x+2]4  
  else [z := y-5]5;  
[w := z-2]6
```

**Position**   **Worklist**   **x**   **y**   **z**   **w**

# Constant Propagation Example

---

```
[x := 3]1;  
[y := x+7]2;  
if [b]3  
  then [z := x+2]4  
  else [z := y-5]5;  
[w := z-2]6
```

Position	Worklist	x	y	z	w
0	1	T	T	T	T
1	2	3	T	T	T
2	3	3	10	T	T
3	4,5	3	10	T	T
4	6,5	3	10	5	T
6	5	3	10	5	3
5	6	3	10	5	T
6		3	10	5	3

# Constant Propagation Example

---

```
[x := 3]1;  
[y := x+7]2;  
if [b]3  
  then [z := x+1]4  
  else [z := y-5]5;  
[w := z-2]6
```

Position	Worklist	x	y	z	w
0	1	T	T	T	T
1	2	3	T	T	T
2	3	3	10	T	T
3	4,5	3	10	T	T

# Constant Propagation Example

---

```
[x := 3]1;  
[y := x+7]2;  
if [b]3  
  then [z := x+1]4  
  else [z := y-5]5;  
[w := z-2]6
```

Position	Worklist	x	y	z	w
0	1	T	T	T	T
1	2	3	T	T	T
2	3	3	10	T	T
3	4,5	3	10	T	T
4	6,5	3	10	4	T
6	5	3	10	4	2
5	6	3	10	5	T
6		3	10	T	T

# Loss of Precision

---

```
if [x = 0]1
  then [y := 1]2;
  else [y := x]3;
[z := 10/y]4
```

Position	Worklist	x	y	z
0	1	MZ	MZ	MZ
1	2,3	MZ	MZ	MZ
2	4,3	MZ	NZ	MZ
4	3	MZ	NZ	MZ
3	4	MZ	MZ	MZ
4		MZ	MZ	MZ

# Branch Sensitivity for Zero Analysis

---

- Existing flow functions
  - $f_{ZA}(\sigma, [x := y]) = [x \mapsto \sigma(y)] \sigma$
  - $f_{ZA}(\sigma, [x := n]) = \text{if } n == 0$ 
    - then  $[x \mapsto Z] \sigma$
    - else  $[x \mapsto NZ] \sigma$
  - $f_{ZA}(\sigma, [x := y \text{ op } z]) = [x \mapsto MZ] \sigma$
  - $f_{ZA}(\sigma, /* \text{ any other } */) = \sigma$

# Branch Sensitivity for Zero Analysis

---

- Existing flow functions
  - $f_{ZA}(\sigma, [x := y]) = [x \mapsto \sigma(y)] \sigma$
  - $f_{ZA}(\sigma, [x := n]) = \text{if } n == 0$ 
    - then  $[x \mapsto Z] \sigma$
    - else  $[x \mapsto NZ] \sigma$
  - $f_{ZA}(\sigma, [x := y \text{ op } z]) = [x \mapsto MZ] \sigma$
  - $f_{ZA}(\sigma, /* \text{ any other } */) = \sigma$
- Propagate different info on branches
  - $f_{ZA}^T(\sigma, [x = 0]) = [x \mapsto Z] \sigma$
  - $f_{ZA}^F(\sigma, [x = 0]) = [x \mapsto NZ] \sigma$
  - Slightly more general:
    - $f_{ZA}^T(\sigma, [x = y]) = [x \mapsto \sigma(y)] \sigma$
    - $f_{ZA}^F(\sigma, [x = y]) = [x \mapsto \neg \sigma(y)] \sigma$
    - Assume  $\neg Z = NZ; \neg NZ = Z; \neg MZ = MZ$

# Precision Regained

Worklist simplified to the statement level

---

```
if [x = 0]1
  then [y := 1]2;
  else [y := x]3;
[z := 10/y]4
```

	Position	Worklist	x	y	z
0		1	MZ	MZ	MZ
1 <sup>T</sup>		2,3	<b>Z</b>	MZ	MZ
1 <sup>F</sup>		2,3	<b>NZ</b>	MZ	MZ
2 (use 1 <sup>T</sup> )		4,3	Z	<b>NZ</b>	MZ
4		3	Z	NZ	MZ
3 (use 1 <sup>F</sup> )		4	NZ	<b>NZ</b>	MZ
4			<b>MZ</b>	<b>NZ</b>	MZ

# Outline

---

- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm
- Example Dataflow Analyses
  - Constant Propagation
  - Reaching Definitions
  - Live Variable Analysis
- Dataflow Analysis Correctness
  - Termination
  - Soundness

# Reaching Definitions Analysis

---

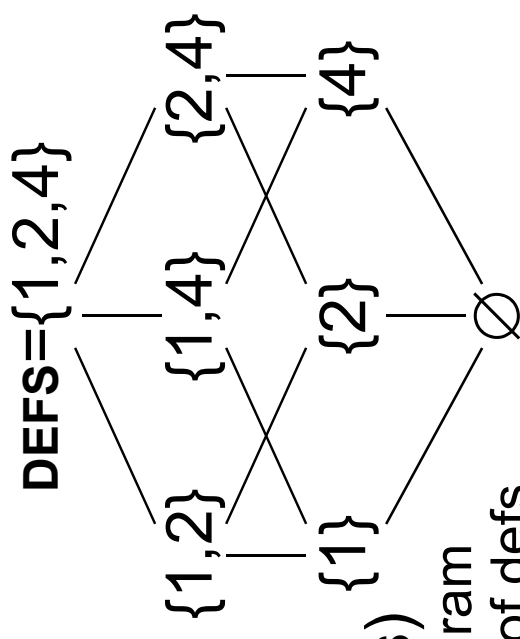
- Goal: determine which is the most recent assignment to a variable that precedes its use:

```
[y := x]1;  
[z := 1]2;  
while [y > 1]3 do  
  [z := z * y]4;  
  [y := y - 1]5;  
[y := 0]6;
```

- Example: definitions 1 and 5 reach the use of y at 4
- Applications
  - Simpler version of constant propagation, zero analysis, etc.
  - Just look at the reaching definitions for constants
  - If definitions reaching use include “undefined” sentinel, then we may be using an undefined variable

# Reaching Definitions

---



- Set Lattice ( $\mathbb{P}^{\text{DEFS}}$ ,  $\Xi_{\text{RD}}$ ,  $\sqcup_{\text{RD}}$ ,  $\sqcap_{\text{RD}}$ ,  $\emptyset$ , **DEFS**)
  - **DEFS** is the set of definitions in the program
  - Each element of the lattice is a subset of defs
    - $\mathbb{P}^{\text{DEFS}}$  is the powerset of **DEFS**, i.e. the set of all subsets of **DEFS**
  - Approximation
    - A definition  $d$  may reach program point  $P$  if  $d$  is in the lattice at  $P$
    - We call this a *may analysis*
  - $x \Xi_{\text{RD}} y$  iff  $x \subseteq y$
  - $x \sqcup_{\text{RD}} y = x \cup y$
  - This is a direct consequence of the definition of  $\Xi_{\text{RD}}$
  - Most precise element  $\perp = \emptyset$  (no reaching definitions)
  - Least precise element  $\top = \text{DEFS}$  (all definitions reach)

# Reaching Definitions

---

- Initially assume dummy assignments
- Represents passed values for parameters
- Represents uninitialized for non-parameters
- $\mathcal{L}_{RD} = \{x_0 \mid x \in \mathbf{Var}\}$
- Flow functions
  - $f_{RD}(\sigma, [x := \dots]_k)$ 
    - $= \sigma - \{x_m \mid x_m \in \mathbf{DEFS}(x)\} \cup \{x_k\}$
    - Kills (removes from set) all other definitions of  $x$
    - Generates (adds to set) the current definition  $x_k$
    - Kill/Gen pattern true in many analyses with set lattices
  - $f_{RD}(\sigma, / * \text{ other } */) = \sigma$

# Reaching Definitions Example

---

	Position	Worklist	Lattice Element
$[y := x]_1$ ;			
$[z := 1]_2$ ;			
while $[y > 1]_3$ do			
$[z := z * y]_4$ ;			
$[y := y - 1]_5$ ;			
$[y := 0]_6$ ;			

# Reaching Definitions Example

---

```
[y := x]1;  
[z := 1]2;  
while [y > 1]3 do  
  [z := z * y]4;  
  [y := y - 1]5;  
[y := 0]6;
```

Position	Worklist	Lattice Element
0	1	{x <sub>0</sub> , y <sub>0</sub> , z <sub>0</sub> }
1	2	{x <sub>0</sub> , y <sub>1</sub> , z <sub>0</sub> }
2	3	{x <sub>0</sub> , y <sub>1</sub> , z <sub>2</sub> }
3	4,6	{x <sub>0</sub> , y <sub>1</sub> , z <sub>2</sub> }
4	5,6	{x <sub>0</sub> , y <sub>1</sub> , z <sub>4</sub> }
5	3,6	{x <sub>0</sub> , y <sub>5</sub> , z <sub>4</sub> }
3	4,6	{x <sub>0</sub> , y <sub>1</sub> , y <sub>5</sub> , z <sub>2</sub> , z <sub>4</sub> }
4	5,6	{x <sub>0</sub> , y <sub>1</sub> , y <sub>5</sub> , z <sub>4</sub> }
5	6	{x <sub>0</sub> , y <sub>5</sub> , z <sub>4</sub> }
6		{x <sub>0</sub> , y <sub>6</sub> , z <sub>2</sub> , z <sub>4</sub> }

# Outline

---

- Dataflow Analysis Frameworks
  - Lattices
  - Abstraction functions
  - Control flow graphs
  - Flow functions
  - Worklist algorithm
- Example Dataflow Analyses
  - Constant Propagation
  - Reaching Definitions
  - **Live Variable Analysis**
- Dataflow Analysis Correctness
  - Termination
  - Soundness

# Live Variables Analysis

---

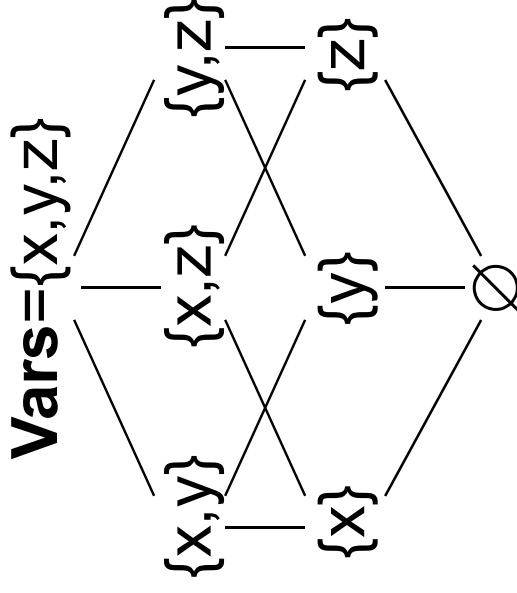
- Goal: determine which variables may be used again before they are redefined (i.e. are live) at the current program point:

```
[y := x]1;  
[z := 1]2;  
while [y > 1]3 do  
    [z := z * y]4;  
    [y := y - 1]5;  
[y := 0]6;
```

- Example: after statement 1, y is live, but x and z are not
- Optimization applications
  - If a variable is not live after it is defined, can remove the definition statement (e.g. 6 in the example)

# Live Variables Definition

---



- Set Lattice ( $\mathbb{P}\mathbf{Vars}$ ,  $\sqsubseteq_{LV}$ ,  $\sqcup_{LV}$ ,  $\perp$ ,  $\top$ ,  $\mathbf{Vars}$ )
- $\mathbf{Vars}$  is the set of variables in the program
- Each element of the lattice is a subset of  $\mathbf{Vars}$ 
  - $\mathbb{P}\mathbf{Vars}$  is the powerset of  $\mathbf{Vars}$ , i.e. the set of all subsets of  $\mathbf{Vars}$
  - $x \sqsubseteq_{LV} y$  iff  $x \subseteq y$
  - $x \sqcup_{LV} y = x \cup y$
  - Most precise element  $\perp = \emptyset$  (no live variables)
  - Least precise element  $\top = \mathbf{DEFS}$  (all variables live)

# Live Variables Definition

---

- Live Variables is a *backwards analysis*
  - To figure out if a variable is live, you have to look at the future execution of the program
- Will  $x$  be used before it is redefined?
  - When  $x$  is defined, assume it is not live
  - When  $x$  is used, assume it is live
  - Propagate lattice elements as usual, except backwards
- Initially assume return value is live
  - $\iota_{LV} = \{x\}$  where  $x$  is the variable returned from the function

# Flow Function Practice

---

- Write flow functions for Live Variable analysis:

- $f_{LV}(\sigma, [x := e]_k) =$

- $f_{LV}(\sigma, /* \text{ any other } */) =$

# Flow Function Practice

---

- Write flow functions for Live Variable analysis:
- $f_{LV}(\sigma, [x := e]_k) = (\sigma - \{x\}) \cup \text{vars}(e)$ 
  - Kills (removes from set) the variable  $x$
  - Generates (adds to set) the variables in  $e$
  - Note: must kill first then generate (what if  $e = x$ ?)

- $f_{LV}(\sigma, [e]_k) = \sigma \cup \text{vars}(e)$

- $f_{LV}(\sigma, /* \text{ any other } */) = \sigma$

# Worklist Practice

---

Show how the worklist algorithm given in class operates on the program given, by filling in the table below.

```
[y := x]1;  
[z := 1]2;  
while [y > 1]3 do  
    [z := z * y]4;  
    [y := y - 1]5;  
[y := 0]6;  
return z;
```

Position	Worklist	Lattice Value

# Live Variables Example

---

```
[y := x]1;  
[z := 1]2;  
while [y > 1]3 do  
  [z := z * y]4;  
  [y := y - 1]5;  
[y := 0]6;  
return z;
```

Position	Worklist	Lattice Element
exit	6	{z}
6	3	{z}
3	5,2	{y,z}
5	4,2	{y,z}
4	3,2	{y,z}
3	2	{y,z}
2	1	{y}
1		{x}