

# Analysis Correctness

(continued)

Reading: NNH 2.2 (optional)

17-654/17-765

Analysis of Software Artifacts

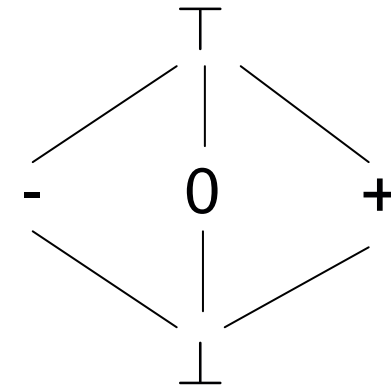
Jonathan Aldrich

# Set Comprehension Notation

- $\{ (x,y) \mid x \in S_1, y \in S_2, x < y \}$ 
  - All pairs  $(x,y)$
  - Such that  $x$  is in set  $S_1$  and  $y$  is in set  $S_2$
  - And  $x < y$
- Common notation error on the homework
  - $\{ (x,y) \mid \forall x \in S_1, \forall y \in S_2, x < y \}$
  - Don't need  $\forall$ , quantification is implicit in set comprehension

# Soundness Example: Sign Analysis

## Custom Lattice



- Transfer functions

- $\sigma$  is input data flow value

- $f^{SA}([x := a], \sigma) = \sigma [x \mapsto SA(a, \sigma)]$

- $f^{SA}([\text{skip}], \sigma) = \sigma$

- $f^{SA}([b], \sigma) = \sigma$

- $SA(n, \sigma) = \text{sign}(n)$

*// returns sign of n*

- $SA(x, \sigma) = \sigma(x)$

- $SA(a_1 + a_2, \sigma) = +$

***// is this sound?***

- $SA(a_1 \text{ op}_a a_2, \sigma) = \top$

*for op<sub>a</sub> ≠ +*

# Local Soundness

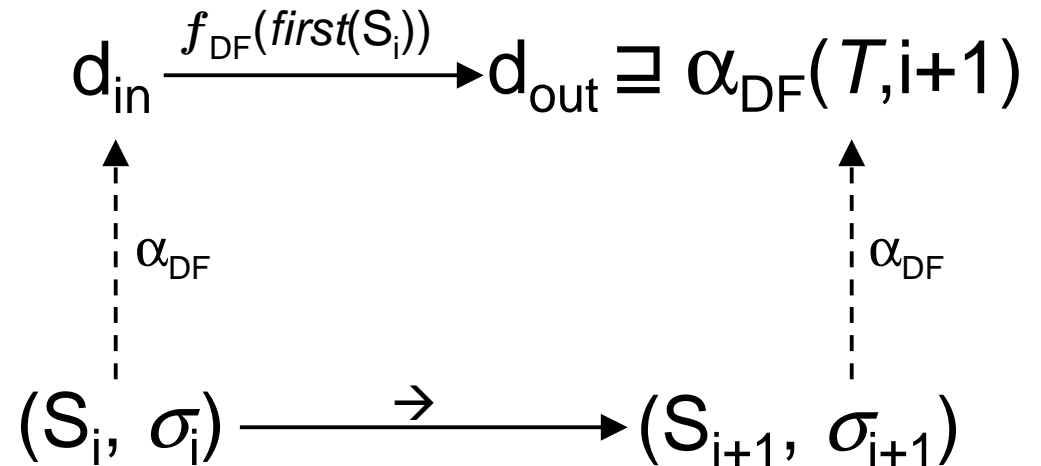
To show:

if  $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$

and  $d_{in} = \alpha_{DF}(T, i)$

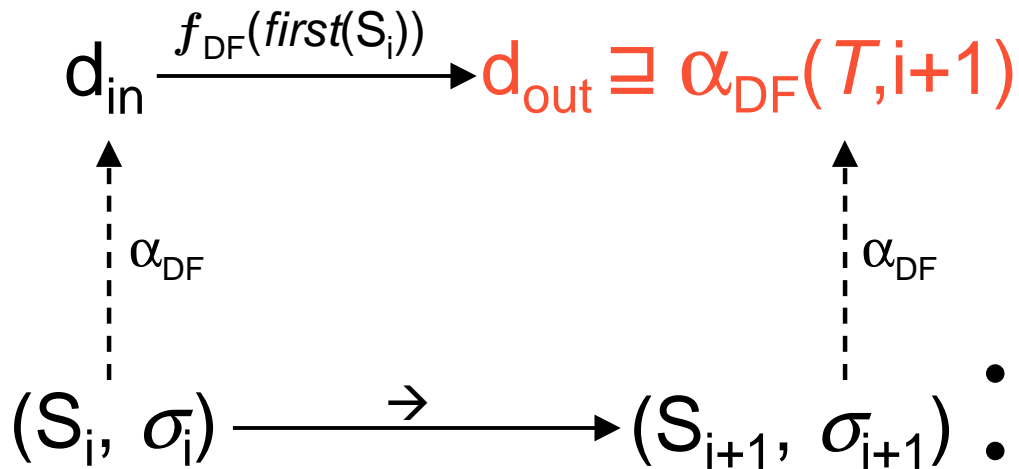
and  $d_{out} = f_{DF}(first(S_i), d_{in})$

then  $\alpha_{DF}(T, i+1) \sqsubseteq d_{out}$



Intuitively, translating from concrete to abstract and applying the flow function will safely approximate ( $\sqsubseteq$ ) taking a step in the trace and translating from concrete to abstract

# Local Soundness Fails



**Program:**

$[z := y-7]^1$

$[x := y+z]^2$

**Trace  $T$ :**

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

- $\sigma_1 = (x=0, y=3, z=-4)$
- $\sigma_2 = (x=-1, y=3, z=-4)$
- $d_{in} = \alpha_{DF}(T, 1) = (x=0, y=+, z=-)$
- $\alpha_{DF}(T, 2) = (x=-, y=+, z=-)$
- $d_{out} = f^{SA}([x := y+z]^2, d_{in})$   
 $= d_{in} [x \mapsto SA(y+z, d_{in})]$   
 $= d_{in} [x \mapsto +]$   
 $= (x=+, y=+, z=-)$
- $(x=+, y=+, z=-) \not\equiv (x=-, y=+, z=-)$

# The Proof Fails Too (*it better!*)

To show:

if  $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$

and  $d_{in} = \alpha_{CP}(T, i)$

and  $d_{out} = f_{CP}(first(S_i), d_{in})$

then  $\alpha_{CP}(T, i+1) \sqsubseteq d_{out}$

- Case:  $S_i = [x := a]^\ell$ 
  - $\sigma_{i+1} = \sigma_i [x \mapsto \mathcal{A}(a, \sigma_i)]$
  - $d_{in} = \alpha_{CP}(T, i) = sign(\sigma_i)$
  - $d_{out} = f_{CP}([x := a]^\ell, d_{in})$   
 $= d_{in} [x \mapsto SA(a, d_{in})]$
  - $\alpha_{CP}(T, i+1) = sign(\sigma_{i+1})$   
 $= sign(\sigma_i [x \mapsto \mathcal{A}(a, \sigma_i)])$
  - Lemma:  $\forall a, \sigma_i \ sign(\mathcal{A}(a, \sigma_i))$   
 $\sqsubseteq SA(a, sign(\sigma_i))$
  - Thus  $sign(\sigma_i [x \mapsto \mathcal{A}(a, \sigma_i)])$   
 $\sqsubseteq sign(\sigma_i) [x \mapsto SA(a, sign(\sigma_i))]$

# The Proof Fails Too (*it better!*)

To show:

if  $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$

and  $d_{in} = \alpha_{CP}(T, i)$

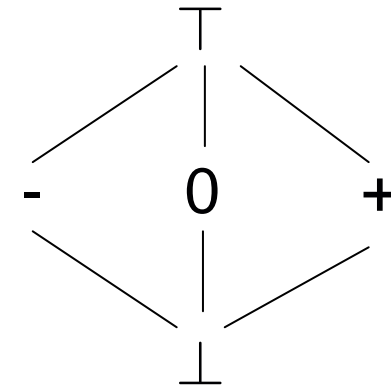
and  $d_{out} = f_{CP}(first(S_i), d_{in})$

then  $\alpha_{CP}(T, i+1) \sqsubseteq d_{out}$

- Lemma:  $\forall a, \sigma_i \text{ sign}(\mathcal{A}(a, \sigma_i)) \sqsubseteq SA(a, \text{sign}(\sigma_i))$ 
  - Proof by induction on structure of a
  - Case:  $a = n$ 
    - $\text{sign}(\mathcal{A}(n, \sigma_i)) = \text{sign}(n)$
    - $SA(n, \text{sign}(\sigma_i)) = \text{sign}(n)$
    - $\text{sign}(n) \sqsubseteq \text{sign}(n)$  **OK!**
  - **Case:  $a = a_1 + a_2$** 
    - Assume ind. hyp. for  $a_1$  and  $a_2$
    - Therefore  $\text{sign}(\mathcal{A}(a_k, \sigma_i)) \sqsubseteq SA(a_k, \text{sign}(\sigma_i))$ ,  $k \in 1, 2$
    - Subcase:  $SA(a_k, \text{sign}(\sigma_i)) = -$   
Then  $\text{sign}(\mathcal{A}(a_k, \sigma_i)) = -$   
Now  $SA(a, \text{sign}(\sigma_i)) = +$   
But clearly  $\text{sign}(\mathcal{A}(a, \sigma_i)) = -$   
So  $\text{sign}(\mathcal{A}(a, \sigma_i)) \not\sqsubseteq SA(a, \text{sign}(\sigma_i))$   
**Fails!**

# Corrected Sign Analysis

## Custom Lattice



- Transfer functions

- $\sigma$  is input data flow value

- $f^{SA}([x := a], \sigma) = \sigma [x \mapsto SA(a, \sigma)]$

- $f^{SA}([\text{skip}], \sigma) = \sigma$

- $f^{SA}([b], \sigma) = \sigma$

- $SA(n, \sigma) = \text{sign}(n)$  // returns sign of  $n$

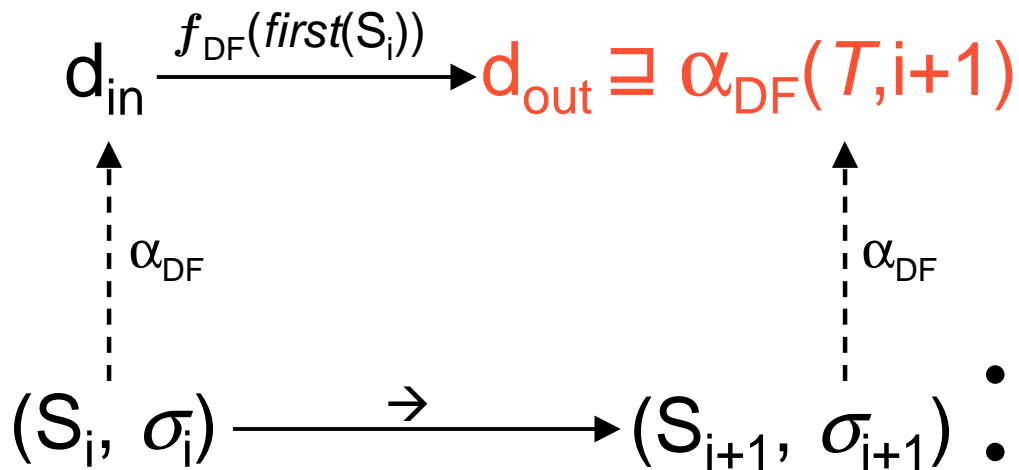
- $SA(x, \sigma) = \sigma(x)$

- $SA(a_1 + a_2, \sigma) = +$  when  $SA(a_1, \sigma) = SA(a_2, \sigma) = +$

- $SA(a_1 + a_2, \sigma) = -$  when  $SA(a_1, \sigma) = SA(a_2, \sigma) = -$

2/8/2005  $SA(a_1 \text{ op}_a a_2, \sigma) = \top$  otherwise

# Local Soundness Succeeds



**Program:**

$[z := y-7]^1$   
 $[x := y+z]^2$

**Trace  $T$ :**

t	pp	x	y	z
0	1	0	3	0
1	2	0	3	-4
2	-	-1	3	-4

- $\sigma_1 = (x=0, y=3, z=-4)$
- $\sigma_2 = (x=-1, y=3, z=-4)$
- $d_{in} = \alpha_{DF}(T, 1) = (x=0, y=+, z=-)$
- $\alpha_{DF}(T, 2) = (x=-, y=+, z=-)$
- $d_{out} = f^{SA}([x := y+z]^2, d_{in})$   
 $= d_{in} [x \mapsto SA(y+z, d_{in})]$   
 $= d_{in} [x \mapsto \top]$   
 $= (x=\top, y=+, z=-)$
- $(x=\top, y=+, z=-) \sqsupseteq (x=-, y=+, z=-)$

# The Proof Succeeds Now

To show:

if  $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$

and  $d_{in} = \alpha_{CP}(T, i)$

and  $d_{out} = f_{CP}(first(S_i), d_{in})$

then  $\alpha_{CP}(T, i+1) \sqsubseteq d_{out}$

- Case:  $S_i = [x := a]^\ell$ 
  - $\sigma_{i+1} = \sigma_i [x \mapsto \mathcal{A}(a, \sigma_i)]$
  - $d_{in} = \alpha_{CP}(T, i) = sign(\sigma_i)$
  - $d_{out} = f_{CP}([x := a]^\ell, d_{in})$   
 $= d_{in} [x \mapsto SA(a, d_{in})]$
  - $\alpha_{CP}(T, i+1) = sign(\sigma_{i+1})$   
 $= sign(\sigma_i [x \mapsto \mathcal{A}(a, \sigma_i)])$
  - Lemma:  $\forall a, \sigma_i \ sign(\mathcal{A}(a, \sigma_i))$   
 $\sqsubseteq SA(a, sign(\sigma_i))$
  - Thus  $sign(\sigma_i [x \mapsto \mathcal{A}(a, \sigma_i)])$   
 $\sqsubseteq sign(\sigma_i) [x \mapsto SA(a, sign(\sigma_i))]$

# The Proof Succeeds Now

To show:

if  $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$

and  $d_{in} = \alpha_{CP}(T, i)$

and  $d_{out} = f_{CP}(first(S_i), d_{in})$

then  $\alpha_{CP}(T, i+1) \sqsubseteq d_{out}$

- Lemma:  $\forall a, \sigma_i \text{ sign}(\mathcal{A}(a, \sigma_i)) \sqsubseteq SA(a, \text{sign}(\sigma_i))$ 
  - Proof by induction on structure of a
  - Case:  $a = n$ 
    - $\text{sign}(\mathcal{A}(n, \sigma_i)) = \text{sign}(n)$
    - $SA(n, \text{sign}(\sigma_i)) = \text{sign}(n)$
    - $\text{sign}(n) \sqsubseteq \text{sign}(n)$  **OK!**
  - **Case:  $a = a_1 + a_2$** 
    - Assume ind. hyp. for  $a_1$  and  $a_2$
    - Therefore  $\text{sign}(\mathcal{A}(a_k, \sigma_i)) \sqsubseteq SA(a_k, \text{sign}(\sigma_i))$ ,  $k \in 1, 2$
    - Subcase:  $SA(a_k, \text{sign}(\sigma_i)) = -$   
 Then  $\text{sign}(\mathcal{A}(a_k, \sigma_i)) = -$   
 Now  $SA(a, \text{sign}(\sigma_i)) = -$   
 And clearly  $\text{sign}(\mathcal{A}(a, \sigma_i)) = -$   
 So  $\text{sign}(\mathcal{A}(a, \sigma_i)) \sqsubseteq SA(a, \text{sign}(\sigma_i))$   
**OK!**
  - **Other cases are similar...**

# What does Correctness Mean?

- Intuition
  - At a fixed point, analysis results are a *conservative abstraction of program execution*
- *Soundness* condition
  - When data flow analysis reaches a **fixed point**  $F$ , then for all traces  $T$  and all times  $t$  in each trace,  $\alpha(T, t) \sqsubseteq F(pp(T, t))$

# Global Soundness

- Theorem (Global Soundness)
  - Assume that  $\forall T \in \text{traces}(S_*) \alpha_{\text{DF}}(T, 0) \sqsubseteq \iota$  and that analysis DF is monotone and locally sound with respect to  $\alpha_{\text{DF}}$
  - Then for any fixed point  $DF_{\text{fix}}$  of DF on program  $S_*$ ,  $\forall T \in \text{traces}(S_*) \forall t \in \text{times}(T)$  we have  $\alpha_{\text{DF}}(T, t) \sqsubseteq DF_{\text{fix}}(pp(T, t))$
- Proof: For each trace  $T$  we do induction on  $t$ 
  - Induction hypothesis:  $\alpha_{\text{DF}}(T, t) \sqsubseteq DF_{\text{fix}}(pp(T, t))$
  - Base case:  $t=0$ 
    - By assumption  $\alpha_{\text{DF}}(T, 0) \sqsubseteq \iota = DF_{\text{fix}}(pp(T, 0))$
  - Inductive case: time  $t$  and statement  $S_t$ 
    - *Simplifying assumption: straight-line control flow*
    - By induction hypothesis we have  $\alpha_{\text{DF}}(T, t-1) \sqsubseteq DF_{\text{fix}}(pp(T, t-1))$
    - By monotonicity of DF we have:  
 $f_{\text{DF}}(S_t, \alpha_{\text{DF}}(T, t-1)) \sqsubseteq f_{\text{DF}}(S_t, DF_{\text{fix}}(pp(T, t-1)))$
    - By local soundness we have  $\alpha_{\text{DF}}(T, t) \sqsubseteq f_{\text{DF}}(S_t, \alpha_{\text{DF}}(T, t-1))$
    - By transitivity we get  $\alpha_{\text{DF}}(T, t) \sqsubseteq f_{\text{DF}}(S_t, DF_{\text{fix}}(pp(T, t-1)))$
    - But  $f_{\text{DF}}(S_t, DF_{\text{fix}}(pp(T, t-1))) = DF_{\text{fix}}(pp(T, t))$  because it's a fixed point
    - So we have  $\alpha_{\text{DF}}(T, t) \sqsubseteq DF_{\text{fix}}(pp(T, t))$

# Abstraction for Reaching Definitions

- $\alpha_{RD}(\langle p_1, m_1 \rangle \dots \langle p_n, m_n \rangle, t) =$   
 $\{ (x, p_k) \mid x \in FV(S_*)$   
and  $stmt(p_k) = [x := a]$   
and  $k < t$   
and  $\forall j, k < j < t \text{ } stmt(p_j) \neq [x := a'] \}$
- Intuition
  - $x$  was assigned at time  $k$
  - $k$  is before the current time  $t$
  - $x$  was not assigned at any time  $j$  between  $k$  and  $t$

# Local Soundness for Reaching Definitions

To show:

if  $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$

and  $d_{in} = \alpha_{RD}(T, i)$

and  $d_{out} = f_{RD}(first(S_i), d_{in})$

then  $\alpha_{RD}(T, i+1) \sqsubseteq d_{out}$

- Case:  $S_i = [x := a]^\ell$ 
  - $d_{in} = \alpha_{RD}(T, i)$
  - $d_{out} = f_{RD}([x := a]^\ell, d_{in})$   
 $= (\alpha_{RD}(T, i) \setminus \{(x, *)\}) \cup \{(x, \ell)\}$
  - Lemma:  $\alpha_{RD}(T, i+1)$   
 $= (\alpha_{RD}(T, i) \setminus \{(x, *)\}) \cup \{(x, \ell)\}$
  - So  $\alpha_{RD}(T, i+1) = d_{out}$
  - Thus  $\alpha_{RD}(T, i+1) \sqsubseteq d_{out}$

# Local Soundness for Reaching Definitions

- Lemma: if  $S_i = [x := a]^\ell$  then
  - $\alpha_{RD}(T, i+1) = (\alpha_{RD}(T, i) \setminus \{(x, *)\}) \cup \{(x, \ell)\}$
  - $\alpha_{RD}(T, i) = \{ (y, p_k) \mid y \in FV(S_*)$   
 and  $stmt(p_k) = [y := a']$   
 and  $k < i$   
 and  $\forall j, k < j < i \text{ } stmt(p_j) \neq [y := a''] \}$
  - $\alpha_{RD}(T, i+1) = \{ (y, p_k) \mid y \in FV(S_*)$   
 and  $stmt(p_k) = [y := a']$   
 and  $k < i+1$   
 and  $\forall j, k < j < i+1 \text{ } stmt(p_j) \neq [y := a''] \}$
  - $(x, \ell) \in \alpha_{RD}(T, i+1)$  by formula with  $k = i, y = x$
  - $(x, \ell') \notin \alpha_{RD}(T, i+1)$  for  $\ell' \neq \ell$  by formula with  $j = i, y = x$
  - Other than substitutions of  $k = i$  and  $j = i$ , the formulas are the same

# Abstraction for Live Variables

- $\alpha_{LV}(\langle p_1, m_1 \rangle \dots \langle p_n, m_n \rangle, t) =$   
 $\{ x \mid x \in FV(stmt(p_k)) \text{ where } k > t$   
 $\text{and } \forall j, t < j < k \text{ } stmt(p_j) \neq [x := a'] \}$
- Intuition
  - $x$  is used at time  $k$
  - $k$  comes after the current time  $t$
  - No statement  $j$  executing between  $k$  and  $t$  assigns  $x$

# Local Soundness for Live Variables

To show:

if  $(S_i, \sigma_i) \rightarrow (S_{i+1}, \sigma_{i+1}) \in T$

and  $d_{in} = \alpha_{LV}(T, i+1)$

and  $d_{out} = f_{LV}(first(S_{i+1}), d_{in})$

then  $\alpha_{LV}(T, i) \sqsubseteq d_{out}$

- Case:  $S_{i+1} = [x := a]^\ell$ 
  - $d_{in} = \alpha_{LV}(T, i+1)$
  - $d_{out} = f_{LV}([x := a]^\ell, d_{in})$   
 $= (\alpha_{LV}(T, i+1) \setminus \{x\}) \cup FV(a)$
  - Lemma:  $\alpha_{LV}(T, i)$   
 $= (\alpha_{LV}(T, i+1) \setminus \{x\}) \cup FV(a)$
  - So  $\alpha_{LV}(T, i) = d_{out}$
  - Thus  $\alpha_{LV}(T, i) \sqsubseteq d_{out}$

Note:  $i$  and  $i+1$  are  
swapped due to reverse  
analysis

# Local Soundness for Live Variables

- Lemma: if  $S_{i+1} = [x := a]^{\ell}$  then
  - $\alpha_{LV}(T, i) = (\alpha_{LV}(T, i+1) \setminus \{x\}) \cup FV(a)$
  - $\alpha_{LV}(\langle p_1, m_1 \rangle \dots \langle p_n, m_n \rangle, i) =$   
 $\{ y \mid y \in FV(stmt(p_k)) \text{ where } k > i$   
 $\text{and } \forall j, i < j < k \text{ } stmt(p_j) \neq [y := a'] \}$
  - $\alpha_{LV}(\langle p_1, m_1 \rangle \dots \langle p_n, m_n \rangle, i+1) =$   
 $\{ y \mid y \in FV(stmt(p_k)) \text{ where } k > i+1$   
 $\text{and } \forall j, i+1 < j < k \text{ } stmt(p_j) \neq [y := a'] \}$
  - $FV(a) \in \alpha_{RD}(T, i)$  by formula with  $k = i+1, y = x$
  - $x \notin \alpha_{RD}(T, i)$  if  $x \notin FV(a)$  by formula with  $j = i+1, y = x$
  - Other than substitutions of  $k = i+1$  and  $j = i+1$ , the formulas are the same

# Iterative Worklist Algorithm

Reading: NNH 2.4

17-654/17-765

Analysis of Software Artifacts

Jonathan Aldrich

# Worklist Algorithm

```
worklist = new Stack();  
 $\forall \ell \in \text{labels}(S_*)$  do  
    analysis[ $\ell$ ] =  $\perp$ ;  
 $\forall \ell \in E$  do  
    analysis[ $\ell$ ] =  $\iota$ ;  
    worklist.addAll( $\{(\ell, \ell_2) \mid (\ell, \ell_2) \in F\}$ );  
  
while (!worklist.isEmpty()) do  
    ( $\ell_1, \ell_2$ ) = worklist.pop();  
    if  $f_{\ell_1}(\text{Analysis}[\ell_1]) \not\sqsubseteq \text{Analysis}[\ell_2]$  then  
        Analysis[ $\ell_2$ ] = Analysis[ $\ell_2$ ]  $\sqcup$   $f_{\ell_1}(\text{Analysis}[\ell_1])$   
         $\forall \ell_3$  where  $(\ell_2, \ell_3) \in F$  do  
            worklist.add( $(\ell_2, \ell_3)$ );
```

# Example of Worklist

	Iter	Edge	Worklist	a <sub>o</sub>	b <sub>o</sub>
[a := 1] <sup>1</sup>	0	-	1→2	T	T
[b := 2] <sup>2</sup>	1	1→2	2→3	1	T
	2	2→3	3→4,3→6	1	2
while [a < 2] <sup>3</sup> do	3	3→4	4→5,3→6	1	2
[b := b * 1] <sup>4</sup> ;	4	4→5	5→3,3→6	1	2
[a := a + 1] <sup>5</sup> ;	5	5→3	3→4,3→6	T	2
	6	3→4	4→5,3→6	T	2
	7	4→5	5→3,3→6	T	2
[a := b + 1] <sup>6</sup> ;	8	5→3	3→6	T	2
	9	3→6	-	T	2

# Worklist: Properties

- Correctness
  - Implements chaotic iteration, therefore correct
- Performance
  - Visits each node only when data changes
    - up to  $h$  = height of lattice
  - Propagates data along control flow edges
    - up to  $e$  = max outbound edges per node
  - Assume lattice operation cost is  $o$
  - Overall,  $O(h * e * o)$

# Worklist Algorithm

```
worklist = new Stack();  
 $\forall \ell \in \text{labels}(S_*)$  do  
    analysis[ $\ell$ ] =  $\perp$ ;  
 $\forall \ell \in E$  do  
    analysis[ $\ell$ ] =  $\nu$ ;  
worklist.addAll( $\{(\ell, \ell_2) \mid (\ell, \ell_2) \in F\}$ );
```

*h*: height of lattice and max  
times any node can change  
*n*: number of nodes  
*e*: number of edges  
*o*: cost of data flow operations

```
while (!worklist.isEmpty()) do  
    ( $\ell_1, \ell_2$ ) = worklist.pop();  
    if  $f_{\ell_1}(\text{Analysis}[\ell_1]) \not\sqsubseteq \text{Analysis}[\ell_2]$  then
```

*may execute*  $O(h \cdot e)$  times  
*cost*  $O(h \cdot e \cdot o)$

```
        Analysis[ $\ell_2$ ] = Analysis[ $\ell_2$ ]  $\sqcup$   $f_{\ell_1}(\text{Analysis}[\ell_1])$ 
```

*may execute*  $O(h \cdot n)$  times  
*cost*  $O(h \cdot n \cdot o)$

```
         $\forall \ell_3$  where  $(\ell_2, \ell_3) \in F$  do  
            worklist.add( $(\ell_2, \ell_3)$ );
```

*may execute*  $O(h \cdot e)$  times  
*cost*  $O(h \cdot e)$   
*cost*  $O(h \cdot e)$

# Performance

	<b>h</b>	<b>o</b>	$O(h * e * o)$
<b>Reaching Definitions</b>	$n$	$n$	$n^2 * e$
<b>Live Variables</b>	$v$	$v$	$v^2 * e$
<b>Constant Propagation (sets)</b>	$\infty$	$\infty$	$\infty$ (May not terminate)
<b>Constant Propagation (lattice)</b>	$2^v$	$2^v$	$v^2 * e$
<b>Sign Analysis</b>	$2^v$	$2^v$	$v^2 * e$

# Nonterminating Analysis

*(Moral: make your lattices finite height!)*

```
[x := 0]1
while [x < y]2 do
  [x := x + 1]3;
[x := 0]4;
```

Iter	Position	x	y
0	--	$\emptyset$	$\emptyset$
1	entry(1)	$\mathbb{Z}$	$\mathbb{Z}$
2	exit(1)	{0}	$\mathbb{Z}$
3	entry(2)	{0}	$\mathbb{Z}$
4	exit(2)	{0}	$\mathbb{Z}$
5	entry(3)	{0}	$\mathbb{Z}$
6	exit(3)	{1}	$\mathbb{Z}$
7	entry(2)	{0,1}	$\mathbb{Z}$
8	exit(2)	{0,1}	$\mathbb{Z}$
9	entry(3)	{0,1}	$\mathbb{Z}$
10	exit(3)	{1,2}	$\mathbb{Z}$
11	entry(2)	{0,1,2}	$\mathbb{Z}$
12	exit(2)	{0,1,2}	$\mathbb{Z}$
13	entry(3)	{0,1,2}	$\mathbb{Z}$
14	exit(3)	{1,2,3}	$\mathbb{Z}$
15	entry(2)	{0,1,2,4}	$\mathbb{Z}$
...			