

# Principles of Program Analysis:

## Data Flow Analysis

Transparencies based on Chapter 2 of the book: Flemming Nielson, Hanne Riis Nielson and Chris Hankin: [Principles of Program Analysis](#). Springer Verlag 2005. ©Flemming Nielson & Hanne Riis Nielson & Chris Hankin.

## Example Language

### Syntax of While-programs

$$a ::= x \mid n \mid a_1 \text{ op}_a a_2$$
$$b ::= \text{true} \mid \text{false} \mid \text{not } b \mid b_1 \text{ op}_b b_2 \mid a_1 \text{ op}_r a_2$$
$$S ::= [x := a]^\ell \mid [\text{skip}]^\ell \mid S_1; S_2 \mid \\ \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \mid \text{while } [b]^\ell \text{ do } S$$

Example:  $[z:=1]^1; \text{while } [x>0]^2 \text{ do } ([z:=z*y]^3; [x:=x-1]^4)$

*Abstract syntax* – parentheses are inserted to disambiguate the syntax

# Building an “Abstract Flowchart”

Example:  $[z:=1]^1; \text{while } [x>0]^2 \text{ do } ([z:=z*y]^3; [x:=x-1]^4)$

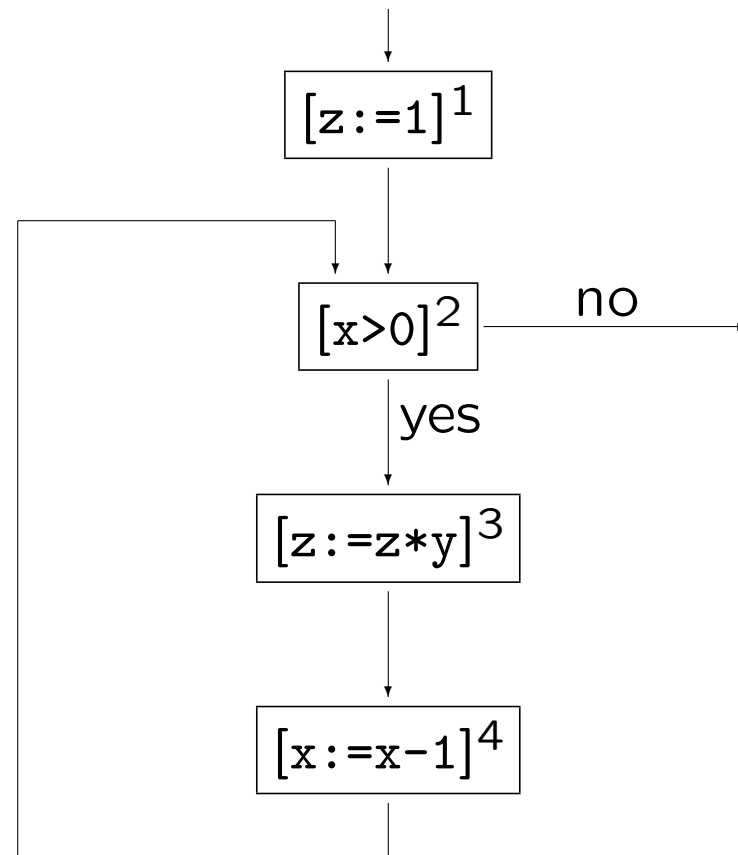
$\text{init}(\dots) = 1$

$\text{final}(\dots) = \{2\}$

$\text{labels}(\dots) = \{1, 2, 3, 4\}$

$\text{flow}(\dots) = \{(1, 2), (2, 3), (3, 4), (4, 2)\}$

$\text{flow}^R(\dots) = \{(2, 1), (2, 4), (3, 2), (4, 3)\}$



## Initial labels

*init*(*S*) is the label of the first elementary block of *S*:

*init* : Stmt  $\rightarrow$  Lab

$$\textit{init}([x := a]^\ell) = \ell$$

$$\textit{init}([\text{skip}]^\ell) = \ell$$

$$\textit{init}(S_1; S_2) = \textit{init}(S_1)$$

$$\textit{init}(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2) = \ell$$

$$\textit{init}(\text{while } [b]^\ell \text{ do } S) = \ell$$

## Example:

$$\textit{init}([z:=1]^1; \text{while } [x>0]^2 \text{ do } ([z:=z*y]^3; [x:=x-1]^4)) = 1$$

## Final labels

*final*(*S*) is the set of labels of the last elementary blocks of *S*:

$$\textit{final} : \text{Stmt} \rightarrow \mathcal{P}(\text{Lab})$$

$$\textit{final}([x := a]^\ell) = \{\ell\}$$

$$\textit{final}([\text{skip}]^\ell) = \{\ell\}$$

$$\textit{final}(S_1; S_2) = \textit{final}(S_2)$$

$$\textit{final}(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2) = \textit{final}(S_1) \cup \textit{final}(S_2)$$

$$\textit{final}(\text{while } [b]^\ell \text{ do } S) = \{\ell\}$$

## Example:

$$\textit{final}([z:=1]^1; \text{while } [x>0]^2 \text{ do } ([z:=z*y]^3; [x:=x-1]^4)) = \{2\}$$

# Labels

*labels*(*S*) is the entire set of labels in the statement *S*:

$$\textit{labels} : \text{Stmt} \rightarrow \mathcal{P}(\text{Lab})$$

$$\textit{labels}([x := a]^\ell) = \{\ell\}$$

$$\textit{labels}([\text{skip}]^\ell) = \{\ell\}$$

$$\textit{labels}(S_1; S_2) = \textit{labels}(S_1) \cup \textit{labels}(S_2)$$

$$\textit{labels}(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2) = \{\ell\} \cup \textit{labels}(S_1) \cup \textit{labels}(S_2)$$

$$\textit{labels}(\text{while } [b]^\ell \text{ do } S) = \{\ell\} \cup \textit{labels}(S)$$

## Example

$$\textit{labels}([z:=1]^1; \text{while } [x>0]^2 \text{ do } ([z:=z*y]^3; [x:=x-1]^4)) = \{1, 2, 3, 4\}$$

# Flows and reverse flows

$flow(S)$  and  $flow^R(S)$  are representations of how control flows in  $S$ :

$$flow, flow^R : \text{Stmt} \rightarrow \mathcal{P}(\text{Lab} \times \text{Lab})$$

$$flow([x := a]^\ell) = \emptyset$$

$$flow([\text{skip}]^\ell) = \emptyset$$

$$flow(S_1; S_2) = flow(S_1) \cup flow(S_2) \\ \cup \{(\ell, init(S_2)) \mid \ell \in final(S_1)\}$$

$$flow(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2) = flow(S_1) \cup flow(S_2) \\ \cup \{(\ell, init(S_1)), (\ell, init(S_2))\}$$

$$flow(\text{while } [b]^\ell \text{ do } S) = flow(S) \cup \{(\ell, init(S))\} \\ \cup \{(\ell', \ell) \mid \ell' \in final(S)\}$$

$$flow^R(S) = \{(\ell, \ell') \mid (\ell', \ell) \in flow(S)\}$$

# Elementary blocks

A statement consists of a set of *elementary blocks*

$$\text{blocks} : \text{Stmt} \rightarrow \mathcal{P}(\text{Blocks})$$

$$\text{blocks}([x := a]^\ell) = \{[x := a]^\ell\}$$

$$\text{blocks}([\text{skip}]^\ell) = \{[\text{skip}]^\ell\}$$

$$\text{blocks}(S_1; S_2) = \text{blocks}(S_1) \cup \text{blocks}(S_2)$$

$$\text{blocks}(\text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2) = \{[b]^\ell\} \cup \text{blocks}(S_1) \cup \text{blocks}(S_2)$$

$$\text{blocks}(\text{while } [b]^\ell \text{ do } S) = \{[b]^\ell\} \cup \text{blocks}(S)$$

A statement  $S$  is *label consistent* if and only if any two elementary statements  $[S_1]^\ell$  and  $[S_2]^\ell$  with the same label in  $S$  are equal:  $S_1 = S_2$

A statement *where all labels are unique* is automatically label consistent



# Intraprocedural Analysis

Classical analyses:

- Available Expressions Analysis
- Reaching Definitions Analysis
- Very Busy Expressions Analysis
- Live Variables Analysis

Derived analysis:

- Use-Definition and Definition-Use Analysis

# Available Expressions Analysis

The aim of the *Available Expressions Analysis* is to determine

For each program point, which expressions must have already been computed, and not later modified, on all paths to the program point.

Example:

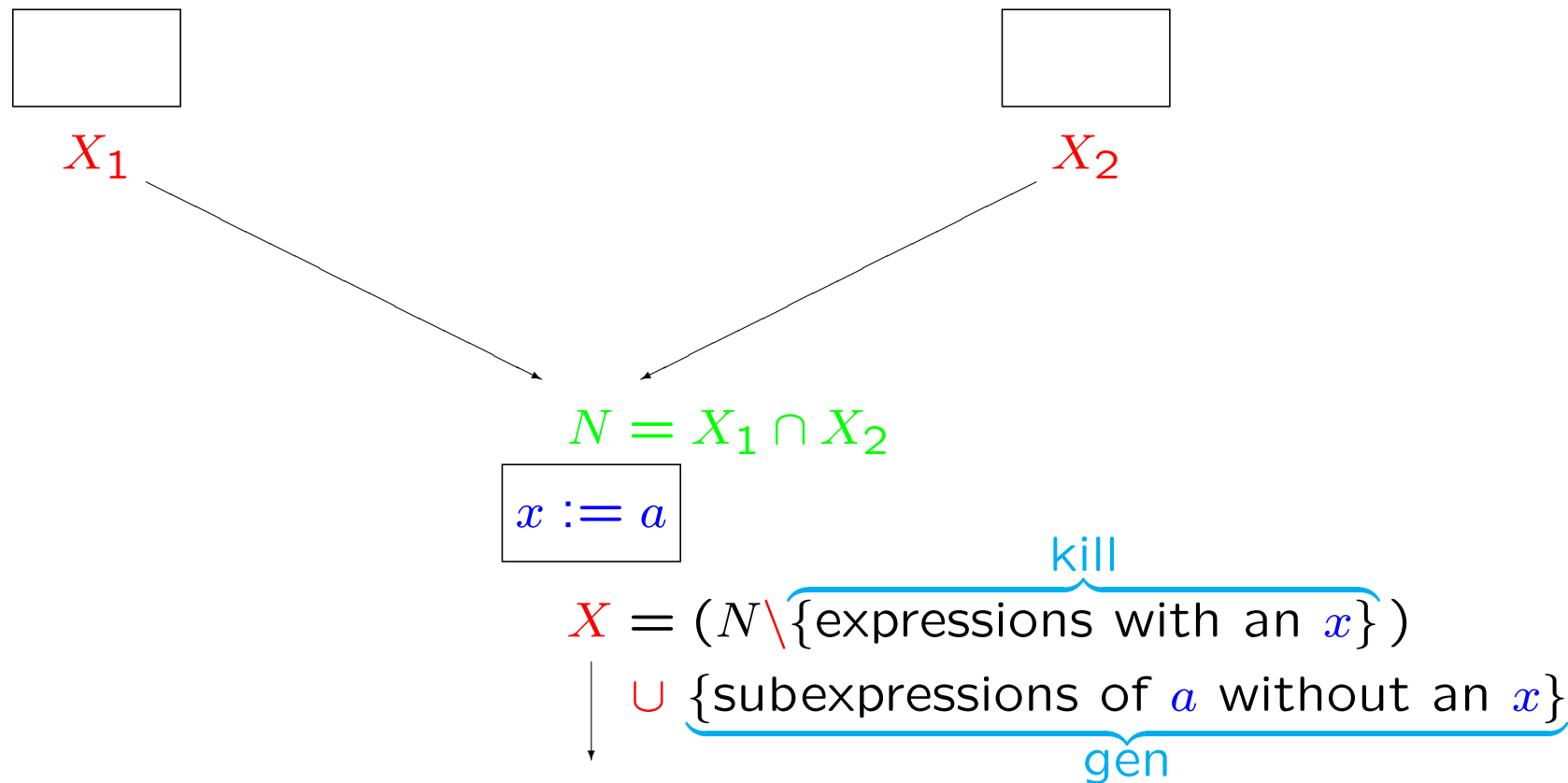
point of interest

$[x := a+b]^1; [y := a*b]^2; \text{while } [y > a+b]^3 \text{ do } ([a := a+1]^4; [x := a+b]^5)$

The analysis enables a transformation into

$[x := a+b]^1; [y := a*b]^2; \text{while } [y > x]^3 \text{ do } ([a := a+1]^4; [x := a+b]^5)$

# Available Expressions Analysis – the basic idea



# Available Expressions Analysis

*kill* and *gen* functions

---

$$\begin{aligned} \textit{kill}_{\text{AE}}([x := a]^\ell) &= \{a' \in \mathbf{AExp}_\star \mid x \in FV(a')\} \\ \textit{kill}_{\text{AE}}([\text{skip}]^\ell) &= \emptyset \\ \textit{kill}_{\text{AE}}([b]^\ell) &= \emptyset \\ \textit{gen}_{\text{AE}}([x := a]^\ell) &= \{a' \in \mathbf{AExp}(a) \mid x \notin FV(a')\} \\ \textit{gen}_{\text{AE}}([\text{skip}]^\ell) &= \emptyset \\ \textit{gen}_{\text{AE}}([b]^\ell) &= \mathbf{AExp}(b) \end{aligned}$$

data flow equations:  $\mathbf{AE}^\equiv$

---

$$\textit{AE}_{\text{entry}}(\ell) = \begin{cases} \emptyset & \text{if } \ell = \textit{init}(S_\star) \\ \bigcap \{\textit{AE}_{\text{exit}}(\ell') \mid (\ell', \ell) \in \textit{flow}(S_\star)\} & \text{otherwise} \end{cases}$$

$$\begin{aligned} \textit{AE}_{\text{exit}}(\ell) &= (\textit{AE}_{\text{entry}}(\ell) \setminus \textit{kill}_{\text{AE}}(B^\ell)) \cup \textit{gen}_{\text{AE}}(B^\ell) \\ &\text{where } B^\ell \in \textit{blocks}(S_\star) \end{aligned}$$

## Example:

$[x:=a+b]^1; [y:=a*b]^2; \text{while } [y>a+b]^3 \text{ do } ([a:=a+1]^4; [x:=a+b]^5)$

*kill* and *gen* functions:

$\ell$	$kill_{AE}(\ell)$	$gen_{AE}(\ell)$
1	$\emptyset$	$\{a+b\}$
2	$\emptyset$	$\{a*b\}$
3	$\emptyset$	$\{a+b\}$
4	$\{a+b, a*b, a+1\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

## Example (cont.):

$[x:=a+b]^1; [y:=a*b]^2; \text{while } [y>a+b]^3 \text{ do } ([a:=a+1]^4; [x:=a+b]^5)$

Equations:

$$AE_{entry}(1) = \emptyset$$

$$AE_{entry}(2) = AE_{exit}(1)$$

$$AE_{entry}(3) = AE_{exit}(2) \cap AE_{exit}(5)$$

$$AE_{entry}(4) = AE_{exit}(3)$$

$$AE_{entry}(5) = AE_{exit}(4)$$

$$AE_{exit}(1) = AE_{entry}(1) \cup \{a+b\}$$

$$AE_{exit}(2) = AE_{entry}(2) \cup \{a*b\}$$

$$AE_{exit}(3) = AE_{entry}(3) \cup \{a+b\}$$

$$AE_{exit}(4) = AE_{entry}(4) \setminus \{a+b, a*b, a+1\}$$

$$AE_{exit}(5) = AE_{entry}(5) \cup \{a+b\}$$

## Example (cont.):

$[x:=a+b]^1; [y:=a*b]^2; \text{while } [y > a+b]^3 \text{ do } ([a:=a+1]^4; [x:=a+b]^5)$

Largest solution:

$\ell$	$AE_{entry}(\ell)$	$AE_{exit}(\ell)$
1	$\emptyset$	$\{a+b\}$
2	$\{a+b\}$	$\{a+b, a*b\}$
3	$\{a+b\}$	$\{a+b\}$
4	$\{a+b\}$	$\emptyset$
5	$\emptyset$	$\{a+b\}$

# Why largest solution?

$[z:=x+y]^\ell; \text{while } [\text{true}]^{\ell'} \text{ do } [\text{skip}]^{\ell''}$

Equations:

$$AE_{\text{entry}}(\ell) = \emptyset$$

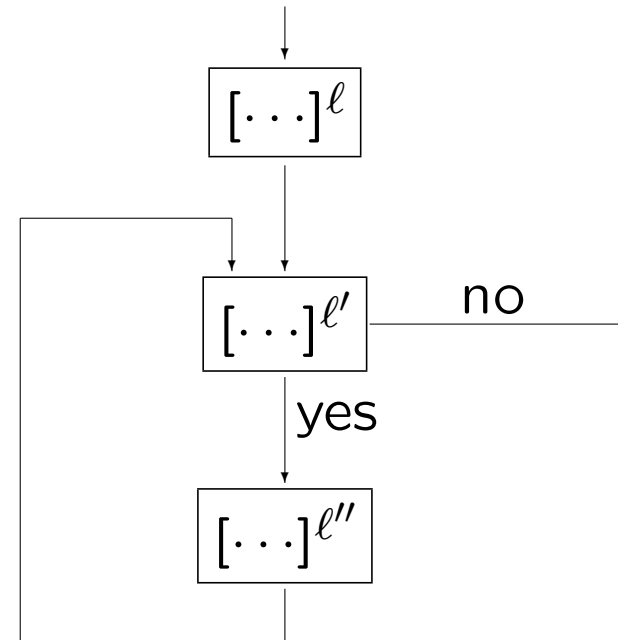
$$AE_{\text{entry}}(\ell') = AE_{\text{exit}}(\ell) \cap AE_{\text{exit}}(\ell'')$$

$$AE_{\text{entry}}(\ell'') = AE_{\text{exit}}(\ell')$$

$$AE_{\text{exit}}(\ell) = AE_{\text{entry}}(\ell) \cup \{x+y\}$$

$$AE_{\text{exit}}(\ell') = AE_{\text{entry}}(\ell')$$

$$AE_{\text{exit}}(\ell'') = AE_{\text{entry}}(\ell'')$$



After some simplification:  $AE_{\text{entry}}(\ell') = \{x+y\} \cap AE_{\text{entry}}(\ell')$

Two solutions to this equation:  $\{x+y\}$  and  $\emptyset$



# Reaching Definitions Analysis


The aim of the *Reaching Definitions Analysis* is to determine

For each program point, which assignments may have been made and not overwritten, when program execution reaches this point along some path.

Example:

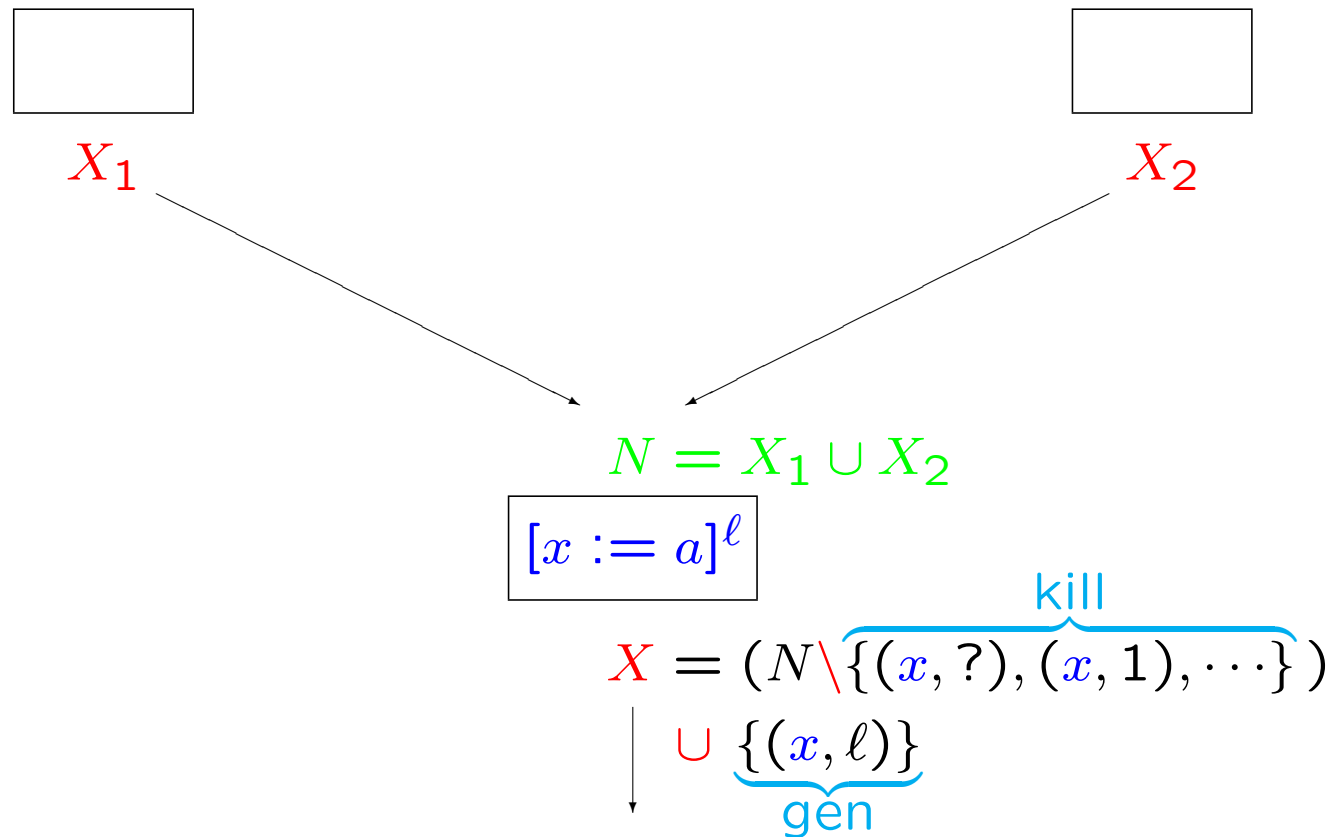
point of interest

$[x:=5]^1; [y:=1]^2; \text{while } [x>1]^3 \text{ do } ([y:=x*y]^4; [x:=x-1]^5)$



useful for definition-use chains and use-definition chains

# Reaching Definitions Analysis – the basic idea



# Reaching Definitions Analysis

*kill* and *gen* functions

---

$$\begin{aligned} \textit{kill}_{\text{RD}}([x := a]^\ell) &= \{(x, ?)\} \\ &\quad \cup \{(x, \ell') \mid B^{\ell'} \text{ is an assignment to } x \text{ in } S_\star\} \end{aligned}$$

$$\textit{kill}_{\text{RD}}([\text{skip}]^\ell) = \emptyset$$

$$\textit{kill}_{\text{RD}}([b]^\ell) = \emptyset$$

$$\textit{gen}_{\text{RD}}([x := a]^\ell) = \{(x, \ell)\}$$

$$\textit{gen}_{\text{RD}}([\text{skip}]^\ell) = \emptyset$$

$$\textit{gen}_{\text{RD}}([b]^\ell) = \emptyset$$

data flow equations:  $\text{RD}^=$

---

$$\text{RD}_{\text{entry}}(\ell) = \begin{cases} \{(x, ?) \mid x \in \text{FV}(S_\star)\} & \text{if } \ell = \textit{init}(S_\star) \\ \cup \{\text{RD}_{\text{exit}}(\ell') \mid (\ell', \ell) \in \textit{flow}(S_\star)\} & \text{otherwise} \end{cases}$$

$$\begin{aligned} \text{RD}_{\text{exit}}(\ell) &= (\text{RD}_{\text{entry}}(\ell) \setminus \textit{kill}_{\text{RD}}(B^\ell)) \cup \textit{gen}_{\text{RD}}(B^\ell) \\ &\text{where } B^\ell \in \textit{blocks}(S_\star) \end{aligned}$$

## Example:

$[x:=5]^1; [y:=1]^2; \text{while } [x>1]^3 \text{ do } ([y:=x*y]^4; [x:=x-1]^5)$

*kill* and *gen* functions:

$\ell$	$\text{kill}_{\text{RD}}(\ell)$	$\text{gen}_{\text{RD}}(\ell)$
1	$\{(x, ?), (x, 1), (x, 5)\}$	$\{(x, 1)\}$
2	$\{(y, ?), (y, 2), (y, 4)\}$	$\{(y, 2)\}$
3	$\emptyset$	$\emptyset$
4	$\{(y, ?), (y, 2), (y, 4)\}$	$\{(y, 4)\}$
5	$\{(x, ?), (x, 1), (x, 5)\}$	$\{(x, 5)\}$

## Example (cont.):

$[x:=5]^1; [y:=1]^2; \text{while } [x>1]^3 \text{ do } ([y:=x*y]^4; [x:=x-1]^5)$

Equations:

$$RD_{entry}(1) = \{(x, ?), (y, ?)\}$$

$$RD_{entry}(2) = RD_{exit}(1)$$

$$RD_{entry}(3) = RD_{exit}(2) \cup RD_{exit}(5)$$

$$RD_{entry}(4) = RD_{exit}(3)$$

$$RD_{entry}(5) = RD_{exit}(4)$$

$$RD_{exit}(1) = (RD_{entry}(1) \setminus \{(x, ?), (x, 1), (x, 5)\}) \cup \{(x, 1)\}$$

$$RD_{exit}(2) = (RD_{entry}(2) \setminus \{(y, ?), (y, 2), (y, 4)\}) \cup \{(y, 2)\}$$

$$RD_{exit}(3) = RD_{entry}(3)$$

$$RD_{exit}(4) = (RD_{entry}(4) \setminus \{(y, ?), (y, 2), (y, 4)\}) \cup \{(y, 4)\}$$

$$RD_{exit}(5) = (RD_{entry}(5) \setminus \{(x, ?), (x, 1), (x, 5)\}) \cup \{(x, 5)\}$$

## Example (cont.):

$[x:=5]^1; [y:=1]^2; \text{while } [x>1]^3 \text{ do } ([y:=x*y]^4; [x:=x-1]^5)$

Smallest solution:

$\ell$	$RD_{entry}(\ell)$	$RD_{exit}(\ell)$
1	$\{(x, ?), (y, ?)\}$	$\{(y, ?), (x, 1)\}$
2	$\{(y, ?), (x, 1)\}$	$\{(x, 1), (y, 2)\}$
3	$\{(x, 1), (y, 2), (y, 4), (x, 5)\}$	$\{(x, 1), (y, 2), (y, 4), (x, 5)\}$
4	$\{(x, 1), (y, 2), (y, 4), (x, 5)\}$	$\{(x, 1), (y, 4), (x, 5)\}$
5	$\{(x, 1), (y, 4), (x, 5)\}$	$\{(y, 4), (x, 5)\}$

## Why smallest solution?

$[z:=x+y]^\ell; \text{while } [\text{true}]^{\ell'} \text{ do } [\text{skip}]^{\ell''}$

Equations:

$$RD_{entry}(\ell) = \{(x, ?), (y, ?), (z, ?)\}$$

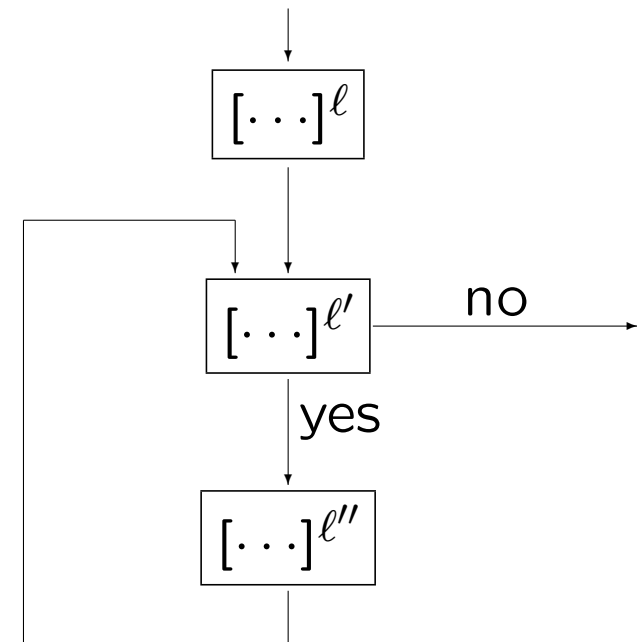
$$RD_{entry}(\ell') = RD_{exit}(\ell) \cup RD_{exit}(\ell'')$$

$$RD_{entry}(\ell'') = RD_{exit}(\ell')$$

$$RD_{exit}(\ell) = (RD_{entry}(\ell) \setminus \{(z, ?)\}) \cup \{(z, \ell)\}$$

$$RD_{exit}(\ell') = RD_{entry}(\ell')$$

$$RD_{exit}(\ell'') = RD_{entry}(\ell'')$$



After some simplification:  $RD_{entry}(\ell') = \{(x, ?), (y, ?), (z, \ell)\} \cup RD_{entry}(\ell')$

Many solutions to this equation: any superset of  $\{(x, ?), (y, ?), (z, \ell)\}$

# Very Busy Expressions Analysis

An expression is *very busy* at the exit from a label if, no matter what path is taken from the label, the expression is always used before any of the variables occurring in it are redefined.

The aim of the *Very Busy Expressions Analysis* is to determine

For each program point, which expressions must be very busy at the exit from the point.

## Example:

point of interest

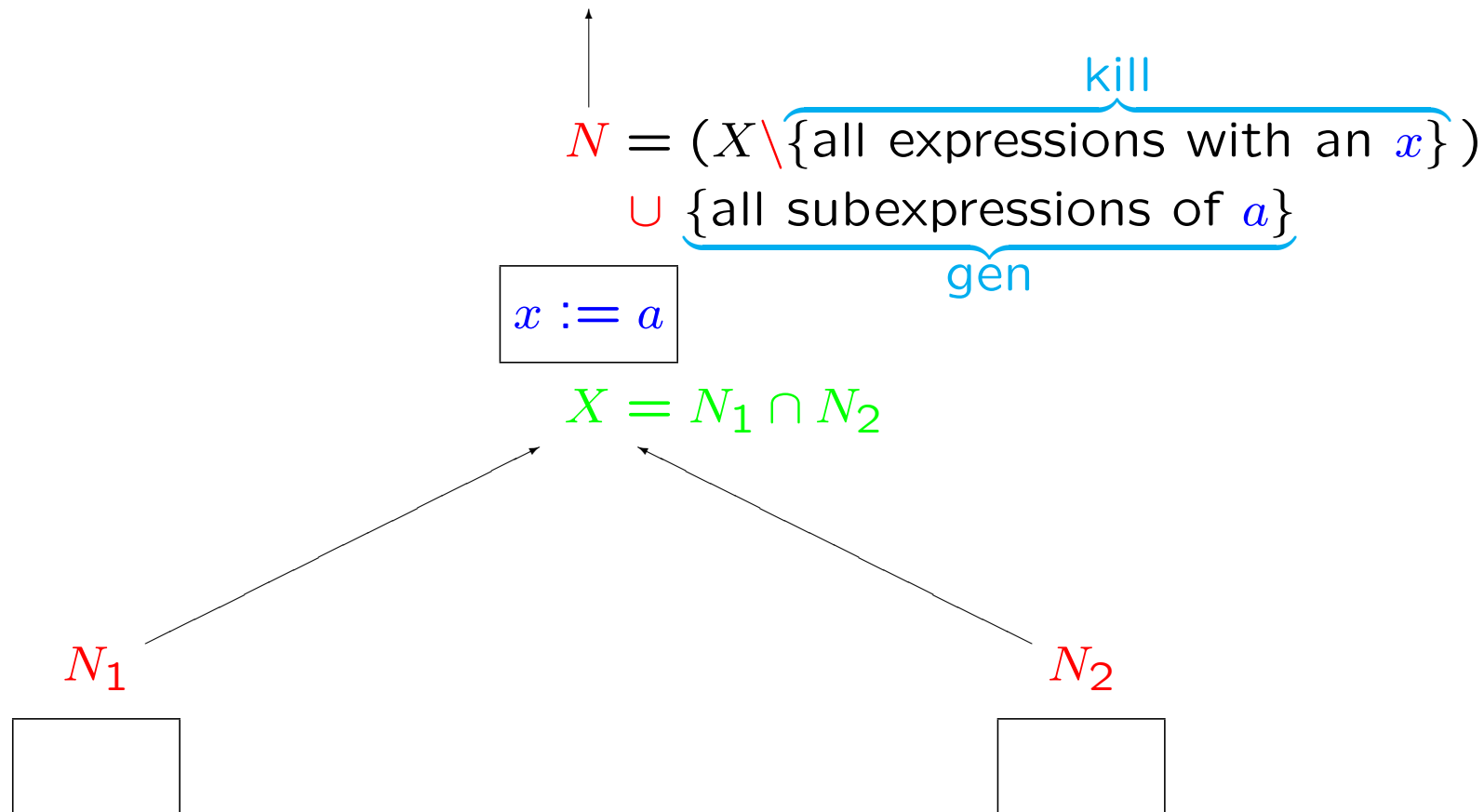
↓ if  $[a > b]^1$  then  $([x := b - a]^2; [y := a - b]^3)$  else  $([y := b - a]^4; [x := a - b]^5)$

The analysis enables a transformation into

$$[t1 := b - a]^A; [t2 := a - b]^B;$$
$$\text{if } [a > b]^1 \text{ then } ([x := t1]^2; [y := t2]^3) \text{ else } ([y := t1]^4; [x := t2]^5)$$



# Very Busy Expressions Analysis – the basic idea



# Very Busy Expressions Analysis

*kill* and *gen* functions

---

$$\textit{kill}_{\text{VB}}([x := a]^\ell) = \{a' \in \mathbf{AExp}_\star \mid x \in \text{FV}(a')\}$$

$$\textit{kill}_{\text{VB}}([\text{skip}]^\ell) = \emptyset$$

$$\textit{kill}_{\text{VB}}([b]^\ell) = \emptyset$$

$$\textit{gen}_{\text{VB}}([x := a]^\ell) = \mathbf{AExp}(a)$$

$$\textit{gen}_{\text{VB}}([\text{skip}]^\ell) = \emptyset$$

$$\textit{gen}_{\text{VB}}([b]^\ell) = \mathbf{AExp}(b)$$

data flow equations:  $\text{VB}^\text{=}$

---

$$\text{VB}_{\text{exit}}(\ell) = \begin{cases} \emptyset & \text{if } \ell \in \text{final}(S_\star) \\ \bigcap \{ \text{VB}_{\text{entry}}(\ell') \mid (\ell', \ell) \in \text{flow}^R(S_\star) \} & \text{otherwise} \end{cases}$$

$$\text{VB}_{\text{entry}}(\ell) = (\text{VB}_{\text{exit}}(\ell) \setminus \textit{kill}_{\text{VB}}(B^\ell)) \cup \textit{gen}_{\text{VB}}(B^\ell)$$

where  $B^\ell \in \text{blocks}(S_\star)$

## Example:

if  $[a > b]^1$  then  $([x := b - a]^2; [y := a - b]^3)$  else  $([y := b - a]^4; [x := a - b]^5)$

*kill* and *gen* function:

$\ell$	$kill_{VB}(\ell)$	$gen_{VB}(\ell)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{b - a\}$
3	$\emptyset$	$\{a - b\}$
4	$\emptyset$	$\{b - a\}$
5	$\emptyset$	$\{a - b\}$

## Example (cont.):

if  $[a > b]^1$  then  $([x := b - a]^2; [y := a - b]^3)$  else  $([y := b - a]^4; [x := a - b]^5)$

Equations:

$$VB_{entry}(1) = VB_{exit}(1)$$

$$VB_{entry}(2) = VB_{exit}(2) \cup \{b - a\}$$

$$VB_{entry}(3) = \{a - b\}$$

$$VB_{entry}(4) = VB_{exit}(4) \cup \{b - a\}$$

$$VB_{entry}(5) = \{a - b\}$$

$$VB_{exit}(1) = VB_{entry}(2) \cap VB_{entry}(4)$$

$$VB_{exit}(2) = VB_{entry}(3)$$

$$VB_{exit}(3) = \emptyset$$

$$VB_{exit}(4) = VB_{entry}(5)$$

$$VB_{exit}(5) = \emptyset$$

## Example (cont.):

if  $[a > b]^1$  then  $([x := b - a]^2; [y := a - b]^3)$  else  $([y := b - a]^4; [x := a - b]^5)$

Largest solution:

$\ell$	$VB_{entry}(\ell)$	$VB_{exit}(\ell)$
1	$\{a - b, b - a\}$	$\{a - b, b - a\}$
2	$\{a - b, b - a\}$	$\{a - b\}$
3	$\{a - b\}$	$\emptyset$
4	$\{a - b, b - a\}$	$\{a - b\}$
5	$\{a - b\}$	$\emptyset$

## Why largest solution?

$(\text{while } [x > 1]^\ell \text{ do } [\text{skip}]^{\ell'}); [x := x + 1]^{\ell''}$

Equations:

$$VB_{entry}(\ell) = VB_{exit}(\ell)$$

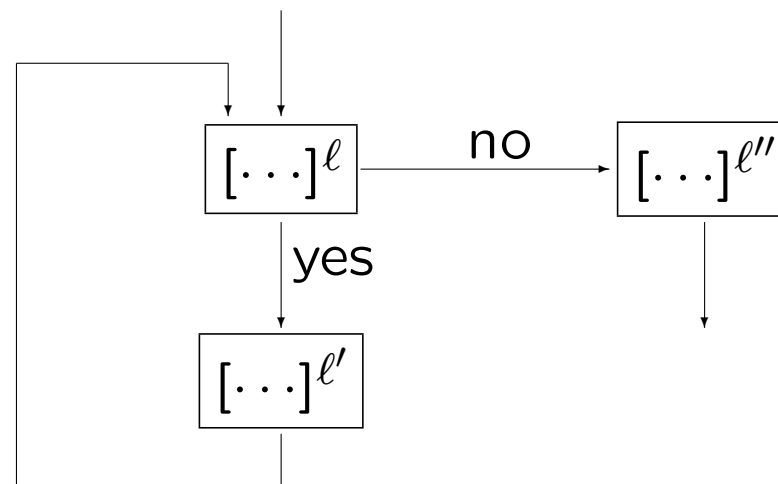
$$VB_{entry}(\ell') = VB_{exit}(\ell')$$

$$VB_{entry}(\ell'') = \{x+1\}$$

$$VB_{exit}(\ell) = VB_{entry}(\ell') \cap VB_{entry}(\ell'')$$

$$VB_{exit}(\ell') = VB_{entry}(\ell)$$

$$VB_{exit}(\ell'') = \emptyset$$



After some simplifications:  $VB_{exit}(\ell) = VB_{exit}(\ell) \cap \{x+1\}$

Two solutions to this equation:  $\{x+1\}$  and  $\emptyset$

# Live Variables Analysis

A variable is *live* at the exit from a label if there is a path from the label to a use of the variable that does not re-define the variable.

The aim of the *Live Variables Analysis* is to determine

For each program point, which variables may be live at the exit from the point.

## Example:

point of interest

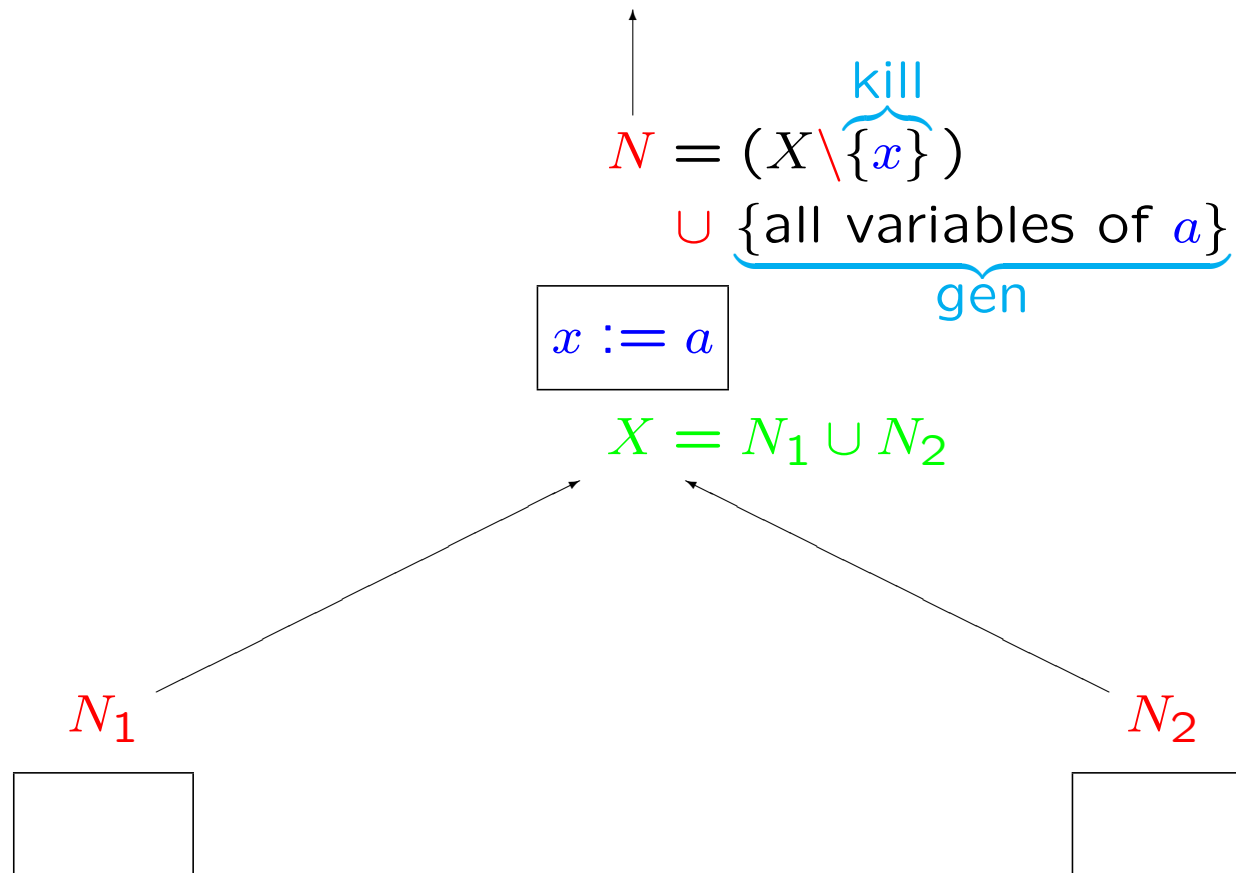


$[x := 2]^1; [y := 4]^2; [x := 1]^3; (\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y * y]^6); [x := z]^7$

The analysis enables a transformation into

$[y := 4]^2; [x := 1]^3; (\text{if } [y > x]^4 \text{ then } [z := y]^5 \text{ else } [z := y * y]^6); [x := z]^7$

# Live Variables Analysis – the basic idea





# Live Variables Analysis

*kill* and *gen* functions

---

$$\textit{kill}_{LV}([x := a]^\ell) = \{x\}$$

$$\textit{kill}_{LV}([\text{skip}]^\ell) = \emptyset$$

$$\textit{kill}_{LV}([b]^\ell) = \emptyset$$

$$\textit{gen}_{LV}([x := a]^\ell) = FV(a)$$

$$\textit{gen}_{LV}([\text{skip}]^\ell) = \emptyset$$

$$\textit{gen}_{LV}([b]^\ell) = FV(b)$$

data flow equations:  $LV^=$

---

$$LV_{exit}(\ell) = \begin{cases} \emptyset & \text{if } \ell \in \textit{final}(S_\star) \\ \bigcup \{LV_{entry}(\ell') \mid (\ell', \ell) \in \textit{flow}^R(S_\star)\} & \text{otherwise} \end{cases}$$

$$LV_{entry}(\ell) = (LV_{exit}(\ell) \setminus \textit{kill}_{LV}(B^\ell)) \cup \textit{gen}_{LV}(B^\ell)$$

where  $B^\ell \in \textit{blocks}(S_\star)$

## Example:

$[x:=2]^1; [y:=4]^2; [x:=1]^3; (\text{if } [y>x]^4 \text{ then } [z:=y]^5 \text{ else } [z:=y*y]^6); [x:=z]^7$

*kill* and *gen* functions:

$\ell$	$\text{kill}_{LV}(\ell)$	$\text{gen}_{LV}(\ell)$
1	$\{x\}$	$\emptyset$
2	$\{y\}$	$\emptyset$
3	$\{x\}$	$\emptyset$
4	$\emptyset$	$\{x, y\}$
5	$\{z\}$	$\{y\}$
6	$\{z\}$	$\{y\}$
7	$\{x\}$	$\{z\}$

## Example (cont.):

$[x:=2]^1; [y:=4]^2; [x:=1]^3; (\text{if } [y>x]^4 \text{ then } [z:=y]^5 \text{ else } [z:=y*y]^6); [x:=z]^7$

Equations:

$LV_{entry}(1) = LV_{exit}(1) \setminus \{x\}$	$LV_{exit}(1) = LV_{entry}(2)$
$LV_{entry}(2) = LV_{exit}(2) \setminus \{y\}$	$LV_{exit}(2) = LV_{entry}(3)$
$LV_{entry}(3) = LV_{exit}(3) \setminus \{x\}$	$LV_{exit}(3) = LV_{entry}(4)$
$LV_{entry}(4) = LV_{exit}(4) \cup \{x, y\}$	$LV_{exit}(4) = LV_{entry}(5) \cup LV_{entry}(6)$
$LV_{entry}(5) = (LV_{exit}(5) \setminus \{z\}) \cup \{y\}$	$LV_{exit}(5) = LV_{entry}(7)$
$LV_{entry}(6) = (LV_{exit}(6) \setminus \{z\}) \cup \{y\}$	$LV_{exit}(6) = LV_{entry}(7)$
$LV_{entry}(7) = \{z\}$	$LV_{exit}(7) = \emptyset$

## Example (cont.):

$[x:=2]^1; [y:=4]^2; [x:=1]^3; (\text{if } [y>x]^4 \text{ then } [z:=y]^5 \text{ else } [z:=y*y]^6); [x:=z]^7$

Smallest solution:

$\ell$	$LV_{entry}(\ell)$	$LV_{exit}(\ell)$
1	$\emptyset$	$\emptyset$
2	$\emptyset$	$\{y\}$
3	$\{y\}$	$\{x, y\}$
4	$\{x, y\}$	$\{y\}$
5	$\{y\}$	$\{z\}$
6	$\{y\}$	$\{z\}$
7	$\{z\}$	$\emptyset$

## Why smallest solution?

$(\text{while } [x > 1]^\ell \text{ do } [\text{skip}]^{\ell'}); [x := x + 1]^{\ell''}$

Equations:

$$LV_{entry}(\ell) = LV_{exit}(\ell) \cup \{x\}$$

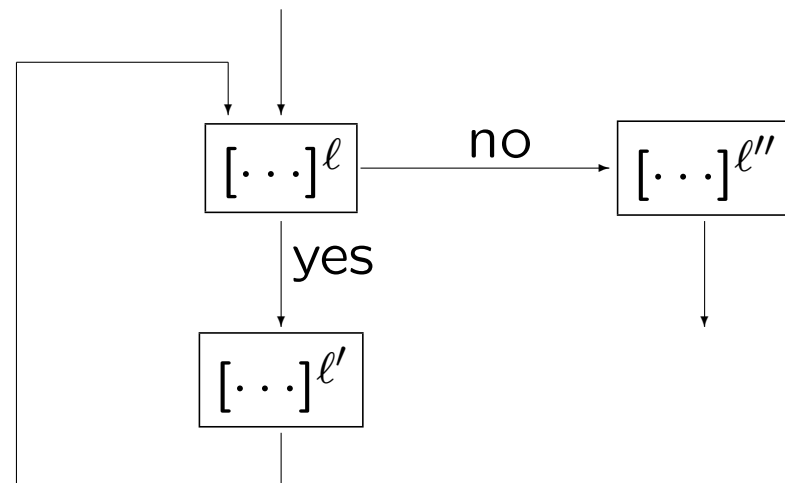
$$LV_{entry}(\ell') = LV_{exit}(\ell')$$

$$LV_{entry}(\ell'') = \{x\}$$

$$LV_{exit}(\ell) = LV_{entry}(\ell') \cup LV_{entry}(\ell'')$$

$$LV_{exit}(\ell') = LV_{entry}(\ell)$$

$$LV_{exit}(\ell'') = \emptyset$$



After some calculations:  $LV_{exit}(\ell) = LV_{exit}(\ell) \cup \{x\}$

Many solutions to this equation: any superset of  $\{x\}$

# Derived Data Flow Information

- *Use-Definition chains* or *ud chains*:

each **use** of a variable is linked to all **assignments** that reach it

$[x:=0]^1; [\textcolor{red}{x}:=3]^2; (\text{if } [z=x]^3 \text{ then } [z:=0]^4 \text{ else } [z:=x]^5); [y:=\textcolor{blue}{x}]^6; [x:=y+z]^7$



- *Definition-Use chains* or *du chains*:

each **assignment** to a variable is linked to all **uses** of it

$[x:=0]^1; [\textcolor{blue}{x}:=3]^2; (\text{if } [z=\textcolor{red}{x}]^3 \text{ then } [z:=0]^4 \text{ else } [z:=\textcolor{red}{x}]^5); [y:=\textcolor{red}{x}]^6; [x:=y+z]^7$



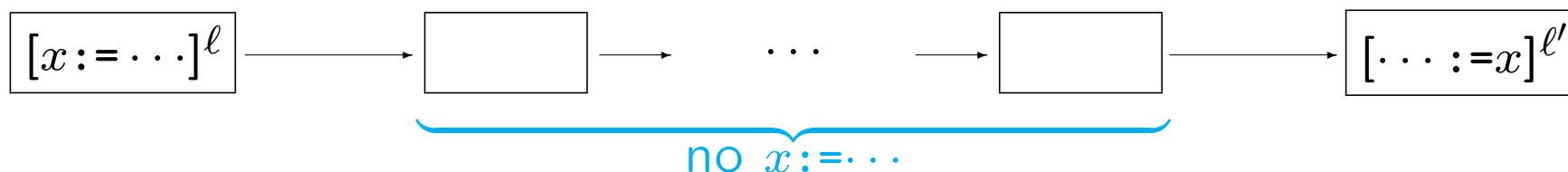
## ud chains

$$ud : \text{Var}_\star \times \text{Lab}_\star \rightarrow \mathcal{P}(\text{Lab}_\star)$$

given by

$$\begin{aligned} ud(x, \ell') &= \{ \ell \mid \text{def}(x, \ell) \wedge \exists \ell'' : (\ell, \ell'') \in \text{flow}(S_\star) \wedge \text{clear}(x, \ell'', \ell') \} \\ &\cup \{ ? \mid \text{clear}(x, \text{init}(S_\star), \ell') \} \end{aligned}$$

where



- $\text{def}(x, \ell)$  means that the block  $\ell$  assigns a value to  $x$
- $\text{clear}(x, \ell, \ell')$  means that none of the blocks on a path from  $\ell$  to  $\ell'$  contains an assignments to  $x$  but that the block  $\ell'$  uses  $x$  (in a test or on the right hand side of an assignment)

## ud chains - an alternative definition

$$\mathbf{UD} : \mathbf{Var}_\star \times \mathbf{Lab}_\star \rightarrow \mathcal{P}(\mathbf{Lab}_\star)$$

is defined by:

$$\mathbf{UD}(x, \ell) = \begin{cases} \{\ell' \mid (x, \ell') \in \mathbf{RD}_{entry}(\ell)\} & \text{if } x \in \mathbf{gen}_{LV}(B^\ell) \\ \emptyset & \text{otherwise} \end{cases}$$

One can show that:

$$\mathbf{ud}(x, \ell) = \mathbf{UD}(x, \ell)$$

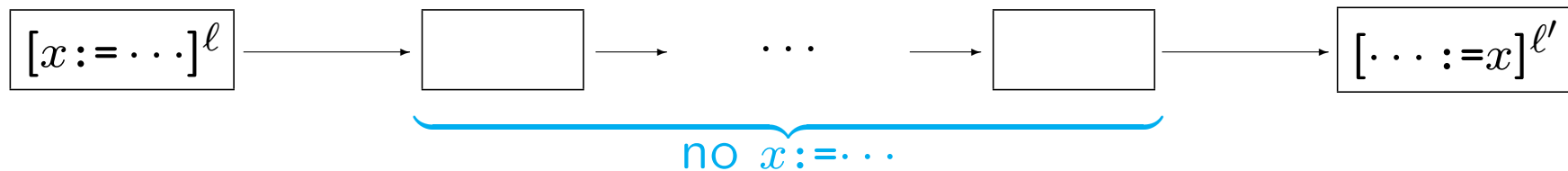


## du chains

$$du : \text{Var}_\star \times \text{Lab}_\star \rightarrow \mathcal{P}(\text{Lab}_\star)$$

given by

$$du(x, \ell) = \begin{cases} \{\ell' \mid \text{def}(x, \ell) \wedge \exists \ell'' : (\ell, \ell'') \in \text{flow}(S_\star) \wedge \text{clear}(x, \ell'', \ell')\} & \text{if } \ell \neq ? \\ \{\ell' \mid \text{clear}(x, \text{init}(S_\star), \ell')\} & \text{if } \ell = ? \end{cases}$$



One can show that:

$$du(x, \ell) = \{\ell' \mid \ell \in ud(x, \ell')\}$$

## Example:

$[x:=0]^1; [x:=3]^2; (\text{if } [z=x]^3 \text{ then } [z:=0]^4 \text{ else } [z:=x]^5); [y:=x]^6; [x:=y+z]^7$

$ud(x, \ell)$	x	y	z	$du(x, \ell)$	x	y	z
1	$\emptyset$	$\emptyset$	$\emptyset$	1	$\emptyset$	$\emptyset$	$\emptyset$
2	$\emptyset$	$\emptyset$	$\emptyset$	2	$\{3, 5, 6\}$	$\emptyset$	$\emptyset$
3	$\{2\}$	$\emptyset$	$\{?\}$	3	$\emptyset$	$\emptyset$	$\emptyset$
4	$\emptyset$	$\emptyset$	$\emptyset$	4	$\emptyset$	$\emptyset$	$\{7\}$
5	$\{2\}$	$\emptyset$	$\emptyset$	5	$\emptyset$	$\emptyset$	$\{7\}$
6	$\{2\}$	$\emptyset$	$\emptyset$	6	$\emptyset$	$\{7\}$	$\emptyset$
7	$\emptyset$	$\{6\}$	$\{4, 5\}$	7	$\emptyset$	$\emptyset$	$\emptyset$
				?	$\emptyset$	$\emptyset$	$\{3\}$

# Theoretical Properties

- Structural Operational Semantics
- Correctness of Live Variables Analysis

# The Semantics

A *state* is a mapping from variables to integers:

$$\sigma \in \mathbf{State} = \mathbf{Var} \rightarrow \mathbf{Z}$$

The semantics of arithmetic and boolean expressions

$$\mathcal{A} : \mathbf{AExp} \rightarrow (\mathbf{State} \rightarrow \mathbf{Z}) \quad (\text{no errors allowed})$$

$$\mathcal{B} : \mathbf{BExp} \rightarrow (\mathbf{State} \rightarrow \mathbf{T}) \quad (\text{no errors allowed})$$

The *transitions* of the semantics are of the form

$$\langle S, \sigma \rangle \rightarrow \sigma' \quad \text{and} \quad \langle S, \sigma \rangle \rightarrow \langle S', \sigma' \rangle$$

# Transitions

$$\langle [x := a]^\ell, \sigma \rangle \rightarrow \sigma[x \mapsto \mathcal{A}[[a]]\sigma]$$

$$\langle [\text{skip}]^\ell, \sigma \rangle \rightarrow \sigma$$

$$\frac{\langle S_1, \sigma \rangle \rightarrow \langle S'_1, \sigma' \rangle}{\langle S_1; S_2, \sigma \rangle \rightarrow \langle S'_1; S_2, \sigma' \rangle}$$

$$\frac{\langle S_1, \sigma \rangle \rightarrow \sigma'}{\langle S_1; S_2, \sigma \rangle \rightarrow \langle S_2, \sigma' \rangle}$$

$$\langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, \sigma \rangle \rightarrow \langle S_1, \sigma \rangle \quad \text{if } \mathcal{B}[[b]]\sigma = \text{true}$$

$$\langle \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2, \sigma \rangle \rightarrow \langle S_2, \sigma \rangle \quad \text{if } \mathcal{B}[[b]]\sigma = \text{false}$$

$$\langle \text{while } [b]^\ell \text{ do } S, \sigma \rangle \rightarrow \langle (S; \text{while } [b]^\ell \text{ do } S), \sigma \rangle \quad \text{if } \mathcal{B}[[b]]\sigma = \text{true}$$

$$\langle \text{while } [b]^\ell \text{ do } S, \sigma \rangle \rightarrow \sigma \quad \text{if } \mathcal{B}[[b]]\sigma = \text{false}$$

## Example:

$\langle [y:=x]^1; [z:=1]^2; \text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{300} \rangle$   
→  $\langle [z:=1]^2; \text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{330} \rangle$   
→  $\langle \text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{331} \rangle$   
→  $\langle [z:=z*y]^4; [y:=y-1]^5;$   
     $\text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{331} \rangle$   
→  $\langle [y:=y-1]^5; \text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{333} \rangle$   
→  $\langle \text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{323} \rangle$   
→  $\langle [z:=z*y]^4; [y:=y-1]^5;$   
     $\text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{323} \rangle$   
→  $\langle [y:=y-1]^5; \text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{326} \rangle$   
→  $\langle \text{while } [y>1]^3 \text{ do } ([z:=z*y]^4; [y:=y-1]^5); [y:=0]^6, \sigma_{316} \rangle$   
→  $\langle [y:=0]^6, \sigma_{316} \rangle$   
→  $\sigma_{306}$

# Equations and Constraints

Equation system  $LV^=(S_\star)$ :

$$LV_{exit}(\ell) = \begin{cases} \emptyset & \text{if } \ell \in \text{final}(S_\star) \\ \bigcup \{LV_{entry}(\ell') \mid (\ell', \ell) \in \text{flow}^R(S_\star)\} & \text{otherwise} \end{cases}$$

$$LV_{entry}(\ell) = (LV_{exit}(\ell) \setminus \text{kill}_{LV}(B^\ell)) \cup \text{gen}_{LV}(B^\ell)$$

where  $B^\ell \in \text{blocks}(S_\star)$

Constraint system  $LV^\subseteq(S_\star)$ :

$$LV_{exit}(\ell) \supseteq \begin{cases} \emptyset & \text{if } \ell \in \text{final}(S_\star) \\ \bigcup \{LV_{entry}(\ell') \mid (\ell', \ell) \in \text{flow}^R(S_\star)\} & \text{otherwise} \end{cases}$$

$$LV_{entry}(\ell) \supseteq (LV_{exit}(\ell) \setminus \text{kill}_{LV}(B^\ell)) \cup \text{gen}_{LV}(B^\ell)$$

where  $B^\ell \in \text{blocks}(S_\star)$

## Lemma

Each solution to the equation system  $LV^=(S_*)$  is also a solution to the constraint system  $LV^{\subseteq}(S_*)$ .

**Proof:** Trivial.

## Lemma

The **least** solution to the equation system  $LV^=(S_*)$  is also the **least** solution to the constraint system  $LV^{\subseteq}(S_*)$ .

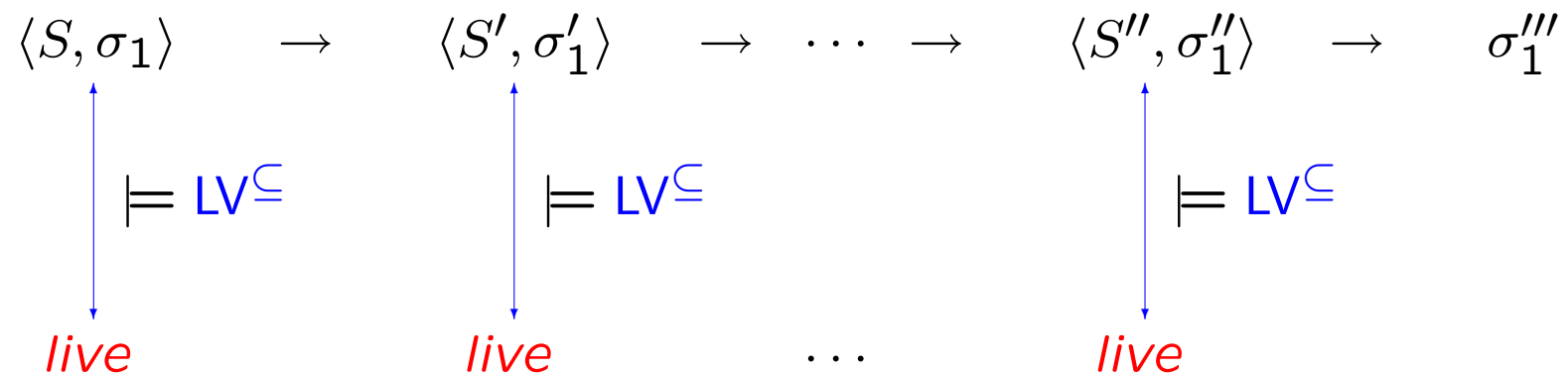
**Proof:** Use Tarski's Theorem.

**Naive Proof:** Proceed by contradiction. Suppose some LHS is strictly greater than the RHS. Replace the LHS by the RHS in the solution. Argue that you still have a solution. This establishes the desired contradiction.



## Lemma

A solution *live* to the constraint system is preserved during computation



**Proof:** requires a lot of machinery — see the book.

## Correctness Relation

$$\sigma_1 \sim_V \sigma_2$$

means that for all practical purposes the two states  $\sigma_1$  and  $\sigma_2$  are equal: only the values of the live variables of  $V$  matters and here the two states are equal.

### Example:

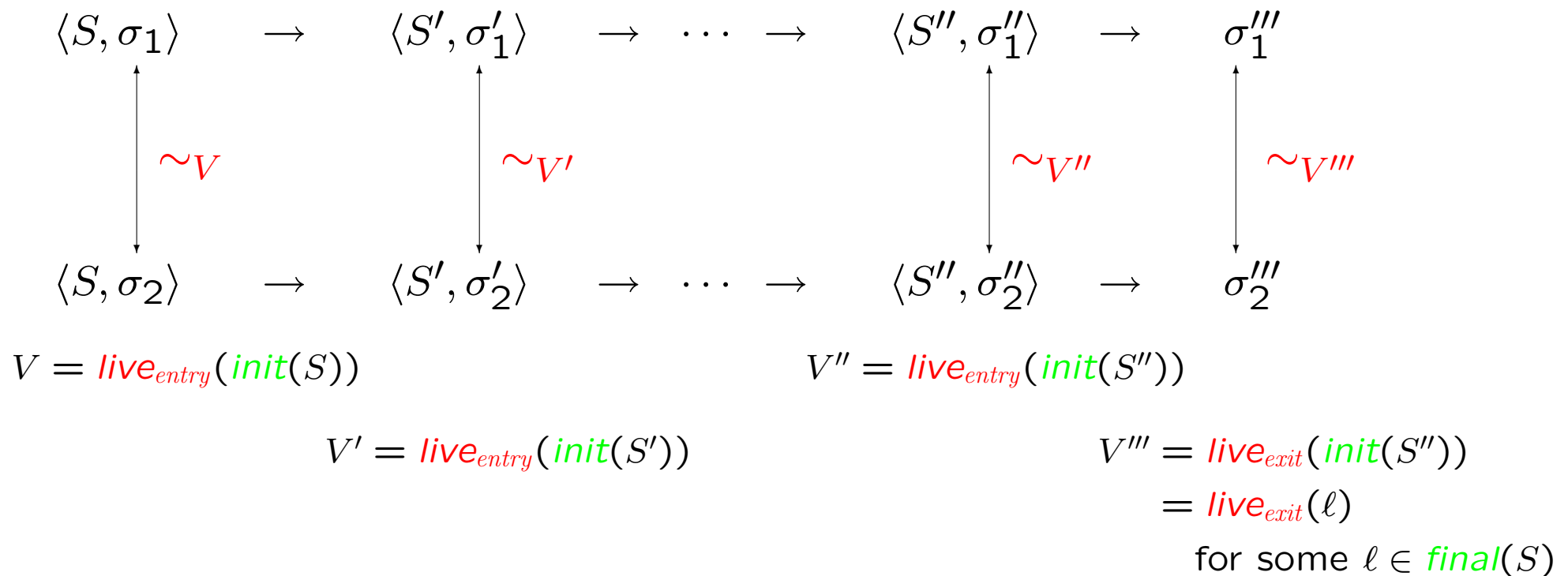
Consider the statement  $[x:=y+z]^\ell$

Let  $V_1 = \{y, z\}$ . Then  $\sigma_1 \sim_{V_1} \sigma_2$  means  $\sigma_1(y) = \sigma_2(y) \wedge \sigma_1(z) = \sigma_2(z)$

Let  $V_2 = \{x\}$ . Then  $\sigma_1 \sim_{V_2} \sigma_2$  means  $\sigma_1(x) = \sigma_2(x)$

# Correctness Theorem

The relation “ $\sim$ ” is *invariant* under computation: the live variables for the initial configuration remain live throughout the computation.



# Monotone Frameworks

- Monotone and Distributive Frameworks
- Instances of Frameworks
- Constant Propagation Analysis

# The Overall Pattern

Each of the four classical analyses take the form

$$\begin{aligned} \textcolor{red}{Analysis}_o(\ell) &= \begin{cases} \iota & \text{if } \ell \in E \\ \sqcup \{ \textcolor{red}{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F \} & \text{otherwise} \end{cases} \\ \textcolor{red}{Analysis}_\bullet(\ell) &= f_\ell(\textcolor{red}{Analysis}_o(\ell)) \end{aligned}$$

where

- $\sqcup$  is  $\cap$  or  $\cup$  (and  $\sqcap$  is  $\cup$  or  $\cap$ ),
- $F$  is either  $\textcolor{green}{flow}(S_\star)$  or  $\textcolor{green}{flow}^R(S_\star)$ ,
- $E$  is  $\{\textcolor{green}{init}(S_\star)\}$  or  $\textcolor{green}{final}(S_\star)$ ,
- $\iota$  specifies the initial or final analysis information, and
- $f_\ell$  is the transfer function associated with  $B^\ell \in \textcolor{green}{blocks}(S_\star)$ .

## The Principle: forward versus backward

- The *forward analyses* have  $F$  to be  $\text{flow}(S_\star)$  and then  $\text{Analysis}_\circ$  concerns entry conditions and  $\text{Analysis}_\bullet$  concerns exit conditions; the equation system presupposes that  $S_\star$  has isolated entries.
- The *backward analyses* have  $F$  to be  $\text{flow}^R(S_\star)$  and then  $\text{Analysis}_\circ$  concerns exit conditions and  $\text{Analysis}_\bullet$  concerns entry conditions; the equation system presupposes that  $S_\star$  has isolated exits.

## The Principle: union versus intersection

- When  $\sqcup$  is  $\cap$  we require the **greatest sets** that solve the equations and we are able to detect properties satisfied by *all execution paths* reaching (or leaving) the entry (or exit) of a label; the analysis is called a **must**-analysis.
- When  $\sqcup$  is  $\cup$  we require the **smallest sets** that solve the equations and we are able to detect properties satisfied by *at least one execution path* to (or from) the entry (or exit) of a label; the analysis is called a **may**-analysis.

# Property Spaces

The *property space*,  $L$ , is used to represent the data flow information, and the *combination operator*,  $\sqcup: \mathcal{P}(L) \rightarrow L$ , is used to combine information from different paths.

- $L$  is a *complete lattice*, that is, a partially ordered set,  $(L, \sqsubseteq)$ , such that each subset,  $Y$ , has a least upper bound,  $\sqcup Y$ .
- $L$  satisfies the *Ascending Chain Condition*; that is, each ascending chain eventually stabilises (meaning that if  $(l_n)_n$  is such that  $l_1 \sqsubseteq l_2 \sqsubseteq l_3 \sqsubseteq \dots$ , then there exists  $n$  such that  $l_n = l_{n+1} = \dots$ ).



## Example: Reaching Definitions

- $L = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$  is partially ordered by subset inclusion so  $\sqsubseteq$  is  $\subseteq$
- the least upper bound operation  $\sqcup$  is  $\cup$  and the least element  $\perp$  is  $\emptyset$
- $L$  satisfies the Ascending Chain Condition because  $\mathbf{Var}_\star \times \mathbf{Lab}_\star$  is finite (unlike  $\mathbf{Var} \times \mathbf{Lab}$ )

## Example: Available Expressions

- $L = \mathcal{P}(\mathbf{AExp}_\star)$  is partially ordered by superset inclusion so  $\sqsubseteq$  is  $\supseteq$
- the least upper bound operation  $\sqcup$  is  $\cap$  and the least element  $\perp$  is  $\mathbf{AExp}_\star$
- $L$  satisfies the Ascending Chain Condition because  $\mathbf{AExp}_\star$  is finite (unlike  $\mathbf{AExp}$ )

# Transfer Functions

The set of transfer functions,  $\mathcal{F}$ , is a set of **monotone functions** over  $L$ , meaning that

$$l \sqsubseteq l' \text{ implies } f_\ell(l) \sqsubseteq f_\ell(l')$$

and furthermore they fulfil the following conditions:

- $\mathcal{F}$  contains *all* the transfer functions  $f_\ell : L \rightarrow L$  in question (for  $\ell \in \mathbf{Lab}_\star$ )
- $\mathcal{F}$  contains the *identity function*
- $\mathcal{F}$  is *closed under composition* of functions

# Frameworks

A *Monotone Framework* consists of:

- a complete lattice,  $L$ , that satisfies the Ascending Chain Condition; we write  $\sqcup$  for the least upper bound operator
- a set  $\mathcal{F}$  of *monotone* functions from  $L$  to  $L$  that contains the identity function and that is closed under function composition

A *Distributive Framework* is a Monotone Framework where additionally all functions  $f$  in  $\mathcal{F}$  are required to be *distributive*:

$$f(l_1 \sqcup l_2) = f(l_1) \sqcup f(l_2)$$

# Instances

An *instance* of a Framework consists of:

- the complete lattice,  $L$ , of the framework
- the space of functions,  $\mathcal{F}$ , of the framework
- a finite flow,  $F$  (typically  $\text{flow}(S_\star)$  or  $\text{flow}^R(S_\star)$ )
- a finite set of *extremal labels*,  $E$  (typically  $\{\text{init}(S_\star)\}$  or  $\text{final}(S_\star)$ )
- an *extremal value*,  $\iota \in L$ , for the extremal labels
- a mapping,  $f.$ , from the labels  $\text{Lab}_\star$  to transfer functions in  $\mathcal{F}$

## Equations of the Instance:

$$\begin{aligned} \textit{Analysis}_\circ(\ell) &= \bigsqcup \{ \textit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F \} \sqcup \iota_E^\ell \\ \text{where } \iota_E^\ell &= \begin{cases} \iota & \text{if } \ell \in E \\ \perp & \text{if } \ell \notin E \end{cases} \end{aligned}$$

$$\textit{Analysis}_\bullet(\ell) = f_\ell(\textit{Analysis}_\circ(\ell))$$

## Constraints of the Instance:

$$\begin{aligned} \textit{Analysis}_\circ(\ell) &\sqsupseteq \bigsqcup \{ \textit{Analysis}_\bullet(\ell') \mid (\ell', \ell) \in F \} \sqcup \iota_E^\ell \\ \text{where } \iota_E^\ell &= \begin{cases} \iota & \text{if } \ell \in E \\ \perp & \text{if } \ell \notin E \end{cases} \end{aligned}$$

$$\textit{Analysis}_\bullet(\ell) \sqsupseteq f_\ell(\textit{Analysis}_\circ(\ell))$$

# The Examples Revisited

	Available Expressions	Reaching Definitions	Very Busy Expressions	Live Variables
$L$	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star \times \mathbf{Lab}_\star)$	$\mathcal{P}(\mathbf{AExp}_\star)$	$\mathcal{P}(\mathbf{Var}_\star)$
$\sqsubseteq$	$\supseteq$	$\subseteq$	$\supseteq$	$\subseteq$
$\sqcup$	$\cap$	$\cup$	$\cap$	$\cup$
$\perp$	$\mathbf{AExp}_\star$	$\emptyset$	$\mathbf{AExp}_\star$	$\emptyset$
$\iota$	$\emptyset$	$\{(x, ?) \mid x \in FV(S_\star)\}$	$\emptyset$	$\emptyset$
$E$	$\{init(S_\star)\}$	$\{init(S_\star)\}$	$final(S_\star)$	$final(S_\star)$
$F$	$flow(S_\star)$	$flow(S_\star)$	$flow^R(S_\star)$	$flow^R(S_\star)$
$\mathcal{F}$	$\{f : L \rightarrow L \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$			
$f_\ell$	$f_\ell(l) = (l \setminus \text{kill}(B^\ell)) \cup \text{gen}(B^\ell)$ where $B^\ell \in \text{blocks}(S_\star)$			

# Bit Vector Frameworks

A *Bit Vector Framework* has

- $L = \mathcal{P}(D)$  for  $D$  finite
- $\mathcal{F} = \{f \mid \exists l_k, l_g : f(l) = (l \setminus l_k) \cup l_g\}$

## Examples:

- Available Expressions
- Live Variables
- Reaching Definitions
- Very Busy Expressions



**Lemma:** Bit Vector Frameworks are always Distributive Frameworks

**Proof**

$$\begin{aligned}
 f(l_1 \sqcup l_2) &= \begin{cases} f(l_1 \cup l_2) \\ f(l_1 \cap l_2) \end{cases} &= \begin{cases} ((l_1 \cup l_2) \setminus l_k) \cup l_g \\ ((l_1 \cap l_2) \setminus l_k) \cup l_g \end{cases} \\
 &= \begin{cases} ((l_1 \setminus l_k) \cup (l_2 \setminus l_k)) \cup l_g \\ ((l_1 \setminus l_k) \cap (l_2 \setminus l_k)) \cup l_g \end{cases} &= \begin{cases} ((l_1 \setminus l_k) \cup l_g) \cup ((l_2 \setminus l_k) \cup l_g) \\ ((l_1 \setminus l_k) \cup l_g) \cap ((l_2 \setminus l_k) \cup l_g) \end{cases} \\
 &= \begin{cases} f(l_1) \cup f(l_2) \\ f(l_1) \cap f(l_2) \end{cases} &= f(l_1) \sqcup f(l_2)
 \end{aligned}$$

- $id(l) = (l \setminus \emptyset) \cup \emptyset$
- $f_2(f_1(l)) = (((l \setminus l_k^1) \cup l_g^1) \setminus l_k^2) \cup l_g^2 = (l \setminus (l_k^1 \cup l_k^2)) \cup ((l_g^1 \setminus l_k^2) \cup l_g^2)$
- monotonicity follows from distributivity
- $\mathcal{P}(D)$  satisfies the Ascending Chain Condition because  $D$  is finite

# The Constant Propagation Framework

An example of a Monotone Framework that is **not** a Distributive Framework

The aim of the *Constant Propagation Analysis* is to determine

For each program point, whether or not a variable has a constant value whenever execution reaches that point.

## Example:

$[x:=6]^1; [y:=3]^2; \text{while } [x > y]^3 \text{ do } ([x:=x-1]^4; [z:=y*y]^6)$

The analysis enables a transformation into

$[x:=6]^1; [y:=3]^2; \text{while } [x > 3]^3 \text{ do } ([x:=x-1]^4; [z:=9]^6)$

## Elements of $L$

$$\widehat{\text{State}}_{\text{CP}} = ((\text{Var}_\star \rightarrow \mathbf{Z}^\top)_\perp, \sqsubseteq)$$

Idea:

- $\perp$  is the least element: no information is available
- $\hat{\sigma} \in \text{Var}_\star \rightarrow \mathbf{Z}^\top$  specifies for each variable whether it is constant:
  - $\hat{\sigma}(x) \in \mathbf{Z}$ :  $x$  is constant and the value is  $\hat{\sigma}(x)$
  - $\hat{\sigma}(x) = \top$ :  $x$  might not be constant

## Partial Ordering on $L$

The partial ordering  $\sqsubseteq$  on  $(\mathbf{Var}_\star \rightarrow \mathbf{Z}^\top)_\perp$  is defined by

$$\forall \hat{\sigma} \in (\mathbf{Var}_\star \rightarrow \mathbf{Z}^\top)_\perp : \quad \perp \sqsubseteq \hat{\sigma}$$

$$\forall \hat{\sigma}_1, \hat{\sigma}_2 \in \mathbf{Var}_\star \rightarrow \mathbf{Z}^\top : \quad \hat{\sigma}_1 \sqsubseteq \hat{\sigma}_2 \quad \text{iff} \quad \forall x : \hat{\sigma}_1(x) \sqsubseteq \hat{\sigma}_2(x)$$

where  $\mathbf{Z}^\top = \mathbf{Z} \cup \{\top\}$  is partially ordered as follows:

$$\forall z \in \mathbf{Z}^\top : z \sqsubseteq \top$$

$$\forall z_1, z_2 \in \mathbf{Z} : (z_1 \sqsubseteq z_2) \Leftrightarrow (z_1 = z_2)$$

## Transfer Functions in $\mathcal{F}$

$$\mathcal{F}_{\text{CP}} = \{f \mid f \text{ is a monotone function on } \widehat{\text{State}}_{\text{CP}}\}$$

### Lemma

Constant Propagation as defined by  $\widehat{\text{State}}_{\text{CP}}$  and  $\mathcal{F}_{\text{CP}}$  is a Monotone Framework

# Instances

Constant Propagation is a forward analysis, so for the program  $S_\star$ :

- the flow,  $F$ , is  $\text{flow}(S_\star)$ ,
- the extremal labels,  $E$ , is  $\{\text{init}(S_\star)\}$ ,
- the extremal value,  $\iota_{\text{CP}}$ , is  $\lambda x. \top$ , and
- the mapping,  $f^{\text{CP}}$ , of labels to transfer functions is as shown next

# Constant Propagation Analysis

$$\begin{array}{c}
 \mathcal{A}_{\text{CP}} : \mathbf{AExp} \rightarrow (\widehat{\text{State}}_{\text{CP}} \rightarrow \mathbf{Z}_{\perp}^{\top}) \\
 \hline
 \mathcal{A}_{\text{CP}}[[x]]\hat{\sigma} = \begin{cases} \perp & \text{if } \hat{\sigma} = \perp \\ \hat{\sigma}(x) & \text{otherwise} \end{cases} \\
 \mathcal{A}_{\text{CP}}[[n]]\hat{\sigma} = \begin{cases} \perp & \text{if } \hat{\sigma} = \perp \\ n & \text{otherwise} \end{cases} \\
 \mathcal{A}_{\text{CP}}[[a_1 \text{ op}_a a_2]]\hat{\sigma} = \mathcal{A}_{\text{CP}}[[a_1]]\hat{\sigma} \widehat{\text{op}}_a \mathcal{A}_{\text{CP}}[[a_2]]\hat{\sigma}
 \end{array}$$

transfer functions:  $f_{\ell}^{\text{CP}}$

$$\begin{array}{lcl}
 [x := a]^{\ell} : & f_{\ell}^{\text{CP}}(\hat{\sigma}) & = \begin{cases} \perp & \text{if } \hat{\sigma} = \perp \\ \hat{\sigma}[x \mapsto \mathcal{A}_{\text{CP}}[[a]]\hat{\sigma}] & \text{otherwise} \end{cases} \\
 [\text{skip}]^{\ell} : & f_{\ell}^{\text{CP}}(\hat{\sigma}) & = \hat{\sigma} \\
 [b]^{\ell} : & f_{\ell}^{\text{CP}}(\hat{\sigma}) & = \hat{\sigma}
 \end{array}$$

## Lemma

Constant Propagation is **not** a Distributive Framework

## Proof

Consider the transfer function  $f_\ell^{\text{CP}}$  for  $[y := x * x]^\ell$

Let  $\hat{\sigma}_1$  and  $\hat{\sigma}_2$  be such that  $\hat{\sigma}_1(x) = 1$  and  $\hat{\sigma}_2(x) = -1$

Then  $\hat{\sigma}_1 \sqcup \hat{\sigma}_2$  maps  $x$  to  $\top$  —  $f_\ell^{\text{CP}}(\hat{\sigma}_1 \sqcup \hat{\sigma}_2)$  maps  $y$  to  $\top$

Both  $f_\ell^{\text{CP}}(\hat{\sigma}_1)$  and  $f_\ell^{\text{CP}}(\hat{\sigma}_2)$  map  $y$  to 1 —  $f_\ell^{\text{CP}}(\hat{\sigma}_1) \sqcup f_\ell^{\text{CP}}(\hat{\sigma}_2)$  maps  $y$  to 1



# Equation Solving

- The MFP solution — “Maximum” (actually least) Fixed Point
  - Worklist algorithm for Monotone Frameworks
- The MOP solution — “Meet” (actually join) Over all Paths

# The MFP Solution

– Idea: iterate until stabilisation.

## Worklist Algorithm

**Input:** An instance  $(L, \mathcal{F}, F, E, \iota, f.)$  of a Monotone Framework

**Output:** The MFP Solution:  $MFP_{\circ}, MFP_{\bullet}$

**Data structures:**

- **Analysis:** the current analysis result for block entries (or exits)
- The worklist **W**: a list of pairs  $(\ell, \ell')$  indicating that the current analysis result has changed at the entry (or exit) to the block  $\ell$  and hence the entry (or exit) information must be recomputed for  $\ell'$

# Worklist Algorithm

## Step 1 Initialisation (of $W$ and Analysis)

$W := \text{nil};$   
for all  $(\ell, \ell')$  in  $F$  do  $W := \text{cons}((\ell, \ell'), W);$   
for all  $\ell$  in  $F$  or  $E$  do  
    if  $\ell \in E$  then  $\text{Analysis}[\ell] := \iota$  else  $\text{Analysis}[\ell] := \perp_L;$

## Step 2 Iteration (updating $W$ and Analysis)

while  $W \neq \text{nil}$  do  
     $\ell := \text{fst}(\text{head}(W)); \ell' = \text{snd}(\text{head}(W)); W := \text{tail}(W);$   
    if  $f_\ell(\text{Analysis}[\ell]) \not\sqsubseteq \text{Analysis}[\ell']$  then  
         $\text{Analysis}[\ell'] := \text{Analysis}[\ell'] \sqcup f_\ell(\text{Analysis}[\ell]);$   
        for all  $\ell''$  with  $(\ell', \ell'')$  in  $F$  do  $W := \text{cons}((\ell', \ell''), W);$

## Step 3 Presenting the result ( $MFP_\circ$ and $MFP_\bullet$ )

for all  $\ell$  in  $F$  or  $E$  do  
     $MFP_\circ(\ell) := \text{Analysis}[\ell];$   
     $MFP_\bullet(\ell) := f_\ell(\text{Analysis}[\ell])$

## Correctness

The worklist algorithm always terminates and it computes the least (or MFP) solution to the instance given as input.

## Complexity

Suppose that  $E$  and  $F$  contain at most  $b \geq 1$  distinct labels, that  $F$  contains at most  $e \geq b$  pairs, and that  $L$  has finite height at most  $h \geq 1$ .

Count as basic operations the applications of  $f_\ell$ , applications of  $\sqcup$ , or updates of Analysis.

Then there will be at most  $O(e \cdot h)$  basic operations.

**Example:** Reaching Definitions (assuming unique labels):

$O(b^2)$  where  $b$  is size of program:  $O(h) = O(b)$  and  $O(e) = O(b)$ .

# The MOP Solution

- Idea: propagate analysis information along **paths**.

## Paths

The paths up to **but not including**  $\ell$ :

$$\text{path}_\circ(\ell) = \{[\ell_1, \dots, \ell_{n-1}] \mid n \geq 1 \wedge \forall i < n : (\ell_i, \ell_{i+1}) \in F \wedge \ell_n = \ell \wedge \ell_1 \in E\}$$

The paths up to **and including**  $\ell$ :

$$\text{path}_\bullet(\ell) = \{[\ell_1, \dots, \ell_n] \mid n \geq 1 \wedge \forall i < n : (\ell_i, \ell_{i+1}) \in F \wedge \ell_n = \ell \wedge \ell_1 \in E\}$$

Transfer functions for a path  $\vec{\ell} = [\ell_1, \dots, \ell_n]$ :

$$f_{\vec{\ell}} = f_{\ell_n} \circ \dots \circ f_{\ell_1} \circ id$$

# The MOP Solution

The solution up to but not including  $\ell$ :

$$MOP_{\circ}(\ell) = \bigsqcup \{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in path_{\circ}(\ell)\}$$

The solution up to and including  $\ell$ :

$$MOP_{\bullet}(\ell) = \bigsqcup \{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in path_{\bullet}(\ell)\}$$

## Precision of the MOP versus MFP solutions

The MFP solution safely approximates the MOP solution:  $MFP \sqsupseteq MOP$  (“because”  $f(x \sqcup y) \sqsupseteq f(x) \sqcup f(y)$  when  $f$  is monotone).

For Distributive Frameworks the MFP and MOP solutions are equal:  $MFP = MOP$  (“because”  $f(x \sqcup y) = f(x) \sqcup f(y)$  when  $f$  is distributive).

## Lemma

Consider the MFP and MOP solutions to an instance  $(L, \mathcal{F}, F, B, \iota, f.)$  of a Monotone Framework; then:

$$MFP_{\circ} \sqsupseteq MOP_{\circ} \text{ and } MFP_{\bullet} \sqsupseteq MOP_{\bullet}$$

If the framework is distributive and if  $path_{\circ}(\ell) \neq \emptyset$  for all  $\ell$  in  $E$  and  $F$  then:

$$MFP_{\circ} = MOP_{\circ} \text{ and } MFP_{\bullet} = MOP_{\bullet}$$

## Decidability of MOP and MFP

The MFP solution is always computable (meaning that it is decidable) because of the **Ascending Chain Condition**.

The MOP solution is often uncomputable (meaning that it is undecidable): the existence of a general algorithm for the MOP solution would imply the decidability of the *Modified Post Correspondence Problem*, which is known to be undecidable.



# Lemma

The MOP solution for Constant Propagation is undecidable.

**Proof:** Let  $u_1, \dots, u_n$  and  $v_1, \dots, v_n$  be strings over the alphabet  $\{1, \dots, 9\}$ ; let  $|u|$  denote the length of  $u$ ; let  $\llbracket u \rrbracket$  be the natural number denoted.

The Modified Post Correspondence Problem is to determine whether or not  $u_{i_1} \dots u_{i_m} = v_{i_1} \dots v_{i_n}$  for some sequence  $i_1, \dots, i_m$  with  $i_1 = 1$ .

```
x :=  $\llbracket u_1 \rrbracket$ ; y :=  $\llbracket v_1 \rrbracket$ ;
while [...] do
  (if [...] then x := x * 10|u1| +  $\llbracket u_1 \rrbracket$ ; y := y * 10|v1| +  $\llbracket v_1 \rrbracket$  else
  :
  if [...] then x := x * 10|un| +  $\llbracket u_n \rrbracket$ ; y := y * 10|vn| +  $\llbracket v_n \rrbracket$  else skip)
[z := abs((x-y)*(x-y))]ℓ
```

Then **MOP**.( $\ell$ ) will map  $z$  to 1 if and only if the Modified Post Correspondence Problem has no solution. This is undecidable.

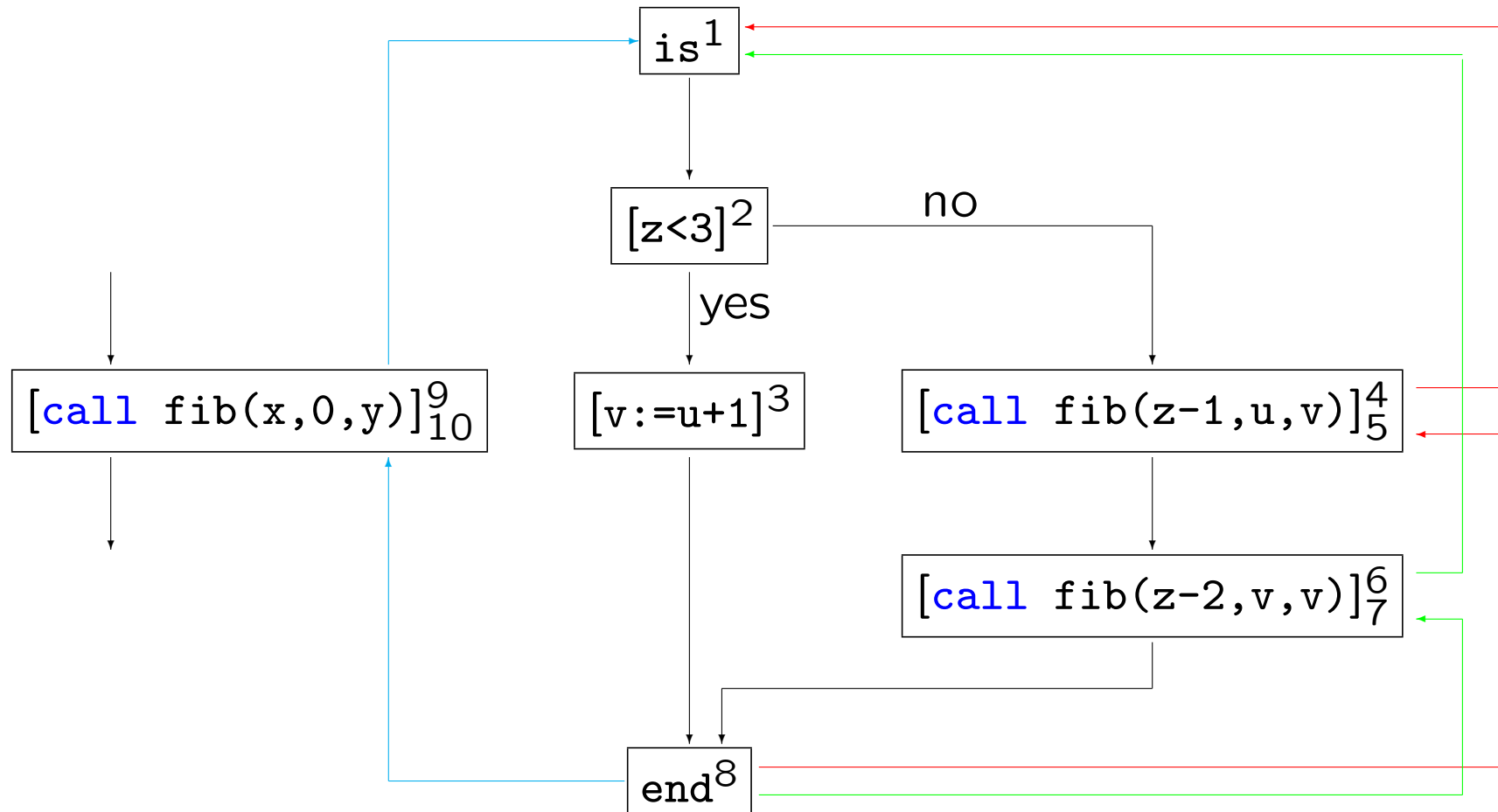
# Interprocedural Analysis

- The problem
- MVP: “Meet” over Valid Paths
- Making context explicit
- Context based on call-strings
- Context based on assumption sets

(A restricted treatment; see the book for a more general treatment.)

# The Problem: match entries with exits

```
proc fib(val z, u; res v)
```



# Preliminaries

## Syntax for procedures

Programs:  $P_{\star} = \text{begin } D_{\star} \ S_{\star} \ \text{end}$

Declarations:  $D ::= D; D \mid \text{proc } p(\text{val } x; \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x}$

Statements:  $S ::= \dots \mid [\text{call } p(a, z)]^{\ell_c}_{\ell_r}$

## Example:

```
begin  proc fib(val z, u; res v) is1
      if [z<3]2 then [v:=u+1]3
      else ([call fib(z-1,u,v)]45; [call fib(z-2,v,v)]67)
      end8;
      [call fib(x,0,y)]910
end
```

# Flow graphs for procedure calls

$$\text{init}([\text{call } p(a, z)]_{\ell_r}^{\ell_c}) = \ell_c$$

$$\text{final}([\text{call } p(a, z)]_{\ell_r}^{\ell_c}) = \{\ell_r\}$$

$$\text{blocks}([\text{call } p(a, z)]_{\ell_r}^{\ell_c}) = \{[\text{call } p(a, z)]_{\ell_r}^{\ell_c}\}$$

$$\text{labels}([\text{call } p(a, z)]_{\ell_r}^{\ell_c}) = \{\ell_c, \ell_r\}$$

$$\text{flow}([\text{call } p(a, z)]_{\ell_r}^{\ell_c}) = \{(\ell_c; \ell_n), (\ell_x; \ell_r)\}$$

if  $\text{proc } p(\text{val } x; \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x}$  is in  $D_\star$

- $(\ell_c; \ell_n)$  is the flow corresponding to *calling* a procedure at  $\ell_c$  and entering the procedure body at  $\ell_n$ , and
- $(\ell_x; \ell_r)$  is the flow corresponding to exiting a procedure body at  $\ell_x$  and *returning* to the call at  $\ell_r$ .

## Flow graphs for procedure declarations

For each procedure declaration  $\text{proc } p(\text{val } x; \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x}$  of  $D_\star$ :

$$\begin{aligned} \text{init}(p) &= \ell_n \\ \text{final}(p) &= \{\ell_x\} \\ \text{blocks}(p) &= \{\text{is}^{\ell_n}, \text{end}^{\ell_x}\} \cup \text{blocks}(S) \\ \text{labels}(p) &= \{\ell_n, \ell_x\} \cup \text{labels}(S) \\ \text{flow}(p) &= \{(\ell_n, \text{init}(S))\} \cup \text{flow}(S) \cup \{(\ell, \ell_x) \mid \ell \in \text{final}(S)\} \end{aligned}$$

# Flow graphs for programs

For the program  $P_\star = \text{begin } D_\star \ S_\star \ \text{end}$ :

$$\text{init}_\star = \text{init}(S_\star)$$

$$\text{final}_\star = \text{final}(S_\star)$$

$$\text{blocks}_\star = \bigcup \{ \text{blocks}(\textcolor{red}{p}) \mid \text{proc } \textcolor{red}{p}(\text{val } x; \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x} \text{ is in } D_\star \} \\ \cup \text{blocks}(S_\star)$$

$$\text{labels}_\star = \bigcup \{ \text{labels}(\textcolor{red}{p}) \mid \text{proc } \textcolor{red}{p}(\text{val } x; \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x} \text{ is in } D_\star \} \\ \cup \text{labels}(S_\star)$$

$$\text{flow}_\star = \bigcup \{ \text{flow}(\textcolor{red}{p}) \mid \text{proc } \textcolor{red}{p}(\text{val } x; \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x} \text{ is in } D_\star \} \\ \cup \text{flow}(S_\star)$$

$$\text{interflow}_\star = \{ (\ell_c, \ell_n, \ell_x, \ell_r) \mid \text{proc } \textcolor{red}{p}(\text{val } x; \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x} \text{ is in } D_\star \\ \text{and } [\text{call } \textcolor{red}{p}(a, z)]_{\ell_r}^{\ell_c} \text{ is in } S_\star \}$$

## Example:

```
begin  proc fib(val z, u; res v) is1
        if [z<3]2 then [v:=u+1]3
        else ([call fib(z-1,u,v)]45; [call fib(z-2,v,v)]67)
        end8;
        [call fib(x,0,y)]910
end
```

We have

$$\begin{aligned} \text{flow}_\star &= \{(1, 2), (2, 3), (3, 8), \\ &\quad (2, 4), (4; 1), (8; 5), (5, 6), (6; 1), (8; 7), (7, 8), \\ &\quad (9; 1), (8; 10)\} \end{aligned}$$

$$\text{interflow}_\star = \{(\textcolor{red}{9}, \textcolor{green}{1}, \textcolor{green}{8}, \textcolor{red}{10}), (\textcolor{red}{4}, \textcolor{green}{1}, \textcolor{green}{8}, \textcolor{red}{5}), (\textcolor{red}{6}, \textcolor{green}{1}, \textcolor{green}{8}, \textcolor{red}{7})\}$$

and  $\text{init}_\star = 9$  and  $\text{final}_\star = \{10\}$ .



## A naive formulation

Treat the three kinds of flow in the same way:

flow	treat as
$(\ell_1, \ell_2)$	$(\ell_1, \ell_2)$
$(\ell_c; \ell_n)$	$(\ell_c, \ell_n)$
$(\ell_x; \ell_r)$	$(\ell_x, \ell_r)$

Equation system:

$$A_{\bullet}(\ell) = f_{\ell}(A_{\circ}(\ell))$$

$$A_{\circ}(\ell) = \bigsqcup \{A_{\bullet}(\ell') \mid (\ell', \ell) \in F \text{ or } (\ell', \ell) \in F \text{ or } (\ell', \ell) \in F\} \sqcup \iota_E^{\ell}$$

But there is no matching between entries and exits.

# MVP: “Meet” over Valid Paths

## Complete Paths

We need to match procedure entries and exits:

A *complete path* from  $\ell_1$  to  $\ell_2$  in  $P_\star$  has proper nesting of procedure entries and exits; and a procedure returns to the point where it was called:

$$\begin{array}{ll} CP_{\ell_1, \ell_2} \longrightarrow \ell_1 & \text{whenever } \ell_1 = \ell_2 \\ CP_{\ell_1, \ell_3} \longrightarrow \ell_1, CP_{\ell_2, \ell_3} & \text{whenever } (\ell_1, \ell_2) \in \text{flow}_\star \\ CP_{\ell_c, \ell} \longrightarrow \ell_c, CP_{\ell_n, \ell_x}, CP_{\ell_r, \ell} & \text{whenever } P_\star \text{ contains } [\text{call } p(a, z)]_{\ell_r}^{\ell_c} \\ & \text{and } \text{proc } p(\text{val } x; \text{res } y) \text{ is } \ell_n S \text{ end } \ell_x \end{array}$$

More generally: whenever  $(\ell_c, \ell_n, \ell_x, \ell_r)$  is an element of  $\text{interflow}_\star$  (or  $\text{interflow}_\star^R$  for backward analyses); see the book.

# Valid Paths

A *valid path* starts at the entry node  $init_\star$  of  $P_\star$ , all the procedure exits match the procedure entries but some procedures might be entered but not yet exited:

$VP_\star \longrightarrow VP_{init_\star, \ell}$	whenever $\ell \in \mathbf{Lab}_\star$
$VP_{\ell_1, \ell_2} \longrightarrow \ell_1$	whenever $\ell_1 = \ell_2$
$VP_{\ell_1, \ell_3} \longrightarrow \ell_1, VP_{\ell_2, \ell_3}$	whenever $(\ell_1, \ell_2) \in flow_\star$
$VP_{\ell_c, \ell} \longrightarrow \ell_c, CP_{\ell_n, \ell_x}, VP_{\ell_r, \ell}$	whenever $P_\star$ contains $[call\ p(a, z)]_{\ell_r}^{\ell_c}$ and $proc\ p(val\ x; res\ y)\ is^{\ell_n}\ S\ end^{\ell_x}$
$VP_{\ell_c, \ell} \longrightarrow \ell_c, VP_{\ell_n, \ell}$	whenever $P_\star$ contains $[call\ p(a, z)]_{\ell_r}^{\ell_c}$ and $proc\ p(val\ x; res\ y)\ is^{\ell_n}\ S\ end^{\ell_x}$

## The MVP solution

$$MVP_{\circ}(\ell) = \bigsqcup \{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in vpath_{\circ}(\ell)\}$$

$$MVP_{\bullet}(\ell) = \bigsqcup \{f_{\vec{\ell}}(\iota) \mid \vec{\ell} \in vpath_{\bullet}(\ell)\}$$

where

$$vpath_{\circ}(\ell) = \{[\ell_1, \dots, \ell_{n-1}] \mid n \geq 1 \wedge \ell_n = \ell \wedge [\ell_1, \dots, \ell_n] \text{ is a valid path}\}$$

$$vpath_{\bullet}(\ell) = \{[\ell_1, \dots, \ell_n] \mid n \geq 1 \wedge \ell_n = \ell \wedge [\ell_1, \dots, \ell_n] \text{ is a valid path}\}$$

The MVP solution may be undecidable for lattices satisfying the Ascending Chain Condition, just as was the case for the MOP solution.

# Making Context Explicit

Starting point: an instance  $(L, \mathcal{F}, F, E, \iota, f.)$  of a Monotone Framework

- the analysis is **forwards**, i.e.  $F = \text{flow}_\star$  and  $E = \{\text{init}_\star\}$ ;
- the complete lattice is a powerset, i.e.  $L = \mathcal{P}(D)$ ;
- the transfer functions in  $\mathcal{F}$  are completely additive; and
- each  $f_\ell$  is given by  $f_\ell(Y) = \bigcup \{ \phi_\ell(d) \mid d \in Y \}$  where  $\phi_\ell : D \rightarrow \mathcal{P}(D)$ .

(A restricted treatment; see the book for a more general treatment.)

# An embellished monotone framework

- $L' = \mathcal{P}(\Delta \times D)$ ;
- the transfer functions in  $\mathcal{F}'$  are completely additive; and
- each  $f'_\ell$  is given by  $f'_\ell(Z) = \bigcup \{ \{\delta\} \times \phi_\ell(d) \mid (\delta, d) \in Z \}$ .

Ignoring procedures, the data flow equations will take the form:

$$A_\bullet(\ell) = f'_\ell(A_\circ(\ell))$$

for all labels that do not label a procedure call

$$A_\circ(\ell) = \bigsqcup \{ A_\bullet(\ell') \mid (\ell', \ell) \in F \text{ or } (\ell'; \ell) \in F \} \sqcup \iota_E^\ell$$

for all labels (including those that label procedure calls)

## Example:

Detection of Signs Analysis as a Monotone Framework:

$(L_{\text{sign}}, \mathcal{F}_{\text{sign}}, F, E, \iota_{\text{sign}}, f^{\text{sign}})$  where  $\mathbf{Sign} = \{-, 0, +\}$  and

$$L_{\text{sign}} = \mathcal{P}(\mathbf{Var}_{\star} \rightarrow \mathbf{Sign})$$

The transfer function  $f_{\ell}^{\text{sign}}$  associated with the assignment  $[x := a]^{\ell}$  is

$$f_{\ell}^{\text{sign}}(Y) = \bigcup \{ \phi_{\ell}^{\text{sign}}(\sigma^{\text{sign}}) \mid \sigma^{\text{sign}} \in Y \}$$

where  $Y \subseteq \mathbf{Var}_{\star} \rightarrow \mathbf{Sign}$  and

$$\phi_{\ell}^{\text{sign}}(\sigma^{\text{sign}}) = \{ \sigma^{\text{sign}}[x \mapsto s] \mid s \in \mathcal{A}_{\text{sign}}[a](\sigma^{\text{sign}}) \}$$

## Example (cont.):

Detection of Signs Analysis as an embellished monotone framework

$$L'_{\text{sign}} = \mathcal{P}(\Delta \times (\text{Var}_\star \rightarrow \text{Sign}))$$

The transfer function associated with  $[x := a]^\ell$  will now be:

$$f_\ell^{\text{sign}'}(Z) = \bigcup \{ \{\delta\} \times \phi_\ell^{\text{sign}}(\sigma^{\text{sign}}) \mid (\delta, \sigma^{\text{sign}}) \in Z \}$$



# Transfer functions for procedure declarations

Procedure declarations

$$\text{proc } p(\text{val } x; \text{res } y) \text{ is}^{\ell_n} S \text{ end}^{\ell_x}$$

have two transfer functions, one for entry and one for exit:

$$f_{\ell_n}, f_{\ell_x} : \mathcal{P}(\Delta \times D) \rightarrow \mathcal{P}(\Delta \times D)$$

For simplicity we take both to be the identity function (thus incorporating procedure entry as part of procedure call, and procedure exit as part of procedure return).

# Transfer functions for procedure calls

Procedure calls  $[\text{call } p(a, z)]_{\ell_r}^{\ell_c}$  have two transfer functions:

For the *procedure call*

$$f_{\ell_c}^1 : \mathcal{P}(\Delta \times D) \rightarrow \mathcal{P}(\Delta \times D)$$

and it is used in the equation:

$$A_{\bullet}(\ell_c) = f_{\ell_c}^1(A_o(\ell_c)) \text{ for all procedure calls } [\text{call } p(a, z)]_{\ell_r}^{\ell_c}$$

For the *procedure return*

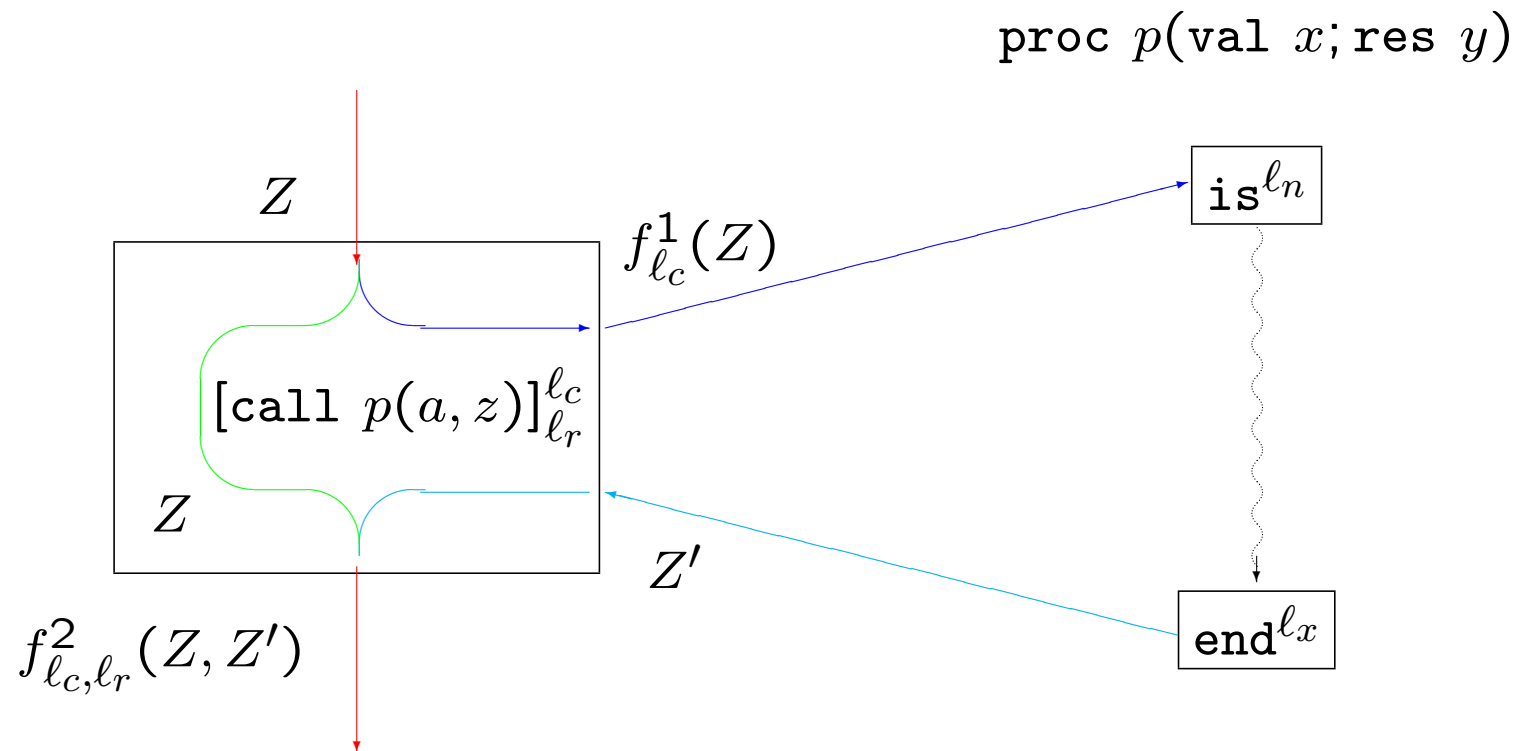
$$f_{\ell_c, \ell_r}^2 : \boxed{\mathcal{P}(\Delta \times D)} \times \mathcal{P}(\Delta \times D) \rightarrow \mathcal{P}(\Delta \times D)$$

and it is used in the equation:

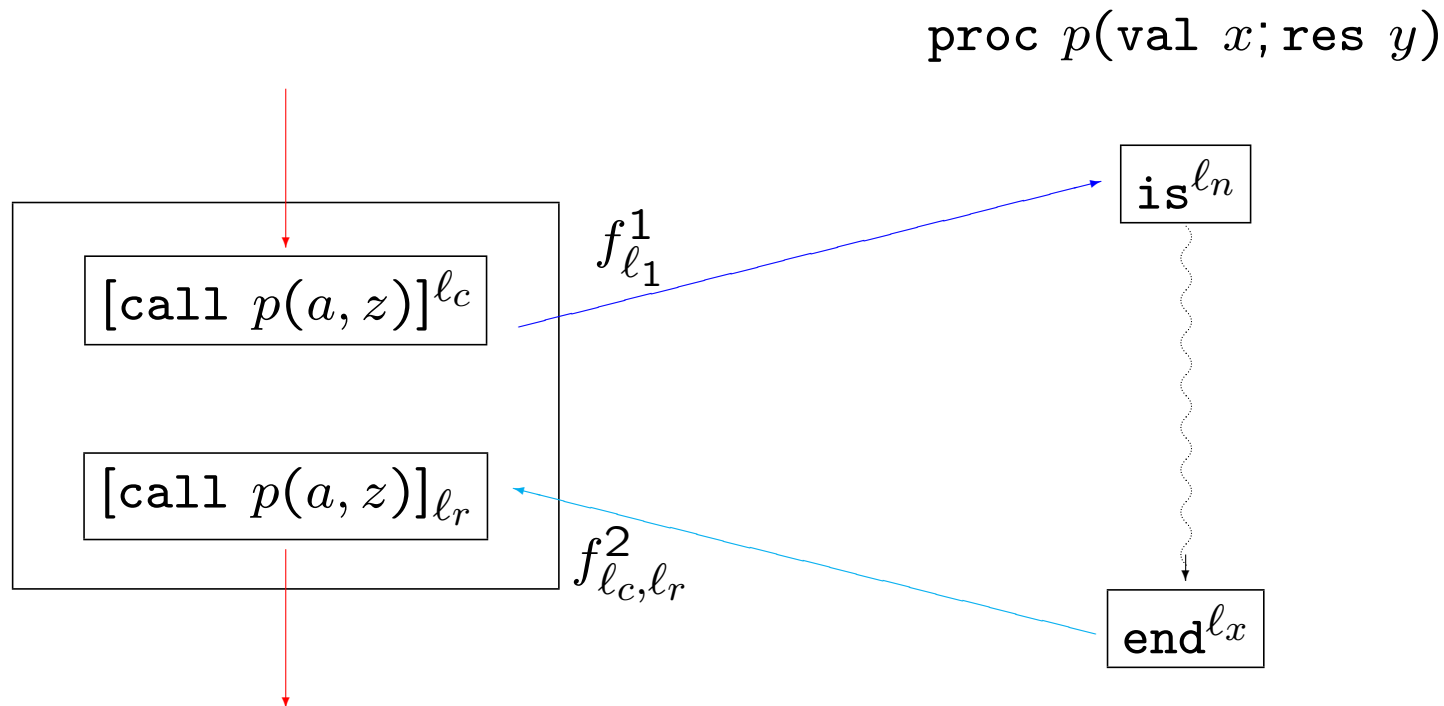
$$A_{\bullet}(\ell_r) = f_{\ell_c, \ell_r}^2(\boxed{A_o(\ell_c)}, A_o(\ell_r)) \text{ for all procedure calls } [\text{call } p(a, z)]_{\ell_r}^{\ell_c}$$

(Note that  $A_o(\ell_r)$  will equal  $A_{\bullet}(\ell_x)$  for the relevant procedure exit.)

# Procedure calls and returns



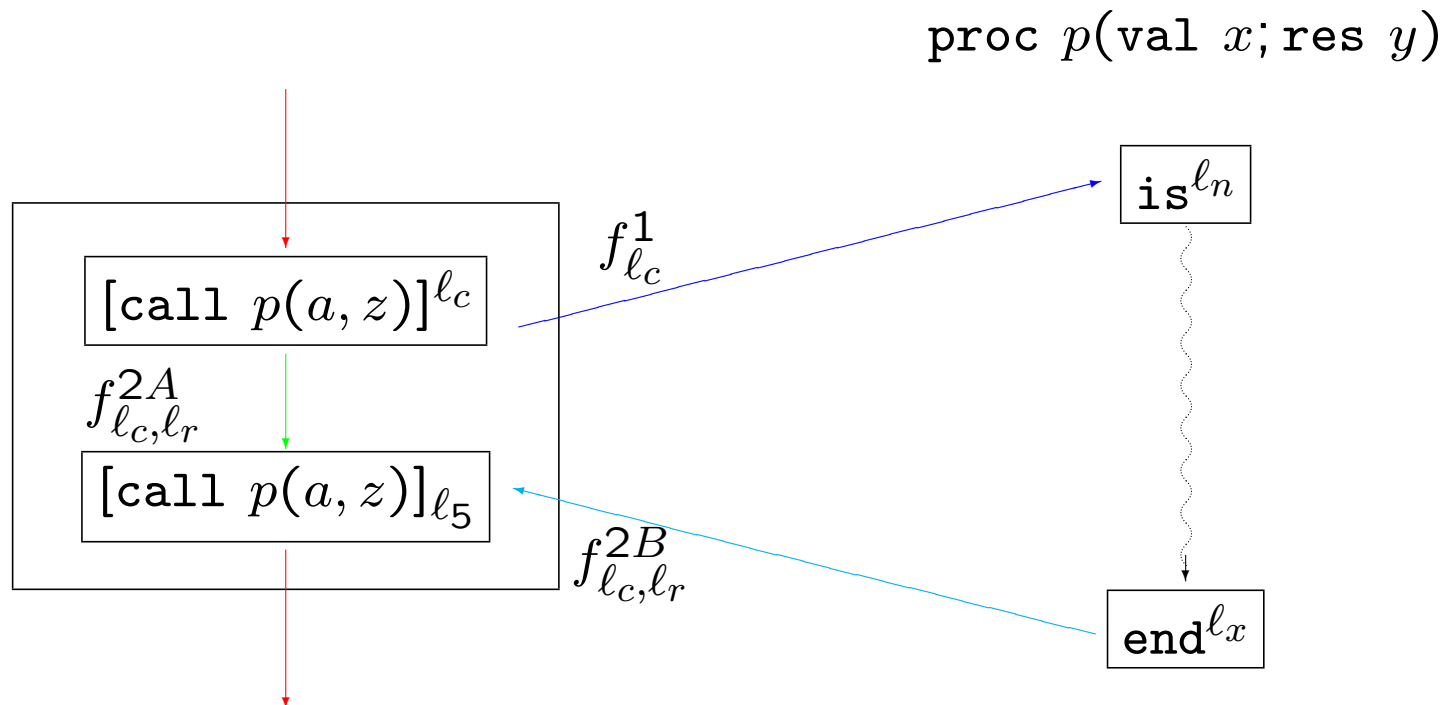
## Variation 1: ignore calling context upon return



$$f^1_{\ell_c}(Z) = \bigcup \{ \{\delta'\} \times \phi^1_{\ell_c}(d) \mid (\delta, d) \in Z \wedge \delta' = \dots \delta \dots d \dots Z \dots \}$$

$$f^2_{\ell_c, \ell_r}(Z, Z') = f^2_{\ell_r}(Z')$$

## Variation 2: joining contexts upon return



$$f_{l_c}^1(Z) = \bigcup \{ \{\delta'\} \times \phi_{l_c}^1(d) \mid (\delta, d) \in Z \wedge \delta' = \dots \delta \dots d \dots Z \dots \}$$

$$f_{l_c, l_r}^2(Z, Z') = f_{l_c, l_r}^{2A}(Z) \sqcup f_{l_c, l_r}^{2B}(Z')$$

# Different Kinds of Context

- **Call Strings** — contexts based on control
  - Call strings of unbounded length
  - Call strings of bounded length ( $k$ )
- **Assumption Sets** — contexts based on data
  - Large assumption sets ( $k = 1$ )
  - Small assumption sets ( $k = 1$ )

## Call Strings of Unbounded Length

$$\Delta = \text{Lab}^*$$

## Transfer functions for procedure call

$$f_{\ell_c}^1(Z) = \bigcup \{ \{\delta'\} \times \phi_{\ell_c}^1(d) \mid (\delta, d) \in Z \wedge \delta' = [\delta, \ell_c] \}$$

$$f_{\ell_c, \ell_r}^2(Z, Z') = \bigcup \{ \{\delta\} \times \phi_{\ell_c, \ell_r}^2(d, d') \mid (\delta, d) \in Z \wedge (\delta', d') \in Z' \wedge \delta' = [\delta, \ell_c] \}$$

## Example:

Recalling the statements:

`proc  $p(\text{val } x; \text{res } y)$  is $\ell_n$   $S$  end $\ell_x$                        $[\text{call } p(a, z)]_{\ell_c}^{\ell_r}$`

Detection of Signs Analysis:

$$\phi_{\ell_c}^{\text{sign1}}(\sigma^{\text{sign}}) = \{\sigma^{\text{sign}} \overbrace{[x \mapsto s][y \mapsto s']}^{\text{initialise formals}} \mid s \in \mathcal{A}_{\text{sign}}[a](\sigma^{\text{sign}}), s' \in \{-, 0, +\}\}$$

$$\phi_{\ell_c, \ell_r}^{\text{sign2}}(\sigma_1^{\text{sign}}, \sigma_2^{\text{sign}}) = \{\sigma_2^{\text{sign}} \underbrace{[x \mapsto \sigma_1^{\text{sign}}(x)][y \mapsto \sigma_1^{\text{sign}}(y)]}_{\text{restore formals}} \underbrace{[z \mapsto \sigma_2^{\text{sign}}(y)]}_{\text{return result}}\}$$



## Call Strings of Bounded Length

$$\Delta = \text{Lab}^{\leq k}$$

## Transfer functions for procedure call

$$f_{\ell_c}^1(Z) = \bigcup \{ \{\delta'\} \times \phi_{\ell_c}^1(d) \mid (\delta, d) \in Z \wedge \delta' = [\delta, \ell_c]_k \}$$

$$f_{\ell_c, \ell_r}^2(Z, Z') = \bigcup \{ \{\delta\} \times \phi_{\ell_c, \ell_r}^2(d, d') \mid (\delta, d) \in Z \wedge (\delta', d') \in Z' \wedge \delta' = [\delta, \ell_c]_k \}$$

A special case: call strings of length  $k = 0$

$$\Delta = \{\wedge\}$$

Note: this is equivalent to having no context information!

Specialising the transfer functions:

$$f_{\ell_c}^1(Y) = \bigcup \{\phi_{\ell_c}^1(d) \mid d \in Y\}$$

$$f_{\ell_c, \ell_r}^2(Y, Y') = \bigcup \{\phi_{\ell_c, \ell_r}^2(d, d') \mid d \in Y \wedge d' \in Y'\}$$

(We use that  $\mathcal{P}(\Delta \times D)$  isomorphic to  $\mathcal{P}(D)$ .)

A special case: call strings of length  $k = 1$

$$\Delta = \mathbf{Lab} \cup \{\Lambda\}$$

Specialising the transfer functions:

$$f_{\ell_c}^1(Z) = \bigcup \{ \{\ell_c\} \times \phi_{\ell_c}^1(d) \mid (\delta, d) \in Z \}$$

$$f_{\ell_c, \ell_r}^2(Z, Z') = \bigcup \{ \{\delta\} \times \phi_{\ell_c, \ell_r}^2(d, d') \mid (\delta, d) \in Z \wedge (\ell_c, d') \in Z' \}$$

## Large Assumption Sets ( $k = 1$ )

$$\Delta = \mathcal{P}(D)$$

### Transfer functions for procedure call

$$f_{\ell_c}^1(Z) = \bigcup \{ \{\delta'\} \times \phi_{\ell_c}^1(d) \mid (\delta, d) \in Z \wedge \delta' = \{d'' \mid (\delta, d'') \in Z\} \}$$

$$f_{\ell_c, \ell_r}^2(Z, Z') = \bigcup \{ \{\delta\} \times \phi_{\ell_c, \ell_r}^2(d, d') \mid (\delta, d) \in Z \wedge (\delta', d') \in Z' \wedge \delta' = \{d'' \mid (\delta, d'') \in Z\} \}$$

## Small Assumption Sets ( $k = 1$ )

$$\Delta = D$$

## Transfer function for procedure call

$$f_{\ell_c}^1(Z) = \bigcup \{ \{d\} \times \phi_{\ell_c}^1(d) \mid (\delta, d) \in Z \}$$

$$f_{\ell_c, \ell_r}^2(Z, Z') = \bigcup \{ \{\delta\} \times \phi_{\ell_c, \ell_r}^2(d, d') \mid (\delta, d) \in Z \wedge (d, d') \in Z' \}$$

# Shape Analysis

Goal: to obtain a **finite representation** of the shape of the heap of a language with pointers.

The analysis result can be used for

- detection of pointer aliasing
- detection of sharing between structures
- software development tools
  - detection of errors like dereferences of `nil`-pointers
- program verification
  - `reverse` transforms a non-cyclic list to a non-cyclic list

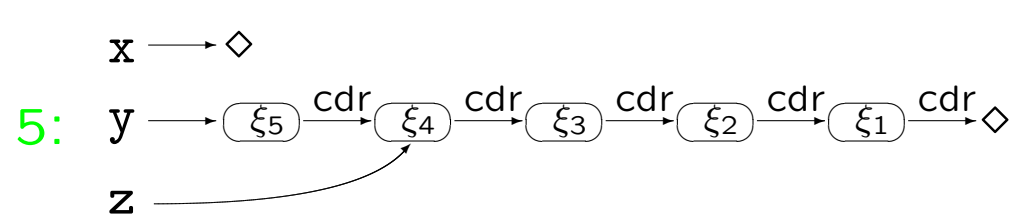
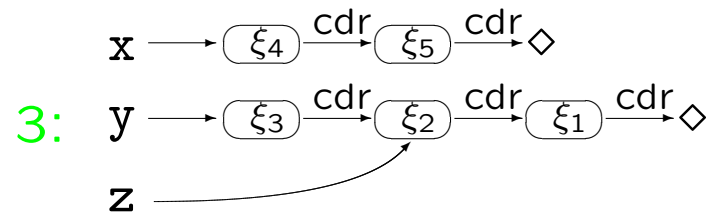
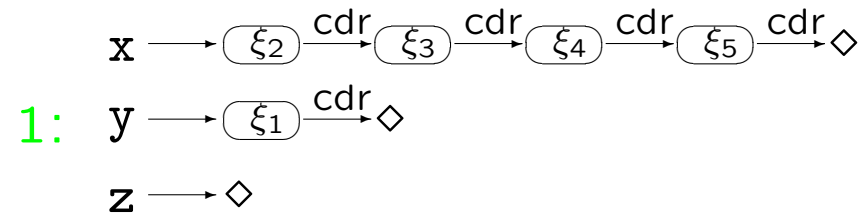
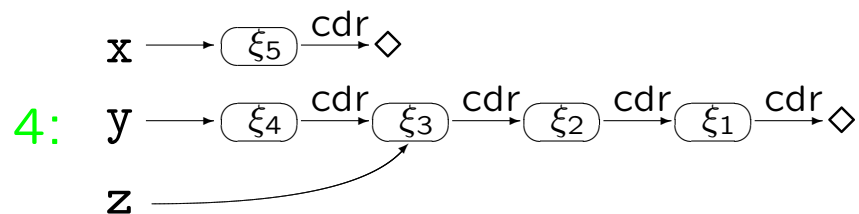
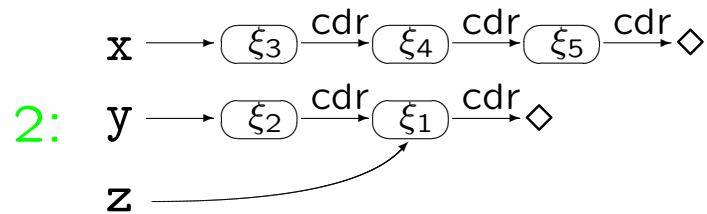
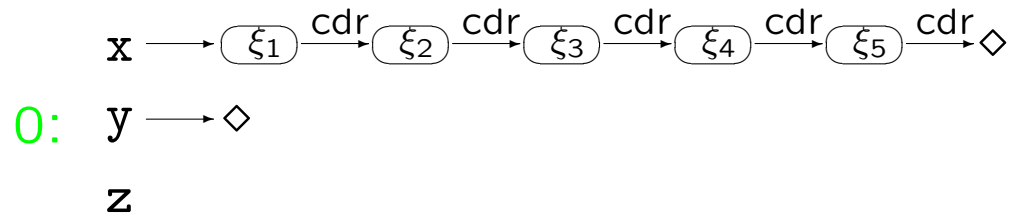
# Syntax of the pointer language

$$\begin{aligned} a &::= p \mid n \mid a_1 \text{ op}_a a_2 \mid \text{nil} \\ p &::= x \mid x.\text{sel} \\ b &::= \text{true} \mid \text{false} \mid \text{not } b \mid b_1 \text{ op}_b b_2 \mid a_1 \text{ op}_r a_2 \mid \text{op}_p p \\ S &::= [p:=a]^\ell \mid [\text{skip}]^\ell \mid S_1; S_2 \mid \\ &\quad \text{if } [b]^\ell \text{ then } S_1 \text{ else } S_2 \mid \text{while } [b]^\ell \text{ do } S \mid \\ &\quad [\text{malloc } p]^\ell \end{aligned}$$

## Example

```
[y:=nil]1;  
while [not is-nil(x)]2 do  
  ([z:=y]3; [y:=x]4; [x:=x.cdr]5; [y.cdr:=z]6);  
[z:=nil]7
```

# Reversal of a list





# Structural Operational Semantics

A configurations consists of

- a state  $\sigma \in \mathbf{State} = \mathbf{Var}_\star \rightarrow (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$   
mapping variables to values, locations (in the heap) or the nil-value
- a heap  $\mathcal{H} \in \mathbf{Heap} = (\mathbf{Loc} \times \mathbf{Sel}) \rightarrow_{\text{fin}} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$   
mapping pairs of locations and selectors to values, locations in the heap or the nil-value

## Pointer expressions

$$\wp : \mathbf{PExp} \rightarrow (\mathbf{State} \times \mathbf{Heap}) \rightarrow_{\text{fin}} (\mathbf{Z} + \{\diamond\} + \mathbf{Loc})$$

is defined by

$$\begin{aligned} \wp[x](\sigma, \mathcal{H}) &= \sigma(x) \\ \wp[x.sel](\sigma, \mathcal{H}) &= \begin{cases} \mathcal{H}(\sigma(x), sel) & \text{if } \sigma(x) \in \mathbf{Loc} \text{ and } \mathcal{H} \text{ is defined on } (\sigma(x), sel) \\ \text{undefined} & \text{otherwise} \end{cases} \end{aligned}$$

## Arithmetic and boolean expressions

$$\mathcal{A} : \mathbf{AExp} \rightarrow (\mathbf{State} \times \mathbf{Heap}) \rightarrow_{\text{fin}} (\mathbf{Z} + \mathbf{Loc} + \{\diamond\})$$

$$\mathcal{B} : \mathbf{BExp} \rightarrow (\mathbf{State} \times \mathbf{Heap}) \rightarrow_{\text{fin}} \mathbf{T}$$

# Statements

Clauses for assignments:

$$\langle [x := a]^\ell, \sigma, \mathcal{H} \rangle \rightarrow \langle \sigma[x \mapsto \mathcal{A}[[a]](\sigma, \mathcal{H})], \mathcal{H} \rangle$$

if  $\mathcal{A}[[a]](\sigma, \mathcal{H})$  is defined

$$\langle [x.sel := a]^\ell, \sigma, \mathcal{H} \rangle \rightarrow \langle \sigma, \mathcal{H}[(\sigma(x), sel) \mapsto \mathcal{A}[[a]](\sigma, \mathcal{H})] \rangle$$

if  $\sigma(x) \in \mathbf{Loc}$  and  $\mathcal{A}[[a]](\sigma, \mathcal{H})$  is defined

Clauses for malloc:

$$\langle [\text{malloc } x]^\ell, \sigma, \mathcal{H} \rangle \rightarrow \langle \sigma[x \mapsto \xi], \mathcal{H} \rangle$$

where  $\xi$  does not occur in  $\sigma$  or  $\mathcal{H}$

$$\langle [\text{malloc } (x.sel)]^\ell, \sigma, \mathcal{H} \rangle \rightarrow \langle \sigma, \mathcal{H}[(\sigma(x), sel) \mapsto \xi] \rangle$$

where  $\xi$  does not occur in  $\sigma$  or  $\mathcal{H}$  and  $\sigma(x) \in \mathbf{Loc}$

# Shape graphs

The analysis will operate on *shape graphs* ( $S, H, is$ ) consisting of

- an abstract state,  $S$ ,
- an abstract heap,  $H$ , and
- sharing information,  $is$ , for the abstract locations.

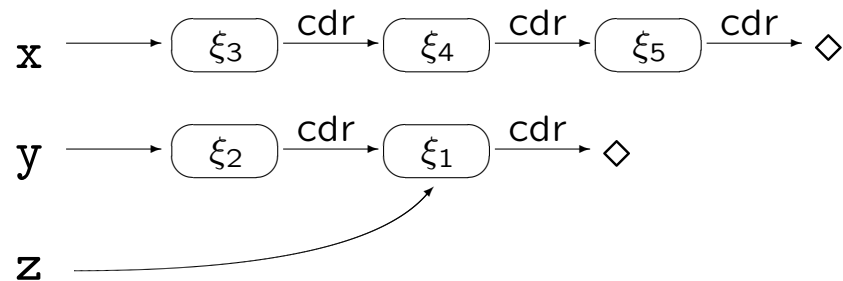
The nodes of the shape graphs are *abstract locations*:

$$\mathbf{ALoc} = \{n_X \mid X \subseteq \mathbf{Var}_\star\}$$

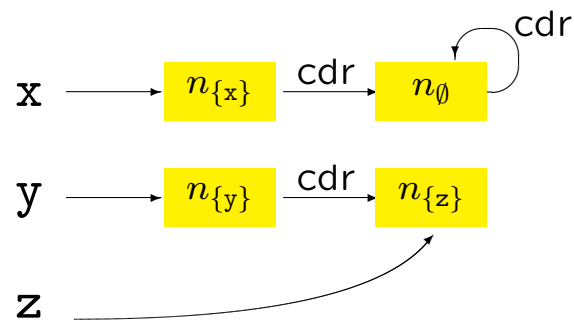
Note: there will only be *finitely* many abstract locations

## Example

In the semantics:



In the analysis:



## Abstract Locations

The abstract location  $n_X$  represents the location  $\sigma(x)$  if  $x \in X$

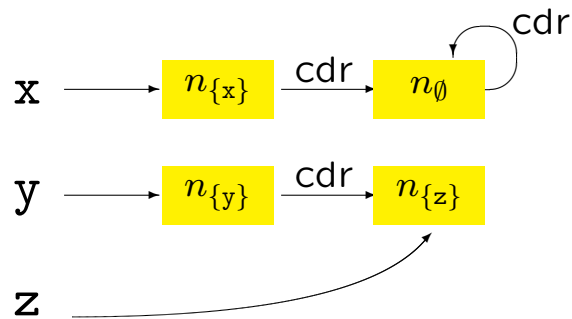
The abstract location  $n_\emptyset$  is called the *abstract summary location*:  $n_\emptyset$  represents all the locations that cannot be reached directly from the state without consulting the heap

**Invariant 1** If two abstract locations  $n_X$  and  $n_Y$  occur in the same shape graph then either  $X = Y$  or  $X \cap Y = \emptyset$

# Abstract states and heaps

$S \in \mathbf{AState} = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{ALoc})$  abstract states

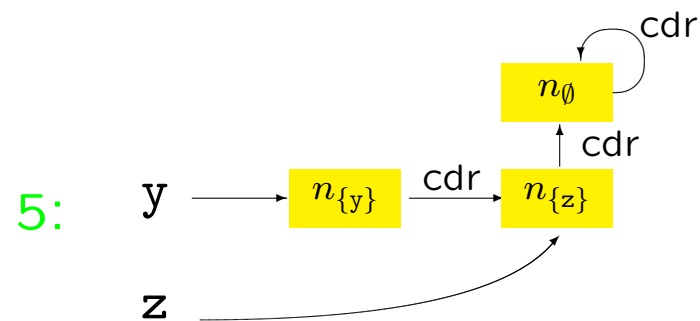
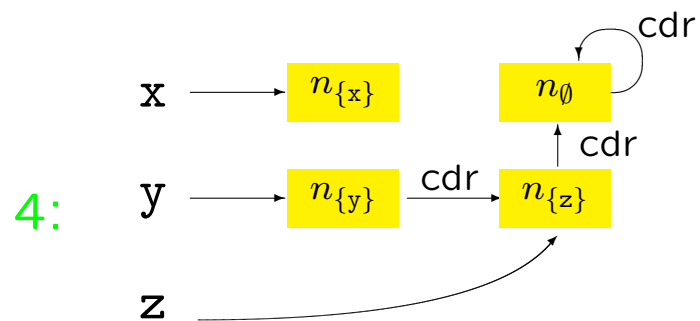
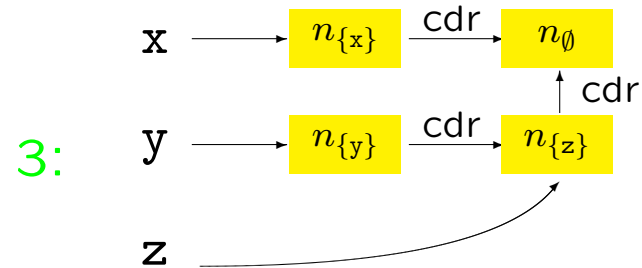
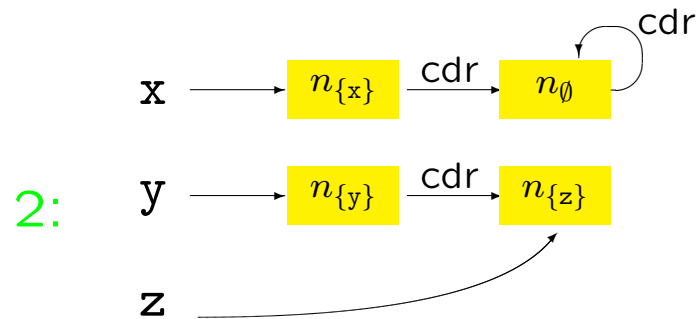
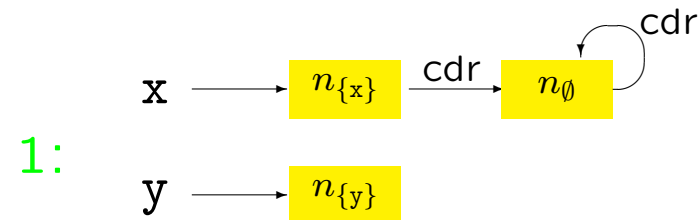
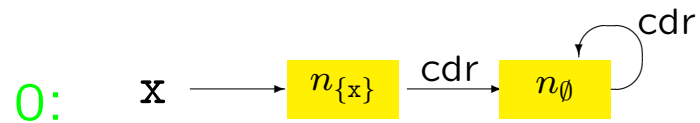
$H \in \mathbf{AHeap} = \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc})$  abstract heap



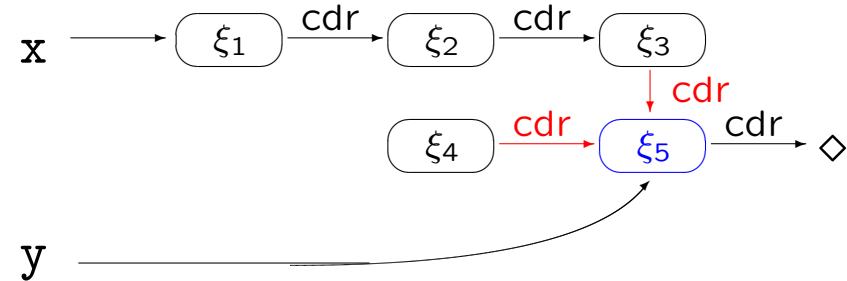
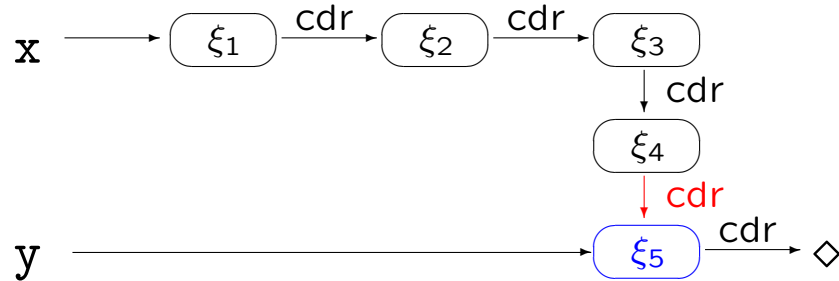
**Invariant 2** If  $x$  is mapped to  $n_X$  by the abstract state  $S$  then  $x \in X$

**Invariant 3** Whenever  $(n_V, sel, n_W)$  and  $(n_V, sel, n_{W'})$  are in the abstract heap  $H$  then either  $V = \emptyset$  or  $W = W'$

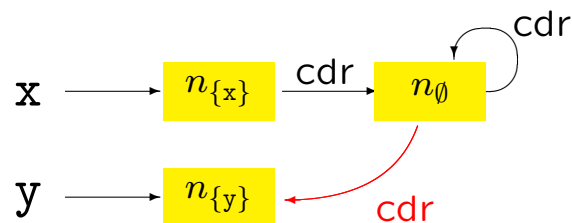
# Reversal of a list



# Sharing in the heap



Give rise to the same shape graph:

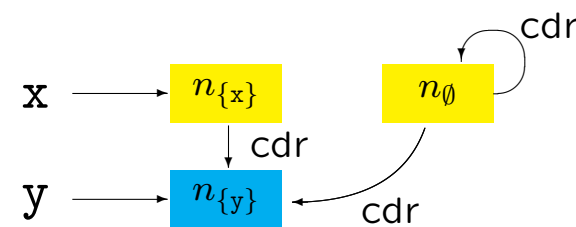
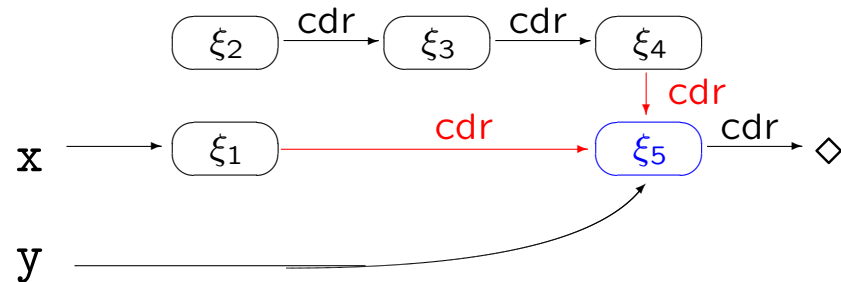
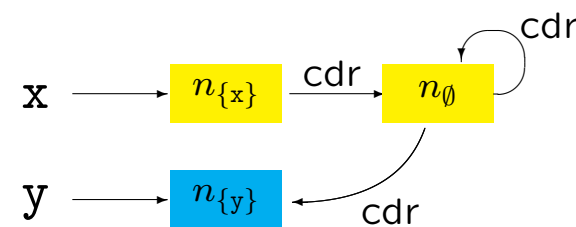
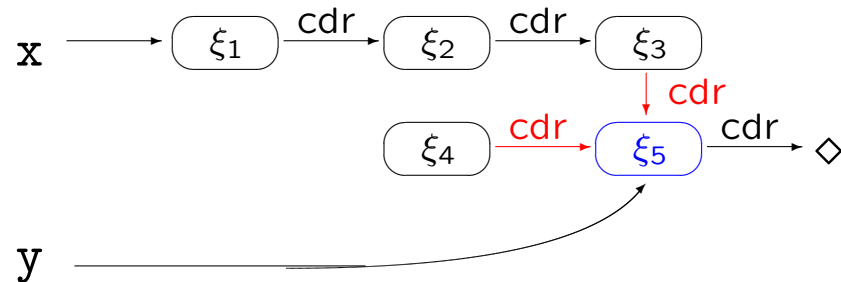
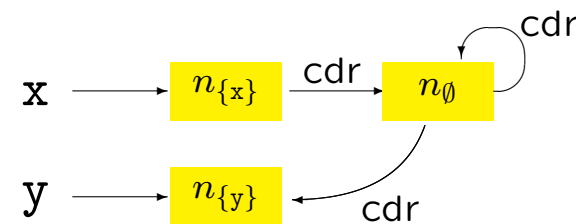
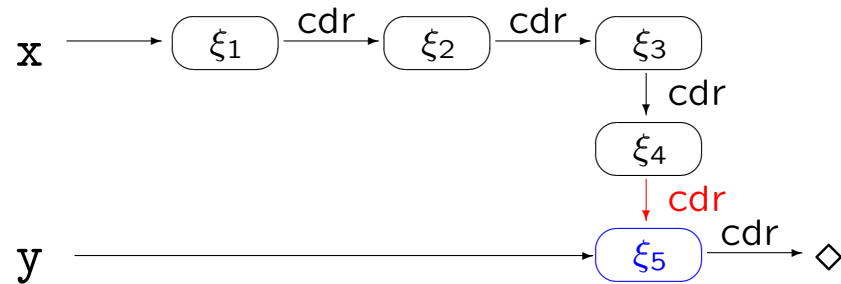


**is**: the abstract locations that *might* be shared due to pointers in the heap:

$n_X$  is included in **is** if it might represents a location that **is the target of more than one pointer** in the heap

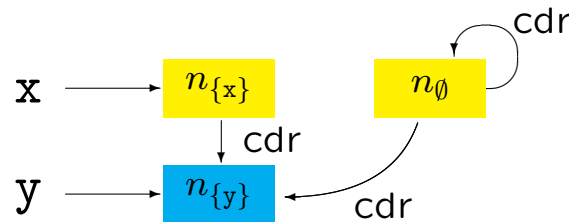


# Examples: sharing in the heap



# Sharing information

The **implicit** sharing information of the abstract heap must be consistent with the **explicit** sharing information:



**Invariant 4** If  $n_X \in is$  then either

- $(n_{\emptyset}, sel, n_X)$  is in the abstract heap for some  $sel$ , or
- there are two distinct triples  $(n_V, sel_1, n_X)$  and  $(n_W, sel_2, n_X)$  in the abstract heap

**Invariant 5** Whenever there are two distinct triples  $(n_V, sel_1, n_X)$  and  $(n_W, sel_2, n_X)$  in the abstract heap and  $X \neq \emptyset$  then  $n_X \in is$

# The complete lattice of shape graphs

A *shape graph* is a triple  $(S, H, is)$  where

$$S \in \mathbf{AState} = \mathcal{P}(\mathbf{Var}_\star \times \mathbf{ALoc})$$

$$H \in \mathbf{AHeap} = \mathcal{P}(\mathbf{ALoc} \times \mathbf{Sel} \times \mathbf{ALoc})$$

$$is \in \mathbf{IsShared} = \mathcal{P}(\mathbf{ALoc})$$

and  $\mathbf{ALoc} = \{n_Z \mid Z \subseteq \mathbf{Var}_\star\}$ .

A shape graph  $(S, H, is)$  is *compatible* if it fulfils the five invariants.

The analysis computes over *sets of compatible shape graphs*

$$\mathbf{SG} = \{(S, H, is) \mid (S, H, is) \text{ is compatible}\}$$

## The analysis

An instance of a *forward* Monotone Framework with the complete lattice of interest being  $\mathcal{P}(\text{SG})$

A *may analysis*: each of the sets of shape graphs computed by the analysis may contain shape graphs that cannot really arise

Aspects of a *must analysis*: each of the individual shape graphs (in a set of shape graphs computed by the analysis) will be the best possible description of some  $(\sigma, \mathcal{H})$

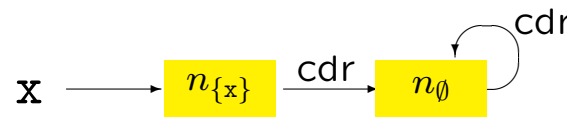
# The analysis

Equations:

$$Shape_o(\ell) = \begin{cases} \iota & \text{if } \ell = \text{init}(S_\star) \\ \bigcup \{ Shape_\bullet(\ell') \mid (\ell', \ell) \in \text{flow}(S_\star) \} & \text{otherwise} \end{cases}$$

$$Shape_\bullet(\ell) = f_\ell^{SA}(Shape_o(\ell))$$

**Example:** The extremal value  $\iota$  for the list reversal program



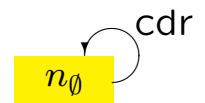
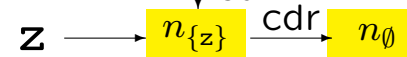
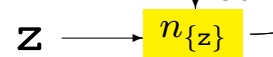
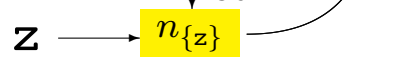
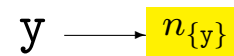
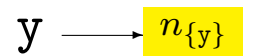
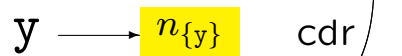
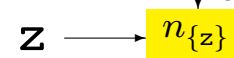
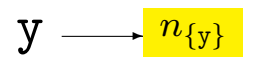
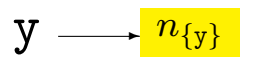
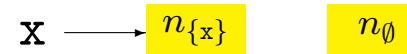
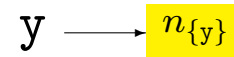
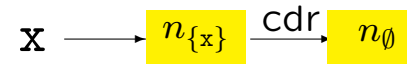
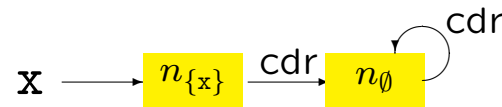
–  $x$  points to a non-cyclic list with at least three elements

*Shape*<sub>•</sub>(1) for  $[y:=\text{nil}]^1$

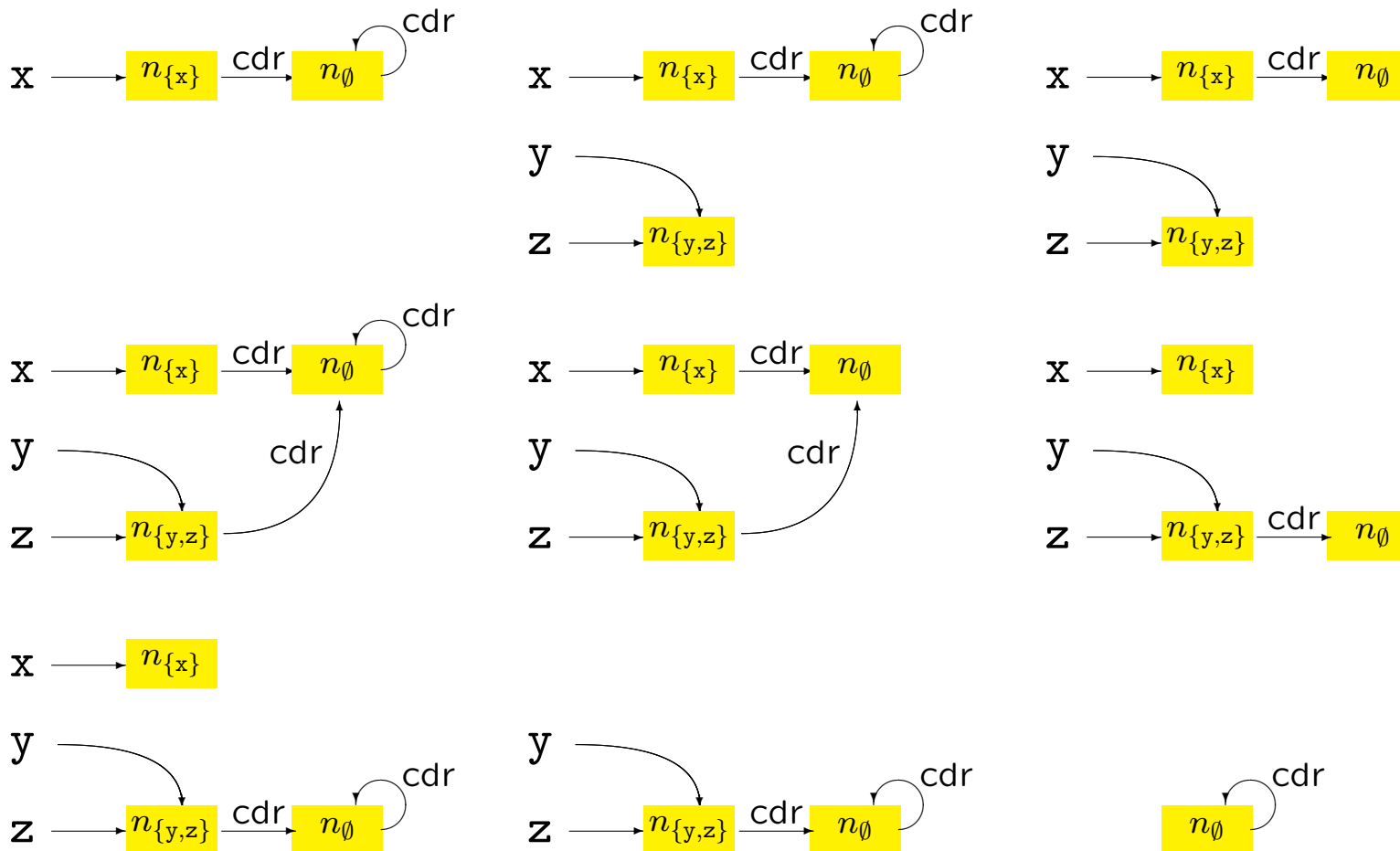


Note: we do not record `nil`-values in the analysis

# Shape.(2) for $[\text{not is-nil}(x)]^2$

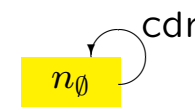
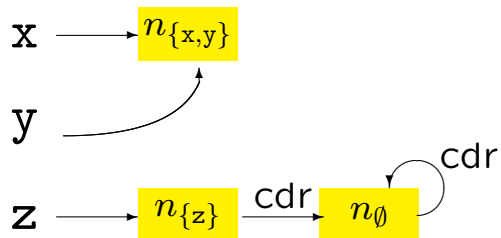
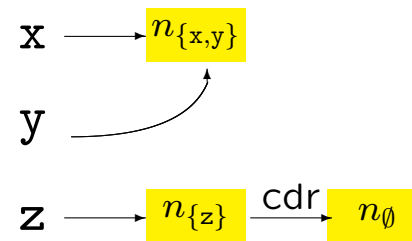
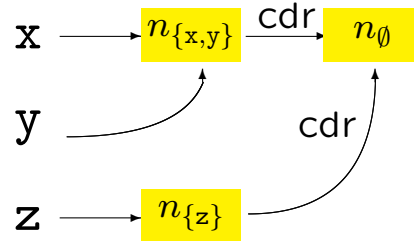
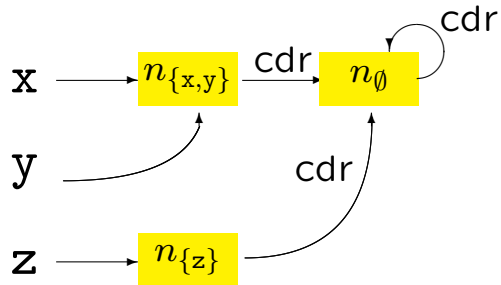
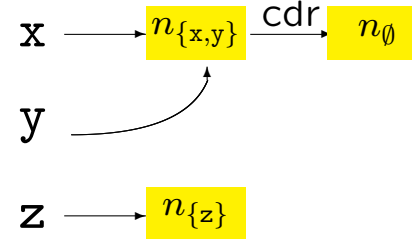
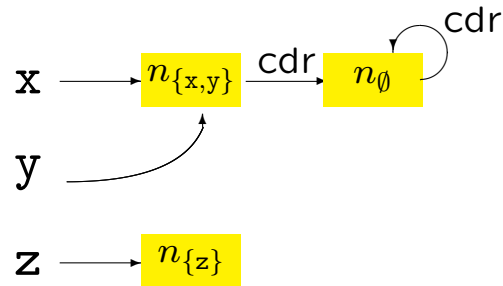
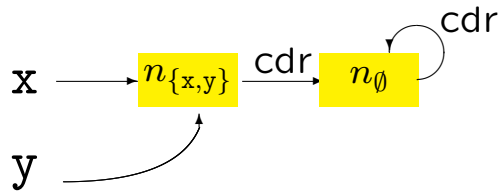


## Shape.(3) for $[z:=y]^3$

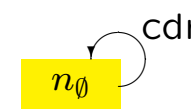
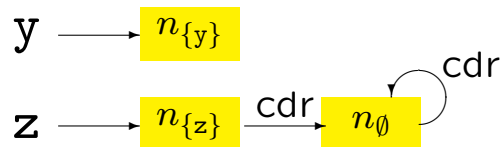
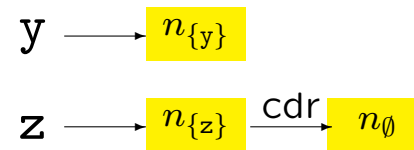
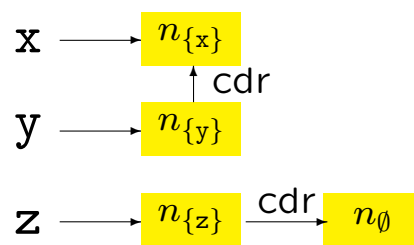
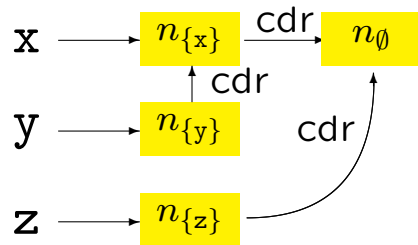
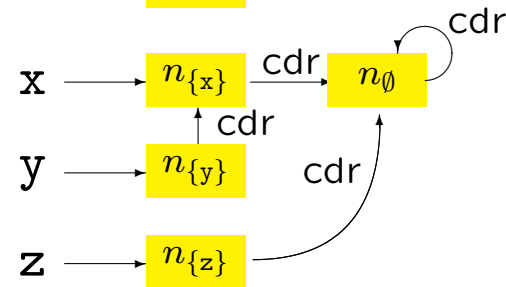
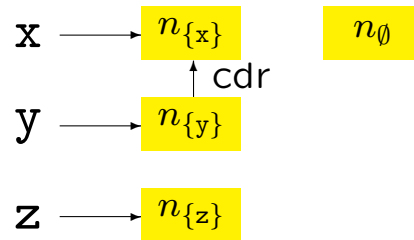
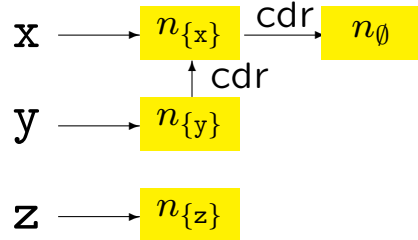
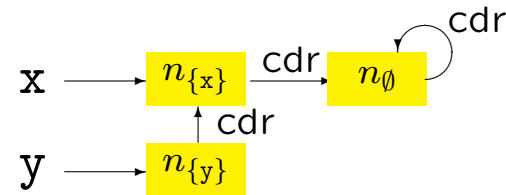
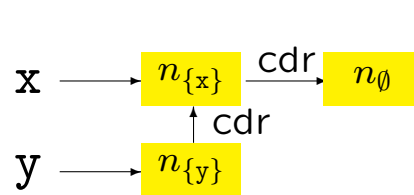
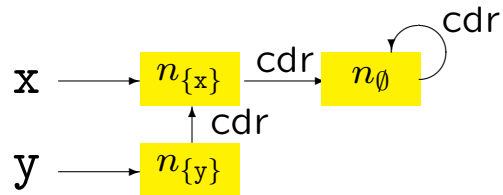




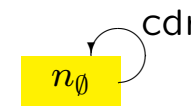
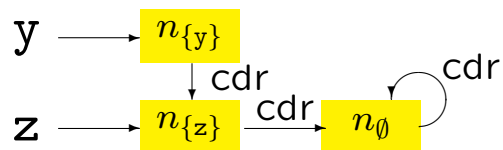
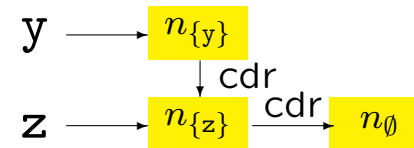
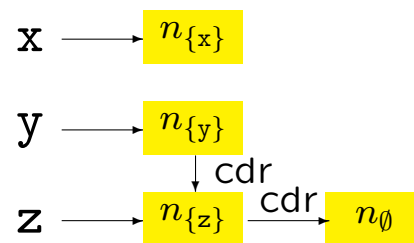
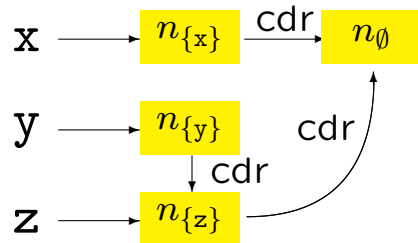
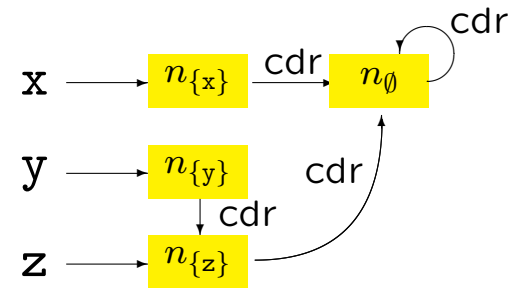
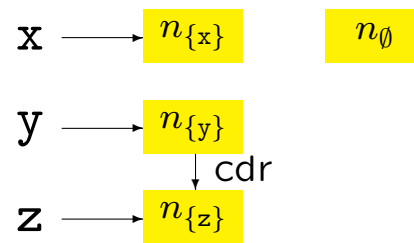
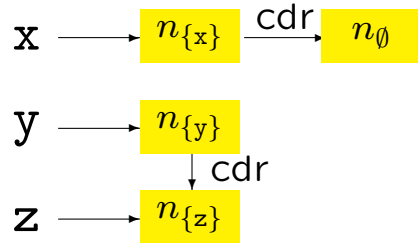
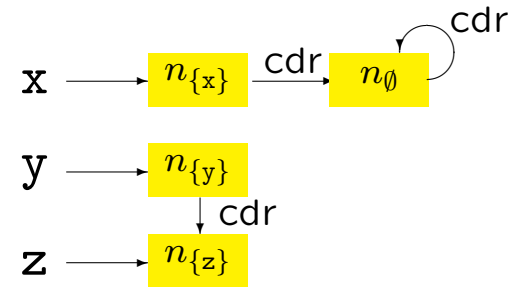
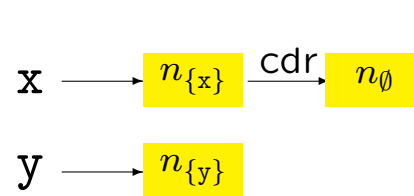
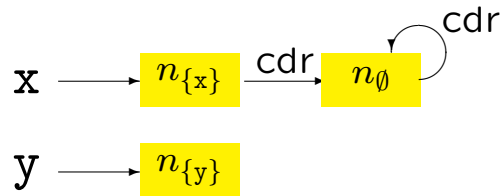
## Shape<sub>•</sub>(4) for $[y:=x]^4$



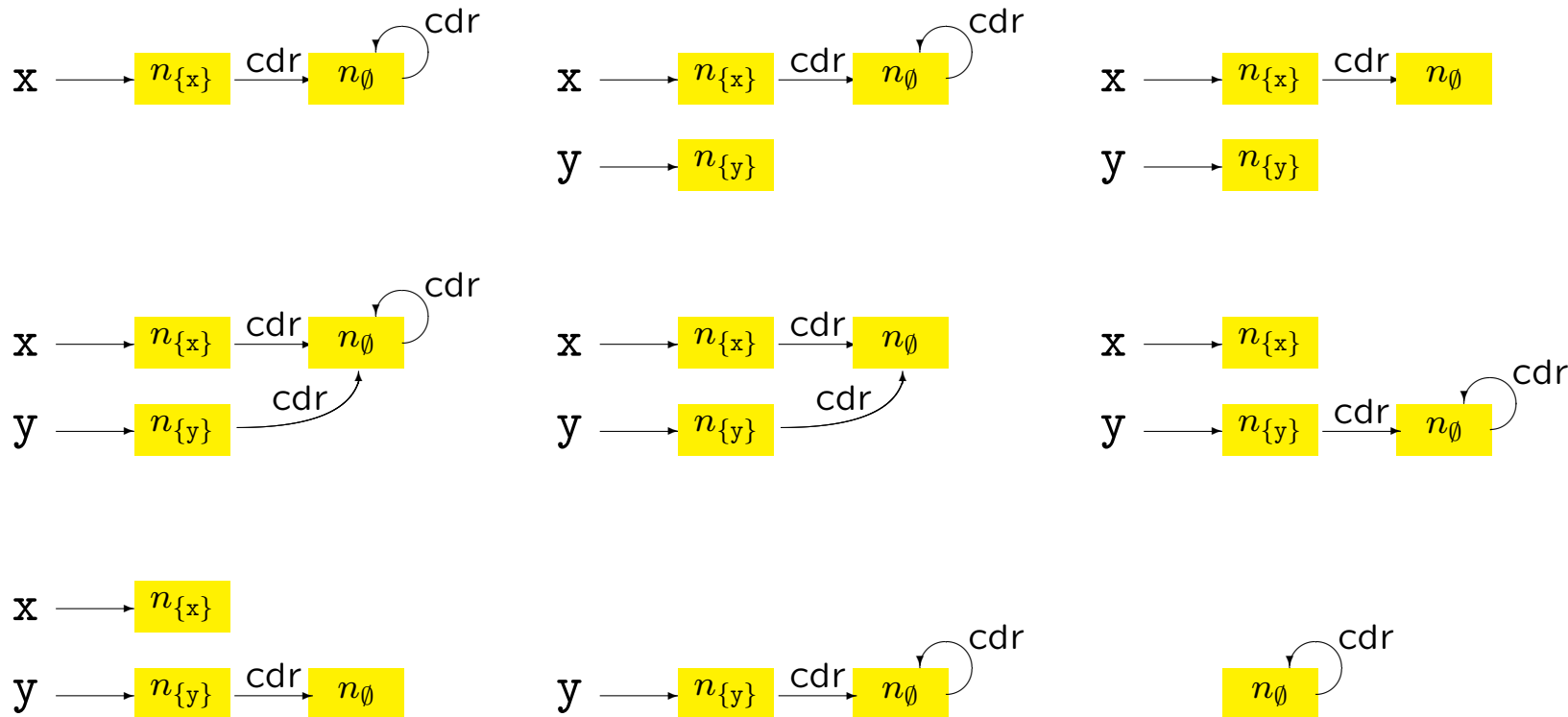
# *Shape*<sub>•</sub>(5) for $[x := x.cdr]^5$



# *Shape*•(6) for $[y.cdr:=z]^6$



## *Shape*<sub>•</sub>(7) for $[z:=\text{nil}]^7$



– upon termination  $y$  points to a non-circular list

– a more precise analysis taking tests into account will know that  $x$  is  $\text{nil}$  upon termination

# Transfer functions

$$f_\ell^{\text{SA}} : \mathcal{P}(\text{SG}) \rightarrow \mathcal{P}(\text{SG})$$

has the form:

$$f_\ell^{\text{SA}}(SG) = \bigcup \{ \phi_\ell^{\text{SA}}((\text{S}, \text{H}, \text{is})) \mid (\text{S}, \text{H}, \text{is}) \in SG \}$$

where

$$\phi_\ell^{\text{SA}} : \text{SG} \rightarrow \mathcal{P}(\text{SG})$$

specifies how a *single* shape graph (in *Shape<sub>o</sub>*( $\ell$ )) may be transformed into a *set* of shape graphs (in *Shape<sub>•</sub>*( $\ell$ )) by the elementary block.

## Transfer function for $[b]^\ell$ and $[\text{skip}]^\ell$

We are only interested in the shape of the heap – and it is not changed by these elementary blocks:

$$\phi_\ell^{\text{SA}}((S, H, \text{is})) = \{(S, H, \text{is})\}$$

## Transfer function for $[x := a]^\ell$

— where  $a$  is of the form  $n$ ,  $a_1 \text{ op}_a a_2$  or  $\text{nil}$

$$\phi_\ell^{\text{SA}}((S, H, \text{is})) = \{\text{kill}_x((S, H, \text{is}))\}$$

where  $\text{kill}_x((S, H, \text{is})) = (S', H', \text{is}')$  is

$$S' = \{(z, k_x(n_Z)) \mid (z, n_Z) \in S \wedge z \neq x\}$$

$$H' = \{(k_x(n_V), \text{sel}, k_x(n_W)) \mid (n_V, \text{sel}, n_W) \in H\}$$

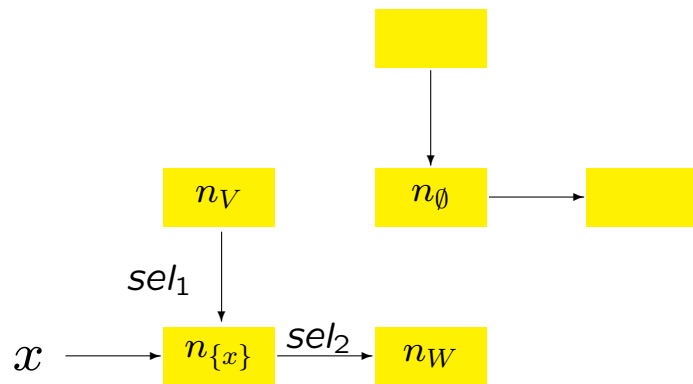
$$\text{is}' = \{k_x(n_X) \mid n_X \in \text{is}\}$$

and

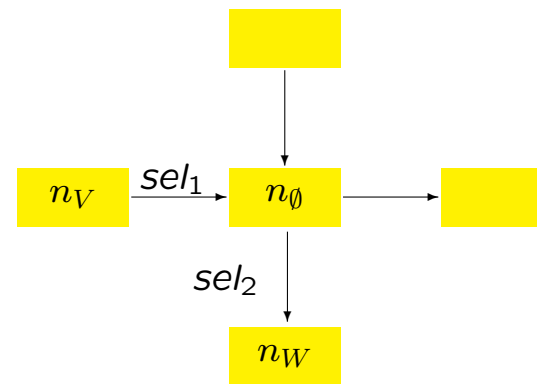
$$k_x(n_Z) = n_{Z \setminus \{x\}}$$

Idea: all abstract locations are renamed to not having  $x$  in their name set

# The effect of $[x := \text{nil}]^\ell$



$(S, H, is)$



$(S', H', is')$



## Transfer function for $[x:=y]^\ell$ when $x \neq y$

$$\phi_\ell^{\text{SA}}((S, H, \text{is})) = \{(S'', H'', \text{is}'')\}$$

where  $(S', H', \text{is}') = \text{kill}_x((S, H, \text{is}))$  and

$$\begin{aligned} S'' &= \{(z, g_x^y(n_Z)) \mid (z, n_Z) \in S'\} \\ &\quad \cup \{(x, g_x^y(n_Y)) \mid (y', n_Y) \in S' \wedge y' = y\} \end{aligned}$$

$$H'' = \{(g_x^y(n_V), \text{sel}, g_x^y(n_W)) \mid (n_V, \text{sel}, n_W) \in H'\}$$

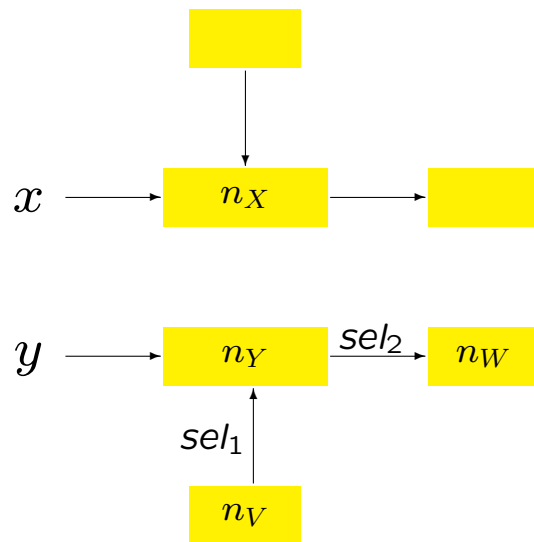
$$\text{is}'' = \{g_x^y(n_Z) \mid n_Z \in \text{is}'\}$$

and

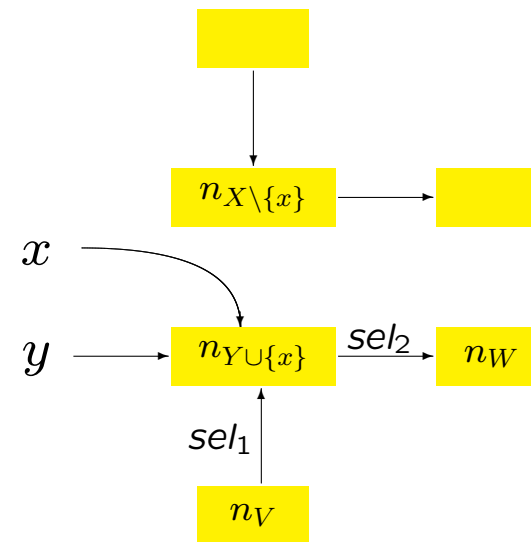
$$g_x^y(n_Z) = \begin{cases} n_{Z \cup \{x\}} & \text{if } y \in Z \\ n_Z & \text{otherwise} \end{cases}$$

Idea: all abstract locations are renamed to also have  $x$  in their name set if they already have  $y$

The effect of  $[x:=y]^\ell$  when  $x \neq y$



$(S, H, is)$



$(S'', H'', is'')$

## Transfer function for $[x:=y.sel]^\ell$ when $x \neq y$

Remove the old binding for  $x$ :

strong nullification

$$(S', H', is') = kill_x((S, H, is))$$

Establish the new binding for  $x$ :

1. There is no abstract location  $n_Y$  such that  $(y, n_Y) \in S'$  – or there is an abstract location  $n_Y$  such that  $(y, n_Y) \in S'$  but no  $n_Z$  such that  $(n_Y, sel, n_Z) \in H'$
2. There is an abstract location  $n_Y$  such that  $(y, n_Y) \in S'$  and there is an abstract location  $n_U \neq n_\emptyset$  such that  $(n_Y, sel, n_U) \in H'$
3. There is an abstract location  $n_Y$  such that  $(y, n_Y) \in S'$  and  $(n_Y, sel, n_\emptyset) \in H'$

## Case 1 for $[x := y.sel]^\ell$

Assume there is no abstract location  $n_Y$  such that  $(y, n_Y) \in S'$

$$\phi_\ell^{\text{SA}}((S, H, is)) = \{(S', H', is')\}$$

OBS: dereference of a nil-pointer

Assume there is an abstract location  $n_Y$  such that  $(y, n_Y) \in S'$  but there is no abstract location  $n$  such that  $(n_Y, sel, n) \in H'$

$$\phi_\ell^{\text{SA}}((S, H, is)) = \{(S', H', is')\}$$

OBS: dereference of a non-existing sel-field

## Case 2 for $[x:=y.sel]^\ell$

Assume there is an abstract location  $n_Y$  such that  $(y, n_Y) \in S'$  and there is an abstract location  $n_U \neq n_\emptyset$  such that  $(n_Y, sel, n_U) \in H'$ .

The abstract location  $n_U$  will be renamed to include the variable  $x$  using the function:

$$h_x^U(n_Z) = \begin{cases} n_{U \cup \{x\}} & \text{if } Z = U \\ n_Z & \text{otherwise} \end{cases}$$

We take

$$\phi_\ell^{\text{SA}}((S, H, is)) = \{(S'', H'', is'')\}$$

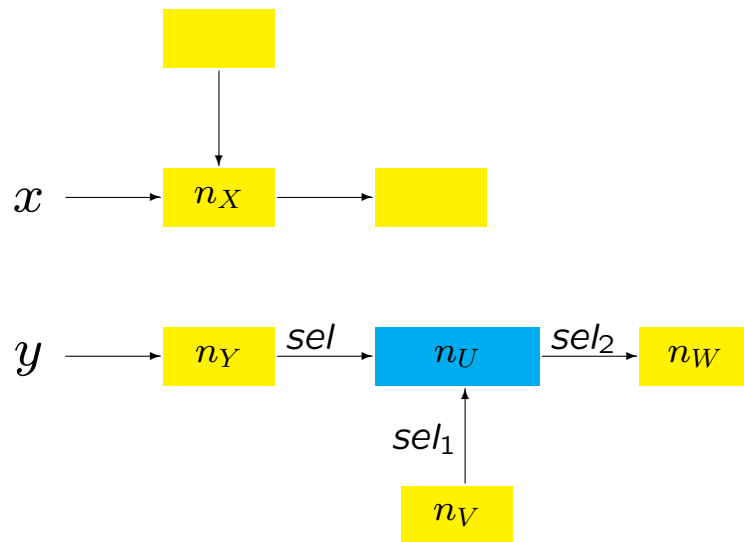
where  $(S', H', is') = \text{kill}_x((S, H, is))$  and

$$S'' = \{(z, h_x^U(n_Z)) \mid (z, n_Z) \in S'\} \cup \{(x, h_x^U(n_U))\}$$

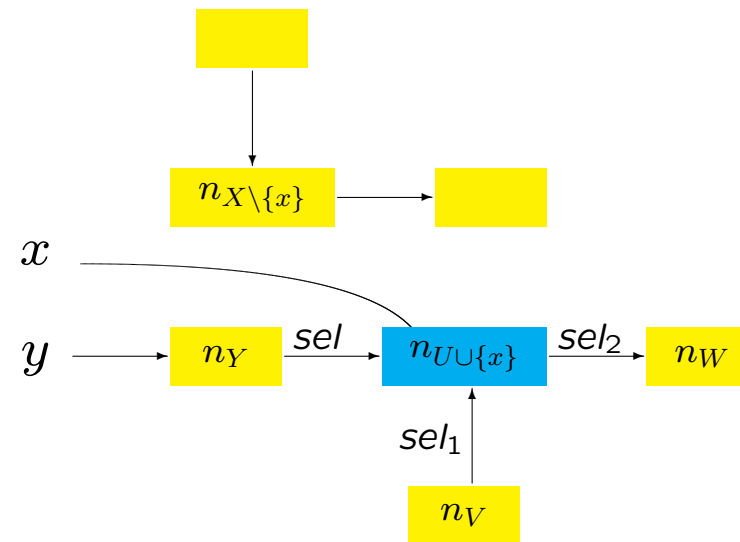
$$H'' = \{(h_x^U(n_V), sel', h_x^U(n_W)) \mid (n_V, sel', n_W) \in H'\}$$

$$is'' = \{h_x^U(n_Z) \mid n_Z \in is'\}$$

## The effect of $[x := y.sel]^\ell$ in Case 2



$(S, H, is)$

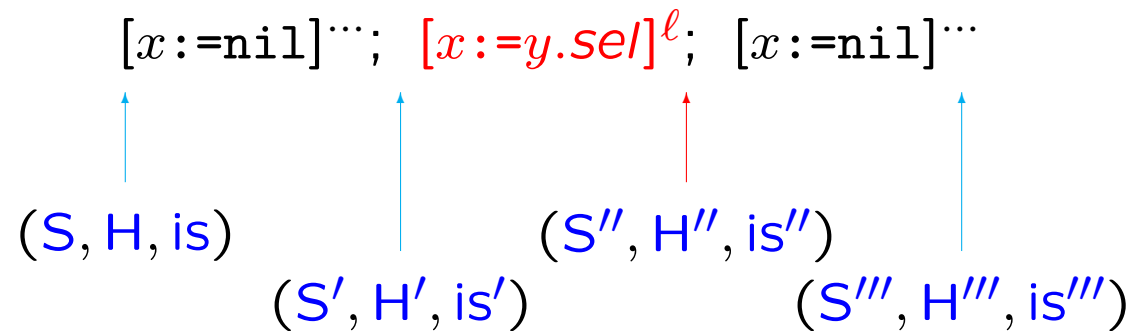


$(S'', H'', is'')$

## Case 3 for $[x:=y.sel]^\ell$ (1)

Assume that there is an abstract location  $n_Y$  such that  $(y, n_Y) \in S'$  and furthermore  $(n_Y, sel, n_\emptyset) \in H'$ .

We have to *materialise* a new abstract location  $n_{\{x\}}$  from  $n_\emptyset$ .



Idea:

$$(S', H', is') = (S''', H''', is''') = kill_x((S'', H'', is''))$$

## Case 3 for $[x:=y.sel]^\ell$ (2)

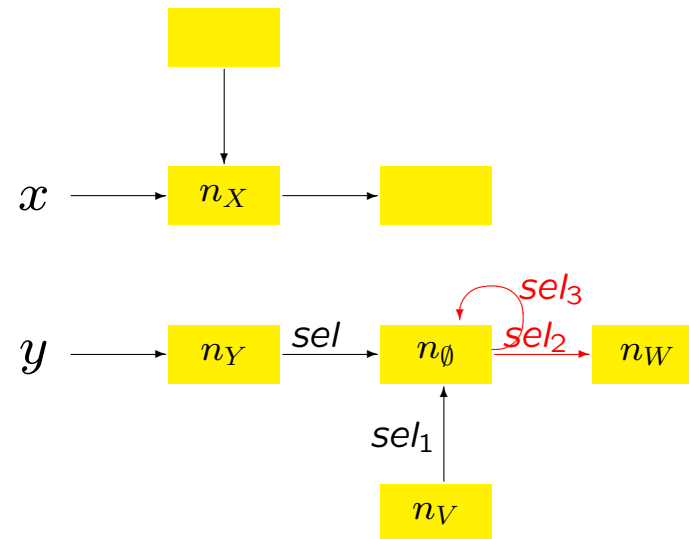
Transfer function:

$$\begin{aligned} \phi_\ell^{\text{SA}}((S, H, is)) = \{ & (S'', H'', is'') \mid (S'', H'', is'') \text{ is compatible} \wedge \\ & kill_x((S'', H'', is'')) = (S', H', is') \wedge \\ & (x, n_{\{x\}}) \in S'' \wedge (n_Y, sel, n_{\{x\}}) \in H'' \} \end{aligned}$$

where  $(S', H', is') = kill_x((S, H, is))$ .

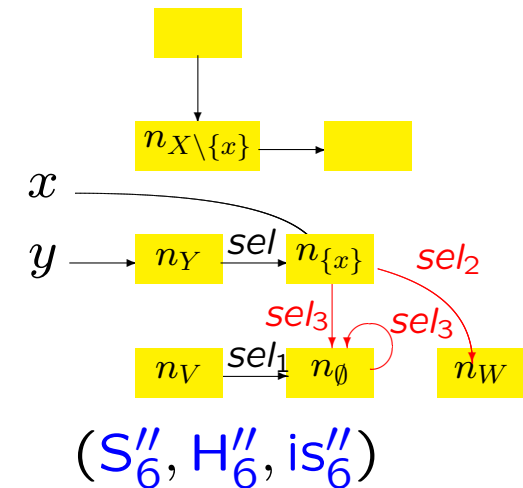
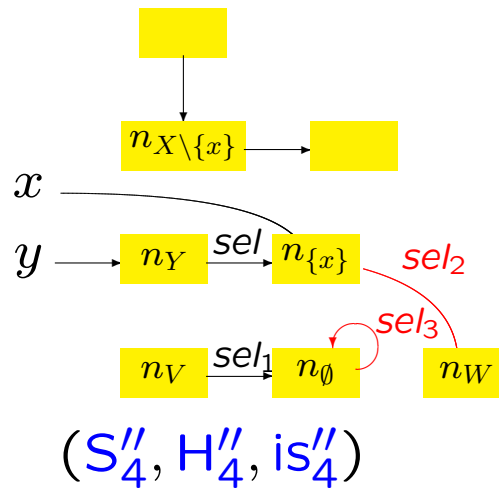
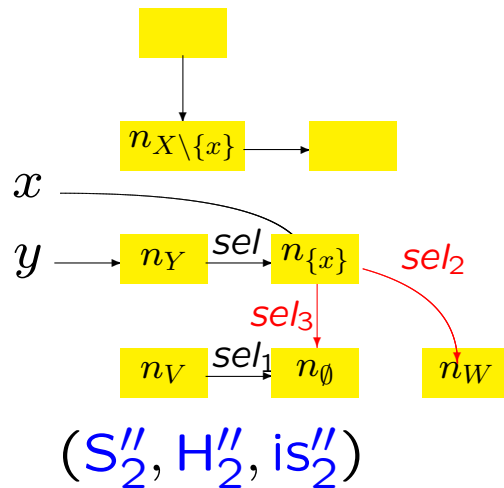
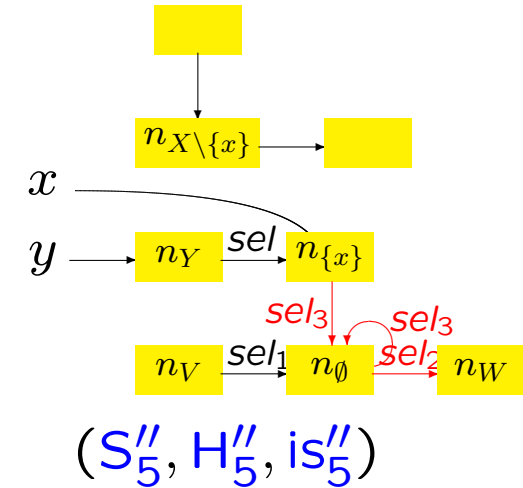
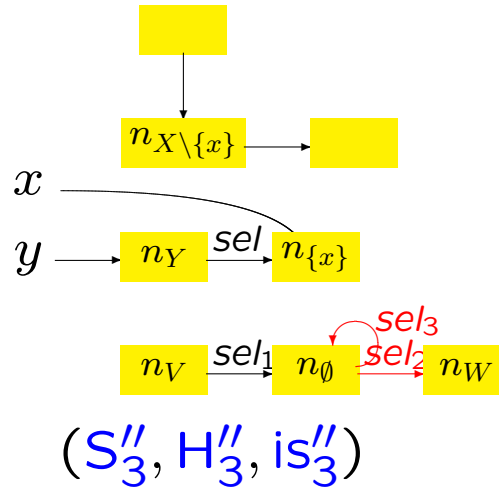
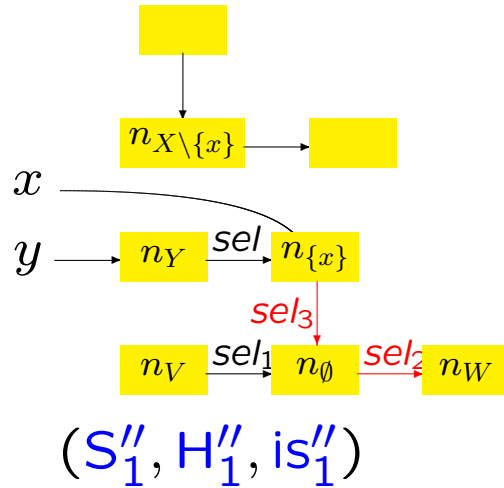


The effect of  $[x := y.sel]^\ell$  in Case 3 (1)



$(S, H, is)$

# The effect of $[x:=y.sel]^\ell$ in Case 3 (2)



## Transfer function for $[x.sel := a]^\ell$

— where  $a$  is of the form  $n$ ,  $a_1 \text{ op}_a a_2$  or  $\text{nil}$ .

If there is no  $n_X$  such that  $(x, n_X) \in S$  then  $f_\ell^{\text{SA}}$  is the identity.

If there is  $n_X$  such that  $(x, n_X) \in S$  but that there is no  $n_U$  such that  $(n_X, sel, n_U) \in H$  then  $f_\ell^{\text{SA}}$  is the identity.

If there are abstract locations  $n_X$  and  $n_U$  such that  $(x, n_X) \in S$  and  $(n_X, sel, n_U) \in H$  then

$$\phi_\ell^{\text{SA}}((S, H, is)) = \{\text{kill}_{x.sel}((S, H, is))\}$$

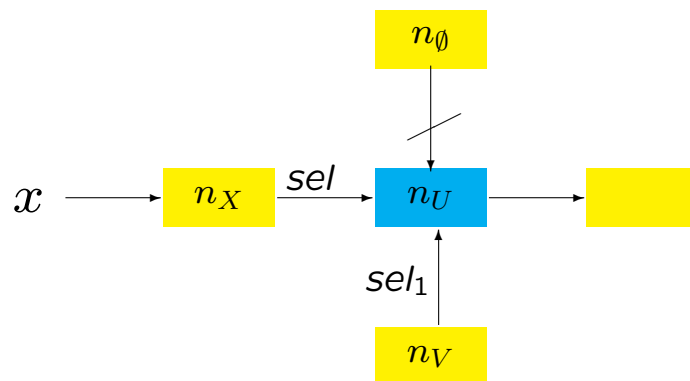
where  $\text{kill}_{x.sel}((S, H, is)) = (S', H', is')$  is given by

$$S' = S$$

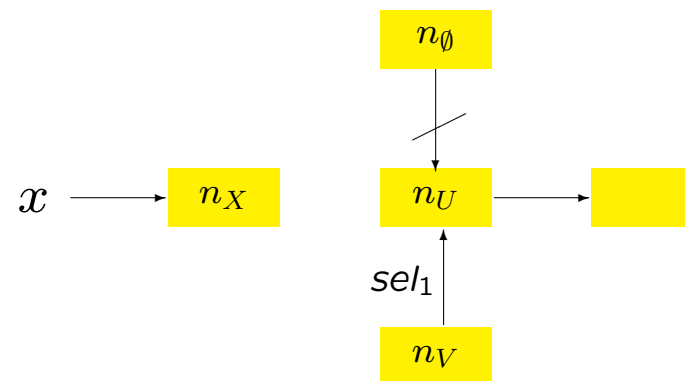
$$H' = \{(n_V, sel', n_W) \mid (n_V, sel', n_W) \in H \wedge \neg(X = V \wedge sel = sel')\}$$

$$is' = \begin{cases} is \setminus \{n_U\} & \text{if } n_U \in is \wedge \#into(n_U, H') \leq 1 \wedge \neg \exists (n_\emptyset, sel', n_U) \in H' \\ is & \text{otherwise} \end{cases}$$

The effect of  $[x.sel := \text{nil}]^\ell$  when  $\#into(n_U, H') \leq 1$



$(S, H, is)$



$(S', H', is')$

## Transfer function for $[x.sel := y]^\ell$ when $x \neq y$

If there is no  $n_X$  such that  $(x, n_X) \in S$  then  $f_\ell^{SA}$  is the identity function.

If  $(x, n_X) \in S$  but there is no  $n_Y$  such that  $(y, n_Y) \in S$  then

$$\phi_\ell^{SA}((S, H, is)) = \{\textit{kill}_{x.sel}((S, H, is))\}$$

If there is  $(x, n_X) \in S$  and  $(y, n_Y) \in S$  then

$$\phi_\ell^{SA}((S, H, is)) = \{(S'', H'', is'')\}$$

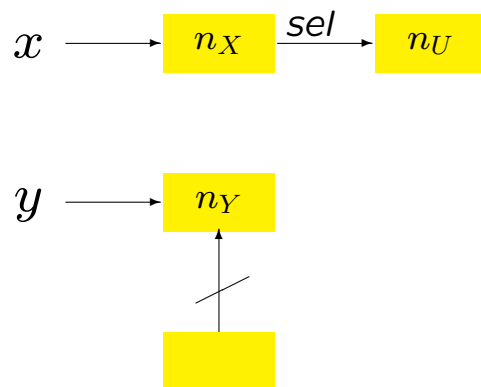
where  $(S', H', is') = \textit{kill}_{x.sel}((S, H, is))$  and

$$S'' = S' \quad (= S)$$

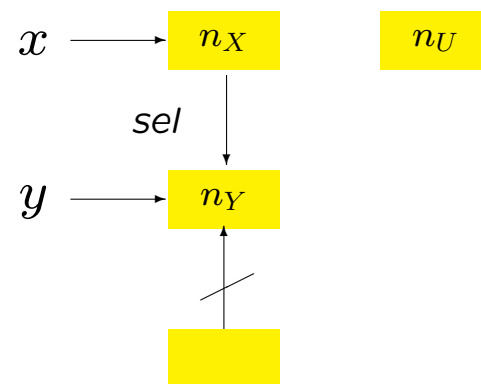
$$H'' = H' \cup \{(n_X, sel, n_Y) \mid (x, n_X) \in S' \wedge (y, n_Y) \in S'\}$$

$$is'' = \begin{cases} is' \cup \{n_Y\} & \text{if } \#into(n_Y, H') \geq 1 \\ is' & \text{otherwise} \end{cases}$$

The effect of  $[x.sel := y]^\ell$  when  $\#into(n_Y, H') \leq 1$



$(S, H, is)$



$(S', H'', is'')$

## Transfer function for $[\text{malloc } x]^\ell$

$$\phi_\ell^{\text{SA}}((S, H, \text{is})) = \{(S' \cup \{(x, n_{\{x\}})\}, H', \text{is}')\}$$

where  $(S', H', \text{is}') = \text{kill}_x(S, H, \text{is})$ .